N° d'ordre : 464 Année 2006

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR
de
L'ÉCOLE NATIONALE SUPÉRIEURE
DE L'AÉRONAUTIQUE ET DE L'ESPACE

ÉCOLE DOCTORALE : Informatique et télécommunications
SPÉCIALITÉ : Informatique

par

Cédric PRALET

Un cadre algébrique général pour représenter et résoudre des problèmes de décision séquentielle avec incertitudes, faisabilités et utilités

A generic framework for representing and solving sequential decision making problems with uncertainties, feasibilities and utilities

Soutenue le 17 novembre 2006 devant le jury :

MM.	M.	GHALLAB	Président
	P.	PERNY	Rapporteur
Mme	F.	ROSSI	Rapporteur
MM.	T.	SCHIEX	Co-directeur de thèse
	G.	VERFAILLIE	Co-directeur de thèse
	N.	WILSON	

Thèse préparée au sein du LAAS-CNRS et de l'INRA Toulouse.

THÈSE

présentée pour obtenir le titre de

DOCTEUR DE l'ÉCOLE NATIONALE SUPÉRIEURE DE L'AÉRONAUTIQUE ET DE L'ESPACE

Spécialité : Informatique

par

Cédric Pralet

Un cadre algébrique général pour représenter et résoudre des problèmes de décision séquentielle avec incertitudes, faisabilités et utilités

A generic algebraic framework for representing and solving sequential decision making problems with uncertainties, feasibilities, and utilities

---

Thèse présentée devant le jury composé de:

Malik Ghallab	**LAAS-CNRS, Toulouse**	Examinateur
Patrice Perny	**LIP6, Paris**	Rapporteur
Francesca Rossi	**Université de Padoue (Italie)**	Rapporteur
Thomas Schiex	**INRA, Toulouse**	Directeur de thèse
Gérard Verfaillie	**ONERA, Toulouse**	Directeur de thèse
Nic Wilson	**4C, Cork (Irlande)**	Examinateur

Thèse préparée au LAAS-CNRS et à l'INRA Toulouse

Contents

Remerciements

Merci tout d'abord à Elyssa de m'avoir toujours supporté (dans les deux sens du terme) pendant ma thèse. Cette thèse est un peu la tienne. Merci aussi à ma famille pour son soutien. Je tiens également à remercier les personnes suivantes, tant sur le plan scientifique que sur le plan humain :

— Thomas Schiex et Gérard Verfaillie, mes deux directeurs de thèse, pour leur disponibilité, l'excellence de leur encadrement, leur ouverture d'esprit, et leur soutien. Merci notamment pour le caractère scientifiquement stimulant de nos réunions, qui, de mon point de vue, ont fait du travail de recherche un pur plaisir.

— Francesca Rossi, de l'université de Padoue, et Patrice Perny, de l'université Paris 6, qui m'ont fait l'honneur de s'intéresser à mon travail en acceptant d'être rapporteurs de cette thèse.

— Malik Ghallab, directeur du LAAS-CNRS, et Nic Wilson, chercheur au Cork Constraint Computation Center, pour avoir accepté de participer à mon jury de thèse. Merci sincèrement à Nic de m'avoir invité à présenter mes travaux à un workshop ECAI'06. Je lui suis réellement reconnaissant de cette belle opportunité.

— Aux membres de mon "comité de thèse" réunis à l'issue de mes premières et deuxièmes années de thèse : Rachid Alami du LAAS-CNRS, Jean-Loup Farges de l'ONERA Toulouse, Jérôme Lang de l'IRIT, et Régis Sabbadin de l'INRA Toulouse. Merci pour leur lecture attentive de mes rapports d'avancement et pour les discussions que j'ai pu avoir avec eux par la suite.

— Plus généralement, merci aux personnes du groupe RIA du LAAS-CNRS et aux personnes de l'INRA pour la bonne ambiance de travail dont j'ai pu bénéficier.

Introduction

In the last decades, numerous formalisms have been developed to express and solve decision making problems. In such problems, an agent must make decisions consisting in either choosing actions and ways to fulfill them (as in action planning, task scheduling, or resource allocation), or choosing explanations of observed phenomena (as in diagnosis or situation assessment). These choices may depend on various parameters listed below:

1. *Plausibilities*: uncertainty measures, which we call *plausibilities*, may describe beliefs about the state of the environment. That is to say, the environment may be non deterministic.

2. *Feasibilities*: preconditions may have to be satisfied for a decision to be *feasible*.

3. *Utilities*: possible states of the environment and possible decisions do not generally have the same value for the decision maker's point of view. *Utilities* can be expressed to model costs, gains, risks, satisfaction degrees, hard requirements, and more generally, preferences (the notion of utility is not restricted here to its additive version).

4. *Sequential aspect and partial observabilities*: when time is involved, decision processes may be *sequential*. This means that there may be several decision steps, and that the values of some variables may be observed between two steps, as in chess where each player plays in turn and can observe the move of the opponent before playing again.

5. *Multi-agent aspect and partial controllabilities*: there may be adversarial or collaborative decision makers, each of them controlling a set of decisions.

In this thesis, we are interested in generic *sequential decision problems including plausibilities, feasibilities, and utilities*. Given (1) the plausibilities defined over the states of the environment, (2) the feasibility constraints on the decisions, (3) the utilities defined over the decisions and the states of the environment, and (4) the possible multiple decision steps, the objective is to provide a decision maker with optimal decision rules for the decision variables he controls, depending on the environment and of decisions of other agents.

Among the formalisms designed to solve problems included in this class, one can find:

- formalisms developed in the boolean satisfiability framework: the satisfiability problem (SAT), quantified boolean formulas, stochastic SAT [82], and extended stochastic SAT [82];

- formalisms developed in the very close constraint satisfaction framework: constraint satisfaction problems (CSPs [84]), valued/semiring CSPs [12] (covering classical, fuzzy, additive, lexicographic, probabilistic CSPs), mixed CSPs and probabilistic mixed CSPs [47], quantified CSPs [15], and stochastic CSPs [138];

- formalisms developed to represent uncertainties and extended to represent decision problems under uncertainties: Bayesian networks [96], Markov random fields [22] (also known as Gibbs networks), chain graphs [55], hybrid or mixed networks [36, 37], influence diagrams [64], unconstrained [68], asymmetric [131, 92], or sequential [67] influence diagrams, valuation networks [128], and asymmetric [130] or sequential [41] valuation networks;
- formalisms developed in the classical planning framework, such as STRIPS planning [49, 58], conformant planning [60], and probabilistic planning [77];
- formalisms such as Markov decision processes (MDPs), probabilistic, possibilistic, or using Spohn's epistemic beliefs [133, 142, 59], factored or not, possibly partially observable [111, 89, 119, 19, 18].

Many of these formalisms present interesting similarities:

- they include variables modeling the state of the environment (environment variables) or the decisions (decision variables);
- they use local functions modeling plausibilities, feasibilities, or utilities;
- they use operators either to combine local information (such as × to aggregate probabilities under independence hypothesis, + to aggregate gains and costs), or to synthesize a global information (such as + to compute a marginal probability, min or max to compute an optimal decision).

Even if the meaning of variables, functions, and combination or synthesis operators may be specific to each formalism, they can all be seen as *graphical models* in the sense that they exploit (implicitly or explicitly) a hypergraph of local functions between variables. This thesis shows that it is possible to build a generic algebraic framework subsuming many of these formalisms by reducing decision making problems to a sequence of so-called "variable eliminations" on an aggregation of local functions.

Motivations Building a generic framework and generic algorithms to represent and solve various decision making problems will be able to provide:

- *A better understanding*: a generic framework has an obvious theoretical and pedagogical interest, since it can bring to light similarities and differences between the formalisms covered and help people of different communities to communicate on a common basis.
- *An increased expressive power*: a generic framework may be able to capture problems that cannot be modeled in any existing formalism. This increased expressiveness should be reachable by capturing the essential algebraic properties of existing frameworks.
- *Generic algorithms*: ultimately, besides a generic framework, it should be possible to define generic algorithms capable of solving problems defined in this framework. This objective fits into a growing effort to identify common algorithmic approaches that were developed for solving different AI problems. It may also facilitate cross-fertilization by allowing each subsumed framework to reuse algorithmic ideas defined in another one.

Thesis outline This thesis is organized in two parts:

1. The first part, which focuses on knowledge representation, introduces a new generic framework for sequential decision making with uncertainties, feasibilities, and utilities.

   After the definition of some notations and notions (Chapter 1), we start by showing, with a catalog of existing formalisms for decision making, that a generic algebraic framework capturing many existing formalisms can be informally identified (Chapter 2).

   This generic framework, called the Plausibility-Feasibility-Utility (PFU) framework, is then formally introduced in three steps:

   - Algebraic structures enabling us to express generic forms of plausibilities, feasibilities, and utilities are introduced in Chapter 3. They specify how to combine and synthesize information.
   - These algebraic structures are exploited inside a network structure (graphical model), defined in Chapter 4. The basic elements involved in such networks are variables and local functions.
   - Problems over such networks are captured by the notion of queries, defined in Chapter 5. In the end, solving a decision making problem means answering a query.

2. The second part of the thesis focuses on generic algorithms able to answer queries.

   - The first generic algorithms presented in Chapter 6 are based on tree search and variable elimination. The second tries to exploit for the best the decomposition of a global information into local functions, and has a theoretical complexity exponential in the so-called constrained induced-width.
   - More advanced techniques which analyze the actual structure of a query are given in Chapter 7. This provides us with a generic computational architecture, called the multi-operator cluster DAG architecture, which explicitly expresses a decomposition of the computations to perform in order to answer queries.
   - Based on this architecture, Chapter 8 introduces structured tree search algorithms, which can be more or less sophisticated depending on whether they use some recording and/or bounds.
   - Last, Chapter 9 presents a generic implemented solver used to answer queries, which shows that the framework and the algorithms defined is this thesis are not just abstractions.

A table recapitulating the main notations used is available in Appendix A and the proofs of all lemmas, propositions, and theorems are given in Appendix B, in order to keep the reading fluid.

Part I

Representing decision making problems in the PFU framework

Chapter 1

Background notations and definitions

This small chapter introduces the essential objects used in the thesis, hence the interest of assimilating the few definitions given below. The main notions manipulated are variables, domains, local functions (called scoped functions), graphical models, combination and elimination operators, decision rules, and some vocabulary concerning graphs. Some of these notions are illustrated by a toy example, which also informally introduces the notions of plausibilities, feasibilities, utilities, partial observability, and controllability.

1.1 Basic definitions

Definition 1.1. *The* domain *of values of a variable x is denoted $dom(x)$ and for every $a \in dom(x)$, (x, a) denotes the assignment of x with value a. By extension, for a set of variables S, we denote by $dom(S)$ the Cartesian product of the domains of the variables in S, i.e. $dom(S) = \prod_{x \in S} dom(x)$. An element A of $dom(S)$ is called an* assignment *of S.*[1]

If A_1, A_2 are assignments of disjoint subsets S_1, S_2, then the concatenation of A_1 and A_2, denoted $A_1.A_2$, is the assignment of $S_1 \cup S_2$ where variables in S_1 are assigned as in A_1 and variables in S_2 are assigned as in A_2.

If A is an assignment of a set of variables S, the projection of A over a set of variables S', denoted $A^{\downarrow S'}$, is the assignment of $S \cap S'$ where variables are assigned to their value in A.

Definition 1.2. *(Scoped function) A* scoped function *is a pair (S, φ) where S is a set of variables and φ is a function mapping elements in $dom(S)$ to a given set E.*

In the following, we will often consider that S is implicit and denote a scoped function (S, φ) as φ alone. The set S of variables is called the scope *of φ and is denoted $sc(\varphi)$. If A is an assignment of a superset of $sc(\varphi)$ and A' is the projection of A onto $sc(\varphi)$, then $\varphi(A)$ will be used as an abbreviation of $\varphi(A')$.*

For example, a scoped function φ mapping assignments of $sc(\varphi)$ to elements of the boolean

[1] An assignment of $S = \{x_1, \ldots, x_k\}$ is actually a set of variable-value pairs $\{(x_1, a_1), \ldots, (x_k, a_k)\}$, but we assume that variables are implicit when using a tuple of values $(a_1, \ldots, a_k) \in dom(S)$.

lattice $\mathbb{B} = \{t, f\}$ is analogous to a constraint describing the subset of $dom(sc(\varphi))$ of authorized tuples in constraint networks.

From this, the general notion of graphical model can be defined:

Definition 1.3. *(Graphical model) A* graphical model *is a pair (V, Φ) such that $V = \{x_1, \ldots, x_n\}$ is a finite set of variables and $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ is a finite set of scoped functions whose scopes are included in V.*

The terminology of *graphical* models is used here simply because a set of scoped functions can be represented as a hypergraph whose hyperedges are the functions scopes. As we will see, this hypergraph captures a form of independence and induces parameters for the time and space complexity of our algorithms. This definition of graphical models generalizes the usual one used in statistics, defining a graphical model as a (directed or not) graph where the nodes represent random variables and where the structure captures probabilistic independence relations.

"Local" scoped functions in a graphical model give a space-tractable definition of a global function over all variables defined by their aggregation. For example, in a Bayesian network [96] a global probability distribution $P_{x,y,z}$ over x, y, z may be defined as the product (using operator $\times$) of a set of scoped functions $\{P_x, P_{y|x}, P_{z|y}\}$. Local scoped functions can also facilitate the projection of the information expressed by a graphical model onto a smaller scope. For example, in order to compute a marginal probability distribution $\mathcal{P}_{y,z}$ from the previous network, we can compute $\sum_x P_{x,y,z} = (\sum_x P_x \times P_{y|x}) \times P_{z|y}$ and avoid taking $P_{z|y}$ into account. Here the operator $\sum$ is used to project information onto a smaller scope by eliminating variable x. Operators used to combine scoped functions will be called *combination* operators, while operators used to project information onto smaller scopes will be called *elimination* operators.

Definition 1.4. *(Combination) Let φ_1, φ_2 be scoped functions to E_1 and E_2 respectively. Let $\otimes : E_1 \times E_2 \rightarrow E$ be a binary operator. The combination of φ_1 and φ_2, denoted by $\varphi_1 \otimes \varphi_2$, is the scoped function to E with scope $sc(\varphi_1) \cup sc(\varphi_2)$ defined by $(\varphi_1 \otimes \varphi_2)(A) = \varphi_1(A) \otimes \varphi_2(A)$ for all assignments A of $sc(\varphi_1) \cup sc(\varphi_2)$. $\otimes$ is called the* combination operator *of φ_1 and φ_2.*

In the rest of part I, all combination operators will be denoted $\otimes$.

Definition 1.5. *(Elimination) Let φ be a scoped function to E. Let op be an associative and commutative operator on E. The elimination of variable x from φ with op, denoted $op_x\, \varphi$, is a scoped function whose scope is $sc(\varphi) - \{x\}$ and whose value for an assignment A of its scope is $(op_x\, \varphi)(A) = op_{a \in dom(x)}\, \varphi(A.(x, a))$. In this context, op is called the* elimination operator *for x.*

The elimination of a set of variables $S = \{x_1, \ldots, x_k\}$ on φ is a function with scope $sc(\varphi) - S$ defined by $(op_S\, \varphi)(A) = op_{A' \in dom(S)}\, \varphi(A.A')$.

Hence, when computing $\sum_x (P_x \times P_{y|x} \times P_{z|x})$, scoped functions are aggregated using the combination operator $\otimes = \times$ and the information is synthesized by eliminating x using the elimination operator $+$. In the rest of Part I, $\oplus$ denotes elimination operators. Actually, the denomination of combination operator or elimination operator depends on the usage of an operator: for example $+$ can be used both as a combination operator to aggregate additive gains and costs, and as an elimination operator used to compute a marginal probability distribution.

In some cases, the elimination of a set of variables S with an operator *op* from a scoped function φ should only be performed on a subset of $dom(S)$ containing assignments that satisfy some property denoted by a boolean scoped function F. Then, one must compute for every $A \in dom(sc(\varphi) - S)$ the value $op_{A' \in dom(S), F(A')=t}\, \varphi(A.A')$. For simplicity and homogeneity, and in order to always use elimination over $dom(S)$, one can equivalently truncate φ so that elements of $dom(S)$ which do not satisfy the property expressed by F are mapped to a special value (denoted $\Diamond$) which is itself defined as an identity for *op*.

Definition 1.6. *(Truncation operator) The* unfeasible value $\Diamond$ *is a new special element and every elimination operator* $op : E \times E \to E$ *is extended to* $op : (E \cup \{\Diamond\}) \times (E \cup \{\Diamond\}) \to E \cup \{\Diamond\}$ *by* $op(\Diamond, e) = op(e, \Diamond) = e$ *for all* $e \in E \cup \{\Diamond\}$.

Let $\{t, f\}$ *be the boolean lattice. For any boolean* b *and any* $e \in E \cup \{\Diamond\}$*, we define* $b \star e$ *to be equal to* e *if* $b = t$ *and* $\Diamond$ *otherwise.* $\star$ *is called the truncation operator.*

Given a boolean scoped function F, the unfeasibility element $\Diamond$ and the truncation operator $\star$ make it possible to write quantities like $op_{A' \in dom(S), F(A')=t}\, \varphi$ as the elimination $op_S\, (F \star \varphi)$.

In order to solve decision problems, one usually wants to compute functions mapping the available information to a decision. The notion of *decision rules* will be used to formalize this:

Definition 1.7. *(Decision rule, policy) A* decision rule *for a variable* x *given a set of variables* S' *is a function* $\delta : dom(S') \to dom(x)$ *mapping each assignment of* S' *to a value in* $dom(x)$*. By extension, a decision rule for a set of variables* S *given a set of variables* S' *is a function* $\delta : dom(S') \to dom(S)$*. A set of decision rules is called a* policy*.*

If φ is a scoped function on a totally $\preceq$-ordered set E and if one computes $\max_S \varphi$, then a decision rule $\delta : dom(sc(\varphi) - S) \to dom(S)$ such that $\varphi(A.\delta(A)) \succeq \varphi(A.A')$ for all $(A, A') \in dom(sc(\varphi) - S) \times dom(S)$ is called an optimal decision rule. Similarly, if one computes $\min_S \varphi$, then we call optimal decision rule for S a decision rule $\delta : dom(sc(\varphi) - S) \to dom(S)$ such that $\varphi(A.\delta(A)) \preceq \varphi(A.A')$ for all $(A, A') \in dom(sc(\varphi) - S) \times dom(S)$. This means that optimal decision rules are examples of decision rules given by argmin and argmax.

Graph concepts In this thesis, we also need some definitions concerning graphs.

Definition 1.8. *Let* $\mathcal{G} = (V, H)$ *be a hypergraph (this means that* V *is a set of variables and* H *is a set of hyperedges over* V*, i.e. a subset of* 2^V*). The* primal graph *of* $\mathcal{G}$ *is the graph* $G = (V, E)$ *such that* E *contains an edge* $\{x, y\} \in V^2$ *iff there exists an hyperedge* h *in* H *such that* $\{x, y\} \subset h$*.*

Definition 1.9. *Let* $G = (V, E)$ *be a graph. A subset* S *of* V *is called a* clique *iff for all* x*,* y *in* S*,* $\{x, y\} \subset E$*.*

Definition 1.10. *A graph* $G = (V, E)$ *is a tree iff it is an undirected connected graph without cycle. It is a rooted tree iff it is a directed connected graph without cycle. The root of the tree is then the unique vertex without parents.*

Definition 1.11. *(Directed Acyclic Graph (DAG)) A directed graph* G *is a DAG iff it contains no directed cycle. When variables are used as vertices,* $pa_G(x)$ *denotes the set of parents of variable* x *in* G*. The set of non-descendants of* x*, denoted* $nd_G(x)$*, corresponds to the set of variables* y *such that there does not exist a directed path from* x *to* y *in* G*. The set of ancestors of* x *is the set of variables* y *such that there is a directed path from* y *to* x *in* G*.*

In the sequel, the cardinality of a set Γ is denoted $|\Gamma|$.

1.2 An illustrative example

The following toy example was created in order to better describe the notions of “plausibilities”, “feasibilities”, “utilities”, “obervability”, “decision variable”, “environment variable”, or “controllability”. It also illustrates how variables and local scoped functions can express a global information in a compact way. Eventually, this example shows why sequences of variable eliminations on combinations of scoped functions are of interest for sequential decision problems.

Example *John faces three doors A, B, C. One of the doors hides a treasure, another a gangster. John can decide to open one of the doors. The gangster will rob him 4,000€ but the treasure is worth 10,000€.*

Modeling To represent the environment and the decisions in a compact way, we introduce three variables: (1) two *environment variables*: one for the door behind which is the gangster, the “gangster door” (denoted ga), and one for the door behind which is the treasure, the “treasure door” (tr); (2) one *decision variable* (do), representing the door John decides to open. Every variable has $\{A, B, C\}$ as domain. Decision variables are variables whose value is controlled by an agent. The other variables are environment variables.

Then, we need two local *utility functions* U_1, U_2 to represent utilities: (1) U_1 expresses that if John opens the gangster door, he must pay $4,000$€ (soft constraint $do = ga$, with utility degree $-4,000$€ if satisfied, and 0 otherwise); (2) U_2 expresses that if John opens the treasure door, he wins $10,000$€ (soft constraint $do = tr$, with utility degree $10,000$€ if satisfied, and 0 otherwise). A soft constraint is also called a cost function.

Associated query Which door should John open if he knows that the gangster is behind door A and that the treasure is behind door C (no uncertainties)? Obviously, he should open door C.

Adding uncertainties

In real problems, the environment may not be completely known: there may be uncertainties (here called *plausibilities*) as well as possible *observations* on this uncertain environment. We make the treasure quest problem more complex in order to illustrate such aspects.

Example *The treasure and the gangster are not behind the same door, and all situations are equiprobable. John is accompanied by Peter. Each of them can decide to listen in to door A, B, or C to try to detect the gangster. The probability of hearing something is* 0.8 *if one eavesdrops at the gangster door ga,* 0.4 *at a door next to it, and* 0 *otherwise.*

Modeling We define four more variables to represent these new specifications:

- two decision variables li_J and li_P, with $\{A, B, C\}$ as domain, model the doors to which John and Peter listen in;
- two environment variables he_J and he_P, with $\{yes, no\}$ as domain, model whether John and Peter hear the gangster.

We then define four local *plausibility functions*:

- $P_1 : ga \neq tr$ and $P_2 = 1/6$ model the probability distribution over the gangster's and treasure's locations;

- $P_3 = P_{he_J \,|\, li_J, ga}$ defines the probability that John hears something given the door at which he eavesdrops and the gangster door;

- similarly, P_4 corresponds to $P_{he_P \,|\, li_P, ga}$.

Implicitly, the local plausibilities satisfy *normalization conditions*. First, as the treasure and the gangster are somewhere, $\sum_{ga,tr}(P_1 \times P_2) = 1$. Second, as John and Peter hear something or not, $\sum_{he_J} P_3 = 1$ and $\sum_{he_P} P_4 = 1$.

Associated queries Which decision rules maximize the expected utility, if first Peter and John eavesdrop, and then John decides to open a door knowing what has been heard?

Such a query can be answered using a standard *decision tree*. In this tree, variables can be considered in the order $li_J \to li_P \to he_J \to he_P \to do \to ga \to tr$. This order corresponds to the following sequence of events: first, John and Peter choose a door to eavesdrop at, then they listen and depending on what they have heard, John decides which door to open; finally the gangster and the treasure are behind given doors with a certain probability. An internal node n in the decision tree corresponds to a variable x. An edge in the decision tree is labeled with an assignment (x, a) of the variable x associated with the node above. Such an edge is also weighted by the probability $P((x, a) \,|\, A)$, where A is the assignment corresponding to the path from the root to x.

The utility of a leaf node is the global utility $(U_1 + U_2)(A)$ of the complete assignment A associated with it. The utility of an internal decision node is given by the value of an optimal children (and it is possible to record an associated optimal decision). The utility of an internal environment node is given by the probabilistic expected utility of the values of its children nodes. The global expected utility is the utility of the root node. It is proved [103] that such a decision tree procedure can be reduced to the computation of

$$\max_{li_J, li_P} \sum_{he_J, he_P} \max_{do} \sum_{ga,tr} ((\prod_{i \in [1,4]} P_i) \times (\sum_{i \in [1,2]} U_i))$$

In other words, the decision tree procedure is equivalent to a *sequence of variable eliminations on a combination of local functions*. Optimal decision rules can be recorded using argmax during the computation.

Different elimination sequences represent different problems or situations: if John thinks that Peter is a traitor and if he lets him choose a door to listen in to first (pessimistic attitude concerning the other agent), the sequence of eliminations $\min_{li_P} \max_{li_J} \sum_{he_J, he_P} \max_{do} \sum_{ga,tr}$ is adequate, because it eliminates li_P with min. If Peter does not even tell John what he has heard, meaning that John does not observe he_P, then the sequence of eliminations becomes $\min_{li_P} \max_{li_J} \sum_{he_J} \max_{do} \sum_{he_P} \sum_{ga,tr}$.

Adding feasibilities

In some cases, certain conditions must be satisfied for a decision to be feasible. For example, if two players accept to respect chess rules, then a move is feasible if and only if it satisfies the rules. Note that unfeasibility is different from infinite utility, because for example none of the players can make an impossible move, whereas each of them may achieve a checkmate, which yields an infinite negative utility for his adversary.

Example *John and Peter cannot eavesdrop at the same door and door A is locked.*

Modeling Two local *feasibility functions* are added to represent this new situation: $F_1 : li_J \neq li_P$ and $F_2 : do \neq A$. We assume that at least one decision is feasible in any situation (no dead-end). This is represented by two normalization conditions on feasibilities: $\vee_{li_J,li_P} F_1 = t$ and $\vee_{do} F_2 = t$. The classical decision tree procedure which can be used to answer the query is then equivalent to the computation of

$$\min_{li_P} \max_{li_J} \sum_{he_J} \max_{do} \sum_{he_P} \sum_{ga,tr} \left(\left(\bigwedge_{i\in[1,2]} F_i \right) \star \left(\prod_{i\in[1,4]} P_i \right) \times \left(\sum_{i\in[1,2]} U_i \right) \right)$$

which uses the truncation operator $\star$ to mask unfeasible decisions. Again, this corresponds to a sequence of variable eliminations on a combination of scoped functions.

In the end, the knowledge modeled with variables and local functions forms a *composite* graphical model defined by a DAG capturing normalization conditions on plausibilities and feasibilities (Figure 1.1(a)), [2] and a network of local functions (Figure 1.1(b)). The network involves several types of variables (decision and environment variables) and several types of local functions (local utility, plausibility, and feasibility functions).

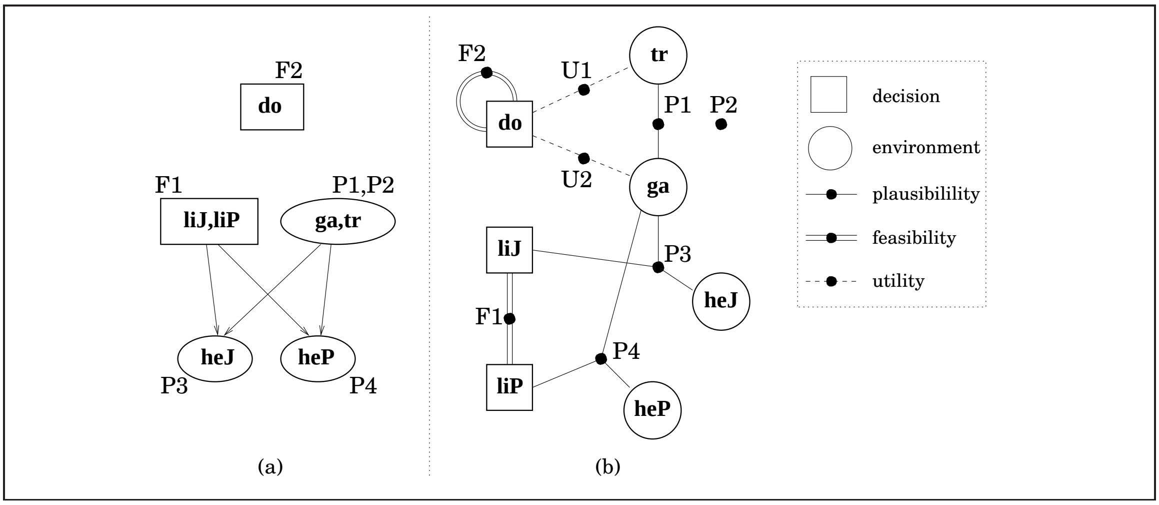


Figure 1.1: Composite graphical model (a) DAG capturing normalization conditions; (b) Network of local functions.

John's treasure quest is an example which illustrates the notion of a sequential decision problem involving plausibilities, feasibilities, and utilities. This notion will be used in the next chapters.

[2] If P denotes the set of local plausibility functions associated with a node corresponding to a set of variables S, then this means that $\sum_S (\prod_{P_i \in P} P_i) = 1$. If F denotes the set of local feasibility functions associated with a node corresponding to a set of variables S, then this means that $\vee_S (\wedge_{F_i \in F} F_i) = t$.

Chapter 2

A guided tour of frameworks for decision making

In order to build a generic framework for decision making, the very first step consists in understanding and analyzing existing formalisms. Their first characteristic is that they are numerous. The reason is that in the last decades, many efforts were made in the AI community in order to build new representation schemes or extensions of existing ones. This led to many proposals, which have different modeling abilities. Some can model preferences, other can model only hard requirements. Some can model uncertainties, others cannot. Some can model sequential decision making, whereas others are designed for one-shot decision processes.

This chapter presents a **non-exhaustive** catalog of such formalisms. This catalog has two main features:

- It is incremental, in the sense that it shows how basic frameworks like Satisfiability problems, Constraint Satisfaction Problems [84], Bayesian Networks [96], classical planning [49, 58], or Markov Decision Processes [111, 89] were extended to integrate the notion of uncertainties for some of them, or the notion of preferences and decisions for others.

- It analyzes the similarities and differences of existing frameworks in terms of knowledge representation. This analysis tries to show that (1) many formalisms reason on the notion of variables and local scoped functions between these variables, and (2) queries asked in these formalisms can be reduced to the computation of a sequence of variable eliminations on a combination of scoped functions, using various operators. Points (1) and (2) can be seen as the guiding line of this catalog.

2.1 SAT-based decision frameworks

The first and probably the oldest framework for decision making is the Satisfiability (SAT) problem. As we shall see, the basic SAT problem was extended to formalisms like quantified boolean formulas or stochastic SAT [82], in order to model uncontrollable variables and partial observabilities.

2.1.1 The satisfiability problem

We start with a few definitions on propositional logic. The syntax of this logic is based on boolean variables, usually called propositional variables or atoms. These variables with $\{t, f\}$ as domain represent properties which are either true or false.

Definition 2.1. *Let V be a finite set of boolean variables.* Boolean formulas *are defined inductively by the following rules:*

1. *if $x \in V$, then x is a boolean formula,*

2. *if φ is a boolean formula, then $\neg\varphi$ is a boolean formula,*

3. *if φ and ψ are boolean formulas, then $\varphi \wedge \psi$ is a boolean formula.*

It is also possible to define $\varphi \vee \psi$ as $\neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \rightarrow \psi$ as $\neg\varphi \vee \psi$. The symbols $\neg$, $\wedge$, $\vee$, and $\rightarrow$ are called logical connectives*.*

In order to provide boolean formulas with a *truth value*, one must define a semantics for the connectives. First, given a boolean variable x, formula x is true iff x is assigned with value *true*.[1] Second, $\neg\varphi$ is true iff φ is false. Third, $\varphi \wedge \psi$ is true iff both φ and ψ are true. This implies that $\varphi \vee \psi$ is true iff φ or ψ is true.

Definition 2.2. *A* literal *is a boolean variable or its negation. A* clause *is a disjunction of literals. A boolean formula is in* conjunctive normal form *if it is a conjunction of clauses.*

Definition 2.3. *The Satisfiability problem (SAT) consists in determining whether a boolean formula in conjunctive normal form has an assignment of its variables which makes the formula true.*

Note that every boolean formula can be put in a conjunctive normal form. SAT enables various reasoning tasks to be modeled, for example in hardware design and more generally in formal verification.

Example 2.4. $(x \vee y) \wedge (y \vee \neg z) \wedge (\neg y \vee z)$ *is a boolean formula in conjunctive normal form. It is satisfiable since for example the assignment $(x,t).(y,t).(z,t)$ makes the formula true. By assuming $f \prec t$, this SAT instance can be seen as a binary optimization problem on boolean variables, consisting in computing*

$$val = \max_{x,y,z} ((x \vee y) \wedge (y \vee \neg z) \wedge (\neg y \vee z)) \tag{2.1}$$

Indeed, if $val = f$, then the formula is not satisfiable; otherwise the formula is satisfiable and a corresponding optimal decision rule for $\{x, y, z\}$ defines a solution. Hence, SAT can be considered as the computation of max-eliminations on a conjunction of clauses.

2.1.2 Quantified boolean formulas: towards pessimistic indeterminism and partial observabilities

The basic SAT problem was extended to Quantified Boolean Formulas (QBFs [57]) in order to model decision problems involving

[1] In propositional logic, the assignment of a propositional variable is called a substitution.

- *uncontrollable* variables that may take any of their values. These variables are quantified with the universal quantifier $\forall$. The other variables are quantified with $\exists$ or are not quantified;
- *partial observabilities*: the alternation of $\exists$ and $\forall$ quantifiers makes the decision problem sequential and the value of some variables may be observed between two decision steps.

The syntax of QBFs is defined by adding the existential and universal quantifiers to the propositional logic.

Definition 2.5. *Let V be a finite set of boolean variables.* Quantified Boolean Formulas *(QBFs) are defined inductively by the following rules:*

1. *if $x \in V$, then x is a QBF,*
2. *if φ is a QBF, then $\neg\varphi$ is a QBF,*
3. *if φ and ψ are QBFs, then $\varphi \wedge \psi$ is a QBF,*
4. *if φ is a QBF and $x \in V$, then $\exists x\varphi$ and $\forall x\varphi$ are QBFs.*

The meaning of a QBF is defined by the standard semantics of the connectives and of the quantifiers, which is: "if φ is a boolean formula, then $\exists x\varphi$ is true iff $\varphi((x,t)) \vee \varphi((x,f))$ is and $\forall x\varphi$ is true iff $\varphi((x,t)) \wedge \varphi((x,f))$ is".

Example 2.6. *Let us consider the boolean formula $(x \vee y) \wedge (y \vee \neg z) \wedge (\neg y \vee z)$ introduced in Example 2.4. Let us assume that variable y is not controllable. Then, does there exist a value for x such that for every value of y, there exists a value for z such that the three clauses $x \vee y$, $y \vee \neg z$, and $\neg y \vee z$ are satisfied? This query can be formalized using a QBF in the so-called* prenex conjunctive normal form, *which is $\exists x\forall y\exists z((x \vee y) \wedge (y \vee \neg z) \wedge (\neg y \vee z))$. The $\forall$-quantification means that variable y may take any of its values. The alternation of $\forall$ and $\exists$ quantifiers means that the value of y is observed only after the assignment of x. By assuming $f \prec t$, this QBF can also be written*

$$\max_x \min_y \max_z((x \vee y) \wedge (y \vee \neg z) \wedge (\neg y \vee z)) \tag{2.2}$$

*Equation 2.2 corresponds to a sequence of eliminations (*max *over x,* min *over y,* max *over z) on a conjunction of clauses. Its value can be shown to equal true, and an optimal policy enabling the three clauses to always be satisfied can be described as "set x to true; then, if y takes value true, set z to true; otherwise, if y takes value false, set z to false".*

An example of QBF whose value is false is $\exists x\forall y\exists z(y \wedge (x \vee z) \wedge (\neg x \vee \neg z))$.

2.1.3 Stochastic SAT and extended stochastic SAT: towards a stochastic indeterminism

In QBFs, uncontrollable variables can take any of their values. Therefore, QBFs can model a pessimistic indeterminism, in the sense that all possible situations are considered. In another direction, the SAT problem was extended to integrate stochastic indeterminisms (i.e. probabilistic uncertainties). The corresponding extension is the Stochastic SAT (SSAT [82]) framework, which uses a special quantifier Я to quantify random variables.

Definition 2.7. *Let V be a finite set of boolean variables.* Stochastic SAT *(SSAT) formulas are defined inductively by the following rules:*

1. *every boolean formula is an SSAT formula;*

2. *if φ is an SSAT formula and $x \in V$, then $\exists x \varphi$ and Я$x\varphi$ are SSAT formulas.*

Given a boolean formula φ, the semantics of Я$x\varphi$ is given by the expected truth value of φ, i.e. $val($Я$x\varphi) = 0.5 \cdot \varphi((x,t)) + 0.5 \cdot \varphi((x,f))$. Hence, the semantics of the Я quantifier enables mutually independent boolean random variables to be modeled: Яx means that variable x takes value t or f with a probability of 0.5. The value of $\exists x \varphi$ becomes $\max(\varphi((x,t)), \varphi((x,f)))$ instead of $\varphi((x,t)) \vee \varphi((x,f))$. If φ is a boolean formula, then its value is 1 if the formula is true and 0 otherwise. $\wedge$ becomes $\times$, in order to be able to combine truth values with probabilities.

Example 2.8. *Let us update the unsatisfiable example $\exists x \forall y \exists z (y \wedge (x \vee z) \wedge (\neg x \vee \neg z))$ given for QBFs in a less pessimistic form, where y takes each of its values with a probability of 0.5. The corresponding SSAT formula is $\exists x$Я$y \exists z (y \wedge (x \vee z) \wedge (\neg x \vee \neg z))$. Its value is given by the semantics of the connectives and of the quantifiers $\exists$ and Я. It equals $\max_x \sum_y 0.5 \cdot \max_z (y \times (x \vee z) \times (\neg x \vee \neg z))$, which is equivalent to:*

$$\max_x \sum_y \max_z (0.5 \times (y \times (x \vee z) \times (\neg x \vee \neg z))) \tag{2.3}$$

The value of this sequence of alternating max- and sum-eliminations on a product of scoped functions is 0.5. It corresponds to the probability for the formula $y \wedge (x \vee z) \wedge (\neg x \vee \neg z)$ to be satisfied.

An easier decision problem associated with SSAT is to determine whether the value of an SSAT formula is greater than a threshold θ. Also, in a version called extended SSAT [82], the universal quantifier $\forall$ is added, the semantics of $\forall x \varphi$ being $\min(\varphi((x,t)), \varphi((x,f)))$.

2.2 CSP-based decision frameworks

Similarly to the SAT framework, the basic CSP formalism was extended in order to improve its abilities to model sequential decision problems involving plausibilities, feasibilities, and utilities. But sequences of eliminations, hidden or not, are still present.

2.2.1 Constraint satisfaction problems

Constraint Satisfaction Problems (CSPs [84]), also known as Constraint Networks (CNs), are graphical models involving scoped functions which are constraints. These constraints can model either hard preferences, or impossibilities.

Definition 2.9. *A CSP is a pair (V, C) where:*

- *V is a finite set of variables;*

- *C is a finite set of constraints. A constraint c is a scoped function (S, φ) where $S \subset V$ is the set of variables on which the constraint holds and $\varphi : dom(S) \rightarrow \{t, f\}$ is a boolean function* *defining the set of assignments of S satisfying the constraint.*[2]

The usual query on a CSP (V, C) can be formulated as "Is there an assignment of V satisfying all the constraints in C?". If the answer is yes (resp. no), the CSP is said to be consistent (resp. inconsistent). By setting $f \prec t$, this decision problem can be reduced to the computation of:

$$\max_V \left(\bigwedge_{c \in C} c \right) \tag{2.4}$$

This quantity can be computed by performing max-eliminations on a conjunction of constraints. If it equals t, then an optimal decision rule for V defines a *solution* (an assignment of V satisfying all the constraints). If it equals f, then the CSP is inconsistent.

Example 2.10. *One must color each vertex of the graph in Figure 2.1 so that two ajdacent vertices have different colors. The available colors are (r)ed, (g)reen, and (b)lue. This problem can be modeled as a CSP (V, C) where*

- $V = \{x_1, x_2, x_3\}$ *and* $dom(x) = \{r, g, b\}$ *for each* $x \in V$*;*
- $C = \{c_1, c_2, c_3\}$ *is a set of constraints defined by* $c_1 : x_1 \neq x_2$*,* $c_2 : x_2 \neq x_3$*,* $c_3 : x_1 \neq x_3$*.*

Thus, one variable is associated with each vertex, the assignment of this variable specifies the vertex color, and binary difference constraints are defined. $(x_1, r).(x_2, g).(x_3, b)$ *is a solution for this CSP. In a two color version where* $dom(x) = \{r, g\}$ *for each* $x \in V$*, the CSP is inconsistent.*

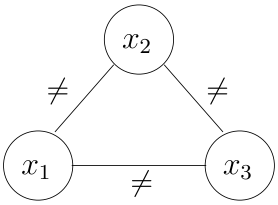


Figure 2.1: Graph coloring problem.

2.2.2 Extension to non-binary uncertainties and utilities: soft constraints

The CSP formalism can model hard constraints which express hard requirements or impossibilities. It was extended in order to represent *soft constraints* expressing soft preferences (such as costs or risks) or uncertainties (such as probabilities or possibilities). This led to formalisms like additive [126], possibilistic [122], probabilistic [44], partial [52], fuzzy [42], or lexicographic CSPs [46]. These extensions as well as usual CSPs are covered by two generic algebraic frameworks: the valued CSP [123] and semiring-based CSP [10, 11] frameworks.

[2] Usually, a constraint is defined as a pair (S, R) where S is the scope of the constraint and R, called a relation, is the set of tuples satisfying the constraint. The definition we take just considers the boolean characteristic function of R instead of R itself.

Valued CSP (VCSP [123])

In a VCSP, the violation of one soft constraint induces a violation degree. Violation degrees are combined using a combination operator $\otimes$ [3] corresponding to min, max, +, ×… The algebraic structure defining the set of violation degrees and the operator $\otimes$ is called a *valuation structure.*

Definition 2.11. *A* valuation structure *is a triple* $(E, \otimes, \preceq)$ *such that:*

- $(E, \preceq)$ *is a totally ordered set equipped with a maximal element* $\top$ *(unacceptable violation) and a minimal element* $\bot$ *(no violation);*
- $\otimes$ *is an associative, commutative, monotonic operator on* E*, with* $\bot$ *as an identity* $(e \otimes \bot = e)$ *and* $\top$ *as an annihilator* $(e \otimes \top = \top)$*.*

$\bot$ is an identity for $\otimes$ because the combination of a violation degree e with no violation yields an unchanged violation degree. $\top$ is an identity for $\otimes$ because the combination of an unacceptable violation with any other violation leads to an unacceptable violation. The monotonicity of $\otimes$ ensures that if a local violation degree decreases, then the global violation degree cannot increase.

Definition 2.12. *A Valued CSP (VCSP) on a valuation structure* $(E, \otimes, \preceq)$ *is a pair* (V, C) *where*

- V *is a finite set of variables;*
- C *is a finite set of* soft constraints*. A soft constraint* c *is a scoped function* (S, φ) *where* $S \subset V$ *is the set of variables on which the constraint holds and* φ *is a function* $dom(S) \to E$ *associating a violation degree with each assignment of* S*.*

A usual query on a VCSP is to search for a complete assignment which has a minimal violation degree. This problem can be solved by computing:

$$\min_{V} \left(\underset{c \in C}{\otimes} c \right) \tag{2.5}$$

An optimal decision rule for V defines a solution for the VCSP. Equation 2.5 is a sequence of min-eliminations on a $\otimes$-combination of scoped functions.

Example 2.13. *Let us soften the inconsistent two color version of the graph coloring problem of Example 2.10. If two adjacent vertices have the same color, this induces a cost of* 1*. Colors (r)ed and (g)reen are available for each vertex. Coloring a vertex in red costs* 1 *and coloring a vertex in green costs* 2*. We assume that costs are additive, i.e. we use the valuation structure* $(\mathbb{R}^+ \cup \{+\infty\}, \leq, +)$*, with* $e + (+\infty) = +\infty$*.* 0 *corresponds to no violation and* $+\infty$ *to an infinite cost.*

A VCSP modeling this new problem is the couple $(V, C) = (\{x_1, x_2, x_3\}, \{c_i \mid i \in [1, 6]\})$*, where*

- $c_1 = (\{x_1\}, \varphi)$*,* $c_2 = (\{x_2\}, \varphi)$*,* $c_3 = (\{x_3\}, \varphi)$*, where* $\varphi(r) = 1$ *and* $\varphi(g) = 2$ *(cost* 1 *for value red and cost* 2 *for value green);*
- $c_4 = (\{x_1, x_2\}, \varphi')$*,* $c_5 = (\{x_2, x_3\}, \varphi')$*,* $c_6 = (\{x_1, x_3\}, \varphi')$*, where* $\varphi'(r, r) = \varphi'(g, g) = 1$ *and* $\varphi'(r, g) = \varphi'(g, r) = 0$ *(cost* 1 *if two adjacent vertices have the same color, no violation otherwise).*

[3] In the usual definition of VCSP, this operator is denoted $\oplus$. We decide to adapt this notation because this operator is actually a combination operator and not an elimination one.

It is possible to show that $A = (x_1, g).(x_2, r).(x_3, r)$ is an optimal solution for this VCSP, with a cost of $\sum_{i\in[1,6]} c_i(A) = 2 + 1 + 1 + 0 + 1 + 0 = 5$.

Semiring-based CSP [10, 11]

In the semiring-based CSP formalism, soft constraints are also defined. They associate with each assignment of their scope a satisfaction degree in a *totally or partially* ordered set E. This set is equipped with two operators, $\otimes$ and $\oplus$, which satisfy some sensible algebraic properties making the structure $(E, \oplus, \otimes)$ a "c-semiring":

Definition 2.14. *A triple $(E, \oplus, \otimes)$ is a c-semiring iff:*

- *$(E, \oplus, \otimes)$ is a commutative semiring (cf. Definition 3.2 page 53); the identity of $\oplus$ is denoted 0_E and the identity of $\otimes$ is denoted 1_E;*
- *$\oplus$ is idempotent and 1_E is an annihilator for $\oplus$.*

Informally, $\otimes$ is a combination operator used to combine satisfaction degrees and $\oplus$ is an elimination operator enabling to synthesize a satisfaction degree obtained from two values of the c-semiring. These operators are associative and commutative so that the result of a combination or of a synthesis does not depend on the way they are performed. 0_E, which is an annihilator for $\otimes$, is associated with complete dissatisfaction (the combination of any satisfaction degree with a complete dissatisfaction yields a complete dissatisfaction), whereas 1_E, which is an identity for $\otimes$ and an annihilator for $\oplus$, stands for a complete satisfaction degree.

The idempotency of $\oplus$ enables a partial order $\preceq$ to be defined, as $(x \preceq y) \leftrightarrow (x \oplus y = y)$. It is shown that $(E, \preceq)$ is a lattice (a lattice is a partially ordered set in which any two elements have a supremum denoted *sup* and an infimum denoted inf) whose supremum is given by $\oplus$, i.e. $x \oplus y = sup(x, y)$. This shows that $\oplus$ enables one to synthesize a kind of maximum satisfaction degree. More precisely, we have $0_E \preceq x \preceq 1_E$ for all $x \in E$. This means that 0_E (complete dissatisfaction) is the minimal element in the lattice whereas 1_E (complete satisfaction) is the maximal one.

Once the c-semiring structure is defined, soft constraint satisfaction problems can be introduced. The definition of a semiring-based CSP (V, C) is exactly the same as Definition 2.12 of VCSPs, except that the valuation structure $(E, \otimes, \preceq)$ is replaced by a c-semiring $(E, \oplus, \otimes)$.

An optimal solution of a semiring-based CSP (V, C) is an assignment A of V such that there is no other assignment A' of V satisfying $\otimes_{c\in C} c(A) \prec \otimes_{c\in C} c(A')$. In other words, a solution is a non-dominated assignment. The best value of a semiring-based CSP is defined by $\oplus_V(\otimes_{c\in C} c)$. One (or all) optimal solution(s) can be recorded during the computation of this $\oplus$-elimination on a $\otimes$-combination of scoped functions. Compared to VCSPs, semiring-based CSPs are more expressive because they can deal with partial orders. When the order $\preceq$ induced by $\oplus$ is total, VCSPs and semiring-based CSPs are equivalent [12].

Example 2.15. *Assume that the cost induced by the color used for each vertex, and the cost induced by the existence of adjacent vertices having the same color are not commensurable. In order to model such a situation, we use the c-semiring $(E, \oplus, \otimes)$, where:*

- $E = (\mathbb{R}^- \cup \{-\infty\}) \times (\mathbb{R}^- \cup \{-\infty\})$; *a pair* (e, e') *models a (cost-color,cost-adjacence) pair: it means that the colors used for each vertex induce a cost of* e*, and that the satisfaction degree induced by adjacent vertices having the same color is* e';
- $\oplus$ *is defined by* $(e_1, e'_1) \oplus (e_2, e'_2) = (\max(e_1, e_2), \max(e'_1, e'_2))$;
- $\otimes$ *is defined by* $(e_1, e'_1) \otimes (e_2, e'_2) = (e_1 + e_2, e'_1 + e'_2)$.

The problem can be modeled by the semiring-based CSP $(V, C) = (\{x_1, x_2, x_3\}, \{c_i \,|\, i \in [1, 6]\})$ *where:*

- $c_1 = (\{x_1\}, \varphi)$, $c_2 = (\{x_2\}, \varphi)$, $c_3 = (\{x_3\}, \varphi)$, *where* $\varphi(r) = (-1, 0)$ *and* $\varphi(g) = (-2, 0)$ *(cost of* 1 *for value red and cost of* 2 *for value green);*
- $c_4 = (\{x_1, x_2\}, \varphi')$, $c_5 = (\{x_2, x_3\}, \varphi')$, $c_6 = (\{x_1, x_3\}, \varphi')$, *where* $\varphi'(r, r) = \varphi'(g, g) = (0, -1)$ *and* $\varphi'(r, g) = \varphi'(g, r) = (0, 0)$ *(cost of* 1 *if two adjacent vertices have the same color, no violation otherwise).*

For example, the satisfaction degree of assignment $A = (x_1, g).(x_2, r).(x_3, r)$ *is* $\otimes_{i\in[1,6]} c_i(A) = (-2, 0) \otimes (-1, 0) \otimes (-1, 0) \otimes (0, 0) \otimes (0, -1) \otimes (0, 0) = (-4, -1)$. *It is possible to show that* A *is an optimal solution, as well as* $A' = (x_1, r).(x_2, r).(x_3, r)$*, which has a value* $(-3, -3)$ *which is not comparable with* $(-4, -1)$. *The best value for this semiring-based CSP is* $(-3, -1)$*, but there is no assignment that achieves this supremum.*

2.2.3 Modeling uncontrollabilities and partial observabilities: mixed CSP

Similarly to the extensions performed from SAT to QBF, the basic CSP framework was also extended to model situations involving uncontrollable variables and partial observabilities. A first step towards indeterminism was made with the Mixed CSP formalism [45], which distinguished controllable variables representing the decisions from uncontrollable variables representing the state of the environment (hence the name of *mixed* CSP).

Definition 2.16. *A mixed CSP is a tuple* (V, C, K) *where*

- V *is a set of variables, partitioned between decision variables and environment variables;*[4]
- C *is a set of hard constraints, each of which involves at least one decision variable;*
- K *is a set of hard constraints involving only environment variables.*

The constraints in C define constraints on the decisions, whereas the constraints in K restrict the possible environments. As constraints in K do not involve decision variables, it is assumed that decisions do not influence the state of the environment. This assumption is called the *contingency assumption.*

Definition 2.17. *A complete assignment of the environment variables is called a world. A complete assignment of the decision variables is called a decision.*

A world is possible if it satisfies every constraint in K. *A possible world is covered by a decision if this world together with this decision satisfy every constraint in* C.

[4] Mixed CSP call the environment variables “contingent variables”. We adapt this terminology in order to make the comparison with other formalisms easier.

Two tasks, defining distinct observational situations, are associated with a mixed CSP:

1. If the state of the environment is completely observed before making the decision, the goal is to seek a *conditional decision rule*, which associates with each possible world a decision such that the number of covered worlds is maximized.

2. If the decision maker does not observe the environment before making his decision, the goal is to compute an *unconditional decision rule* covering as many worlds as possible.

Example 2.18. *Let us use the graph coloring problem of Example 2.10 again. In this new example, the colors of vertices x_2 and x_3 are not controlled. They are determined by some external phenomena independent of the color chosen for x_1. The contingency assumption therefore holds. The only available knowledge is that vertices x_2 and x_3 are of different colors (constraint $k_1 : x_2 \neq x_3$), x_2 is not blue (constraint $k_2 : x_2 \neq b$), and x_3 is red whenever x_2 is green (constraint $k_3 : (x_2 = g) \to (x_3 = r)$). The constraints on the decisions specifying that two adjacent vertices must have different colors still hold (constraints $c_1 : x_1 \neq x_2$ and $c_2 : x_1 \neq x_3$).*

This problem can be modeled by the mixed CSP (V, C, K) where $V = \{x_1, x_2, x_3\}$, $C = \{c_1, c_2\}$, and $K = \{k_1, k_2, k_3\}$. x_1 is the unique decision variable and x_2, x_3 are environment variables.

Three assignments of $\{x_2, x_3\}$ exist which satisfy the constraints in K, i.e. there are three possible worlds. If the colors of x_2 and x_3 are known when a color is chosen for x_1, then an optimal conditional *decision rule, covering the three possible worlds defined by K, is given below.*

Possible worlds for $\{x_2, x_3\}$	*Conditional decision for x_1*
$(x_2, r).(x_3, g)$	b
$(x_2, r).(x_3, b)$	g
$(x_2, g).(x_3, r)$	b

If the colors of x_2 and x_3 are not known before a color is chosen for x_1, then an optimal unconditional *decision is (x_1, b). It covers two worlds among the three possible ones.*

What is the link between these solutions and sequences of eliminations? First, given an assignment A of $\{x_2, x_3\}$, there exists an assignment of x_1 covering A iff $\max_{x_1}((\prod_{i\in[1,2]} c_i(A)) \times (\prod_{i\in[1,3]} k_i(A))) = 1$. An associated optimal decision is given by argmax. More generally, when the decision maker is aware of the colors of x_2 and x_3 before choosing the color of x_1, the number of covered worlds is

$$\sum_{x_2,x_3} \max_{x_1}((\prod_{i\in[1,2]} c_i) \times (\prod_{i\in[1,3]} k_i)) \tag{2.6}$$

An optimal decision rule $\delta_{x_1} : dom(\{x_2, x_3\}) \to dom(x_1)$ for x_1 corresponds to what is called an optimal conditional decision rule in the mixed CSP terminology.

Similarly, when x_2 and x_3 are not observed before assigning x_1, the number of worlds covered by an unconditional decision is

$$\max_{x_1} \sum_{x_2,x_3} ((\prod_{i\in[1,2]} c_i) \times (\prod_{i\in[1,3]} k_i)) \tag{2.7}$$

An unconditional decision rule for x_1 is simply obtained using argmax.

In both cases, the problems associated with a mixed CSP can be reduced to the computation of a sequence of eliminations ($\max\sum$ *or* $\sum\max$*) eliminating decision variables using* max *and environment variables using* $\sum$*. The scoped functions are the constraints in* C *and* K*.*

2.2.4 Quantified CSP for modeling multi-step decision processes

The sequential aspect in mixed CSPs is reduced to a unique decision step, either before or after an observation step. In order to model multi-step decision processes where some variables are uncontrollable and may take any of their values, Quantified CSPs (QCSPs [15]) were introduced.

QCSP is to CSP what QBF is to SAT. This means that the only difference from the knowledge modeling point of view between QCSP and QBF is that clauses are replaced by constraints.

Definition 2.19. *A QCSP on a set of variables* V *is a formula of the form* $Q(c_1 \wedge \ldots \wedge c_m)$ *where:*

- Q *is a sequence of quantifiers* $(Q_1x_1)(Q_2x_2)\ldots(Q_nx_n)$ *such that each* Q_i *equals* $\exists$ *or* $\forall$ *and each variable of* V *appears exactly once in* Q*;*

- $c_1, \ldots, c_m$ *are constraints whose scope is included in* V*.*

Definition 2.20. *The value of a QCSP* q *is defined inductively as follows (with* $f \prec t$*):*

- *if* $q = t$ *(*t *corresponds to a constraint always taking value true), then* $val(q) = t$*, and if* $q = f$*, then* $val(q) = f$*;*

- *if* $q = (\exists x_1)(Q_2x_2)\ldots(Q_nx_n)(c_1 \wedge \ldots \wedge c_m)$*, then* $val(q) = \max_{a \in dom(x_1)} val(q'(a))$*, where* $q'(a) = (Q_2x_2)\ldots(Q_nx_n)((c_1 \wedge \ldots \wedge c_m)(x_1, a))$*;*

- *if* $q = (\forall x_1)(Q_2x_2)\ldots(Q_nx_n)(c_1 \wedge \ldots \wedge c_m)$*, then* $val(q) = \min_{a \in dom(x_1)} val(q'(a))$*, where* $q'(a) = (Q_2x_2)\ldots(Q_nx_n)((c_1 \wedge \ldots \wedge c_m)(x_1, a))$*.*

As in QBFs, problems associated with QCSPs can be answered using sequences of min- and max-eliminations on a conjunction of constraints.

2.2.5 Integrating probabilistic uncertainties: stochastic CSP

Stochastic CSPs [138] enhance the CSP framework to model probabilistic uncertainties on uncontrollable variables, just as SSAT enhances SAT to be able to express stochastic indeterminisms.

Similarly to SSAT, SCSPs tackle multi-step decision making problems the goal of which is to maximize the probability that all constraints are satisfied or to make that probability greater than a given threshold θ. Globally, SCSPs are defined by an alternation of decision-observation steps. In a one-step SCSP, one must assign decision variables in a set D_1 without observing random variables in a set S_1. In a two-step SCSP, one first assigns decision variables in a set D_1, then observes random variables in a set S_1, then assigns decision variables in a set D_2 depending on the observations made, but without observing the values of random variables in a set S_2. A k-step SCSP is defined similarly.

Definition 2.21. *A Stochastic CSP (SCSP) is a tuple* (V, P, C) *where:*

- *V is a sequence of variables. The order in which the variables appear in the sequence is their order through the SCSP stages. Variables in V are either random variables or decision variables;*
- *P is a set of scoped functions whose product gives a probability distribution on the random variables, and whose scopes do not involve any decision variable (contingency assumption);*
- *C is a set of hard constraints to be satisfied.*

Thus, SCSPs extend CSPs first by adding uncontrollable random variables and then by adding a sequential aspect in the decision process. They can also be updated to integrate aspects such as additive costs, as in Stochastic Constraint Optimization Problems (SCOPs [138]). However, a restriction is that decision variables cannot have any influence on random ones. This *contingency assumption* is violated in fields like medicine, where the treatment chosen by a doctor influences the patient health state. Definition 2.21 is actually an enhanced definition of SCSPs, since in the basic version of SCSPs, the random variables are assumed to be mutually independent and $P = \{P_s \mid s \in S\}$ is a set of unary probability distributions.

Definition 2.22. *(SCSP-policy) A* SCSP-policy *is a tree involving nodes labeled with variables. The root is labeled with the first variable in V and the nodes just upon the leaves are labeled with the last variable in V. Edges in the tree are labeled with variable values. Nodes labeled with a decision variable only have one son which corresponds to the value chosen for this variable, while nodes labeled with a random variable x have one son per value in $dom(x)$.*

Each leaf can be associated with a complete assignment A of V. It is labeled with 1 if A satisfies all the constraints in C, with 0 otherwise. Moreover, it can be be associated with a probability of occurrence $p = \prod_{\varphi \in P} \varphi(A)$. The value of a SCSP-policy is the sum of the leaf values weighted by their probabilities.

A SCSP is satisfiable iff there exists a SCSP-policy whose value is greater than a given threshold θ. A SCSP-policy is optimal iff it has a maximal SCSP-policy value.

Example 2.23. *The graph coloring problem of Example 2.10 is made more complex by assuming that variable x_2 is uncontrollable and takes value red, green, blue with a probability of 0.2, 0.5, and 0.3 respectively. We further assume that first the color of x_1 must be chosen, then the color of x_2 is observed, and last a color for x_3 must be chosen. The associated SCSP is (V, P, C) where*

- *$V = [x_1, x_2, x_3]$; x_1, x_3 are decision variables, x_2 is a random variable;*
- *$P = \{P_{x_2}\}$ contains the probability distribution over x_2;*
- *$C = \{c_1, c_2, c_3\}$ is a set of three difference constraints $c_1 : x_1 \neq x_2$, $c_2 : x_2 \neq x_3$, and $c_3 : x_3 \neq x_1$.*

Figure 2.2 shows an optimal SCSP-policy. Its value of 0.8 means that it enables constraints to be satisfied with a probability of 0.8.

It is possible to show that Equation 2.8 below can be used to seek an optimal SCSP-policy. We assume that f and t are mapped onto 0 and 1 respectively, to be combinable with probabilities.

$$\max_{x_1} \sum_{x_2} \max_{x_3} \left(P_{x_2} \times \left(\prod\nolimits_{i \in [1,3]} c_i \right) \right) \tag{2.8}$$

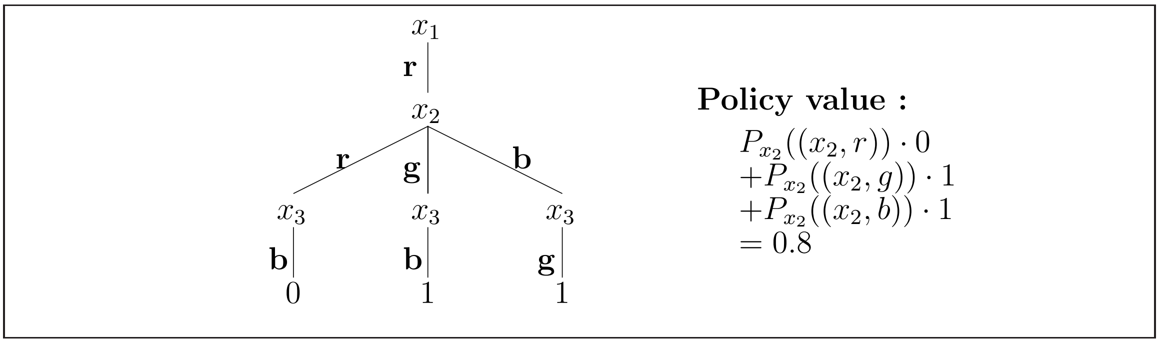

x_1
r
x_2
r
g
b
x_3
x_3
x_3
b
b
g
0
1
1
Policy value :
$P_{x_2}((x_2, r)) \cdot 0$
$+P_{x_2}((x_2, g)) \cdot 1$
$+P_{x_2}((x_2, b)) \cdot 1$
$= 0.8$

Figure 2.2: An optimal SCSP-policy for the updated graph coloring problem.

Equation 2.8 corresponds to a sequence of max *and* + *eliminations (over decision and random variables respectively) on a combination of scoped functions.*

2.3 Bayesian network-based decision frameworks

The overall approach previously described consisted in extending the expressiveness of the basic SAT and CSP frameworks by introducing plausibilities, either in the form of probabilistic uncertainties as in stochastic CSPs, or in the form of boolean pessimistic indeterminism as in QBFs. At the same time and in an opposite direction, formalisms like Bayesian Networks (BNs [96]) were developed to model uncertainties and then extended to integrate aspects such as decisions, utilities, and even constraints. We describe such extensions starting from the standard BN framework.

2.3.1 Bayesian networks

Bayesian networks (BNs [96]) enable a global joint probability distribution P_V over a set of random variables V to be represented using “local” scoped functions, the same way as CSPs enable a global constraint on all the variables to be represented using “local” constraints. Such a factored representation is useful for two reasons. First, recording a joint probability distribution when V is large can be difficult or even impossible. Second, using a factored representation of a joint distribution over V is algorithmically decisive.

Definition 2.24. *A* Bayesian network *is a triple* (V, G, P) *such that:*

- V *is a finite set of variables;*
- G *is a directed acyclic graph (DAG) over* V*;*
- $P = \{P_{x \,|\, pa_G(x)} \,|\, x \in V\}$ *is a set of conditional probability distributions of each variable* $x \in V$ *given its parents in* G*, which are multiplicative factors of the joint probability distribution* $P_V = \prod_{x \in V} P_{x \,|\, pa_G(x)}$.

This means that the joint probability distribution P_V is represented by local conditional probability distributions $P_{x \,|\, pa_G(x)}$. The main property of Bayesian networks is an equivalence theorem between factorization and conditional independence.

Definition 2.25. *Let P_V be a joint probability distribution over V and let G be a DAG over V. G is said to be compatible with P_V iff every variable $x \in V$ is conditionally independent of its non-descendants given its parents, i.e. $P_{x \,|\, nd_G(x)} = P_{x \,|\, pa_G(x)}$.*

Theorem 2.26. *[96] Let P_V be a joint probability distribution over V and let G be a DAG over V. Then, $P_V = \prod_{x \in V} P_{x \,|\, pa_G(x)}$ iff G is compatible with P_V.*

In fact, there are two major definitions for Bayesian networks. The first one, used in Definition 2.24, introduces BNs starting from the *factorization into conditional distributions*. The second one, which starts instead from *conditional independence*, is "*Let P_V be a joint probability distribution over V and let G be a DAG over V. The pair (G, P_V) is a Bayesian network iff G is compatible with P_V*". The two definitions are equivalent thanks to Theorem 2.26. The choice of one of the two definitions is a matter of perspective and both points of view are used.

One possible query on a BN is to compute the marginal probability distribution of a variable $y \in V$:

$$P_y \quad = \quad \sum_{V-\{y\}} P_V = \sum_{V-\{y\}} \big(\prod_{x \in V} P_{x \,|\, pa_G(x)}\big) \tag{2.9}$$

Equation 2.9 corresponds to sum-eliminations on a product of scoped functions. In other queries on BNs such as MAP (Maximum A Posteriori hypothesis), used to seek an optimal explanation to some observations, max-eliminations are also performed, in elimination sequences such as $\max_D \sum_{V-D} (\prod_{x \in V} P_{x \,|\, pa_G(x)})$.

Example 2.27. *[95] Mr Holmes has equipped his house with an alarm which can ring if a burglary or if an earthquake occurs. If it sounds, then his two neighbors John and Mary are likely to call him.*

This problem can be modeled using 5 boolean random variables: bu, representing the occurrence of a burglary, eq, representing the occurrence of an earthquake, al, modeling whether the alarm sounds, mc, specifying whether Mary calls, and jc, modeling whether John calls.

The DAG represented on Figure 2.3 can then be used to model conditional independences qualitatively. It says that each variable is conditionally independent of its non descendants given its parents. For example, mc is conditionally independent of eq, bu, jc given al. This means that as soon as one knows whether the alarm sounds, mc does not depend on the other variables. Similarly, jc is conditionally independent of eq, bu, mc given al. Moreover, eq is conditionally independent of bu given no other information. However, as soon as the value of the descendant al is known, eq and bu become correlated.

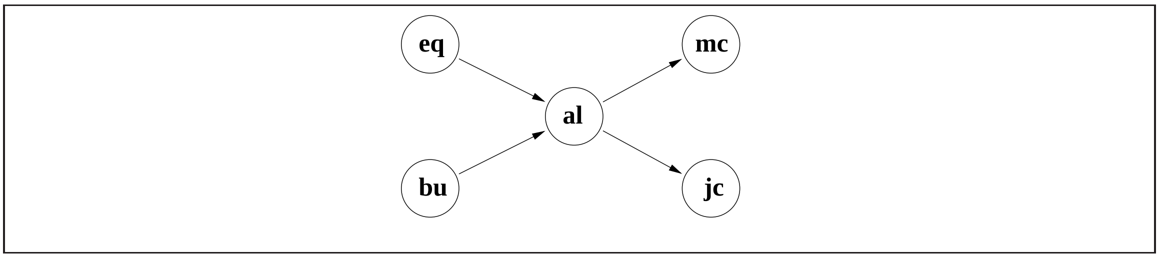


Figure 2.3: DAG of the Bayesian network of Mr Holmes' alarm problem.

Besides the qualitative information expressed by the DAG, BNs also specify conditional probability distributions of each variable given its parents, such as $P_{al\,|\,eq,bu}$*, the conditional probability that the alarm sounds or not given the occurrence of an earthquake and a burglary.*

The joint probability distribution then factors as $P_{eq,bu,al,jc,mc} = P_{eq} \cdot P_{bu} \cdot P_{al\,|\,eq,bu} \cdot P_{jc\,|\,al} \cdot P_{mc\,|\,al}$*. In order to make a diagnosis and get the probability that the alarm sounds or not, one must compute* $P_{al} = \sum_{eq,bu,jc,mc} \big(P_{eq} \cdot P_{bu} \cdot P_{al\,|\,eq,bu} \cdot P_{jc\,|\,al} \cdot P_{mc\,|\,al}\big)$*.*

Similarly, queries on so-called Dynamic Bayesian Networks (DBNs [31]), which extend BNs by integrating a temporal aspect, can be reduced to the computation of eliminations on a product of conditional probability distributions.

2.3.2 Possibilistic networks

BNs use probabilities to model uncertainties. Possibilistic networks [51, 69] extend BNs to a possibilistic representation of uncertainty, and enable a global joint possibility distribution to be represented by local conditional possibility distributions.

Definition 2.28. *A* possibilistic network *is a triple* (V, G, P) *such that:*

- V *is a finite set of variables;*
- G *is a directed acyclic graph (DAG) over* V*;*
- $P = \{\pi_{x\,|\,pa_G(x)} \,|\, x \in V\}$ *is a set of conditional possibility distributions of each variable* $x \in V$ *given its parents in* G*, which are factors of the joint possibility distribution* $\pi_V = \min_{x \in V} \pi_{x\,|\,pa_G(x)}$*.*[5]

In order to get the marginal possibility distribution of a variable $y \in V$, one must compute

$$\pi_y = \max_{V-\{y\}} \pi_V = \max_{V-\{y\}} \big(\min_{x \in V} \pi_{x\,|\,pa_G(x)}\big) \tag{2.10}$$

The latter equation is a max-elimination on a min-combination of scoped functions.

2.3.3 Mixed networks

BNs were extended to use CSP techniques such as constraint propagation. This extension is called *mixed networks* [36, 37]. In this formalism, constraints are introduced over the random variables of a BN, in order to model:

- either the *deterministic part* extracted from conditional probability distributions (0-1 probabilities): for example, if x, y, and z are boolean variables such that $P_{z\,|\,x,y}(A) = 0$ whenever A contains $(x, t).(z, t)$, one can extract the constraint $c : \neg(x \wedge z)$ as a redundant but algorithmically important information;
- or *evidences* (i.e. observations). They correspond either to the assignment of a single variable, or to more complex evidences expressed e.g. as boolean formulas. For instance, if one hears

[5] Actually, the joint possibility distribution can take other forms depending on the operator used to define possibilistic conditioning. Typically, the joint possibility distribution represented by a possibilistic network can also be $\pi_V = \prod_{x \in V} \pi_{x\,|\,pa_G(x)}$.

a sound in a room containing two sources s_1 and s_2, then the complex evidence $s_1 \vee s_2$ can be inferred ($s = t$ if a source s has produced a noise).

Definition 2.29. *A* mixed network *is a tuple* (V, G, P, C) *where:*

- V *is a finite set of variables;*
- G *is a DAG over* V*;*
- $P = \{P_{x \,|\, pa_G(x)} \,|\, x \in V\}$ *is a set of conditional probability distributions. When* $P_{x \,|\, pa_G(x)}$ *is a conditional probability distribution taking values* 0 *or* 1 *only, it is called a deterministic conditional distribution. It can then be represented as a deterministic function* $dom(pa_G(x)) \rightarrow dom(x)$ *or as a constraint;*
- $C = \{c_1, \ldots, c_k\}$ *is a finite set of constraints whose scopes are included in* V*.*

A query on a mixed network can be for instance to determine the probability p_c that constraints in C are satisfied. Such a query can be answered by computing the sum of the probabilities of the complete assignments satisfying all the constraints, i.e.

$$\sum_{A \in dom(V), c_1(A) \wedge \ldots \wedge c_k(A) = t} \left(\prod_{x \in V} P_{x \,|\, pa_G(x)}(A)\right) \quad = \quad \sum_{V} \left(\left(\prod_{x \in V} P_{x \,|\, pa_G(x)}\right) \times \left(\prod_{i \in [1,k]} c_i\right)\right) \qquad (2.11)$$

Equation 2.11 combines local probabilities using ×, combines constraints using ×, combines probabilities with constraints using ×, and eliminates variables using $\sum$.

When the CSP (V, C) is consistent, a mixed network actually represents the joint "mixed" probability distribution $\mathcal{MP}_V$ such that for all complete assignments A of V, $\mathcal{MP}_V(A)$ is the probability of occurrence of assignment A given that the constraints in C are satisfied. More formally,

$$\mathcal{MP}_V(A) = \begin{cases} \frac{1}{p_c} \cdot P_V(A) \text{ if } c_1(A) \wedge \ldots \wedge c_k(A) = t \\ 0 \text{ otherwise} \end{cases} \qquad (2.12)$$

2.3.4 Influence diagrams

In another direction, BNs only define probabilistic relations between random variables. They allow to model diagnosis problems. Influence diagrams (IDs [64]) extend BNs by adding the notions of decision and additive utility.

Definition 2.30. *An* influence diagram *is a composite graphical model defined on three sets of variables organized in a DAG G:*

- *a set* S *of* chance variables*, represented by circles. For each* $x \in S$*, a conditional probability distribution* $P_{x \,|\, pa_G(x)}$ *on* x *given its parents in* G *is specified (as in a BN);*
- *a set* D *of* decision variables*, represented by squares. For each* $x \in D$*,* $pa_G(x)$ *is the set of variables observed before decision* x *is made. Hence, arcs pointing to decision variables are information arcs, since they define available information when the decision is made.*

  There must exist a directed path $d_1 \rightarrow d_2 \rightarrow \ldots \rightarrow d_q$ *containing all decision variables, so that the order in which decisions are made is completely determined (*regularity *assumption).*

*Moreover, even if this is not represented in the DAG, the parents of a decision variable must be parents of all subsequent decision variables (*no-forgetting *assumption).*[6]

- *a set Γ of* utility variables*, represented by diamonds. For each $u \in \Gamma$, an additive utility function $U_{pa_G(u)}$ of scope $pa_G(u)$ is specified. Utility variables must be leaves in the DAG.*

Similarly to a stochastic CSP, the problem associated with an influence diagram is to search for an optimal ID-policy, as defined below.

Definition 2.31. *(ID-policy) An* ID-policy *is a set of* decision rules $\delta_x : dom(pa_G(x) \cap S) \to dom(x)$*, one per decision variable $x \in D$.*[7]

For every complete assignment A of the chance variables, an ID-policy Δ defines a unique complete assignment $\Delta(A) = A.\delta_{d_1}(A). \cdots .\delta_{d_q}(A)$, such that

- *the probability that A occurs is $P_\Delta(A) = \prod_{x \in S} P_{x \mid pa_G(x)}(\Delta(A))$,*
- *the utility associated with A is $U_\Delta(A) = \sum_{u \in \Gamma} U_{pa_G(u)}(\Delta(A))$.*

The value of an ID-policy Δ is $val_\Delta = \sum_{A \in dom(S)} (P_\Delta(A) \cdot U_\Delta(A))$. It corresponds to the probabilistic expected utility of Δ.

An optimal ID-policy Δ^* is an ID-policy of maximal value.

The above definition can be related to sequences of eliminations. If one denotes by I_0 the set of chance variables observed before the first decision d_1, by I_k the set of chance variables observed between decisions d_k and d_{k+1}, and by I_q the set of chance variables unobserved before the last decision d_q, then computing an optimal ID-policy is equivalent [66] to computing optimal decision rules for the quantity

$$\sum_{I_0} \max_{d_1} \ldots \sum_{I_{q-1}} \max_{d_q} \sum_{I_q} ((\prod_{x \in S} P_{x \mid pa_G(x)}) \times (\sum_{u \in \Gamma} U_{pa_G(u)})) \tag{2.13}$$

Again, Equation 2.13 is a sequence of eliminations (alternating eliminations using max and +) on a combination of scoped functions (probabilities combined using ×, utilities combined using +, probabilities and utilities combined using ×).

Example 2.32. *Mr Holmes does not just want to perform diagnosis tasks to know the probability that he is burglarized. He also wants to plan actions in order to maximize an expected utility:*

- *Mr Holmes can decide to call a neighbor in order to know whether the alarm is ringing: we remove variables mc and jc, and we add a boolean decision variable ca modeling whether Mr Holmes calls a neighbor. However, a phone call makes him lose a* 1000€ *contract which he is negotiating. This is represented by a utility variable u_1 with ca as parent.*
- *If Mr Holmes calls, he gets, with a certain probability, a result re equal to na (no answer, if his neighbor does not answer), t (if the neighbor tells him that the alarm is ringing), or f (if the neighbor tells him that the alarm is not ringing). If Mr Holmes does not call, he gets $re = na$ (no answer).*

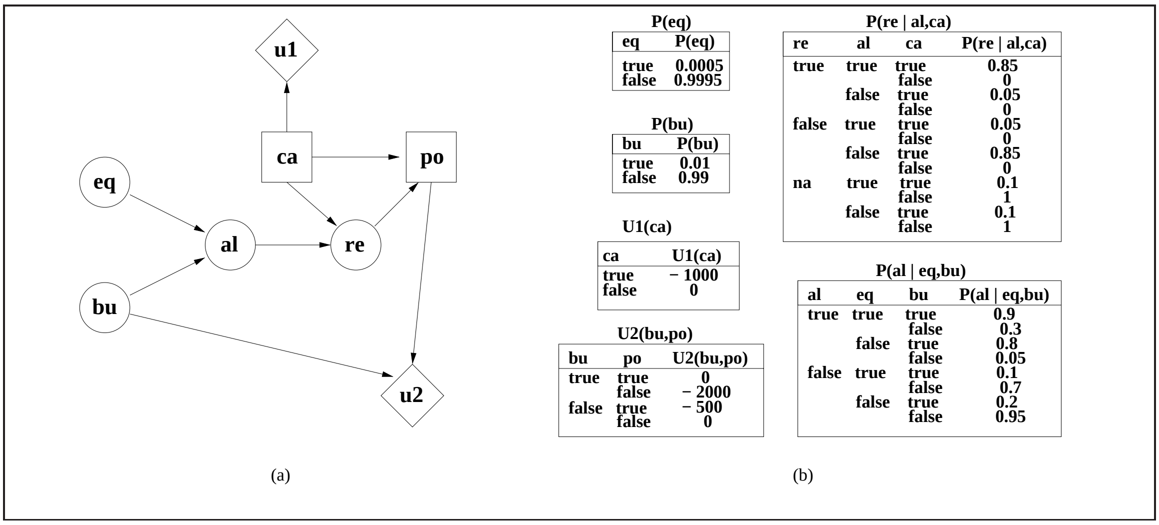
[6] Extensions of IDs exist which relax the no-forgetting or the regularity assumptions, such as decision networks [144]. In some extensions, arcs pointing into a decision variable x can also model that some values in $dom(x)$ are forbidden for some assignments of $pa_G(x)$. This allows so-called *asymmetric* decision problems to be modeled.

[7] We do not make any assumption on the way this set of decision rules is recorded. We only assume that for each $x \in D$, it implicitly or explicitly specifies a decision to make depending on the assignment of $pa_G(x) \cap S$.

- *Depending on the call and the answer, Mr Holmes can decide to call the police. This is modeled by a boolean decision variable po. If he calls the police and his house is not being burglarized, he will pay a* 500€ *penalty. If he does not call and his house is being burglarized, he loses* 2000€*. This is modeled using a utility function u_2 with $\{bu, po\}$ as scope.*

The associated influence diagram is shown in Figure 2.4. The unique optimal ID-policy consists in calling neither a neighbor, nor the police. Its value is −20€*. It can be obtained by directly applying Definition 2.31 or by computing a sequence of variable eliminations on the combination of the scoped functions defined by the ID, as in Equation 2.14.*

$$\max_{ca} \sum_{re} \max_{po} \sum_{bu,eq,al} \left((P_{bu} \cdot P_{eq} \cdot P_{al \,|\, bu,eq} \cdot P_{re \,|\, al,ca}) \times (U_{ca} + U_{bu,po}) \right) \tag{2.14}$$



P(eq)

eq	P(eq)
true	0.0005
false	0.9995

P(bu)

bu	P(bu)
true	0.01
false	0.99

U1(ca)

ca	U1(ca)
true	− 1000
false	0

U2(bu,po)

bu	po	U2(bu,po)
true	true	0
	false	− 2000
false	true	− 500
	false	0

P(re | al,ca)

re	al	ca	P(re | al,ca)
true	true	true	0.85
		false	0
	false	true	0.05
		false	0
false	true	true	0.05
		false	0
	false	true	0.85
		false	0
na	true	true	0.1
		false	1
	false	true	0.1
		false	1

P(al | eq,bu)

al	eq	bu	P(al | eq,bu)
true	true	true	0.9
		false	0.3
	false	true	0.8
		false	0.05
false	true	true	0.1
		false	0.7
	false	true	0.2
		false	0.95

Figure 2.4: An influence diagram: (a) Qualitative part; (b) Quantitative part.

Compared to the most advanced SAT and CSP-based formalisms, influence diagrams can capture uncertainties without assuming any contingency. Decisions can therefore influence the state of the environment (e.g. decision *ca* influences random variable *re*). This is mainly due to the fact that modeling uncertainties is one of the bases of influence diagrams and not just an added component. Nevertheless, as far as we know, influence diagrams are algorithmically less developed on some points, since e.g. they do not use any soft constraint propagation mechanisms.

2.4 Beyond conditional probabilities for modeling uncertainties

All the previous BN-based formalisms represent uncertainties using local conditional distributions. In another direction, some formalisms emphasize factorization and do not require to handle only conditional distributions. We briefly present some of them, which can be seen as alternatives to Bayesian networks and influence diagrams.

2.4.1 Markov random fields and chain graphs

In order to explain why BN is not always the best formalism to model uncertainties, and why factorization-based models can be more efficient, we use a statistical physics example.

Example 2.33. *A spin glass is a disordered magnetic material, for instance a material made of copper (Cu) and containing some atoms of manganese (Mn) distributed on some* sites*, as in Figure 2.5. The magnetic state of the manganese atoms can be described by a random variable taking value* $+1$ *or* -1*. Some atoms of manganese want to have the same magnetic state ("friend" atoms), while others want to have opposite magnetic states ("antagonist" atoms). Last, some atoms do not directly interact, because they are too far from each other.*

The interactions between the manganese atoms are such that no state exists where all atoms magnetic preferences are satisfied. For example, let us consider three atoms placed on sites s_1*,* s_2*,* s_3*. If* s_1 *wants to have the same magnetic state as both* s_2 *and* s_3*, whereas* s_2 *and* s_3 *want to have distinct magnetic states, there is no perfect situation.*

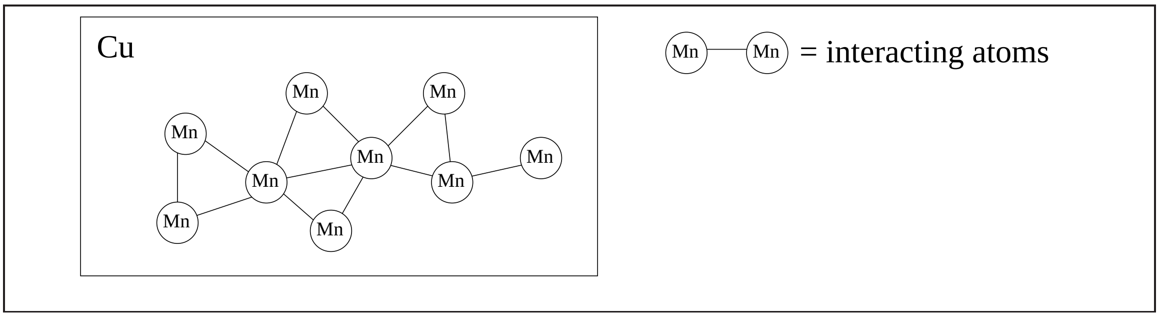


Figure 2.5: A copper (Cu) / manganese (Mn) spin glass.

Let us assume that there are n *sites and that the magnetic state* (± 1) *of site* i *is given by variable* s_i*. Let* J_{ij} *be a parameter equal to* 1 *if the atoms in* s_i *and* s_j *are friend atoms,* -1 *if they are antagonist atoms, and* 0 *if they do not interact. Then, in order to describe the global state of the copper-manganese alloy, statistical physicians write the joint probability distribution over* $\{s_1, \ldots, s_n\}$ *as* $P_{s_1,\ldots,s_n} = \frac{1}{Z} exp(-\beta \cdot E_{s_1,\ldots,s_n})$*, where* Z *is a normalizing constant,* β *is a constant, and* $E_{s_1,\ldots,s_n}$ *is the* energy function *equal to* $E_{s_1,\ldots,s_n} = -\sum_{(i,j)} J_{ij} s_i s_j$*. This enables us to write*

$$P_{s_1,\ldots,s_n} = \frac{1}{Z} \times \prod_{(i,j)} exp(-\beta J_{ij} s_i s_j) \tag{2.15}$$

Equation 2.15 expresses a joint probability distribution as a combination of factors which are not conditional probability distributions. Expressing this joint distribution with Bayesian networks is not only unnatural, but also less efficient, because BNs could involve scoped functions whose largest scope can be linear in n! This is due to the difference between conditional independences expressible in a directed graph and in an undirected one.

Markov Random Fields (MRFs [22]) is a formalism which enables probability distributions such as the one in Equation 2.15 to be modeled. We only present discrete state MRFs.

Definition 2.34. *Let* $S = \{s_1, \ldots, s_n\}$ *be a finite set of finite domain random variables organized* *in an undirected graph G. Each variable $x \in S$ has a set of neighbors $N_G(x)$ given by G. Let P_S denote a probability distribution over S.*

(G, P_S) is a Markov Random Field iff for every variable $x \in S$, $P_{x \mid S-\{x\}} = P_{x \mid N_G(x)}$, i.e. each variable is probabilistically independent of its non-neighbors given its neighbors in G.

The Hammersley-Clifford theorem [63] establishes that (G, P_S) is a MRF iff P_S can be factored as a Gibbs distribution:

$$P_{s_1,\ldots,s_n} = \frac{1}{Z} \times \prod_{cl \in \mathcal{C}l} exp(-\beta \cdot \varphi_{cl}) \tag{2.16}$$

where Z is a normalization constant, $\mathcal{C}l$ is the set of cliques of G, and φ_{cl} is a scoped function of scope cl called the potential of clique cl. This shows that MRFs can be used to model problems like spin glasses. This formalism is also used in vision and neuronal biology. Roughly speaking, its "philosophy" is that BNs are more often used to model temporal (causal) relations between random variables, whereas MRFs are more appropriate to model spatial correlations.

Given a MRF, one can compute a most probable configuration by performing max-eliminations on a multiplication of scoped functions, as follows:

$$\max_{s_1,\ldots,s_n} \left(\frac{1}{Z} \times \prod_{cl \in \mathcal{C}l} exp(-\beta \cdot \varphi_{cl})\right) \tag{2.17}$$

BNs and MRFs are unified by *Chain graphs* [55]. A chain graph uses a graph containing both directed and undirected arcs, and such that cycles in this graph involve undirected links only. The set $\mathcal{C}$ of connected components obtained when removing directed arcs are called the *components of the chain graph*. A chain graph can then be seen as a DAG G whose vertices are the components in $\mathcal{C}$.

It represents a joint probability distribution P_V in a factored form $P_V = \prod_{c \in \mathcal{C}} P_{c \mid pa_G(c)}$, each conditional distribution $P_{c \mid pa_G(c)}$ being itself specified as in Markov random fields by a set of scoped functions Φ_c and by a normalization constant $Z_{pa_G(c)}$ whose scope is included in $pa_G(c)$, so that $P_{c \mid pa_G(c)} = \frac{1}{Z_{pa_G(c)}} \prod_{\varphi \in \Phi_c} \varphi$.

2.4.2 Valuation networks

The statement made for Bayesian networks also holds for influence diagrams. Basically, IDs use conditional probability distributions to model uncertainties. They were extended so as to integrate models like MRFs. The corresponding extension is called Valuation Networks (VNs [128]) and is also known as valuation-based systems for Bayesian decision. VNs emphasize the multiplicative decomposition of a joint probability distribution, and not conditional independence.

Definition 2.35. *A* Valuation Network *is a tuple $(V, P, U, \prec)$ where:*

- *V is a finite set of variables, partitioned between decision and environment variables;*
- *$P = \{P_1, \ldots, P_s\}$ is a set of scoped functions[8] whose multiplication gives a family of joint probability distributions on the environment variables (one joint distribution per possible assignment of the decision variables);*

[8] In the valuation network terminology, scoped functions are called valuations.

- *$U = \{U_1, \ldots, U_t\}$ is a set of scoped functions which are additive factors of a global utility;*
- *$\prec$ defines precedence constraints, indicating which observations are available when a decision is made. Some consistency conditions are imposed on these precedence constraints (we omit them for simplicity [128]).*

The usual query on a VN is the same as for influence diagrams. It can be answered to by computing a sequence of eliminations like

$$\sum_{I_0} \max_{d_1} \ldots \sum_{I_{q-1}} \max_{d_q} \sum_{I_q} ((\prod_{P_i \in P} P_i) \times (\sum_{U_i \in U} U_i)) \tag{2.18}$$

in which the P_i functions are not necessarily conditional probability distributions.

The VN formalism was also extended in order to handle asymmetric decision problems in a better way, as in sequential valuation networks [41]. Informally, a decision problem is asymmetric if some variable assignments are impossible given the assignment of other variables, leading to an asymmetric tree in a decision tree representation. In such extensions, e.g. with asymmetric valuation networks [130], sequential decision problems with probabilistic uncertainties, feasibilities, and additive utilities can be modeled. The difference with usual VNs is that a set $F = \{F_1, \ldots, F_r\}$ of boolean scoped functions, called indicator valuations, is added. These indicator valuations are local feasibility constraints. They specify that the assignments of some variables are unfeasible given the assignment of other variables.

If the precedence constraints look like $d_1 \prec d_2 \prec s_1 \prec d_3 \prec d_4 \prec s_2$, it can be shown that optimal decision rules for d_1, d_2, d_3, d_4 are defined via Equation 2.19:

$$\max_{d_1,d_2} \sum_{s_1} \max_{d_3,d_4} \sum_{s_2} \left(\left(\bigwedge_{F_i \in F} F_i \right) \star \left(\prod_{P_i \in P} P_i \right) \times \left(\sum_{U_i \in U} U_i \right) \right) \tag{2.19}$$

Local feasibility constraints are combined using $\wedge$, and combined with other scoped functions using the truncation operator $\star$ (cf Definition 1.6). And, again, a sequence of eliminations is performed.

2.5 Classical planning-based frameworks

The previous sections have offered a quick overview of some existing variable-based representation frameworks for sequential decision making with uncertainties, feasibilities, and utilities. In another direction, classical planning problems can use different representations [58]. We describe only the most popular one, the *classical planning representation*, which is namely linked with the planning system STRIPS [49] and the famous PDDL planning language [86]. This representation is of interest because it uses a knowledge representation which differs from the variable-based modeling seen so far with SAT, CSPs, and BNs.

2.5.1 Classical planning

The presentation of classical planning requires some definitions concerning first order languages.

Definition 2.36. *A first order language $\mathcal{L}$ is based on four types of symbols: constant, variable, predicate, and function symbols. The following definitions hold when there are no function symbols.*

A term *is either a constant symbol or a variable symbol.*

If pr is an n-place predicate and $t_1, \ldots, t_n$ are terms, then $pr(t_1, \ldots, t_n)$ is called an atom.

A literal *is an atom or its negation. A* positive literal *is an atom and a* negative literal *is the negation of an atom. Given a set of literals L, we denote by L^+ and L^- the set of positive and negative literals in L respectively.*

An atom (resp. a literal) is said to be a ground *atom (resp. a* ground *literal) if it involves only constant symbols.*

For example, in a first-order language $\mathcal{L}$ without functions where the constant symbols are $b1$, $b2$, $b3$, where variable symbols are x and y, and where there is a two-place predicate pr, the terms are $b1$, $b2$, $b3$, x, and y, $pr(b1, x)$, $pr(b2, b3)$, and $pr(x, y)$ are examples of atoms, among which only $pr(b2, b3)$ is a ground atom, and $pr(b1, x)$ and $\neg pr(b2, b3)$ are literals, among which only $\neg pr(b2, b3)$ is a ground literal.

Definition 2.37. *Let $\mathcal{L}$ be a first-order language with a finite number of symbols and without function symbols. A* planning operator *in $\mathcal{L}$ is a triple $o = (name(o), precond(o), effects(o))$ such that:*

- *$name(o)$ is the operator name, looking like $n(x_1, \ldots, x_k)$ (n is called the operator symbol and $x_1, \ldots, x_k$ are the variables appearing in the definition of o);*
- *$precond(o)$ and $effects(o)$ are sets of literals in $\mathcal{L}$ defining the preconditions and effects of o respectively.*

If $\mathcal{O}$ is a set of planning operators in $\mathcal{L}$, then $(\mathcal{L}, \mathcal{O})$ defines a classical planning domain.

The definition of a planning domain is purely syntactic. Semantically speaking, a planning domain defines a so-called “restricted state-transition system” [58] $\Sigma = (\mathcal{S}, \mathcal{A}, \gamma)$ where:

- $\mathcal{S} \subseteq 2^{\{\text{all ground atoms of } \mathcal{L}\}}$ is a finite set of states. We make the *closed-world assumption*, i.e. an atom which is not explicitly specified in a state does not hold in that state;
- $\mathcal{A} = \{\text{all ground instances of operators in } \mathcal{O}\}$ is a finite set of actions. A ground instance of a planning operator o is simply a planning operator obtained from o by replacing variable symbols by constant symbols;
- $\gamma : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a state-transition function. Given $(s, a) \in \mathcal{S} \times \mathcal{A}$, if $precond^+(a) \subseteq s$ and $precond^-(a) \cap s = \emptyset$, then a is applicable to s. In this case, $\gamma(s, a) = (s - effects^-(a)) \cup effects^+(a)$ is a state in $\mathcal{S}$ obtained if action a is performed in state s: a deletes the negative effects and add the positive ones. Otherwise, if action a is not applicable in state s, $\gamma(s, a)$ is undefined. In other words, planning operators explicitly say that preconditions must be satisfied for a decision to be feasible, and they define deterministic effects of actions.

Definition 2.38. *The* statement of a planning problem *is a triple $(\mathcal{P}, s_0, g)$ where $\mathcal{P} = (\mathcal{L}, \mathcal{O})$ is a planning domain, s_0 is a set of ground atoms in $\mathcal{L}$ defining the initial state, and g is a set of ground literals in $\mathcal{L}$ representing the goal.*

The set of goal states S_g is the set of all states $s \in \mathcal{S}$ such that every positive literal in g is in s and no negative literal in g is in s.

Definition 2.39. *Let $(\mathcal{P}, s_0, g)$ be the statement of a planning problem. A* plan *is a sequence of actions $[a_1, \ldots, a_k]$. A plan is* applicable *iff $\gamma(\gamma(\ldots\gamma(\gamma(\gamma(s_0, a_1), a_2), a_3)\ldots, a_{k-1}), a_k)$ is not undefined, where γ is the transition function of the restricted state-transition system associated with $\mathcal{P}$. A plan is a* solution *iff $\gamma(\gamma(\ldots\gamma(\gamma(\gamma(s_0, a_1), a_2), a_3)\ldots, a_{k-1}), a_k) \in S_g$*

The planning problem consists in finding a plan which is a solution.

Example 2.40. *The "Blocks World" problem described below illustrates planning operators. Initially, a stack of numbered blocks lies on a table, as in Figure 2.6(a). A robot arm can unstack the highest block of a pile or pick up a block from the table. A block held by the arm can be put down on the table or stacked on the top of a pile of blocks. The arm cannot hold more than one block.*

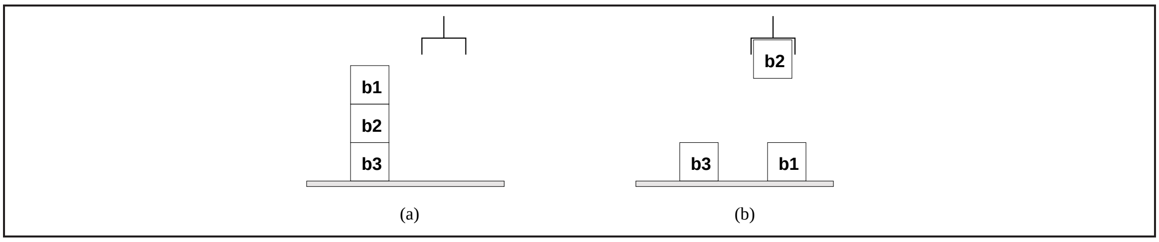


Figure 2.6: A blocks world problem: (a) initial state; (b) state reached after applying the plan [unstack($b1, b2$), put-down($b1$), unstack($b2, b3$)].

In order to model this problem, we first define the planning domain*:*

- Language $\mathcal{L}$*:*
  - *Predicate symbols: $clear(x)$, $ontable(x)$, $on(x, y)$, $emptyarm$, $holding(x)$;*
  - *Constant symbols: $b1$, $b2$, $b3$;*
- Planning operators:

o_1 : *stack*(x, y)	
precond(o_1) :	$\{holding(x), clear(y)\}$
effects(o_1) :	$\{\neg holding(x), \neg clear(y), clear(x), emptyarm, on(x, y)\}$
o_2 : *unstack*(x, y)	
precond(o_2) :	$\{emptyarm, on(x, y), clear(x)\}$
effects(o_2) :	$\{\neg emptyarm, \neg on(x, y), \neg clear(x), clear(y), holding(x)\}$
o_3 : *pick-up*(x)	
precond(o_3) :	$\{clear(x), ontable(x), emptyarm\}$
effects(o_3) :	$\{\neg clear(x), \neg ontable(x), \neg emptyarm, holding(x)\}$
o_4 : *put-down*(x)	
precond(o_4) :	$\{holding(x)\}$
effects(o_4) :	$\{\neg holding(x), clear(x), emptyarm, ontable(x)\}$

An action is an instantiation of a planning operator. For example, $stack(b1, b2)$ is an action which puts block $b1$ on block $b2$ if the preconditions $holding(b1)$ and $clear(b2)$ both hold.

A planning problem statement can then be defined on the previous planning domain, e.g. by

- *the initial state $s_0 = \{ontable(b3), on(b2, b3), on(b1, b2), clear(b1), emptyarm\}$ represented in Figure 2.6(a);*

- *the goal* $g = \{clear(b3), \neg emptyarm\}$*, which says a state is a goal state iff there is no block on top of* $b3$ *and the robot arm holds a block.*

$[unstack(b2, b3)]$ *and* $[pick\text{-}up(b2), put\text{-}down(b3)]$ *are examples of plans which are not applicable, because they violate some preconditions.* $[unstack(b1, b2), put\text{-}down(b1), pick\text{-}up(b1), stack(b1, b2)]$ *is an applicable plan. It says that the robot must take* $b1$ *on top of the initial stack, put it on the table, pick it up from the table, and put it again on the blocks stack. An example of a plan which is a solution to the planning problem is the sequence of actions* $[unstack(b1, b2), put\text{-}down(b1), unstack(b2, b3)]$. *The state obtained when performing this plan, which is a goal state, is shown in Figure 2.6(b).*

The classical planning framework offers extensions in which preconditions and action effects can be more general than just sets of literals or atoms, and in which goals can be more general than just states to reach (e.g., the number of actions of a plan can be a plan ranking parameter).

As previously stated, the classical planning framework uses a knowledge representation which is different from the variable-based one used in SAT, CSP, or BN. Nevertheless, the classical planning representation is equivalent to another variable-based representation [58]. A classical planning problem can indeed be formulated as a CSP in order to search for a solution plan with a length $\leq$ k. This shows that a classical planning problem can be formulated as a sequence of max-eliminations on a conjunction of scoped functions.

In all the following, we will use a variable-based representation. This choice is motivated both in terms of models and algorithms:

- From a modeling point of view, many formalisms reason about variables and local functions. In order to build a generic encompassing framework, it is more natural (and easier) to reuse this common basis.

- From an algorithmic point of view, frameworks like CSPs or BNs already offer various techniques which are strongly related with the variable-based representation. In order to generalize them, working on a similar representation can be helpful.

2.5.2 Conformant planning and probabilistic planning

The classical planning framework was extended in order to model either pessimistic indeterminisms, as in conformant planning [60], or stochastic indeterminisms, as in probabilistic planning [77]. The two main ideas are first that there can be uncertainties on the initial state of the environment, and second that action effects may be non-deterministic.

In conformant planning, the initial state s_0 is replaced by a set of possible initial states S_0. S_0 can be defined either explicitly, or implicitly via boolean formulas or constraints. Planning operators become non-deterministic, meaning that they describe all the possible states which can be reached when they are applied. The objective is to find an unconditional plan (the environment is assumed to be unobservable) which guarantees that the goal is reached, whatever the evolution of the environment is.

In conformant probabilistic planning, a probability distribution over the initial state is specified, and actions have probabilistic effects. The objective is then to search for an unconditional plan maximizing the probability that a goal state is reached.

In probabilistic planning, the state of the environment becomes observable. Hence one can seek conditional plans. Probabilistic planning problems can be expressed using the PPDDL planning language [143].

2.6 Sequential decision making under uncertainty with MDPs

Frameworks like MDPs also use a state-based modeling and describe the evolution of the whole state of the environment. This section introduces MDPs and their extensions to non-probabilistic uncertainties and partial observabilities. It also shows that despite the basic state-based representation, variable eliminations can still be used.

2.6.1 Markov decision processes

Markov Decision Processes (MDPs [111, 89]) model sequential decision problems such that, at each step t of the decision process, an agent must make a decision d depending on the state s of the environment at t. This decision d induces an immediate reward $U(s, d)$ and a stochastic evolution of the whole state of the environment, which becomes s' with a certain probability $P(s' \mid s, d)$. This reward $U(s, d)$ and this evolution $P(s' \mid s, d)$ are assumed to depend only on s and d (Markov hypothesis). Figure 2.7 describes the unrolled form of a 4-step MDP.

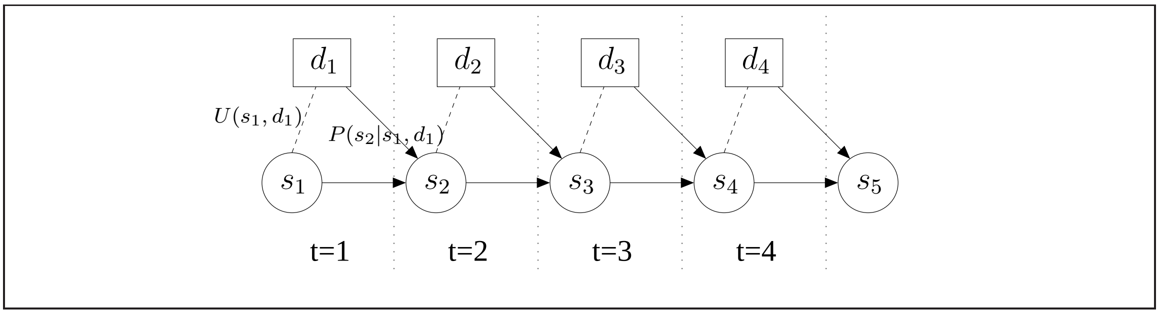


Figure 2.7: A 4-step MDP. A vertex s_i represents the state at step i and a vertex d_i represents the decision made at step i. An undirected dotted edge between s_i and d_i represents the reward induced when decision d_i is made in state s_i, and arcs into vertex s_{i+1}, coming from s_i and d_i, point out that the state at step $i + 1$ depends on the state and decision at step i (via the transition function $P(s' \mid s, d)$).

Definition 2.41. *A MDP is a tuple* $(\mathcal{S}, \mathcal{D}, P(. \mid ., .), U(., .))$ *where*

- $\mathcal{S}$ *is a finite set of states of the environment;*
- $\mathcal{D}$ *is a finite set of decisions;*
- $P(. \mid ., .) : \mathcal{S} \times \mathcal{S} \times \mathcal{D} \to [0, 1]$ *is a function such that* $P(s' \mid s, d)$ *is the conditional probability of reaching state* s' *if decision* d *is made in state* s *(transition model);*
- $U(., .) : \mathcal{S} \times \mathcal{D} \to \mathbb{R}$ *is a function such that* $U(s, d)$ *is the immediate reward obtained if decision* d *is made in state* s *(reward model).*[9]

[9] The literature offers various definitions of MDPs. In Definition 2.41, we consider only stationary MDPs, that is the actions available, the transition model, and the immediate rewards do not depend on the step t considered.

MDPs do not basically involve variables: they reason about a state space $\mathcal{S}$ and a decision space $\mathcal{D}$. However, the state at one step can be described by one state variable $\mathbf{s}$ with $\mathcal{S}$ as domain, and the decision made at one step can be seen as one decision variable $\mathbf{d}$ with $\mathcal{D}$ as domain.

A MDP can be either a finite, or an infinite horizon MDP. In the former, there exists a step T such that what occurs after T is not considered.[10] In the latter, a discount factor γ ($0 \leq \gamma < 1$) is introduced to model that the sooner the rewards the better. This discount factor can be used with finite horizon MDPs as well, in which case it does not need to be strictly lesser than 1. Given the transition model and the reward model, the goal is to search for a sequence of decisions of maximal expected utility.

Definition 2.42. *(MDP-policy) A MDP-decision rule is a function $\delta : \mathcal{S} \to \mathcal{D}$ specifying a decision $\delta(s) \in \mathcal{D}$ to make depending on the current state $s \in \mathcal{S}$. A MDP-policy Δ is a set of MDP-decision rules $\Delta = \{\delta_1, \delta_2, \delta_3, \ldots\}$ (δ_t is the MDP-decision rule associated with the t^{th}-to-last step). If $\delta_1 = \delta_2 = \delta_3 = \ldots = \delta$, then the MDP-policy is said to be* stationary *and is specified simply as $\Delta = \{\delta\}$.*

The value of a MDP-policy Δ is its associated expected utility, defined inductively as follows. Let $val_{\Delta,t}(s)$ be the expected utility if Δ is applied during t steps starting from state s. For $t = 1$, the expected utility is simply the immediate reward obtained when making decision $\delta_1(s)$, i.e. $val_{\Delta,1}(s) = U(s, \delta_1(s))$. For $t > 1$, the expected utility obtained when applying Δ during t steps is the sum of the immediate reward obtained when making decision $\delta_t(s)$ in state s and of the expected utility obtained when applying policy Δ during the $t - 1$ remaining steps. In other words,

$$val_{\Delta,t}(s) = U(s, \delta_t(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s' \mid s, \delta_t(s)) \cdot val_{\Delta,t-1}(s')$$

For a MDP with a finite horizon T, the value val_Δ of a MDP-policy $\Delta = \{\delta_1, \ldots, \delta_T\}$ is given by $val_\Delta = val_{\Delta,T}$. With an infinite horizon, the value of Δ is $\lim_{t\to\infty} val_{\Delta,t}$.

An optimal MDP-policy Δ^* is a MDP-policy of highest value. Standard results on MDPs show that when the horizon is infinite, there always exists an optimal MDP-policy which is stationary ($\Delta^* = \{\delta\}$). This does not hold when the horizon is finite.

Example 2.43. *[118] A robot is in position $(1,1)$ on the 4×3 grid of Figure 2.8. Reaching position $(4,3)$ offers a reward of $+1$ and reaching the undesired position $(4,2)$ gives a reward of -1. All other positions on the grid are rewarded with -0.04. At each time step, the robot decides to move up, down, left, or right. The intended effect occurs with probability 0.8, and the rest of the time, the robot moves at right angles to the intended direction. If an obstacle prevents the robot from moving, then it stays in the same position. If the robot reaches position $(4,2)$ or $(4,3)$, then it gets out of the grid. Last, the robot is always aware of its position.*

This problem can be modeled by the MDP $(\mathcal{S}, \mathcal{D}, P(. \mid ., .), U(., .))$, where

- $\mathcal{S} = (\{1,2,3,4\} \times \{1,2,3\}) \cup \{out\}$ *is the set of positions on the grid plus the out position;*
- $\mathcal{D} = \{up, down, left, right\}$ *is the set of available decisions at each time step;*

[10] Or one defines a function $G : \mathcal{S} \to \mathbb{R}$ specifying, for each state s at step T, a global expected gain concerning what occurs after T.

- *$P(.\,|\,.,.)$ is the transition model defining e.g. $P((1,2)\,|\,(1,1),up) = 0.8$, $P((2,1)\,|\,(1,1),up) = 0.1$, and $P((1,1)\,|\,(1,1),up) = 0.1$;*

- *$U(.,.)$ defines the rewards: $U((4,2),.) = -1$, $U((4,3),.) = 1$, $U(out,.) = 0$, and $U((i,j),.) = -0.04$ otherwise.*

The goal is to find an optimal MDP-policy. If each action produced the expected effect, then the sequence $[up, up, right, right, right]$ would be optimal. But with uncertainties, an optimal stationary MDP-policy when the horizon is infinite and $\gamma = 0.99$ consists of moving up if the position is $(1,1)$, $(1,2)$, or $(3,2)$, right if the position is $(1,3)$, $(2,3)$, or $(3,3)$, and left if the position is $(2,1)$, $(3,1)$, or $(4,1)$. Its expected utility is approximately 0.7.

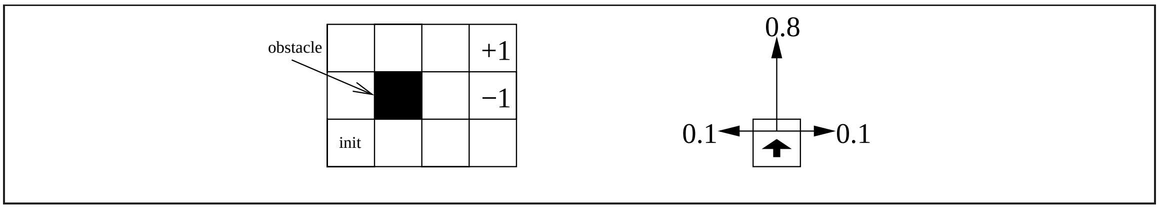


Figure 2.8: A sequential decision problem modelable with MDPs.

MDP-policies can be computed systematically. First, the maximal expected utility which can be obtained in one step starting from state s is $val_1^*(s) = \max_{d\in\mathcal{D}} U(s,d)$. A corresponding optimal decision rule is $\delta_1^*(s) \in \text{argmax}_{d\in\mathcal{D}} U(s,d)$. Second, the maximal expected utility which can be obtained if there are $t > 1$ remaining steps is given by the *Bellman equation*:

$$val_t^*(s) \quad = \quad \max_{d\in\mathcal{D}} (U(s,d) + \gamma \cdot \sum_{s'\in\mathcal{S}} P(s'\,|\,s,d) \cdot val_{t-1}^*(s'))$$

An associated decision rule, denoted δ_t^*, is obtained using $\text{argmax}_{d\in\mathcal{D}}$. For a T-step finite horizon MDP, this so-called *value iteration* mechanism gives an optimal policy $\Delta^* = \{\delta_1^*, \ldots, \delta_T^*\}$. For an infinite horizon MDP, this mechanism converges to an optimal stationary MDP-policy $\delta^* = \lim_{t\to\infty} \delta_t^*$.

The Bellman equation can be seen as a sequence of max and sum variable eliminations. Indeed, let $\mathbf{s}$ and $\mathbf{s'}$ (boldface letters) be variables with the state space as domain ($dom(\mathbf{s}) = dom(\mathbf{s'}) = \mathcal{S}$) and let $\mathbf{d}$ be a variable with the decision space as domain ($dom(\mathbf{d}) = \mathcal{D}$). Let $P_{\mathbf{s'}\,|\,\mathbf{s},\mathbf{d}}$ be the scoped function $(\{\mathbf{s'},\mathbf{s},\mathbf{d}\}, P(.\,|\,.,.))$, let $U_{\mathbf{s},\mathbf{d}}$ be the scoped function $(\{\mathbf{s},\mathbf{d}\}, U(.,.))$, let $val_{\mathbf{s}}^*$ be the scoped function $(\{\mathbf{s}\}, val_t^*(.))$, and let $val_{\mathbf{s'}}^*$ be the scoped function $(\{\mathbf{s'}\}, val_{t-1}^*(.))$. Then, the Bellman equation can be rewritten as:

$$val_{\mathbf{s}}^* \quad = \quad \max_{\mathbf{d}} \sum_{\mathbf{s'}} P_{\mathbf{s'}\,|\,\mathbf{s},\mathbf{d}} \cdot (U_{\mathbf{s},\mathbf{d}} + \gamma \cdot val_{\mathbf{s'}}^*) \qquad (2.20)$$

Similarly, given a finite horizon MDP with $\gamma = 1$, one can even unroll it to get the complete elimination sequence it performs. If $\mathbf{s_t}$ and $\mathbf{d_t}$ are variables denoting the state and decision at step t respectively ($dom(\mathbf{s_t}) = \mathcal{S}$ and $dom(\mathbf{d_t}) = \mathcal{D}$), and if $P_{\mathbf{s_{t+1}}\,|\,\mathbf{s_t},\mathbf{d_t}}$ and $U_{\mathbf{s_t},\mathbf{d_t}}$ denote the scoped functions $(\{\mathbf{s_{t+1}},\mathbf{s_t},\mathbf{d_t}\}, P(.\,|\,.,.))$ and $(\{\mathbf{s_t},\mathbf{d_t}\}, U(.,.))$ respectively, then the sequence of variable eliminations equivalent to the whole value iteration algorithm is:

$$\max_{\mathbf{d_1}} \sum_{\mathbf{s_2}} \max_{\mathbf{d_2}} \ldots \sum_{\mathbf{s_T}} \max_{\mathbf{d_T}} \left(\left(\prod_{t \in [1,T[} P_{\mathbf{s_{t+1}} | \mathbf{s_t}, \mathbf{d_t}} \right) \times \left(\sum_{t \in [1,T]} U_{\mathbf{s_t}, \mathbf{d_t}} \right)\right) \tag{2.21}$$

2.6.2 Partially observable MDPs

MDPs assume that the state s of the environment is completely observable. But in many problems, the state of the environment is not exactly known when a decision is made. Only noisy observations of the actual state are available to the decision maker. The formalism extending MDPs to integrate such aspects is the Partially Observable MDP (POMDP [132, 89, 83, 71]) formalism. [11]

Definition 2.44. *A POMDP is a tuple* $(\mathcal{S}, \mathcal{D}, P(.\,|\,.,.), U(.,.), \Omega, O(.\,|\,.))$ *where:*

- $(\mathcal{S}, \mathcal{D}, P(.\,|\,.,.), U(.,.))$ *is a MDP;*
- Ω *is a finite set of possible observations;*
- $O(.\,|\,.) : \Omega \times \mathcal{S} \to [0,1]$ *is a function such that* $O(o\,|\,s)$ *is the probability of making observation* o *in state* s *(observational model).*

The goal is still to seek a policy which maximizes the expected utility. A first naive and suboptimal approach is to specify at each step t a decision to be made depending on the observation made at t. The optimal method is to specify at each step t a decision to be made depending on all previous observations.

For a finite horizon POMDP, POMDP-policies are defined by a tree, as in a stochastic CSP. The root of this tree corresponds to the first decision to be made. It has as many sons as possible observations. A son of the root corresponds to the second decision to be made, depending on the first observation. And so on to the last stage.

The value of a POMDP-policy is defined by its expected utility, as in a stochastic CSP. Without further details, it can be shown that if there are T steps, the search for an optimal POMDP policy can be reduced to the computation of a sequence of eliminations of the form:

$$\sum_{\mathbf{o_1}} \max_{\mathbf{d_1}} \sum_{\mathbf{o_2}} \max_{\mathbf{d_2}} \ldots \sum_{\mathbf{o_T}} \max_{\mathbf{d_T}} \sum_{\mathbf{s_1}, \ldots, \mathbf{s_T}} \left(\left(\prod_{t \in [1,T[} P_{\mathbf{s_{t+1}} | \mathbf{s_t}, \mathbf{d_t}} \times \prod_{t \in [1,T]} P_{\mathbf{o_t} | \mathbf{s_t}} \right) \times \left(\sum_{t \in [1,T]} U_{\mathbf{s_t}}, \mathbf{d_t} \right)\right) \tag{2.22}$$

2.6.3 Other uncertainty-utility models: towards algebraic MDPs

In another direction, the initial probabilistic MDP framework was adapted to other representations of uncertainties and utilities.

In *possibilistic* MDPs [119], conditional probabilities $P(s'\,|\,s, d)$ are replaced by conditional possibilities $\pi(s'\,|\,s, d)$ and additive rewards $U(s, d)$ by preferences $\mu(s, d)$ combined using min.

In *pessimistic* possibilistic finite horizon MDPs, which use the pessimistic possibilistic expected utility [43], the search for an optimal policy can be reduced to the computation of optimal decision rules for the quantity:

$$\max_{\mathbf{d_1}} \min_{\mathbf{s_2}} \max_{\mathbf{d_2}} \ldots \min_{\mathbf{s_T}} \max_{\mathbf{d_T}} \left(\max\left(1 - \min_{t \in [1,T[} \pi_{\mathbf{s_{t+1}} | \mathbf{s_t}, \mathbf{d_t}}, \min_{t \in [1,T]} \mu_{\mathbf{s_t}, \mathbf{d_t}}\right)\right) \tag{2.23}$$

[11] When there is no decision, the corresponding model is Hidden Markov Model [112].

In other words, plausibilities are combined using min, utilities are combined using min, plausibilities and utilities are combined using $(p, u) \to \max(1 - p, u)$, min-eliminations are performed for environment variables, and max-eliminations are performed for decision variables.

In *optimistic* possibilistic MDPs [119], which use the optimistic possibilistic expected utility [43], the sequence of eliminations is

$$\max_{\mathbf{d_1}} \max_{\mathbf{s_2}} \max_{\mathbf{d_2}} \ldots \max_{\mathbf{s_T}} \max_{\mathbf{d_T}} (\min(\min_{t \in [1,T[} \pi_{\mathbf{s_{t+1}} | \mathbf{s_t}, \mathbf{d_t}}, \min_{t \in [1,T]} \mu_{\mathbf{s_t}, \mathbf{d_t}}) \tag{2.24}$$

In MDPs using κ-rankings [133, 142] as a model of uncertainties and using only positive utilities [59], the sequence of eliminations looks like:

$$\min_{\mathbf{d_1}} \min_{\mathbf{s_2}} \min_{\mathbf{d_2}} \ldots \min_{\mathbf{s_T}} \min_{\mathbf{d_T}} ((\sum_{t \in [1,T[} \kappa_{\mathbf{s_{t+1}} | \mathbf{s_t}, \mathbf{d_t}}) + (\sum_{t \in [1,T]} U_{\mathbf{s_t}, \mathbf{d_t}})) \tag{2.25}$$

Algebraic MDPs

The fact that only the elimination and combination operators used change between MDPs using different kinds of uncertainty-utility models was recognized, in the finite horizon case, by the Algebraic MDP (AMDP [97]) framework. AMDPs are based on generic existing structures for modeling plausibilities [54, 62] and expected utilities [23]. The transition model is expressed by a so-called *conditional plausibility measure* of reaching a state s' starting from state s and applying decision d, denoted $\mathcal{P}(s' \mid s, d)$. Rewards are combined using an abstract operator $\otimes$. AMDPs define an algebraic form of the Bellman equation, which uses two abstract operators $\boxplus$ and $\boxtimes$ enabling to compute an expected utility. This algebraic Bellman equation is of the form:

$$val_t^*(s) \quad = \quad \max_{d \in \mathcal{D}} (U(s,d) \otimes (\boxplus_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, d) \boxtimes val_{t-1}^*(s'))) \tag{2.26}$$

AMDPs impose axioms on the operators used, which, once again, reduce the computations they perform to a sequence of variable eliminations on a combination of scoped functions.

2.6.4 Back to a variable-based representation: factored MDPs

We have argued at the beginning of Subsection 2.6.1 that MDPs basically use a state-based representation. The drawbacks of this rather raw representation were however surmounted with an adaptation of MDPs, the factored MDP [19, 18] framework, which uses variables representing the basic features of the state of the system. The following small example illustrates factored MDPs and the interest of such a variable-based representation.

Example 2.45. *The robot in the* 4×3 *grid of Figure 2.8 has a limited amount of energy varying from* 0 *to* 9. *This amount is decremented by* 1 *at each step, and the robot can move on the grid only if it has a strictly positive energy level.*

With these new specifications, the state of the robot can be described by its position pos and by its level of energy en. At each step, the global state s corresponds to the aggregation of pos and en. In this case, there are $|\mathcal{S}| = 130$ *possible states for the robot (*13 *positions with the out position, and* 10 *energy levels). In order to define the transition model* $P(s' \mid s, d)$, *one must define* $130 \times 130 \times 4 = 67600$ *individual probabilities.*

This unfactored representation can be improved. Indeed, the conditional probability distribution $P(s' \mid s, d)$ can be factored as $P(pos' \mid pos, en, d) \times P(en' \mid en)$, since the energy level at step $t+1$ does not depend on the position and on the decision made at step t. With this factored representation, we need to specify only $13 \times 13 \times 10 \times 4 + 10 \times 10 = 6860$ individual probabilities! The factored and unfactored representations are given in Figure 2.9. The state representation is inadequate because the number of states grows exponentially with the number of variables describing the state.[12]

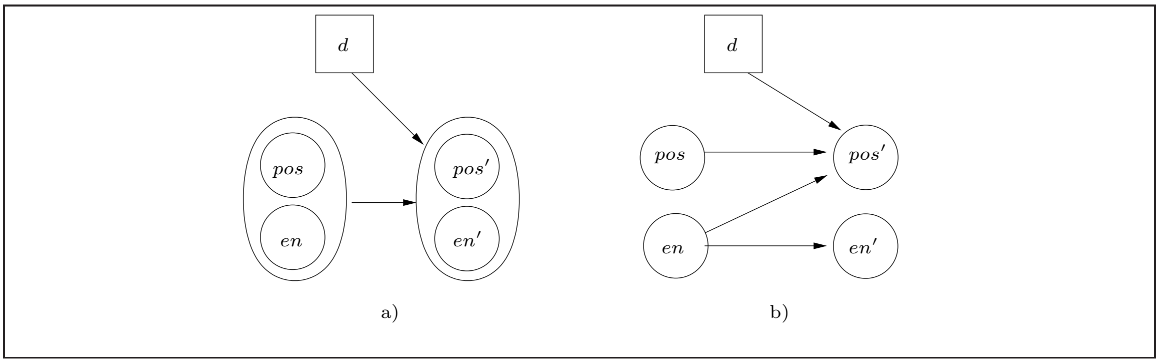


Figure 2.9: (a) Unfactored MDP representation; (b) Factored MDP representation.

Factored MDPs actually correspond to MDPs where the transition model $P(. \mid ., .)$ is given by one so-called Dynamic Bayesian Network (DBN [31]) per decision. This DBN gives the probabilistic dependences between the variables representing the full state. When decisions are themselves represented by decision variables, the obtained formalism is called Dynamic Decision Network (DDN [118]).

2.7 Valuation algebras

The previous study shows that the formalisms developed in the CSP, BN, or MDP frameworks present many interesting similarities in that various queries in these formalisms can be reduced to the computation of a sequence of variable eliminations on a combination of scoped functions. The idea of a generic algebraic framework for modeling and solving decision problems based on variables and local functions between these variables was actually already proposed for the mono-operator case (one elimination operator and one combination operator) under the name of *valuation algebras*[127, 128, 75].

In order to introduce valuation algebras, it is first necessary to define the notions of valuation, combination of two valuations, and marginalization of a valuation.

First, a valuation is strictly identical to a scoped function. For instance, given a BN, a conditional probability distribution $P_{x \mid pa_G(x)}$ is a valuation of scope $sc(P_{x \mid pa_G(x)}) = \{x\} \cup pa_G(x)$. In the valuation algebras terminology, the scope of a valuation is called its domain. Second, let Ψ denote a set of valuations. Two abstract operators are defined directly on valuations (and not on the image E of a valuation $\varphi : dom(sc(\varphi)) \to E$):

1. A *combination* operator $\boxtimes : \Psi \times \Psi \to \Psi$ associating with two valuations φ_1, φ_2 their combination $\varphi_1 \boxtimes \varphi_2$. In order to handle CSPs, the combination operator $\boxtimes$ equals $\wedge$. In order to

[12] The factorization can even be improved by using for example a decision diagram representation [1, 21] where the fact that the robot does not move if its level of energy equals 0 is explicitly taken into account.

handle BNs, $\boxtimes = \times$.

2. A *marginalization* operator denoted $\downarrow: \Psi \times 2^V \rightarrow \Psi$ associating with a valuation φ and a set of variables $S \subset V$ a projected valuation $\varphi^{\downarrow S}$. In order to seek a solution to a CSP, this marginalization corresponds to an elimination of the variables in $sc(\varphi) - S$ using max. In order to compute a marginal probability distribution on a BN, this marginalization corresponds to an elimination of the variables in $sc(\varphi) - S$ using $+$.

As a unique combination operator is used, the information is combined in the same way independently of what it represents. As a unique marginalization operator is used, the information is synthesized in the same way independently of the variable considered. The addition of some axioms defines *valuation algebras.*

Definition 2.46. *A* valuation algebra *is a tuple* $(V, \Psi, \boxtimes, \downarrow)$ *such that* V *is a set of variables,* Ψ *is the set of all valuations whose scopes are included in* V*, and* $\boxtimes$ *and* $\downarrow$ *satisfy the following axioms:*

1. $(\Psi, \boxtimes)$ *is a semigroup, i.e.* $\boxtimes$ *is associative, commutative, and, for each* $S \subset V$, $\boxtimes$ *has an identity on each* $\Psi_S = \{\varphi \in \Psi \mid sc(\varphi) = S\}$*, i.e.* $\exists e_S \in \Psi_S \ \forall \varphi \in \Psi_S \ , \ \varphi \boxtimes e_S = \varphi$.

2. *Combining two valuations* φ_1*,* φ_2 *gives a valuation with scope* $sc(\varphi_1) \cup sc(\varphi_2)$.

3. *Every marginalization* $\varphi^{\downarrow S}$ *gives a valuation with scope* $sc(\varphi) \cap S$*. Moreover,* $\varphi^{\downarrow sc(\varphi)} = \varphi$ *and* $\varphi^{\downarrow S} = \varphi^{\downarrow S \cap sc(\varphi)}$.

4. *Transitivity of marginalization: for every valuation* φ *and for every* S_1*,* S_2 *subsets of* V $(\varphi^{\downarrow S_1})^{\downarrow S_2} = \varphi^{\downarrow S_1 \cap S_2}$.

5. *Distributivity of marginalization over combination: for all valuations* φ_1*,* φ_2*,* $(\varphi_1 \boxtimes \varphi_2)^{\downarrow sc(\varphi_1)} = \varphi_1 \boxtimes ({\varphi_2}^{\downarrow sc(\varphi_1)})$.

6. *Identity elements:* $\forall S_1, S_2 \subset V \ , \ e_{S_1} \boxtimes e_{S_2} = e_{S_1 \cup S_2}$.

Given a set of valuations $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ and a set of variables $S \subset V$, a possible query on valuation algebras is to compute $(\varphi_1 \boxtimes \ldots \boxtimes \varphi_n)^{\downarrow S}$. This corresponds to eliminating variables in $sc(\varphi_1) \cup \ldots \cup sc(\varphi_n) - S$ on the combination of the scoped functions in Φ. One of the most significant contribution of the valuation algebra framework is that it contains sufficient axioms for generic variable elimination algorithms to be used. The main idea is to choose an order in which variables in $V - S$ are eliminated, and then to use the distributivity of $\boxtimes$ over $\downarrow$, in order to write decompositions like

$$(\varphi_1 \boxtimes \ldots \boxtimes \varphi_n)^{\downarrow S-x} = (\mathop{\boxtimes}\limits_{\varphi \in \Phi, x \notin sc(\varphi)} \varphi) \boxtimes ((\mathop{\boxtimes}\limits_{\varphi \in \Phi, x \in sc(\varphi)} \varphi)^{\downarrow S-x}) \tag{2.27}$$

The computations performed are local in the sense that when a variable x is eliminated, only valuations having x in their scopes need to be considered.

Example 2.47. *Let us consider Mr Holmes' alarm again. One possible query was to compute* $P_{al} = \sum_{eq,bu,jc,mc} \big(P_{eq} \times P_{bu} \times P_{al \,|\, eq,bu} \times P_{jc \,|\, al} \times P_{mc \,|\, al}\big)$.

In order to solve this problem, one can use the valuation algebra $(V, \Psi, \boxtimes, \downarrow)$ *where* $V = \{eq, bu, al, jc, mc\}$*,* Ψ *is the set of valuations* $dom(S) \to \mathbb{R}^+$ *with* $S \subset V$*,* $\boxtimes = \times$*, and* $\varphi^{\downarrow S} = \sum_{sc(\varphi)-S} \varphi$*.*

The goal is then to compute $(P_{eq} \boxtimes P_{bu} \boxtimes P_{al\,|\,eq,bu} \boxtimes P_{jc\,|\,al} \boxtimes P_{mc\,|\,al})^{\downarrow al}$*. Variables in* $V - \{al\} = \{eq, bu, jc, mc\}$ *must be eliminated. If one chooses to eliminate variables in the order eq, bu, jc, mc, then the decomposition of the global computation to be performed into local computations is given below. It uses the basic axioms of valuation algebras so that when eliminating a variable* x*, only scoped functions with* x *in their scopes are considered. Note that normalization conditions could be used to simplify the computations.*

step 0	$P_{eq} \boxtimes P_{bu} \boxtimes P_{al\,|\,eq,bu} \boxtimes P_{jc\,|\,al} \boxtimes P_{mc\,|\,al}$
step 1: elim(eq)	$P_{bu} \boxtimes P_{jc\,|\,al} \boxtimes P_{mc\,|\,al} \boxtimes (P_{eq} \boxtimes P_{al\,|\,eq,bu})^{\downarrow eq}$
step 2: elim(bu)	$P_{jc\,|\,al} \boxtimes P_{mc\,|\,al} \boxtimes (P_{bu} \boxtimes (P_{eq} \boxtimes P_{al\,|\,eq,bu})^{\downarrow eq})^{\downarrow bu}$
step 3: elim(jc)	$(P_{jc\,|\,al})^{\downarrow jc} \boxtimes P_{mc\,|\,al} \boxtimes (P_{bu} \boxtimes (P_{eq} \boxtimes P_{al\,|\,eq,bu})^{\downarrow eq})^{\downarrow bu}$
step 4: elim(mc)	$(P_{jc\,|\,al})^{\downarrow jc} \boxtimes (P_{mc\,|\,al})^{\downarrow mc} \boxtimes (P_{bu} \boxtimes (P_{eq} \boxtimes P_{al\,|\,eq,bu})^{\downarrow eq})^{\downarrow bu}$

2.8 The three basic ingredients of a generic framework for sequential decision making with uncertainties, feasibilities, and utilities

The previous subsections show that usual queries considered in various existing formalisms can be reduced to a sequence of variable eliminations on a combination of scoped functions. These formalisms can all be seen as graphical models and differ mainly in the way eliminations and combinations are performed and in what variables and scoped functions represent.

This kind of observation has led to the definition of algebraic MDPs [97] or to the definition of valuation algebras [127, 75], the latter being a generic algebraic framework in which eliminations can be performed on a combination of scoped functions. However, valuation algebras are defined using only one combination operator, whereas several combination operators may be needed to manipulate the different types of scoped functions in *composite* graphical models. Moreover, valuation algebras can deal with only one type of elimination, whereas several elimination operators may be required for handling the different types of variables. In valuation networks [130], plausibilities are necessarily represented as probabilities, and min-eliminations cannot be performed. Essentially, a more powerful framework is needed.

In order to be simple and yet general enough to cover various queries asked in various formalisms, the generic form we need to consider is:

$$Sov\left(\left(\bigwedge_{F_i \in F} F_i\right) \star \left(\underset{P_i \in P}{\otimes_p} P_i\right) \otimes_{pu} \left(\underset{U_i \in U}{\otimes_u} U_i\right)\right) \tag{2.28}$$

where (1) $\wedge$ is used to combine local feasibilities, $\otimes_p$ is used to combine local plausibilities, $\otimes_u$ is used to combine local utilities, $\otimes_{pu}$ is used to combine plausibilities and utilities, and the truncation operator $\star$ is used to ignore unfeasible decisions without having to deal with elimination operations on restricted domains;[13] (2) F, P, U are (possibly empty) sets of local feasibility, plausibility, and utility functions respectively; (3) Sov is an operator-variable(s) sequence, indicating how to eliminate variables. Sov involves min or max on decision variables and an operator $\oplus_u$ on environment variables.

Equation 2.28 is still very informal. To define it formally, and to provide it with a clear semantics, we need to define three key elements.

1. First, we must define $\otimes_p$, $\otimes_u$, $\otimes_{pu}$, the operators used to respectively combine plausibilities, utilities, and plausibilities with utilities, as well as $\oplus_u$, the operator used to eliminate environment variables. An elimination operator $\oplus_p$ on plausibilities, enabling us to synthesize information coming from plausibilities, is also introduced.

   These operators define the flexible *algebraic structure* of the PFU framework. Semantically speaking, they define the plausibility/utility model and must satisfy some basic algebraic properties.

2. Second, we must organize the information as a graphical model involving a set of *variables*, and sets of *scoped functions* expressing plausibilities, feasibilities, and utilities (sets P, F, U). Together, they define a *PFU network*, exploiting graphical models concepts (locality, conditional independence). The possibility to express information in such a structured form must also be justified, e.g. using the notion of conditional independence.

3. Last, in order to formulate decision making problems, we need to define *queries* on PFU networks, by introducing a sequence of operator-variable(s) pairs Sov applied on the combination of the scoped functions as in Equation 2.28. Queries must allow to model various situations in terms of partial observability and controllability. We must also show why computing such quantities is of interest from the decision theory point of view by comparing Equation 2.28 with a standard decision tree approach.

2.9 Summary

We have informally shown that usual queries formulated in various formalisms reasoning about plausibilities and/or feasibilities and/or utilities can be reduced to sequences of variable eliminations on combinations of scoped functions, using various operators. They can *intuitively* be covered by Equation 2.28. The three key elements (an algebraic structure, a PFU network, and a sequence of variable eliminations) needed to *formally* define and give sense to this equation are introduced in Chapters 3, 4, and 5.

[13] In Equation 2.28, all local plausibilities are combined using the same operator $\otimes_p$ and all local utilities are combined using the same operator $\otimes_u$: the proposed graphical model is composite only in the sense that there are different types of scoped functions (plausibilities, feasibilities, and utilities). However, the generic form of Equation 2.28 does not prevent from having different kinds of information contained among each type of scoped functions: e.g., if one wants to manipulate both probabilities and possibilities, one can take $\otimes_p$ defined on (probability,possibility) pairs by $(p_1, \pi_1) \otimes_p (p_2, \pi_2) = (p_1 \times p_2, \min(\pi_1, \pi_2))$.

Chapter 3

A generic algebraic structure for sequential decision under uncertainty

The first element of the PFU framework is an algebraic structure specifying how the information provided by plausibilities, feasibilities, and utilities is combined and synthesized. This algebraic structure is obtained by adapting previous structures from Friedman, Chu, and Halpern [54, 62, 23] for representing uncertainties and expected utilities. It involves combination and elimination operators which satisfy some algebraic properties. Moreover, it covers various existing algebraic structures used in different plausibility/utility models.

3.1 Some algebraic definitions

Definition 3.1. $(E, \circledast)$ *is a* commutative monoid *iff* E *is a set and* $\circledast$ *is a binary operator on* E *which is associative* $(x \circledast (y \circledast z) = (x \circledast y) \circledast z)$*, commutative* $(x \circledast y = y \circledast x)$*, and which has an identity* $1_E \in E$ $(x \circledast 1_E = 1_E \circledast x = x)$*.*

Definition 3.2. $(E, \oplus, \otimes)$ *is a* commutative semiring *iff*

- $(E, \oplus)$ *is a commutative monoid, with an identity denoted* 0_E*,*
- $(E, \otimes)$ *is a commutative monoid, with an identity denoted* 1_E*,*
- 0_E *is annihilator for* $\otimes$ $(x \otimes 0_E = 0_E)$*,*
- $\otimes$ *distributes over* $\oplus$ $(x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z))$*.*

Definition 3.3. *Let* $(E_a, \oplus_a, \otimes_a)$ *be a commutative semiring. Then,* $(E_b, \oplus_b, \otimes_{ab})$ *is a semimodule on* $(E_a, \oplus_a, \otimes_a)$ *iff*

- $(E_b, \oplus_b)$ *is a commutative monoid, with an identity denoted* 0_{E_b}*,*
- $\otimes_{ab} : E_a \times E_b \rightarrow E_b$ *satisfies*

- $\otimes_{ab}$ *distributes over* $\oplus_b$ $(a \otimes_{ab} (b_1 \oplus_b b_2) = (a \otimes_{ab} b_1) \oplus_b (a \otimes_{ab} b_2))$*,*
- $\otimes_{ab}$ *distributes over* $\oplus_a$ $((a_1 \oplus_a a_2) \otimes_{ab} b = (a_1 \otimes_{ab} b) \oplus_b (a_2 \otimes_{ab} b))$*,*
- *linearity property:* $a_1 \otimes_{ab} (a_2 \otimes_{ab} b) = (a_1 \otimes_a a_2) \otimes_{ab} b$*,*
- *for all* $b \in E_b$*,* $0_{E_a} \otimes_{ab} b = 0_{E_b}$ *and* $1_{E_a} \otimes_{ab} b = b$*.*

Definition 3.4. *Let* E *be a set with a partial order* $\preceq$*. An operator* $\circledast$ *on* E *is monotonic iff* $(x \preceq y) \rightarrow (x \circledast z \preceq y \circledast z)$ *for all* $x, y, z \in E$*.*

3.2 Plausibility structure

Various forms of plausibilities exist. The most usual one is *probabilities*. As shown previously, for example with Equation 2.9 page 33 which involves the quantity $\sum_{V-\{y\}} \left(\prod_{x \in V} P_{x \mid pa_G(x)}\right)$, one uses $\otimes_p = \times$ to combine probabilities and $\oplus_p = +$ as an elimination operator.

But plausibilities can also be expressed as *possibility degrees* in $[0, 1]$. Possibilities are eliminated using $\oplus_p = \max$ and *usually* combined using $\otimes_p = \min$. An interesting case appears when possibility degrees are booleans describing which states of the environment are completely possible or impossible. Plausibilities are then combined using $\otimes_p = \wedge$ and eliminated using $\oplus_p = \vee$.

Another example is Spohn's epistemic beliefs, also known as κ-rankings (kappa rankings) [133, 142, 59]. In this case, plausibilities are elements of $\mathbb{N} \cup \{+\infty\}$ called *surprise degrees*, 0 is associated with non-surprising situations, $+\infty$ is associated with completely surprising (impossible) situations, and more generally a surprise degree k can be viewed as a probability of ϵ^k for an infinitesimal ϵ. Surprise degrees are combined using $\otimes_p = +$ and eliminated using $\oplus_p = \min$.

To capture these various plausibility modeling frameworks, we start from Friedman-Halpern's work on *plausibility measures* [54, 62] (similar approaches are developed in [140, 27]).

Friedman-Halpern's structure Assume we want to express plausibilities over the assignments of a set of variables S. Each subset of $dom(S)$ is called an *event*. In [54, 62], plausibilities are elements of a set E_p called the plausibility domain. E_p is equipped with a partial order $\preceq_p$ and with two special elements 0_p and 1_p satisfying $0_p \preceq_p p \preceq_p 1_p$ for all $p \in E_p$. A function $Pl : 2^{dom(S)} \rightarrow E_p$ is a plausibility measure over S iff it satisfies $Pl(\emptyset) = 0_p$, $Pl(dom(S)) = 1_p$, and $(W_1 \subset W_2) \rightarrow (Pl(W_1) \preceq_p Pl(W_2))$. This means that 0_p is associated with impossibility, 1_p is associated with the highest plausibility degree, and the plausibility degree of a set is as least as high as the plausibility degree of each of its subsets.

Among all plausibility measures, we focus on so-called *algebraic conditional plausibility measures*, which use abstract functions $\oplus_p$ and $\otimes_p$ which are analogous to $+$ and $\times$ for probabilities. These measures satisfy properties such as *decomposability*: for all disjoint events W_1, W_2, $Pl(W_1 \cup W_2) = Pl(W_1) \oplus_p Pl(W_2)$. As $\cup$ is associative and commutative, it follows for example that $\oplus_p$ is associative and commutative on representations of disjoint events, i.e. $(a \oplus_p b) \oplus_p c = a \oplus_p (b \oplus_p c)$ and $a \oplus_p b = b \oplus_p a$ if there exist pairwise disjoint sets W_1, W_2, W_3 such that $Pl(W_1) = a$, $Pl(W_2) = b$, $Pl(W_3) = c$.

Restriction of Friedman-Halpern's structure An important point in Friedman-Halpern's work is that the algebraic properties of $\oplus_p$ and $\otimes_p$ hold only on the domains of definition of $\oplus_p$ and

$\otimes_p$. Although this is sufficient to express and manipulate plausibilities, it can be algorithmically restrictive. Indeed, consider a Bayesian network involving two boolean variables $\{x_1, x_2\}$ and define P_{x_1,x_2} as $P_{x_1} \times P_{x_2 \,|\, x_1}$. Assume that P_{x_1} is a constant factor $L_0 = 0.5$. In order to evaluate $P_{x_2}((x_2, t))$, the quantity $\sum_{x_1} L_0 \times P_{x_2 \,|\, x_1}((x_2, t))$ must be computed. To do so, it is simpler to factor it and compute $L_0 \times \sum_{x_1} P_{x_2 \,|\, x_1}((x_2, t))$. If $P_{x_2 \,|\, x_1}((x_2, t).(x_1, t)) = 0.6$ and $P_{x_2 \,|\, x_1}((x_2, t).(x_1, f)) = 0.8$, the answer is $0.5 \times (0.6 + 0.8) = 0.7$. Performing $0.6 + 0.8$ requires applying addition outside of the range of usual probabilities, for which $a \oplus_p b$ is defined only if $a + b \leq 1$, since two probabilities whose sum exceeds 1 cannot be associated with disjoint events.

To take such issues into account, we adapt Friedman-Halpern's E_p, $\oplus_p$, $\otimes_p$ so that $\oplus_p$ and $\otimes_p$ become closed in E_p and so that Friedman-Halpern's axioms hold in the closed structure. Once this closure is performed, we obtain a *plausibility structure*.

Definition 3.5. *A* plausibility structure *is a tuple* $(E_p, \oplus_p, \otimes_p)$ *such that*

- $(E_p, \oplus_p, \otimes_p)$ *is a commutative semiring (identities for* $\oplus_p$ *and* $\otimes_p$ *are denoted* 0_p *and* 1_p *respectively),*
- E_p *is equipped with a partial order* $\preceq_p$ *such that* $0_p = \min(E_p)$ *and such that* $\oplus_p$ *and* $\otimes_p$ *are monotonic with respect to* $\preceq_p$.

Elements of E_p *are called* plausibility degrees.

Note that 1_p is not necessarily the maximal element of E_p. For probabilities, Friedman and Halpern's structure would be $([0,1], +', \times)$, where $a +' b = a + b$ if $a + b \leq 1$ and is undefined otherwise. In order to get closed operators, we take $(E_p, \oplus_p, \otimes_p) = (\mathbb{R}^+, +, \times)$ and therefore $1_p = 1$ is not the maximal element in E_p. In some cases, Friedman-Halpern's structure does need to be closed. This is the case with κ-rankings (already closed: $(E_p, \oplus_p, \otimes_p) = (\mathbb{N} \cup \{+\infty\}, \min, +)$) and with possibilities (already closed: $(E_p, \oplus_p, \otimes_p)$ is typically $([0,1], \max, \min)$, although other choices such as $([0,1], \max, \times)$ are possible).

Given two plausibility structures $(E_p, \oplus_p, \otimes_p)$ and $(E'_p, \oplus'_p, \otimes'_p)$, if we define $E = E_p \times E'_p$, $(p_1, p'_1) \oplus (p_2, p'_2) = (p_1 \oplus_p p_2, p'_1 \oplus'_p p'_2)$ and $(p_1, p'_1) \otimes (p_2, p'_2) = (p_1 \otimes_p p_2, p'_1 \otimes'_p p'_2)$, then $(E, \oplus, \otimes)$ is a plausibility structure too. This allows us to deal with different kinds of plausibilities (such as probabilities and possibilities) or with families of probability distributions.

From plausibility measures to plausibility distributions

Let us consider a plausibility measure [54, 62] $Pl : 2^{dom(S)} \rightarrow E_p$ over a set of variables S. Assume that $Pl(W_1 \cup W_2) = Pl(W_1) \oplus_p Pl(W_2)$ for all disjoint sets $W_1, W_2 \in 2^{dom(S)}$, as is the case with Friedman-Halpern's *algebraic plausibility measures*. This assumption entails that $Pl(W) = \oplus_{p\,A \in W}\, Pl(\{A\})$ for all $W \in 2^{dom(S)}$. This holds even for $W = \emptyset$ since 0_p is the identity of $\oplus_p$. Hence, defining $Pl(\{A\})$ for all complete assignments A of S suffices to describe Pl. Moreover, in this case, the three conditions defining plausibility measures ($Pl(dom(S)) = 1_p$, $Pl(\emptyset) = 0_p$, and $(W_1 \subset W_2) \rightarrow (Pl(W_1) \preceq_p Pl(W_2))$) are equivalent to just $\oplus_{p\,A \in dom(S)}\, Pl(\{A\}) = 1_p$, using the monotonicity of $\oplus_p$ for the third condition. This means that we can deal with *plausibility distributions* instead of plausibility measures:

Definition 3.6. *A* plausibility distribution *over* S *is a function* $\mathcal{P}_S : dom(S) \rightarrow E_p$ *such that* $\oplus_{p\,A \in dom(S)}\, \mathcal{P}_S(A) = 1_p$.

The normalization condition imposed on plausibility distributions is simply a generalization of the convention that probabilities sum up to 1. It captures the fact that the disjunction of all the assignments of S has 1_p as a plausibility degree.

Proposition 3.7. *A plausibility distribution $\mathcal{P}_S$ can be extended to give a plausibility distribution $\mathcal{P}_{S'}$ over every $S' \subset S$, defined by $\mathcal{P}_{S'} = \oplus_{p\,S-S'} \mathcal{P}_S$.*

3.3 Feasibility structure

Feasibilities define whether a decision is possible or not, and are therefore expressed as booleans in $\{t, f\}$. This set is equipped with the total order $\preceq_{bool}$ satisfying $f \prec_{bool} t$.

Boolean scoped functions, expressing feasibilities, are combined using the operator $\wedge$ since an assignment of decision variables is feasible iff all feasibility functions agree that this assignment is feasible.

Given a scoped function F_i expressing feasibilities, it is possible to know whether an assignment A of a set of variables S is feasible according to F_i by computing $\vee_{sc(F_i)-S}\, F_i(A)$, since A is feasible according to F_i iff one of its extensions over $sc(F_i)$ is feasible. This means that feasibilities are synthesized using the elimination operator $\vee$.

As a result, feasibilities are expressed using the *feasibility structure* $S_f = (\{t, f\}, \vee, \wedge)$. S_f is not only a commutative semiring, but also a plausibility structure. Therefore, all plausibility notions and properties apply to feasibility. We may therefore speak of feasibility distributions, and the normalization condition $\vee_S \mathcal{F}_S = t$ imposed on a feasibility distribution $\mathcal{F}_S$ over S means that at least one decision must be feasible.

3.4 Utility structure

Utilities express preferences and can take various forms. If additive utilities, combined using $+$, are the most usual, utilities can also model priorities combined using $\otimes_u = \min$. When utilities represent absolute requirements, they can be modeled as booleans combined using $\otimes_u = \wedge$.

More generally, utility degrees are defined as elements of a set E_u equipped with a partial order $\preceq_u$. Smaller utility degrees are associated with less preferred events. Utility degrees are combined using an operator $\otimes_u$ which is assumed to be associative and commutative. This guarantees that combined utilities do not depend on the way combination is performed. We also assume that $\otimes_u$ admits an identity $1_u \in E_u$, representing indifference. This ensures the existence of a default utility degree when there are no utility scoped functions. We also assume that $\otimes_u$ is monotonic, so that if a local utility decreases, the global utility cannot increase. These properties are captured in the following notion of *utility structure*.

Definition 3.8. *$(E_u, \otimes_u)$ is a* utility structure *iff it is a commutative monoid and E_u is equipped with a partial order $\preceq_u$ such that $\otimes_u$ is monotonic. Elements of E_u are called utility degrees.*

E_u may have a minimum element $\bot_u$ representing unacceptable events and which will be an annihilator for $\otimes_u$ (the combination of any event with an unacceptable one must be unacceptable too). But these properties are not necessary to establish the forthcoming results.

The distinction between plausibilities, feasibilities, and utilities is important and can be justified using algebraic arguments. Since $\otimes_p$ and $\otimes_u$ may be different operators (for example, $\otimes_p = \times$ and $\otimes_u = +$ in usual probabilities with additive utilities), we must distinguish plausibilities and utilities. It is also necessary to distinguish feasibilities from utilities or plausibilities. Indeed, imagine a simple card game involving two players P_1 and P_2, each having three cards: a jack J, a queen Q, and a king K. P_1 must first play one card $x \in \{J, Q, K\}$, then P_2 must play a card $y \in \{J, Q, K\}$, and last P_1 must play a card $z \in \{J, Q, K\}$. A rule forbids to play the same card consecutively (feasibility functions $F_{xy} : x \neq y$ and $F_{yz} : y \neq z$). The goal for P_1 is that his two cards x and z have a value strictly better than P_2's card y. By setting $J < Q < K$, this requirement corresponds to two utility functions $U_{xy} : x > y$ and $U_{yz} : z > y$. In order to compute optimal decisions in presence of unfeasibilities, we must restrict optimizations (eliminations of decision variables with max or min) to feasible values: instead of $\max_x \min_y \max_z (U_{xy} \wedge U_{yz})$, we must compute:

$$\max_{a \in dom(x)} \left(\min_{b \in dom(y), F_{xy}(a,b)=t} \left(\max_{c \in dom(z), F_{yz}(b,c)=t} (U_{xy}(a,b) \wedge U_{yz}(b,c)) \right) \right)$$

which, by setting $f \prec t$, is logically equivalent to

$$\max_x \min_y \left(F_{xy} \rightarrow \max_z (F_{yz} \wedge (U_{xy} \wedge U_{yz})) \right)$$

In the latter quantity, feasibility functions concerning P_2's play (y) are taken into account using logical connective $\rightarrow$, so that P_2's unfeasible decisions are ignored in the set of all scenarios considered. Feasibility functions concerning P_1's last move (z) are taken into account using $\wedge$, so that P_1 does not consider scenarios in which he achieves a forbidden move. Therefore, feasibility functions cannot be handled simply by using the same combination operator as for utility functions: we need to dissociate what is unfeasible for all decision makers (unfeasibility is absolute) from what is unacceptable or required for one decision maker only (utility is relative).

At a more general level, for example when U_{xy} and U_{yz} are soft requirements or when we do not know exactly in advance who controls which variable, the logical connectives $\wedge$ and $\rightarrow$ cannot be used anymore. In order to ignore unfeasible values in decision variables elimination, we use the truncating operator $\star$ introduced in Definition 1.6. In order to eliminate a variable x from a local function φ while ignoring unfeasibilities indicated by a feasibility function F_i, we simply perform the elimination of x on $(F_i \star \varphi)$ instead of φ. This maps unfeasibilities to the value $\diamondsuit$, which is defined as an identity for elimination operators (see Definition 1.6). On the example above, if U_{xy} and U_{yz} were additive gains and costs, we would compute

$$\max_x \min_y \left(F_{xy} \star \max_z (F_{yz} \star (U_{xy} + U_{yz})) \right)$$

3.5 Expected utility structure

To define expected utilities, plausibilities and utilities must be combined. Consider a situation where a utility u_i is obtained with a plausibility p_i for all $i \in [1, N]$, with $p_1 \oplus_p \ldots \oplus_p p_N = 1_p$. $\mathcal{L} = ((p_1, u_1), \ldots, (p_N, u_N))$ is classically called a lottery [137]. When we speak of expected utility, we implicitly speak of the expected utility $EU(\mathcal{L})$ of a lottery $\mathcal{L}$.

A standard way to combine plausibilities and utilities is to use the probabilistic expected utility theory [137] defining $EU(\mathcal{L})$ as $\sum_{i\in[1,N]}(p_i \times u_i)$: it aggregates plausibilities and utilities using the combination operator $\otimes_{pu} = \times$ and synthesizes the aggregated information using the elimination operator $\oplus_u = +$. However, alternative definitions exist:

- If plausibilities are possibilities, then $EU(\mathcal{L}) = \min_{i\in[1,N]} \max(1-p_i, u_i)$ with the possibilistic *pessimistic* expected utility [43] (i.e. $\oplus_u = \min$ and $\otimes_{pu} : (p,u) \rightarrow \max(1-p, u)$) and $EU(\mathcal{L}) = \max_{i\in[1,N]} \min(p_i, u_i)$ with the possibilistic *optimistic* expected utility [43] (i.e. $\oplus_u = \max$ and $\otimes_{pu} = \min$).

- If plausibilities are κ-rankings and utilities are positive integers [59], then the expected utility of $\mathcal{L}$ is $EU(\mathcal{L}) = \min_{i\in[1,N]}(p_i + u_i)$ (i.e. $\oplus_u = \min$ and $\otimes_{pu} = +$).

To generalize these definitions of $EU(\mathcal{L})$, we start from Chu-Halpern's work on generalized expected utility [23, 24].

Chu-Halpern's structure Generalized expected utility is defined in an *expectation domain*, which is a tuple $(E_p, E_u, E'_u, \oplus_u, \otimes_{pu})$ such that: (1) E_p is a set of plausibility degrees and E_u is a set of utility degrees; (2) $\otimes_{pu} : E_p \times E_u \rightarrow E'_u$ combines plausibilities with utilities and satisfies $1_p \otimes_{pu} u = u$; (3) $\oplus_u : E'_u \times E'_u \rightarrow E'_u$ is a commutative and associative operator which can aggregate the information combined using $\otimes_{pu}$.

When a decision problem is *additive*, i.e. when, for all plausibility degrees p_1, p_2 associated with disjoint events, $(p_1 \oplus_p p_2) \otimes_{pu} u = (p_1 \otimes_{pu} u) \oplus_u (p_2 \otimes_{pu} u)$, the generic definition of the expected utility of a lottery is:

$$EU(\mathcal{L}) = \underset{i\in[1,N]}{\oplus_u} (p_i \otimes_{pu} u_i)$$

Classical expectation domains also satisfy additional properties such as "$\oplus_u$ is monotonic" and "$0_p \otimes_{pu} u = 0_u$, where 0_u is the identity of $\oplus_u$".

Adapting Chu-Halpern's structure for sequential decision making If we use $\otimes_{pu} : E_p \times E_u \rightarrow E'_u$ and $\oplus_u : E'_u \times E'_u \rightarrow E'_u$ to compute expected utilities at the first decision step, then we need to introduce operators $\otimes'_{pu} : E_p \times E'_u \rightarrow E''_u$ and $\oplus'_u : E''_u \times E''_u \rightarrow E''_u$ to compute expected utilities at the second decision step. In the end, if there are T decision steps, we must define T operators $\otimes_{pu}$ and T operators $\oplus_u$. In order to avoid the definition of an algebraic structure that would depend on the number of decision steps, we take $E_u = E'_u$ and work with only one operator $\otimes_{pu} : E_p \times E_u \rightarrow E_u$ and one operator $\oplus_u : E_u \times E_u \rightarrow E_u$.

As for plausibilities, and for the sake of the future algorithms, we restrict Chu-Halpern's expectation domains $(E_p, E_u, E_u, \oplus_u, \otimes_{pu})$ so that $\oplus_u$ and $\otimes_{pu}$ become closed and generalize properties of the initial $\oplus_u$ and $\otimes_{pu}$. However, this closure is not sufficient to deal with *sequential* decision making, because Chu-Halpern's expected utility is designed for *one-step* decision processes only. This is why we introduce three additional axioms for $\oplus_u$ and $\otimes_{pu}$:

- The first axiom is similar to a standard axiom for lotteries [137] defining compound lotteries. It states that if a lottery $\mathcal{L}_2$ involves a utility u with plausibility p_2, and if one of the utilities of a lottery $\mathcal{L}_1$ is the expected utility of $\mathcal{L}_2$ with plausibility p_1, then it is as if utility u had been obtained with plausibility $p_1 \otimes_p p_2$. This gives the axiom $p_1 \otimes_{pu} (p_2 \otimes_{pu} u) = (p_1 \otimes_p p_2) \otimes_{pu} u$.

- We further require that $\otimes_{pu}$ distributes over $\oplus_u$. To justify this point, assume that a lottery $\mathcal{L} = ((p_1, u_1), (p_2, u_2))$ is obtained with plausibility p. Two different versions of the contribution of $\mathcal{L}$ to the global utility degree can be derived: the first is $p \otimes_{pu} ((p_1 \otimes_{pu} u_1) \oplus_u (p_2 \otimes_{pu} u_2))$, and the second, which uses compound lotteries, is $((p \otimes_p p_1) \otimes_{pu} u_1) \oplus_u ((p \otimes_p p_2) \otimes_{pu} u_2)$. We want these two quantities to be equal for all p, p_1, p_2, u_1, u_2.

  This can be shown to be equivalent to the simpler property $p \otimes_{pu} (u_1 \oplus_u u_2) = (p \otimes_{pu} u_1) \oplus_u (p \otimes_{pu} u_2)$, i.e. to the distributivity of $\otimes_{pu}$ over $\oplus_u$.

- Finally, we assume that $\otimes_{pu}$ is right monotonic, i.e. $(u_1 \preceq_u u_2) \to (p \otimes_{pu} u_1 \preceq_u p \otimes_{pu} u_2)$. This means that if an agent prefers (strictly or not) an event ev_2 to another event ev_1, and if both events have the same plausibility degree p, then the contribution of ev_2 to the global expected utility degree must not be lesser than the contribution of ev_1.

These axioms define the notion of expected utility structure.

Definition 3.9. *Let $(E_p, \oplus_p, \otimes_p)$ be a plausibility structure and let $(E_u, \otimes_u)$ be a utility structure. $(E_p, E_u, \oplus_u, \otimes_{pu})$ is an* expected utility structure *iff*

- *$(E_u, \oplus_u, \otimes_{pu})$ is a semimodule on $(E_p, \oplus_p, \otimes_p)$ (cf. Definition 3.3 page 53),*
- *$\oplus_u$ is monotonic for $\preceq_u$ and $\otimes_{pu}$ is right monotonic for $\preceq_u$ ($(u_1 \preceq_u u_2) \to (p \otimes_{pu} u_1 \preceq_u p \otimes_{pu} u_2)$).*

3.6 Structures covered

Many structures considered in the literature are instances of expected utility structures, as shown in Proposition 3.10. The results presented in the remaining of the thesis hold not only for these usual expected utility structures, but more generally for all structures satisfying the axioms specified in Definitions 3.5, 3.8, and 3.9.

Proposition 3.10. *The structures in Table 3.1 are expected utility structures.*

It is possible to define more complex expected utility structures from existing ones. For example, from two expected utility structures $(E_p, E_u, \oplus_u, \otimes_{pu})$ and $(E'_p, E'_u, \oplus'_u, \otimes'_{pu})$, it is possible to build a compound expected utility structure $(E_p \times E'_p, E_u \times E'_u, \oplus''_u, \otimes''_{pu})$. This can be used to deal simultaneously with probabilistic and possibilistic expected utilities or more generally to deal with tuples of expected utilities.

The business dinner example To flesh out these definitions, we consider the following toy example, which will be referred to in the sequel. It does not correspond to a concrete real-life problem, but is used for its simplicity. *Peter invites John and Mary (a divorced couple) to a business dinner in order to convince them to invest in his company. Peter knows that if John is present at the end of the dinner, he will invest $10K€$. The same holds for Mary with $50K€$. Peter knows that John and Mary will not come together (one of them has to baby-sit their child), that*

	E_p	$\preceq_p$	$\oplus_p$	$\otimes_p$	$0_p, 1_p$	E_u	$\preceq_u$	$\otimes_u$	$\oplus_u$	$\otimes_{pu}$	$\bot_u, 0_u, 1_u$
1	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 1$	$\mathbb{R} \cup \{-\infty\}$	$\leq$	$+$	$+$	$\times$	$-\infty, 0, 0$
2	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 1$	$\mathbb{R}^+$	$\leq$	$\times$	$+$	$\times$	$0, 0, 1$
3	$[0, 1]$	$\leq$	max	min	$0, 1$	$[0, 1]$	$\leq$	min	max	min	$0, 0, 1$
4	$[0, 1]$	$\leq$	max	min	$0, 1$	$[0, 1]$	$\leq$	min	min	$\max(1{-}p, u)$	$0, 1, 1$
5	$\mathbb{N} \cup \{\infty\}$	$\geq$	min	$+$	$\infty, 0$	$\mathbb{N} \cup \{\infty\}$	$\geq$	$+$	min	$+$	$\infty, \infty, 0$
6	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\wedge$	$\vee$	$\wedge$	f, f, t
7	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\wedge$	$\wedge$	$\rightarrow$	f, t, t
8	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\vee$	$\wedge$	f, f, f
9	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	$\rightarrow$	f, t, f

Table 3.1: Expected utility structures for: 1. probabilistic expected utility with additive utilities (allows the probabilistic expected utility of a cost or a gain to be computed), 2. probabilistic expected utility with multiplicative utilities, also called probabilistic expected satisfaction (allows the probability of satisfaction of some constraints to be computed), 3. possibilistic optimistic expected utility, 4. possibilistic pessimistic expected utility, 5. qualitative utility with κ-rankings and with only positive utilities, 6. boolean optimistic expected utility with conjunctive utilities (allows one to know whether there exists a possible world in which all goals of a set of goals G are satisfied), 7. boolean pessimistic expected utility with conjunctive utilities (allows one to know whether in all possible worlds, all goals of a set of goals G are satisfied), 8. boolean optimistic expected utility with disjunctive utilities (allows one to know whether there exists a possible world in which at least one goal of a set of goals G is satisfied), 9. boolean pessimistic expected utility with disjunctive utilities (allows one to know whether in all possible worlds, at least one goal of a set of goals G is satisfied).

at least one of them will come, and that the case "John comes and Mary does not" occurs with a probability of 0.6*. As for the menu, Peter can order fish or meat for the main course, and white or red for the wine. However, the restaurant does not serve fish and red wine together. John does not like white wine and Mary does not like meat. If the menu does not suit them, they will leave the dinner. If John comes, Peter does not want him to leave the dinner because he is his best friend.*

Example 3.11. *The dinner problem uses the expected utility structure representing probabilistic expected additive utility (row 1 in Table 3.1): the plausibility structure is* $(\mathbb{R}^+, +, \times)$*,* $\oplus_u = +$*,* $\otimes_{pu} = \times$*, and utilities are additive gains:* $(E_u, \otimes_u) = (\mathbb{R} \cup \{-\infty\}, +)$*, with the convention that* $u + (-\infty) = -\infty$*.*

3.7 Relations with other existing structures

If we compare the structures defined with those defined in [54, 62, 23], we can observe that:

- The structures defined here are less general than Friedman-Chu-Halpern's, since additional axioms have been introduced. For example, plausibility structures are not able to model *belief functions* [125], which are not decomposable, whereas this is possible using Friedman-Halpern's plausibility measures (however, we are not aware of existing schemes for decision theory using belief functions directly; some proposals using the so-called "pignistic probability distribution" induced by a belief function together with the probabilistic expected utility exist [141], but they do not work directly on belief functions).

  Moreover, for one-step decision processes, Chu-Halpern's generalized expected utility is more general, since it assumes that $\otimes_{pu} : E_p \times E_u \to E'_u$ whereas we consider $\otimes_{pu} : E_p \times E_u \to E_u$.

- Conversely, the structures defined here can deal with multi-step decision processes whereas Chu-Halpern's generalized expected utility does not. Beyond this, other axioms such as the use of closed operators are essentially motivated by operational reasons. In fact, we use a slightly less expressive structure for the sake of future algorithms.

As a set E_p of plausibility degrees and a set E_u of utility degrees are defined, plausibilities and utilities must be *cardinal*. Purely *ordinal* approaches such as CP-nets [17], which, like Bayesian networks, exploit the notion of conditional independence to express a network of purely ordinal preference relations, are not covered.

As $\otimes_{pu}$ takes values in E_u, it is implicitly assumed that plausibilities and utilities are *commensurable*: works such as [48], describing a purely ordinal approach where qualitative preferences and plausibilities are not necessarily commensurable, are not captured either. Furthermore, some axioms entail that only *distributional plausibilities* are covered (the plausibility of a set of variable assignments is determined by the plausibilities of each covered complete assignment): Dempster-Shafer *belief functions* [125] are not encompassed. Finally, as only one partial order $\preceq_u$ on E_u is defined, it is assumed that the decision makers share the same preferences over utilities.

3.8 Summary

In this chapter, we have introduced *expected utility structures*, which are the first element of the PFU framework. They specify how plausibilities are combined and projected (using $\otimes_p$ and $\oplus_p$), how utilities are combined (using $\otimes_u$), and how plausibilities and utilities are aggregated to define generalized expected utility (using $\oplus_u$ and $\otimes_{pu}$). More precisely, the basic algebraic structures used are:

- a commutative semiring $(E_p, \oplus_p, \otimes_p)$ to handle plausibilities,
- a commutative monoid $(E_u, \otimes_u)$ to handle utilities,
- a semimodule $(E_p, E_u, \oplus_p, \otimes_{pu})$ to compute expected utilities.

The addition of monotonicity axioms on these classical structures leads to the notions of plausibility structure, utility structure, and expected utility structure respectively. These cover various existing plausibility/utility models and are inspired by Friedman-Chu-Halpern's plausibility measures and generalized expected utility. The main differences lie in the addition of axioms to deal with multi-step decision processes and in the use of closed operators motivated by operational reasons.

Chapter 4

Plausibility-Feasibility-Utility networks

The second element of the PFU framework is a network of scoped functions P_i, F_i, and U_i (cf. Equation 2.28 page 51) over a set of variables V. This network defines a compact and structured representation of the state of the environment, of the decisions, and of the global plausibilities, feasibilities, and utilities which hold over them. This chapter defines such networks and analyzes the relations between local functions and the global quantity they model, mainly based on conditional independence.

In the rest of the thesis, a *plausibility function* denotes a scoped function onto E_p (the set of plausibility degrees), a *feasibility function* is a scoped function onto $\{t, f\}$ (the set of feasibility degrees), and a *utility function* is a scoped function onto E_u (the set of utility degrees).

4.1 Decision and environment variables

In structured representations, decisions are represented using *decision variables*, which are controlled by one of the agents, and the state of the environment is represented by *environment variables*, which are not directly controlled by an agent. We use V_D to denote the set of decision variables and V_E to denote the set of environment variables. V_D and V_E form a partition of V.

Example 4.1. *The dinner problem can be modeled using six variables: bp_J and bp_M (value t or f), representing John's and Mary's presence at the beginning, ep_J and ep_M (value t or f), representing their presence at the end, mc (value fish or meat), representing the main course choice, and w (value white or red), representing the wine choice. Thus, we have $V_D = \{mc, w\}$ and $V_E = \{bp_J, bp_M, ep_J, ep_M\}$.*

4.2 Towards local plausibility and feasibility functions

Using combined local functions to represent a global one raises some considerations: how and when such local functions can be obtained from a global one, and conversely, when such local functions are directly used, which implicit assumptions are made on the global function.

We now show that all these questions boil down to the notion of conditional independence. In the following definitions and propositions, $(E_p, \oplus_p, \otimes_p)$ corresponds to a plausibility structure.

4.2.1 A first factorization step using conditional independence

Preliminaries: generalization of Bayesian networks results

Assume that one wants to express a global plausibility distribution $\mathcal{P}_S$ (cf. Definition 3.6 page 55) as a combination of local plausibility functions P_i. As work on Bayesian networks [96] has shown, the factorization of a joint distribution is essentially related to the notion of conditional independence. To introduce conditional independence, we first define *conditional plausibility distributions.*

Definition 4.2. *A plausibility distribution $\mathcal{P}_S$ over S is said to be* conditionable *iff there exists a set of functions denoted $\mathcal{P}_{S_1 \,|\, S_2}$ (one function for each pair S_1, S_2 of disjoint subsets of S) such that if S_1, S_2, S_3 are disjoint subsets of S, then*

(a) for all assignments A of S_2 such that $\mathcal{P}_{S_2}(A) \neq 0_p$, $\mathcal{P}_{S_1 \,|\, S_2}(A)$ is a plausibility distribution over S_1, [1]

(b) $\mathcal{P}_{S_1 \,|\, \emptyset} = \mathcal{P}_{S_1}$,

(c) $\oplus_{p_{S_1}} \mathcal{P}_{S_1,S_2 \,|\, S_3} = \mathcal{P}_{S_2 \,|\, S_3}$,

(d) $\mathcal{P}_{S_1,S_2 \,|\, S_3} = \mathcal{P}_{S_1 \,|\, S_2,S_3} \otimes_p \mathcal{P}_{S_2 \,|\, S_3}$,

(e) $(\mathcal{P}_{S_1,S_2,S_3} = \mathcal{P}_{S_1 \,|\, S_3} \otimes_p \mathcal{P}_{S_2 \,|\, S_3} \otimes_p \mathcal{P}_{S_3}) \rightarrow (\mathcal{P}_{S_1,S_2 \,|\, S_3} = \mathcal{P}_{S_1 \,|\, S_3} \otimes_p \mathcal{P}_{S_2 \,|\, S_3})$.

$\mathcal{P}_{S_1 \,|\, S_2}$ *is called the* conditional plausibility distribution of S_1 given S_2*.*

Condition (a) means that conditional plausibility distributions must be normalized. Condition (b) means that the information given by an empty set of variables does not change the plausibilities over the states of the environment. Condition (c) means that conditional plausibility distributions are consistent from the marginalization point of view. Condition (d) is the analog of the so-called chain rule with probabilities. Condition (e) is a kind of weak division axiom. [2]

Theorem 4.3 gives simple conditions on a plausibility structure, satisfied in all usual frameworks, that suffice for plausibility distributions to be conditionable.

Theorem 4.3. *If $(E_p, \oplus_p, \otimes_p)$ satisfies the axioms:*

- *if $p_1 \preceq_p p_2$ and $p_2 \neq 0_p$, then $\max\{p \in E_p \,|\, p_1 = p \otimes_p p_2\}$ exists and is $\preceq_p 1_p$,*
- *if $p_1 \prec_p p_2$, then there exists a unique $p \in E_p$ such that $p_1 = p \otimes_p p_2$,*
- *if $p_1 \prec_p p_2$, then there exists a unique $p \in E_p$ such that $p_2 = p \oplus_p p_1$,*

it is called a conditionable plausibility structure*, since all plausibility distributions are then conditionable: it suffices to define $\mathcal{P}_{S_1 \,|\, S_2}$ by $\mathcal{P}_{S_1 \,|\, S_2}(A) = \max\{p \in E_p \,|\, \mathcal{P}_{S_1,S_2}(A) = p \otimes_p \mathcal{P}_{S_2}(A)\}$ for all $A \in dom(S_1 \cup S_2)$ satisfying $\mathcal{P}_{S_2}(A) \neq 0_p$.*

[1] To avoid specifying that properties of $\mathcal{P}_{S_1 \,|\, S_2}$ hold only for assignments A of $S_1 \cup S_2$ satisfying $\mathcal{P}_{S_2}(A) \neq 0_p$, we use expressions such as "$\mathcal{P}_{S_1 \,|\, S_2} = L$" to denote "$\forall A \in dom(S_1 \cup S_2)$, $(\mathcal{P}_{S_2}(A) \neq 0_p) \rightarrow (\mathcal{P}_{S_1 \,|\, S_2}(A) = L(A))$".

[2] Compared to Friedman and Halpern's conditional plausibility measures [54, 62], (c) is the analog of axiom (Alg1), (d) is the analog of axiom (Alg2), (e) is the analog of axiom (Alg4), and axiom (Alg3) corresponds to the distributivity of $\otimes_p$ over $\oplus_p$.

The systematic definition of conditional plausibility distributions given in Theorem 4.3 fits with the usual definitions of conditional distributions, which are, with probabilities, "$\mathcal{P}_{S_1 \,|\, S_2}(A) = \mathcal{P}_{S_1,S_2}(A)/\mathcal{P}_{S_2}(A)$", with κ-rankings, "$\mathcal{P}_{S_1 \,|\, S_2}(A) = \mathcal{P}_{S_1,S_2}(A) - \mathcal{P}_{S_2}(A)$", and with possibility degrees combined using min, "$\mathcal{P}_{S_1 \,|\, S_2}(A) = \mathcal{P}_{S_1,S_2}(A)$ if $\mathcal{P}_{S_1,S_2}(A) < \mathcal{P}_{S_2}(A)$, 1 otherwise". In the following, every conditioning statement $\mathcal{P}_{S_1 \,|\, S_2}$ for conditionable plausibility structures will refer to the canonical notion of conditioning given in Proposition 4.3. Conditional independence can now be defined.

Definition 4.4. *Let $(E_p, \oplus_p, \otimes_p)$ be a conditionable plausibility structure. Let $\mathcal{P}_S$ be a plausibility distribution over S and S_1, S_2, S_3 be disjoint subsets of S. S_1 is said to be conditionally independent of S_2 given S_3, denoted $I(S_1, S_2 \,|\, S_3)$, iff $\mathcal{P}_{S_1,S_2 \,|\, S_3} = \mathcal{P}_{S_1 \,|\, S_3} \otimes_p \mathcal{P}_{S_2 \,|\, S_3}$.*

This means that S_1 is conditionally independent of S_2 given S_3 iff the problem can be split into one part depending on S_1 and S_3, and another part depending on S_2 and S_3.[3] This definition satisfies the usual properties of conditional independence, as proved by Proposition 4.5. These usual properties, known as the *semigraphoid axioms* [96], were shown to be the basis of the notion of information relevance in a wide variety of models.

Proposition 4.5. *$I(.,. \,|\, .)$ satisfies the semigraphoid axioms:*

1. *symmetry: $I(S_1, S_2 \,|\, S_3) \to I(S_2, S_1 \,|\, S_3)$,*
2. *decomposition: $I(S_1, S_2 \cup S_3 \,|\, S_4) \to I(S_1, S_2 \,|\, S_4)$,*
3. *weak union: $I(S_1, S_2 \cup S_3 \,|\, S_4) \to I(S_1, S_2 \,|\, S_3 \cup S_4)$,*
4. *contraction: $(I(S_1, S_2 \,|\, S_4) \wedge I(S_1, S_3 \,|\, S_2 \cup S_4)) \to I(S_1, S_2 \cup S_3 \,|\, S_4)$.*

Informally, the symmetry axiom states that if a set of variables S_1 does not provide any information about a set of variables S_2 given a third set of variables S_3, then S_2 gives no information about S_1 given S_3. The decomposition axiom asserts that if S_1 does not depend on both S_2 and S_3 given S_4, then S_1 does not depend on S_2 and S_3 considered independently. The weak union axiom states that if $S_2 \cup S_3$ is irrelevant to S_1 given S_4, then knowing S_3 does not change the irrelevance of S_2 with regard to S_1. Last, the contraction axiom tells that if S_3 is irrelevant to S_1 after knowing an irrelevant information about S_2, then S_3 must be irrelevant to S_1 before learning S_2.

Proposition 4.5 makes it possible to use Bayesian network techniques to express information in a compact way. With Bayesian networks, a DAG of variables is used to represent conditional independences between variables [96]. In some cases, such as image processing or statistical physics, it is more natural to express conditional independences between sets of variables. If probabilities are used, such situations can be modeled using *chain graphs* [55] presented in Chapter 2 page 38. In a chain graph, the DAG defined is not a DAG of variables, but a DAG of sets of variables, called components. Conditional probability distributions $P_{x \,|\, pa_G(x)}$ of variables are replaced by

[3] Definition 4.4 differs from Halpern's, which is "S_1 is conditionally independent (CI) of S_2 given S_3 iff $\mathcal{P}_{S_1 \,|\, S_2,S_3} = \mathcal{P}_{S_1 \,|\, S_3}$ and $\mathcal{P}_{S_2 \,|\, S_1,S_3} = \mathcal{P}_{S_2 \,|\, S_3}$". In [62], the definition we adopt is called non-interactivity (NI) and is shown to be weaker than CI. This implies that NI is satisfied more often and may lead to more factorizations. [62] gives a simple axiom (axiom (Alg4')) under which CI and NI are equivalent. Though this axiom holds in many usual frameworks, it does not hold with possibility degrees combined using min, a case covered by the PFU algebraic structure.

conditional probability distributions $P_{c\,|\,pa_G(c)}$ of components, each $P_{c\,|\,pa_G(c)}$ being expressed in a factored form $\varphi_1^c \times \varphi_2^c \times \ldots \times \varphi_{k_c}^c$.

We now formally introduce DAGs over sets of variables, called *DAGs of components*, and then use them to factor plausibility distributions.

Definition 4.6. *A DAG G is said to be a* DAG of components *over a set of variables S iff the vertices of G form a partition of S. $\mathcal{C}(G)$ denotes the set of components of G. For each $c \in \mathcal{C}(G)$, $pa_G(c)$ denotes the set of variables included in the parents of c in G, and $nd_G(c)$ denotes the set of variables included in the non-descendant components of c in G.*

Definition 4.7. *Let $(E_p, \oplus_p, \otimes_p)$ be a conditionable plausibility structure. Let $\mathcal{P}_S$ be a plausibility distribution over S and let G be a DAG of components over S. G is said to be* compatible *with $\mathcal{P}_S$ iff $I(c, nd_G(c) - pa_G(c)\,|\,pa_G(c))$ for all $c \in \mathcal{C}(G)$ (c is conditionally independent of its non-descendants given its parents).*

Theorem 4.8. *(Conditional independence and factorization) Let $(E_p, \oplus_p, \otimes_p)$ be a conditionable plausibility structure and let G be a DAG of components over S.*

(a) If G is compatible with a plausibility distribution $\mathcal{P}_S$ over S, then $\mathcal{P}_S = \otimes_{p_{c \in \mathcal{C}(G)}} \mathcal{P}_{c\,|\,pa_G(c)}$.

(b) If, for all $c \in \mathcal{C}(G)$, there is a function $L_{c,pa_G(c)}$ such that $L_{c,pa_G(c)}(A)$ is a plausibility distribution over c for all assignments A of $pa_G(c)$, then $\gamma_S = \otimes_{p_{c \in \mathcal{C}(G)}} L_{c,pa_G(c)}$ is a plausibility distribution over S with which G is compatible.

Theorem 4.8 links conditional independence and factorization. Theorem 4.8(a) is a generalization of the usual result of Bayesian networks [96] which says that if a DAG of variables is compatible with a probability distribution P_S, then P_S can be factored as $P_S = \prod_{x \in S} P_{x\,|\,pa_G(x)}$. Theorem 4.8(b) is a generalization of the standard result of Bayesian networks [96] which says that, given a DAG G of variables in S, if conditional probabilities $P_{x\,|\,pa_G(x)}$ are defined for each variable $x \in S$, then $\prod_{x \in S} P_{x\,|\,pa_G(x)}$ defines a probability distribution over S with which G is compatible. Both results are generalizations since they hold for arbitrary plausibility distributions (and not for probability distributions only).

Theorem 4.8(a) entails that in order to factor a global plausibility distribution $\mathcal{P}_S$, it suffices to define a DAG of components compatible with it, i.e. to express conditional independences. To define such a DAG, the following systematic procedure can be used. The initial DAG of components is an empty DAG G. While $\mathcal{C}(G) = \{c_1, \ldots, c_{k-1}\}$ is not a partition of S, do:

1. Let $S_k = c_1 \cup \ldots \cup c_{k-1}$; choose a subset c_k of the set $S - S_k$ of variables not already considered by following two rules:

   (R1) *Consider causes before effects*: in the dinner problem, this suggests not putting ep_J in c_k if its causes bp_J and w are not in S_k.

   (R2) *Gather in a component variables that are correlated even when all variables in S_k are assigned*: bp_J and bp_M are correlated and (R1) does not apply. Indeed, we cannot say that bp_J has a causal influence on bp_M, or that bp_M has a causal influence on bp_J, since it is not specified whether Mary or John chooses first if (s)he baby-sits. bp_J and bp_M could also be correlated via an unmodeled common cause such as a coin toss that determines the baby-sitter. Hence, bp_J and bp_M can be put in the same component $c = \{bp_J, bp_M\}$.[4]

2. Add c_k as a component to G and find a minimal subset pa_k of S_k such that $I(c_k, S_k - pa_k \mid pa_k)$. Add edges directed from components containing at least one variable in pa_k to c_k.

The resulting DAG of components is guaranteed to be compatible with $\mathcal{P}_S$, which implies, using Theorem 4.8(a), that the local functions P_i representing $\mathcal{P}_S$ can simply be defined as the functions in the set $\{\mathcal{P}_{c \mid pa_G(c)}, c \in \mathcal{C}(G)\}$. We say that (R1) and (R2) build a DAG respecting causality. They must be seen just as *possible mechanisms* that help in identifying conditional independences.

All the previous results extending Bayesian networks results to plausibility distributions also apply to feasibilities. Indeed, the feasibility structure $S_f = (\{t, f\}, \vee, \wedge)$ is a particular case of a conditionable plausibility structure, since it satisfies the axioms of Theorem 4.3. We may therefore speak of conditional feasibility distribution. If S is a set of decision variables, the construction of a DAG compatible with a feasibility distribution $\mathcal{F}_S$ leads to the factorization $\mathcal{F}_S = \wedge_{c \in \mathcal{C}(G)} \mathcal{F}_{c \mid pa_G(c)}$.

Taking the different types of variables into account

In general, the situation is a bit more complex because variables may be either decision or environment variables. In this case, we cannot simply deal with a plausibility or a feasibility distribution over all variables. We must express a plausibility distribution over the set of environment variables V_E, but decision variables can influence the environment (for example, the health state of a patient depends on the treatment chosen for him by a doctor). This means that we want to express a family of plausibility distributions over V_E (one for each assignment of V_D) rather than only one plausibility distribution over V_E. To make this clear, we define *controlled plausibility distributions*.

Definition 4.9. *A plausibility distribution over V_E controlled by V_D, denoted $\mathcal{P}_{V_E \,||\, V_D}$, is a function $dom(V_E \cup V_D) \to E_p$, such that for all assignments A_D of V_D, $\mathcal{P}_{V_E \,||\, V_D}(A_D)$ is a plausibility distribution over V_E. $\mathcal{P}_{V_E \,||\, V_D}$ is called a* controlled plausibility distribution.

As for feasibilities, we want to express a feasibility distribution over the set of decision variables V_D, but environment variables can constrain the possible decisions (for example, if a blackout occurs, an agent cannot switch on the light). Thus, we want to express a family of feasibility distributions over V_D (one for each assignment of V_E) rather than only one feasibility distribution over V_D. In other words, we want to express a controlled feasibility distribution $\mathcal{F}_{V_D \,||\, V_E}$.

In order to directly reuse the previous theorems for controlled distributions, we introduce the notion of the completion of a controlled distribution. This allows us to extend a distribution to the full set of variables V by assigning the same plausibility (resp. feasibility) degree to every assignment of V_D (resp. V_E).

Proposition 4.10. *Let $(E_p, \oplus_p, \otimes_p)$ be a conditionable plausibility structure. Then, for all $n \in \mathbb{N}^*$, there exists a unique p_0 such that $\oplus_{p_{i \in [1,n]}} p_0 = 1_p$.*

[4] Components such as $\{bp_J, bp_M\}$ could be broken by assuming for example that bp_M causally influences bp_J, i.e. that Mary chooses if she baby-sits first. We can (and prefer to) keep the component as $\{bp_J, bp_M\}$ because in general, "breaking" components can increase the scopes of the functions involved. For example, assume that one wants to model plausibilities over variables representing colors of pixels of an $N \times N$ image such that the color of a pixel probabilistically depends on the colors of its 4 neighbors only. With a component approach, results of Markov random fields [22] show that the local functions obtained have scopes of size 5 only, whereas with a component-breaking mechanism, the size of the largest scope is linear in N.

Definition 4.11. *Let $(E_p, \oplus_p, \otimes_p)$ be a conditionable plausibility structure and let $\mathcal{P}_{V_E \,||\, V_D}$ be a controlled plausibility distribution. Then, the* completion *of $\mathcal{P}_{V_E \,||\, V_D}$ is a function denoted $\mathcal{P}_{V_E, V_D}$ and defined by $\mathcal{P}_{V_E, V_D} = \mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0$, where p_0 is the unique element of E_p such that $\oplus_{p\, i \in [1, |dom(V_D)|]}\, p_0 = 1_p$.*

In other words, $\mathcal{P}_{V_E, V_D}$ is defined from $\mathcal{P}_{V_E \,||\, V_D}$ by assigning the same plausibility degree p_0 to all assignments of V_D. In the case of probability theory, it corresponds to saying that all assignments of V_D are equiprobable.

Proposition 4.12. *Let $\mathcal{P}_{V_E, V_D}$ be the completion of a controlled plausibility distribution $\mathcal{P}_{V_E \,||\, V_D}$. Then, $\mathcal{P}_{V_E, V_D}$ is a plausibility distribution over $V_E \cup V_D$ and $\mathcal{P}_{V_E \,|\, V_D} = \mathcal{P}_{V_E \,||\, V_D}$.*

As a result, we use $\mathcal{P}_{V_E \,|\, V_D}$ to denote $\mathcal{P}_{V_E \,||\, V_D}$ (and this is equivalent). Similarly, it is possible to complete a controlled feasibility distribution $\mathcal{F}_{V_D \,||\, V_E}$. Proposition 4.14 below, entailed by Theorem 4.8(a), shows how to obtain a first factorization of $\mathcal{P}_{V_E \,|\, V_D}$ and $\mathcal{F}_{V_D \,|\, V_E}$.

Definition 4.13. *A DAG G is a* typed DAG of components over $V_E \cup V_D$ *iff the vertices of G form a partition of $V_E \cup V_D$ such that each element of this partition is a subset of either V_D or V_E. Each vertex of G is called a* component*. The set of components contained V_E (environment components) is denoted $\mathcal{C}_E(G)$ and the set of components included in V_D (decision components) is denoted $\mathcal{C}_D(G)$.*

Proposition 4.14. *Let G be a typed DAG of components over $V_E \cup V_D$. Let G_p be the partial graph of G induced by the arcs of G incident to environment components. Let G_f be the partial graph of G induced by the arcs of G incident to decision components.*

If G_p is compatible with the completion of $\mathcal{P}_{V_E \,||\, V_D}$ (cf. Definition 4.7) and G_f is compatible with the completion of $\mathcal{F}_{V_D \,||\, V_E}$, then

$$\mathcal{P}_{V_E \,|\, V_D} = \underset{c \in \mathcal{C}_E(G)}{\otimes_p} \mathcal{P}_{c \,|\, pa_G(c)} \text{ and } \mathcal{F}_{V_D \,|\, V_E} = \bigwedge_{c \in \mathcal{C}_D(G)} \mathcal{F}_{c \,|\, pa_G(c)}.$$

This allows us to specify local P_i and F_i functions: it suffices to express each $\mathcal{P}_{c \,|\, pa_G(c)}$ and each $\mathcal{F}_{c \,|\, pa_G(c)}$ to express $\mathcal{P}_{V_E \,|\, V_D}$ and $\mathcal{F}_{V_D \,|\, V_E}$ in a compact way. In fact, we could have defined two DAGs, one for the factorization of $\mathcal{P}_{V_E \,|\, V_D}$ and the other for the factorization of $\mathcal{F}_{V_D \,|\, V_E}$, but these two DAGs can actually always be merged as soon as one makes the (undemanding) assumption that it is impossible, given $x \in V_D$ and $y \in V_E$, that both x influences y, and y constrains the possible decision values for x. This assumption ensures that the union of the two DAGs does not create cycles. We use just one DAG for simplicity.

Example 4.15. *Consider the dinner problem to illustrate the first factorization step. One way to obtain G is to use the causality-based reasoning described after Theorem 4.8. We start with an empty DAG. As ep_J and ep_M are both effects of other variables, they are not considered in the first component c_1. bp_J can be chosen as a variable to add to c_1, because bp_J is not necessarily an effect of another variable. As previously explained, bp_J can be a cause of bp_M or an effect of bp_M, or bp_J may be correlated with bp_M via an unmodeled cause. As a result, we get $c_1 = \{bp_J, bp_M\}$ as a first component. c_1 gets no parent because it is the first created component.*

Then, as ep_J and ep_M are effects of w or mc, we do not consider ep_J or ep_M in the second component c_2. Since w is not necessarily an effect of mc, one can add w to c_2. The dinner problem *specifies that ordering fish and red wine simultaneously is not feasible, but we do not know whether the wine is chosen before or after the main course, i.e. w can be a cause or an effect of mc. As a result, we take $c_2 = \{mc, w\}$. As the menu choice is independent from who is present at the beginning, c_2 has no parent in the temporary DAG.*

As ep_J is a direct effect of bp_J and w only (John leaves the dinner if white wine is chosen), we can add ep_J to a third component c_3. Moreover, ep_J is not correlated with ep_M when $c_1 \cup c_2 = \{bp_J, bp_M, mc, w\}$ is assigned. Hence, we take $c_3 = \{ep_J\}$. Given that ep_J depends both on bp_J and w, c_3 gets $\{bp_J, bp_M\}$ and $\{mc, w\}$ as parents. Finally, $c_4 = \{ep_M\}$, and as ep_M is independent of other variables given bp_M and mc (because Mary leaves iff meat is chosen), we have that $I(\{ep_M\}, \{ep_J, bp_J, w\} \,|\, \{bp_M, mc\})$. This entails that $c_4 = \{ep_M\}$ is added to the DAG with $\{bp_J, bp_M\}$ and $\{mc, w\}$ as parents. Therefore, we get $\mathcal{C}_D(G) = \{\{mc, w\}\}$ as the set of decision components and $\mathcal{C}_E(G) = \{\{bp_J, bp_M\}, \{ep_J\}, \{ep_M\}\}$ as the set of environment components. The DAG of components is shown in Figure 4.1(a) page 71.

Proposition 4.14 ensures that the joint probability and feasibility distributions factor as $\mathcal{P}_{V_E \,|\, V_D} = \mathcal{P}_{bp_J, bp_M} \times \mathcal{P}_{ep_J \,|\, bp_J, bp_M, mc, w} \times \mathcal{P}_{ep_M \,|\, bp_J, bp_M, mc, w}$ and $\mathcal{F}_{V_D \,|\, V_E} = \mathcal{F}_{mc, w}$ respectively.

4.2.2 Further factorization steps

Proposition 4.14 provides us with a decomposition of $\mathcal{P}_{V_E \,|\, V_D}$ and $\mathcal{F}_{V_D \,|\, V_E}$ based on the conditional independence relation $I(.,.\,|\,.)$ of Definition 4.4. It may be possible to perform further factorization steps by factoring each $\mathcal{P}_{c \,|\, pa_G(c)}$ as a set of local plausibility functions P_i and factoring each $\mathcal{F}_{c \,|\, pa_G(c)}$ as a set of local feasibility functions F_i.

- In some cases, expressing factors of $\mathcal{P}_{c \,|\, pa_G(c)}$ or $\mathcal{F}_{c \,|\, pa_G(c)}$ is quite natural. For example, if $\otimes_p = \wedge$, if variables in an environment component $c = \{x_{i,j} \,|\, i, j \in [1, n]\}$ without parents represent pixel colors, and if one wants to model in $\mathcal{P}_c$ that adjacent pixels have different colors, it is natural to define a set of binary difference constraints $\delta_{x_{i,j}, x_{k,l}}$ and to factor $\mathcal{P}_c$ as $\mathcal{P}_c = \left(\wedge_{(i,j) \in [1,n-1] \times [1,n]} \delta_{x_{i,j}, x_{i+1,j}}\right) \wedge \left(\wedge_{(i,j) \in [1,n] \times [1,n-1]} \delta_{x_{i,j}, x_{i,j+1}}\right)$. Such a decomposition cannot be obtained based only on the conditional independence relation $I(.,.\,|\,.)$ of Definition 4.4.

- In some settings, as in Markov random fields [22], systematic techniques exist to obtain such factorizations. For Bayesian networks, systematic techniques also exist: with hybrid networks [36], we can extract the deterministic information contained in a conditional probability distribution $P_{x \,|\, pa_G(x)}$ by expressing it as $P_{x \,|\, pa_G(x)} = P_{x \,|\, pa_G(x)} \times \Gamma$, where Γ is the 0-1 function defined by $\Gamma(A) = 0$ iff $P_{x \,|\, pa_G(x)}(A) = 0$. Thus, a conditional probability distribution can be specified by several functions. Adding such redundant deterministic information, with a possible smallest arity, generally improves algorithmic efficiency.

- One may use another weaker definition of conditional independence: in valuation-based systems [129], S_1 and S_2 are said to be conditionally independent given S_3 with regard to a function γ_{S_1, S_2, S_3} if this function factors into two scoped functions with scopes $S_1 \cup S_3$ and $S_2 \cup S_3$. This definition is not used for the first factorization step because it destroys the normalization conditions which may be useful from a computational point of view.

These additional factorization steps are of interest because decreasing the size of the scopes of the functions involved or adding redundant information in the problem can be computationally useful.

For every environment component c, if "$P_i \in Fact(c)$" stands for "P_i is a factor of $\mathcal{P}_{c\,|\,pa_G(c)}$", the second factorization gives us

$$\mathcal{P}_{c\,|\,pa_G(c)} = \underset{P_i \in Fact(c)}{\otimes_p} P_i$$

Given that $\oplus_{p_c} \mathcal{P}_{c\,|\,pa_G(c)} = 1_p$, the P_i functions in $Fact(c)$ satisfy the normalization condition $\oplus_{p_c} \left(\otimes_{p\,P_i \in Fact(c)} P_i \right) = 1_p$. Their scopes $sc(P_i)$ are naturally contained in $sc(\mathcal{P}_{c\,|\,pa_G(c)}) = c \cup pa_G(c)$.

For every decision component c, if "$F_i \in Fact(c)$" stands for "F_i is a factor of $\mathcal{F}_{c\,|\,pa_G(c)}$", the second factorization gives us

$$\mathcal{F}_{c\,|\,pa_G(c)} = \underset{F_i \in Fact(c)}{\wedge} F_i$$

Given that $\vee_c \mathcal{F}_{c\,|\,pa_G(c)} = t$, the F_i functions in $Fact(c)$ satisfy the normalization condition $\vee_c \left(\wedge_{F_i \in Fact(c)} F_i \right) = t$. Moreover, $sc(F_i) \subset c \cup pa_G(c)$.

Other factorizations, which do not decrease the scopes of the functions involved, could also be exploited. Indeed, each scoped function P_i or F_i can itself have an internal *local structure*, as for instance when P_i is a noisy-OR gate [96] in a Bayesian network, or in presence of context-specific independence [20]. Such internal local structures can be made explicit by representing functions with tools such as Algebraic Decision Diagrams [113].

Example 4.16. *$\mathcal{P}_{bp_J,bp_M}$ can be expressed in terms of a first plausibility function P_1 specifying the probability of John and Mary being present at the beginning. P_1 is defined by $P_1((bp_J,t).(bp_M,f)) = 0.6$, $P_1((bp_J,f).(bp_M,t)) = 0.4$, and $P_1((bp_J,t).(bp_M,t)) = P_1((bp_J,f).(bp_M,f)) = 0$. One can also add redundant deterministic information with a second plausibility function P_2 defined as the constraint $bp_J \neq bp_M$ ($P_2(A) = 1$ if the constraint is satisfied, 0 otherwise). We get $\mathcal{P}_{bp_J,bp_M} = P_1 \otimes_p P_2$ and $Fact(\{bp_J, bp_M\}) = \{P_1, P_2\}$.*

$\mathcal{P}_{ep_J\,|\,bp_J,bp_M,mc,w}$ can be specified as a combination of two plausibility functions P_3 and P_4. P_3 expresses that if John is absent at the beginning, he is absent at the end: P_3 is the hard constraint $(bp_J = f) \rightarrow (ep_J = f)$ ($P_3(A) = 1$ if the constraint is satisfied, 0 otherwise). Then, $P_4 : (bp_J = t) \rightarrow ((ep_J = t) \leftrightarrow (w \neq white))$ is a hard constraint specifying that John leaves iff white wine is chosen. Hence, we have $\mathcal{P}_{ep_J\,|\,bp_J,bp_M,mc,w} = P_3 \otimes_p P_4$ and $Fact(\{ep_J\}) = \{P_3, P_4\}$. Similarly, $\mathcal{P}_{ep_M\,|\,bp_J,bp_M,mc,w} = P_5 \otimes_p P_6$, with P_5, P_6 defined as constraints, and $Fact(\{ep_M\}) = \{P_5, P_6\}$.

As for feasibilities, $\mathcal{F}_{mc,w}$ can be specified by a feasibility function F_1 expressing that ordering fish with red wine is not allowed: $F_1 : \neg((mc = fish) \wedge (w = red))$ and $Fact(\{mc, w\}) = \{F_1\}$. The association of local functions with components appears in Figure 4.1(a).

4.3 Local utilities

Local utilities can be defined over the states of the environment only (as in the utility of the health state of a patient), over decisions only (as in the utility of the decision of buying a car or not), or over the states of the environment and decisions (as in the utility of the result of a horse race and a bet on the race).

In order to specify local utilities, one standard approach, used in CSPs and influence diagrams, is to directly define a set U of local utility functions, modeling preferences or hard requirements, over decision and environment variables. This set implicitly defines a global utility $\mathcal{U}_V = \otimes_{u U_i \in U} U_i$ over all variables. If this factored form is obtained from a global joint utility, one may rely, when $\otimes_u = +$, on the work of [50, 3], which introduces a notion of conditional independence for utilities.

No normalization condition is imposed on local utilities, which can always be combined without generating any impossibility (their combination can only generate *unacceptability*).

Example 4.17. *In the dinner problem, three local utility functions can be defined. A binary utility function U_1 expresses that Peter does not want John to leave the dinner: U_1 is the hard constraint $(bp_J = t) \rightarrow (ep_J = t)$ ($U_1(A) = 0$ if the constraint is satisfied, $-\infty$ otherwise). Two unary utility functions U_2 and U_3 over ep_J and ep_M respectively express the gains expected from the presences at the end: $U_2((ep_J, t)) = 10$ and $U_2((ep_J, f)) = 0$ (John invests $10K€$ if he is present at the end), while $U_3((ep_M, t)) = 50$ and $U_3((ep_M, f)) = 0$ (Mary invests $50K€$ if she is present at the end). U_2 and U_3 can be viewed as soft constraints. All the local functions are represented in a composite graphical model in Figure 4.1(b).*

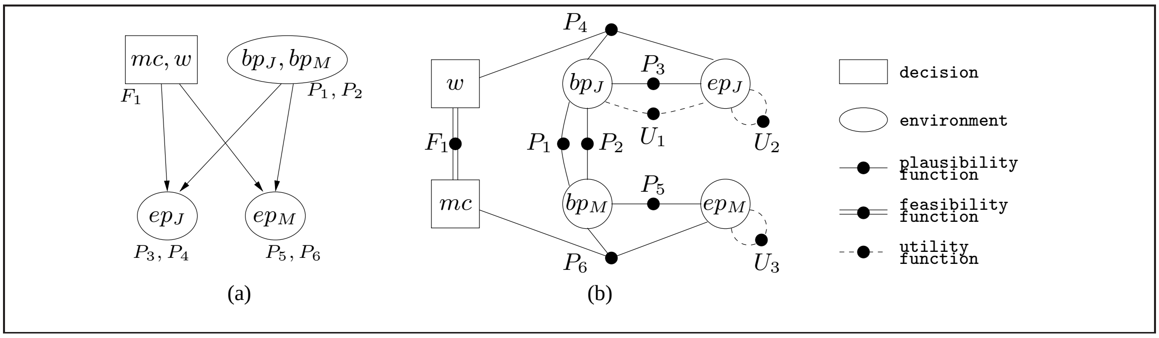


Figure 4.1: (a) DAG of components (b) Network of scoped functions.

4.4 Formal definition of PFU networks

We can now formally define Plausibility-Feasibility-Utility networks. The definition is justified by the previous construction process, but it holds even if the plausibility structure is not conditionable.

Definition 4.18. *A* Plausibility-Feasibility-Utility network *on an expected utility structure is a tuple $\mathcal{N} = (V, G, P, F, U)$ such that:*

- *$V = \{x_1, x_2, \ldots\}$ is a finite set of finite domain* variables. *V is partitioned into V_D (decision variables) and V_E (environment variables).*
- *G is a typed DAG of components over $V_E \cup V_D$ (cf. Definition 4.6).*
- *$P = \{P_1, P_2, \ldots\}$ is a finite set of* plausibility functions. *Each $P_i \in P$ is associated with a unique component $c \in \mathcal{C}_E(G)$ such that $sc(P_i) \subset c \cup pa_G(c)$. The set of $P_i \in P$ associated with a component $c \in \mathcal{C}_E(G)$ is denoted $Fact(c)$ and must satisfy $\oplus_{p\, c} \left(\otimes_{p P_i \in Fact(c)} P_i\right) = 1_p$.*

- $F = \{F_1, F_2, \ldots\}$ *is a finite set of* feasibility functions*. Each function F_i is associated with a unique component $c \in \mathcal{C}_D(G)$ such that $sc(F_i) \subset c \cup pa_G(c)$. The set of $F_i \in F$ associated with a component $c \in \mathcal{C}_D(G)$ is denoted $Fact(c)$ and must satisfy $\underset{c}{\vee}\left(\wedge_{F_i \in Fact(c)} F_i\right) = t$.*

- $U = \{U_1, U_2, \ldots\}$ *is a finite set of* utility functions*.*

4.5 From PFU networks to global functions

We have seen how to obtain a PFU network expressing a global controlled plausibility distribution $\mathcal{P}_{V_E \,||\, V_D}$, a global controlled feasibility distribution $\mathcal{F}_{V_D \,||\, V_E}$, and a global utility $\mathcal{U}_V$.

Conversely, let $\mathcal{N} = (V, G, P, F, U)$ be a PFU network, i.e. a set of variables, a typed DAG of components, and sets of scoped functions. Then

- the global function $\Psi = \otimes_{p\, P_i \in P} P_i$ is a controlled plausibility distribution of V_E given V_D. Moreover, by Theorem 4.8(b), if the plausibility structure is conditionable and if G_p is the partial DAG of G induced by the arcs incident to environment components, then G_p is compatible with the completion of Ψ;

- the global function $\Phi = \wedge_{F_i \in F} F_i$ is a controlled feasibility distribution of V_D given V_E. Moreover, by Theorem 4.8(b), if G_f is the partial DAG of G induced by the arcs of G incident to decision components, then G_f is compatible with the completion of Φ;

- $\mu = \otimes_{u\, U_i \in U} U_i$ is necessarily a global utility.

We can therefore denote Ψ by $\mathcal{P}_{V_E \,||\, V_D}$, Φ by $\mathcal{F}_{V_D \,||\, V_E}$, and μ by $\mathcal{U}_V$.

4.6 Back to existing frameworks

Let us consider some formalisms described in Chapter 2. A CSP (hard or soft) can easily be represented as a PFU network $\mathcal{N} = (V, G, \emptyset, \emptyset, U)$: all variables in V are decision variables, G is reduced to a single decision component containing all variables, and constraints are represented by utility functions. Using feasibility functions to represent constraints, it would be impossible to represent inconsistent networks because of the normalization conditions on feasibilities. SAT is modeled similarly; the only difference is that constraints are replaced by clauses.

The same PFU network is used to represent the local functions of a quantified boolean formula or of a quantified CSP. The differences with CSP or SAT appear when we consider queries on the network (see next chapter).

A Bayesian network can be modeled as $\mathcal{N} = (V, G, P, \emptyset, \emptyset)$: all variables in V are environment variables, G is the DAG of the BN, and $P = \{P_{x \,|\, pa_G(x)}, x \in V\}$. There is no feasibility or utility function. A chain graph is also modeled as $\mathcal{N} = (V, G, P, \emptyset, \emptyset)$, with G the DAG of components of the chain graph and P the set of factors of each $P_{c \,|\, pa_G(c)}$.

A stochastic CSP is represented by a PFU network $\mathcal{N} = (V, G, P, \emptyset, U)$, where V is partitioned into V_D, the set of decision variables, and V_E, the set of stochastic variables, G is a DAG which depends on the relations between the stochastic variables, P is the set of probability distributions over the stochastic variables, and U is the set of constraints.

An influence diagram can be modeled by $\mathcal{N} = (V, G, P, \emptyset, U)$ such that V_D contains the decision variables, V_E contains the chance variables, G is the DAG of the influence diagram without the utility nodes and with arcs into random variables only (i.e. we keep only the so-called *influence* arcs), and $P = \{P_{x \mid pa_G(x)}, x \in V_E\}$. There are no feasibilities, and one utility function U_i is defined per utility variable u, the scope of U_i being $pa_G(u)$. To represent valuation networks, a set F of feasibility functions is added. Note that the business dinner example could not have been modeled using a standard influence diagram, since influence diagrams cannot deal with feasibilities (suitable extensions exist however [130]).

A finite horizon probabilistic MDP can be modeled as $\mathcal{N} = (V, G, P, \emptyset, U)$. If there are T time-steps, then $V_D = \{d_t, t \in [1, T]\} \cup \{s_1\}$ and $V_E = \{s_t, t \in [2, T]\}$;[5] G is a DAG of components such that (a) each component contains one variable, (b) decision components have no parents, and (c) the parents of an environment component $\{s_{t+1}\}$ are $\{s_t\}$ and $\{d_t\}$; $P = \{P_{s_{t+1}|s_t,d_t}, t \in [1, T-1]\}$ and $U = \{U_{s_t,d_t}, t \in [1, T]\}$. Modeling a finite horizon possibilistic MDP is similar.

4.7 Summary

In this chapter, we have introduced the second element of the PFU framework: a network of variables linked by local plausibility, feasibility, and utility functions, with a DAG capturing normalization conditions. The factorization of global plausibilities, feasibilities, and utilities into scoped functions has been linked to conditional independence. This provides us with a constructive method to specify local functions representing a given global function. From a pure technical point of view, the definition of PFU networks (Definition 4.18) is quite simple.

[5] As there is no plausibility distribution over the initial state s_1, s_1 is not considered as an environment variable. This corresponds to the special case where decision variables model problem parameters.

Chapter 5

Queries on a PFU network

A query corresponds to a reasoning task on the information expressed by a PFU network. Examples of informal queries about the dinner problem are

1. "What is the best menu choice if Peter does not know who is present at the beginning?"

2. "What is the best menu choice if Peter knows who is present at the beginning?"

3. "How should we maximize the expected investment if the restaurant chooses the main course first and Peter is pessimistic about this choice, then the presences at the beginning are observed, and last Peter chooses the wine?"

Dissociating PFU networks from queries is consistent with the trend in the influence diagram community to relax the so-called *information links*, as in Unconstrained Influence Diagrams [68] or Limited Memory Influence Diagrams [81]: it explicitly figures that queries do not change the local relations between variables.

In this chapter, we define a simple class of queries on PFU networks. We assume that a *sequence of decisions* must be performed, and that the order in which decisions and observations are made is known. We also make a *no-forgetting* assumption, that is, when making a decision, an agent is aware of all previous decisions and observations. From now on, the set of utility degrees E_u is assumed to be *totally* ordered. Actually, in the context of a systematic computation and execution of a sequence of decisions, this total order assumption, which holds in various usual frameworks, allows one to always identify optimal decision rules. See Section 5.7 for a discussion of how to extend the results to a partial order.

Two definitions of the answer to a query are given, the first being based on decision trees, and the second being more operational. An equivalence between these two definitions is then established.

5.1 Query definition

In order to formulate reasoning tasks on a PFU network, we use a sequence *Sov* of operator-variable(s) pairs. This sequence captures different aspects of the query:

- *Partial observabilities*: *Sov* specifies the order in which decisions are made and environment variables are observed. If $x \in V_E$ appears to the left of $y \in V_D$ (for example $Sov = \ldots(\oplus_u, \{x\})\ldots(\max, \{y\})\ldots$), this means that the value of x is known (observed) when a value for y is chosen. Conversely, if $Sov = \ldots(\max, \{y\})\ldots(\oplus_u, \{x\})\ldots$, x is not observed when choosing y.

- *Optimistic/pessimistic attitude* concerning the decision makers: $(\max, \{y\})$ is inserted in the elimination sequence if one is optimistic about the behavior of the agent controlling a decision variable y, and $(\min, \{y\})$ if one is pessimistic. The operator used for environment variables will always be $\oplus_u$, to model that expected utilities are sought.

- *Parameters of the decision making problem*: if one wants to compute optimal expected utilities or optimal policies without assigning a subset S of the decision variables, then variables in S do not appear in *Sov*.

Example 5.1. *The sequence corresponding to the informal query: "How should we maximize the expected investment if the restaurant chooses the main course first and Peter is pessimistic about this choice, then the presences at the beginning of the dinner are observed, and last Peter chooses the wine before knowing who is present at the end?" is*

$$Sov = (\min, \{mc\}).(\oplus_u, \{bp_J, bp_M\}).(\max, \{w\}).(\oplus_u, \{ep_J, ep_M\})$$

*This sequence models that: (1) Peter is pessimistic about the main course (*min *over mc), which is chosen without observing any variable (no variable to the left of mc in Sov); (2) Peter chooses the wine for the best (*max *over w) after the main course has been chosen and after knowing who is present at the beginning (w appears to the right of mc, bp_J, and bp_M in Sov), but before knowing who is present at the end (w appears to the left of ep_J, ep_M). Specifically, bp_J and bp_M are partially observable, whereas ep_J and ep_M are unobservable.*

Definition 5.2. *A query on a PFU network is a pair $Q = (Sov, \mathcal{N})$ where $\mathcal{N}$ is a PFU network and $Sov = (op_1, S_1) \cdot (op_2, S_2) \cdots (op_k, S_k)$ is a sequence of* operator-variable(s) *pairs such that*

(1) all the S_i are disjoint;

(2) either "$S_i \subset V_D$ and $op_i = \min$ or $\max$", or "$S_i \subset V_E$ and $op_i = \oplus_u$";

(3) variables not involved in any of the S_i, called free variables*, are decision variables;*

(4) for all variables x, y of different types (one is a decision variable, the other is an environment variable), if there is a directed path from the component which contains x to the component which contains y in the DAG of the PFU network $\mathcal{N}$, then x does not appear to the right of y in Sov, i.e. either x appears to the left of y, or x is a free variable.

Condition (1) ensures that each variable is eliminated at most once. Condition (2) means that optimal decisions are sought for decision variables, whereas expected utilities are sought for environment variables. Condition (3) means that variables which are not eliminated in *Sov* act as problem parameters and are viewed as decision variables. Condition (4) means that if x

and y are of different types and x is an ancestor of y, then x is assigned before y. This ensures that causality is respected for variables of different types: for the dinner problem example, $((\oplus_u, \{bp_J, bp_M, ep_J, ep_M\}).(\max, \{mc, w\}), \mathcal{N})$, which violates condition (4), violates causality since the menu cannot be chosen after knowing who is present at the end.

Variables appearing in Sov are called *quantified variables*, by analogy with quantified boolean formulas. The set of free variables is denoted by V_{fr}. Note that the definition of queries does not prevent an environment variable from being "quantified" by min or max, because we may have $\oplus_u = \min$ or $\oplus_u = \max$.

For all $i \in [1, k]$, we define the set of variables appearing in V_{fr} or to the left of S_i in Sov by $l(S_i) = V_{fr} \cup (\cup_{j \in [1,i-1]} S_j)$. Similarly, we define the set of variables appearing to the right of S_i in Sov by $r(S_i) = \cup_{j \in [i+1,k]} S_j$.

Proposition 5.3. *For every PFU network $\mathcal{N}$, there exists at least one query $(Sov, \mathcal{N})$ without free variables.*

5.2 Answer to a query: a semantic definition based on decision trees

In this subsection, we assume that the plausibility structure is conditionable (cf. Theorem 4.3 page 64). The controlled plausibility distribution $\mathcal{P}_{V_E \,||\, V_D} = \otimes_{p\, P_i \in P} P_i$ can then be completed (cf. Definition 4.11 page 68) to give a plausibility distribution $\mathcal{P}_{V_E, V_D}$ over $V_E \cup V_D$. Similarly, the controlled feasibility distribution $\mathcal{F}_{V_D \,||\, V_E} = \wedge_{F_i \in F} F_i$ can be completed to give a feasibility distribution $\mathcal{F}_{V_E, V_D}$ over $V_E \cup V_D$. We also use the global utility $\mathcal{U}_V = \otimes_{u U_i \in U} U_i$ defined by the PFU network.

Imagine that we want to answer the query $Q = (Sov, \mathcal{N})$, where $\mathcal{N}$ is the network of the dinner problem and $Sov = (\min, \{mc\}).(\oplus_u, \{bp_J, bp_M\}).(\max, \{w\}).(\oplus_u, \{ep_J, ep_M\})$.

To answer such a query, one can use a decision tree. First, the restaurant chooses the worst possible main course, taking into account the feasibility distribution over mc. Here, $\mathcal{F}_{mc}((mc, \mathit{meat})) = \mathcal{F}_{mc,w}((mc, \mathit{meat}).(w, \mathit{white})) \vee \mathcal{F}_{mc,w}((mc, \mathit{meat}).(w, \mathit{red})) = t \vee t = t$. Similarly, $\mathcal{F}_{mc}((mc, \mathit{fish})) = t$. Both choices are feasible. Then, if A_1 denotes the assignment of mc, the uncertainty over those present at the beginning given the main course choice is described by the probability distribution $\mathcal{P}_{bp_J, bp_M \,|\, mc}(A_1)$. For each possible assignment A_2 of $\{bp_J, bp_M\}$, i.e. for each A_2 such that $\mathcal{P}_{bp_J, bp_M \,|\, mc}(A_1.A_2) \neq 0_p$, Peter chooses the best wine while taking into account the feasibility $\mathcal{F}_{w \,|\, mc, bp_J, bp_M}(A_1.A_2)$: if the restaurant chooses meat, Peter chooses an optimal value between red and white, and if the restaurant chooses fish, Peter can choose white wine only. Then, for each feasible assignment A_3 of w, the uncertainty regarding the presence of John and Mary at the end of the dinner is given by $\mathcal{P}_{ep_J, ep_M \,|\, bp_J, bp_M, mc, w}(A_1.A_2.A_3)$.

Note that the conditional probabilities used in the decision tree above are not directly defined by the network. They must be computed from the global distribution; this computation can be a challenge on large problems.

Utility $\mathcal{U}_V(A_1.A_2.A_3.A_4)$ can be associated with each possible complete assignment $A_1.A_2.A_3.A_4$ of the variables. For each possible assignment $A_1.A_2.A_3$ of $\{bp_J, bp_M, mc, w\}$, the last stage, i.e. the one in which ep_J and ep_M are assigned, can be seen as a *lottery* [137] whose expected utility is $\sum_{A_4 \in dom(\{ep_J, ep_M\})} p(A_4) \times u(A_4)$, where $p(A_4) = \mathcal{P}_{ep_J, ep_M \,|\, bp_J, bp_M, mc, w}(A_1.A_2.A_3.A_4)$ and $u(A_4) = \mathcal{U}_V(A_1.A_2.A_3.A_4)$. This expected utility becomes the reward of the scenario over $\{bp_M, bp_J, mc, w\}$ described by $A_1.A_2.A_3$. It provides us with a criterion for choosing an optimal value for w. The step in which bp_J and bp_M are assigned can then be seen as a lottery, which provides us with a criterion for choosing a worst value for mc. The computation associated with the previously described process is:

$$\min_{A_1 \in dom(mc), \mathcal{F}_{mc}(A_1) = t} \Bigg(\sum_{A_2 \in dom(\{bp_J, bp_M\}), \mathcal{P}_{bp_J, bp_M \,|\, mc}(A_1.A_2) \neq 0} \mathcal{P}_{bp_J, bp_M \,|\, mc}(A_1.A_2) \times \Bigg(\max_{A_3 \in dom(w), \mathcal{F}_{w \,|\, mc, bp_J, bp_M}(A_1.A_2.A_3) = t} \Bigg(\sum_{\substack{A_4 \,\in\, dom(\{ep_J, ep_M\}) \\ \mathcal{P}_{ep_J, ep_M \,|\, bp_J, bp_M, mc, w}(A_1.A_2.A_3.A_4) \,\neq\, 0}} \mathcal{P}_{ep_J, ep_M \,|\, bp_J, bp_M, mc, w}(A_1.A_2.A_3.A_4) \times \mathcal{U}_V(A_1.A_2.A_3.A_4) \Bigg) \Bigg) \Bigg)$$

Decision rules for the decision variables (argmin and argmax) can be recorded during the computation. This formulation represents the decision process as a decision tree in which each internal level corresponds to variables assignments. Arcs associated with the assignment of a set of decision variables are weighted by the feasibility of the decision given the previous assignments. Arcs associated with the assignment of environment variables are weighted by the plausibility degree of the assignment given the previous assignments. Leaf nodes correspond to the utilities of complete assignments, and a node collects the values of its children to compute its own value.

Formalization of the decision tree procedure

In order to formalize the decision tree procedure, some technical results are first introduced in Proposition 5.5. These results can be skipped for a first reading.

Definition 5.4. *Let $\mathcal{P}_{S_1 \,|\, S_2}$ be the conditional plausibility distribution of S_1 given S_2 and let $A \in dom(S_2)$. The function $\mathcal{P}_{S_1 \,|\, S_2}(A)$ is said to be well-defined iff $\mathcal{P}_{S_2}(A) \neq 0_p$. In this case, $\mathcal{P}_{S_1 \,|\, S_2}(A)$ is a plausibility distribution over S_1, which ensures the existence of at least one $A' \in dom(S_1)$ satisfying $\mathcal{P}_{S_1 \,|\, S_2}(A.A') \neq 0_p$. Similarly, for all $A \in dom(S_2)$, $\mathcal{F}_{S_1 \,|\, S_2}(A)$ is said to be well-defined iff $\mathcal{F}_{S_2}(A) = t$.*

Proposition 5.5. *Assume that the plausibility structure used is conditionable. Let $Q = (Sov, \mathcal{N})$ be a query where $Sov = (op_1, S_1) \cdot (op_2, S_2) \cdots (op_k, S_k)$. Let V_{fr} denote the set of free variables of Q. Then,*

(1) If $V_E \neq \emptyset$, let S_i be the leftmost set of environment variables appearing in Sov.

Then, for all $A \in dom(l(S_i))$, $\mathcal{P}_{S_i \,|\, l(S_i)}(A)$ is well-defined.

(2) Let $i, j \in [1, k]$ such that $i < j$, $S_i \subset V_E$, $S_j \subset V_E$, and $r(S_i) \cap l(S_j) \subset V_D$ (S_j is the first set of environment variables appearing to the right of S_i in Sov). Let $(A, A') \in dom(l(S_i)) \times dom(S_i)$. If $\mathcal{P}_{S_i \,|\, l(S_i)}(A)$ is well-defined and $\mathcal{P}_{S_i \,|\, l(S_i)}(A.A') \neq 0_p$, then, for all A'' extending $A.A'$ over $l(S_j)$, $\mathcal{P}_{S_j \,|\, l(S_j)}(A'')$ is well-defined.

(3) Let $i, j \in [1, k]$ such that $i < j$, $S_i \subset V_D$, $S_j \subset V_D$, and $r(S_i) \cap l(S_j) \subset V_E$ (S_j is the first set of decision variables appearing to the right of S_i in Sov). Let $(A, A') \in dom(l(S_i)) \times dom(S_i)$.

If $\mathcal{F}_{S_i \,|\, l(S_i)}(A)$ is well-defined and $\mathcal{F}_{S_i \,|\, l(S_i)}(A.A') = t$, then, for all A'' extending $A.A'$ over $l(S_j)$, $\mathcal{F}_{S_j \,|\, l(S_j)}(A'')$ is well-defined.

(4) The conditioning can be defined directly for controlled plausibility distributions as follows: for all $A \in dom(V_D)$, $\mathcal{P}_{V_E \,||\, V_D}(A)$ is a plausibility distribution over V_E. Thus, one can define from it conditional plausibility distributions, denoted $\mathcal{P}_{S \,|\, S' \,||\, V_D}(A)$, for all S, S' disjoint subsets of V_E, as in Theorem 4.3 page 64. Then, for all $i \in [1,k]$ such that $S_i \subset V_E$, $\mathcal{P}_{S_i \,|\, l(S_i) \cap V_E \,||\, V_D}$ is a function with scope $S_i \cup l(S_i) \cup V_D$, which does not depend on the assignment of $V_D - l(S_i)$. It can therefore be denoted by $\mathcal{P}_{S_i \,|\, l(S_i) \cap V_E \,||\, l(S_i) \cap V_D}$.

Moreover, if $\mathcal{P}_{V_E, V_D}$ is the completion of $\mathcal{P}_{V_E \,||\, V_D}$, then $\mathcal{P}_{S_i \,|\, l(S_i)} = \mathcal{P}_{S_i \,|\, l(S_i) \cap V_E \,||\, l(S_i) \cap V_D}$. This means that the conditioning on the completion of $\mathcal{P}_{V_E \,||\, V_D}$ coincides with the conditioning done directly on $\mathcal{P}_{V_E \,||\, V_D}$. As a result, completing $\mathcal{P}_{V_E \,||\, V_D}$ is useless to compute $\mathcal{P}_{S \,|\, l(S)}$. The situation is similar for feasibilities.

The technical results of Property 5.5 ensure that all the quantities involved in the following semantic answer to a query are defined and have a clear meaning.

Definition 5.6. *The semantic answer Sem-Ans(Q) to a query $Q = (Sov, \mathcal{N})$ is a function of the set V_{fr} of free variables of Q defined by*[1]

$$Sem\text{-}Ans(Q)(A) = \begin{cases} \Diamond \text{ if } \mathcal{F}_{V_{fr}}(A) = f \\ Qs_r(\mathcal{N}, Sov, A) \text{ otherwise} \end{cases}$$

with Qs_r inductively defined by:

$$(1) \qquad Qs_r(\mathcal{N}, \emptyset, A) = \mathcal{U}_V(A)$$

$$(2) \qquad Qs_r(\mathcal{N}, (op, S) \,.\, Sov, A) =$$

$$\begin{cases} \min\limits_{\substack{A' \in dom(S) \\ \mathcal{F}_{S|l(S)}(A.A') = t}} Qs_r\,(\mathcal{N}, Sov, A.A') & \text{if } (S \subset V_D) \wedge (op = \min) \\ \max\limits_{\substack{A' \in dom(S) \\ \mathcal{F}_{S|l(S)}(A.A') = t}} Qs_r\,(\mathcal{N}, Sov, A.A') & \text{if } (S \subset V_D) \wedge (op = \max) \\ \bigoplus_{u}\limits_{\substack{A' \in dom(S) \\ \mathcal{P}_{S|l(S)}(A.A') \neq 0_p}} \left(\mathcal{P}_{S|l(S)}(A.A') \otimes_{pu} Qs_r\,(\mathcal{N}, Sov, A.A')\right) & \text{if } (S \subset V_E) \end{cases}$$

In other words, each step involving decision variables (first two cases) is considered as an optimization step among the feasible choices, and each step involving environment variables (third case) is considered as a lottery [137] such that the rewards are the $Qs_r\,(\mathcal{N}, Sov, A.A')$, and such that the plausibility attributed to a reward is $\mathcal{P}_{S \,|\, l(S)}(A.A')$ (the formula looking like $\oplus_{ui}\,(p_i \otimes_{pu} u_i)$ is the expected utility of this lottery). When a set of decision variables S is eliminated, a decision rule for S can be recorded, using an argmax (resp. an argmin) if max (resp. min) is performed.

Example 5.7. *What is the maximum investment Peter can expect, and which associated decision(s) should he make if he chooses the menu without knowing who will attend? To answer this question, we can use a query in which bp_J, bp_M, ep_J, and ep_M are eliminated to the right of mc*

[1] $\Diamond$ is the unfeasible value, cf. Definition 1.6 page 17.

and w to represent the fact that their values are not known when the menu is chosen. This query is:

$$((\max, \{mc, w\}).(\oplus_u, \{bp_J, bp_M, ep_J, ep_M\}), \mathcal{N})$$

The answer is 6K€, with $(mc, meat).(w, red)$ as a decision. If Peter knows who comes, the query becomes

$$((\oplus_u, \{bp_J, bp_M\}).(max, \{mc, w\}).(\oplus_u, \{ep_J, ep_M\}), \mathcal{N})$$

and optimal values for mc and w can depend on bp_J and bp_M. The answer is 26K€ with a 20K€ gain from the observability of who is present at the beginning. The decision rule for $\{mc, w\}$ is $(mc, meat).(w, red)$ if John is present and Mary is not, $(mc, fish).(w, white)$ otherwise. Consider the query introduced at the beginning of Section 5.1:

$$((\min, \{mc\}).(\oplus_u, \{bp_J, bp_M\}).(\max, \{w\}).(\oplus_u, \{ep_J, ep_M\}), \mathcal{N})$$

The answer is $-\infty$: in the worst main course case, even if Peter chooses the wine, the situation can be unacceptable. In order to compute the expected utility for each menu choice, one can use a query in which mc and w are free variables:

$$((\oplus_u, \{bp_J, bp_M, ep_J, ep_M\}), \mathcal{N})$$

The answer is a function over $\{mc, w\}$. These examples show how queries can capture various situations in terms of partial observabilities or optimistic/pessimistic attitude, and how they can allow to evaluate various scenarios simultaneously by using free variables.

5.3 Answer to a query: a second more operational definition

The quantities $\mathcal{P}_{S \,|\, l(S)}(A.A')$ and $\mathcal{F}_{S \,|\, l(S)}(A.A')$ involved in the definition of the semantic answer to a query are not directly available from the local functions and can be very expensive to compute. For instance, with probabilities, $\mathcal{P}_{S \,|\, l(S)}(A.A')$ equals $\mathcal{P}_{S,l(S)}(A.A')/\mathcal{P}_{l(S)}(A)$. Computing $\mathcal{P}_{S,l(S)}(A.A') = \sum_{A'' \in dom(V-(S \cup l(S)))} \mathcal{P}_{V_E,V_D}(A.A'.A'')$ can require a time exponential in $|V - (S \cup l(S))|$. Moreover, such quantities must be computed at each node of the decision tree. Fortunately, there exists an alternative definition of the query meaning, which can be directly expressed using a PFU instance, that is, using the local plausibility, feasibility, and utility functions defined by a PFU network.

Definition 5.8. *The operational answer Op-Ans(Q) to a query $Q = (Sov, \mathcal{N})$ is a function of the free variables of Q: if A is an assignment of the free variables, then $(Op\text{-}Ans(Q))(A)$ is defined inductively as follows:*

$$(Op\text{-}Ans(Q))(A) = Qo_r\,(\mathcal{N}, Sov, A)$$

$$Qo_r(\mathcal{N}, (op, S)\,.\,Sov, A) = op_{A' \in dom(S)}\, Qo_r\,(\mathcal{N}, Sov, A.A') \tag{5.1}$$

$$Qo_r(\mathcal{N}, \emptyset, A) = \left(\left(\bigwedge_{F_i \in F} F_i \right) \star \left(\underset{P_i \in P}{\otimes_p} P_i \right) \otimes_{pu} \left(\underset{U_i \in U}{\otimes_u} U_i \right) \right)(A) \tag{5.2}$$

By Equation 5.2, if all the problem variables are assigned, the answer to the query is the combination of the plausibility degree, the feasibility degree, and the utility degree of the corresponding complete assignment. By Equation 5.1, if the variables are not all assigned and (op, S) is the leftmost operator-variable(s) pair in Sov, the answer to the query is obtained by eliminating S using *op* as an elimination operator. Again, optimal decision rules for the decision variables can be recorded if needed, using argmin and argmax. Equivalently, $Op\text{-}Ans(Q)$ can be written:

$$Op\text{-}Ans(Q) = Sov\left(\left(\bigwedge_{F_i \in F} F_i\right) \star \left(\underset{P_i \in P}{\otimes_p} P_i\right) \otimes_{pu} \left(\underset{U_i \in U}{\otimes_u} U_i\right)\right)$$

This shows that $Op\text{-}Ans(Q)$ actually corresponds to the generic form of Equation 2.28 page 51.

5.4 Equivalence theorem

Theorem 5.9 proves that the semantic definition $Sem\text{-}Ans(Q)$ gives semantic foundations to what is computed with the operational definition $Op\text{-}Ans(Q)$.

Theorem 5.9. *If the plausibility structure is conditionable, then, for all queries Q on a PFU network, $Sem\text{-}Ans(Q) = Op\text{-}Ans(Q)$ and the optimal policies for the decisions are the same with $Sem\text{-}Ans(Q)$ and $Op\text{-}Ans(Q)$.*

In other words, Theorem 5.9 shows that it is possible to perform computations in a completely generic algebraic framework, while providing the result of the computations with decision-theoretic foundations, based on decision trees. Hence, computing $Op\text{-}Ans(Q)$ is meaningful.

Due to this equivalence theorem, $Op\text{-}Ans(Q)$ is denoted simply by $Ans(Q)$ in the following. Note that the operational definition applies even in a non-conditionable plausibility structure. Giving a decision-theoretic based semantics to *Op-Ans* when the plausibility structure is not conditionable is an open issue.

5.5 Bounded queries

It may be interesting to relax the problem of computing the exact answer to a query. Assume that the leftmost operator-variable(s) pair in the sequence Sov is $(\max, \{x\})$, with x a decision variable. From the decision maker point of view, computing decision rules providing an expected utility greater than a given threshold θ may be sufficient. This is the case for the E-MAJSAT problem, defined as "*Given a boolean formula over a set of variables $V = V_D \cup V_E$, does there exist an assignment of V_D such that the formula is satisfied for at least half of the assignments of V_E?*" Extending the generic PFU framework to answer such queries is done in Definitions 5.10 and 5.11, which introduce bounded queries.

Definition 5.10. *A bounded query $B\text{-}Q$ is a triple $(Sov, \mathcal{N}, \theta)$, such that $(Sov, \mathcal{N})$ is a query and $\theta \in E_u$ (θ is the threshold).*

Definition 5.11. *The answer $Ans(B\text{-}Q)$ to a bounded query $B\text{-}Q = (Sov, \mathcal{N}, \theta)$ is a boolean function of the free variables of the "unbounded" query $Q = (Sov, \mathcal{N})$. For every assignment A of these free variables,*

$$(Ans(B\text{-}Q))(A) = \begin{cases} t & \text{if } Ans(Q)(A) \succeq_u \theta \\ f & \text{otherwise.} \end{cases}$$

As the threshold θ may be used to prune the search space during the resolution, computing the answer to a bounded query is easier than computing the answer to an unbounded one.

5.6 Back to existing frameworks

Let us consider again some frameworks mentioned in Chapter 2. Solving a CSP (Equation 2.4 page 25) or a totally ordered soft CSP (Equation 2.5 page 26) corresponds to the query $Q = ((\max, V), \mathcal{N})$, with $\mathcal{N}$ the PFU network corresponding to the CSP and V the set of variables of the CSP. Computing the probability distribution of a variable y for a Bayesian network (Equation 2.9 page 33) can be modeled using $Sov = (+, V - \{y\})$. These examples are *mono-operator queries*, involving only one type of elimination operator.

Let us consider *multi-operator queries.* The search for an optimal policy for the stochastic CSP associated with Equation 2.8 page 31 is captured by $Sov = (\max, \{x_1\}).(+, \{x_2\}).(\max, \{x_3\})$. The modeling is similar for the query on influence diagrams of Equation 2.14 page 37, which can be modeled using $Sov = (\max, \{ca\}).(+, \{re\}).(\max, \{po\}).(+, \{bu, eq, al\})$.

For a finite horizon MDP with T time-steps (Equation 2.21 page 47), the query looks like $Q = ((\max, \{d_1\}).(\oplus_u, \{s_2\}).(\max, \{d_2\}) \ldots (\oplus_u, \{s_T\}).(\max, \{d_T\}), \mathcal{N})$, where $\oplus_u = +$ with probabilistic MDP and $\oplus_u = \min$ with pessimistic possibilistic MDP. The initial state s_1 is a free variable. With a quantified CSP or a quantified boolean formula, elimination operators min and max are used to represent $\forall$ and $\exists$.

More formally, we can show:

Theorem 5.12. *Queries and bounded queries can be used to express and solve the following list of problems:*

1. *SAT framework: SAT, MAJSAT, E-MAJSAT, quantified boolean formula, stochastic SAT (SSAT) and extended-SSAT [82].*

2. *CSP (or CN) framework:*

   - *Check consistency for a CSP [84]; find a solution to a CSP; count the number of solutions of a CSP.*
   - *Seek a solution of a valued CSP [123].*
   - *Solve a quantified CSP [15].*
   - *Find a conditional decision or an unconditional decision for a mixed CSP or a probabilistic mixed CSP [47].*
   - *Find an optimal policy for a stochastic CSP or a policy with a value greater than a threshold; solve a stochastic COP (Constraint Optimization Problem) [138].*

3. *Integer Linear Programming [124] with finite domain variables.*

4. *Search for a solution plan with a length $\leq k$ in a classical planning problem (STRIPS-like planning [49, 58]).*

5. *Answer classical queries on Bayesian networks [96], Markov random fields [22], and chain graphs [55], with plausibilities expressed as probabilities, possibilities, or κ-rankings:*

- *Compute plausibility distributions.*
- *MAP (Maximum A Posteriori hypothesis).*
- *MPE (Most Probable Explanation).*
- *Compute the plausibility of an evidence.*
- *CPE task for hybrid networks [36] (CPE means CNF Probability Evaluation, a CNF being a formula in Conjunctive Normal Form).*

6. *Solve an influence diagram [64].*

7. *With a finite horizon, solve a probabilistic MDP, a possibilistic MDP, a MDP based on κ-rankings, completely or partially observable (POMDP), factored or not [111, 89, 119, 19, 18].*

5.7 Extensions to other classes of queries

Queries can be made more complex by relaxing some assumptions:

- In the definition of queries, the order $\preceq_u$ on E_u is assumed to be total. Extending the results to a *partial* order is possible if $(E_u, \preceq_u)$ defines a lattice (partially ordered set closed under least upper and greatest lower bounds) and if $\otimes_{pu}$ distributes over the least upper bound *lub* and greatest lower bound *glb* (i.e. $p \otimes_{pu} lub(u_1, u_2) = lub(p \otimes_{pu} u_1, p \otimes_{pu} u_2)$ and $p \otimes_{pu} glb(u_1, u_2) = glb(p \otimes_{pu} u_1, p \otimes_{pu} u_2)$). This allows semiring CSPs [10, 11] to be captured in the framework. We believe that other extensions to partial orders on utilities should allow algebraic MDPs [97] to be captured.

- One can try to relax the *no-forgetting* assumption, as in limited memory influence diagrams (LIMIDs [81]), which show that this can be relevant for decision processes involving multiple decision makers or memory constraints on the policy recording. In a LIMID, the goal is to search for decision rules $\delta_d : dom(S_d) \rightarrow dom(d)$, one per decision variable d, where S_d is the set of variables on which decision d is allowed to depend. These sets S_d are explicitly specified and may violate the no-forgetting assumption. In such cases, optimal decisions can become *nondeterministic* (decisions such as "choose $x = 0$ with probability p and $x = 1$ with probability $1 - p$").

- The order in which decisions are made and environment variables are observed is total and completely determined by the query. One may wish to compute not only an optimal policy for the decisions, but also an *optimal order* in which to perform decisions, without exactly knowing the steps at which other agents make decisions or the steps at which observations are made. Work on influence diagrams with unordered decisions, such as [68], is a good starting point to try and extend our work in this direction.

- Finally, relaxing the *finite domain* variables assumption is not direct, since transforming $\oplus_u = +$ into integrals is not straightforward, and performing min- or max-eliminations over continuous domains requires the guarantee of existence of a supremum. In this direction, Simple Temporal Problems (STPs [39]) and their extensions could be considered. In such problems, variables are timepoints taking values in continuous intervals, and constraints concern durations between two timepoints, which represent for example durations of activities. Among the extensions of STPs, Simple Temporal Problems with Preferences(STPPs [72]), Simple Temporal Problems with Uncertainties (STPUs [134, 136]), and Simple Temporal Problems with Preferences and Uncertainties (STPPUs [117]) are good starting points. Note that in these formalisms, uncertainties correspond to boolean indetermisms, which means that the only uncertainties involved are that some timepoints, called contingent timepoints, are not controllable and can take any value in an interval. In order to extend the PFU framework to encompass these formalisms, we actually need to handle continuous plausibility distributions and to use elimination operators $\oplus_p$, $\oplus_u$ defined on intervals of values.

5.8 Summary

In Chapter 5, the last element of the PFU framework, a class of queries on PFU networks, has been introduced. A decision-tree based definition of the answer to a query has been provided. The first main result of this chapter is Theorem 5.9, which gives theoretical foundations to another equivalent operational definition, reducing the answer to a query to a sequence of eliminations on a combination of scoped functions. The latter is best adapted to future algorithms, because it directly handles the local functions defined by a PFU network. The second important result is Theorem 5.12, which shows that many standard queries are PFU queries. Overall, the PFU framework definition lies in Definitions 3.5, 3.8, 3.9 for the algebraic structure, Definition 4.18 for the network, and Definitions 5.2, 5.8 for queries.

The PFU formulation of a concrete problem which involves plausibilities, feasibilities, utilities, and sequential decision making (a problem of deployment and maintenance of a constellation of satellites [61]), is given in Appendix C.

5.9 Conclusion of Part I: gains and costs of the PFU framework

A better understanding Theorem 5.12 shows that many existing frameworks are instances of the PFU framework. Through this unification, similarities and differences between existing formalisms can be analyzed. For instance, comparing VCSPs and the optimistic version of finite horizon possibilistic MDPs through the operational definition of the answer to a query, one will notice that algebraically speaking, a finite horizon optimistic possibilistic MDP (partially observable or not) is a fuzzy CSP. Libraries available for VCSPs can then be used to solve such MDPs.

From the complexity theory point of view, studying the time and space complexity for computing Equation 2.28 (page 51) can lead to upper bounds on the complexity for several frameworks simultaneously. One may also try to characterize which properties lead to a given theoretical complexity.

Increased expressive power The expressive power of PFU networks is the result of a number of features: (1) flexibility of the plausibility/utility model; (2) flexibility of the possible networks; (3)

flexibility of the queries in terms of situation modeling. This enables queries on PFU networks to cover generic finite horizon sequential decision making problems with plausibilities, feasibilities, and utilities, cooperative or adversarial decision makers, partial observabilities, and possible parameters in the decision process modeled through free variables.

As none of the frameworks indicated in Theorem 5.12 presents such a flexibility, for every subsumed formalism X indicated in Theorem 5.12, it is possible to find a problem which can be represented with PFUs but *not directly* with X. More specifically, compared to influence diagrams [64, 68, 131, 92, 67] or valuation networks (VNs [128, 130, 41]), PFUs can deal with more than the probabilistic expected utility structure and allow us to perform eliminations with min to model the presence of adversarial agents. Thus, quantified boolean formulas cannot be represented with influence diagrams or VNs, but are covered by PFU queries (see Theorem 5.12). Moreover, PFU networks use a DAG which captures normalization conditions of plausibilities or feasibilities, whereas with VNs, this information is lost. Compared to sequential influence diagrams [67] or sequential VNs [41], PFUs can express some so-called *asymmetric decision problems* (problems in which some variables may not even need to be considered in a decision process) by adding dummy values to variables.

Actually, some simple problems which can be expressed with PFUs cannot be apparently directly expressed in other frameworks. The simple instance "feasibilities *with normalization conditions* + hard requirements" is not captured by any of the subsumed frameworks (using a CSP to model it would result in a loss of the information provided by the normalization conditions on feasibilities). The same occurs for "influence diagrams - like sequential decision processes based on possibilistic expected utility", which could be called possibilistic influence diagrams.[2] Same again for the instance "stochastic CSPs without contingency assumption", for the instance "max-QBF" (analogous to max-SAT), or for the instance "quantified VCSPs", which could correspond to VCSPs involving alternating min and max eliminations modeling the presence of antagonist decision makers. Thus, the PFU framework also covers yet-unpublished frameworks.

The cost of greater flexibility and increased expressive power is that the PFU framework cannot be described as simply and straightforwardly as, for example, constraint networks.

Generic algorithms Part II will show that generic algorithms can be built to answer queries on PFU networks. As previously said, building generic algorithms should facilitate cross-fertilization in the sense that any of the subsumed formalisms will directly benefit from the techniques developed in another subsumed formalism. This fits into a growing effort to generalize resolution methods used for different AI problems. For example, soft constraint propagation drastically improves the resolution of VCSPs; integrating such a tool in a generic algorithm on PFUs could improve the resolution of influence diagrams. Using abstract operators may enable us to identify algorithmically interesting properties, or to infer necessary or sufficient conditions for a particular algorithm to be usable.

However, one could argue that some techniques are highly specific to one formalism or to one type of problem, and that, in this case, dedicated approaches certainly outperform a generic algorithm. A solution for this can be to characterize the actual properties used by a dedicated

[2] Possibilistic influence diagrams were proposed very recently, in a work parallel to this thesis [56]. This formalism is a simple instantiation of the PFU framework.

approach, in order to generalize it as much as possible. Moreover, even if specialized schemes usually improve over generic ones, there exist cases in which general tools can be more efficient than specialized algorithms. See, for example, [121] or the use of SAT solvers for solving optimal STRIPS planning problems.

Part II

Generic algorithms for answering PFU queries

Chapter 6

First generic algorithms

The PFU framework is flexible and unifies several existing AI formalisms. One may think that the cost to pay for such a genericity is that answering a PFU query is necessarily intractable. One of the aims of the following chapters is to contradict this idea, by showing that tractability is more a consequence of the query considered than a side effect of genericity.

In fact, the PFU framework has been built not only for its knowledge representation abilities, but also to be able to define generic algorithms capable of answering queries. Some of our choices have even been justified by algorithmic reasons. In other words, we want to be able to answer queries as efficiently as possible, and not only to express them.

In the sequel, we introduce generic resolution schemes which are either generalizations of already existing algorithms, or new techniques applicable to all PFU subsumed formalisms. This chapter presents two first generic algorithms which answer arbitrary PFU queries without any further assumption on the algebraic structure. These algorithms both work on the operational definition of the answer to a query, defined as $Ans(Q) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{p\,P_i \in P} P_i) \otimes_{pu} (\otimes_{u\,U_i \in U} U_i))$. More precisely, we introduce:

- a basic tree search algorithm;
- a generic variable elimination algorithm [7], which intends to exploit the factorization into local scoped functions for the best.

Complexity results are also provided, notably using a parameter called *constrained induced-width*.

6.1 A basic tree search algorithm

The operational definition of the answer to a query Q (cf Definition 5.8 page 80) defines a naive exponential time algorithm to compute $Ans(Q)$ using a tree exploration procedure. This algorithm is given in Figure 6.1.

For each assignment A of the free variables of Q, a tree is explored. Each node in this tree corresponds to a partial assignment of the variables, and variables are assigned in an order "compatible" with Sov. The value of a leaf is provided by the combination of the scoped functions of the PFU network, applied to the complete assignment defined by the path from the root to this leaf. Depending on the operator used, the value of an internal node is obtained by performing a min, max, or $\oplus_u$ operation on the values of its children. The root node returns $(Ans(Q))(A)$. For a query $(Sov, \mathcal{N})$, the first call is **TreeSearchAnswerQ**$(Sov, \mathcal{N})$. It returns $Ans(Q)$.

```
TreeSearchAnswerQ($Sov, (V, G, P, F, U)$)
begin
    foreach $A \in dom(V_{fr})$ do $\varphi(A) \leftarrow$ AnswerQ($Sov, (V, G, P, F, U), A$)
    return $\varphi$
end

AnswerQ($Sov, (V, G, P, F, U), A$)
begin
    if $Sov = \emptyset$ then  return $\Big((\wedge_{F_i \in F} F_i) \star (\otimes_{p\, P_i \in P} P_i) \otimes_{pu} (\otimes_{u\, U_i \in U} U_i)\Big)(A)$
    else
        $(op, S).Sov' \leftarrow Sov$
        choose $x \in S$
        if $S = \{x\}$ then  $Sov \leftarrow Sov'$  else  $Sov \leftarrow (op, S - \{x\}).Sov'$
        $dom \leftarrow dom(x)$
        $res \leftarrow \Diamond$
        while $dom \neq \emptyset$ do
            choose $a \in dom$
            $dom \leftarrow dom - \{a\}$
            $res \leftarrow op\,(res,$ AnswerQ($Sov, (V, G, P, F, U), A.(x, a)$))
        return $res$
end
```

Figure 6.1: A generic tree search algorithm to answer a query $Q = (Sov, (V, G, P, F, U))$.

If one assumes that every operator returns a result in a constant time and that each memory read also takes a constant time, then the time complexity of this algorithm is $O(m \cdot d^n)$, where m stands for the number of scoped functions, d stands for the maximum domain size, and n stands for the number of variables.[1]

The space complexity is linear, hence computing the answer to a bounded query is *PSPACE*. Moreover, the satisfiability of a QBF is a PSPACE-complete problem which can be expressed as a bounded query (cf. Theorem 5.12 page 82), hence computing the answer to a bounded query is *PSPACE-hard*. Being PSPACE and PSPACE-hard, the decision problem consisting in answering a bounded query is *PSPACE-complete*. This result is not surprising, but it gives an idea of the level of expressiveness which can be reached by the PFU framework. More work is needed to identify subclasses of queries with a lower complexity, although many are already known.

Nevertheless, if one wants to record a policy for decision variables eliminated with max, then the space complexity of the policy recording can become exponential in the number of variables not eliminated with max. In order to recover a polynomial recording space, one can simply record a "horizon-restricted" policy for the k first decisions only.

[1] In fact, an upper bound on the time needed to get $\varphi(A)$ for a scoped function φ represented as a table is $O(n \cdot log(d))$, and an operator returns a result in a time depending on the size of its arguments. We decide to adopt the same convention as [93], where these two operations are assumed to be in constant time. Such an assumption does not change the complexity class, and can be relaxed simply by adding factors such as $n \cdot log(d)$ in all time complexity results. For example, we should say that the time complexity of algorithm TreeSearchAnswerQ is $O(m \cdot n \cdot log(d) \cdot d^n)$ instead of $O(m \cdot d^n)$. Similarly, logarithmic factors should be integrated to all space complexities, since numbers are recorded using bits.

6.2 A first naive variable elimination algorithm

Quite naturally, a generic Variable Elimination (VE) algorithm can also be defined to answer PFU queries. This algorithm, inspired by the seminal Bertelé and Brioschi's proposal [7] and by [127, 32, 75], is given in Figure 6.2. It eliminates variables from the right to the left of the sequence Sov of the query, whereas with the tree search procedure, variables are assigned from the left to the right. This right-to-left processing entails that the algorithm naturally returns a function of the free variables of the query. The first call is **Basic-VE-answerQ**$(Sov, (V, G, P, F, U))$. The time and space complexities of this algorithm are $O(m \cdot d^n)$.

```
**Basic-VE-answerQ**$(Sov, (V, G, P, F, U))$
**begin**
    $\varphi_0 \leftarrow \Big((\wedge_{F_i \in F} F_i) \star (\otimes_{p\,P_i \in P} P_i) \otimes_{pu} (\otimes_{u\,U_i \in U} U_i)\Big)$
    **while** $Sov \neq \emptyset$ **do**
        $Sov'.(op, S) \leftarrow Sov$
        choose $x \in S$
        **if** $S = \{x\}$ **then** $Sov \leftarrow Sov'$ **else** $Sov \leftarrow Sov'.(op, S - \{x\})$
        $\varphi_0 \leftarrow op_x\, \varphi_0$
    **return** $\varphi_0$
**end**
```

Figure 6.2: A first generic variable elimination algorithm for answering a query $Q = (Sov, (V, G, P, F, U))$.

Improving the basic scheme

The **Basic-VE-answerQ** algorithm is actually a very naive variable elimination scheme, because it begins by combining all the scoped functions (first line) before eliminating variables, whereas the advantage of a standard variable elimination algorithm is primarily to use the factorization into local functions [7]. Ideally, we would like to perform computations as local as possible by considering only scoped functions having x in their scope when computing a quantity like $op_x(F \star P \otimes_{pu} U)$.

Let us first introduce some additional notations and conventions:

- Given a set Φ of scoped functions, we denote by Φ^{+x} (resp. Φ^{-x}) the set of scoped functions in Φ having (resp. not having) x in their scope: $\Phi^{+x} = \{\varphi \in \Phi \,|\, x \in sc(\varphi)\}$ (resp. $\Phi^{-x} = \{\varphi \in \Phi \,|\, x \notin sc(\varphi)\}$).

- A quantity like $op_x((\wedge_{F_i \in F} F_i) \star (\otimes_{p\,P_i \in P} P_i) \otimes_{pu} (\otimes_{u\,U_i \in U} U_i))$, where P, F, U are sets of plausibility, feasibility, and utility functions respectively, is simply denoted as $op_x(F \star P \otimes_{pu} U)$: we consider the combination of scoped functions of the same "type" as implicit.

- Every combination operator $\otimes$ defined on a set E not containing $\Diamond$ is extended on $E \cup \{\Diamond\}$ by $\Diamond \otimes e = e \otimes \Diamond = \Diamond$ (combining anything with something unfeasible is unfeasible too).[2] This implies that $f \star (p \otimes_{pu} u) = p \otimes_{pu} (f \star u)$.

Proposition 6.1 is a first step towards the use of factorizations.

[2] An operator op can be used both as a combination operator between scoped functions and as an elimination operator on some variables. In this case, the extension of op used as a combination operator creates an operator op' such that $op'(e, \Diamond) = \Diamond$, whereas the extension of op considered as an elimination operator creates an operator op'' such that $op''(e, \Diamond) = e$. op' and op'' coincide on E but differ on $E \cup \{\Diamond\}$.

Proposition 6.1. *Let $(E_p, E_u, \oplus_u, \otimes_{pu})$ be a totally ordered expected utility structure (EU structure). Then, for all sets P, F, U of plausibility, feasibility, and utility functions respectively, and for all $op \in \{\min, \max, \oplus_u\}$, $op_x(F \star P \otimes_{pu} U) = F^{-x} \star P^{-x} \otimes_{pu} (op_x\,(F^{+x} \star P^{+x} \otimes_{pu} U))$.*

Moreover, if $P^{+x} = \emptyset$ and $op \in \{\min, \max\}$, $op_x(F^{+x} \star U) = U^{-x} \otimes_u (op_x\,(F^{+x} \star U^{+x}))$.

Proposition 6.1 asserts that when a variable x is eliminated, it is not necessary to consider plausibility functions or feasibility functions without x in their scope. Furthermore, if there are no plausibility functions depending on x quantified with min or max, then it is not necessary to consider utility functions without x in their scope either. This means that (1) it is always possible to take advantage of the factorization of the global plausibility and feasibility into local plausibility and feasibility functions, and (2) the factorization into local utility functions is directly usable if $P^{+x} = \emptyset$ and the elimination operator is min or max. In order to see how general the condition "$P^{+x} = \emptyset$" is, we use the following proposition.

Proposition 6.2. *Let $(Sov, (V, G, P, F, U))$ be a query. Let x be a variable involved in the rightmost operator-variable(s) pair in Sov. Then, $(x \in V_D) \to (P^{+x} = \emptyset)$ and $(x \in V_E) \to (F^{+x} = \emptyset)$.*

Therefore, if x denotes a rightmost variable in *Sov*, then Proposition 6.2 enables us to infer that at the first elimination step:

- If x is a decision variable, then $P^{+x} = \emptyset$. Proposition 6.1 entails that

$$\underset{x}{op}(F \star P \otimes_{pu} U) = F^{-x} \star P^{-x} \otimes_{pu} (U^{-x} \otimes_u (\underset{x}{op}\,(F^{+x} \star U^{+x})) \tag{6.1}$$

  As a result, only scoped functions having x in their scope need to be considered: it suffices to compute $\max_x(F^{+x} \star U^{+x})$ if x is quantified with max and $\min_x(F^{+x} \star U^{+x})$ otherwise.

- If $x \in V_E$, then $F^{+x} = \emptyset$. In this case, Proposition 6.1 entails that

$$\underset{x}{\oplus_u}(F \star P \otimes_{pu} U) = F^{-x} \star P^{-x} \otimes_{pu} (\underset{x}{\oplus_u}\,(P^{+x} \otimes_{pu} U)) \tag{6.2}$$

  In general, the computation $\oplus_{ux}\,(P^{+x} \otimes_{pu} U)$ in Equation 6.2 cannot be decomposed any further, in order to avoid considering scoped functions in U^{-x}. The basic reason for this is that the PFU algebraic structure makes no assumption on the relation between $\oplus_u$ and $\otimes_u$. This problem is referred to as the *undecomposability problem.*

6.3 Solving the undecomposability problem via two distinct sufficient conditions

We give two axioms, each of which makes it possible to avoid considering scoped functions in U^{-x} when computing $\oplus_{ux}\,(P^{+x} \otimes_{pu} U)$. This means that they allow us to use factorizations for the best. These two axioms, denoted Ax^{SG} and Ax^{SR}, are enounced as follows:

$$Ax^{SR} : \begin{cases} \otimes_u \text{ distributes over } \oplus_u \\ \text{and } p \otimes_{pu} (u_1 \otimes_u u_2) = (p \otimes_{pu} u_1) \otimes_u u_2 \text{ for all } (p, u_1, u_2) \in E_p \times E_u \times E_u \end{cases}$$

$$Ax^{SG} : \text{“}\otimes_u = \oplus_u \text{ on } E_u\text{” (and not on } E_u \cup \{\Diamond\})$$

The first sufficient decomposability axiom is denoted Ax^{SR} as “axiom for the semiring case”, because it makes $(E_u, \oplus_u, \otimes_u)$ a semiring (see Proposition 6.3 below). The second sufficient decomposability axiom is denoted Ax^{SG} as “axiom for the semigroup case”, because it makes the structure $(E_u, \oplus_u, \otimes_u)$ similar to the structure $(E_u, \oplus_u)$, which is a semigroup. These two disjoint axioms cover various standard EU structures, as shown in Table 6.1.

	E_p	E_u	$\otimes_u$	$\oplus_u$	$\otimes_{pu}$	Ax^{SR}	Ax^{SG}
1	$\mathbb{R}^+$	$\mathbb{R} \cup \{-\infty\}$	$+$	$+$	$\times$		$\surd$
2	$\mathbb{R}^+$	$\mathbb{R}^+$	$\times$	$+$	$\times$	$\surd$	
3	$[0,1]$	$[0,1]$	min	max	min	$\surd$	
4	$[0,1]$	$[0,1]$	min	min	$\max(1{-}p, u)$		$\surd$
5	$\mathbb{N} \cup \{\infty\}$	$\mathbb{N} \cup \{\infty\}$	$+$	min	$+$	$\surd$	
6	$\{t, f\}$	$\{t, f\}$	$\wedge$	$\vee$	$\wedge$	$\surd$	
7	$\{t, f\}$	$\{t, f\}$	$\wedge$	$\wedge$	$\rightarrow$		$\surd$
8	$\{t, f\}$	$\{t, f\}$	$\vee$	$\vee$	$\wedge$		$\surd$
9	$\{t, f\}$	$\{t, f\}$	$\vee$	$\wedge$	$\rightarrow$	$\surd$	

Table 6.1: Expected utility structures satisfying Ax^{SR} or Ax^{SG}: 1. probabilistic expected utility with additive utilities (allows the probabilistic expected utility of a cost or a gain to be computed), 2. probabilistic expected utility with multiplicative utilities (allows the probability of satisfaction of some constraints to be computed), 3. possibilistic optimistic expected utility, 4. possibilistic pessimistic expected utility, 5. qualitative utility with κ-rankings and with only positive utilities, 6. boolean optimistic expected utility with conjunctive utilities (allows one to know whether there exists a possible world in which all goals of a set of goals G are satisfied), 7. boolean pessimistic expected utility with conjunctive utilities (allows one to know whether in all possible worlds, all goals of a set of goals G are satisfied), 8. boolean optimistic expected utility with disjunctive utilities (allows one to know whether there exists a possible world in which at least one goal of a set of goals G is satisfied), 9. boolean pessimistic expected utility with disjunctive utilities (allows one to know whether in all possible worlds, at least one goal of a set of goals G is satisfied).

Proposition 6.3. *Let $(E_p, E_u, \oplus_u, \otimes_{pu})$ be an EU structure satisfying Ax^{SR} (the underlying utility structure being $(E_u, \otimes_u)$). Then, $(E_u, \oplus_u, \otimes_u)$ is a commutative semiring.*

Proposition 6.4 asserts that as soon as one of these two axioms holds, the undecomposability problem is solved.

Proposition 6.4. *Let $(E_p, E_u, \oplus_u, \otimes_{pu})$ be an EU structure. Let P and U be sets of plausibility and utility functions respectively.*

If Ax^{SR} holds, then

$$\bigoplus_{u\,x} (P^{+x} \otimes_{pu} U) \quad = \quad U^{-x} \otimes_u (\bigoplus_{u\,x} (P^{+x} \otimes_{pu} U^{+x})) \tag{6.3}$$

If Ax^{SG} holds, then

$$\bigoplus_{u\,x} (P^{+x} \otimes_{pu} U) \quad = \quad ((\bigoplus_{p\,x} P^{+x}) \otimes_{pu} U^{-x}) \otimes_u (\bigoplus_{u\,x} (P^{+x} \otimes_{pu} U^{+x})) \tag{6.4}$$

This shows that when eliminating an environment variable x with $\oplus_u$, only plausibility and utility functions having x in their scope need to be considered. Note that in Equation 6.4, there is no reason for the quantity $\oplus_{p_x} P^{+x}$ to equal 1_p. Proposition 6.4 can be illustrated by the probabilistic expected satisfaction and the probabilistic expected additive utility. In the first case, we have $\sum_x (P^{+x} \times U) = U^{-x} \times (\sum_x (P^{+x} \times U^{+x}))$, whereas in the second one, we have $\sum_x (P^{+x} \times (U^{-x} + U^{+x})) = ((\sum_x P^{+x}) \times U^{-x}) + (\sum_x (P^{+x} \times U^{+x}))$.

6.4 Definition of an improved variable elimination algorithm

As we shall see, Ax^{SR} and Ax^{SG} enable us to compute the answer to a query using a variable elimination algorithm which considers only scoped functions having x in their scopes when eliminating a variable x.

6.4.1 Improved VE algorithm in the semiring case

When Ax^{SR} holds, it is actually possible to simplify Equation 6.3 with no loss of generality, by transforming the problem specification via an expected utility structure morphism. Indeed, let us consider the simpler axiom

$$Ax^{SR'} : \begin{cases} (E_p, \preceq_p) = (E_u, \preceq_u) = (E, \preceq) \\ \otimes_p = \otimes_{pu} = \otimes_u = \otimes \\ \oplus_p = \oplus_u = \oplus \end{cases}$$

Theorem 6.5. *Let $S = (E_p, E_u, \oplus_u, \otimes_{pu})$ be a totally ordered EU structure whose underlying plausibility and utility structures are $(E_p, \oplus_p, \otimes_p)$ and $(E_u, \otimes_u)$ respectively. Let $\phi : E_p \to E_u$ be the function defined by $\phi(p) = p \otimes_{pu} 1_u$.*

(a) If S satisfies $Ax^{SR'}$, then S satisfies Ax^{SR}.

(b) If S satisfies Ax^{SR}: let $(E, \preceq) = (E_u, \preceq_u)$, $\oplus = \oplus_u$, and $\otimes = \otimes_u$. Then,

- *The structure $S' = (E, E, \oplus, \otimes)$ is a totally $\preceq$-ordered EU structure, with $(E, \oplus, \otimes)$ as a plausibility structure and $(E, \otimes)$ as a utility structure. Moreover, it satisfies $Ax^{SR'}$.*
- *For every PFU network $\mathcal{N} = (V, G, P, F, U)$ on S, $\mathcal{N}' = (V, G, \{\phi(P_i) \,|\, P_i \in P\}, F, U)$ is a PFU network on S'. $\mathcal{N}'$ is denoted $\phi(\mathcal{N})$.*
- *For every query $Q = (Sov, \mathcal{N})$ on a PFU network $\mathcal{N}$ defined on S, $Q' = (Sov, \phi(\mathcal{N}))$ is a query on the PFU network $\phi(\mathcal{N})$. Moreover, $Ans(Q) = Ans(Q')$ and the optimal policies for the decision variables are the same with Q and Q'.*

Theorem 6.5(a) shows that axiom Ax^{SR} is weaker than axiom $Ax^{SR'}$. Theorem 6.5(b) shows that if an expected utility structure satisfies Ax^{SR}, then it is possible to recover $Ax^{SR'}$ thanks to the morphism $\phi : p \to p \otimes_{pu} 1_u$, which enables us to transform a query Q on a PFU network $\mathcal{N}$ into an equivalent query on the PFU network $\phi(\mathcal{N})$.

As a result, $Ax^{SR'}$ is equivalent to Ax^{SR} and we can deal with $Ax^{SR'}$ instead of Ax^{SR}. The interest of $Ax^{SR'}$ is that it involves only two customizable operators $\oplus$ and $\otimes$ and one ordered set $(E, \preceq)$, which will simplify the future algorithms. The axioms making a structure an expected utility structure also become simpler, as shown in Proposition 6.7.

Definition 6.6. *$(E, \oplus, \otimes)$ is a totally ordered Monotonic Commutative Semiring (totally ordered MCS) iff it is a commutative semiring equipped with a total order $\preceq$ such that $\oplus$ and $\otimes$ are monotonic with respect to $\preceq$.*

Proposition 6.7. *$(E_p, E_u, \oplus_u, \otimes_{pu})$ is a totally ordered EU structure satisfying $Ax^{SR'}$ (the underlying plausibility and utility structures being $(E_p, \oplus_p, \otimes_p)$ and $(E_u, \otimes_u)$ respectively) if and only if $(E_u, \oplus_u, \otimes_u)$ is a totally ordered MCS.*

Therefore, when $Ax^{SR'}$ holds, the algebraic structure of the PFU framework becomes just a totally ordered MCS $(E, \oplus, \otimes) = (E_u, \oplus_u, \otimes_u)$. The normalization condition imposed on environment components becomes

$$\underset{c}{\oplus}(\underset{P_i \in Fact(c)}{\otimes} P_i) = 1_E$$

and the operational answer to a query becomes

$$Ans(Q) = Sov((\underset{F_i \in F}{\wedge} F_i) \star (\underset{\varphi \in P \cup U}{\otimes} \varphi)) \tag{6.5}$$

Moreover, instead of expressing feasibilities on $\{t, f\}$, we can express them on $\{1_E, \Diamond\}$ by mapping t onto 1_E and f onto $\Diamond$. This preserves the value of the answer to a query since $t \star u = 1_E \otimes u$ and $f \star u = \Diamond \otimes u$. The answer $Ans(Q)$ to a query Q becomes $Ans(Q) = Sov(\otimes_{\varphi \in P \cup F \cup U} \varphi)$. As a result, answering a query in the semiring case can require several elimination operators (min, max, and $\oplus$), but it actually requires only one combination operator ($\otimes$).

Proposition 6.8. *Let $(E, \oplus, \otimes)$ be a totally ordered MCS. We extend $\oplus$ and $\otimes$ to $E \cup \{\Diamond\}$ by $u \oplus \Diamond = \Diamond \oplus u = u$ and $u \otimes \Diamond = \Diamond \otimes u = \Diamond$. Then, for every $op \in \{\min, \max, \oplus\}$, $(E \cup \{\Diamond\}, op, \otimes)$ is a commutative semiring.*

Corollary 6.9. *Let $(E, \oplus, \otimes)$ be a totally ordered MCS and let Φ be a set of scoped functions taking values in $E \cup \{\Diamond\}$. Then, for all variables x and for all $op \in \{\min, \max, \oplus\}$, $op_x\, (\otimes_{\varphi \in \Phi} \Phi) = (\otimes_{\varphi \in \Phi^{-x}} \varphi) \otimes (op_x \otimes_{\varphi \in \Phi^{+x}} \varphi)$*

Using Corollary 6.9, the algorithm in Figure 6.3 defines a generic VE algorithm when Ax^{SR} holds. The first call is **VE-answerQ**$(Sov, \otimes, P \cup F \cup U)$. This time, the factorization available in a PFU network is fully exploited, since when eliminating a variable x, only local functions involving x in their scope are considered. Complexity results on this algorithm are given in Section 6.5.

```
VE-answerQ($Sov, \circledast, \Phi$)
begin
    if $Sov = \emptyset$ then return $\Phi$
    else
        $Sov'.(op, S) \leftarrow Sov$
        choose $x \in S$
        if $S = \{x\}$ then $Sov \leftarrow Sov'$ else $Sov \leftarrow Sov'.(op, S - \{x\})$
        $\varphi_0 \leftarrow op_x\,(\circledast_{\varphi \in \Phi^{+x}} \varphi)$
        $\Phi \leftarrow (\Phi - \Phi^{+x}) \cup \{\varphi_0\}$
        return $VE\text{-}answerQ(Sov, \circledast, \Phi)$
end
```

Figure 6.3: A generic variable elimination algorithm using factorization (Sov: sequence of eliminations, $\circledast$: combination operator, Φ: set of scoped functions).

Proposition 6.10. ***VE-answerQ***$(Sov, \otimes, P \cup F \cup U)$ *returns a set of scoped functions Ψ such that $\otimes_{\psi \in \Psi} \psi = Ans(Q)$.*

6.4.2 Improved VE algorithm in the semigroup case

The definition of an improved variable elimination algorithm in the semigroup case requires a bit more work. In fact, Equation 6.4 page 93 does not create one new utility function resulting from the elimination of x. It creates one new plausibility function $\oplus_{p_x} P^{+x}$ which must be combined with all functions in U^{-x}, and one new utility function $\oplus_{u x} (P^{+x} \otimes_{pu} U^{+x})$. In other words, the global quantity obtained after the elimination of x is not formed as $Sov'(F' \star P' \otimes_{pu} U')$, where Sov' is the resulting sequence of eliminations and F', P', and U' are new sets of scoped functions.

A solution to recover a global form which does not vary during the elimination steps consists in working on pairs of plausibility-utility functions called *potentials* [91]. The definition introduced below however differ from the standard one. [3]

Definition 6.11. *A* potential *is a pair (P_0, U_0) composed of one plausibility function P_0 and one utility function U_0. Two operators are defined on plausibility-utility pairs:*

- *a combination operator $\boxtimes$ defined by $(p_1,u_1) \boxtimes (p_2,u_2) = (p_1 \otimes_p p_2, (p_1 \otimes_{pu} u_2) \otimes_u (p_2 \otimes_{pu} u_1))$,* [4]
- *an elimination operator $\boxplus$ defined by $(p_1,u_1) \boxplus (p_2,u_2) = (p_1 \oplus_p p_2, u_1 \oplus_u u_2)$.*

Last, a partial order on plausibility-utility pairs can be defined as "$(p,u_1) \preceq (p,u_2)$ iff $u_1 \preceq u_2$".

In the sequel, we also consider each feasibility function as a potential. Since there is only a partial order on plausibility-utility pairs (for example $\max((0.2,4),(0.6,3))$ does not exist), some technical steps are required to ensure that when a min- or a max-elimination on a decision variable x is being performed, there does not exist any potential whose plausibility part depends on x. These technical steps are addressed by Propositions 6.12 to 6.15, and lead us to the main result given in Proposition 6.16.

Proposition 6.12. *Let $\mathcal{N} = (V,G,P,F,U)$ be a PFU network. Then, there exists a PFU network $\mathcal{N}' = (V,G',P',F',U)$, which is called a* refinement *of $\mathcal{N}$, such that*

- *every component c in G' is included in one component of G, and the hypergraph having the variables in c as vertices and $\{sc(\varphi) \,|\, \varphi \in Fact(c)\}$ as a set of hyperedges is connected (to mean that variables in a component are somehow correlated);*
- *$\otimes_{p\,P_i \in P} P_i = \otimes_{p\,P_i \in P'} P_i$ and $\wedge_{F_i \in F} F_i = \wedge_{F_i \in F'} F_i$.*

Proposition 6.12 enables us to assume that all PFU networks considered are already refined, notably because the proof of Proposition 6.12 is constructive.

Given a set Φ of scoped functions, we slightly update the definitions of Φ^{+x}/Φ^{-x} by

$$\left\{\begin{array}{l} \Phi^{+x} = \{\varphi \in \Phi \,|\, x \in sc(\varphi)\} \cup \Phi_0 \\ \Phi^{-x} = \Phi - \Phi^{+x} \end{array}\right., \text{ where } \Phi_0 = \left\{\begin{array}{l} \Phi \cap Fact(c(x)) \text{ if } sc(\Phi) \cap c(x) \subset \{x\} \\ \emptyset \text{ otherwise} \end{array}\right.$$

Informally, the set Φ_0 added to $\{\varphi \in \Phi \,|\, x \in sc(\varphi)\}$ means that when x is the last variable of its component $c(x)$ to be eliminated (test $sc(\Phi) \cap c(x) \subset \{x\}$), we add in Φ^{+x} the scoped functions

[3] The notion of potentials introduced here differs from the one used in [91] for influence diagrams: in [91], potentials are combined using $(p_1,u_1) \boxtimes' (p_2,u_2) = (p_1 \times p_2, u_1 + u_2)$, and variable eliminations are performed by $\boxplus'_x(P,U) = (\sum_x P, \frac{\sum_x (P \times U)}{\sum_x P})$. Our proposal does not use any division operation, which is great since the structures manipulated are not assumed to be equipped with a division.

[4] The utility part of the obtained pair may be a bit surprising. In fact, p_1 informally corresponds to a plausibility which is already "integrated" in u_1 but not in all other utilities, hence the combination $p_1 \otimes_{pu} u_2$. Similarly, p_2 is already "integrated" in u_2 and must weigh all other utilities, hence the combination $p_2 \otimes_{pu} u_1$.

in $Fact(c(x))$ which are still in Φ. These added scoped functions are exactly the scoped functions in $Fact(c(x))$ whose scope is included in $pa_G(c(x))$. This technical step is required in order to use normalization conditions ensuring that some minimization and maximization operations on potentials are defined. Also, if $P_i \in Fact(c)$ for a component c, then the potential $(P_i, 1_u)$ is considered to be in $Fact(c)$ too.

The next propositions show that $Ans(Q)$ can be computed using potentials (Proposition 6.13), and that the global form obtained when working with potentials uses the factorizations and is unchanged during the elimination steps (Proposition 6.15).

Proposition 6.13. *Let $Q = (Sov, \mathcal{N})$ be a query on a PFU network $\mathcal{N}$, defined on a totally ordered EU structure satisfying Ax^{SG}. Let $T(Sov)$ be the sequence of operator-variable(s) pairs obtained from Sov by replacing $\oplus_u$ by $\boxplus$. Let Π be the set of potentials $\Pi = \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\}$. Then, for all assignments A of the free variables of Q,*

$$T(Sov)(\underset{\varphi\in\Pi}{\boxtimes}\varphi(A)) = \begin{cases} (1_p, Ans(Q)(A)) \text{ if } Ans(Q)(A) \neq \Diamond \\ \Diamond \text{ otherwise} \end{cases}$$

Lemma 6.14. *Let us consider a totally ordered EU structure satisfying Ax^{SG}. Then, for every set of potentials Π,*

- $\boxplus_x(\Pi) = \Pi^{-x} \boxtimes (\boxplus_x(\Pi^{+x}))$.
- *Assume that for all $(P_0, U_0) \in \Pi$, $x \notin sc(P_0)$. Then, $\max_x(\Pi)$ exists and $\max_x(\Pi) = \Pi^{-x} \boxtimes \max_x(\Pi^{+x})$.*[5] *Similarly, $\min_x(\Pi)$ exists and $\min_x(\Pi) = \Pi^{-x} \boxtimes \min_x(\Pi^{+x})$.*

Proposition 6.15. *Let $Q = (Sov, \mathcal{N})$ be a query on a PFU network $\mathcal{N} = (V, G, P, F, U)$ defined on a totally ordered EU structure satisfying Ax^{SG}, where $Sov = (op_1, S_1) \cdot (op_2, S_2) \cdots (op_k, S_k)$.*

Let $|Sov|$ denote the number of variables in Sov. Let $[x_{|Sov|}, \ldots, x_1]$ be a sequence of variables such that $(x_i \in S_j) \rightarrow (x_{i-1} \in S_j \cup S_{j+1})$.[6] *Let $op(x)$ denote the operator* min *if $x \in V_D$ and x is quantified with* min *in Sov,* max *if $x \in V_D$ and x is quantified with* max*, and $\boxplus$ otherwise.*

Let Π_1 be the initial set of potentials $\Pi_1 = \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\}$. For all $i \in \{1, \ldots, |Sov|\}$, let Π_{i+1} be the set of potentials defined from Π_i by:

$$\Pi_{i+1} = \begin{cases} \text{undefined if } \Pi_i \text{ is undefined or if } op(x_i)_{x_i}\Pi_i^{+x_i} \text{ does not exist} \\ (\Pi_i - \Pi_i^{+x}) \cup \{op(x_i)_{x_i}\,\Pi_i^{+x_i}\} \text{ otherwise} \end{cases}$$

Then, $\Pi_{|Sov|+1}$ is defined and $\boxtimes_{\varphi\in\Pi_{|Sov|+1}}\varphi = T(Sov)(\boxtimes_{\varphi\in\Pi}\varphi)$.

Proposition 6.15 shows that when eliminating a variable x on a set of potentials Π, with an elimination operator $op \in \{\min, \max, \boxplus\}$, only potentials having x in their scope need to be considered. After the elimination of x, one gets a new set of potentials $\Pi' = (\Pi - \Pi^{+x}) \cup \{op_x(\Pi^{+x})\}$. The condition "for all $(P_0, U_0) \in \Pi$, $x \notin sc(P_0)$" involved in Lemma 6.14, required when a decision variable is eliminated, is always satisfied during the elimination steps and entails that the partial order defined on potentials suffices to compute a min or a max when needed.

Eventually, the algorithm for the semigroup case is identical to the one used for the semiring case, except that the first call is **VE-answerQ**$(T(Sov), \boxtimes, \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\})$.

[5] Given a set of potentials Π, $\max_x(\Pi)$ does not necessarily exist since only a partial order is given on plausibility-utility pairs. For example, $\max((0.2, 4), (0.6, 3))$ does not exist.

[6] Informally, this means that the sequence $[x_{|Sov|}, \ldots, x_1]$ corresponds to a variable elimination order which can be used when considering the variables in an order "compatible" with *Sov*.

Proposition 6.16. ***VE-answerQ***$(T(Sov), \boxtimes, \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\})$ *returns a set of potentials* Π *such that* $\boxtimes_{\varphi \in \Pi} \varphi(A) = \begin{cases} (1_p, Ans(Q)(A)) \text{ if } Ans(Q)(A) \neq \Diamond \\ \Diamond \text{ otherwise} \end{cases}$

6.4.3 General case

The semiring and semigroup cases define two *sufficient* conditions allowing us to use the factorization into local plausibility, feasibility, and utility functions. Showing how *necessary* they are is still an open issue. It may occur that neither Ax^{SR}, nor Ax^{SG} holds.

Example 6.17. $(E_p, E_u, \oplus_u, \otimes_{pu}) = (\mathbb{R}^+, \mathbb{R}, +, \times)$ *is an EU structure defined on the plausibility structure* $(E_p, \oplus_p, \otimes_p) = (\mathbb{R}^+, +, \times)$ *and on the utility structure* $(E_u, \otimes_u) = (\mathbb{R}, \min)$*. It can be used to compute the expected utility of risks combined using* min*. It satisfies: (1) neither the semigroup axiom* Ax^{SG}*, since* $\oplus_u \neq \otimes_u$*; (2) nor the semiring axiom* Ax^{SR}*, because* $\otimes_u$ *does not distribute over* $\oplus_u$*: indeed, "*$\min(a, b+c) = \min(a,b) + \min(a,c)$*" does not always hold.*

As a result, cases exist for which the undecomposability problem, consisting of decomposing a quantity such as "$\oplus_{ux} (P^{+x} \otimes_{pu} U)$", is not solved. In those cases, it is as if there was a unique global utility function $U_0 = \otimes_{u U_i \in U} U_i$ whose factorization cannot be used. We can assume that $(E_p, \preceq_p) = (E_u, \preceq_u) = (E, \preceq)$, $\oplus_p = \oplus_u = \oplus$, and $\otimes_p = \otimes_{pu} = \otimes$ by using a transformation similar to the one performed for the semiring case. The quantity to compute then becomes

$$Ans(Q) = Sov((\underset{F_i \in F}{\wedge} F_i) \star (\underset{P_i \in P}{\otimes} P_i) \otimes U_0) = Sov(\underset{\varphi \in P \cup F \cup \{U_0\}}{\otimes} \varphi) \tag{6.6}$$

This means that the general case can be seen as a sub-case of the semiring case, at the price of aggregating all utility functions. Hence, algorithm **VE-answerQ** can still be used, with **VE-answerQ**$(Sov, \otimes, P \cup F \cup \{U_0\})$ as a first call.

Table 6.2 summarizes how the generic algorithm **VE-answerQ** can be used to answer PFU queries. Note that for each case, *no additional assumption is necessary on the PFU framework.* Only transformations of the initial problem into an equivalent one are required, such as the one induced by morphism $\phi : p \to p \otimes_{pu} 1_u$ when Ax^{SR} holds, or the one yielding a *refined* PFU network (cf. Proposition 6.12) when Ax^{SG} is satisfied.

CASE	FIRST CALL
semiring (Ax^{SR})	**VE-answerQ**$(Sov, \otimes, P \cup F \cup U)$
semigroup (Ax^{SG})	**VE-answerQ**$(T(Sov), \boxtimes, \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\})$
general case	**VE-answerQ**$(Sov, \otimes, P \cup F \cup \{U_0\})$, with $U_0 = \otimes_{u U_i \in U} U_i$

Table 6.2: Use of the generic variable elimination algorithm **VE-answerQ**.

6.4.4 Simplifying the problem specification in the semigroup case

As in the semiring case and for future discussion, let us reformulate the answer to a query in order to use only one set E and only two abstract operators $\oplus$ and $\otimes$, instead of having several sets (E_p and E_u) and several abstract operators ($\otimes_p$, $\otimes_u$, $\otimes_{pu}$, $\oplus_p$, $\oplus_u$). Behind this, the basic idea is to obtain a simplified structure, so that future generic algorithms become easier to define and easier to read. Let us consider axiom $Ax^{SG'}$ below:

$$Ax^{SG'} : \begin{cases} (E_p, \preceq_p) = (E_u, \preceq_u) = (E, \preceq) \\ \otimes_p = \otimes_{pu} = \otimes \\ \oplus_p = \oplus_u = \otimes_u = \oplus \end{cases}$$

The only difference between axioms $Ax^{SG'}$ and $Ax^{SR'}$ is that axiom $Ax^{SG'}$ postulates that $\otimes_u = \oplus$, whereas axiom $Ax^{SR'}$ postulates that $\otimes_u = \otimes$. The assumption "$(E_p, \preceq_p) = (E_u, \preceq_u) = (E, \preceq)$, $\oplus_p = \oplus_u = \oplus$, and $\otimes_p = \otimes_{pu} = \otimes$", which is common to the general case, the semiring case, and the semigroup case, can also be axiomatically justified using the Algebraic Expected Utility (AEU) theory recently introduced in [139]. This theory is a sub-case of Chu-Halpern's expected utility. In order to show the relation between Ax^{SG} and the simpler axiom $Ax^{SG'}$, we first introduce two propositions which enable us to deal with either only positive utility degrees, or only negative utility degrees, thanks to translation operations.

Proposition 6.18. *Let $S = (E_p, E_u, \oplus_u, \otimes_{pu})$ be a totally ordered EU structure. Let $E_u^+ = \{u \in E_u \mid u \succeq_u 0_u\}$. Let $\otimes_u^+$, $\oplus_u^+$, and $\otimes_{pu}^+$ denote the restrictions of $\otimes_u$, $\oplus_u$, $\otimes_{pu}$ on E_u^+ respectively. Similarly, let $E_u^- = \{u \in E_u \mid u \preceq_u 0_u\}$ and let $\otimes_u^-$, $\oplus_u^-$, and $\otimes_{pu}^-$ denote the restrictions of $\otimes_u$, $\oplus_u$, $\otimes_{pu}$ on E_u^-.*

Then, $(E_u^+, \otimes_u^+)$ is a utility structure and $S^+ = (E_p, E_u^+, \oplus_u^+ \otimes_{pu}^+)$ is a totally ordered EU structure, as well as $(E_u^-, \otimes_u^-)$ and $S^- = (E_p, E_u^-, \oplus_u^- \otimes_{pu}^-)$.

Proposition 6.19. *Let $S = (E_p, E_u, \oplus_u, \otimes_{pu})$ be a totally ordered EU structure satisfying Ax^{SG}. Let $\mathcal{N} = (V, G, P, F, U)$ be a PFU network defined on S, and let $Q = (Sov, \mathcal{N})$ be a query on $\mathcal{N}$.*

- *Assume that hypothesis (H^+) holds:*

  $$(H^+) : \forall (u_1, u_2) \in E_u^2,\ ((u_1 \preceq_u u_2) \rightarrow (\exists u_3 \succeq_u 0_u,\ u_2 = u_1 \otimes_u u_3)).$$

  Given a utility function φ, let $\varphi^- = \min\{\varphi(A) \mid A \in dom(sc(\varphi))\}$ and let $translate^+(\varphi)$ denote a function satisfying $\varphi = \varphi^- \otimes_u translate^+(\varphi)$ (such a function exists because of (H^+)). Let $\mathcal{N}^+ = (V, G, P, F, U^+)$, where $U^+ = \{translate^+(\varphi) \mid \varphi \in U\}$.

  Then, $\mathcal{N}^+$ is a PFU network on S^+, and $Q^+ = (Sov, \mathcal{N}^+)$ is a query which satisfies $Ans(Q) = Ans(Q^+) \otimes_u (\otimes_{u\varphi \in U} \varphi^-)$. Also, every policy optimal in Q^+ is also optimal in Q.

- *Similarly, assume that hypothesis (H^-) holds:*

  $$(H^-) : \forall (u_1, u_2) \in E_u^2,\ ((u_1 \preceq_u u_2) \rightarrow (\exists u_3 \preceq_u 0_u,\ u_1 = u_2 \otimes_u u_3)).$$

  Given a utility function φ, let $\varphi^+ = \max\{\varphi(A) \mid A \in dom(sc(\varphi))\}$ and let $translate^-(\varphi)$ denote a function satisfying $\varphi = \varphi^+ \otimes_u translate^-(\varphi)$ (such a function exists because of (H^-)). Let $\mathcal{N}^- = (V, G, P, F, U^-)$, where $U^- = \{translate^-(\varphi) \mid \varphi \in U\}$.

  Then, $\mathcal{N}^-$ is a PFU network on S^-, and $Q^- = (Sov, \mathcal{N}^-)$ is a query which satisfies $Ans(Q) = Ans(Q^-) \otimes_u (\otimes_{u\varphi \in U} \varphi^+)$. Also, every policy optimal in Q^- is also optimal in Q.

Proposition 6.19 says that as soon as hypothesis (H^+) or (H^-) holds, it is possible to deal with only positive utility degrees, or only negative utility degrees, i.e. to work on a non bipolar expected utility structure.

In the standard cases given in Table 6.1 page 93, either the structure is already non bipolar, or hypothesis (H^+) or (H^-) holds. To be more concrete, if $E_u = \mathbb{R}^+ \cup \{-\infty\}$ and φ is a utility function whose greatest value is 10, if suffices to transform φ into "$(\varphi - 10)$" and add 10 to the final result. Note however that there exists cases where neither (H^-) nor (H^+) holds, like in bipolar preference structures having an infinite positive utility together with an infinite negative utility [98]. In such cases, the utility scale cannot be translated.

We can now introduce the main proposition establishing a relation between Ax^{SG} and $Ax^{SG'}$. This proposition uses a non bipolarity assumption.

Proposition 6.20. *Let $S = (E_p, E_u, \oplus_u, \otimes_{pu})$ be a* non bipolar *and totally ordered EU structure.*

(a) If S satisfies $Ax^{SG'}$, then S satisfies Ax^{SG}.

(b) If S satisfies Ax^{SG}: let $E = E_u$ and $\oplus = \oplus_u$. If there exists an operator $\otimes$ on E and a function $\phi : E_p \to E$ such that

- *$\otimes$ is associative, commutative, monotonic, and distributive over $\oplus$,*
- *$\phi(p_1 \otimes_p p_2) = \phi(p_1) \otimes \phi(p_2)$, $\phi(p_1 \oplus_p p_2) = \phi(p_1) \oplus \phi(p_2)$, and $p \otimes_{pu} u = \phi(p) \otimes u$,*

then, for every query $Q = (Sov, \mathcal{N})$ on a PFU network $\mathcal{N} = (V, G, P, F, U)$ defined on S

- *$S' = (E, E, \oplus, \otimes)$ is a totally ordered EU structure with $(E, \oplus, \otimes)$ as a plausibility structure and $(E, \oplus)$ as a utility structure (the identity for $\otimes$ is $1_E = \phi(1_p)$, and its annihilator is $0_E = 0_u = \phi(0_p)$). Moreover, S' satisfies $Ax^{SG'}$;*
- *$\mathcal{N}' = (V, G, \{\phi(P_i) \mid P_i \in P\}, F, U)$ is a PFU network on S';*
- *$Q' = (Sov, \mathcal{N}')$ is a query on $\mathcal{N}'$ such that $Ans(Q) = Ans(Q')$ and such that the sets of optimal policies are the same with Q and Q'.*

Proposition 6.21. *$(E, E, \oplus, \otimes)$ is a totally ordered EU structure satisfying $Ax^{SG'}$ with $(E, \oplus, \otimes)$ as a plausibility structure and $(E, \oplus)$ as a utility structure iff $(E, \oplus, \otimes)$ is a totally ordered MCS.*

The two conditions on ϕ and $\otimes$ in Proposition 6.20(b) hold in all standard cases associated with the semigroup axiom: (1) for the probabilistic expected additive utility case (row 1 in Table 3.1 page 60), translated to $E_u = \mathbb{R}^- \cup \{-\infty\}$, $\phi = -id$ and $\otimes : (a, b) \to -a \cdot b$ fit; if we had $E_u = \mathbb{R}^+ \cup \{+\infty\}$, then $\phi = id$ and $\otimes = \times$ would fit; (2) for the possibilistic pessimistic expected utility (row 4 in Table 3.1), $\phi : p \to 1 - p$ and $\otimes = \max$ fit; (3) for the boolean pessimistic expected conjunctive utility (row 7 in Table 3.1), ϕ defined by $\phi(p) = \neg p$ and $\otimes = \vee$ fit; (4) for the boolean optimistic expected disjunctive utility (row 8 in Table 3.1), ϕ defined by $\phi = id$ and $\otimes = \wedge$ fit. In all these cases, Proposition 6.20 says that axioms Ax^{SG} and $Ax^{SG'}$ are in some sense equivalent.

This is why in the following, we assume that $Ax^{SG'}$ (and not Ax^{SG}) is satisfied. This assumption is not necessary to use algorithm **VE-answerQ**; it will be used later in Chapter 7. When $Ax^{SG'}$ holds, the computation to be performed, using only $\oplus$ and $\otimes$ as customizable operators, is:

$$Ans(Q) = Sov((\bigwedge_{F_i \in F} F_i) \star (\bigotimes_{P_i \in P} P_i) \otimes (\bigoplus_{U_i \in U} U_i)) \tag{6.7}$$

6.5 Quantifying the theoretical complexity via the constrained induced-width

After this small algebraic digression concerning axiom $Ax^{SG'}$, let us come back to algorithms. The previous section shows that it is possible to design a generic variable elimination algorithm in order to answer PFU queries. Most dedicated variable elimination approaches are actually specific versions of this generic algorithm, that is to say, they correspond to its instantiation to a specific expected utility structure. Section 6.5 gives upper bounds on the time and space complexities of this **VE-answerQ** algorithm, using a parameter called the constrained induced-width [66, 94]. These bounds hold for every formalism subsumed by the PFU framework.

6.5.1 Induced-width

The induced-width [35, 34] is a parameter defining an upper bound on the theoretical complexity of standard VE algorithms. It is also known as tree-width [115], k-tree number [2], or max-clique size -1. Given a mono-operator query on a graphical model (V, Φ), the induced-width is defined from the hypergraph $\mathcal{G} = (V, \{sc(\varphi) \mid \varphi \in \Phi\})$ associated with this graphical model.

Definition 6.22. *An elimination order o on a set of variables $V = \{x_1, \dots, x_n\}$ is a bijection from $\{1, \dots, n\}$ to V. For all $k \in \{1, \dots, n\}$, $o(k)$ is called the kth variable eliminated in o.*

An elimination order o induces a total order $\preceq$ on V, defined by $o(n) \prec \dots \prec o(2) \prec o(1)$, where $x \prec y$ means that y must be eliminated before x. This allows us to assimilate o to a total order on V.

Definition 6.23. *(Induced-width of an elimination order) Let $\mathcal{G} = (V_{\mathcal{G}}, H_{\mathcal{G}})$ be a hypergraph. Let o be an elimination order on $V_{\mathcal{G}}$. o can be used to induce a sequence of hypergraphs $\mathcal{G}_1, \dots, \mathcal{G}_{n+1}$ (where $n = |V_{\mathcal{G}}|$), defined by*

- $\mathcal{G}_1 = \mathcal{G}$
- *if $\mathcal{G}_k = (V_k, H_k)$ and x is the kth variable eliminated in o, then $\mathcal{G}_{k+1} = (V_k - \{x\}, (H_k - H_k^{+x}) \cup \{h_{k+1}\})$, where H_k^{+x} is the set of hyperedges in H_k involving variable x and $h_{k+1} = (\cup_{h \in H_k^{+x}} h) - \{x\}$ is the hyperedge created from step k to $k+1$ (variable elimination step).*

The induced-width of $\mathcal{G}$ under the elimination order o, *denoted $w_{\mathcal{G}}(o)$, is the maximum size of the created hyperedges, i.e. $w_{\mathcal{G}}(o) = \max_{k \in \{1, \dots, n\}} |h_{k+1}|$.*[7]

Informally, the hyperedge h_{k+1} created from step k to $k+1$ is obtained by considering the set H_k^{+x} of all hyperedges in $\mathcal{G}_k$ which "depend" on x and by "linking" all variables involved in H_k^{+x} except for x. This points out that the elimination of x creates a new scoped function of scope h_{k+1}. $1 + w_{\mathcal{G}}(o)$ corresponds to the maximum number of variables to simultaneously consider during the variable elimination steps.

Example 6.24. *Let us consider a CSP (V, C) where the set of variables is $V = \{x_1, x_2, x_3, x_4, x_5\}$ and the set of constraints is $C = \{c_{x_1,x_2}, c_{x_2,x_3}, c_{x_2,x_4}, c_{x_2,x_5}, c_{x_4,x_5}\}$. The hypergraph $\mathcal{G}$ associated with it is $\mathcal{G} = (V, H_{\mathcal{G}})$ where $H_{\mathcal{G}} = \{sc(c) \mid c \in C\} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_2, x_4\}, \{x_2, x_5\}, \{x_4, x_5\}\}$.*

[7] To be more formal, we should speak of the induced-width of the primal graph of $\mathcal{G}$, since the usual definition of the induced-width holds on graphs (and not on hypergraphs).

The induced-width of $\mathcal{G}$ under the elimination order $o_1 : x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$ equals 2. It is obtained by generating the sequence of hypergraphs introduced in Definition 6.23, as done in Figure 6.4. An induced-width of 2 means that at most $2 + 1 = 3$ variables must be considered simultaneously when using the elimination order o_1 to compute $\max_{x_1,x_2,x_3,x_4,x_5}(c_{x_1,x_2} \wedge c_{x_2,x_3} \wedge c_{x_2,x_4} \wedge c_{x_2,x_5} \wedge c_{x_4,x_5})$. The decompositions obtained graphically with the sequence of hypergraphs can also be algebraically described by a sequence of computations:

$$
\begin{aligned}
& \max_{x_1}\max_{x_2}\max_{x_3}\max_{x_4}\max_{x_5}(c_{x_1,x_2}\wedge c_{x_2,x_3}\wedge c_{x_2,x_4}\wedge c_{x_2,x_5}\wedge c_{x_4,x_5})\\
= & \max_{x_1}\max_{x_2}\max_{x_3}\max_{x_4}(c_{x_1,x_2}\wedge c_{x_2,x_3}\wedge c_{x_2,x_4}\wedge \underbrace{\max_{x_5}(c_{x_2,x_5}\wedge c_{x_4,x_5})}_{=c'_{x_2,x_4}\textit{(computation involving 3 variables)}})\\
= & \max_{x_1}\max_{x_2}\max_{x_3}(c_{x_1,x_2}\wedge c_{x_2,x_3}\wedge \underbrace{\max_{x_4}(c_{x_2,x_4}\wedge c'_{x_2,x_4})}_{=c'_{x_2}\textit{(computation involving 2 variables)}})\\
= & \max_{x_1}\max_{x_2}(c_{x_1,x_2}\wedge c'_{x_2}\wedge \underbrace{\max_{x_3}(c_{x_2,x_3})}_{=c''_{x_2}\textit{(computation involving 2 variables)}})\\
= & \max_{x_1}(\underbrace{\max_{x_2}(c_{x_1,x_2}\wedge c'_{x_2}\wedge c''_{x_2})}_{=c'_{x_1}\textit{(computation involving 2 variables)}})\\
= & \underbrace{\max_{x_1} c'_{x_1}}_{=c'_{\emptyset}\textit{(computation involving 1 variable)}}
\end{aligned}
$$

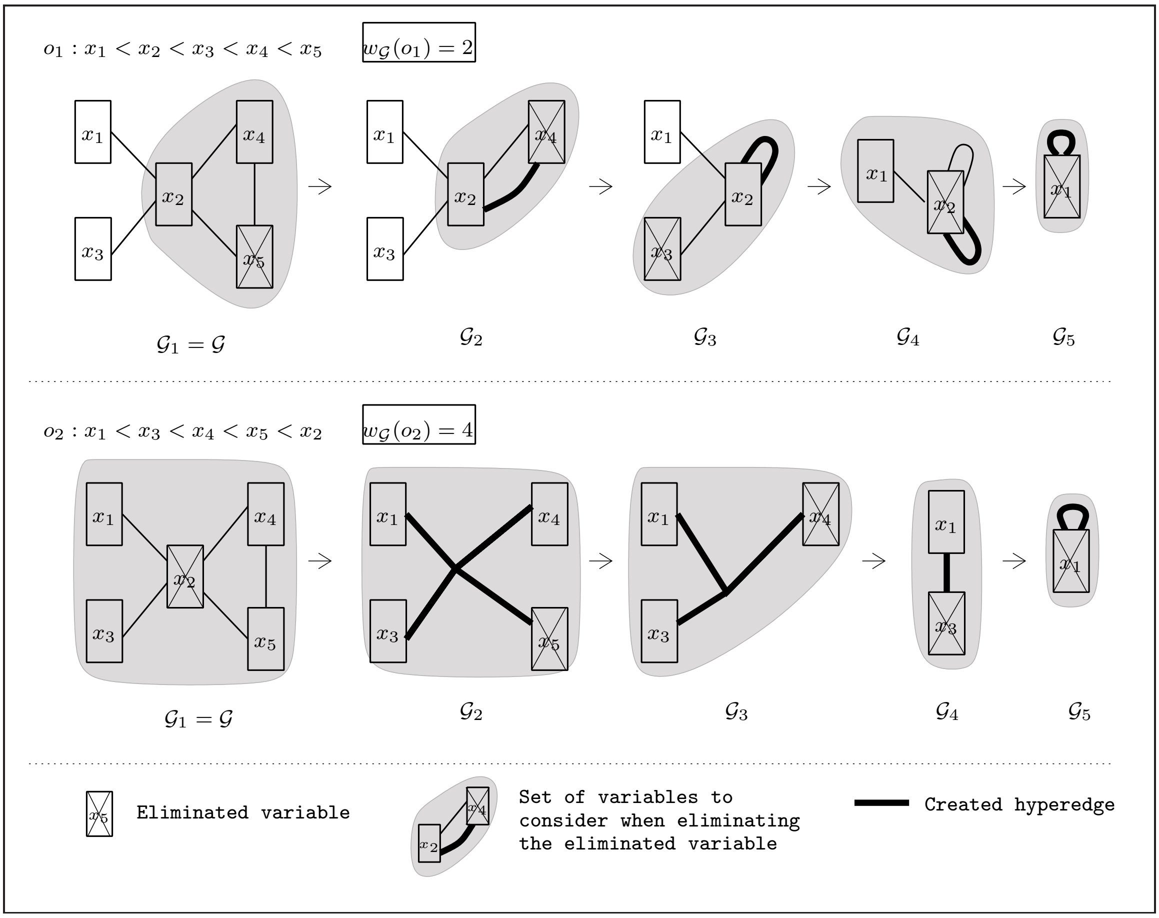

$o_1 : x_1 < x_2 < x_3 < x_4 < x_5$
$w_{\mathcal{G}}(o_1) = 2$
x_1
x_2
x_3
x_4
x_5
$\mathcal{G}_1 = \mathcal{G}$
$\mathcal{G}_2$
$\mathcal{G}_3$
$\mathcal{G}_4$
$\mathcal{G}_5$
$o_2 : x_1 < x_3 < x_4 < x_5 < x_2$
$w_{\mathcal{G}}(o_2) = 4$
Eliminated variable
Set of variables to consider when eliminating the eliminated variable
Created hyperedge

Figure 6.4: Illustration of the induced-width under an elimination order.

With this algebraic perspective, the induced-width under the elimination order o_1 is the maximum number of variables to simultaneously consider, minus 1*, i.e. the induced-width is* $3 - 1 = 2$*. In other words, the induced-width under the elimination order o_1 is the maximum scope size of the constraints c'_S created during the eliminations. The scopes of these constraints actually correspond to the hyperedges created when generating the sequence of hypergraphs.*

The induced-width of $\mathcal{G}$ under the elimination order $o_2 : x_1 \prec x_3 \prec x_4 \prec x_5 \prec x_2$ is $w_{\mathcal{G}}(o_2) = 4$. The successive hypergraphs obtained with o_2 are also shown in Figure 6.4.

The time and space complexities of a VE algorithm using an elimination order o on a graphical model (V, Φ) are known to be $O(|\Phi| \cdot d^{1+w_{\mathcal{G}}(o)})$, where $\mathcal{G}$ is the hypergraph associated with the graphical model. [8]

Definition 6.25. *(Induced-width of $\mathcal{G}$) Let $\mathcal{G} = (V_{\mathcal{G}}, H_{\mathcal{G}})$ be a hypergraph. The* induced-width of *$\mathcal{G}$, denoted $w_{\mathcal{G}}$, is the minimal induced-width under an elimination order on $V_{\mathcal{G}}$. In other words, if $\mathcal{O}$ denotes the set of all possible elimination orders on $V_{\mathcal{G}}$, then $w_{\mathcal{G}} = \min_{o \in \mathcal{O}} w_{\mathcal{G}}(o)$.*

The induced-width of a hypergraph is the minimal number of variables to simultaneously consider in a VE algorithm when using an optimal elimination order. The decision problem associated with the problem of finding an optimal elimination order is known to be NP-complete [2].

If only a subset S of $V_{\mathcal{G}}$ must be eliminated, as is the case when there are free variables, the definition of the induced-width of $\mathcal{G}$ for the elimination of the variables in S is similar. The only difference is that the sequence if hypergraphs stops when all variables in S have been eliminated. In the following, we consider that the set of variables to eliminate is implicit.

Example 6.26. *The induced-width of the hypergraph $\mathcal{G}$ associated with the CSP of the previous example can be shown to be $w_{\mathcal{G}} = 2$ (o_1 is an optimal elimination order).*

6.5.2 Constrained induced-width

In the multi-operator case however, there are constraints on the elimination order because the alternating elimination operators do not generally commute. The complexity can then be quantified using the *constrained induced-width* [66, 94].

Definition 6.27. *Let $\preceq$ be a partial order on V. The set of* linearizations *of $\preceq$, denoted $lin(\preceq)$, is the set of total orders $\preceq'$ on V satisfying $(x \preceq y) \rightarrow (x \preceq' y)$.*

Definition 6.28. *(Constrained induced-width) Let $\mathcal{G} = (V_{\mathcal{G}}, H_{\mathcal{G}})$ be a hypergraph and let $\preceq$ be a partial order on $V_{\mathcal{G}}$. The constrained induced-width $w_{\mathcal{G}}(\preceq)$ of $\mathcal{G}$ with constraints on the elimination order given by $\preceq$ ("$x \prec y$" stands for "y must be eliminated before x") is defined by $w_{\mathcal{G}}(\preceq) = \min_{o \in lin(\preceq)} w_{\mathcal{G}}(o)$.*

The constraints on the elimination order induced by the sequence of variable eliminations *Sov* can be formally defined.

[8] More precisely, when eliminating one variable x, $nbv \leq 1 + w_{\mathcal{G}}(o)$ variables are considered. For each of the d^{nbv} assignments of these variables, one must combine the values given by r scoped functions. In the end, the time complexity of a variable elimination step is $O(r \cdot d^{nbv}) \leq O(r \cdot d^{1+w_{\mathcal{G}}(o)})$. Summing on all the elimination steps can be shown to give a time complexity $O(|\Phi| \cdot d^{1+w_{\mathcal{G}}(o)})$ [78]. Similarly, the space complexity is $O(|\Phi| \cdot d^{1+w_{\mathcal{G}}(o)})$ too.

Definition 6.29. *Let $Q = (Sov, \mathcal{N})$ be a query on a PFU network such that $Sov = (op_1, S_1) \cdot (op_2, S_2) \cdots (op_q, S_q)$. The* partial order *$\preceq_{Sov}$ induced by Sov is given by $S_1 \prec_{Sov} S_2 \prec_{Sov} \ldots \prec_{Sov} S_q$. It forces variables in S_j to be eliminated before variables in S_i whenever $i < j$.*

For example, the partial order induced by the sequence of operator-variables pairs $Sov = \min_{x_1,x_2} \sum_{x_3,x_4} \max_{x_5}$ is defined by $\{x_1, x_2\} \prec_{Sov} \{x_3, x_4\} \prec_{Sov} x_5$.

The theoretical complexity of algorithm **VE-answerQ** can now be provided, using the constrained induced-width. Note that this complexity result holds for any formalism covered by the PFU framework.

Proposition 6.30. *Let $\mathcal{G} = (V, \Phi)$ be a graphical model. Let Sov be a sequence of operator-variable(s) pairs on V. If an induced-width optimal elimination order is used, algorithm **VE-answerQ**$(Sov, \circledast, \Phi)$ is time and space $O(|\Phi| \cdot d^{1+w_{\mathcal{G}}(\preceq_{Sov})})$, where d is the maximum domain size of the variables in V.*

Therefore, given a query $Q = (Sov, \mathcal{N})$ on a PFU network $\mathcal{N} = (V, G, P, F, U)$,

- answering a query in the semiring case is time and space $O(|P \cup F \cup U| \cdot d^{1+w_{\mathcal{G}}(\preceq_{Sov})})$, where $\mathcal{G} = (V, \{sc(\varphi) \,|\, \varphi \in P \cup F \cup U\})$ is the hypergraph associated with the PFU network;

- provided that condition (C): “$sc(\{\varphi \in Fact(c) \,|\, sc(\varphi) \subset pa_G(c)\}) \subset sc(\{\varphi \in Fact(c) \,|\, sc(\varphi) \not\subseteq pa_G(c)\})$” holds for every component c, answering a query in semigroup case is also time and space $O(|P \cup F \cup U| \cdot d^{1+w_{\mathcal{G}}(\preceq_{Sov})})$, where $\mathcal{G} = (V, \{sc(\varphi) \,|\, \varphi \in P \cup F \cup U\})$ is the hypergraph associated with the PFU network.

  Condition (C) is a technical point ensuring that the updating of the definition of Φ^{+x} in the semigroup case (for which $\Phi^{+x} = \{\varphi \in \Phi \,|\, x \in sc(\varphi)\} \cup \Phi_0$, where Φ_0 equals $\emptyset$ or $\Phi \cap Fact(c(x))$) does not change the constrained induced-width.[9] It can be shown that as soon as the plausibility structure satisfies “$(p \otimes_p p_1 = p \otimes_p p_2 = 1_p) \rightarrow (p_1 = p_2)$”, condition (C) can be enforced on every PFU network. This sufficient condition “$(p \otimes_p p_1 = p \otimes_p p_2 = 1_p) \rightarrow (p_1 = p_2)$” is satisfied in all standard plausibility structures;

- in the general case, answering a query is time and space $O((|P|+|F|+1) \cdot d^{1+w_{\mathcal{G}}(\preceq_{Sov})})$, where $\mathcal{G} = (V, \{sc(\varphi) \,|\, \varphi \in P \cup F \cup \{U_0\}\})$ is the hypergraph associated with the PFU network after merging all utility functions into a unique utility function U_0.

6.6 Decreasing the constrained induced-width

Since a linear variation of the constrained induced-width yields an exponential variation of the theoretical complexity, it is worth working on the two parameters $w_{\mathcal{G}}(\preceq_{Sov})$ depends on: the partial order $\preceq_{Sov}$ and the hypergraph $\mathcal{G}$.

[9] (C) enables us to assume without loss of generality that for every environment component c, the scope of $\oplus_{p_c} (\otimes_{p\, P_i \in Fact(c), sc(P_i) \cap c \neq \emptyset} P_i)$ contains the scope of each plausibility function $P_i \in Fact(c)$ such that $sc(P_i) \subset pa_G(c)$. Informally, (C) says that a parent must be “linked” with variables of its son components.

6.6.1 Weakening constraints on the elimination order

Weakening the partial order $\preceq_{Sov}$ induced by a sequence of eliminations Sov is known to be useless in contexts like Maximum A Posteriori hypothesis [94] on Bayesian networks, where there is only one alternation of max and sum marginalizations. But it can decrease the constrained induced-width as soon as there are more than two levels of alternation.

Indeed, let us consider a stochastic CSP (V, P, C) (cf Definition 2.21 page 30) where V is the sequence of variables $[x_1, \ldots, x_q, y, x_{q+1}]$, $P = \{P_y\}$ contains a probability distribution over y, the unique stochastic variable, and $C = \{c_{y,x_1}\} \cup \{c_{x_i,x_{q+1}} \mid i \in \{1, \ldots, q\}\})$ contains constraints c_S over sets of variables S. The PFU-representation of this problem is given in Figure 6.5. Solving this stochastic CSP is equivalent to computing

$$\max_{x_1,\ldots,x_q} \sum_y \max_{x_{q+1}} \left(P_y \times c_{y,x_1} \times \prod_{i\in\{1,\ldots,q\}} c_{x_i,x_{q+1}}\right).$$

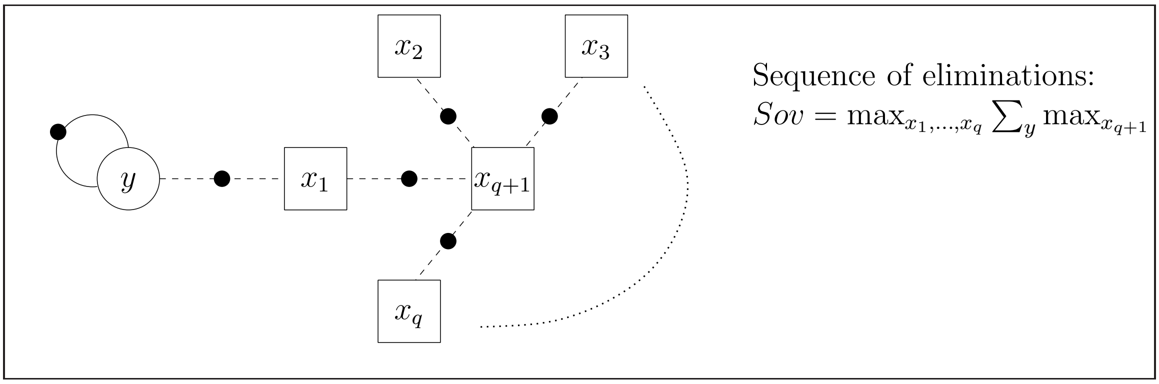

x_2
x_3
Sequence of eliminations:
$Sov = \max_{x_1,\ldots,x_q} \sum_y \max_{x_{q+1}}$
y
x_1
x_{q+1}
x_q

Figure 6.5: Stochastic CSP example.

If one uses $\mathcal{G} = (V_\mathcal{G}, H_\mathcal{G})$, with $V_\mathcal{G} = \{x_1, \ldots, x_{q+1}, y\}$ and $H_\mathcal{G} = \{sc(c) \mid c \in C\}$ together with $\preceq_1 = \preceq_{Sov}$ $(\{x_1, \ldots, x_q\} \prec_1 y \prec_1 x_{q+1})$, the constrained induced-width is $w_\mathcal{G}(\preceq_1) = q$, because $\preceq_1$ forces x_{q+1} to be eliminated first, which creates the hyperedge $\{x_1, \ldots, x_q\}$ of size q.

However, the scopes of the functions involved, and namely the fact that y is "linked" only with x_1, enable us to write the quantity to compute as

$$\max_{x_1} \left(\left(\sum_y P_y \times c_{y,x_1}\right) \times \left(\max_{x_2,\ldots,x_{q+1}} \left(\prod_{i\in\{1,\ldots,q\}} c_{x_i,x_{q+1}}\right)\right)\right).$$

This rewriting shows that the only actual constraint on the elimination order is that y must be eliminated before x_1. This constraint, modeled by $\preceq_2$ defined by $x_1 \prec_2 y$, gives $w_\mathcal{G}(\preceq_2) = 1$, for example with the elimination order $x_{q+1} \prec x_q \prec \ldots \prec x_2 \prec x_1 \prec y$. Hence, the complexity decreases from $O((q + 2) \cdot d^{1+q})$ to $O((q + 2) \cdot d^2)$ (there is a $(q + 2)$ factor because there are $q + 2$ scoped functions).

This example shows that defining constraints on the elimination order from the sequence of operator-variables Sov only is uselessly strong and may be exponentially suboptimal compared to a method considering the function scopes. In other words, it may be possible to reveal extra freedoms in the elimination order. It is also obvious that weakening constraints on the elimination order can only decrease the constrained induced-width:

Proposition 6.31. *If $\mathcal{G} = (V_\mathcal{G}, H_\mathcal{G})$ is a hypergraph and if $\preceq_1$, $\preceq_2$ are two partial orders on $V_\mathcal{G}$ such that $(x \preceq_2 y) \rightarrow (x \preceq_1 y)$ ($\preceq_2$ is weaker than $\preceq_1$), then $w_\mathcal{G}(\preceq_2) \leq w_\mathcal{G}(\preceq_1)$.*

6.6.2 Working on the hypergraph

Let us show how the constrained induced-width can be decreased by working on the hypergraph $\mathcal{G}$.

First, normalization conditions can be used in order to avoid some useless computations. For example, computing $\sum_x P_{x \mid pa(x)}$ is useless if $P_{x \mid pa(x)}$ denotes a conditional probability distribution of x given $pa(x)$. This means that x and the hyperedges associated with $P_{x \mid pa(x)}$ can be removed from the hypergraph $\mathcal{G}$.

Second, decompositions may exist which enable us to use more than just the distributivity of a combination operator $\otimes$ over an elimination operator $\oplus$. To illustrate this point, let us consider an influence diagram equivalent to the computation of $\max_{x_1,\ldots,x_q} \sum_y P_y \cdot \left(U_{y,x_1} + \cdots + U_{y,x_q}\right)$. Its PFU-representation is given in Figure 6.6 (left part).

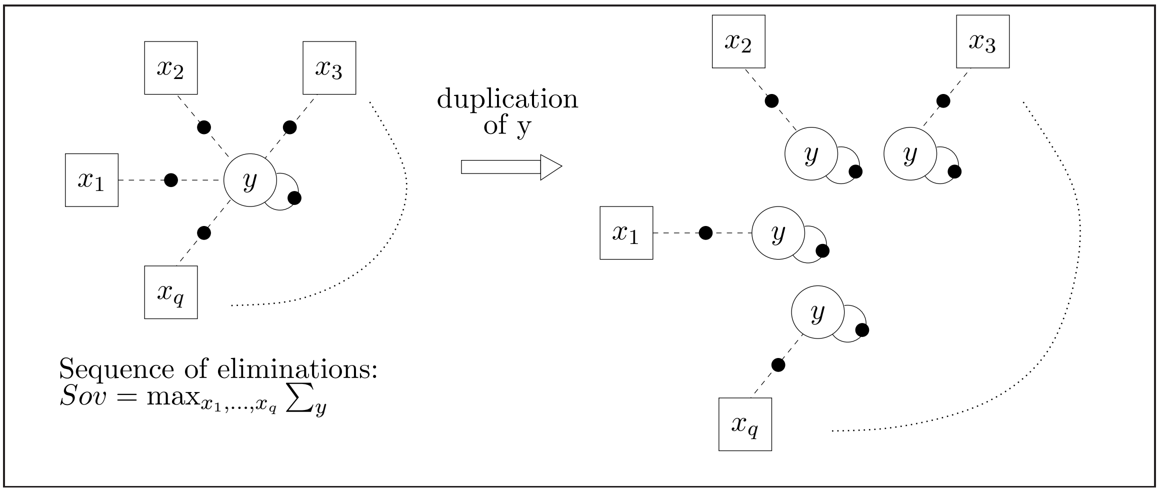


Figure 6.6: Influence diagram example (before and after duplication).

The basic hypergraph $\mathcal{G}_1 = (\{x_1,\ldots,x_q,y\},\{\{y\},\{y,x_1\},\ldots,\{y,x_q\}\})$, together with $\preceq_1$ defined by $\{x_1,\ldots,x_q\} \prec_1 y$, gives a theoretical complexity $O((q+1)\cdot d^{w_{\mathcal{G}_1}(\preceq_1)+1}) = O((q+1)\cdot d^{q+1})$. However, one can write:

$$\max_{x_1,\ldots,x_q} \sum_y P_y \cdot \left(U_{y,x_1} + \cdots + U_{y,x_q}\right) = (\max_{x_1} \sum_y P_y \cdot U_{y,x_1}) + \cdots + (\max_{x_q} \sum_y P_y \cdot U_{y,x_q})$$

Such an implicit repeated *duplication* of y makes the complexity decrease to $O(q \cdot d^2) = O(q \cdot d^{1+w_{\mathcal{G}_2}(\preceq_2)})$, where $\mathcal{G}_2$ is the hypergraph defined by the variables $\{x_1,\ldots,x_q,y^{(1)},\ldots,y^{(q)}\}$ and by the set of hyperedges $\{\{x_1,y^{(1)}\},\ldots,\{x_q,y^{(q)}\}\}$, and where $\preceq_2$ is given by $x_1 \prec_2 y^{(1)}$, ..., $x_q \prec_2 y^{(q)}$. This method, which uses the property $\sum_S (U_1 + U_2) = (\sum_S U_1) + (\sum_S U_2)$, duplicates variables “quantified” by $\sum$, so that computations become more local.

Another example where duplication is applicable is QCSP. For example, a QCSP equivalent to computing $\exists x_1 \ldots \exists x_q \forall y \left(\varphi_{x_1,y} \wedge \ldots \wedge \varphi_{x_q,y}\right)$ can also be written, after duplicating y, as $\exists x_1,\ldots,\exists x_q \left((\forall y_1 \varphi_{x_1,y_1}) \wedge \ldots \wedge \left(\forall y_q \varphi_{x_q,y_q}\right)\right)$. This makes the constrained induced-width decrease from q to 1.

Proposition 6.32 shows that such a duplication mechanism can be used only in one specific case, when the elimination operator is equal to the combination operator. This applies to eliminations with $\forall$ on QBFs and QCSPs, with min on possibilistic MDPs, or with + on influence diagrams.

Proposition 6.32. *Let $\circledast$ and $\odot$ be two operators such that $(E, \circledast)$ and $(E, \odot)$ are monoids. Then,*

$$(\circledast_x (\varphi_1 \odot \varphi_2) = (\circledast_x \varphi_1) \odot (\circledast_x \varphi_2) \text{ for all scoped functions } \varphi_1,\ \varphi_2) \leftrightarrow (\circledast = \odot).$$

When feasibilities are involved, the above result must be slightly updated.

Proposition 6.33. *Let $\circledast$ and $\odot$ be two operators such that $(E, \circledast)$ and $(E, \odot)$ are monoids. $\circledast$ and $\odot$ are extended to $E \cup \{\Diamond\}$ by $x \circledast \Diamond = \Diamond \circledast x = x$ and $x \odot \Diamond = \Diamond \odot x = \Diamond$.*

If $\circledast = \odot$ on E, then, for all scoped functions φ_1, φ_2 such that $(\varphi_1(A) = \Diamond) \leftrightarrow (\varphi_2(A) = \Diamond)$, $\circledast_x (\varphi_1 \odot \varphi_2) = (\circledast_x \varphi_1) \odot (\circledast_x \varphi_2)$.

This entails for example that if F_0 is a feasibility function, if U_1, U_2 are two real utility functions, then $\sum_x (F_0 \star (U_1 + U_2)) = (\sum_x (F_0 \star U_1)) + (\sum_x (F_0 \star U_2))$.

Proposition 6.34 proves that duplicating is always better than not.

Proposition 6.34. *Let ϕ_{x,S_i} be a scoped function of scope $\{x\} \cup S_i$ onto a set E for any $i \in [1, m]$. For all commutative and associative operator $\circledast$ on E, the direct computation of $\psi = \circledast_x (\phi_{x,S_1} \circledast \cdots \circledast \phi_{x,S_m})$ always requires more operations than the direct computation of $(\circledast_x \phi_{x,S_1}) \circledast \cdots \circledast (\circledast_x \phi_{x,S_m})$.*

Moreover, the direct computation of ψ results in a time complexity $O(m \cdot d^{1+|S_1 \cup \ldots \cup S_m|})$, whereas the direct computation of the m quantities in the set $\left\{\circledast_x \varphi_{x,S_j} \mid j \in \{1, \ldots, m\}\right\}$ is $O(m \cdot d^{1+\max_{j\in\{1,\ldots,m\}} |S_j|})$.

6.7 Summary

This chapter has introduced a generic variable elimination algorithm, called **VE-answerQ**, capable of answering PFU queries. This algorithm is able to benefit from the factorization into local functions as soon as one of the two disjoint decomposability axioms Ax^{SR} and Ax^{SG} is satisfied. Its use is summarized in Table 6.2 page 98, which shows that in the semiring case, its application is very natural, in the semigroup case, it requires the use of potentials, and in the general case, it requires to combine all utility functions into a unique global utility.

The principle of this algorithm is to eliminate variables in an order somehow compatible with the sequence *Sov* of multi-operator eliminations, and its time and space complexities are exponential in the constrained induced-width. Such an approach suffices to obtain the correct result, but, as shown in the last part of the chapter, does not take advantage of all the *actual structural features* of multi-operator queries:

1. First, defining constraints on the elimination order only from the sequence of operator-variable(s) pairs *Sov* can be restrictive, since reordering freedoms can appear if the scopes of the local functions involved are considered.

2. Second, algorithm **VE-answerQ** uses just the distributivity of a combination operator over elimination operators. But additional decompositions may exist based on the duplication mechanism mentioned earlier.

3. Third, PFU networks include some normalization conditions. These have not been used so far. Not using them can completely mask the real complexity of a problem.

Using the three previous mechanisms can lead to an improved constrained induced-width, and doing so to possible exponential gains in theoretical complexity. These statements lead us to introduce more advanced techniques able to reveal the actual structure of multi-operator queries.

Chapter 7

Structuring multi-operator queries

The constrained induced-width can be decreased and exponential gains in complexity obtained thanks to an accurate structural analysis of multi-operator queries. As previously mentioned, this analysis can bring to light freedoms in the elimination order, reveal some possible decompositions, and remove useless computations. The goal of this chapter is to systematize the structuration of multi-operator queries in a preprocessing step, and then to exploit it for the best in a new variable elimination algorithm.

It is important to note that the techniques we introduce are not just generalizations of existing methods defined in formalisms subsumed by the PFU framework. Thus, they contribute to all subsumed formalisms, including QBFs, stochastic SAT, extended-stochastic SAT, QCSPs, stochastic CSPs, probabilistic and possibilistic influence diagrams, or factored MDPs. This again shows the interest of defining generic algorithms in a generic algebraic framework.

As we shall see, structuration steps lead us to define a new generic computational architecture called the *multi-operator cluster DAG* architecture. The latter answers queries more efficiently than algorithm **VE-answerQ** introduced in the previous chapter, in terms of induced-width.

7.1 Back on the multi-operator queries considered

In the following, we consider that either $Ax^{SR'}$ or $Ax^{SG'}$ holds (cf. Chapter 6 pages 94 and 99; note that the general case is a sub-case of the semiring one, at the price of aggregating all utility functions). This is equivalent to assume that:

- Instead of having a plausibility structure, a utility structure, and an expected utility structure, we simply have one totally ordered MCS $(E, \oplus, \otimes)$ (cf Definition 6.6 page 94).
- The normalization conditions over environment components c of a PFU network (V, G, P, F, U) become $\oplus_c(\otimes_{P_i \in Fact(c)} P_i) = 1_E$.
- The operational answer to a query $Q = (Sov, (V, G, P, F, U))$ becomes:
  - $Ans(Q) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} P_i) \otimes (\otimes_{U_i \in U} U_i))$ in the semiring case ($Ax^{SR'}$),
  - $Ans(Q) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} P_i) \otimes (\oplus_{U_i \in U} U_i))$ in the semigroup case ($Ax^{SG'}$).

These three points exactly state the axioms which are assumed to hold in the following.

7.2 From queries to computation nodes

Before introducing the structuration process, we define new elements, called *computations nodes.* The introduction of such elements is motivated by the fact that the representation tools used so far prevent us from exploiting some mechanisms. To be more concrete, the duplication mechanism cannot be used on potentials, since in general $\boxplus_x(\pi_1 \boxtimes \pi_2) \neq (\boxplus_x \pi_1) \boxtimes (\boxplus_x \pi_2)$ even if $\oplus_{ux}(U_1 \otimes_u U_2) = (\oplus_{ux} U_1) \otimes_u (\oplus_{ux} U_2)$. We need to come back to a more basic representation enabling us to benefit from all algebraic properties.

Definition 7.1. *A* computation node *on a set E is:*

- *either a scoped function φ taking values in E (*atomic *computation node);*
- *or a triple $(sov, \circledast, N)$ such that $(E, \circledast)$ is a commutative monoid, N is a set of computation nodes, and sov is a sequence of operator-variables pairs involving operators op such that (E, op) is a commutative monoid.*

For example, if P_1, P_2 are two plausibility functions and if U_1, U_2 are two utility functions, then P_1, P_2, U_1, U_2 are atomic computation nodes. The triples $n_1 = (\sum_x, \times, \{P_1\})$ and $n_2 = (\sum_{y,z,t}, \times, \{P_2, U_2\})$ are also computation nodes, as well as $n_3 = (\min_q \max_r, +, \{n_1, n_2, U_1\})$. Informally, a computation node represents a computation to perform. This is made explicit by the definition of the value of a computation node.

Definition 7.2. *Let n be a computation node. The* value of n*, denoted $val(n)$, is defined by*

$$val(n) = \begin{cases} n \text{ if } n \text{ is atomic} \\ sov(\circledast_{n' \in N}\, val(n')) \text{ if } n = (sov, \circledast, N) \end{cases}$$

The set of variables eliminated by n*, denoted $V_e(n)$, is empty if n is atomic, and equals the set of variables appearing in sov if $n = (sov, \circledast, N)$.*

The scope of n*, denoted $sc(n)$, is defined by* $sc(n) = \begin{cases} sc(\varphi) \text{ if } n = \varphi \text{ is atomic} \\ (\cup_{n' \in N}\, sc(n')) - V_e(n) \text{ if } n = (sov, \circledast, N) \end{cases}$

The set of sons of n*, denoted $Sons(n)$, is a set of computation nodes which is empty if n is atomic, and which equals N if $n = (sov, \circledast, N)$.*

For example, the value of n_1 is $val(n_1) = \sum_x P_1$, the value of n_2 is $val(n_2) = \sum_{y,z,t}(P_2 \times U_2)$, and the value of n_3 is $val(n_3) = \min_q \max_r(val(n_1) + val(n_2) + U_1)$. Hence, a node $(sov, \circledast, N)$ defines a sequence of eliminations sov on a $\circledast$-combination of computation nodes. It can be represented as in Figure 7.1 as the root of a tree of computation nodes.

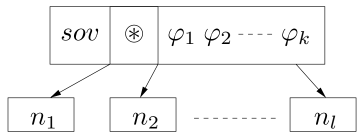


Figure 7.1: A computation node $(sov, \circledast, N)$, where $\{\varphi_1, \ldots, \varphi_k\}$ (resp. $\{n_1, \ldots, n_l\}$) is the set of atomic (resp. non-atomic) computation nodes in N.

We extend the previous definitions to sets of computation nodes N by $sc(N) = \cup_{n' \in N} sc(n')$, $V_e(N) = \cup_{n' \in N} V_e(n')$, and $Sons(N) = \cup_{n' \in N} Sons(n')$.

Moreover, for all $op \in \{\min, \max, \oplus\}$, we define the set of nodes in N performing eliminations only with op by $N[op] = \{n \in N \,|\, n = (op_S, \circledast, N')\}$. The set $N - N[op]$ is denoted $N[\neg op]$. For example, for $N = \{n_1, n_2, n_3\}$, we have $N[+] = \{n_1, n_2\}$ and $N[\neg +] = \{n_3\}$.

Finally, given a set of computation nodes N, we define N^{+x} (resp. N^{-x}) as the set of nodes in N whose scope contains x (resp. does not contain x): $N^{+x} = \{n \in N \,|\, x \in sc(n)\}$ (resp. $N^{-x} = \{n \in N \,|\, x \notin sc(n)\}$).

It is easy to express the answer to a query $Q = (Sov, (V, G, P, F, U))$ as the value of a computation node:

- In the semiring case, $Ans(Q) = val(n_0)$ where $n_0 = (Sov, \otimes, P \cup F \cup U)$.
- In the semigroup case, $Ans(Q) = val(n_0)$ where $n_0 = (Sov, \oplus, \{(\emptyset, \otimes, P \cup F \cup \{U_i\}), U_i \in U\})$. Indeed, $val(n_0) = Sov(\oplus_{U_i \in U}(\otimes_{\varphi \in P \cup F \cup \{U_i\}} \varphi)) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} P_i) \otimes (\oplus_{U_i \in U} U_i))$.

We also explicitly define the notion of elimination order compatible with a sequence of eliminations.

Definition 7.3. *An elimination order o over V is* compatible with *a sequence Sov over V iff $o \in lin(\preceq_{Sov})$. If $op(x)$ corresponds to the elimination operator of x in Sov, then $Sov(o)$ denotes the sequence of operator-variable ($o(k)$ is the kth variable eliminated in o):*

$$Sov(o) = op(o(n))_{o(n)} \cdots op(o(2))_{o(2)} \cdot op(o(1))_{o(1)}$$

Example 7.4. *Let $Sov = \min_{x_1,x_2} \sum_{x_3,x_4} \max_{x_5}$. The elimination order $o : x_1 \prec x_2 \prec x_4 \prec x_3 \prec x_5$ is compatible with Sov and $Sov(o) = \min_{x_1} \min_{x_2} \sum_{x_4} \sum_{x_3} \min_{x_5}$. The elimination order $o' : x_4 \prec x_2 \prec x_1 \prec x_3 \prec x_5$ is not compatible with Sov because $x_4 \prec x_2$ whereas $x_2 \prec_{Sov} x_4$.*

Towards a two-step structuration process

Exhibiting the query structure is equivalent to rewriting the initial computation node n_0 in order to reveal hidden structures. This is done thanks to a two-step structuration process:

1. We first seek the *macrostructure* of a multi-operator query. This corresponds to determine the actual freedoms in the elimination order and the possible decompositions (but not to determine an optimal elimination order).

   This macrostructure is obtained by using rewriting rules which *simulate* the decompositions induced by the variable eliminations from the right to the left of $Sov(o)$ for an elimination order o compatible with Sov. Rewriting rules $R : n_1 \rightsquigarrow n_2$ transform a computation node n_1 into another computation node n_2 denoted $n_2 = R(n_1)$. Their use may be restricted by preconditions. Three types of rewriting rules are used to get the macrostructure:

   - *decomposition rules*, which decompose the structure using the duplication technique;
   - *recomposition rules*, which reveal freedoms in the elimination order;
   - *simplification rules*, which remove useless computations from the architecture, thanks to normalization conditions.

2. Once the macrostructure is built, the second structuration step consists in exploiting the freedoms in the elimination order revealed by the first step. This will be done using cluster-tree decomposition techniques, enabling us to take advantage of finer structural features.

The structuration process differs between the semiring and semigroup cases, which do not have the same structural characteristics. We present the whole structuration for the semiring case first.

7.3 Structuring multi-operator queries in the semiring case

We here assume that there is no feasibility function since it simplifies the presentation greatly. The case with feasibilities is considered in Section 7.3.6. Also, in order for the rewriting rules to be more readable, computation nodes $(sov, \otimes, N)$ are written simply as (sov, N), because the combination operator of computation nodes is always $\otimes$ in the semiring case.

7.3.1 Building the macrostructure of a query using rewriting rules

Let o be an elimination order compatible with the sequence Sov of the query. The initial unstructured computation node is $n_0 = (Sov(o), \otimes, P \cup U)$, denoted $(Sov(o), P \cup U)$. This node can be seen as a *tree of Computation Nodes* (CNT) and is therefore denoted as $CNT_0(Q, o)$. In the example of Figure 7.2, $CNT_0(Q, o)$ is the first node. The application of rewriting rules generates a sequence of trees of computation nodes. For all $k \in \{0, \ldots, |Sov| - 1\}$, the macrostructure at step $k+1$, denoted $CNT_{k+1}(Q, o)$, is obtained from $CNT_k(Q, o)$ by considering the rightmost remaining elimination and by applying a decomposition rule DR and a recomposition rule RR:

1. Decomposition rule DR uses the distributivity of $\otimes$ over the elimination operators (so that when eliminating a variable x, only scoped functions having x in their scopes are considered), together with possible duplications. Rule DR implements both types of decompositions.

$$\boxed{DR} \qquad \left(sov.\underset{x}{op}, N \right) \rightsquigarrow \begin{cases} (sov, N^{-x} \cup \{(op_x, \{n\}) \mid n \in N^{+x}\}) \text{ if } op = \otimes \\ (sov, N^{-x} \cup \{(op_x, N^{+x})\}) \text{ otherwise} \end{cases}$$

   In Figure 7.2, DR transforms the initial structure $CNT_0(Q, o) = (\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4} \max_{x_5}, \{\varphi_{x_3,x_4}, \varphi_{x_1,x_4}, \varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})$ into $CNT_1(Q, o) = (\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4}, \{\varphi_{x_3,x_4}, \varphi_{x_1,x_4}, (\max_{x_5}, \{\varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})\})$ (case $op \neq \otimes$, using just the distributivity of $\wedge$ over max).

   Eliminating x_4 using min then transforms $CNT_1(Q, o)$ into $CNT_2(Q, o) = (\min_{x_1} \max_{x_2} \max_{x_3}, \{(\min_{x_4}, \{\varphi_{x_3,x_4}\}), (\min_{x_4}, \{\varphi_{x_1,x_4}\}), (\max_{x_5}, \{\varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})\})$ (case $op = \otimes = \min$, using a duplication of x_4). Note that in the semiring case, the duplication is actually usable iff op is idempotent. [1]

2. Recomposition rule RR aims at revealing freedoms in the elimination order for the nodes

[1] Indeed, assume that $op = \otimes$, with $op \in \{\min, \max, \oplus\}$. If $op = \oplus$, then, for all $e \in E$, $0_E \oplus e = 0_E \otimes e$, i.e. $e = 0_E$, hence $E = \{0_E\}$ and $op = \max = \otimes = \min$. If $op = \min$ or max, then it is also obviously idempotent.

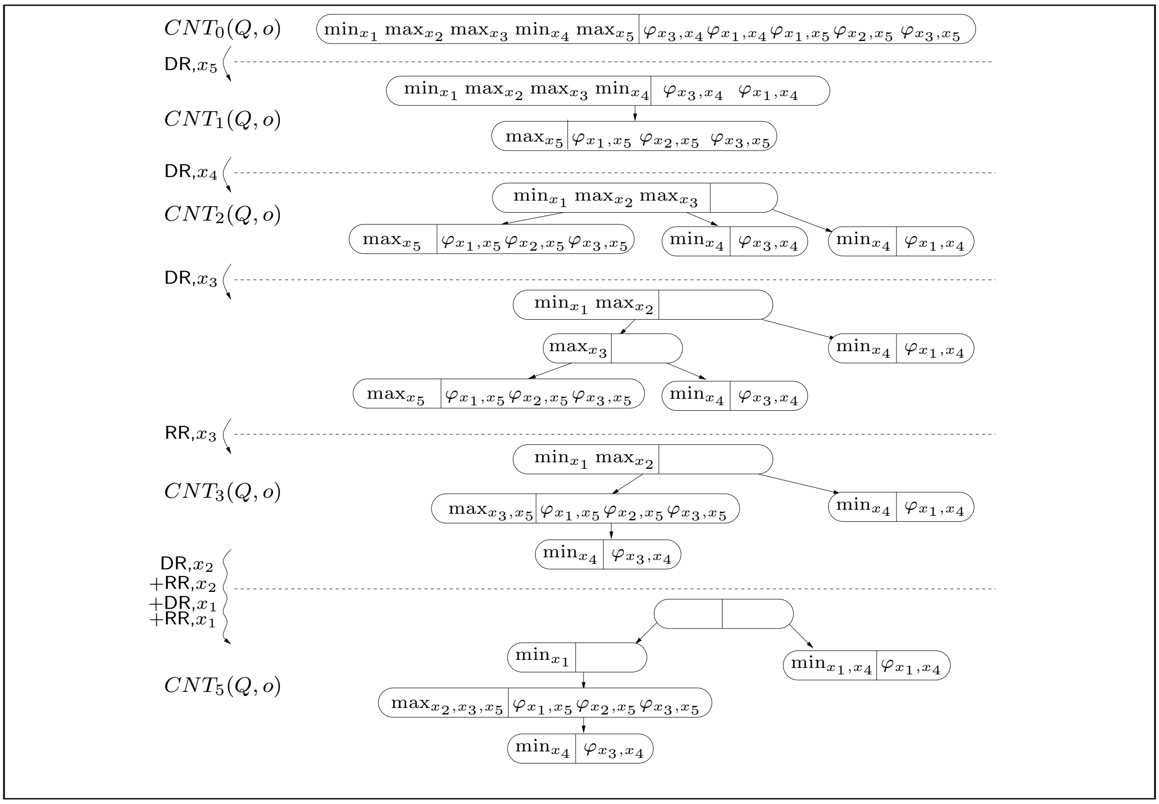

$CNT_0(Q,o)$
$\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4} \max_{x_5}$ | $\varphi_{x_3,x_4} \varphi_{x_1,x_4} \varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
DR,x_5
$\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4}$ | φ_{x_3,x_4} φ_{x_1,x_4}
$CNT_1(Q,o)$
$\max_{x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
DR,x_4
$\min_{x_1} \max_{x_2} \max_{x_3}$
$CNT_2(Q,o)$
$\max_{x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_3,x_4}
$\min_{x_4}$ | φ_{x_1,x_4}
DR,x_3
$\min_{x_1} \max_{x_2}$
$\max_{x_3}$
$\min_{x_4}$ | φ_{x_1,x_4}
$\max_{x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_3,x_4}
RR,x_3
$\min_{x_1} \max_{x_2}$
$CNT_3(Q,o)$
$\max_{x_3,x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_1,x_4}
$\min_{x_4}$ | φ_{x_3,x_4}
DR,x_2
+RR,x_2
+DR,x_1
+RR,x_1
$\min_{x_1}$
$\min_{x_1,x_4}$ | φ_{x_1,x_4}
$CNT_5(Q,o)$
$\max_{x_2,x_3,x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_3,x_4}

Figure 7.2: Application of the rewriting rules on a QCSP example: $\min_{x_1} \max_{x_2,x_3} \min_{x_4} \max_{x_5} (\varphi_{x_3,x_4} \wedge \varphi_{x_1,x_4} \wedge \varphi_{x_1,x_5} \wedge \varphi_{x_2,x_5} \wedge \varphi_{x_3,x_5})$, with the elimination order $o : x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$.

created by DR.

$$\boxed{RR} \qquad \left(\underset{x}{op}, N\right) \rightsquigarrow \left(\underset{\{x\}\cup V_e(N[op])}{op}, N[\neg op] \cup Sons(N[op])\right)$$

RR means that if a computation node performs an elimination op_x and has sons performing eliminations op_S with op too, then there is no reason to eliminate variables in S before x. RR makes it explicit by merging the corresponding computation nodes. In Figure 7.2, RR transforms the node $(\max_{x_3}, \{(\min_{x_4}, \{\varphi_{x_3,x_4}\}), (\max_{x_5}, \{\varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})\})$, created by $DR(CNT_2(Q,o))$, into $(\max_{x_3,x_5}, \{(\min_{x_4}, \{\varphi_{x_3,x_4}\}), \varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})$, which appears in $CNT_3(Q,o)$. In other words, RR reveals that although $x_3 \prec_{Sov} x_5$, there is actually no need to eliminate x_5 before x_3.

More formally, for all $k \in \{0, \ldots, |Sov| - 1\}$, the structure $CNT_{k+1}(Q,o)$ at step $k+1$ is obtained from the structure $CNT_k(Q,o)$ at step k by

$$CNT_{k+1}(Q,o) = rewrite(CNT_k(Q,o)) \tag{7.1}$$

where

$$rewrite((sov \cdot \underset{x}{op}, N)) = \begin{cases} (sov, N^{-x} \cup \{RR((op_x, \{n\})), n \in N^{+x}\}) \text{ if } op = \otimes \\ (sov, N^{-x} \cup \{RR((op_x, N^{+x}))\}) \text{ otherwise} \end{cases}$$

This means that when variable x is eliminated, we decompose the computations, using duplication if $op = \otimes$, and then recompose the created node(s) in order to reveal freedoms in the elimination order. In fact, function *rewrite* specifies explicitly an order in which rules must be applied because a chaotic iteration of the rules does not converge (for example, rules DR and RR may be infinitely alternately applied).

Given a query $Q = (Sov, \mathcal{N})$ and an elimination order o compatible with Sov, the final computation nodes tree obtained, denoted $CNT(Q, o)$, is

$$CNT(Q, o) = CNT_{|Sov|}(Q, o) = rewrite^{|Sov|}(CNT_0(Q, o))$$

also denoted as

$$CNT(Q, o) = rewrite^*(CNT_0(Q, o))$$

At each step, a non-duplicated variable appears exactly *once in the tree* and a duplicated one appears at most *once in each branch* of the tree.

Using normalization conditions We have not used so far normalization conditions such as $\oplus_c(\otimes_{P_i \in Fact(c)} P_i) = 1_E$ for every environment component c. These normalization conditions can allow useless computations to be removed from the architecture. That is why we introduce a simplification rule SR:

$$\boxed{SR} \quad \text{[Precond. : } (c \in \mathcal{C}_E(G)) \wedge (c \cap (S \cup sc(N)) = \emptyset)]$$
$$(\underset{S \cup c}{\oplus}, N \cup Fact(c)) \rightsquigarrow (\underset{S}{\oplus}, N)$$

For example, SR transforms a node $n = (\sum_{x,y,z}, \{P_{x \mid y,z}, P_y, P_z, c_y\})$, obtained e.g. when structuring a stochastic CSP, into a simplified node $n' = (\sum_{y,z}, \{P_y, P_z, c_y\})$ by using $\sum_x P_{x \mid y,z} = 1$. Applying SR again gives an even simpler computation node $n'' = (\sum_y, \{P_y, c_y\})$. SR cannot be applied again on n''. Intrinsically, although simplifications are available, they can remain undetected during the specification of a query because it can be difficult for a specifier to identify and use all available conditional independences.

Proposition 7.5. *Let $Q = (Sov, \mathcal{N})$ be a query and let $o \in lin(\preceq_{Sov})$. Let n be a computation node obtained during the construction of $CNT(Q, o)$.*

Then, SR cannot be applied an infinite number of times on n. Moreover, if n_1 and n_2 are two computation nodes obtained by applying SR as many times as possible on n, then $n_1 = n_2$.

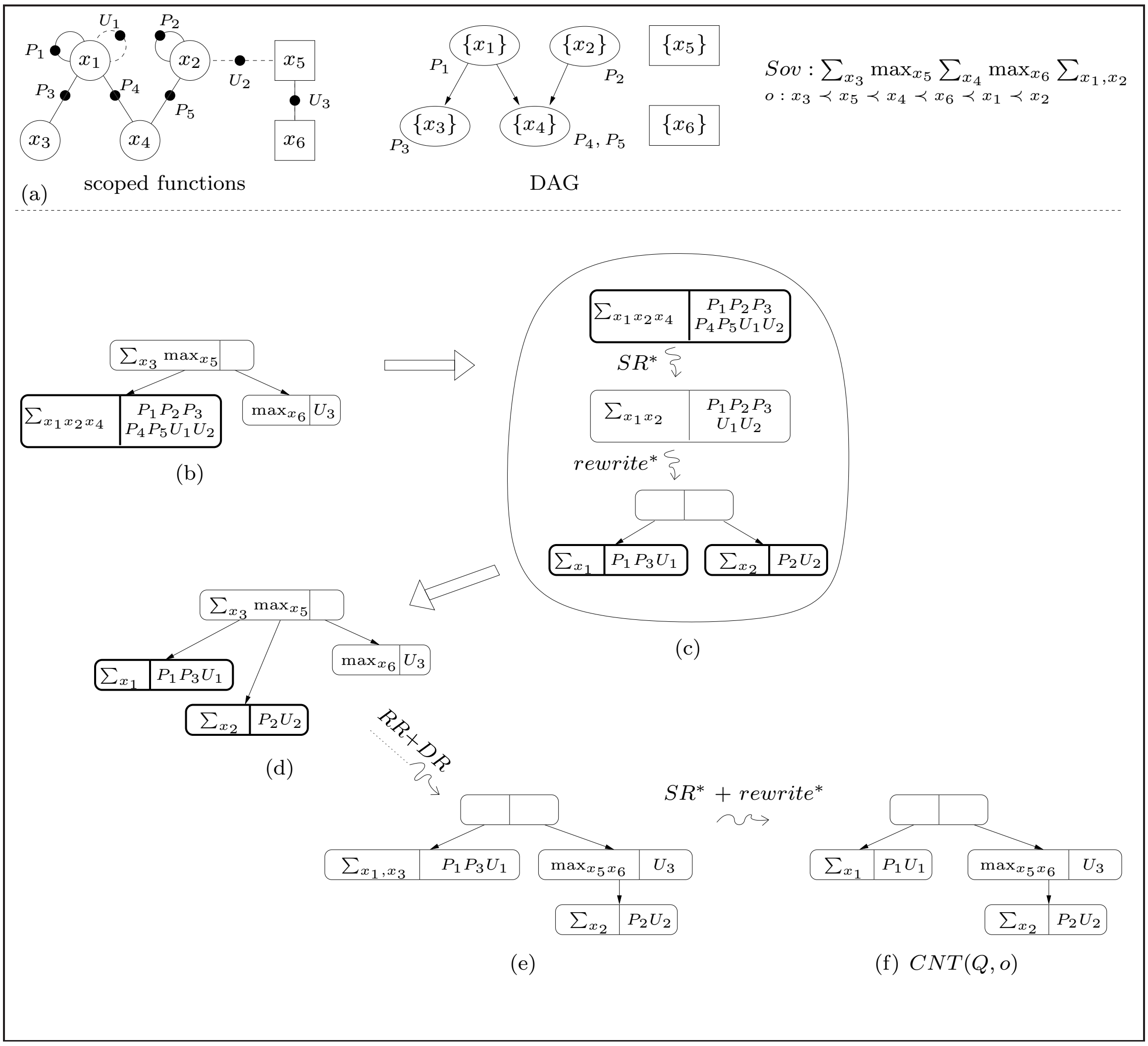
Proposition 7.5 shows that a recursive application of rewriting rule SR leads to a unique fixed point. In the following, this fixed point is denoted by $SR^*(n)$.

It is important to note that SR itself can reveal new decompositions and new reordering freedoms, as shown below.

Example 7.6. *Assume that $Ax^{SR'}$ holds with $\oplus = +$ and $\otimes = \times$. Let us consider the query $Q = (\sum_{x_3} \max_{x_5} \sum_{x_4} \max_{x_6} \sum_{x_1,x_2}, \mathcal{N})$, where $\mathcal{N} = (V, G, P, F, U)$ is the PFU network given in Figure 7.3(a). We use the elimination order $o : x_3 \prec x_5 \prec x_4 \prec x_6 \prec x_1 \prec x_2$. After applying DR and RR for $\sum_{x_2}$, $\sum_{x_1}$, $\max_{x_6}$, and $\sum_{x_4}$ successively, we obtain the macrostructure given in Figure 7.3(b).*

Using normalization condition $\sum_{x_4}(P_4 \cdot P_5) = 1$ on node $n = (\sum_{x_1,x_2,x_4}, \{P_1, P_2, P_3, P_4, P_5, U_1, U_2\})$ leads to the simplified node $SR^*(n) = (\sum_{x_1,x_2}, \{P_1, P_2, P_3, U_1, U_2\})$. It is then possible to rewrite $SR^*(n)$ itself, since it makes appear a new possible decomposition, as shown in Figure 7.3(c). This decomposition was hidden because x_4 created links between x_1 and x_2, which are actually completely unrelated. The computation node $rewrite^*(SR^*(n))$, equal to $(\emptyset, N')$, can be reintegrated to the global macrostructure by replacing $\{n\}$ by N', as done in Figure 7.3(d).

Applying rewriting rules DR and RR for the remaining eliminations $\max_{x_5}$ and $\sum_{x_3}$ leads to the macrostructure given in Figure 7.3(e), which can be simplified by replacing node $n' = (\sum_{x_1,x_3}, \{P_1, P_3, U_1\})$ by $n'' = (\sum_{x_1}, \{P_1, U_1\})$, thanks to the normalization condition $\sum_{x_3} P_3 = 1$. The final macrostructure obtained is given in Figure 7.3(f). We can say that this macrostructure was not obvious in the initial Sov sequence.



(a)
scoped functions
DAG
P_1 U_1 P_2 x_1 x_2 x_5 U_2 P_3 P_4 P_5 U_3 x_3 x_4 x_6
$\{x_1\}$ $\{x_2\}$ $\{x_5\}$ $\{x_3\}$ $\{x_4\}$ $\{x_6\}$ P_1 P_2 P_3 P_4, P_5
$Sov : \sum_{x_3} \max_{x_5} \sum_{x_4} \max_{x_6} \sum_{x_1,x_2}$
$o : x_3 \prec x_5 \prec x_4 \prec x_6 \prec x_1 \prec x_2$
(b)
$\sum_{x_3} \max_{x_5}$
$\sum_{x_1x_2x_4}$ $P_1P_2P_3$ $P_4P_5U_1U_2$
$\max_{x_6}$ U_3
(c)
SR^*
$\sum_{x_1x_2}$ $P_1P_2P_3$ U_1U_2
$rewrite^*$
$\sum_{x_1}$ $P_1P_3U_1$
$\sum_{x_2}$ P_2U_2
(d)
RR+DR
(e)
$\sum_{x_1,x_3}$ $P_1P_3U_1$
$\max_{x_5x_6}$ U_3
$SR^* + rewrite^*$
(f) $CNT(Q, o)$
$\sum_{x_1}$ P_1U_1

Figure 7.3: Macrostructuration of a query using simplification rule SR.

The use of rule SR is formalized as follows. We introduce a function *simplify* such that:

$$simplify((sov, N)) \quad = \quad (sov, \{n \in N \mid SR^*(n) = n\} \cup (\bigcup_{n \in N, SR^*(n) \neq n} Sons(rewrite^*(SR^*(n)))))$$

In other words, function *simplify* enables us to simplify some nodes using SR^* and to restructure them using $rewrite^*$, in order to make new decompositions appear in the simplified nodes. In the previous example, *simplify* transforms

$$(\sum_{x_3} \max_{x_5}, \{(\sum_{x_1,x_2,x_4}, \{P_1, P_2, P_3, P_4, P_5, U_1, U_2\}), (\max_{x_6}, \{U_3\})\})$$

into

$$(\sum_{x_3} \max_{x_5}, \{(\sum_{x_1}, \{P_1, P_3, U_1\}), (\sum_{x_2}, \{P_2, U_2\}), (\max_{x_6}, \{U_3\})\}),$$

i.e. it transforms the structure given in Figure 7.3(b) into the structure given in Figure 7.3(d).

Function *simplify* is applied after the treatment of each block of variables eliminated with $\oplus$, so that as many normalization conditions as possible can be used simultaneously.

More formally, we update the previous formulation given in Equation 7.1 by: for all $k \in \{0, \ldots, |Sov| - 1\}$,

$$CNT_{k+1}(Q, o) = \begin{cases} \textit{simplify}(rewrite(CNT_k(Q, o))) & \text{if } op(o(k+1)) = \oplus \neq op(o(k+2)) \\ rewrite(CNT_k(Q, o)) & \text{otherwise} \end{cases}$$

The tree of computation nodes obtained after these steps is still denoted $CNT(Q, o)$.

Some good properties of the final macrostructure obtained

Unicity Theorem 7.9 shows that the tree of computation nodes $CNT(Q, o)$ obtained given a query $Q = (Sov, \mathcal{N})$ and an elimination order o is actually independent from the arbitrary elimination order o compatible with Sov chosen at the beginning.

Lemma 7.7. *For all $op \in \{\min, \max, \oplus\}$, if $CNT = (sov \cdot op_x \cdot op_y, N)$ and $CNT' = (sov \cdot op_y \cdot op_x, N)$, then $rewrite^2(CNT) = rewrite^2(CNT')$.*

Lemma 7.8. *Given an elimination order $o \in lin(\preceq_{Sov})$, any elimination order $o' \in lin(\preceq_{Sov})$ can be obtained from o by successive permutations of adjacent eliminations.*

Theorem 7.9. *Let $Q = (Sov, \mathcal{N})$ be a query. Then, for all $o, o' \in lin(\preceq_{Sov})$, $CNT(Q, o) = CNT(Q, o')$.*

This allows us to denote $CNT(Q, o)$ simply as $CNT(Q)$.

Soundness The soundness of the created macrostructure, which has not been proved so far, is provided by Theorem 7.16. This theorem is preceded by preliminary lemmas which show that the rewriting process preserves nodes values.

Lemma 7.10. *Rewriting rule DR is sound, i.e. $val(DR(n)) = val(n)$ holds.*

Lemma 7.11. *Let $RR' : (op_S, N_1 \cup \{(op_{S'}, N_2)\}) \rightsquigarrow (op_{S \cup S'}, N_1 \cup N_2)$. If $S' \cap (S \cup sc(N_1)) = \emptyset$ and $N_1 \cap N_2 = \emptyset$, then RR' is a sound rewriting rule.*

Lemma 7.12. *Let $n = (op_x, N)$ be a computation node such that for all $(n_1, n_2) \in N^2$, $(n_1 \neq n_2) \rightarrow ((V_e(n_1) \cap V_e(n_2) = \emptyset) \wedge (V_e(n_1) \cap sc(n_2) = \emptyset))$, and such that $x \notin V_e(n)$ for all $n \in N$. Then $val(RR(n)) = val(n)$.*

Lemma 7.13. *Let $Q = (Sov, \mathcal{N})$ be a query and let $o \in lin(\preceq_{Sov})$. Let $k \in \{0, \ldots, |Sov|\}$ and let $n = (sov, N)$ be a computation node in $CNT_k(Q, o)$.*

Then, for all $(n_1, n_2) \in N[\neg\otimes]^2$, $(n_1 \neq n_2) \rightarrow ((V_e(n_1) \cap V_e(n_2) = \emptyset) \wedge (V_e(n_1) \cap sc(n_2) = \emptyset))$. Moreover, for all $n \in N$, $V_e(n) \cap V_e(CNT_k(Q, o)) = \emptyset$.

Lemma 7.14. *Rewriting rule SR is sound i.e. $val(SR(n)) = val(n)$ whenever its preconditions are satisfied.*

Lemma 7.15. *Let $Q = (Sov, \mathcal{N})$ be a query and let $o \in lin(\preceq_{Sov})$. Then, for all $k \in \{0, \ldots, |Sov|-1\}$, $val(CNT_{k+1}(Q, o)) = val(CNT_k(Q, o))$.*

Theorem 7.16. *Let $Q = (Sov, \mathcal{N})$ be a query. Then, $val(CNT(Q)) = Ans(Q)$.*

Complexity of the macrostructuration process The macrostructure is usable only if its computation is tractable. Based on the algorithm of Figure 7.4, which implements the macrostructuration of a query, Proposition 7.17 gives an upper bound on the complexity when simplification rule SR is not used. It shows that rewriting a query as a tree of mono-operator computation nodes is easy.

```
begin
    $root \leftarrow$ newNode$(\emptyset, \emptyset, P \cup U, \emptyset)$
    while $(sov = sov' \cdot \oplus_x)$ do
        $sov \leftarrow sov'$
        if $\oplus \neq \otimes$ then
            $n \leftarrow$ newNode$(\oplus, \{x\}, \emptyset, \emptyset)$
            foreach $n' \in Sons(root)$ s.t. $x \in sc(n')$ do
                $sc(n) \leftarrow sc(n) \cup sc(n')$
                $Sons(root) \leftarrow Sons(root) - \{n'\}$
                if $op(n') = \oplus$ then
                    $V_e(n) \leftarrow V_e(n) \cup V_e(n')$
                    $Sons(n) \leftarrow Sons(n) \cup Sons(n')$
                else $Sons(n) \leftarrow Sons(n) \cup \{n'\}$
            $sc(n) \leftarrow sc(n) - \{x\}$
            $Sons(root) \leftarrow Sons(root) \cup \{n\}$
        else
            foreach $n' \in Sons(root)$ s.t. $x \in sc(n')$ do
                if $op(n') = \oplus$ then
                    $V_e(n') \leftarrow V_e(n') \cup \{x\}$
                    $sc(n') \leftarrow sc(n') - \{x\}$
                else
                    $n \leftarrow$ newNode$(\oplus, \{x\}, \{n'\}, sc(n') - \{x\})$
                    $Sons(root) \leftarrow (Sons(root) - \{n'\}) \cup \{n\}$
    return $(root)$
end
```

Figure 7.4: MacroStruct$(sov, (V, P, U))$ (instruction newNode$(op, V_e, Sons, sc)$ creates a computation node $n = (op_{V_e}, Sons)$ and sets $sc(n)$ to sc.

In the algorithm of Figure 7.4, the root node of the tree of computation nodes is rewritten. With each node $n = (op_S, N)$ are associated an operator $op(n) = op$, a set of sons $Sons(n) = N$ modeled as a list, and a set of variables eliminated $V_e(n) = S$ modeled as a list too. The scope of n is modeled using a table of $|V|$ booleans. As long as the sequence of operator-variables is not empty, the rightmost remaining elimination is considered. The pseudo-code just implements the function *rewrite*, which dissociates the cases $\oplus \neq \otimes$ and $\oplus = \otimes$.

Proposition 7.17. *If the simplification rule is not used, the time and space complexities of the rewriting process in the semiring case are $O(|V|^2 \cdot |P \cup U|)$ and $O(|V| \cdot |P \cup U|)$ respectively (if $P \cup U \neq \emptyset$ and $V \neq \emptyset$).*

When SR is used, the complexity is still polynomial.[2]

Towards a second structuration step The macrostructure obtained is a tree of mono-operator computation nodes. We can now try to structure more finely the computations to be performed in each of these mono-operator nodes. To do so, cluster-tree decomposition techniques can be helpful.

7.3.2 Preliminaries: cluster-tree decompositions

Cluster-tree decomposition techniques are generic tools, used for example for CSPs or BNs, which exploit the topological properties of graphical models in order to split a problem into several smaller and easier to solve independent parts [116, 2, 115, 73, 13, 76]. They are designed for problems involving one combination operator and one elimination operator, which is the case of all individual mono-operator computation nodes obtained after the macrostructuration phase.

We adapt the usual definition of a cluster-tree decomposition [115] in order to deal directly with graphical models.

Definition 7.18. *A cluster-tree decomposition of a graphical model $\mathcal{M} = (V, \Phi)$ given a set of variables $S \subset V$ is a triple $(T, V(.), \Phi(.))$ where:*

- *$T = (C, E)$ is a tree.[3] Each $c \in C$ is called a cluster;*
- *$V(.)$ is a labeling function associating with each cluster c a set of variables $V(c)$ such that*
  - *$\cup_{c \in C} V(c) = V$;*
  - *for all $c_1, c_2, c_3 \in C$, if c_3 is on the path from c_1 to c_2, then $V(c_1) \cap V(c_2) \subset V(c_3)$; this is called the* running intersection property*;*
  - *there exists $c \in C$ such that $S \subset V(c)$;*
- *$\Phi(.)$ is a labeling function associating with each cluster c a set of scoped functions $\Phi(c)$ such that $\{\Phi(c) \,|\, c \in C\}$ is a partition of Φ and $sc(\varphi) \subset V(c)$ for every $\varphi \in \Phi(c)$.*

The width of a cluster-tree decomposition is $w = \max_{c \in C} |V(c)| - 1$. The tree-width of a graphical model $\mathcal{M}$ given S is the minimal width over all the cluster-tree decompositions of $\mathcal{M}$ given S.

[2] Indeed, in order to recursively apply SR on a computation node $(\oplus_S, N)$, we can first detect the set C of components c such that $Fact(c) \subset N$. This step is $O(|N|)$. Then, for each $c \in C$, we can test whether c can be removed by traversing N and S. This step is $O(|V| \cdot |N| + |S|) = O(|V| \cdot |N|)$. As there are lesser than $|V|$ components in the PFU network, an upper bound on the time needed for one recursive application of SR on $(\oplus_S, N)$ is $O(|V|^2 \cdot |N|) = O(|V|^2 \cdot |P \cup U|)$. As the root always has at most $|P \cup U|$ sons, each step of recursive application of SR on all sons of the root is $O(|V|^2 \cdot |P \cup U|^2)$, and therefore, as at most $|V|$ variables are eliminated in Sov, the application of SR during the rewriting process is $O(|V|^3 \cdot |P \cup U|^2)$. This bound is very naive and may be improved.

[3] C is the set of vertices of T and E is the set of edges of T.

A standard result concerning cluster-tree decompositions is that the tree-width of a graphical model $\mathcal{M}$ given S equals the induced-width of the hypergraph associated with $\mathcal{M}$ for the elimination of the variables in $V - S$. Therefore, seeking a cluster-tree decomposition with small width is equivalent to seeking an elimination order yielding a small induced-width.

Several methods to build cluster-tree decompositions exist. One of the most popular is based on graph triangulation techniques and proceeds as follows. Let G be the primal graph of the hypergraph $\mathcal{G} = (V, \{sc(\varphi), \varphi \in \Phi\} \cup \{S\})$ associated with a graphical model $\mathcal{M} = (V, \Phi)$ given S. If G is triangulated, i.e. if every cycle of length ≥ 4 has a chord, then it is easy to compute a cluster-tree decomposition $(T, V(.), \Phi(.))$ of $\mathcal{M}$ given S which has a minimal width. It suffices to perform the following steps: [4]

1. For each maximal clique of G, add a cluster c to the set of vertices of T and take $V(c)$ as the set of variables of the clique. This gives the set of clusters C and the labeling function $V(.)$.

2. Build the weighted undirected graph $G' = (C, E')$ for which there is an edge $\{c, c'\}$ of weight $-|V(c) \cap V(c')|$ in E' iff clusters c and c' share common variables.

3. In order to get the edges of $T = (C, E)$, build a *minimum spanning tree* of G', for example by using Prim's algorithm [110]:

   - create a set C_{tmp} containing one cluster $c \in C$
   - while $C_{tmp} \neq C$, choose an edge $\{c, c'\}$ in E' with a minimum weight, and such that $c \in C_{tmp}$ and $c' \notin C_{tmp}$. Add this edge to E and add c' to C_{tmp}.

4. Put each scoped function $\varphi \in \Phi$ in a unique cluster $c \in C$ satisfying $sc(\varphi) \subset V(c)$.

When G is not triangulated, one can first triangulate G and then build a cluster-tree decomposition of $\mathcal{M}$ given S based on the triangulated graph. Depending on the triangulation, the decomposition obtained may have a suboptimal width, and seeking a triangulation which gives an optimal width is NP-hard [2].

Example 7.19. *Consider the CSP (V, Φ) where $V = \{x_1, \ldots, x_{15}\}$ and $\Phi = \{\varphi_{x_1x_2}, \varphi_{x_1x_3}, \varphi_{x_2x_4}, \varphi_{x_3x_4}, \varphi_{x_4x_6}, \varphi_{x_5x_8x_9}, \varphi_{x_6x_7}, \varphi_{x_6x_{10}}, \varphi_{x_7x_8}, \varphi_{x_7x_{11}}, \varphi_{x_{10}x_{11}}, \varphi_{x_{10}x_{13}x_{14}}, \varphi_{x_{12}x_{13}}, \varphi_{x_{14}x_{15}}\}$. Let us compute a cluster-tree decomposition of this CSP given the set of variables $\{x_1, x_2\}$.*

The primal graph G of the hypergraph associated with this CSP is given in Figure 7.5(a). G is not triangulated because for example the cycle $x_1 \to x_2 \to x_4 \to x_3$ of length 4 is chordless. In order to triangulate G, we add the two dashed edges of Figure 7.5(b).

We then consider the set $\{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$ of maximal cliques of the triangulated graph. In order to get an associated cluster-tree decomposition, we first build the weighted undirected graph representing connected cliques, as in Figure 7.5(c). The weights are given by the number of common variables between two cliques. Second, we build a minimum spanning tree of this graph using Prim's algorithm. This provides us with the edges of the cluster-tree decomposition. Last, we associate each $\varphi \in \Phi$ with a cluster c satisfying $sc(\varphi) \subset V(c)$ and obtain the cluster-tree decomposition given in Figure 7.5(d).

[4] We assume that G is connected; if not, one cluster-tree decomposition can be built per connected component of G.

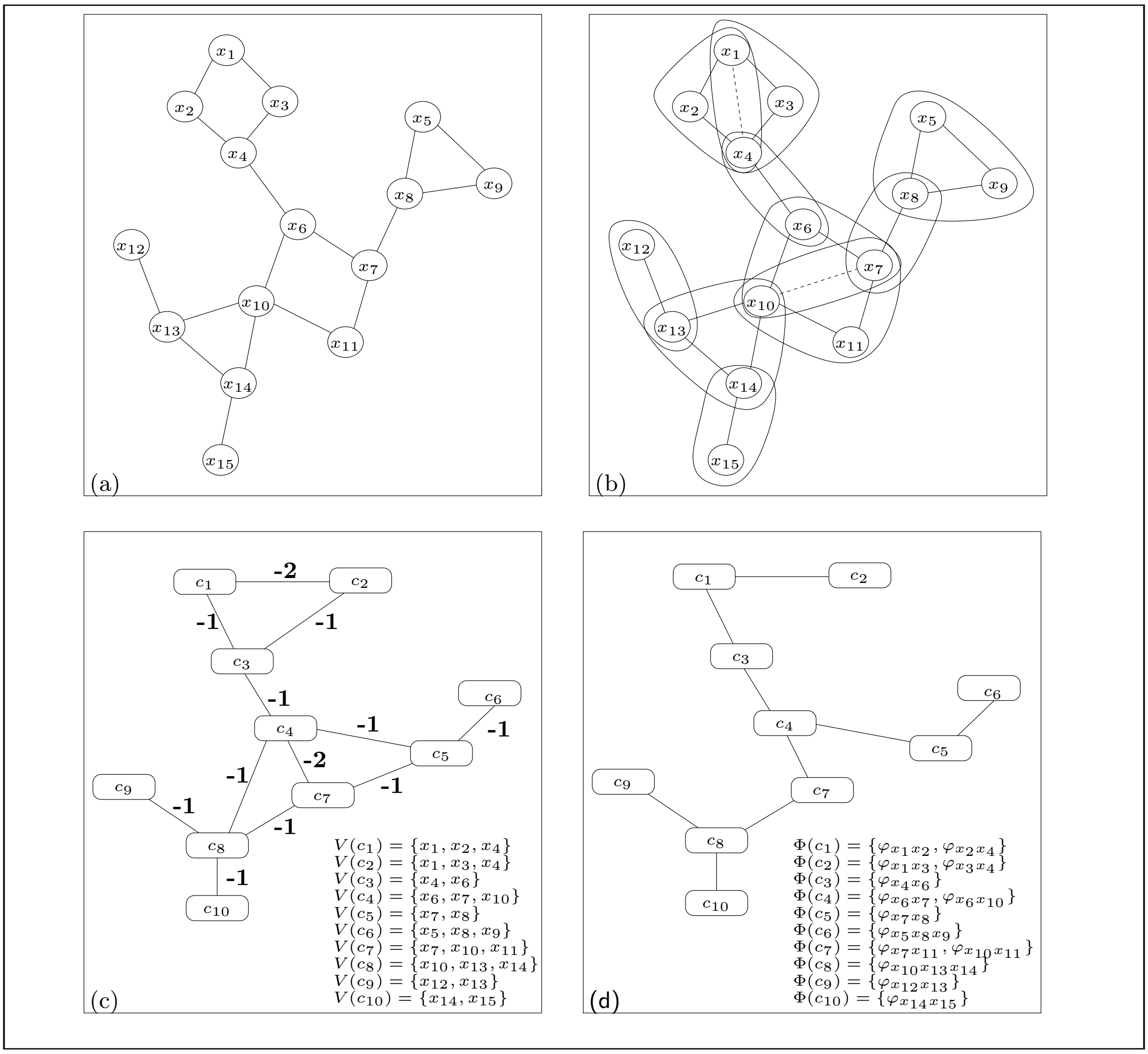

x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x_{11}
x_{12}
x_{13}
x_{14}
x_{15}
(a)
(b)
c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_8
c_9
c_{10}
-2
-1
(c)
$V(c_1) = \{x_1, x_2, x_4\}$
$V(c_2) = \{x_1, x_3, x_4\}$
$V(c_3) = \{x_4, x_6\}$
$V(c_4) = \{x_6, x_7, x_{10}\}$
$V(c_5) = \{x_7, x_8\}$
$V(c_6) = \{x_5, x_8, x_9\}$
$V(c_7) = \{x_7, x_{10}, x_{11}\}$
$V(c_8) = \{x_{10}, x_{13}, x_{14}\}$
$V(c_9) = \{x_{12}, x_{13}\}$
$V(c_{10}) = \{x_{14}, x_{15}\}$
(d)
$\Phi(c_1) = \{\varphi_{x_1x_2}, \varphi_{x_2x_4}\}$
$\Phi(c_2) = \{\varphi_{x_1x_3}, \varphi_{x_3x_4}\}$
$\Phi(c_3) = \{\varphi_{x_4x_6}\}$
$\Phi(c_4) = \{\varphi_{x_6x_7}, \varphi_{x_6x_{10}}\}$
$\Phi(c_5) = \{\varphi_{x_7x_8}\}$
$\Phi(c_6) = \{\varphi_{x_5x_8x_9}\}$
$\Phi(c_7) = \{\varphi_{x_7x_{11}}, \varphi_{x_{10}x_{11}}\}$
$\Phi(c_8) = \{\varphi_{x_{10}x_{13}x_{14}}\}$
$\Phi(c_9) = \{\varphi_{x_{12}x_{13}}\}$
$\Phi(c_{10}) = \{\varphi_{x_{14}x_{15}}\}$

Figure 7.5: Construction of a cluster-tree decomposition: (a) A primal graph; (b) Triangulation of the primal graph (dashed edges); (c) Undirected graph corresponding to the set of maximal cliques, where two cliques having k common variables are connected by an edge of weight $-k$; (d) Cluster-tree decomposition, obtained by building a minimum spanning tree of the undirected graph given in (c) and by assigning each scoped function to exactly one clique.

Cluster tree-decompositions are of interest from a computational point of view because they implicitly express computational decompositions:

Proposition 7.20. *Let $(T, V(.), \Phi(.))$ be a cluster-tree decomposition of a graphical model $\mathcal{M} = (V, \Phi)$ given a set of variables $S \subset V$, where the scoped functions in Φ take values in a commutative semiring $(E, \oplus, \otimes)$. Let r be a cluster of T such that $S \subset V(r)$. Let $Sons(c)$ denote the set of sons of a cluster c when T is rooted in r. Even if r has no parents, we take the convention $V(pa(r)) = S$. The value $val(c)$ of a cluster c is defined as $val(c) = \oplus_{V(c)-V(pa(c))}((\otimes_{\varphi\in\Phi(c)}\varphi) \otimes (\otimes_{s\in Sons(c)} val(s)))$. Then, $val(r) = \oplus_{V-S}(\otimes_{\varphi\in\Phi}\varphi)$.*

Proposition 7.20 is the key point showing the interest of cluster-tree decompositions. It says that $\oplus_{V-S}(\otimes_{\varphi\in\Phi}\varphi)$ can be computed by local computations on a root cluster r and its descendants.

This result holds thanks to the running intersection property. Moreover, cluster-tree decompositions induce a natural variable elimination algorithm [5] whose associated induced-width equals the width of the cluster-tree decomposition.

Example 7.21. *For the previous CSP example, Proposition 7.20 implies that* $\max_{x_3,\ldots,x_{15}}(\wedge_{\varphi\in\Phi}\varphi)$ *can be computed via the following local computations. We take* c_1 *as the root of the cluster-tree. Once a cluster* c *has received one value* $val(s)$ *per son* $s \in Sons(c)$ *in the rooted tree, it computes its own value* $val(c) = \oplus_{V(c)-V(pa(c))}((\otimes_{\varphi\in\Phi(c)}\varphi) \otimes (\otimes_{s\in Sons(c)} val(s)))$*. For example, at the beginning, cluster* c_9 *can compute* $val(c_9) = \max_{x_{12}} c_{x_{12}x_{13}}$ *and cluster* c_{10} *can compute* $val(c_{10}) = \max_{x_{15}} \varphi_{x_{14}x_{15}}$*. Then cluster* c_8 *can compute* $val(c_8) = \max_{x_{13},x_{14}}(\varphi_{x_{10},x_{13},x_{14}} \wedge val(c_9) \wedge val(c_{10}))$*. At the last step,* c_1 *computes* $val(c_1) = \max_{x_4}(\varphi_{x_1x_2} \wedge \varphi_{x_2x_4} \wedge val(c_2) \wedge val(c_3))$*.*

Proposition 7.20 ensures that $val(c_1) = \max_{x_3,\ldots,x_{15}}(\wedge_{\varphi\in\Phi}\varphi)$*. As the width of the cluster-tree decomposition is* $3-1=2$*, computing* $val(c_1)$ *is time and space* $O(|\Phi|\cdot d^3)$*. With a standard tree search, which does not exploit possible decompositions, the theoretical time complexity is* $O(|\Phi|\cdot d^{15})$*.*

7.3.3 Towards multi-operator cluster trees using cluster-tree decompositions

Let us come back to the macrostructure obtained after the macrostructuration process. The application of rewriting rules in the semiring case gives a tree of mono-operator computation nodes such as $(\min_S, \otimes, N)$, $(\max_S, \otimes, N)$, or $(\oplus_S, \otimes, N)$. Cluster-tree decomposition techniques can enable us to take advantage of the freedoms in the elimination order *inside* each of these mono-operator computation nodes.

More precisely, given a computation node $n = (op_S, \otimes, N)$, we can build a rooted cluster-tree decomposition of the graphical model $(sc(n), \{val(n'), n' \in N\})$ associated with it, given the variables in $sc(n) - S$ (which are not eliminated by n). This directly provides us with a structuration of $val(n)$ into local computations.

The structure obtained then contains both a macrostructure given by the computation nodes and an internal rooted cluster-tree structure given by each of their decompositions. It is called *multi-operator cluster tree.*

Definition 7.22. *A* Multi-operator Cluster Tree *(MCTree) is a rooted tree* (C, E) *with root* r*, where every vertex* $c \in C$*, called a cluster, is labeled with three elements:*

- *a set of variables* $V(c)$*,*
- *a set of scoped functions* $\Phi(c)$ *taking values in a set* E*,*
- *and a couple* $(\oplus^c, \otimes^c)$ *of operators on* E *such that* $(E, \oplus^c, \otimes^c)$ *is a commutative semiring.*

The width of a MCTree is defined as $w = \max_{c\in C}|V(c)| - 1$*.*

We explicitly specify a combination operator and an elimination operator to be used inside each cluster. This allows us to properly handle the multi-operator nature of multi-operator queries.

Figure 7.6 shows an example of MCTree which can be obtained from an Extended-SSAT [82] problem.

[5] We also speak of variable elimination algorithms when sets of variables must be eliminated. Such algorithms are also called non-serial dynamic programming or cluster-tree elimination algorithms.

Definition 7.23. *The value of a cluster c of a MCTree is given by*

$$val(c) = \bigoplus^{c}_{V(c)-V(pa(c))} \left(\left(\bigotimes^{c}_{\varphi \in \Phi(c)} \varphi \right) \otimes^{c} \left(\bigotimes^{c}_{s \in Sons(c)} val(s) \right) \right)$$

The value of a MCTree is the value of its root node.

Theorem 7.24. *Let Q be a query. Let M be a MCTree obtained from $CNT(Q)$. Then, $val(M) = Ans(Q)$. Moreover, every optimal decision rule in $val(M)$ for a non-duplicated decision variable is also an optimal decision rule in $Ans(Q)$, and for every duplicated decision variable, there exists at least one optimal decision rule in $val(M)$ which is also optimal in $Ans(Q)$.*

In fact, optimal decision rules can be recorded on the separators of the MCTree (the separator between two clusters c and $s \in Sons(c)$ is $V(c) \cap V(s)$).

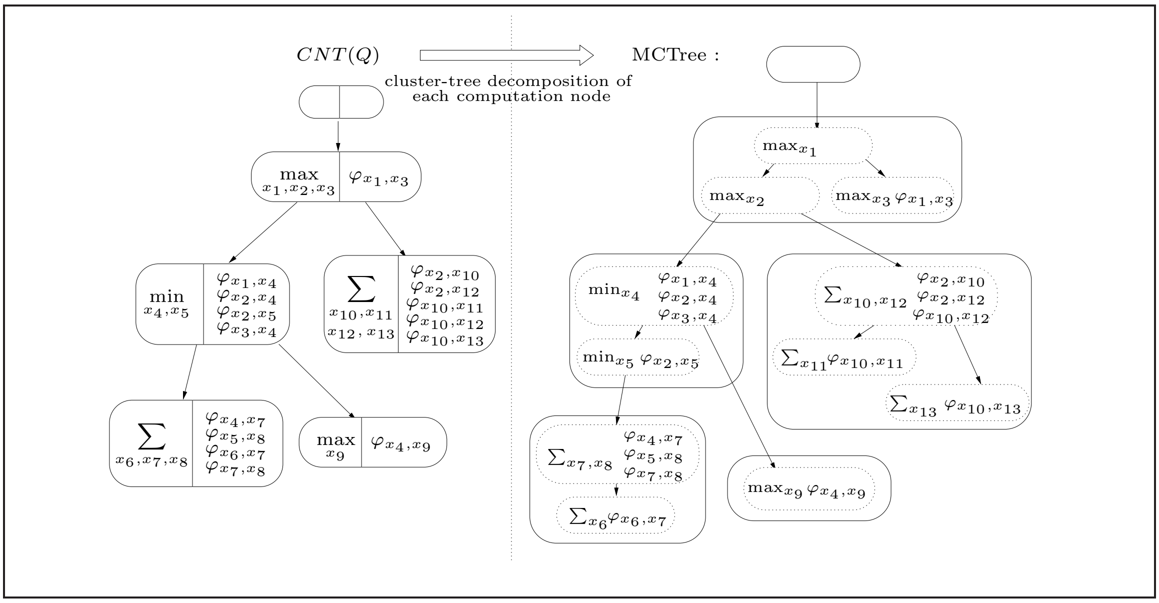

$CNT(Q)$
cluster-tree decomposition of each computation node
MCTree :
$\max_{x_1,x_2,x_3}$ φ_{x_1,x_3}
$\min_{x_4,x_5}$ φ_{x_1,x_4} φ_{x_2,x_4} φ_{x_2,x_5} φ_{x_3,x_4}
$\sum_{x_{10},x_{11},x_{12},x_{13}}$ $\varphi_{x_2,x_{10}}$ $\varphi_{x_2,x_{12}}$ $\varphi_{x_{10},x_{11}}$ $\varphi_{x_{10},x_{12}}$ $\varphi_{x_{10},x_{13}}$
$\sum_{x_6,x_7,x_8}$ φ_{x_4,x_7} φ_{x_5,x_8} φ_{x_6,x_7} φ_{x_7,x_8}
$\max_{x_9}$ φ_{x_4,x_9}
$\max_{x_1}$
$\max_{x_2}$
$\max_{x_3}\varphi_{x_1,x_3}$
$\min_{x_4}$ φ_{x_1,x_4} φ_{x_2,x_4} φ_{x_3,x_4}
$\min_{x_5}\varphi_{x_2,x_5}$
$\sum_{x_{10},x_{12}}$ $\varphi_{x_2,x_{10}}$ $\varphi_{x_2,x_{12}}$ $\varphi_{x_{10},x_{12}}$
$\sum_{x_{11}}\varphi_{x_{10},x_{11}}$
$\sum_{x_{13}}\varphi_{x_{10},x_{13}}$
$\sum_{x_7,x_8}$ φ_{x_4,x_7} φ_{x_5,x_8} φ_{x_7,x_8}
$\sum_{x_6}\varphi_{x_6,x_7}$
$\max_{x_9}\varphi_{x_4,x_9}$

Figure 7.6: Example of a MCTree obtained from $CNT(Q)$. Note that a cluster c is represented by (1) the set $V(c) - V(pa(c))$ of variables it eliminates, its elimination operator $\oplus^c$, and the set of functions $\Phi(c)$ associated with it, all these elements being put in a dotted box; in the semiring case, we always have $\otimes^c = \otimes$; (2) the set of its sons.

As a conclusion, the multi-operator query macrostructuration and the use of cluster-tree decompositions yield a generic computational architecture called MCTree. Note that if duplicated variables are relabeled, the MCTrees obtained satisfy the running intersection property (cf. Definition 7.18 page 118).

7.3.4 Comparison with an unstructured approach

Analyzing the query structure can induce exponential gains in theoretical complexity, as shown on some examples introduced in Section 6.6. A stronger result can be stated, proving that in terms of induced-width, the structured approach is *always as least as good* as the approach used in algorithm **VE-answerQ**.

Definition 7.25. *The width of a tree of computation nodes CNT, denoted w_{CNT}, is the minimal width over all MCTrees which can be obtained by cluster-tree decomposing CNT.*

Proposition 7.26. *Let $Q = (Sov, (V, G, P, \emptyset, U))$ be a query. Computing $Ans(Q)$ with a variable elimination algorithm on an optimal MCTree associated with Q is time and space $O(|P \cup U| \cdot d^{1+w_{CNT(Q)}})$.*

One can say that $1 + w_{CNT}$ is the maximum number of variables to consider simultaneously when using optimal cluster-tree decompositions for each computation node in CNT. Note that optimizing the cluster-tree decomposition of each computation node is stronger than optimizing the width of the MCTree alone. Also, one can use parameters which differ from the width to evaluate the quality of cluster-tree decompositions (more details in the next chapter).

Theorem 7.27 shows that the structuration mechanisms previously introduced can only decrease the induced-width (or tree-width). This implies that the theoretical complexity of a variable elimination algorithm on MCTrees is better than the complexity of algorithm **VE-answerQ**.

Theorem 7.27. *Let $Q = (Sov, \mathcal{N})$ be a query on a PFU network $\mathcal{N} = (V, G, P, \emptyset, U)$. Let $\mathcal{G} = (V, \{sc(\varphi), \varphi \in P \cup U\})$ be the hypergraph associated with $\mathcal{N}$. Then, $w_{CNT(Q)} \leq w_{\mathcal{G}}(\preceq_{Sov})$.*

For the QCSP example in Figure 7.2, $w_{CNT(Q)} = 1$, whereas the initial constrained induced-width is $w_{\mathcal{G}}(\preceq_{Sov}) = 3$: the complexity decreases from $O(|\Phi| \cdot d^4)$ to $O(|\Phi| \cdot d^2)$.

More important gaps between $w_{CNT(Q)}$ and $w_{\mathcal{G}}(\preceq_{Sov})$ can be observed on larger problems. We performed experiments on instances of the QBF library.[6] The results are shown in Table 7.1. In order to compute widths and constrained induced-widths, we built cluster-tree decompositions using the so-called *min-fill* heuristic. The results show that for low numbers of elimination operator alternations, analyzing the macrostructure of queries brings no gain. It is the case with instances of the "robot" problem, which involve only three alternations of elimination operators. But as soon as the number of alternations increases, revealing freedoms in the elimination order can be greatly beneficial.

Problem instance	w	w'	nbv,nbc,nba	Problem instance	w	w'	nbv,nbc,nba
adder-2-sat	12	24	332, 113, 5	k-branch-n-1	22	43	133, 314, 7
adder-4-sat	28	101	726, 534, 5	k-branch-n-2	39	103	294, 793, 9
adder-8-sat	60	411	1970, 2300, 5	k-branch-n-3	54	185	515, 1506, 11
adder-10-sat	76	644	2820, 3645, 5	k-branch-n-4	70	296	803, 2565, 13
adder-12-sat	92	929	3822, 5298, 5	k-branch-n-5	89	427	1149, 3874, 15
robots-1-5-2-1.6	2213	2213	6916, 23176, 3	k-branch-n-6	107	582	1557, 5505, 17
robots-1-5-2-1.7	1461	1461	7904, 26810, 3	k-branch-n-7	131	761	2027, 7482, 19
robots-1-5-2-1.8	3933	3933	8892, 30444, 3	k-branch-n-8	146	973	2568, 10117, 21
robots-1-5-2-1.9	1788	1788	9880, 34078, 3	k-branch-n-9	166	1201	3163, 12930, 23

Table 7.1: Comparison between $w = w_{CNT(Q)}$ and $w' = w_{\mathcal{G}}(\preceq_{Sov})$ on some instances of the QBF library (*nbv*, *nbc*, *nba* denote respectively the number of variables, the number of clauses, and the number of elimination operator alternations of an instance).

7.3.5 Comparison with existing approaches

The rewriting rules used in the semiring case can be compared with the *quantifier tree* approach [6] recently introduced for QBFs. This approach analyzes hidden structures of "flat" prenex normal

[6] See "http://www.qbflib.org/".

form QBFs, by using structuration mechanisms. This leads to important gains in terms of solving time. The structuration techniques used for quantifier trees are exactly the instantiation of rewriting rule DR to the algebraic structure associated with QBFs, using $\oplus = \vee$ and $\otimes = \wedge$.

MCTrees provide a theoretical explanation to the experimental gains observed when using *quantifier trees* on QBFs, in terms of tree-width. Also, since our approach is defined in a generic algebraic framework, it extends and generalizes the whole quantifier tree proposal. It is indeed applicable to multiple formalisms, including QCSP, SSAT, or stochastic CSP. Moreover, quantifier trees use: (1) neither recomposition rule RR together with cluster-tree decompositions, so as to minimize the width; (2) nor a simplification rule, since there are no normalization conditions on the clauses of a QBF.

7.3.6 Adding feasibilities

The difficulty in adding feasibilities lies in the use of the duplication mechanism, which is more complex if feasibilities are involved (see Proposition 6.33 page 107).

A solution to handle feasibilities consists in

- not using the duplication mechanism at all,
- and adding a simplification rule SR' allowing normalization conditions on feasibilities to be used:

$$\boxed{SR'} \qquad [\text{Precond.} : (op \in \{\min, \max\}) \wedge (c \in \mathcal{C}_D(G)) \wedge (c \cap (S \cup sc(N)) = \emptyset)]$$

$$(\underset{c\cup S}{op}, N \cup Fact(c)) \rightsquigarrow (\underset{S}{op}, N)$$

All the results previously given then still hold. Another solution not formalized enough yet can be to specify rewriting rules able to handle both feasibilities and duplications.

7.4 Structuring multi-operator queries in the semigroup case

The structuration of multi-operator queries in the semiring case leads to the MCTree architecture, which involves several elimination operators and one combination operator. The structuration in the semigroup case is different (and a bit more difficult) because it also involves several combination operators ($\otimes$ and $\oplus$). Again, we have a two-step structuration, involving a macrostructuration phase and a cluster-tree decomposition phase. In the following, we deal with the case where there are no feasibility functions, because it simplifies the presentation greatly. The case with feasibility is considered in Section 7.4.5.

7.4.1 Building the macrostructure of a query using rewriting rules

The initial computation node in the semigroup case is $n_0 = (Sov(o), \oplus, \{(\emptyset, \otimes, P \cup \{U_i\}), U_i \in U\})$. This time, the application of rewriting rules will generate a sequence of DAGs of computation nodes (CNDAGs), instead of trees of computation nodes. The first CNDAG of the sequence, denoted $CNDAG_0(Q, o)$, is $CNDAG_0(Q, o) = (Sov(o), \oplus, \{(\emptyset, \otimes, P \cup \{U_i\}), U_i \in U\})$.

In the following, we will manipulate sets of sets of computation nodes for notation issues. We use character $\mathfrak{N}$ to denote a set of sets of computation nodes, whereas a set of computation nodes is denoted by N. If we use sets of sets of computation nodes to express $CNDAG_0(Q,o)$, this gives:

$$CNDAG_0(Q,o) = (Sov(o), \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}), \text{ with } \mathfrak{N} = \{P \cup \{U_i\}, U_i \in U\}.$$

We can also define $\mathfrak{N}^{+x}$ as $\mathfrak{N}^{+x} = \{N \in \mathfrak{N} \,|\, x \in sc(N)\}$ and $\mathfrak{N}^{-x}$ as $\mathfrak{N}^{-x} = \mathfrak{N} - \mathfrak{N}^{+x}$.

Example 7.28. *For the influence diagram associated with the computation of* $\max_d \sum_{r_2,r_1} P_{r_1} \cdot P_{r_2|r_1} \cdot (U_{d,r_1} + U_{d,r_2} + U_d)$ *and for the elimination order* $o : d \prec r_2 \prec r_1$*, we have*

$$CNDAG_0(Q,o) = \left(\max_d \textstyle\sum_{r_2} \sum_{r_1}, +, \left\{ \begin{array}{c} (\emptyset, \times, \{P_{r_1}, P_{r_2|r_1}, U_{d,r_1}\}), \\ (\emptyset, \times, \{P_{r_1}, P_{r_2|r_1}, U_{d,r_2}\}), \\ (\emptyset, \times, \{P_{r_1}, P_{r_2|r_1}, U_d\}) \end{array} \right\} \right)$$

It corresponds to the first computation node in Figure 7.7.

We can also denote it as $CNDAG_0(Q,o) = (\max_d \sum_{r_2} \sum_{r_1}, +, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$*, where* $\mathfrak{N} = \{\{P_{r_1}, P_{r_2|r_1}, U_{d,r_1}\}, \{P_{r_1}, P_{r_2|r_1}, U_{d,r_2}\}, \{P_{r_1}, P_{r_2|r_1}, U_d\}\}$*. We then have* $\mathfrak{N}^{+r_1} = \mathfrak{N}$*. With* $\mathfrak{N} = \{\{P_{r_1}, U_{d,r_2}\}, \{P_{r_2\,|\,r_1}, P_{r_1}, U_{d,r_1}\}, \{P_{r_1}, U_d\}\}$*, we would have* $\mathfrak{N}^{-r_2} = \{\{P_{r_1}, U_d\}\}$ *and* $\mathfrak{N}^{+r_2} = \{\{P_{r_1}, U_{d,r_2}\}, \{P_{r_2\,|\,r_1}, P_{r_1}, U_{d,r_1}\}\}$*.*

For all $k \in \{0, \ldots, |Sov| - 1\}$, the macrostructure at step $k+1$, denoted $CNDAG_{k+1}(Q,o)$, is obtained from $CNDAG_k(Q,o)$ by considering the rightmost remaining elimination and, as in the semiring case, by applying three types of rewriting rules (decomposition, recomposition, and simplification). Rewriting rules are presented first for the case of $\oplus$-eliminations, and then for the case of max-eliminations. The case of min-eliminations when $\min \neq \oplus$ is analogous to the case of max-eliminations.

Rewriting rules for $\oplus_x$ When a $\oplus$-elimination must be performed, a decomposition rule $DR_\oplus$ and the rewriting rules of the semiring case are used. The mechanism is illustrated in Figure 7.7, which corresponds to the influence diagram associated with the computation of

$$\max_d \textstyle\sum_{r_2,r_1} P_{r_1} \cdot P_{r_2|r_1} \cdot (U_{d,r_1} + U_{d,r_2} + U_d).$$

1. *Decomposition rule* $DR_\oplus$ simply implements the duplication mechanism, i.e. it uses a mechanism looking like $\oplus_x(P \otimes (U_1 \oplus U_2)) = (\oplus_x(P \otimes U_1)) \oplus (\oplus_x(P \otimes U_2))$:

$$\boxed{DR_\oplus} \qquad \begin{array}{l} (sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) \\ \rightsquigarrow (sov, \oplus, \{(\oplus_x, \otimes, N), N \in \mathfrak{N}\}) \end{array}$$

   In the example of Figure 7.7, the first applied rule is $DR_\oplus$. It treats the operator-variable pair $\sum_{r_1}$ and transforms $CNDAG_0(Q,o)$ into another structure in which r_1 is duplicated.

2. The computation nodes created by $DR_\oplus$ look like $n = (\oplus_x, \otimes, N)$. This is exactly the form of a computation node in the semiring case. Hence, each node n created by $DR_\oplus$ can be structured thanks to the rewriting rules DR, RR, and SR defined in the semiring case. In other words, as in the semiring case, n can be transformed into *rewrite*(n), or into *simplify*(*rewrite*(n)) if one wants the simplification rule to be used.

   In Figure 7.7, function *rewrite* (which uses decomposition rule DR and recomposition rule RR) enables us to transform the structure obtained after the application of $DR_\oplus$ into structure $CNDAG_1(Q,o)$ given just below. $CNDAG_1(Q,o)$ is an actual DAG of computation

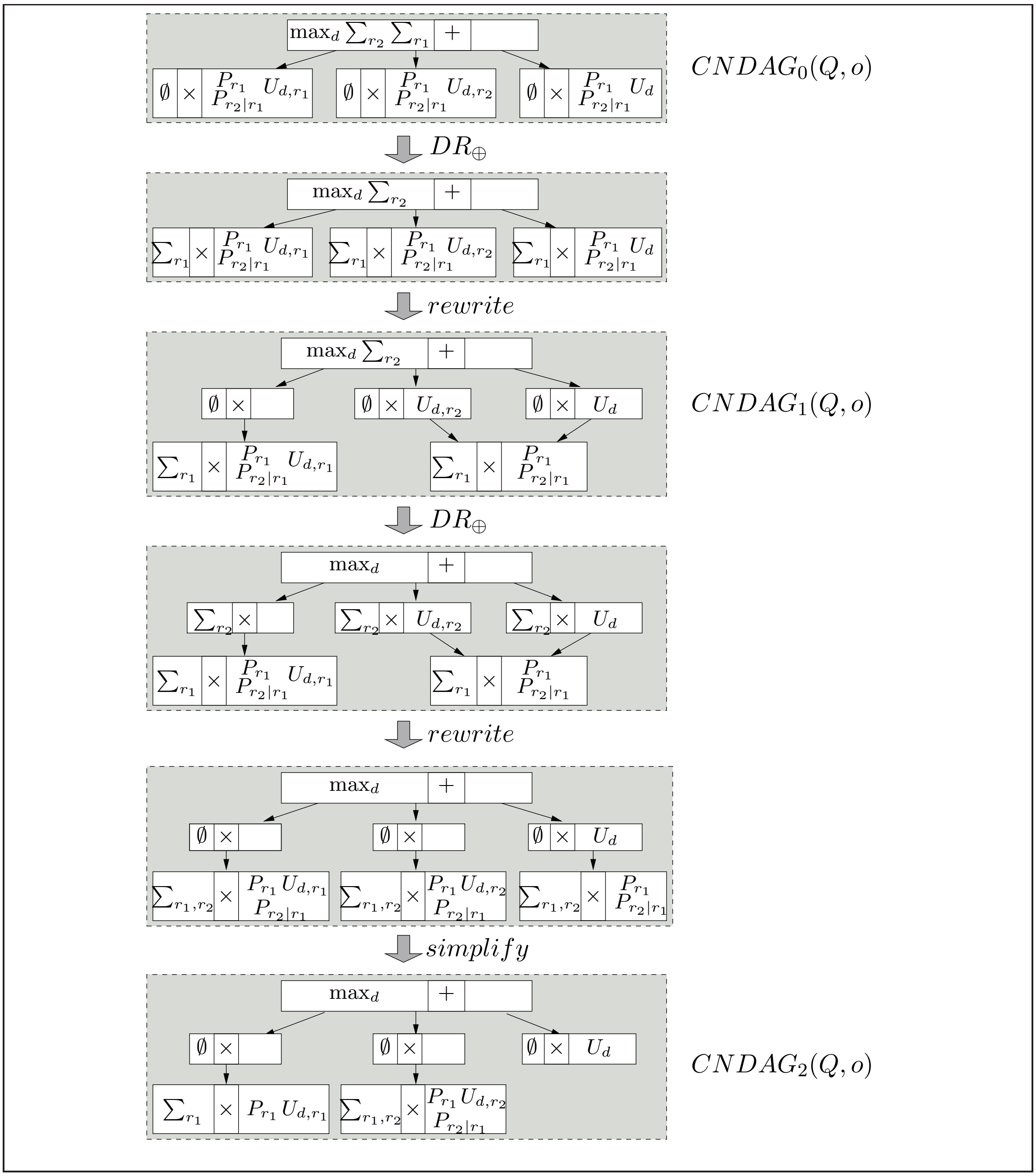

$\max_d \sum_{r_2} \sum_{r_1}$ +
∅ × P_{r_1} U_{d,r_1} $P_{r_2|r_1}$
∅ × P_{r_1} U_{d,r_2} $P_{r_2|r_1}$
∅ × P_{r_1} U_d $P_{r_2|r_1}$
$CNDAG_0(Q,o)$
$DR_\oplus$
$\max_d \sum_{r_2}$ +
$\sum_{r_1}$ × P_{r_1} U_{d,r_1} $P_{r_2|r_1}$
$\sum_{r_1}$ × P_{r_1} U_{d,r_2} $P_{r_2|r_1}$
$\sum_{r_1}$ × P_{r_1} U_d $P_{r_2|r_1}$
$rewrite$
$\max_d \sum_{r_2}$ +
∅ ×
∅ × U_{d,r_2}
∅ × U_d
$\sum_{r_1}$ × P_{r_1} U_{d,r_1} $P_{r_2|r_1}$
$\sum_{r_1}$ × P_{r_1} $P_{r_2|r_1}$
$CNDAG_1(Q,o)$
$DR_\oplus$
$\max_d$ +
$\sum_{r_2}$ ×
$\sum_{r_2}$ × U_{d,r_2}
$\sum_{r_2}$ × U_d
$\sum_{r_1}$ × P_{r_1} U_{d,r_1} $P_{r_2|r_1}$
$\sum_{r_1}$ × P_{r_1} $P_{r_2|r_1}$
$rewrite$
$\max_d$ +
∅ ×
∅ ×
∅ × U_d
$\sum_{r_1,r_2}$ × P_{r_1} U_{d,r_1} $P_{r_2|r_1}$
$\sum_{r_1,r_2}$ × P_{r_1} U_{d,r_2} $P_{r_2|r_1}$
$\sum_{r_1,r_2}$ × P_{r_1} $P_{r_2|r_1}$
$simplify$
$\max_d$ +
∅ ×
∅ ×
∅ × U_d
$\sum_{r_1}$ × P_{r_1} U_{d,r_1}
$\sum_{r_1,r_2}$ × P_{r_1} U_{d,r_2} $P_{r_2|r_1}$
$CNDAG_2(Q,o)$

Figure 7.7: Application of rewriting rules for $\oplus$ when $\oplus = +$.

nodes since common computation nodes such as $(\sum_{r_1}, \times, \{P_{r_1}, P_{r_2\,|\,r_1}\})$ are shared. It is not hard to detect such shared nodes when applying the rewriting rules. After some further rewriting steps, we get structure $CNDAG_2(Q,o)$ given in Figure 7.7.

In the end, in the example of Figure 7.7, no computation involves more than two variables in $CNDAG_2(Q,o)$ if we eliminate r_1 first in the node $(\sum_{r_1,r_2}, \times, \{P_{r_1}, P_{r_2|r_1}, U_{d,r_2}\})$. With a potential-based approach, it would be necessary to process three variables simultaneously: indeed, r_1 would be involved in potentials $(P_{r_1}, 0), (P_{r_2|r_1}, 0), (1, U_{d,r_1})$ if eliminated first, and r_2 would be involved in potentials $(P_{r_2|r_1}, 0), (1, U_{d,r_2})$ if eliminated first.

In order to systematize the rules application order, we write that for all $k \in \{0, \ldots, |Sov| - 1\}$ such that $CNDAG_k(Q, o) = (sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, the structure $CNDAG_{k+1}(Q, o)$ at step $k + 1$ is defined by [7]

$$\begin{aligned} &CNDAG_{k+1}(Q, o) \\ &= \begin{cases} (sov, \oplus, \{\mathit{simplify}(\mathit{rewrite}\,(\oplus_x, \otimes, N)), N \in \mathfrak{N}\}) & \text{if } sov = sov'.op_x \text{ and } op \neq \oplus \\ (sov, \oplus, \{\mathit{rewrite}\,(\oplus_x, \otimes, N)\,, N \in \mathfrak{N}\}) & \text{otherwise} \end{cases} \end{aligned}$$

In other words, when eliminating variable x, we decompose the computations using duplication, and then use the rewriting rules defined in the semiring case.

Rewriting rules for $\max_x$ When a max-marginalization must be performed, a decomposition rule $DR_{\max}$ and a recomposition rule $RR_{\max}$ are used. No simplification rule is required since no normalization condition is available on decision variables when there are no feasibilities.

The rewriting rules are a bit more complex than the previous ones and are illustrated in Figure 7.8, which corresponds to the influence diagram $\max_{d_1} \sum_{r_2} \max_{d_2} \sum_{r_1} \max_{d_3} P_{r_1} \cdot P_{r_2|r_1} \cdot (U_{d_1} + U_{d_2,d_3} + U_{r_2,d_1,d_3} + U_{r_1,d_2})$.

1. Decomposition rule $DR_{\max}$ enables us to consider only scoped functions having x in their scope when x is eliminated using max.

   $$\boxed{DR_{\max}} \quad (sov.\max_x, \oplus, \{(\emptyset, \otimes, N)\,, N \in \mathfrak{N}\})$$
   $$\rightsquigarrow \begin{cases} (sov, \oplus, \{(\emptyset, \otimes, N)\,, N \in \mathfrak{N}\}) \text{ if } \mathfrak{N}^{+x} = \emptyset \\ (sov, \oplus, \{(\emptyset, \otimes, N)\,, N \in \mathfrak{N}^{-x}\} \cup \{(\emptyset, \otimes, N_1 \cup \{(\max_x, \oplus, N_2)\})\}) \text{ otherwise} \end{cases}$$
   $$\text{where } N_1 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x} \text{ and } N_2 = \left\{(\emptyset, \otimes, N - N_1)\,, N \in \mathfrak{N}^{+x}\right\}$$

   $DR_{\max}$ says that when x is eliminated, it is not necessary to consider nodes $(\emptyset, \otimes, N)$ such that $x \notin sc(N)$, and we factor the parts independent from x that the other $(\emptyset, \otimes, N)$ nodes have in common.

   In Figure 7.8, $DR_{\max}$ transforms $CNDAG_0(Q, o)$ into $CNDAG_1(Q, o)$, by treating the elimination $\max_{d_3}$. It uses the fact that among root sons, only $(\emptyset, \times, \{P_{r_1}, P_{r_2\,|\,r_1}, U_{d_2,d_3}\})$ and $(\emptyset, \times, \{P_{r_1}, P_{r_2\,|\,r_1}, U_{r_2,d_1,d_3}\})$ depend on x_3. They share common factors, P_{r_1} and $P_{r_2\,|\,r_1}$, both independent from x_3, which is explicitly taken into account in $CNDAG_1(Q, o)$.

   Then, $DR_{\oplus}$ can be used to process $\sum_{r_1}$ and give $CNDAG_2(Q, o)$, and $DR_{\max}$ can be used to process $\max_{d_2}$.

2. Recomposition rule $RR_{\max}$ enables us to reveal freedoms in the elimination order. Among

[7] Given $N \in \mathfrak{N}$, computation nodes in N can look like $(\oplus_S, \otimes, N')$, as standard nodes of the semiring case. But they can also look like $(\max_S, \oplus, N')$. This does not matter to apply the rewriting rules of the semiring case because these latter nodes will never be recomposed with a node performing eliminations with $\oplus$.

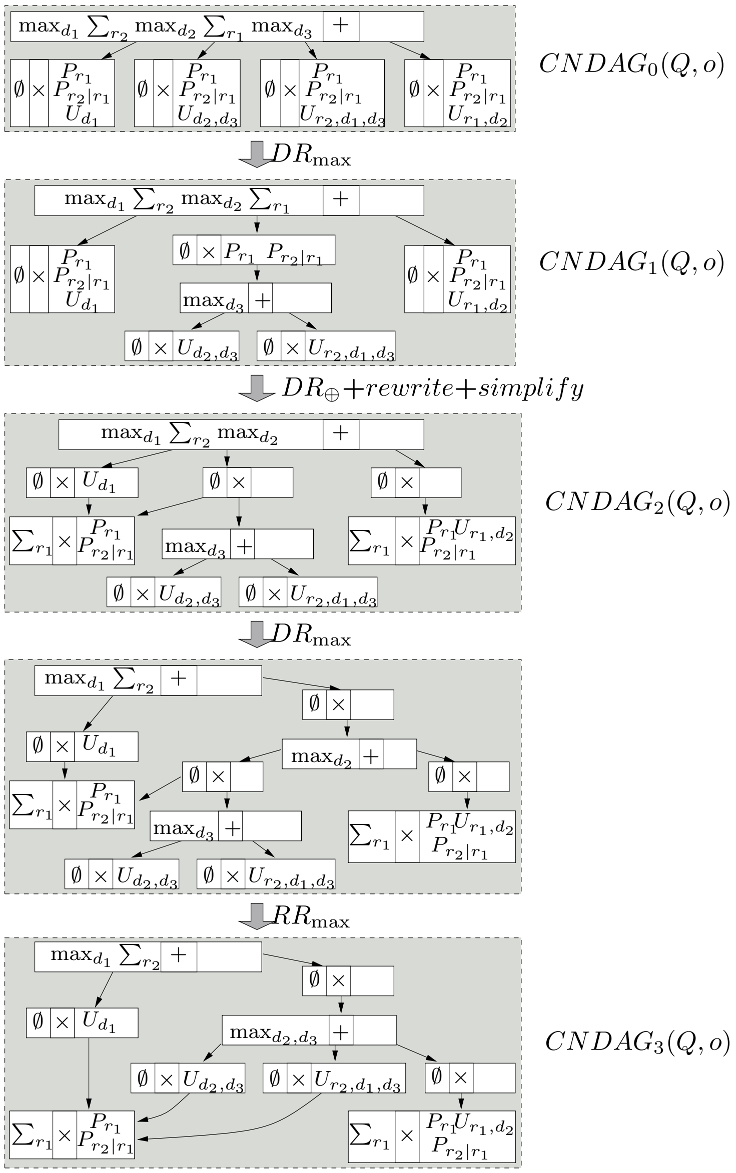

$\max_{d_1} \sum_{r_2} \max_{d_2} \sum_{r_1} \max_{d_3}$ +
$\emptyset$ × P_{r_1} $P_{r_2|r_1}$ U_{d_1}
$\emptyset$ × P_{r_1} $P_{r_2|r_1}$ U_{d_2,d_3}
$\emptyset$ × P_{r_1} $P_{r_2|r_1}$ U_{r_2,d_1,d_3}
$\emptyset$ × P_{r_1} $P_{r_2|r_1}$ U_{r_1,d_2}
$CNDAG_0(Q,o)$
$DR_{\max}$
$\max_{d_1} \sum_{r_2} \max_{d_2} \sum_{r_1}$ +
$\emptyset$ × P_{r_1} $P_{r_2|r_1}$
$\max_{d_3}$ +
$\emptyset$ × U_{d_2,d_3}
$\emptyset$ × U_{r_2,d_1,d_3}
$CNDAG_1(Q,o)$
$DR_{\oplus}$+$rewrite$+$simplify$
$\max_{d_1} \sum_{r_2} \max_{d_2}$ +
$\emptyset$ × U_{d_1}
$\sum_{r_1}$ × P_{r_1} $P_{r_2|r_1}$
$\sum_{r_1}$ × $P_{r_1}U_{r_1,d_2}$ $P_{r_2|r_1}$
$CNDAG_2(Q,o)$
$DR_{\max}$
$\max_{d_1} \sum_{r_2}$ +
$\max_{d_2}$ +
$RR_{\max}$
$\max_{d_2,d_3}$ +
$CNDAG_3(Q,o)$

Figure 7.8: Application of rewriting rules for max (the application of the rules may create nodes such as $(\emptyset, \otimes, \{n\})$, which perform no computations; these nodes can be removed at a final step).

all rewriting rules, $RR_{\max}$ has the most complicated form. Intuitively, $RR_{\max}$ gathers max-eliminations. The best explanation of $RR_{\max}$ is actually provided by the proof of Lemma 7.35.

$$\boxed{RR_{\max}} \quad \begin{aligned} &(\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) \\ &\rightsquigarrow (\max_{\{x\} \cup (\cup_{N \in \mathfrak{N}} V_e(N[\max]))}, \oplus, \\ &\qquad\qquad \{(\emptyset, \otimes, N), (N \in \mathfrak{N}) \wedge (N[\max] = \emptyset)\} \\ &\qquad\qquad \cup \left\{ (\emptyset, \otimes, N[\neg \max] \cup N'), \begin{array}{c} (N \in \mathfrak{N}) \wedge (N[\max] \neq \emptyset) \\ \wedge (\emptyset, \otimes, N') \in Sons(N[\max]) \end{array} \right\}) \end{aligned}$$

In Figure 7.8, recomposition rule $RR_{\max}$ yields $CNDAG_3(Q,o)$ by revealing the freedom in the elimination order between d_2 and d_3. This freedom was hidden in the initial multi-operator sequence of eliminations.

A systematic rewriting using $DR_{\max}$ and $RR_{\max}$ is: if $\max \neq \oplus$, then for all $k \in \{0, \ldots, |Sov|-1\}$ such that $CNDAG_k(Q,o) = (sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, the DAG of computation nodes at the next step is

$$
\begin{aligned}
&CNDAG_{k+1}(Q,o)\\
&= \begin{cases} (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) \text{ if } \mathfrak{N}^{+x} = \emptyset \\ (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}^{-x}\} \cup \{(\emptyset, \otimes, N_1 \cup \{RR_{\max}(\max_x, \oplus, N_2)\})\}) \text{ otherwise} \end{cases}\\
&\quad \text{where } N_1 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x} \text{ and } N_2 = \left\{(\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\right\}
\end{aligned}
$$

This means that we decompose the computations as specified by $DR_{\max}$ and then recompose the created node performing the elimination of x by using $RR_{\max}$. For eliminations using min, $CNDAG_{k+1}(Q,o)$ has exactly the same form. The only difference is that max must be replaced by min.

The final macrostructure obtained given a query $Q = (Sov, \mathcal{N})$ and an elimination order $o \in lin(\preceq_{Sov})$ is $CNDAG_{|Sov|}(Q,o)$. It is also denoted $CNDAG(Q,o)$.[8]

Some good properties of the macrostructure obtained

Unicity The independence of the macrostructure obtained with regard to the chosen elimination order compatible with $\preceq_{Sov}$ is provided by Theorem 7.30.

Lemma 7.29. *Let $Q = (Sov, \mathcal{N})$ be a query and let $o, o' \in lin(\preceq_{Sov})$. Let $op \in \{\min, \max, \oplus\}$ and let $k \in \{0, \ldots, |Sov|-2\}$. If* $\begin{cases} CNDAG_k(Q,o) = (sov \cdot op_x \cdot op_y, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) \\ CNDAG_k(Q,o') = (sov \cdot op_y \cdot op_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) \end{cases}$, *then $CNDAG_{k+2}(Q,o) = CNDAG_{k+2}(Q,o')$.*

Theorem 7.30. *Let $Q = (Sov, \mathcal{N})$ be a query. Then, for all $o, o' \in lin(\preceq_{Sov})$, $CNDAG(Q,o) = CNDAG(Q,o')$*

This allows us to denote the final macrostructure as $CNDAG(Q)$ instead of $CNDAG(Q,o)$.

Soundness The soundness of the macrostructure obtained is provided by the soundness of the rewriting rules, which leads us to Theorem 7.38.

Lemma 7.31. *Rewriting rule $DR_{\oplus}$ is sound.*

Lemma 7.32. *Let $Q = (Sov, \mathcal{N})$ be a query and let $o \in lin(\preceq_{Sov})$. Let $k \in \{0, \ldots, |Sov|-1\}$ and let $\mathfrak{N}$ be a set of sets of computation nodes such that $CNDAG_k(Q,o) = (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$.*

Then,

[8] Note that the rewriting rules imply that at each step, the root computation node always looks like $\{(sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})\}$, hence the rewriting rules for $\oplus$ are applicable if $sov = sov'.\oplus_x$, the rewriting rules for max are applicable if $sov = sov'.\max_x$, and the rewriting rules for min are applicable if $sov = sov'.\min_x$. This shows that $CNDAG_{k+1}(Q,o)$ is defined for every $k \in \{0, \ldots, |Sov|-1\}$.

- *for all* $N \in \mathfrak{N}$*, for all* $(n_1, n_2) \in N^2$*,*
$$(n_1 \neq n_2) \rightarrow ((V_e(n_1) \cap V_e(n_2) = \emptyset) \wedge (V_e(n_1) \cap sc(n_2) = \emptyset))$$
Moreover, *for all* $(n_1, n_2) \in (N[\oplus])^2$*,* $(n_1 \neq n_2) \rightarrow (Sons(n_1) \cap Sons(n_2) = \emptyset)$*,*
for all $n \in N[\oplus]$*,* $Sons(n) \cap N[\neg\oplus] = \emptyset$*.*

- *if* $\max \neq \oplus$*, then, for all* $(N_1, N_2) \in \mathfrak{N}^2$*,*
$$(N_1 \neq N_2) \rightarrow ((V_e(N_1[\max]) \cap V_e(N_2[\max]) = \emptyset) \wedge (V_e(N_1[\max]) \cap sc(N_2) = \emptyset))$$
Moreover, for all $N \in \mathfrak{N}$*,* $|N[\max]| \leq 1$
for all $(\emptyset, \otimes, N_s) \in Sons(N[\max])$*,* $N_s \cap N[\neg \max] = \emptyset$*.*

Idem for min *when* $\min \neq \oplus$*.*

Lemma 7.33. *Let* $Q = (Sov, \mathcal{N})$ *be a query and let* $o \in lin(\preceq_{Sov})$*. Let* $k \in \{0, \dots, |Sov| - 1\}$ *such that* $CNDAG_k(Q, o) = (sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$*. Then,* $val(CNDAG_{k+1}(Q, o)) = val(CNDAG_k(Q, o))$*.*

Lemma 7.34. *Rewriting rule* $DR_{\max}$ *is sound.*

Lemma 7.35. *Let* $RR'_{\max}$ *be the rewriting rule defined as:*
$$RR'_{\max} \;:\; (\max_S, \oplus, N_1 \cup \{(\emptyset, \otimes, N_2 \cup \{(\max_{S'}, \oplus, \{(\emptyset, \otimes, N_3), N_3 \in \mathfrak{N}\})\})\})$$
$$\rightsquigarrow (\max_{S \cup S'}, \oplus, N_1 \cup \{(\emptyset, \otimes, N_2 \cup N_3), N_3 \in \mathfrak{N}\})$$
If $S' \cap (S \cup sc(N_1) \cup sc(N_2)) = \emptyset$ *and* $\forall N_3 \in \mathfrak{N}$*,* $N_2 \cap N_3 = \emptyset$*, then* $RR'_{\max}$ *is a sound rewriting rule.*

Lemma 7.36. *Let* $Q = (Sov, \mathcal{N})$ *be a query and let* $o \in lin(\preceq_{Sov})$*. Let* $k \in \{0, \dots, |Sov| - 1\}$ *such that* $CNDAG_k(Q, o) = (sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$*. Then,* $val(CNDAG_{k+1}(Q, o)) = val(CNDAG_k(Q, o))$*. Similarly, if* $CNDAG_k(Q, o) = (sov.\min_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$*, then* $val(CNDAG_{k+1}(Q, o)) = val(CNDAG_k(Q, o))$*.*

Lemma 7.37. *Let* $Q = (Sov, \mathcal{N})$ *be a query and let* $o \in lin(\preceq_{Sov})$*. Then, for all* $k \in \{0, \dots, |Sov| - 1\}$*,* $val(CNDAG_{k+1}(Q, o)) = val(CNDAG_k(Q, o))$*.*

Theorem 7.38. *Let* $Q = (Sov, \mathcal{N})$ *be a query. Then,* $val(CNDAG(Q)) = Ans(Q)$*.*

Complexity results

An architecture is usable only if it is reasonable to build it. Proposition 7.39 gives upper bounds on the complexity of the rewriting process when the simplification rule is not used. As in the semiring case, the complexity is still polynomial when simplification rule is used. An explicit algorithm implementing the rewriting rule in the semigroup case is given in the proof of Proposition 7.39. It notably manipulates pointers to computation nodes, so that computation nodes can be shared, i.e. so that the DAG structure is explicit.

Proposition 7.39. *If the simplification rule is not used, the time and space complexities of the rewriting process in the semigroup case are* $O(|U| \cdot |V| \cdot (|P| + |V|) \cdot (1 + |P|))$ *and* $O(|U| \cdot |V| \cdot (|V| + |P|^2))$ *respectively.*

7.4.2 Cluster-tree decompositions to structure DAGs of computation nodes: towards multi-operator cluster-DAGs (MCDAGs)

In the semigroup case, the rewriting rules yield a DAG of mono-operator computation nodes such as $(\min_S, \oplus, N)$, $(\max_S, \oplus, N)$, $(\sum_S, \otimes, N)$, and $(\emptyset, \otimes, N)$. As in the semiring case, the second finer structuration step consists of taking advantage of freedoms in the elimination order inside each of these mono-operator computation nodes by using cluster-tree decompositions.

Given a computation node $n = (op_S, \circledast, N)$, it suffices to build a rooted cluster-tree decomposition of the graphical model $(sc(n), \{val(n'), n' \in N\})$ associated with it, given the variables in $sc(n) - S$ which are not eliminated by n. This directly provides us with a structuration of the computation of $val(n)$. The structure obtained then contains both a macrostructure given by the computation nodes and an internal cluster-tree structure given by each of their decompositions. After this second structuration step, we obtain a so-called *multi-operator cluster DAG* (MCDAG).

Definition 7.40. *A* Multi-operator Cluster DAG *is a DAG where every vertex c, called a cluster, is labeled with three elements:*

- *a set of variables $V(c)$,*
- *a set of scoped functions $\Phi(c)$ taking values in a set E,*
- *and a couple $(\oplus^c, \otimes^c)$ of operators on E such that $(E, \oplus^c, \otimes^c)$ is a commutative semiring.*

The width of a MCDAG is defined by $w = \max_{c \in C} |V(c)| - 1$. The height of a MCDAG is the maximum number of variables which appear in a path from the root to a leaf in the MCDAG.

Definition 7.41. *The value of a cluster c of a MCDAG is given by*

$$val(c) = \oplus^c{}_{V(c)-V(pa(c))} \left(\left(\otimes^c{}_{\varphi \in \Phi(c)} \varphi \right) \otimes^c \left(\otimes^c{}_{s \in Sons(c)} val(s) \right) \right)$$

The value of a MCDAG is the value of its root node.

We explicitly specify a combination operator and an elimination operator to be used inside each cluster because these operators may vary depending on the cluster considered. If duplicated variables are relabeled, then MCDAGs obtained from a query Q satisfy a kind of running intersection property, which is "for all clusters c_1, c_2, c_3, if c_3 is on a path from c_1 to c_2 which uses only non convergent connections, then $V(c_1) \cap V(c_2) \subset V(c_3)$".

Decreasing the MCDAG width

The next three pages correspond to a technical part which shows that given a computation node $n = (op_S, \circledast, N)$, building a cluster-tree decomposition of $(sc(n), \{val(n'), n' \in N\})$ given $sc(n) - S$ yields MCDAGs which have a suboptimal width. The reason for this is that in the semigroup case, the computation nodes performing eliminations with min or max have a particular structure. We begin with an illustrative example.

Example 7.42. *Let us consider a computation node* $n = (\max_{x,y,z}, \oplus, \left\{ \begin{array}{c} (\emptyset, \otimes, \{U_{z,t}\}) \\ (\emptyset, \otimes, \{U_{y,t}\}) \\ (\emptyset, \otimes, \{n_t, U_{x,y}\}) \\ (\emptyset, \otimes, \{n_t, U_x\}) \end{array} \right\})$, *where n_t is a shared computation node of scope $\{t\}$, which can typically correspond to a factor performing operations on plausibility functions.*

Assume that we want to exploit the freedoms in the elimination order between x, y, and z, thanks to a cluster-tree decomposition. If we use the mechanism previously proposed, then we consider the graphical model $\mathcal{M} = (\{x, y, z, t\}, \{\varphi_{z,t}, \varphi_{y,t}, \varphi_{x,y,t}, \varphi_{x,t}\})$ where $\varphi_{z,t} = val((\emptyset, \otimes, \{U_{z,t}\}))$, $\varphi_{y,t} = val((\emptyset, \otimes, \{U_{y,t}\}))$, $\varphi_{x,y,t} = val((\emptyset, \otimes, \{n_t, U_{x,y}\}))$, and $\varphi_{x,t} = val((\emptyset, \otimes, \{n_t, U_x\}))$. Then, we build a cluster-tree decomposition of $\mathcal{M}$ given $\{t\}$. An optimal cluster-tree decomposition of $\mathcal{M}$ given $\{t\}$ has a width of 2. This means that at least $2 + 1 = 3$ variables need to be considered simultaneously in order to compute $val(n)$.

However, a decomposition of the computations exists which allows us to consider at most two variables simultaneously. Indeed, if we use the elimination order $z \prec y \prec x$, we can write:

$$\begin{aligned}
val(n) &= \max_z \max_y \max_x (U_{z,t} \oplus U_{y,t} \oplus (val(n_t) \otimes U_{x,y}) \oplus (val(n_t) \otimes U_x)) \\
&= \max_z \max_y (U_{z,t} \oplus U_{y,t} \oplus \max_x ((val(n_t) \otimes U_{x,y}) \oplus (val(n_t) \otimes U_x))) \\
&= \max_z \max_y (U_{z,t} \oplus U_{y,t} \oplus (val(n_t) \otimes \max_x (U_{x,y} \oplus U_x))) \\
&= \max_z (U_{z,t} \oplus \max_y (U_{y,t} \oplus (val(n_t) \otimes \max_x (U_{x,y} \oplus U_x))))
\end{aligned}$$

The decomposition above considers

- *two variables (x and y) to compute* $\max_x(U_{x,y} \oplus U_x) = U'_y$,
- *two variables (y and t) to compute* $\max_y(U_{y,t} \oplus (val(n_t) \otimes U'_y) = U'_t$,
- *two variables (z and t) to compute* $\max_z(U_{z,t} \oplus U'_t)$.

In order to consider only two variables simultaneously, the key mechanism is to use the fact that n_t is a factor of both $U_{x,y}$ and U_x.

The goal of this technical part is to generalize the decomposition method used in the previous example, in order to obtain cluster-tree decomposition having a smaller width. We take the example of computation nodes $(\max_S, \oplus, N)$ when $\max \neq \oplus$, but everything that follows applies to computation nodes $(\min_S, \oplus, N)$ as well (when $\min \neq \oplus$).

We first need some additional notations, defining the *type* of a computation node.

Definition 7.43. *Let n be a computation node.*

- *If n is atomic, then the type of n is $t(n) = u$ if $n \in U$, and $t(n) = p$ if $n \in P$.*
- *Otherwise, if $n = (sov, \circledast, N)$ then $t(n) = u$ if there exists $n' \in N$ such that $t(n') = u$, and $t(n) = p$ otherwise.*

This means that a computation node is of type u iff at least one utility function is involved in its descendants.

Actually, the rewriting rules in the semigroup case imply that the computation nodes in $CNDAG(Q)$ which use max as an elimination operator are always of the form $(\max_S, \oplus, \{(\emptyset, \otimes, N),$ $N \in \mathfrak{N}\})$. Proposition 7.44 below gives key properties satisfied by these nodes. As we shall see, these properties allow us to decrease the MCDAG width.

For the remaining of the chapter, we assume that during the application of rewriting rules in the semigroup case and given a set of computation nodes N, the definitions of N^{+x}/N^{-x} are updated by $N^{+x} = \{n \in N \mid x \in sc(n)\} \cup N_0$, where $N_0 = N \cap Fact(c(x))$ if x is the last variable in $c(x)$ to be eliminated, and $N^{-x} = N - N^{+x}$.[9]

[9] This modification is identical to the modification performed in Chapter 6 when using potentials in the semigroup case.

Proposition 7.44. *Let Q be a query. Let us consider a computation node $(op_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$ in $CNDAG(Q)$, when $op \neq \oplus$.*

Then, for every $N \in \mathfrak{N}$, there exists a unique $n \in N$ such that $t(n) = u$. This node is denoted $u(N)$. The set of nodes in $N - \{u(N)\}$ is denoted $P(N)$. It satisfies $S \cap sc(P(N)) = \emptyset$.

Moreover, for all $N_1, N_2 \in \mathfrak{N}$, $((n \in N_1) \wedge (t(n) = p)) \rightarrow ((n \in N_2) \vee (sc(n) \subset sc(u(N_2))))$.

Informally, Proposition 7.44 shows that given a computation node $(\max_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, computation nodes of type p are either shared between several $N \in \mathfrak{N}$, or their scope is included in another computation node of type u. Furthermore, their scopes do not involve variables in S.

Then, let x be a variable in S. If x is the first variable to be eliminated, how many variables need to be considered? The elimination of x can be decomposed as follows:

$$\begin{aligned}
& \max_S \Big(\bigoplus_{N \in \mathfrak{N}} \Big(\bigotimes_{n \in N} val(n) \Big) \Big) \\
= \; & \max_{S-\{x\}} \Big(\Big(\bigoplus_{N \in \mathfrak{N}^{-x}} \Big(\bigotimes_{n \in N} val(n) \Big) \Big) \oplus \Big(\max_x \Big(\bigoplus_{N \in \mathfrak{N}^{+x}} \Big(\bigotimes_{n \in N} val(n) \Big) \Big) \Big) \Big) \\
= \; & \max_{S-\{x\}} \Big(\Big(\bigoplus_{N \in \mathfrak{N}^{-x}} \Big(\bigotimes_{n \in N} val(n) \Big) \Big) \oplus \Big(\Big(\bigotimes_{n \in N_1} val(n) \Big) \otimes \max_x \Big(\bigoplus_{N \in \mathfrak{N}^{+x}} \Big(\bigotimes_{n \in N - N_1} val(n) \Big) \Big) \Big) \Big)
\end{aligned}$$

where $N_1 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$.

Let n be a node of type p which is in $N - N_1$ for one $N \in \mathfrak{N}^{+x}$. Thanks to Proposition 7.44, we know that $x \notin sc(n)$. Moreover, as n is not common to all computation nodes in $\mathfrak{N}^{+x}$, we know that there exists $N' \in \mathfrak{N}^{+x}$ such that $x \in sc(u(N'))$. Hence, in order to determine the number of variables to consider to eliminate x and the scope of the function created after the elimination of x, it actually suffices to consider computation nodes in $\{u(N), N \in \mathfrak{N}\}$, instead of computation nodes in $\{(\emptyset, \otimes, N), N \in \mathfrak{N}\}$. Roughly speaking, this means that computation nodes of type p play only a weighing role and do not basically modify the scopes of the functions manipulated.

This leads us to define several steps to obtain MCDAGs with an improved width. In order to decompose a computation node $n = (\max_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, we proceed as follows:

1. First, we build a rooted cluster-tree decomposition of the graphical model $(sc(n), \{val(u(N)), N \in \mathfrak{N}\})$ given $sc(n) - S$.

2. Second, we transform this decomposition into a MCDAG where weights given by plausibility nodes are reintegrated. To do so, for every cluster c:

   - for every $\varphi \in \Phi(c)$, there exists $N \in \mathfrak{N}$ such that $\varphi = val(u(N))$. Then, create a cluster s and add it to $Sons(c)$; remove φ from $\Phi(c)$ and put it in $\Phi(s)$; add in $\Phi(s)$ scoped functions in $P(N) \cap P$, and add in $Sons(s)$ scoped functions in $P(N) - P$. Informally, this step weighs utility functions with plausibility functions left apart for the computation of a cluster-tree decomposition;

   - for every $s \in Sons(c)$, create an intermediate level between c and s: remove s from $Sons(c)$; create a cluster c' such that $Sons(c') = \{s\}$; add c' to $Sons(c)$; take $\Phi(c') = \emptyset$ and $(\oplus_{c'}, \otimes_{c'}) = (\emptyset, \otimes)$. Informally, this step create intermediate level in the architecture, which will be useful for the next step.

3. Third, we move the plausibility weights as "high" as possible in the MCDAG: starting from the leaves, for every cluster c, we remove the plausibilities which weigh every son of c. More precisely, we transfer the scoped functions in $\cap_{s \in Sons(c)} \Phi(s)$ to $\Phi(pa(c))$, and the clusters in $\cap_{s \in Sons(c)} Sons(s)$ to $Sons(pa(c))$. Finally, we "clean" the obtained structure by removing useless clusters.

Example 7.45. *Let us consider the computation node* $n = (\max_{x,y,z}, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$ *given in Example 7.42 again, where* $\mathfrak{N} = \{\{U_{z,t}\}, \{U_{y,t}\}, \{n_t, U_{x,y}\}, \{n_t, U_x\}\}$.

We first build a rooted cluster-tree decomposition of the graphical model $(sc(n), \{val(u(N)), N \in \mathfrak{N}\})$ *given* $\{t\}$, *i.e. of the graphical model* $(\{x, y, z, t\}, \{U_{z,t}, U_{y,t}, U_{x,y}, U_x\})$ *given* $\{t\}$. *The primal graph of this graphical model is given in Figure 7.9(a). A rooted cluster-tree decomposition is given in Figure 7.9(b).*

We then add intermediate levels to get a MCDAG, enabling us to weigh the utility functions and to "prepare" the structure for the weights migration. This is done in Figure 7.9(c).

Last, we put the weights as high as possible in the MCDAG by detecting common weights between the sons of a given cluster, and we remove useless clusters. This gives the MCDAG in Figure 7.9(d). This MCDAG exactly corresponds to the smart decomposition given in Example 7.42.

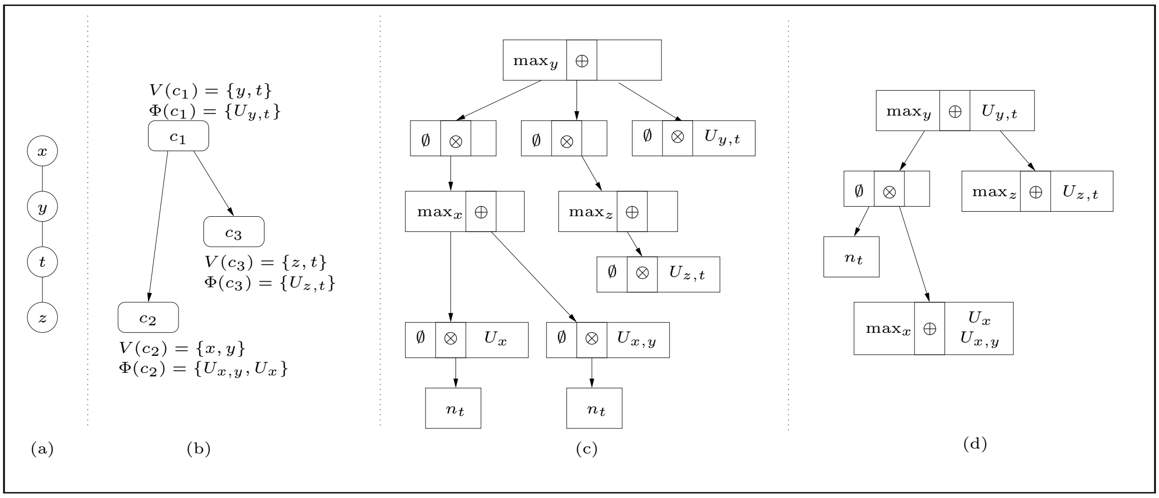

x
y
t
z
$V(c_1) = \{y, t\}$
$\Phi(c_1) = \{U_{y,t}\}$
c_1
c_3
$V(c_3) = \{z, t\}$
$\Phi(c_3) = \{U_{z,t}\}$
c_2
$V(c_2) = \{x, y\}$
$\Phi(c_2) = \{U_{x,y}, U_x\}$
$\max_y$ $\oplus$
$\emptyset$ $\otimes$
$\emptyset$ $\otimes$
$\emptyset$ $\otimes$ $U_{y,t}$
$\max_x$ $\oplus$
$\max_z$ $\oplus$
$\emptyset$ $\otimes$ $U_{z,t}$
$\emptyset$ $\otimes$ U_x
$\emptyset$ $\otimes$ $U_{x,y}$
n_t
n_t
$\max_y$ $\oplus$ $U_{y,t}$
$\emptyset$ $\otimes$
$\max_z$ $\oplus$ $U_{z,t}$
n_t
$\max_x$ $\oplus$ U_x $U_{x,y}$
(a)
(b)
(c)
(d)

Figure 7.9: Example of a specific cluster-tree decomposition for a max computation node: (a) primal graph of the graphical model to be decomposed; (b) rooted cluster-tree decomposition; (c) MCDAG with utility functions weighted by plausibilities; (d) final MCDAG where weights are put as high as possible and where useless clusters are removed.

Theorem 7.46 proves that the obtained MCDAG still enables us to compute the answer to a query and to find optimal decision rules for the decision variables. Optimal decision rules can be recorded on the separators of the MCDAG (the separator between two clusters c and $s \in Sons(c)$ is $V(c) \cap V(s)$).

Theorem 7.46. *The value of the MCDAG obtained after having decomposed the macrostructure is equal to the answer to the query. Moreover, for every non duplicated decision variable* x, *optimal decision rules for* x *in the MCDAG are also optimal in* $Ans(Q)$.

Merging some computations

Some clusters in the MCDAG may perform exactly the same computations, even if the computation nodes they come from are distinct. For example, a computation node $n_1 = (\sum_{x,y}, \times, \{P_x, P_{y|x}, U_{y,z})$ may be decomposed into one cluster c_1 such that $val(c_1) = \sum_x (P_x \cdot P_{y|x})$ and one cluster c_1' such that $val(c_1') = \sum_y (U_{y,z} \cdot val(c_1))$. A computation node $n_2 = (\sum_{x,y}, \times, \{P_x, P_{y|x}, U_{y,t})$ may be decomposed into one cluster c_2 such that $val(c_2) = \sum_x (P_x \cdot P_{y|x})$ and one cluster c_2' such that $val(c_2') = \sum_y (U_{y,t} \cdot val(c_2'))$. As $val(c_1) = val(c_2)$, clusters c_1 and c_2 can be merged in order to save some computations. Detecting common clusters is not as easy as detecting common computation nodes.

Figure 7.10 is an example of MCDAG obtained from a DAG of computation nodes $CNDAG(Q)$ thanks to cluster-tree decompositions.

7.4.3 Comparison with an unstructured approach

Definition 7.47. *Let Q be a query. The width of $CNDAG(Q)$, denoted $w_{CNDAG(Q)}$, is the minimal width of a MCDAG which can be obtained from $CNDAG(Q)$ using cluster-tree decompositions.*

Proposition 7.48. *Let $Q = (Sov, (V, G, P, F, U))$ be a query. Computing $Ans(Q)$ with a variable elimination algorithm on a MCDAG associated with Q is time $O((1+|U|) \cdot (1+|P|) \cdot d^{1+w_{CNDAG(Q)}})$ and space $O(|P \cup U| \cdot d^{1+w_{CNDAG(Q)}})$.*

Theorem 7.49 below shows that non surprisingly, structuring multi-operator queries can only decrease the tree-width. This entails that in terms of tree-width (or induced-width), a variable elimination algorithm on a MCDAG is as least as good as algorithm **VE-answerQ** given in the previous chapter.

Theorem 7.49. *Let $Q = (Sov, \mathcal{N})$ be a query on a PFU network $\mathcal{N} = (V, G, P, \emptyset, U)$. Let $\mathcal{G} = (V, \{sc(\varphi), \varphi \in P \cup U\})$ be the hypergraph associated with $\mathcal{N}$. Then, $w_{CNDAG(Q)} \leq w_{\mathcal{G}}(\preceq_{Sov})$.*

7.4.4 Comparison with existing approaches

Compared to existing architectures for example on influence diagrams, MCDAGs can be exponentially more efficient by strongly decreasing the tree-width, thanks to (1) the duplication technique, (2) the analysis of extra reordering freedoms, and (3) the use of normalizations conditions. One can compare these three points with existing works:

- The idea behind duplication is to use all the decompositions (independences) available in influence diagrams. An influence diagram actually expresses independences both on the global probability distribution and on the global utility function. MCDAGs separately use these two kinds of independences, whereas a potential-based approach uses a kind of weaker "mixed" independence relation. Using the duplication mechanism during the construction of the MCDAG is better in terms of induced-width than using it "on-the-fly" as in [33]. [10]

[10] E.g., for the quite simple influence diagram used in Figure 7.7, the algorithm in [33] gives 2 as an induced-width, whereas MCDAGs give an induced-width of 1. The reason is that MCDAGs allow to eliminate both r_1 before r_2 in the subproblem corresponding to U_{d,r_2} and r_2 before r_1 in the subproblem corresponding to U_{d,r_1}.

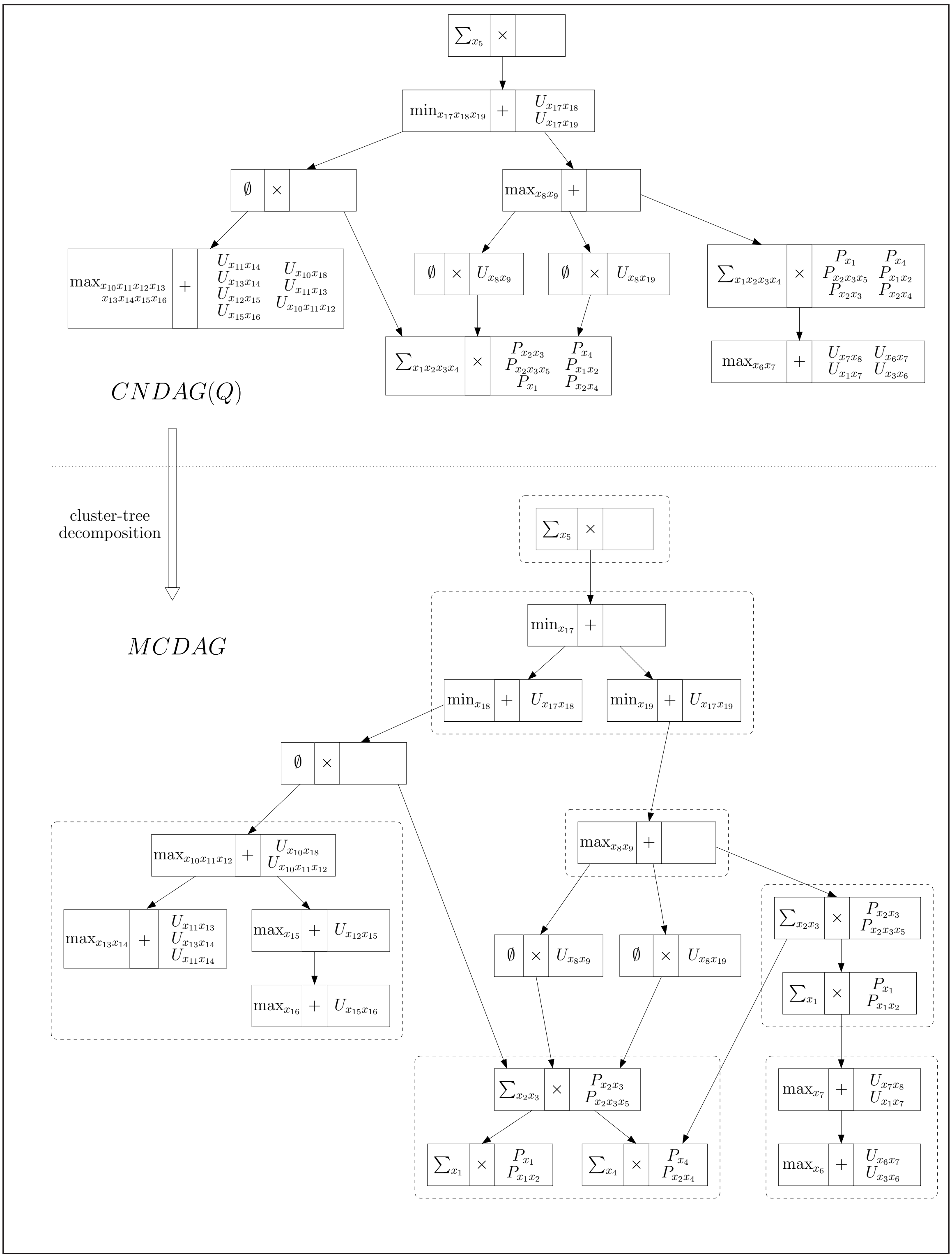

$\sum_{x_5}$ $\times$
$\min_{x_{17}x_{18}x_{19}}$ $+$ $U_{x_{17}x_{18}}$ $U_{x_{17}x_{19}}$
$\emptyset$ $\times$
$\max_{x_8x_9}$ $+$
$\max_{x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}}$ $+$ $U_{x_{11}x_{14}}$ $U_{x_{13}x_{14}}$ $U_{x_{12}x_{15}}$ $U_{x_{15}x_{16}}$ $U_{x_{10}x_{18}}$ $U_{x_{11}x_{13}}$ $U_{x_{10}x_{11}x_{12}}$
$\emptyset$ $\times$ $U_{x_8x_9}$
$\emptyset$ $\times$ $U_{x_8x_{19}}$
$\sum_{x_1x_2x_3x_4}$ $\times$ P_{x_1} $P_{x_2x_3x_5}$ $P_{x_2x_3}$ P_{x_4} $P_{x_1x_2}$ $P_{x_2x_4}$
$\sum_{x_1x_2x_3x_4}$ $\times$ $P_{x_2x_3}$ $P_{x_2x_3x_5}$ P_{x_1} P_{x_4} $P_{x_1x_2}$ $P_{x_2x_4}$
$\max_{x_6x_7}$ $+$ $U_{x_7x_8}$ $U_{x_1x_7}$ $U_{x_6x_7}$ $U_{x_3x_6}$
$CNDAG(Q)$
cluster-tree decomposition
$MCDAG$
$\sum_{x_5}$ $\times$
$\min_{x_{17}}$ $+$
$\min_{x_{18}}$ $+$ $U_{x_{17}x_{18}}$
$\min_{x_{19}}$ $+$ $U_{x_{17}x_{19}}$
$\emptyset$ $\times$
$\max_{x_{10}x_{11}x_{12}}$ $+$ $U_{x_{10}x_{18}}$ $U_{x_{10}x_{11}x_{12}}$
$\max_{x_{13}x_{14}}$ $+$ $U_{x_{11}x_{13}}$ $U_{x_{13}x_{14}}$ $U_{x_{11}x_{14}}$
$\max_{x_{15}}$ $+$ $U_{x_{12}x_{15}}$
$\max_{x_{16}}$ $+$ $U_{x_{15}x_{16}}$
$\max_{x_8x_9}$ $+$
$\emptyset$ $\times$ $U_{x_8x_9}$
$\emptyset$ $\times$ $U_{x_8x_{19}}$
$\sum_{x_2x_3}$ $\times$ $P_{x_2x_3}$ $P_{x_2x_3x_5}$
$\sum_{x_1}$ $\times$ P_{x_1} $P_{x_1x_2}$
$\sum_{x_2x_3}$ $\times$ $P_{x_2x_3}$ $P_{x_2x_3x_5}$
$\sum_{x_1}$ $\times$ P_{x_1} $P_{x_1x_2}$
$\sum_{x_4}$ $\times$ P_{x_4} $P_{x_2x_4}$
$\max_{x_7}$ $+$ $U_{x_7x_8}$ $U_{x_1x_7}$
$\max_{x_6}$ $+$ $U_{x_6x_7}$ $U_{x_3x_6}$

Figure 7.10: Example of a MCDAG obtained from $CNDAG(Q)$ by cluster-tree decomposition and merging of some clusters performing the same computation.

- Weakening constraints on the elimination order can be linked with the usual notion of *relevant information* for decision variables. With MCDAGs, this notion is not used only for the sake of conciseness of decision rules: it is also used to reveal reordering freedoms, which can decrease the time complexity. Also, some of the ordering freedoms here are obtained by synergism with the duplication.

- Thanks to simplification rule SR, the normalization conditions enable us not only to avoid useless computations, but also to improve the architecture structure (SR may indirectly weaken some constraints on the elimination order). This is stronger than Lazy Propagation architectures [85], which use the first point only.

Last, the MCDAG architecture contradicts a common belief that using *division operations* is necessary to solve influence diagrams with VE algorithms.

If one uses our structuration process to structure the computations performed by MDPs, then one exactly gets the value iteration algorithm. However, as soon as the MDP becomes factored, gains can be observed in terms of tree-width. Also, MCDAGs can namely be directly applied to possibilistic influence diagrams using the possibilistic pessimistic utility theory, or to classical planning problems using the boolean optimistic expected disjunctive utility (in order to search for a sequence of decisions to reach one goal of a set of goal states).

7.4.5 Adding feasibilities

As said in the semiring case, feasibilities have a very specific status from the duplication property point of view. A simple solution to integrate them is to work with potentials, as introduced in Definition 6.11 page 96, and then to use

- the semiring rewriting rules (without duplication)

- the cluster-tree decomposition techniques for semiring computation nodes.

With this approach, the architecture obtained is a MCTree involving:

- several elimination operators: min, max, and $\boxplus$, the elimination operator on potentials given in Definition 6.11 page 96;

- but only combination operator $\boxtimes$ (the combination operator between potentials).

Finer rewriting rules, not yet mature enough, can be described in the case "semigroup with feasibilities". They avoid using potentials which prevent from exploiting some available decompositions.

7.5 Conclusion: a generic computational architecture, the MCDAG architecture

This chapter has shown how to systematically structure multi-operator queries. The structuration process involves two major steps:

- A macrostructuration step using rewriting rules. This step aims at revealing all possible decompositions and reordering freedoms, and at exploiting normalization conditions.

- A cluster-tree decomposition step. This step exploits the freedoms in the elimination order. It provides us with the MCTree architecture in the semiring case (cf. Definition 7.22 page 121), and with the MCDAG architecture in the semigroup case (cf. Definition 7.40 page 131). These two architectures satisfy unicity and soundness properties. Also, they lead to a better induced-width (or tree-width). Compared to existing variable elimination-based computational architectures, the MCDAG architecture we introduce is the only one which uses both multiple elimination operators and multiple combination operators.

As MCTrees are particular instances of MCDAGs, we actually obtain a unique generic computational architecture, the MCDAG one, which can be used both in the semiring and semigroup cases. This allows us not to consider the semigroup and semiring cases separately anymore, as illustrated in Figure 7.11.

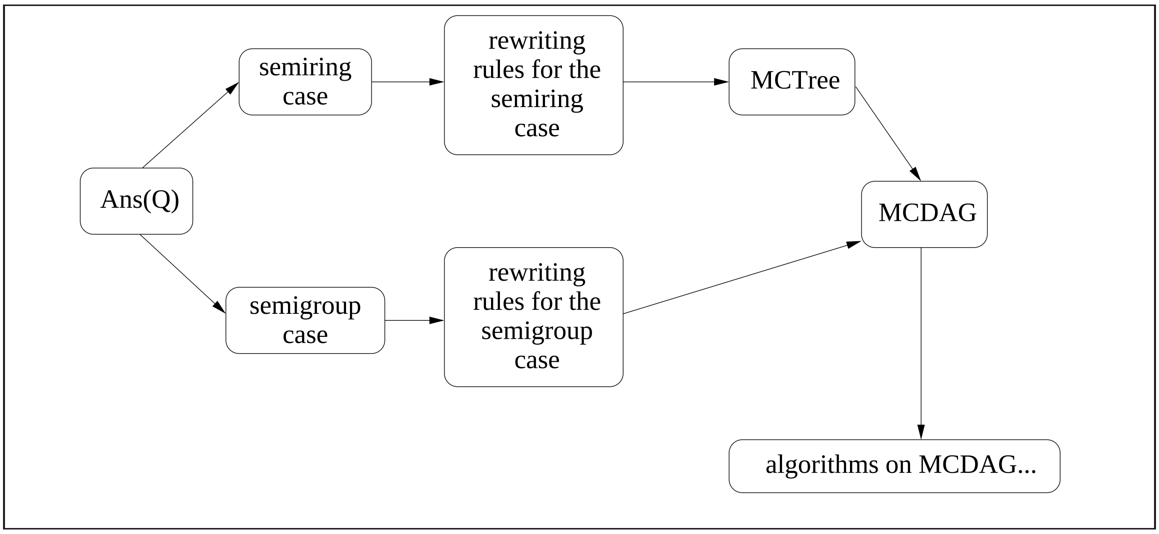

semiring case
rewriting rules for the semiring case
MCTree
Ans(Q)
MCDAG
semigroup case
rewriting rules for the semigroup case
algorithms on MCDAG...

Figure 7.11: Towards a unique computational architecture.

Another way to formulate this conclusion is that the computation of $Ans(Q)$ and of optimal decision rules for a query Q can be reduced to the following problem:

Let $(E, \oplus, \otimes)$ be a totally ordered MCS.
Let M be a MCDAG involving scoped functions taking values in E and clusters
using $(\oplus^c, \otimes^c) \in \{(\min, \oplus), (\max, \oplus), (\min, \otimes), (\max, \otimes), (\oplus, \otimes)\}$
Compute the value of M and optimal decision rules for the decision variables.

The generic variable algorithm proposed in this chapter consists in saying that as soon as a cluster c has received $val(s)$ from all its children $s \in Sons(c)$, it computes its own value $val(c) = \oplus^c{}_{V(pa(c))-V(c)} \left(\left(\otimes^c{}_{\varphi \in \Phi(c)} \varphi \right) \otimes^c \left(\otimes^c{}_{s \in Sons(c)} val(s) \right) \right)$ and sends it to each of its parents. The value of the root cluster then equals the answer to the query.

For each cluster c, $val(c)$ can be computed either by eliminating variables in $V(pa(c)) - V(c)$ step-by-step, as done in this chapter, or by considering all variables in $V(pa(c)) - V(c)$ simultaneously. The latter approach, known as a *Cluster-Tree Elimination* (CTE [7]) algorithm, generalizes VE algorithms and yields the same theoretical time complexity together with a better space complexity, exponential in the size of the largest separator between two clusters in the MCDAG. Such methods were also used in the litterature under the names of dynamic programming, junction tree algorithm, or perfect relaxation [90].

Even if these algorithms can answer queries, they use neither backtrack nor branch and bound techniques. The next step is to enhance the MCDAG architecture with tree search techniques able to prune the search space. Such an enhancement is the objective of the next chapter.

Chapter 8

A generic structured tree search on the MCDAG architecture

Answering a PFU query is equivalent to computing the value of a MCDAG. This can be achieved using a quite natural variable elimination (VE) or cluster-tree elimination (CTE) algorithm which computes stepwise the value of each cluster of the MCDAG, from the leaves to the root. The VE algorithm offers a time complexity exponential in the MCDAG-width, but at the price of a space complexity exponential in the MCDAG-width too. The CTE algorithm gives the same time complexity and a space complexity exponential in the size of the largest separator between two clusters.

At the same time, a search technique as depth-first tree search provides a linear space complexity. Moreover, despite its greater theoretical time complexity, tree search often outperforms variable elimination algorithms in practice, especially when it is enhanced with bound techniques pruning the search space.

In order to benefit both from the practical efficiency of tree search and from the good theoretical time complexity of variable elimination, we introduce a generic structured tree search algorithm which takes advantage of the structural decompositions expressed by the MCDAG architecture. Such an idea is not new. In particular, several tree search schemes exploiting problems structures were defined in the last decade [26, 65, 38].

However, these existing schemes are basically designed to compute sequences of mono-operator eliminations on a mono-operator combination of scoped functions. This mono-operator nature significantly facilitates the way bounds can be used to prune the search space. Also, the existing schemes tackle either problems using specific combination and elimination operators, or problems built upon an algebraic structure making assumptions stronger than those made with a totally ordered MCS (cd Definition 6.6 page 94).

As a result, structured tree search algorithms capable of handling the multi-operator nature of generic PFU queries (or, equivalently, of generic MCDAGs) are needed. As previously mentioned, this raises new questions concerning the use of bounds in the context of alternating min-, max-, and $\oplus$-eliminations.

8.1 Existing structured tree search algorithms

Before defining a new algorithm on MCDAGs, we briefly explain how existing structured tree search proposals work. Three proposals are presented: algorithms on AND/OR search spaces [38], recursive conditioning [26], and BTD (Backtrack bounded by Tree Decomposition [65]).

AND/OR search spaces [38] enable mono-operator eliminations on a mono-operator combination of scoped functions of a graphical model $\mathcal{M}$ to be computed, and can be used to solve problems associated with CSPs or BNs. The simplest form of AND/OR search spaces is AND/OR search trees. They exploit independences represented thanks to a *pseudo-tree* of the primal graph of $\mathcal{M}$. Such an approach was initially defined in [53].

Definition 8.1. *Given an undirected graph $G = (V, E)$, a rooted tree $T = (V, E')$ is a pseudo-tree of G iff any edge in $E - E'$ is an edge connecting a vertex to one of its ancestors in T.* [1]

Figure 8.1(b) shows an example of pseudo-tree associated with the graphical model depicted in Figure 8.1(a). A pseudo-tree induces a search space called an AND/OR search tree. An AND/OR search tree is a tree containing two types of nodes: (1) OR nodes, labeled with a variable $x \in V$, and (2) AND nodes, labeled with an assignment (x, a). The successors of an OR node x are AND nodes (x, a), one for each $a \in dom(x)$, while the successors of an AND node (x, a) are OR nodes y, one for each son of x in the pseudo-tree. The AND/OR search tree associated with the pseudo-tree of Figure 8.1(b) is given in Figure 8.1(c).

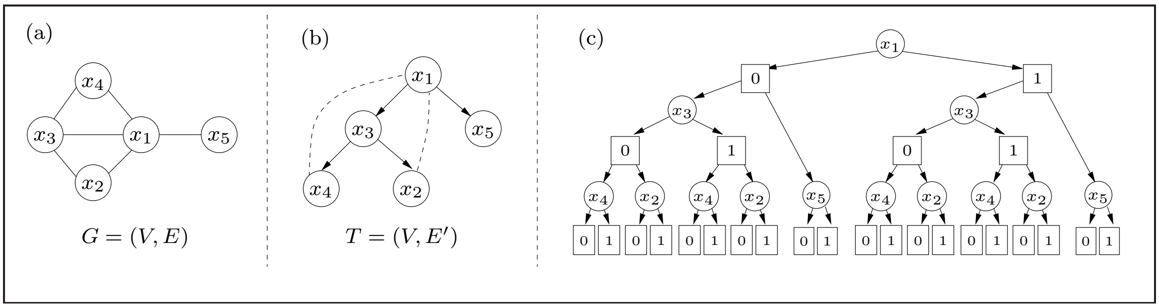


Figure 8.1: Example of AND/OR search tree: (a) Primal graph of the graphical model $\mathcal{M} = (\{x_1, x_2, x_3, x_4, x_5\}, \{\varphi_{x_1x_2x_3}, \varphi_{x_1x_3x_4}, \varphi_{x_1x_5}\})$; (b) A pseudo-tree of this primal graph (dotted lines represent edges of the primal graph which are not in the pseudo-tree); (c) AND/OR search tree obtained from the pseudo-tree (with boolean variables).

Informally, an AND/OR search tree expresses that the subproblems rooted at an OR node x are independent and can be processed separately. For instance, as soon as x_1 is assigned, variables in $\{x_2, x_3, x_4\}$ and x_5 become independent, as soon as x_1 and x_3 are assigned, x_2 and x_4 become independent. In this sense, an AND/OR search tree enables one to define a kind of structured tree search. This is interesting because it can be much faster to explore an AND/OR search tree than a standard search tree which assigns variables linearly. With an AND/OR search tree, the time complexity becomes exponential in the height of the pseudo-tree. [2]

From AND/OR search trees, other search spaces, yielding different time and space complexities, can be defined, such as AND/OR search graphs, obtained by merging equivalent nodes in the

[1] Examples of pseudo-trees are DFS spanning trees, whose edges are obtained by building a spanning tree of the primal graph of G, using an edge selection heuristic called Depth First Search (DFS).

[2] An optimal height is $O(w \cdot log(|V|))$, where w is the tree-width of the graphical model [70, 14, 5].

AND/OR tree. AND/OR search graphs can induce the same time and space complexities as VE and CTE algorithms. Algorithms on AND/OR search graphs, which use caching, can be tuned depending on the memory size available

Recursive conditioning (RC [26]) is an algorithm for exact inference in Bayesian networks. It exploits the structure of a BN as follows. Given an initial BN, RC conditions on a set of variables S of the BN (i.e. it assigns a set of variables of the BN), so that the removal of the variables in S yields two disconnected subnetworks. S is called a *cutset*. Each disconnected subnetwork is solved independently by using the same mechanism. This recursive process is applied until subnetworks contain a unique variable. In the example of Figure 8.1(a), we can first condition on $\{x_1\}$, and doing so, we create two disconnected subnetworks. The first subnetwork contains only variable x_5. The second one, containing variables x_2, x_3, and x_4, can itself be split by conditioning on x_3, which creates two disconnected subnetworks containing one variable only.

In fact, an implicit tree structure, called a *dtree*, exists behind the conditioning mechanism and can be used to find good cutsets.

Definition 8.2. *A dtree for a BN (V, G, P) is a rooted binary tree whose leaves correspond to the conditional probabilities in P. The set of variables involved in a leaf is the scope of the conditional probability distribution associated with this leaf.*

Given a node in the dtree, the cutset associated with it is the set of variables shared between its left and right subtrees. In the end, RC can be seen as structured tree search exploring a dtree by assigning cutsets in a depth-first manner. In order to avoid redundant computations, RC can also trade time for space by using caching strategies. It can also be tuned depending on the memory size available. This makes RC an *any-space* algorithm capable of providing a space complexity exponential in the size of the largest cutset and a time complexity exponential in the BN tree-width w, as well as a linear space complexity and a time complexity exponential in $w \cdot log(|V|)$.

The BTD algorithm (Backtrack bounded by Tree Decomposition [65]) also achieves a structured tree search to solve CSPs or valued CSPs. Compared to AND/OR search spaces and RC, which use pseudo-trees and dtrees, BTD uses standard cluster-tree decompositions (as defined in Definition 7.18 page 118), which are the main topic of many existing works [116, 2, 115, 73, 13, 76].

Given a rooted cluster-tree decomposition, the BTD algorithm first performs stepwise assignments of the variables in the root cluster. If the VCSP considered corresponds to a cost minimization task, backtrack occurs if the cost provided by the current assignment is too high. If all variables in the root cluster c_0 are assigned, then a son cluster c_1 of c_0 is explored. This means that unassigned variables in c_1 are assigned step-by-step. Again, backtrack occurs if the cost of the current assignment is greater than some upper bound. If all variables in c_1 are assigned, then a son cluster of c_1 is explored similarly. If there is no unexplored son cluster, backtrack occurs. BTD additionally uses recording on cluster *separators*.

Definition 8.3. *The* separator *between a cluster c and one of its sons $s \in Sons(c)$ is the set of variables defined by $sep(c, s) = V(c) \cap V(s)$, also denoted improperly $c \cap s$.*

Given an assignment A of the ancestors of s, the cluster-tree structure entails that the value $val(s)(A)$ given by cluster s only depends on the assignment $A^{\downarrow sep(c,s)}$ of the separator $sep(c, s)$.

Hence, if $val(s)(A)$ is computed once and recorded, then it is useless to compute $val(s)(A')$ for all assignments A' such that $A^{\downarrow sep(c,s)} = A'^{\downarrow sep(c,s)}$. Local consistencies are also used during the search in order to get bounds on the cost of any extension of the current assignment [65, 29]. If these bounds violate requirements imposed for example by the best solution found so far, then backtrack occurs.

From existing works to a generic structured tree search on MCDAGs The three previous algorithms present many similarities, since pseudo-trees, dtrees, and cluster-tree decompositions share many common properties (but we do not know formal works establishing precisely equivalence relations between these structures). In order to develop a generic structured tree search, we choose to start from the BTD algorithm, since the MCDAG obtained after the query structuration process is closer to a cluster-tree decomposition than to a pseudo-tree or a dtree.

We incrementally present a generalized BTD algorithm on MCDAGs, starting from a structured tree search without bounds and caching to a structured tree search using both bounds and caching. As we shall see, the main difficulty in adapting the BTD algorithm to MCDAGs resides in the use of bounds to prune the search space.

In the sequel, we assume without loss of generality that there are no free variables in the query. If the set of free variables V_{fr} is not empty, it suffices to call the forthcoming algorithms once for each assignment of V_{fr}. Also, we assume that there are no feasibilities. The integration of feasibilities is discussed in Section 8.7.

8.2 A first generic structured tree search

The first algorithm we define simply traverses the MCDAG from the root to the leaves, instead of propagating information from the leaves to the root as in a variable elimination scheme. We first introduce a definition essential for the understanding of the rest of the chapter.

Definition 8.4. *Let c be a cluster of a MCDAG. Let $V \subset V(c) - V(pa(c))$ be a subset of the variables to eliminate in c. Let $\Phi \subset \Phi(c)$ be a subset of the scoped functions associated with c. Let A be an assignment of the variables involved in the ancestors of c in the MCDAG and in $V(c) - V$. We define $val(c, A, V, \Phi)$ by*

$$val(c, A, V, \Phi) = \bigoplus_{V}^{c} \left(\left(\bigotimes_{\varphi \in \Phi}^{c} \varphi(A) \right) \otimes^{c} \left(\bigotimes_{s \in Sons(c)}^{c} val(s)(A) \right) \right)$$

where $val(s)(A)$ is given by Definition 7.41 page 131.

In other words, $val(c, A, V, \Phi)$ corresponds to the elimination of the variables in V on the combination of the scoped functions in Φ together with the values of the son clusters of c. This is realized for assignment A and using the elimination operator $\oplus^c$ and the combination operator $\otimes^c$ of cluster c.

Proposition 8.5. *Let M be a MCDAG associated with a query Q. Let c be a cluster in M.*

(a) Let r be the root cluster of M. Then, $Ans(Q) = val(r, \emptyset, V(r), \Phi(r))$.

(b) $\forall x \in V,\ val(c, A, V, \Phi) = \bigoplus^c_{a \in dom(x)} \left(\left(\bigotimes^c_{\varphi \in \Phi_0} \varphi(A.(x, a)) \right) \otimes^c val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0) \right),$ *where* $\Phi_0 = \{\varphi \in \Phi \mid sc(\varphi) \cap (V - \{x\}) = \emptyset\}$

(c) $val(c, A, \emptyset, \Phi) = \left(\bigotimes^c_{\varphi \in \Phi} \varphi(A) \right) \otimes^c \left(\bigotimes^c_{s \in Sons(c)} val(s, A, V(s) - V(c), \Phi(s)) \right)$

Proposition 8.5 helps us define a first generic structured tree search. More precisely, Proposition 8.5(a) says that in order to answer a query Q whose associated MCDAG has r as a root, it suffices to compute $val(r, \emptyset, V(r), \Phi(r))$. A recursive use of Proposition 8.5(b) then gives a method to compute $val(r, \emptyset, V(r), \Phi(r))$, by assigning step-by-step the variables in $V(r)$. Once all variables in $V(r)$ are assigned, quantities like $val(r, A, \emptyset, \Phi)$ must be computed. Proposition 8.5(c), then says that $val(r, A, \emptyset, \Phi) = ({\otimes^c}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left({\otimes^c}_{s \in Sons(r)}\, val(s, A, V(s) - V(r), \Phi(s))\right)$. Each $val(s, A, V(s) - V(r), \Phi(s))$ can be computed by using Proposition 8.5(b) again. Therefore, an alternation of applications of Propositions 8.5(b) and 8.5(c) enables us to compute $Ans(Q)$.

The associated generic structured tree search on MCDAGs, directly defined from Proposition 8.5, is called **TS-mcdag** (like "Tree Search on MCDAG") and is shown in Figure 8.2. As it uses structured problems, it is expected to be much more efficient than the unstructured tree search algorithm given in Section 6.1.

```
TS-mcdag($c, A, V, \Phi$)
begin
    if ($V = \emptyset$) then
        $\mathcal{S} \leftarrow Sons(c)$
        $val \leftarrow {\otimes^c}_{\varphi \in \Phi}\, \varphi(A)$
        while $\mathcal{S} \neq \emptyset$ do
            Choose $s \in \mathcal{S}$
            $\mathcal{S} \leftarrow \mathcal{S} - \{s\}$
            $val \leftarrow val \otimes^c$ TS-mcdag($s, A, V(s) - V(c), \Phi(s)$)
        return ($val$)
    else
        Choose $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi(c)\, ,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $val \leftarrow \Diamond$
        while $d \neq \emptyset$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val \leftarrow val \oplus^c(({\otimes^c}_{\varphi \in \Phi_0}\, \varphi(A.(x, a))) \otimes^c$ TS-mcdag($c, A.(x, a), V - \{x\}, \Phi - \Phi_0$))
        return ($val$)
end
```

Figure 8.2: A generic structured tree search algorithm on a MCDAG.

The first call is **TS-mcdag**$(r, \emptyset, V(r), \Phi(r))$. **TS-mcdag**$(c, A, V, \Phi)$ actually computes the quantity $val(c, A, V, \Phi)$. If the set V of unassigned variables is empty, then the value of each son cluster is computed, as specified in Proposition 8.5(c). Otherwise, if $V \neq \emptyset$, then a variable x to be assigned is chosen, and the computations specified in Proposition 8.5(b) are performed.

Proposition 8.6. *Algorithm **TS-mcdag** is sound and complete, i.e. it returns* $Ans(Q)$.

Proposition 8.7. *Let* M *be a MCDAG associated with a query* $Q = (Sov, (V, G, P, \emptyset, U))$. *Then, the space complexity of algorithm **TS-mcdag** is* $O(h \cdot (d + m))$, *and its time complexity is*

$$O(m \cdot \mu \cdot d^h),$$

where d is the maximum domain size, h is the MCDAG-height, μ is the maximum number of parents of a node in the MCDAG ($\mu = 1$ if the MCDAG is a MCTree), and $m = |P \cup U|$ in the semiring case and $m = (1 + |P|)(1 + |U|)$ in the semigroup case.

Proposition 8.7 shows that in addition to the MCDAG-width, the MCDAG-height can be a criterion to search for good cluster-tree decompositions.

8.3 Adding caching to the structured tree search

Algorithm **TS-mcdag** may perform many redundant computations, and it is possible to trade space for time thanks to some caching.

Indeed, let c be a cluster, let $s \in Sons(c)$, and let $V_{anc(s)}$ be the set of variables involved in the ancestors of s in the MCDAG. Let A_{sep} be an assignment of $sep(c, s)$. For all assignments A, A' of $V_{anc(s)} - V(s)$, the MCDAG structure entails that $val(s)(A.A_{sep}) = val(s)(A'.A_{sep})$. **TS-mcdag** does not use this structural property at all and computes $val(s)(A.A_{sep}) = val(s, A.A_{sep}, V(s) - V(c), \Phi(s))$ for every assignment A of $V_{anc(s)} - V(s)$. If there are 10 boolean variables in $V_{anc(s)} - V(s)$, this means that the same computation is performed 2^{10} times instead of once.

A solution to this problem is to record the result of evaluations of quantities such as $val(s)(A)$, which actually equals $val(s)(A^{\downarrow c \cap s})$. The value recorded for $val(s)(A^{\downarrow c \cap s})$ is denoted $rec(s, A^{\downarrow c \cap s})$. It equals **nil** if no value is recorded. The space required for this caching depends on the size of the separators, which can therefore be another parameter quantifying the quality of a cluster-tree decomposition. The updated algorithm, called **RecTS-mcdag** as TS-mcdag with Recording, is shown in Figure 8.3.

```
RecTS-mcdag($c, A, V, \Phi$)
begin
    if ($V = \emptyset$) then
        $\mathcal{S} \leftarrow Sons(c)$
        $val \leftarrow \otimes^c{}_{\varphi \in \Phi}\, \varphi(A)$
        while $\mathcal{S} \neq \emptyset$ do
            Choose $s \in \mathcal{S}$
            $\mathcal{S} \leftarrow \mathcal{S} - \{s\}$
            if $\underline{(\mathbf{rec}(\mathbf{s}, \mathbf{A}^{\downarrow \mathbf{c} \cap \mathbf{s}}) = \mathbf{nil})}$ then $\underline{\mathbf{rec}(\mathbf{s}, \mathbf{A}^{\downarrow \mathbf{c} \cap \mathbf{s}}) \leftarrow \mathbf{RecTS\text{-}mcdag}(\mathbf{s}, \mathbf{A}, \mathbf{V}(\mathbf{s}) - \mathbf{V}(\mathbf{c}), \mathbf{\Phi}(\mathbf{s}))}$
            $val \leftarrow val \otimes^c rec(s, A^{\downarrow c \cap s})$
        return ($val$)
    else
        Choose $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\,,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $val \leftarrow \Diamond$
        while $d \neq \emptyset$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val \leftarrow val \oplus^c ((\otimes^c{}_{\varphi \in \Phi_0}\, \varphi(A.(x, a))) \otimes^c$ RecTS-mcdag$(c, A.(x, a), V - \{x\}, \Phi - \Phi_0))$
        return ($val$)
end
```

Figure 8.3: A structured tree search algorithm using caching.

Proposition 8.8. *Algorithm **RecTS-mcdag** is sound and complete, i.e. it returns $Ans(Q)$.*

Proposition 8.9. *Let M be a MCDAG associated with a query $Q = (Sov, (V, G, P, \emptyset, U))$. Let w be the MCDAG-width. Computing $Ans(Q)$ with algorithm **RecTS-mcdag** on the MCDAG M is time $O(m \cdot d^{w+1})$, where $m = |P \cup U|$ in the semiring case and $m = (1 + |P|) \cdot (1 + |U|)$ in the semigroup case. The space complexity is $O(N \cdot s \cdot d^s)$, where N is the number of clusters in the MCDAG and s is the size of the largest separator.*

In fact, algorithm **RecTS-mcdag** has the same time and space complexities as a cluster-tree elimination algorithm on a MCDAG, and it performs the same computations. The only difference is the order in which these computations are made (top-down or bottom-up processing in the MCDAG).

However, an advantage of **RecTS-mcdag** is that it can easily be tuned to an any-space version: if the amount of space required for caching is greater than the memory size available, some recorded values can simply be destroyed, at the price of a greater time complexity.

8.4 A structured tree search using both bounds and caching

One of the main interest of tree search is to prune the search space using bounds, which leads to so-called *branch-and-bound* techniques. These techniques can improve both the practical time and space complexities, since they can allow some recordings on useless parts of the search space to be avoided. In other words, by pruning the search space, bounds can enable us to avoid considering all instantiations of all separators.

8.4.1 A small additional algebraic assumption

For some bounds initializations, we need an additional algebraic assumption enabling us to consider totally $\preceq$-ordered MCS $(E, \oplus, \otimes)$ having a minimum element $\perp$ and a maximum element $\top$. Some of the MCS considered, such as $(\{t, f\}, \vee, \wedge)$, already admit such elements. In fact, if $0_E \preceq 1_E$, then the structure admits $\perp = 0_E$ as a minimum element, and if $1_E \preceq 0_E$, then it admits $\top = 0_E$ as a maximum element.

In the first case ($0_E \preceq 1_E$), we can always add to the structure an element $\top$ such that for all $x \in E \cup \{\top\}$, $x \preceq \top$, $\top \oplus x = x \oplus \top = \top$, and $\top \otimes x = x \otimes \top = \begin{cases} \top \text{ if } x \neq 0_E \\ 0_E \text{ otherwise} \end{cases}$. The structure obtained is still a totally ordered MCS provided that $(x \otimes y = 0_E) \to ((x = 0_E) \vee (y = 0_E))$ holds. The latter property is satisfied in all standard expected utility structures.

In the second case ($1_E \preceq 0_E$), it is always possible to invert $\preceq$, which gives a total order $\preceq'$ such that $0_E \preceq' 1_E$, and then to perform a similar extension.

To sum up, *we consider totally ordered MCS equipped with a minimum element* $\perp = 0_E$ *and a maximum element* $\top$ in the following.

8.4.2 Using bounds in presence of several elimination operators

A first difficulty in adapting branch-and-bound techniques to MCDAGs is to handle bounds in the context of alternating multiple elimination operators. A good starting point to solve this problem is the alpha-beta algorithm [74] used in game theory, where min and max operators alternate. This algorithm is briefly described below.

Let us consider a two-player game whose game tree is shown in Figure 8.4(a). Each internal node corresponds to a choice of one player, and branches below this node correspond to the different possible moves. An internal node is labeled with min or max, depending on which player controls the associated move. Each leaf node is labeled with a value which evaluates the position obtained if the players play as indicated by the path from the root to this leaf.

A first method to compute the best first move is to perform a depth-first tree search computing the value $v(n)$ of each max (resp. min) node n as the maximum (resp. minimum) value of its children. The value of each node as well as the best first move are given in Figure 8.4(a), which shows that the max-player can achieve a value of 4. The corresponding algorithm, called the *MiniMax* algorithm, explores the whole game tree without using bounds.

The MiniMax algorithm actually performs useless computations because several nodes in the game tree of Figure 8.4(a) do not need to be considered. For example, in Figure 8.4(b), after the exploration of the two first branches of A, we know that min-node A can achieve a value lesser than 4. The exploration of the first value of max-node B shows that the value of B is greater than 5 and consequently is useless for the computation of the value of A. Pruning can occur. Similarly, after the exploration of the two first branches of min-node C, the value of C is known to be lesser than 3. This means that if the max-player chooses the move corresponding to the second branch of the root, the min-player can achieve a value of 3. As the first branch of the root gives a value of 4, the max player will never choose the second move, and consequently exploring the rest of the second branch is useless: pruning can occur.

The alpha-beta algorithm enables one to exactly know when the search space can be pruned. Technically speaking, it uses two bounds called α and β, in $\mathbb{Z} \cup \{-\infty, +\infty\}$. During search, each node n needs to satisfy a requirement such as $\alpha < v(n) < \beta$, which means that α is a lower bound and β is an upper bound. If $\alpha = -\infty$, this means that there is no lower requirement, and if $\beta = +\infty$, this means that there is no upper requirement. During the tree exploration, min-nodes can decrease the upper bound β, which means that a min-node always seeks values worse than the ones found so far. At the same time, max-nodes can increase lower bound α, which means that a max-node always seeks values better than the ones found so far. Pruning occurs when $\beta \leq \alpha$, i.e. when the requirements on a node value cannot be satisfied.

Alpha-beta techniques have also been adapted for stochastic games [4], where min, max, and $+$ operators alternate.

In the context of a structured tree search on MCDAGs, we simply use a lower bound LB and an upper bound UB, as in the alpha-beta algorithm. This enables us to deal with both several elimination operators and bounds. Informally, min-clusters will tighten UB while max-clusters will tighten LB.

Also, in order to have counterparts of $-\infty$ and $+\infty$, which mean that there is no lower or upper requirement respectively, we introduce two new elements denoted $\bot^-$ and $\top^+$. Given a totally ordered MCS $(E, \oplus, \otimes)$, $\bot^-$ is an element outside of E which is lesser than any element in E, and $\top^+$ is an element outside of E which is greater than any element in $E \cup \{\bot^-\}$.

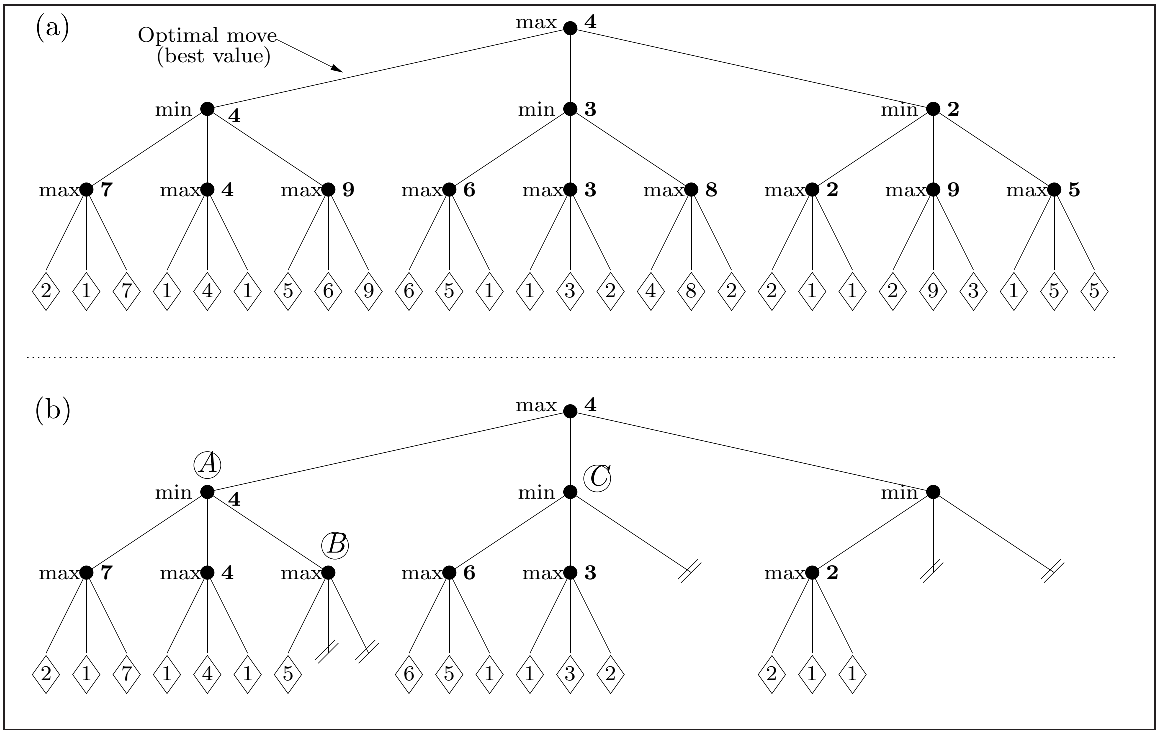


Figure 8.4: Example of alpha-beta pruning: (a) Game tree explored by the MiniMax algorithm; (b) Pruned game tree explored by the alpha-beta algorithm.

8.4.3 Using bounds without inverse for the combination operations

A second difficulty consists in dealing simultaneously with bounds and combination operations, mainly because the algebraic structure we use, a MCS $(E, \oplus, \otimes)$, does not assume the existence of inverse operations for $\oplus$ or $\otimes$. This problem does not appear with the alpha-beta algorithm because it manipulates a unique global function providing the leaves values.

Due to the factorization into local functions, one may want to impose a requirement like $e_\otimes \otimes val \prec UB$ on some values val to be computed, where $e_\otimes$ is a factor which must be combined with val using $\otimes$. Since we do not assume the existence of a division operation $\oslash$, one cannot directly impose $val \prec UB \oslash e_\otimes$ and take $UB' = UB \oslash e_\otimes$ as a new simple upper bound for val.

The same holds for requirements such as $val \oplus e_\oplus \prec UB$, where $e_\oplus$ is a factor which must be combined with val using $\oplus$, because we do not assume the existence of a difference operation $\ominus$ inverse of $\oplus$.

In the end, we need to be able to enforce complex requirements such as $e_\otimes \otimes val \oplus e_\oplus \prec UB$ or $LB \prec e_\otimes \otimes val \oplus e_\oplus$. Furthermore, factors $e_\otimes$ and $e_\oplus$ may not even be exactly known. Only lower and upper bounds $lb_\otimes$ and $ub_\otimes$ on $e_\otimes$ and lower and upper bounds $lb_\oplus$ and $ub_\oplus$ on $e_\oplus$ may be available. In order to manipulate constant factors only (except for the value val to be computed), one can impose the following weaker requirements:

$$(LB \prec ub_\otimes \otimes val \oplus ub_\oplus) \wedge (lb_\otimes \otimes val \oplus lb_\oplus \prec UB) \tag{8.1}$$

This leads us to define the notion of *complex bounds.*

Definition 8.10. *A complex bound is a tuple* $(LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$ *such that* $LB \prec UB$*,*

$lb_\otimes \preceq ub_\otimes$, and $lb_\oplus \preceq ub_\oplus$.

Informally, imposing a complex bound $(LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$ on a quantity val means imposing Equation 8.1. Thanks to complex bounds, some branches of the search space may be cut. That is to say, if a branch of the structured tree must compute $val(c, A, V, \Phi)$ while satisfying a complex bound $\mathcal{B}$, then the exact value of $val(c, A, V, \Phi)$ is not needed if $\mathcal{B}$ is proved to be violated. In order to represent this, we define the notion of *bounded evaluation*.

Definition 8.11. *Let $\mathcal{B} = (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$ be a complex bound. An evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$, is a couple $(lb, ub) \in E^2$ such that $lb \preceq val(c, A, V, \Phi) \preceq ub$, and such that $(lb = ub) \vee (lb_\otimes \otimes lb \oplus lb_\oplus = ub_\otimes \otimes ub \oplus ub_\oplus) \vee (LB \succeq ub_\otimes \otimes ub \oplus ub_\oplus) \vee (UB \preceq lb_\otimes \otimes lb \oplus lb_\oplus)$.*

In other words, an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$ must provide us with lower and upper bounds lb, ub on $val(c, A, V, \Phi)$, such that one of the following conditions holds:

1. $lb = ub$, i.e. we have the exact value of $val(c, A, V, \Phi)$;

2. $lb_\otimes \otimes lb \oplus lb_\oplus = ub_\otimes \otimes ub \oplus ub_\oplus$: in this case, one can infer that $e_\otimes \otimes val(c, A, V, \Phi) \oplus e_\oplus = lb_\otimes \otimes lb \oplus lb_\oplus = ub_\otimes \otimes ub \oplus ub_\oplus$. Informally, this means that whatever the exact local value of $val(c, A, V, \Phi)$ is, knowing lb and ub suffices to ensure that a unique global degree is obtained after combination with the rest of the problem;

3. $LB \succeq ub_\otimes \otimes ub \oplus ub_\oplus$, i.e. the upper bound ub proves that $val(c, A, V, \Phi)$ does not satisfy the requirements imposed by $\mathcal{B}$;

4. $UB \preceq lb_\otimes \otimes lb \oplus lb_\oplus$, i.e. the lower bound lb proves that $val(c, A, V, \Phi)$ does not satisfy the requirements imposed by $\mathcal{B}$.

8.4.4 Algorithm definition

In order to specify a structured tree search algorithm using bounds and caching, we use several functions, which satisfy some specifications:

- A main function called *BTD-mcdag*(), which returns the answer $Ans(Q)$ to a query Q.

- A function $bound(c, A, V, \Phi, val_0)$, which returns a pair (lb, ub) such that $lb \preceq val(c,\ A,\ V,\ \Phi) \preceq ub$. Parameter val_0 is an additional parameter which will be combined with lb and ub after the execution of function *bound*. It can be used to avoid computing too precise bounds when not needed: for example, if $val_0 = 0_E$, then $(lb, ub) = (\bot, \top)$ is sufficient to infer that $val_0 \otimes lb = val_0 \otimes ub = 0_E$.

- Three functions $evalClusterMin(c, A, V, \Phi, \mathcal{B})$, $evalClusterMax(c, A, V, \Phi, \mathcal{B})$, and $evalClusterPlus(c, A, V, \Phi, \mathcal{B})$, which compute an evaluation of $val(c, A, V, \Phi)$ bounded by the complex bound $\mathcal{B}$. We use the generic notation *evalCluster*-$\oplus^c$ to denote one of these functions.

- A function $evalSons(c, A, \Phi, \mathcal{B})$, which computes an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$. It is called *evalSons* because computing $val(c, A, \emptyset, \Phi)$ requires computing the combination of the values of the son clusters of c.

A function satisfying its specifications is said to be sound and complete. A function satisfying its specifications for all clusters c of depth h is said to be sound and complete for clusters of depth h (the depth of a cluster being the size of the longest path from the root of the MCDAG to c). We informally introduce each function and then establish formal soundness and completeness results.

Function BTD-mcdag (Figure 8.5) Given the root r of a MCDAG, **BTD-mcdag**() computes $(lb, ub) \leftarrow$ evalCluster-$\oplus^r(r, \emptyset, V(r), \Phi(r), \mathcal{B}_0)$, using complex bound $\mathcal{B}_0 = (\bot^-, \top^+, 1_E, 1_E, 0_E, 0_E)$. If *evalClusterMin*, *evalClusterMax*, and *evalClusterPlus* satisfy their specifications, then (lb, ub) is an evaluation of $val(r, \emptyset, V(r), \Phi(r))$ bounded by $\mathcal{B}_0$. As $val(r, \emptyset, V(r), \Phi(r)) = Ans(Q)$ (cf. Proposition 8.5(a) page 144), (lb, ub) is an evaluation of $Ans(Q)$ bounded by $\mathcal{B}_0$. It can easily be shown that this means that $lb = ub = Ans(Q)$, because $\mathcal{B}_0$ is an "empty" requirement.

```
BTD-mcdag()
begin
    $r \leftarrow root(MCDAG)$
    $(lb, ub) \leftarrow$ evalCluster-$\oplus^r$ $(r, \emptyset, V(r), \Phi(r), (\bot^-, \top^+, 1_E, 1_E, 0_E, 0_E))$
    return $(lb)$
end
```

Figure 8.5: Main function: **BTD-mcdag**.

Function bound This function can simply return $(\bot, \top)$ as the lower and upper bounds on a quantity $val(c, A, V, \Phi)$. However, more advanced versions can obviously be defined, thanks to techniques discussed in Section 8.6.

Function evalClusterMax (Figure 8.6) This function must return an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. If V is empty, then the bounded evaluation is provided by *evalSons*. Otherwise, the algorithm chooses an unassigned variable $x \in V$ and computes the set Φ_0 of scoped functions whose scope is assigned if x is assigned. As $val(c, A, V, \Phi) = \max_{a \in dom(x)} val(c, A.(x, a), V - \{x\}, \Phi)$, we successively evaluate each $val(c, A.(x, a).V - \{x\}, \Phi) = (\otimes^c{}_{\varphi \in \Phi_0}\, \varphi(A.(x, a))) \otimes^c val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0)$ (while loop).

Informally, the algorithm is designed so that at each iteration of the while loop, (lb, ub) is an evaluation of $\max_{a \in dom(x) - d} val(c, A.(x, a), V - \{x\}, \Phi)$ bounded by $\mathcal{B}$, where d is the set of values of x which have not been considered yet. This property holds at the beginning, where $(lb, ub) = (\bot, \bot)$ and $dom(x) - d = \emptyset$.

At each iteration, a value $a \in d$ is considered. The combination of the scoped functions in Φ_0 gives a value val_0. A lower bound lb' and an upper bound ub' on $val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0)$ are computed thanks to the *bound* function. This implies that $val_0 \otimes^c lb'$ and $val_0 \otimes^c ub'$ are respectively lower and upper bounds on $val(c, A.(x, a), V - \{x\}, \Phi)$. If these lower and upper bounds do not define a bounded evaluation of $val(c, A.(x, a), V - \{x\}, \Phi)$ (test in the "if" block), then a more precise evaluation of $val(c, A.(x, a), V - \{x\}, \Phi)$ is sought, using an updated complex bound which depends on the combination operator used by the max-cluster.

After the "if" block, a bounded evaluation of $val(c, A.(x, a), V - \{x\}, \Phi)$ is available. Lower and upper bounds lb and ub are updated, and the max-cluster may tighten lower bound LB'.

The iterations of the while loop are stopped if all values of x have been considered (case $d = \emptyset$), if the requirements cannot be satisfied (case $LB' \succeq UB$), or if the exact value of $val(c, A, V, \Phi)$ is known (case $lb = \top$, which implies that $lb = ub = val(c, A, V, \Phi) = \top$). If some values a in $dom(x)$ have not been considered during the iterations of the while loop, then, as no upper bound on $val(c, A.(x, a), V - \{x\}, \Phi)$ is available, ub is set to $\top$. Finally, (lb, ub) is returned.

```
evalClusterMax$(c, A, V, \Phi, (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus))$
begin
    if $(V = \emptyset)$ then return (evalSons$(c, A, \Phi, (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus))$)
    else
        Choose $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi \,,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $(lb, ub) \leftarrow (\bot, \bot)$
        $LB' \leftarrow LB$
        while $((d \neq \emptyset) \wedge (LB' \prec UB) \wedge (lb \neq \top))$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val_0 \leftarrow {\otimes^c}_{\varphi \in \Phi_0} \varphi(A.(x, a))$
            $(lb', ub') \leftarrow$ bound$(c, A.(x, a), V - \{x\}, \Phi - \Phi_0, val_0)$
            if $((LB' \prec (ub_\otimes \otimes (val_0 \otimes^c ub')) \oplus ub_\oplus) \wedge (lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus \prec UB) \wedge (val_0 \otimes^c lb' \neq$
            $val_0 \otimes^c ub') \wedge (lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus \neq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus))$ then
                if $\otimes^c = \otimes$ then $\mathcal{B}' \leftarrow (LB', UB, val_0 \otimes lb_\otimes, val_0 \otimes ub_\otimes, lb_\oplus, ub_\oplus)$
                else $\mathcal{B}' \leftarrow (LB', UB, lb_\otimes, ub_\otimes, lb_\oplus \oplus lb_\otimes \otimes val_0, ub_\oplus \oplus ub_\otimes \otimes val_0)$
                $(lb', ub') \leftarrow$ evalClusterMax$(c, A.(x, a), V - \{x\}, \Phi - \Phi_0, \mathcal{B}')$
            $ub \leftarrow \max(ub, val_0 \otimes^c ub')$
            $lb \leftarrow \max(lb, val_0 \otimes^c lb')$
            $LB' \leftarrow \max(LB', lb_\otimes \otimes lb \oplus lb_\oplus)$
        if $(d \neq \emptyset)$ then $ub \leftarrow \top$
        return $((lb, ub))$
end
```

Figure 8.6: Bounded evaluation of a max-cluster.

Function evalClusterMin (Figure 8.7) Function evalClusterMin$(c, A, V, \Phi, \mathcal{B})$ must return an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. Its pseudo-code is similar to *evalClusterMax*. The unique difference is that at each iteration of the while loop, (lb, ub) is an evaluation of $\min_{a \in dom(x) - d} val(c, A.(x, a), V - \{x\}, \Phi)$ bounded by $\mathcal{B}$ (hence the initialization $(lb, ub) \leftarrow (\top, \top)$). Moreover, instead of strengthening the global lower bound LB', *evalClusterMin* may strengthen the global upper bound UB' in order to find assignments with an ever worse value.

Function EvalClusterPlus (Figure 8.8) The evaluation of a cluster having $\oplus$ as an elimination operator is different from max or min clusters evaluations when $\oplus \notin \{\min, \max\}$. If the set of unassigned variables is empty, *evalClusterPlus*$(c, A, V, \Phi, \mathcal{B})$ must return an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$. Such an evaluation is provided by evalSons$(c, A, \Phi, \mathcal{B})$.

Otherwise, we choose a variable $x \in V$. For each value a in $dom(x)$, a lower bound $tablb[a]$ and an upper bound $tabub[a]$ on $val(c, A.(x, a), V - \{x\}, \Phi)$ are computed. This enables us to initialize a lower bound lb and an upper bound ub on $val(c, A, V, \Phi) = \oplus_{a \in dom(x)} val(c, A.(x, a), V - \{x\}, \Phi)$.

As long as a bounded evaluation of $val(c, A, V, \Phi)$ is not available, the while loop is processed, i.e. a value a not yet considered in $dom(x)$ is chosen. A more precise evaluation of

```
evalClusterMin($c, A, V, \Phi, (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$)
begin
    if ($V = \emptyset$) then return (evalSons($c, A, \Phi, (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$))
    else
        Choose $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\, ,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $(lb, ub) \leftarrow (\top, \top)$
        $UB' \leftarrow UB$
        while $((d \neq \emptyset) \wedge (LB \prec UB') \wedge (ub \neq \bot))$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val_0 \leftarrow {\otimes^c}_{\varphi \in \Phi_0}\, \varphi(A.(x,a))$
            $(lb', ub') \leftarrow$ bound$(c, A.(x,a), V - \{x\}, \Phi - \Phi_0, val_0)$
            if $((LB \prec (ub_\otimes \otimes (val_0 \otimes^c ub')) \oplus ub_\oplus) \wedge (lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus \prec UB') \wedge (val_0 \otimes^c lb' \neq$
            $val_0 \otimes^c ub') \wedge (lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus \neq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus))$ then
                if $\otimes^c = \otimes$ then $\mathcal{B}' \leftarrow (LB, UB', val_0 \otimes lb_\otimes, val_0 \otimes ub_\otimes, lb_\oplus, ub_\oplus)$
                else $\mathcal{B}' \leftarrow (LB, UB', lb_\otimes, ub_\otimes, lb_\oplus \oplus lb_\otimes \otimes val_0, ub_\oplus \oplus ub_\otimes \otimes val_0)$
                $(lb', ub') \leftarrow$ evalClusterMin$(c, A.(x,a), V - \{x\}, \Phi - \Phi_0, \mathcal{B}')$
            $ub \leftarrow \min(ub, val_0 \otimes^c ub')$
            $lb \leftarrow \min(lb, val_0 \otimes^c lb')$
            $UB' \leftarrow \min(UB', ub_\otimes \otimes ub \oplus ub_\oplus)$
        if $(d \neq \emptyset)$ then $lb \leftarrow \bot$
        return $((lb, ub))$
end
```

Figure 8.7: Bounded evaluation of a min-cluster.

$val(c, A.(x,a), V - \{x\}, \Phi)$ is computed using an updated complex bound $\mathcal{B}'$. The computation of this new bound uses $lb_{\neg a}$ and $ub_{\neg a}$, which are lower and upper bounds respectively over $\oplus_{a' \in dom(x) - \{a\}}\, val(c, A.(x,a'), V - \{x\}, \Phi)$. Once a bounded evaluation of $val(c, A.(x,a), V - \{x\}, \Phi)$ is available, lb and ub are updated, as well as variable res. It can be shown that when the conditions of the while loop hold, res always equals $\oplus_{a \in dom(x) - d}\, val(c, A.(x,a), V - \{x\}, \Phi)$.

If the conditions of the while loop are not satisfied, then this exactly means that (lb, ub) is an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$, hence (lb, ub) is returned.

Function evalSons (Figure 8.9) This function must return an evaluation of $val(c, A, \emptyset, \Phi) = ({\otimes^c}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c ({\otimes^c}_{s \in Sons(c)}\, val(s)(A))$ bounded by $\mathcal{B} = (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$.

It does not record the exact value of $val(s)(A^{\downarrow s \cap c})$ for each son cluster $s \in Sons(c)$ using the caching structure of algorithm **RecTS-mcdag**, since because of pruning, backtrack can occur before the exact value of $val(s)(A^{\downarrow s \cap c})$ is known. The caching structure instead records a lower bound denoted $LB(s, A^{\downarrow s \cap c})$ and an upper bound denoted $UB(s, A^{\downarrow s \cap c})$ on $val(s)(A^{\downarrow s \cap c})$. These bounds are initialized with $\bot$ and $\top$ respectively, and they always satisfy $LB(s, A^{\downarrow s \cap c}) \preceq val(s)(A^{\downarrow s \cap c}) \preceq UB(s, A^{\downarrow s \cap c})$. If $LB(s, A^{\downarrow s \cap c}) = UB(s, A^{\downarrow s \cap c})$, then $val(s)(A^{\downarrow s \cap c})$ is known. The data structures used to record $LB(s, A^{\downarrow s \cap c})$ and $UB(s, A^{\downarrow s \cap c})$ can be sparse, since for example Binary Decision Diagrams [1, 21] or hash tables can be used instead of large tables in which many recorded values equal $\bot$ or $\top$. Moreover, it is possible to forget some bounds when the memory size available becomes too small.

Function *evalSons* works as follows. If $Sons(c) = \emptyset$, then $val(c, A, \emptyset, \Phi) = {\otimes^c}_{\varphi \in \Phi}\, \varphi(A)$ and it

```
evalClusterPlus($c, A, V, \Phi, (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$)
begin
    if ($V = \emptyset$) then return (evalSons($c, A, \Phi, (LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$))
    else
        Choose $x \in V$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\,,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        foreach $a \in d$ do $(tablb[a], tabub[a]) \leftarrow$ bound$(c, A.(x,a), V - \{x\}, \Phi, 1_E)$
        $d_0 \leftarrow \{a \in dom(x), tablb[a] = tabub[a]\}$
        $res \leftarrow \oplus_{a \in d_0} tablb[a]$
        $d \leftarrow dom(x) - d_0$
        $(lb, ub) \leftarrow (res \oplus (\oplus_{a \in d} tablb[a]), res \oplus (\oplus_{a \in d} tabub[a]))$
        while $((LB \prec ub_\otimes \otimes ub \oplus ub_\oplus) \wedge (lb_\otimes \otimes lb \oplus lb_\oplus \prec UB) \wedge (lb \neq ub) \wedge (lb_\otimes \otimes lb \oplus lb_\oplus \neq$
        $ub_\otimes \otimes ub \oplus ub_\oplus))$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val_0 \leftarrow \otimes_{\varphi \in \Phi_0} \varphi(A.(x,a))$
            $(lb_{\neg_a}, ub_{\neg_a}) \leftarrow (res \oplus (\oplus_{a' \in d} tablb[a']), res \oplus (\oplus_{a' \in d} tabub[a']))$
            $\mathcal{B}' \leftarrow (LB, UB, lb_\otimes \otimes val_0, ub_\otimes \otimes val_0, lb_\oplus \oplus (lb_\otimes \otimes lb_{\neg a}), ub_\oplus \oplus (ub_\otimes \otimes ub_{\neg a}))$
            $(lb_a, ub_a) \leftarrow$ evalClusterPlus$(c, A.(x,a), V - \{x\}, \Phi - \Phi_0, \mathcal{B}')$
            $(lb, ub) \leftarrow (lb_{\neg a} \oplus (val_0 \otimes lb_a), ub_{\neg a} \oplus (val_0 \otimes ub_a))$
            $res \leftarrow res \oplus (val_0 \otimes lb_a)$
        return $((lb, ub))$
end
```

Figure 8.8: Bounded evaluation of a $\oplus$ cluster.

is straightforward that the pair returned is $(lb, ub) = (\otimes^c{}_{\varphi\in\Phi}\,\varphi(A), \otimes^c{}_{\varphi\in\Phi}\,\varphi(A))$.

Otherwise, a son cluster s is considered and a bounded evaluation of $val(s)(A)$ is sought. We first compute lower and upper bounds $lb_{\neg s}$ and $ub_{\neg s}$ on $(\otimes^c{}_{\varphi\in\Phi}\,\varphi(A))\otimes^c(\otimes^c{}_{s'\in Sons(c)-\{s\}}\,val(s')(A))$, i.e. on the part of the problem which does not depend on s. We use $lb_{\neg s}$ and $ub_{\neg s}$ as parameters to compute a complex bound to be imposed on the function in charge of providing a bounded evaluation (lb_s, ub_s) of $val(s)(A) = val(s, A, V(s) - V(c), \Phi(s))$. Once (lb_s, ub_s) is available, the recorded lower and upper bound on $val(s)(A)$ are updated. More precisely, as both $val(s)(A) \succeq lb_s$ and $val(s)(A) \succeq LB(s, A^{\downarrow s\cap c})$, we can infer that $val(s)(A) \succeq \max(lb_s, LB(s, A^{\downarrow s\cap c}))$. Similarly, as both $val(s)(A) \preceq ub_s$ and $val(s)(A) \preceq UB(s, A^{\downarrow s\cap c})$, we can infer that $val(s)(A) \preceq \min(ub_s, UB(s, A^{\downarrow s\cap c}))$. This explains the updating of $LB(s, A^{\downarrow s\cap c})$ and $UB(s, A^{\downarrow s\cap c})$. A local variable res is updated too, and it can be shown that if the conditions of the while loop are satisfied, then we have $res = (\otimes^c{}_{\varphi\in\Phi}\,\varphi(A)) \otimes^c (\otimes^c{}_{s\in Sons(c)-S}\,val(s)(A))$. Lower and upper bounds lb and ub on $val(c, A, \emptyset, \Phi)$ are also updated, using $lb_{\neg s}$, $ub_{\neg s}$, $LB(s, A^{\downarrow s\cap c})$, and $UB(s, A^{\downarrow s\cap c})$.

When the conditions of the while loop are not satisfied, this exactly means that (lb, ub) is an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$, hence (lb, ub) is returned.

Soundness and completeness of algorithm BTD-mcdag

Lemma 8.12. *If function* bound *is sound and complete and if* evalSons *is sound and complete for clusters of depth h, then* evalClusterMax *is sound and complete for clusters of depth h.*

Lemma 8.13. *If function* bound *is sound and complete and if* evalSons *is sound and complete for clusters of depth h, then* evalClusterMin *is sound and complete for clusters of depth h.*

```
evalSons$(c, A, \Phi, (LB, UB, lb_{\otimes}, ub_{\otimes}, lb_{\oplus}, ub_{\oplus}))$
begin
    $S_0 \leftarrow \{s \in Sons(c), LB(s, A^{\downarrow s}) = UB(s, A^{\downarrow s})\}$
    $res \leftarrow (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c (\otimes^c{}_{s \in S_0}\, LB(s, A^{\downarrow s}))$
    $S \leftarrow Sons(c) - S_0$
    $(lb, ub) \leftarrow (res \otimes^c (\otimes^c{}_{s \in S}\, LB(s, A^{\downarrow s})), res \otimes^c (\otimes^c{}_{s \in S}\, UB(s, A^{\downarrow s})))$
    while $((LB \prec ub_{\otimes} \otimes ub \oplus ub_{\oplus}) \wedge (lb_{\otimes} \otimes lb \oplus lb_{\oplus} \prec UB) \wedge (lb \neq ub) \wedge (lb_{\otimes} \otimes lb \oplus lb_{\oplus} \neq ub_{\otimes} \otimes ub \oplus ub_{\oplus}))$
    do
        Choose $s \in S$
        $S \leftarrow S - \{s\}$
        $(lb_{\neg s}, ub_{\neg s}) \leftarrow (res \otimes^c \left(\otimes^c{}_{s' \in S}\, LB(s', A^{\downarrow s'})\right), res \otimes^c \left(\otimes^c{}_{s' \in S}\, UB(s', A^{\downarrow s'})\right))$
        if $\otimes^c = \otimes$ then $\mathcal{B}' \leftarrow (LB, UB, lb_{\neg s} \otimes lb_{\otimes}, ub_{\neg s} \otimes ub_{\otimes}, lb_{\oplus}, ub_{\oplus})$
        else $\mathcal{B}' \leftarrow (LB, UB, lb_{\otimes}, ub_{\otimes}, lb_{\oplus}, lb_{\oplus} \oplus lb_{\otimes} \otimes lb_{\neg s}, ub_{\oplus} \oplus ub_{\otimes} \otimes ub_{\neg s})$
        $(lb_s, ub_s) \leftarrow$ EvalCluster-$\oplus^s(s, A, V(s) - V(c), \Phi(s), \mathcal{B}')$
        $LB(s, A^{\downarrow s}) \leftarrow \max(lb_s, LB(s, A^{\downarrow s}))$
        $UB(s, A^{\downarrow s}) \leftarrow \min(ub_s, UB(s, A^{\downarrow s}))$
        $(lb, ub) \leftarrow (lb_{\neg s} \otimes^c LB(s, A^{\downarrow s}), ub_{\neg s} \otimes^c UB(s, A^{\downarrow s}))$
        $res \leftarrow res \otimes^c LB(s, A^{\downarrow s})$
    return $((lb, ub))$
end
```

Figure 8.9: Bounded evaluation of the sons of a cluster.

Lemma 8.14. *If function* bound *is sound and complete and if* evalSons *is sound and complete for clusters of depth h, then* evalClusterPlus *is sound and complete for clusters of depth h.*

Lemma 8.15. *Function* evalSons *is sound and complete for clusters of maximal depth.*

Lemma 8.16. *If function* bound *is sound and complete, if* evalClusterMin, evalClusterMax, *and* evalClusterPlus *are sound and complete for clusters of depth h, then* evalSons *is sound and complete for clusters of depth $h - 1$.*

Lemma 8.17. *If function* bound *is sound and complete, then* evalSons *is sound and complete.*

Theorem 8.18. *If function* bound *is sound and complete, then algorithm **BTD-mcdag** is sound and complete, i.e. it returns $Ans(Q)$.*

Proposition 8.19. *Let M be a MCDAG associated with a query $Q = (Sov, (V, G, P, \emptyset, U))$. Then, the time complexity of algorithm **BTD-mcdag** is $O(m \cdot \mu \cdot d^h)$, where h is the MCDAG-height, μ is the maximum number of parents of a node in the MCDAG ($\mu = 1$ if the MCDAG is a MCTree), and $m = |P \cup U|$ in the semiring case and $m = (1 + |P|)(1 + |U|)$ in the semigroup case. The space complexity is $O(N \cdot s \cdot d^s)$, where N is the number of clusters in the MCDAG and s is the size of the largest separator.*

The theoretical time complexity of algorithm **BTD-mcdag** is worse than the theoretical time complexity of algorithm **RecTS-mcdag**, and both algorithms have the same space complexity.[3] However, this does not mean that **RecTS-mcdag** always outperforms **BTD-mcdag**, since these elements are just theoretical complexities. In practice, a tree search using bounds, despite its worse theoretical time complexity, often outperforms variable elimination algorithms.

[3] When the set E of the MCS $(E, \oplus, \otimes)$ is known to be finite, it is possible to show that the time complexity becomes $O(m \cdot |E| \cdot d^{w+1})$, where w is the width of the MCDAG.

Note that algorithm **BTD-mcdag** generalizes both the alpha-beta algorithm used in game theory and the BTD algorithm used to solve CSPs and VCSPs. It can be used to solve stochastic SAT problems, stochastic CSPs, QBFs, QCSPs, influence diagrams, factored MDPs, possibilistic influence diagrams, MAP (Maximum A Posteriori hypothesis) problems, or probabilistic planning problems. It suffices to replace $\otimes^c$ and $\oplus^c$ by their instantiations in each of these formalisms. This shows the interest of defining generic algorithms.

8.5 Using division and difference operators

Complex bounds make it possible to define a structured tree search using bounds and caching. Nevertheless, using complex bounds is not free, because for each test involving the global lower bound LB or the global upper bound UB, one $\otimes$ operation and one $\oplus$ operation are performed, in addition to the comparison operation testing whether LB or UB is satisfied.

This section shows how additional algebraic assumptions enable us to use simple bounds (LB, UB) and simple comparisons such as $(LB \prec ub) \wedge (lb \prec UB)$, instead of complex bounds $(LB, UB, lb_{\otimes}, ub_{\otimes}, lb_{\oplus}, ub_{\oplus})$ and complex comparisons such as $(LB \prec ub_{\otimes} \otimes ub \oplus ub_{\oplus}) \wedge (lb_{\otimes} \otimes lb \oplus lb_{\oplus} \prec UB)$.

Basically, the additional algebraic assumptions allowing us to use simple bounds are related to the existence of inverse operations for $\otimes$ and $\oplus$. They are similar to the assumptions used in VCSPs that are said to be *fair* [25] or in semiring-based CSPs enhanced with a division operation [9]. They can be enounced as follows:

- Additional axiom on $\oplus$, denoted "$Ax^{\ominus}$":

  For all $x, y \in E$ such that $x \preceq y$, the set $\{z \in E \,|\, y = z \oplus x\}$ has a maximum element denoted $y \ominus x$. In other words, we assume that there is a maximal difference of y and x.

- Additional axiom on $\otimes$, denoted "$Ax^{\oslash}$", with two disjoint versions:

  - $Ax_1^{\oslash}$: either $1_E = \top$ and for all $x, y \in E$ such that $x \preceq y$, the set $\{z \in E \,|\, x = z \otimes y\}$ has a maximum element denoted $x \oslash y$ (i.e. there is a maximal division of x and y).
  - $Ax_2^{\oslash}$: or $1_E \neq \top$ and $\top^+ = \top$ [4] and for all $x, y \in E$ such that $y \notin \{0_E, \top\}$, there exists a unique $z \in E$, denoted $x \oslash y$, such that $x = y \otimes z$

We also adopt the conventions $\bot^- \ominus x = \bot^-$, $\top^+ \ominus x = \top^+$, $\bot^- \oslash x = \bot^-$, and $\top^+ \oslash x = \top^+$ for all $x \in E$.

Axioms $Ax^{\ominus}$ and $Ax^{\oslash}$ are satisfied in several usual cases. For example, the extra assumption on $\oplus$ holds with $(E, \preceq, \oplus) = ([0, +\infty], \leq, +)$, $(E, \preceq, \oplus) = ([0, +\infty], \geq, \min)$, or $(E, \preceq, \oplus) = ([0, 1], \leq, \max)$. The extra assumption on $\otimes$ is satisfied with $(E, \preceq, \otimes) = ([0, +\infty], \geq, +)$ or $(E, \preceq, \otimes) = ([0, 1], \leq, \min)$ for the first case ($1_E = \top$), and with $(E, \preceq, \otimes) = ([0, +\infty], \leq, \times)$ for the second case ($1_E \neq \top$).

As shown in Table 8.1, as soon as $Ax^{\ominus}$ and $Ax^{\oslash}$ hold, it is possible to avoid using complex bounds. This table shows that given a quantity val to be computed, requirements such as $\alpha \oplus val \prec$

[4] This means that $\top$ is not initially is the MCS and is added as described in Section 8.4.1 page 147.

UB, $\alpha \oplus val \succ LB$, $\alpha \otimes val \prec UB$, or $\alpha \otimes val \succ LB$ can be transformed into requirements for which it suffices to compare val with an updated lower bound LB' or with an updated upper bound UB'.

For example, row 1 imposes the requirement $\alpha \oplus val \prec UB$. If $UB \preceq \alpha$, then, as $\alpha = \alpha \oplus 0_E \preceq \alpha \oplus val$, we can infer that the requirement is never satisfied. Hence, we can impose an equivalent unsatisfiable requirement on val, written as $val \prec \perp^-$. As for row 2, if $\alpha \prec UB$, then $UB \ominus \alpha$ is defined and $\alpha \oplus val \prec UB$ implies that $val \prec UB \ominus \alpha$ (because if $val \succeq UB \ominus \alpha$, then $val \oplus \alpha \succeq UB$ by monotonicity of $\oplus$). In general, the inverse implication $(val \prec UB \ominus \alpha) \rightarrow (\alpha \oplus val \prec UB)$ does not hold, which means that the complex requirement $\alpha \oplus val \prec UB$ can yield more pruning than the simpler requirement $val \prec UB \ominus \alpha$. However, as soon as $\oplus$ is strictly monotonic, the equivalence $(val \prec UB \ominus \alpha) \leftrightarrow (\alpha \oplus val \prec UB)$ holds whenever $\alpha \prec UB$.

In both cases, rows 1 and 2 enable us to replace $\alpha \oplus val \prec UB$ by $val \prec UB'$, where UB' is a new simple upper bound.

For row 6, the requirement $\alpha \otimes val \prec UB$ is imposed and $\alpha \prec UB$ holds. Then, as $1_E = \top$ with $Ax_1^{\oslash}$, we can infer that $\alpha \otimes val \preceq \alpha \otimes 1_E \prec UB$, hence the requirement is always satisfied. This is equivalent to impose $val \prec UB'$ with $UB' = \top^+$ as a new upper bound.

Case	Complex requirement	Condition	Simpler requirement
$Ax^{\ominus}$	$\alpha \oplus val \prec UB$	$UB \preceq \alpha$	$val \prec \perp^-$
		$\alpha \prec UB$	$val \prec UB \ominus \alpha$
	$LB \prec \alpha \oplus val$	$LB \prec \alpha$	$val \succ \perp^-$
		$\alpha \preceq LB$	$val \succ LB \ominus \alpha$
$Ax_1^{\oslash}$	$\alpha \otimes val \prec UB$	$UB \preceq \alpha$	$val \prec UB \oslash \alpha$
		$\alpha \prec UB$	$val \prec \top^+$
	$LB \prec \alpha \otimes val$	$LB \prec \alpha$	$val \succ LB \oslash \alpha$
		$\alpha \preceq LB$	$val \succ \top^+$
$Ax_2^{\oslash}$	$\alpha \otimes val \prec UB$	$\alpha \notin \{0_E, \top\}$	$val \prec UB \oslash \alpha$
		$\alpha = 0_E$	$val \prec \top^+$
	$LB \prec \alpha \otimes val$	$\alpha \notin \{0_E, \top\}$	$val \succ LB \oslash \alpha$
		$(\alpha = 0_E) \wedge (LB \neq \perp^-)$	$val \succ \top^+$
		$(\alpha = 0_E) \wedge (LB = \perp^-)$	$val \succ \perp^-$
		$\alpha = \top$	$val \succ \perp^-$

Table 8.1: From complex bounds to simple bounds using difference and division operations, for $(\alpha, val) \in E^2$ and $LB \prec UB$. For $Ax_2^{\oslash}$, the requirement $\alpha \otimes val \prec UB$ together with the case $\alpha = \top$ is not considered because it will never be used in practice (roughly speaking, when $Ax_2^{\oslash}$ holds and when $\alpha \otimes val \prec UB$ will be required required, α will always be a lower bound on some quantity in $E - \{\top\}$, hence it does not equal $\top$).

In fact, in order to be simpler and to be always able to write $(\alpha \oplus val \prec UB) \rightarrow (val \prec UB \ominus \alpha)$ and $(LB \prec \alpha \oplus val) \rightarrow (val \succ LB \ominus \alpha)$, it suffices to extend the definition of $\ominus$ by $y \ominus x = \perp^-$ whenever $y \prec x$. In order to write $(\alpha \otimes val \prec UB) \rightarrow (val \prec UB \oslash \alpha)$ and $(LB \prec \alpha \otimes val) \rightarrow (val \succ LB \oslash \alpha)$ when $Ax_1^{\oslash}$ holds, it suffices to extend the definition of $\oslash$ by $x \oslash y = \top^+$ whenever $y \prec x$. In order to write $(\alpha \otimes val \prec UB) \rightarrow (val \prec UB \oslash \alpha)$ and $(LB \prec \alpha \otimes val) \rightarrow (val \succ LB \oslash \alpha)$ when $Ax_2^{\oslash}$ holds, it suffices to extend the definition of $\oslash$ by $x \oslash 0_E = \top^+$ if $x \neq \perp^-$, $\perp^- \oslash 0_E = \perp^-$, and $x \oslash \top = \perp^-$ if $x \prec \top$.

Thanks to $Ax^{\ominus}$ and $Ax^{\oslash}$, new algorithms using simple bounds can be specified. Simple bounds enable us to define a much simpler notion of bounded evaluation.

Definition 8.20. *An evaluation of* $val(c, A, V, \Phi)$ *bounded by a simple bound* (LB, UB)*, is a couple*

$(lb, ub) \in E^2$ such that $lb \preceq val(c, A, V, \Phi) \preceq ub$ and $(lb = ub) \vee (UB \preceq lb) \vee (ub \preceq LB)$.

In other words, an evaluation of $val(c, A, V, \Phi)$ bounded by (LB, UB) is simply a pair of lower and upper bounds on $val(c, A, V, \Phi)$ which either provides the exact value of $val(c, A, V, \Phi)$, or proves that one of the bounds is not satisfied.

The new functions **evalClusterMax**(c, A, V, Φ, LB, UB), **evalClusterMin**(c, A, V, Φ, LB, UB), and **evalClusterPlus**(c, A, V, Φ, LB, UB) are required to compute an evaluation of $val(c, A, V, \Phi)$ bounded by (LB, UB), and the new function $evalSons(c, A, \Phi, LB, UB)$ is required to compute an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by (LB, UB). The new main function is called **BTD-answerQ**().

Function BTD-answerQ() (Figure 8.10) This main function simply computes an evaluation of the root cluster using $(\bot^-, \top^+)$ as inviolable simple bounds. Therefore, it gets lower and upper bounds (lb, ub) such that $lb = ub = val(r, \emptyset, V(r), \Phi(r))$, i.e. $lb = ub = Ans(Q)$.

```
BTD-answerQ()
begin
    $r \leftarrow root(MCDAG)$
    $(lb, ub) \leftarrow$ evalCluster- $\oplus^r$ $(r, \emptyset, V(r), \Phi(r), \bot^-, \top^+)$
    return $(lb)$
end
```

Figure 8.10: Main function: **BTD-answerQ**.

Other functions (Figures 8.11 to 8.14) The other functions are similar to the previous ones. The differences are the stopping conditions determining whether a bounded evaluation is available, and the use of division and difference operations to compute new simple bounds. The instructions associated with the handling of simple bounds are underlined. Given a cluster c, we denote by $\oslash^c$ the operation $\oslash$ if $\otimes^c = \otimes$, and $\ominus$ if $\otimes^c = \oplus$.

For example, for *evalClusterMax*, the new bounds (LB'', UB'') computed when further computations are needed simply mean that a complex requirement such as $LB' \prec val_0 \otimes^c val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0) \prec UB$ is transformed into the simpler requirement $LB' \oslash^c val_0 \prec val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0) \prec UB \oslash^c val_0$. The modification of *evalClusterMin* is similar.

As for *evalClusterPlus*, $lb_{\neg a}$ and $ub_{\neg a}$ are respectively lower and upper bounds on the quantity $\oplus_{a' \in dom(x) - \{a\}} val(c, A.(x, a'), V - \{x\}, \Phi)$. We can impose on $val(c, A.(x, a), V - \{x\}, \Phi)$ the requirements $LB \prec val(c, A.(x, a), V - \{x\}, \Phi) \oplus ub_{\neg a}$ and $val(c, A.(x, a), V - \{x\}, \Phi) \oplus lb_{\neg a} \prec UB$. Using $val(c, A.(x, a), V - \{x\}, \Phi) = val_0 \otimes val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0)$, these requirements can be transformed into the weaker but simpler requirements $val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0) \succ (LB \ominus ub_{\neg a}) \oslash val_0$ and $val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0) \prec (UB \ominus lb_{\neg a}) \oslash val_0$. This explains the new simple bounds used.

Concerning *evalSons*, the complex requirements $LB \prec ub_{\neg s} \otimes^c val(s, A, V(s) - V(c), \Phi(s))$ and $lb_{\neg s} \otimes^c val(s, A, V(s) - V(c), \Phi(s)) \prec UB$ can be transformed into the simpler requirements $val(s, A, V(s) - V(c), \Phi(s)) \succ LB \oslash^c ub_{\neg s}$ and $val(s, A, V(s) - V(c), \Phi(s)) \prec UB \oslash^c lb_{\neg s}$, hence the new bounds used.

```
evalClusterMax$(c, A, V, \Phi, LB, UB)$
begin
    if $(V = \emptyset)$ then  return (evalSons$(c, A, \Phi, LB, UB)$)
    else
        Choose $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\,,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $(lb, ub) \leftarrow (\bot, \bot)$
        $LB' \leftarrow LB$
        while $((d \neq \emptyset) \wedge (LB' \prec UB) \wedge (lb \neq \top))$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val_0 \leftarrow {\otimes^c}_{\varphi \in \Phi_0}\, \varphi(A.(x, a))$
            $(lb', ub') \leftarrow \mathrm{bound}(c, A.(x, a), V - \{x\}, \Phi - \Phi_0, val_0)$
            if $((LB' \prec val_0 \otimes^c ub') \wedge (val_0 \otimes^c lb' \prec UB) \wedge (val_0 \otimes^c lb' \neq val_0 \otimes^c ub'))$ then
                $\underline{\mathbf{(LB'', UB'') \leftarrow (LB' \oslash^c val_0, UB \oslash^c val_0)}}$
                $(lb', ub') \leftarrow$ evalClusterMax$(c, A.(x, a), V - \{x\}, \Phi - \Phi_0, LB'', UB'')$
            $ub \leftarrow \max(ub, val_0 \otimes^c ub')$
            $lb \leftarrow \max(lb, val_0 \otimes^c lb')$
            $LB' \leftarrow \max(LB', lb)$
        if $(d \neq \emptyset)$ then $ub \leftarrow \top$
        return $((lb, ub))$
end
```

Figure 8.11: Bounded evaluation of a max-cluster using simple bounds.

```
evalClusterMin$(c, A, V, \Phi, LB, UB)$
begin
    if $(V = \emptyset)$ then  return (evalSons$(c, A, \Phi, LB, UB)$)
    else
        Choose $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\,,\, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $(lb, ub) \leftarrow (\top, \top)$
        $UB' \leftarrow UB$
        while $((d \neq \emptyset) \wedge (LB \prec UB') \wedge (ub \neq \bot))$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val_0 \leftarrow {\otimes^c}_{\varphi \in \Phi_0}\, \varphi(A.(x, a))$
            $(lb', ub') \leftarrow \mathrm{bound}(c, A.(x, a), V - \{x\}, \Phi - \Phi_0, val_0)$
            if $((LB \prec (val_0 \otimes^c ub') \wedge (val_0 \otimes^c lb' \prec UB') \wedge (val_0 \otimes^c lb' \neq val_0 \otimes^c ub'))$ then
                $\underline{\mathbf{(LB'', UB'') \leftarrow (LB \oslash^c val_0, UB' \oslash^c val_0)}}$
                $(lb', ub') \leftarrow$ evalClusterMin$(c, A.(x, a), V - \{x\}, \Phi - \Phi_0, LB'', UB'')$
            $ub \leftarrow \min(ub, val_0 \otimes^c ub')$
            $lb \leftarrow \min(lb, val_0 \otimes^c lb')$
            $UB' \leftarrow \min(UB', ub)$
        if $(d \neq \emptyset)$ then $lb \leftarrow \bot$
        return $((lb, ub))$
end
```

Figure 8.12: Bounded evaluation of a min-cluster using simple bounds.

```
evalClusterPlus($c, A, V, \Phi, LB, UB$)
begin
    if ($V = \emptyset$) then return (evalSons($c, A, \Phi, LB, UB$))
    else
        Choose $x \in V$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\,,\ sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        foreach $a \in d$ do  $(tablb[a], tabub[a]) \leftarrow \text{bound}(c, A.(x,a), V - \{x\}, \Phi, 1_E)$
        $d_0 \leftarrow \{a \in dom(x), tablb[a] = tabub[a]\}$
        $res \leftarrow \oplus_{a \in d_0} tablb[a]$
        $d \leftarrow dom(x) - d_0$
        $(lb, ub) \leftarrow (res \oplus (\oplus_{a \in d} tablb[a]), res \oplus (\oplus_{a \in d} tabub[a]))$
        while $((LB \prec ub) \wedge (lb \prec UB) \wedge (lb \neq ub))$ do
            Choose $a \in d$
            $d \leftarrow d - \{a\}$
            $val_0 \leftarrow \otimes_{\varphi \in \Phi_0} \varphi(A.(x,a))$
            $(lb_{\neg_a}, ub_{\neg_a}) \leftarrow (res \oplus (\oplus_{a' \in d} tablb[a'])\,, res \oplus (\oplus_{a' \in d} tabub[a']))$
            $\underline{(\mathbf{LB'}, \mathbf{UB'}) \leftarrow ((\mathbf{LB} \ominus \mathbf{ub_{\neg a}}) \oslash \mathbf{val_0}, (\mathbf{UB} \ominus \mathbf{lb_{\neg a}}) \oslash \mathbf{val_0})}$
            $(lb_a, ub_a) \leftarrow$ evalClusterPlus($c, A.(x,a), V - \{x\}, \Phi - \Phi_0, LB', UB'$)
            $(lb, ub) \leftarrow (lb_{\neg a} \oplus (val_0 \otimes lb_a), ub_{\neg a} \oplus (val_0 \otimes ub_a))$
            $res \leftarrow res \oplus (val_0 \otimes lb_a)$
        return $((lb, ub))$
end
```

Figure 8.13: Bounded evaluation of a $\oplus$ cluster using simple bounds.

```
evalSons($c, A, \Phi, LB, UB$)
begin
    $S_0 \leftarrow \{s \in Sons(c), LB(s, A^{\downarrow s}) = UB(s, A^{\downarrow s})\}$
    $res \leftarrow (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c (\otimes^c{}_{s \in S_0} LB(s, A^{\downarrow s}))$
    $S \leftarrow Sons(c) - S_0$
    $(lb, ub) \leftarrow (res \otimes^c (\otimes^c{}_{s \in S} LB(s, A^{\downarrow s})), res \otimes^c (\otimes^c{}_{s \in S} UB(s, A^{\downarrow s})))$
    while $((LB \prec ub) \wedge (lb \prec UB) \wedge (lb \neq ub))$ do
        Choose $s \in S$
        $S \leftarrow S - \{s\}$
        $(lb_{\neg s}, ub_{\neg s}) \leftarrow \left(res \otimes^c \left(\otimes^c{}_{s' \in S} LB(s', A^{\downarrow s'})\right), res \otimes^c \left(\otimes^c{}_{s' \in S} UB(s', A^{\downarrow s'})\right)\right)$
        $\underline{(\mathbf{LB'}, \mathbf{UB'}) \leftarrow (\mathbf{LB} \oslash^{\mathbf{c}} \mathbf{ub_{\neg s}}, \mathbf{UB} \oslash^{\mathbf{c}} \mathbf{lb_{\neg s}})}$
        $(lb_s, ub_s) \leftarrow$ EvalCluster-$\oplus^s(s, A, V(s) - V(c), \Phi(s), LB', UB')$
        $LB(s, A^{\downarrow s}) \leftarrow \max(lb_s, LB(s, A^{\downarrow s}))$
        $UB(s, A^{\downarrow s}) \leftarrow \min(ub_s, UB(s, A^{\downarrow s}))$
        $(lb, ub) \leftarrow (lb_{\neg s} \otimes^c LB(s, A^{\downarrow s}), ub_{\neg s} \otimes^c UB(s, A^{\downarrow s}))$
        $res \leftarrow res \otimes^c LB(s, A^{\downarrow s})$
    return $((lb, ub))$
end
```

Figure 8.14: Bounded evaluation of the sons of a cluster using simple bounds.

Theorem 8.21. *If function* bound *is sound and complete, then* ***BTD-answerQ*** *is sound and complete too, i.e. it returns $Ans(Q)$.*

8.6 Computing bounds by inference mechanisms

This section defines a catalog of techniques which can be used to provide lower and upper bounds on the answer $Ans(Q)$ to a query Q. They are also interesting to compute bounds on a quantity such as $val(c, A, V, \Phi)$, because $val(c, A, V, \Phi)$ can actually be seen as a query too, if we recompose the scoped functions and the eliminations which are in the descendants of cluster c in the MCDAG.

The techniques presented enable the *bound* function to return bounds better than poor $(\perp, \top)$.

Computing bounds by propagation A first possible mechanism is to propagate information, in the spirit of constraint propagation [84]. The algebraic structure offers two specific elements, $\perp = 0_E$ and $\top$. The first is an annihilator for $\otimes$, and the second is an annihilator for $\oplus$. Hence, given a query $Q = (Sov, (V, G, P, \emptyset, U))$, it is possible to propagate information as follows:

- In the semiring case, where $Ans(Q) = Sov(\otimes_{\varphi \in P \cup U} \varphi)$, we can enforce any level of consistency (forward checking, arc consistency, k-consistency...) in order to propagate degree 0_E. This can prune the search space by removing values in the current variables domains. As with usual constraint propagation techniques, once the domain of a variable is empty, the algorithm can backtrack because the result of the currently explored subtree then necessarily equals 0_E.

- In the semigroup case, where $Ans(Q) = Sov((\otimes_{\varphi \in P} \varphi) \otimes (\oplus_{\varphi \in U} \varphi))$, we can proceed as follows. First, as in the semiring case, degree 0_E can be propagated amongst plausibility functions in order to remove values in the variables domains. Second, value $\top$ can be propagated among utility functions. Backtrack can then occur if we prove that the current assignment A satisfies either $\otimes_{\varphi \in P} \varphi(A) = 0_E$, or $(\otimes_{\varphi \in P} \varphi(A) \neq 0_E) \wedge (\oplus_{\varphi \in U} \varphi(A) = \top)$.

As it has been done for QCSPs, it should be possible to adapt Quantified Arc Consistency (QAC [15]) to the MCS case. This could lead to better bounds. Works concerning this kind of generalized arc-consistency are not presented in this thesis for maturity reasons. The main difficulty resides in the presence of $\oplus$ eliminations when $\oplus \notin \{\min, \max\}$.

Works on soft local consistencies [84, 11, 25, 79] for semiring CSPs or VCSPs could also be considered in order to prune the search space by propagating all elements of E (and not only 0_E or $\top$).

Computing bounds by switching quantifiers

Proposition 8.22. *Let φ be a scoped function taking values in a totally $\preceq$-ordered set E. let S and S' be two disjoint sets of finite domain variables. Then,*

$$\begin{aligned}
\max_S \min_{S'} \varphi &\preceq \min_{S'} \max_S \varphi \\
\max_S \bigoplus_{S'} \varphi &\preceq \bigoplus_{S'} \max_S \varphi \\
\bigoplus_S \min_{S'} \varphi &\preceq \min_{S'} \bigoplus_S \varphi
\end{aligned}$$

By relaxing the constraints on the elimination order, this technique can help to reduce the tree-width (or induced-width) of the considered computation.

For example, in order to get bounds on $val = \min_{x_1,\ldots,x_n} \max_y (\wedge_{i\in[1,n]} \varphi_{x_i,y})$, one can write $val \succeq lb$, with $lb = \max_y \min_{x_1,\ldots,x_n} (\wedge_{i\in[1,n]} \varphi_{x_i,y})$. The tree-width associated with the computation of val is n, because y is necessarily eliminated first, whereas the tree-width associated with the computation of lb is 1. Therefore, even if the quantity to be bounded is hard to compute, computing a bound by switching some eliminations can be easy.

Computing bounds by relaxing quantifiers We continue the catalog of possible techniques to compute bounds, with a technique consisting in replacing some quantifiers in the sequence of eliminations to be performed. More precisely, this technique uses Proposition 8.23:

Proposition 8.23. *Let $(V, G, P, \emptyset, U)$ be a PFU network, and let $c \in \mathcal{C}_E(G)$ be an environment component in G. Then, for every scoped function φ,*

$$\min_c \varphi \preceq \oplus_c((\otimes_{P_i \in Fact(c)} P_i) \otimes \varphi) \preceq \max_c \varphi$$

Together with the quantifier switching mechanism, this technique can give lower and upper bounds on the answer to a query:

- In order to get a lower bound on $Ans(Q)$ for a query Q, it suffices to replace all max- and $\oplus$-eliminations by min-eliminations and to remove all plausibility functions from the PFU network. For example, let us consider a query $Q = (Sov, (V, G, P, \emptyset, U))$ where $Sov = \min_{x_1} \max_{x_2,x_3} \oplus_{x_4} \max_{x_5} \oplus_{x_6} \min_{x_7}$ and where $P = \{P_{x_4\,|\,x_2}, P_{x_6\,|\,x_4,x_3}\}$ contains two local plausibility functions. We can write:

$$\begin{aligned} Ans(Q) &\succeq \min_{x_1} \min_{x_2,x_3} \bigoplus_{x_4} \min_{x_5} \bigoplus_{x_6} \min_{x_7} (P_{x_4\,|\,x_2} \otimes P_{x_6\,|\,x_4,x_3} \otimes (\oplus_{U_i\in U} U_i)) \\ &\succeq \min_{x_1} \min_{x_2,x_3} \bigoplus_{x_4} (P_{x_4\,|\,x_2} \otimes \min_{x_5} \bigoplus_{x_6} (P_{x_6\,|\,x_4,x_3} \otimes \min_{x_7} (\oplus_{U_i\in U} U_i))) \\ &\succeq \min_{x_1} \min_{x_2,x_3} \min_{x_4} \min_{x_5} \min_{x_6} \min_{x_7} (\oplus_{U_i\in U} U_i) \end{aligned}$$

- Similarly, in order to get an upper bound on $Ans(Q)$ for a query Q, it suffices to replace all min- and $\oplus$-eliminations by max-eliminations and to remove all plausibility functions from the PFU network. With the same query as above, we can write:

$$\begin{aligned} Ans(Q) &\preceq \max_{x_1} \max_{x_2,x_3} \bigoplus_{x_4} \max_{x_5} \bigoplus_{x_6} \max_{x_7} (P_{x_4\,|\,x_2} \otimes P_{x_6\,|\,x_4,x_3} \otimes (\oplus_{U_i\in U} U_i)) \\ &\preceq \max_{x_1} \max_{x_2,x_3} \bigoplus_{x_4} (P_{x_4\,|\,x_2} \otimes \max_{x_5} \bigoplus_{x_6} (P_{x_6\,|\,x_4,x_3} \otimes \max_{x_7} (\oplus_{U_i\in U} U_i))) \\ &\preceq \max_{x_1} \max_{x_2,x_3} \max_{x_4} \max_{x_5} \max_{x_6} \max_{x_7} (\oplus_{U_i\in U} U_i) \end{aligned}$$

The key point which can make such a mechanism efficient in practice is that in order to obtain the lower and upper bounds given above, we must compute a mono-operator sequence of eliminations. As there are no constraints on the elimination order, the computation of lb and ub can be easy even if the initial problem is hard, all the more so, since the plausibility functions are removed. For example, let us consider the influence diagram associated with the computation of

$\max_{x_1} \sum_{x_2,x_3} \max_{x_4} \sum_{x_5}(P_{x_2} \cdot P_{x_5\,|\,x_1,x_2} \cdot P_{x_3\,|\,x_5} \cdot (U_{x_1,x_4} + U_{x_3} + U_{x_4,x_5}))$. Using MCDAGs, this computation has a tree-width of 4. In order to compute $lb = \min_{x_1,x_2,x_3,x_4,x_5}(U_{x_1,x_4} + U_{x_3} + U_{x_4,x_5})$ and $ub = \max_{x_1,x_2,x_3,x_4,x_5}(U_{x_1,x_4} + U_{x_3} + U_{x_4,x_5})$, the tree-width is only 1.

For QBFs, the mechanism consisting in replacing min by max in order to get bounds has already been used and proved to be efficient in practice [120]. The corresponding proposal defines a solver which, at some steps during search, replaces $\forall$ quantifiers by $\exists$ quantifiers in order to get an upper bound on a QBF, this upper bound being computed by using a SAT solver. The authors are not aware of the use of such methods for stochastic CSPs or influence diagrams.

Mini-buckets [40] Mini-buckets are generic tools which can be used to approximate and bound a computation to be performed, e.g. in constraint optimization or Bayesian networks. They were shown to be very successful in practice on various problems.

The idea is to force an inference algorithm such as a VE algorithm to consider only a limited number of variables simultaneously, which ensures a bounded computation time at the price of giving only a bound on the exact value which should be computed. The number of variables which can be considered simultaneously is a parameter of the mini-bucket technique. It defines a trade-off between the quality of the bound obtained and its computation time.

For example, in order to compute $\max_x(\varphi_{x,y} + \varphi_{x,z} + \varphi_{x,t})$, a VE algorithm needs to consider four variables simultaneously. The mini-buckets technique can consist in writing $\max_x(\varphi_{x,y} + \varphi_{x,z} + \varphi_{x,t}) \preceq (\max_x \varphi_{x,y}) + (\max_x \varphi_{x,z}) + (\max_x \varphi_{x,t})$. The right part of this inequality is an upper bound computable by considering only 2 variables simultaneously. Similarly, in order to obtain an upper bound on a quantity such as $\sum_x(\varphi_{x,y} \cdot \varphi_{x,z} \cdot \varphi_{x,t})$ by considering at most two variables simultaneously, it suffices to compute $(\sum_x \varphi_{x,y}) \cdot (\sum_x \varphi_{x,z}) \cdot (\sum_x \varphi_{x,t})$.

Transposed to the MCS algebraic structure, the mini-bucket technique can be described as in Proposition 8.24.

Proposition 8.24. *Let $(E, \oplus, \otimes)$ be a totally ordered MCS having 0_E as a minimum element. Let φ_1, φ_2 be two scoped functions onto E. Then, for every set of variables S,*

$$\begin{array}{rcl}
\max_S(\varphi_1 \otimes \varphi_2) & \preceq & (\max_S \varphi_1) \otimes (\max_S \varphi_2) \\
\max_S(\varphi_1 \oplus \varphi_2) & \preceq & (\max_S \varphi_1) \oplus (\max_S \varphi_2) \\
\min_S(\varphi_1 \otimes \varphi_2) & \succeq & (\min_S \varphi_1) \otimes (\min_S \varphi_2) \\
\min_S(\varphi_1 \oplus \varphi_2) & \succeq & (\min_S \varphi_1) \oplus (\min_S \varphi_2) \\
\oplus_S(\varphi_1 \otimes \varphi_2) & \preceq & (\oplus_S \varphi_1) \otimes (\oplus_S \varphi_2)
\end{array}$$

Obtaining bounds by simplifying the algebraic structure It should also be possible to reuse approaches modifying the agebraic structure at stake in order to obtain bounds on a given quantity. As in [8], which introduces the notion of abstraction of semiring CSPs, the basic idea can be to work on a transformed version of an initial problem (obtained via an algebraic transformation preserving some properties and easier to solve), and then to bring some information back to the initial problem.

A similar idea is developed in [30] for bounding the optimum value of a valued CSP. More precisely, given an initial VCSP P expressed on a valuation structure S, it is possible first to simplify it to obtain a VCSP P' expressed on a simpler valuation structure S', second to solve

P', and third to induce lower and upper bounds by bringing back some information to the initial VCSP P. Such an approach is shown to be efficient both on random and real problems.

8.7 Integrating feasibilities

Again, feasibilities have been left apart in this chapter. But again, integrating them in the previous scheme is possible. Two main mechanisms can be used:

- The first mechanism is easy and works as follows: when variable x is assigned with value a, we can directly test whether $A.(x,a)$ is feasible, for example by using a SAT solver in parallel with the BTD algorithm. If $A.(x,a)$ is not feasible, then another value of x is considered. Otherwise, the search progresses normally. This first technique can be implemented by adding a single line in the algorithm in order to test whether the current assignment is feasible.
- Second, one can maintain lower and upper bounds on the feasibility of the current assignment. If the upper bound on this feasibility equals f, then the current assignment is not feasible and the algorithm backtracks. If the lower bound on the feasibility degree equals t, then it is sure that the assignment is feasible. Compared to the first method, this second technique is harder to implement since it modifies the structure of the algorithm itself, but it has the advantage of not solving a potentially hard satisfiability problem at each step of the search.

8.8 Summary and perspectives

This chapter has shown how a generic structured tree search using bounds can be defined to compute the value of a MCDAG. The key points are the handling of multiple elimination operators and the handling of bounds. Complexity results have also been provided. They can vary depending on the amount of space used by the algorithm and are characterized by the MCDAG-width, the MCDAG-height, or the maximum separator size.

In another direction, approximate algorithms using sampling and local search could also have been considered: sampling when eliminations with $+$ ($+$, and not $\oplus$) are performed [87, 114], local search when eliminations with min or max are performed [88]. This is one of the perspectives in the quest for other generic approaches.

From a practical point of view, the algorithms developed in this chapter present several elements whose influence remains to be studied:

- Heuristic for the choice of the variable to be assigned in the current cluster, heuristic for the choice of a value for a variable, heuristic for the choice of a son cluster to be considered...
- Computation of bounds: some clues have been provided concerning the computation of bounds, but there is still a lot to do in order to determine good settings (e.g. concerning the degree of local consistency).

Many elements are well-known concerning these parameters in each of the formalisms subsumed by the PFU framework. In order to get a better knowledge concerning their "generic" influence, and also in order to test the practical efficiency of the algorithms defined, experiments are needed. That is why we have developed a generic solver to answer generic PFU queries.

Chapter 9

A generic solver for answering PFU queries

This chapter briefly introduces the solver developed to answer generic PFU queries. It first focuses on problems description formats and then briefly presents the generic implemented solver. The main goal of this chapter is to convince the reader that the PFU framework is not just an abstraction.

9.1 Description of problems

Before introducing the PFU solver, we describe how instances of PFU networks and PFU queries are represented. An XML format has been defined, and some existing formats representing problems in formalisms subsumed by the PFU framework can also be used. The XML format dissociates the description of PFU networks and the description of queries, because several queries can be asked on a given PFU network. The algebraic structure is not described as an XML file (more details in Section 9.2).

9.1.1 XML representation of PFU networks

In order to specify an XML representation of PFU networks, it is important to note that we dissociate functions from scoped functions. This distinction is done for conciseness reasons because a function φ can be used by several scoped functions (S, φ). Similarly, we explicitly define domains as elements dissociated from variables, because a given domain can be used by several variables. In fact, the XML representation used is close to the representation used in [16], which defines an XML representation format for CSPs.

A PFU network is represented, as shown Figure 9.1, by an element called *pfunet*, which contains several elements defining the tuple (V, G, P, F, U):

- The elements called *name*, *author*, *date* contain respectively a name for the PFU network, the name(s) of the author(s), and a date.

```
<pfunet>

<name>Business Dinner Problem</name>
<author>Cedric Pralet</author>
<date>02-02-2006</date>


<domains nbDom="3">
      <domain id="mcval" type="string" description="extension" values="meat fish"/>
      <domain id="wval"  type="string" description="extension" values="white red"/>
      <domain id="bool"  type="bool"   description="extension" values="true false"/>
</domains>


<plausfunctions nbPlausFunctions="4">
      <plausfunction id="pfunc1" domains="bool bool" default_degree="0" nbInst="2">
            <instance assignment="true false" degree="0.6"/>
            <instance assignment="false true" degree="0.4"/>
      </plausfunction>
      <plausfunction id="pfunc2" domains="bool bool" default_degree="1" nbInst="1">
            <instance assignment="false true" degree="0"/>
      </plausfunction>
      ...
</plausfunctions>


<feasfunctions nbFeasFunctions="1">
      <feasfunction id="ffunc1" domains="mcval wval" default_degree="true" nbInst="1">
            <instance assignment="fish red" degree="false"/>
      </feasfunction>
</feasfunctions>


<utilfunctions nbUtilFunctions="3">
      <utilfunction id="ufunc1" domains="bool bool" default_degree="0" nbInst="1">
            <instance assignment="true false" degree="bottom"/>
      </utilfunction>
      <utilfunction id="ufunc2" domains="bool" default_degree="0" nbInst="1">
            <instance assignment="true" degree="10"/>
      </utilfunction>
      ...
</utilfunctions>


<variables nbVar="6">
      <variable id="mc"  nature="decision"    domain="mcval" description="main course choice"/>
      <variable id="w"   nature="decision"    domain="wval"  description="wine choice"/>
      <variable id="bpJ" nature="environment" domain="bool"  description="John's beginning pres."/>
      <variable id="bpM" nature="environment" domain="bool"  description="Mary's beginning pres."/>
      ...
</variables>


<plausibilities nbPlaus="5">
      <plausibility id="p1" scope="bpJ bpM"    function="pfunc1"/>
      <plausibility id="p2" scope="bpJ epJ"    function="pfunc2"/>
      <plausibility id="p3" scope="bpJ w epJ"  function="pfunc3"/>
      ...
</plausibilities>


<feasibilities nbFeas="1">
      <feasibility id="f1" scope="mc w" function="ffunc1"/>
</feasibilities>


<utilities nbUtil="3">
      <utility id="u1" scope="bpJ epJ" function="ufunc1"/>
      <utility id="u2" scope="epJ"     function="ufunc2"/>
      ...
</utilities>


<components nbComp="4">
      <component id="c1" nature="decision"    vars="mc w"    scoped_f="f1"    parents=""/>
      <component id="c2" nature="environment" vars="bpJ bpM" scoped_f="p1"    parents=""/>
      <component id="c3" nature="environment" vars="epJ"     scoped_f="p2 p3" parents="c1 c2"/>
      ...
</components>

</pfunet>
```

Figure 9.1: XML representation of the PFU network of the dinner problem.

- The element called *domains* has an attribute called *nbDom* and contains elements called *domain*. Attribute *nbDom* equals the number of occurrences of elements *domain*.

  Each element *domain* is empty and contains attributes *id* (identifier for the domain), *type* (the types allowed are string, int, float, double, and bool), *description* (says if the domain is represented in extension as a set of values, or in intension as an interval plus a constant step between two values in the interval), and *values* (which specifies either the set of values, or the bounds of the interval and the step).

- The element called *plausfunctions* has an attribute called *nbPlausFunctions* and contains elements called *plausfunction*. *nbPlausFunctions* is the number of occurrence of elements *plausfunction*.

  Each element *plausfunction* defines an unscoped plausibility function φ. It has a set of attributes called *id* (identifier), *domains* (list of domain identifiers), *default_degree* (default degree given by the function), and *nbInst* (number of assignments A of the domains such that $\varphi(A) \neq$ default_degree). Each element *plausfunction* also contains elements called *instance*. *nbInst* is the number of occurrences of elements *instance*. Each element *instance* is empty and admits attributes called *assignment* and *degree*, which correspond respectively to an assignment A of the domains and to $\varphi(A)$.

  The elements called *feasfunctions* and *utilfunctions* satisfy similar specifications. A possible improvement of the XML format could be to allow for functions defined by formulas.

- The element called *variables* admits an attribute *nbVar* and contains elements *variable*. Attribute *nbVar* is the number of occurrences of elements *variable*.

  Each element *variable* is empty and has a set of attributes called *id* (variable name), *nature* (decision or environment variable), *domain* (domain of values of the variable), and *description* (what the variable represents).

  Therefore, the element called *variables* defines the set V of variables of a PFU network.

- The element called *plausibilities* has an attribute *nbPlaus* and contains occurrences of elements *plausibility*. Attribute *nbPlaus* is the number of occurrences of elements *plausibility*.

  Each element *plausibility* is empty and has a set of attributes called *id* (identifier), *scope* (scope of the plausibility function, defined by a list of variables), and *function* (an identifier which must correspond to a *plausfunction* element).

  In other words, the elements *plausibilities* define the set P of plausibility functions of the PFU network. The description of elements *feasibilities* and *utilities* is similar, and they define the sets F and U of feasibility and utility functions of a PFU network respectively.

- The element called *components* admits an attribute called *nbComp* and contains elements called *component*. Attribute *nbComp* is the number of occurrences of elements *component*.

  Each element *component* is empty and has a set of attributes called *id* (name of a component c), *nature* (decision of environment component), *vars* (variables involved in the component), *scoped_f* (scoped functions in $Fact(c)$), and *parents* (list of parent components of c in the DAG of the PFU network).

Therefore, the element called *components* enables us to model the DAG G of a PFU network.

More formally, the DTD (Document Type Definition) which defines the syntax of the XML documents describing PFU networks is given in Appendix D.

9.1.2 XML representation of queries

An XML representation of queries is also available. Basically, queries are defined by a PFU network and by a sequence of operator-variable(s) pairs. We also explicitly specifiy the decision variables for which optimal decision rules are sought.

Figure 9.2 gives an example of an XML representation of a query on the dinner problem. The associated query corresponds to a situation where Peter chooses both the wine and the main course after knowing who is present at the beginning, and optimal decision rules for the main course choice mc and for the wine choice w are sought. A query is defined by an element called *query*, which contains several elements:

- The elements called *name*, *author*, and *date* contain a name for the query, the name(s) of the author(s), and a date.
- The element called *pfunet* is an empty element which has an attribute called *file*. This attribute indicates the XML file describing the PFU network used by the query.
- The element called *sov* has an attribute *nbStages* and contains elements called *op_var_pair*. Attribute *nbStages* is the number of occurrence of elements *op_var_pair*.

  Each element *op_var_pair* is an empty element which has three attributes: *op* (elimination operator equal to "MIN", "MAX", or "PLUS"), *vars* (list of variables to eliminate), and *record* (list of variables for which a decision rule must be recorded).

More formally, the DTD associated with XML files describing queries is given in Appendix D.

```
<query>

<name>If Peter knows who is present at the beginning</name>
<author>Cedric Pralet</author>
<date>19-09-2005</date>

<pfunet file="pfunet.xml"/>

<sov nbStages="3">
      <op_vars_pair op="PLUS" vars="bpJ bpM"/>
      <op_vars_pair op="MAX"  vars="mc w" record="mc w"/>
      <op_vars_pair op="PLUS" vars="epJ epM"/>
</sov>

</query>
```

Figure 9.2: XML description of a query.

9.1.3 Reading others formats

The solver is also able to read existing description formats defined in formalisms subsumed by the PFU framework: the QDIMACS format, which enables QBFs to be defined, the ERGO format, which enables Bayesian networks to be defined, and a format ".net" used to specify influence diagrams. Also, problems can be defined via an XML format called ".dpfu". Roughly speaking, this format enables us to specify kinds of “dynamic” PFU networks, in which we describe first a standard PFU network associated with step 0, and second transition functions (as in MDPs) specifying plausibility and feasibility functions associated with the variables in the PFU network at step $t+1$, depending on the variables in PFU network at step t.

9.2 Solver description

The solver is written in C++. It is generic because it can work with different instances of elimination and combination operators, and with different data types (bool, int, float, double). We briefly describe its main features, by explaining how PFU networks and queries are represented, how the algebraic structure is defined, how problems are read, and which algorithms are currently implemented. The global structure of the classes involved in our generic solver is given in Figure 9.3 (this figure assumes that the reader is familiar with the UML representation language).

PFU networks and queries The main classes which enable PFU networks and queries to be defined are:

- A class called *Domain*, which enables a domain of values to be represented. It has two specializations called *TypedDomainExt* (for a domain represented in extension) and *TypedDomainInt* (for a domain represented in intension as an interval and a constant step between two values in the interval).

- A class called *Variable*, which enables variables to be represented. A variable notably has an instance of class *Domain* in its attributes.

- A class called *Scope*: instances of this class are list of variables. This class offers some functions to manipulate scopes.

- A class called *Component*: an instance of this class corresponds to a component of the PFU network. A component has a scope which defines the variables involved in the component, a list of parent components, and a list of scoped functions associated with it.

- A class called *Function*: instances of this class correspond to functions (without a scope). This class has a list of domains as an attribute. The cartesian product of these domains represents the domain of definition of the function.

  Class *Function* has four specializations called *Clause*, *FunctionExt*, *FunctionTrie*, and *MultiFunction*. These specializations correspond to different representations of the function: instances of class *Clause* are functions represented as boolean clauses (this is useful to treat QBFs), instance of class *FunctionExt* are functions represented as a table of values, one for each element of the cartesian product of the domains, instance of class *FunctionTrie* are represented using a sparse data structure classically called a *trie*, and instance of class *MultiFunction* are represented as a set of functions (class *MultiFunction* is useful for example to represent the aggregation of all utility functions in a compact way).

- A class called *ScopedFunction*: instances of this class are scoped functions. In its attributes, this class has an instance of class *Function* (the function of the scoped function), an instance

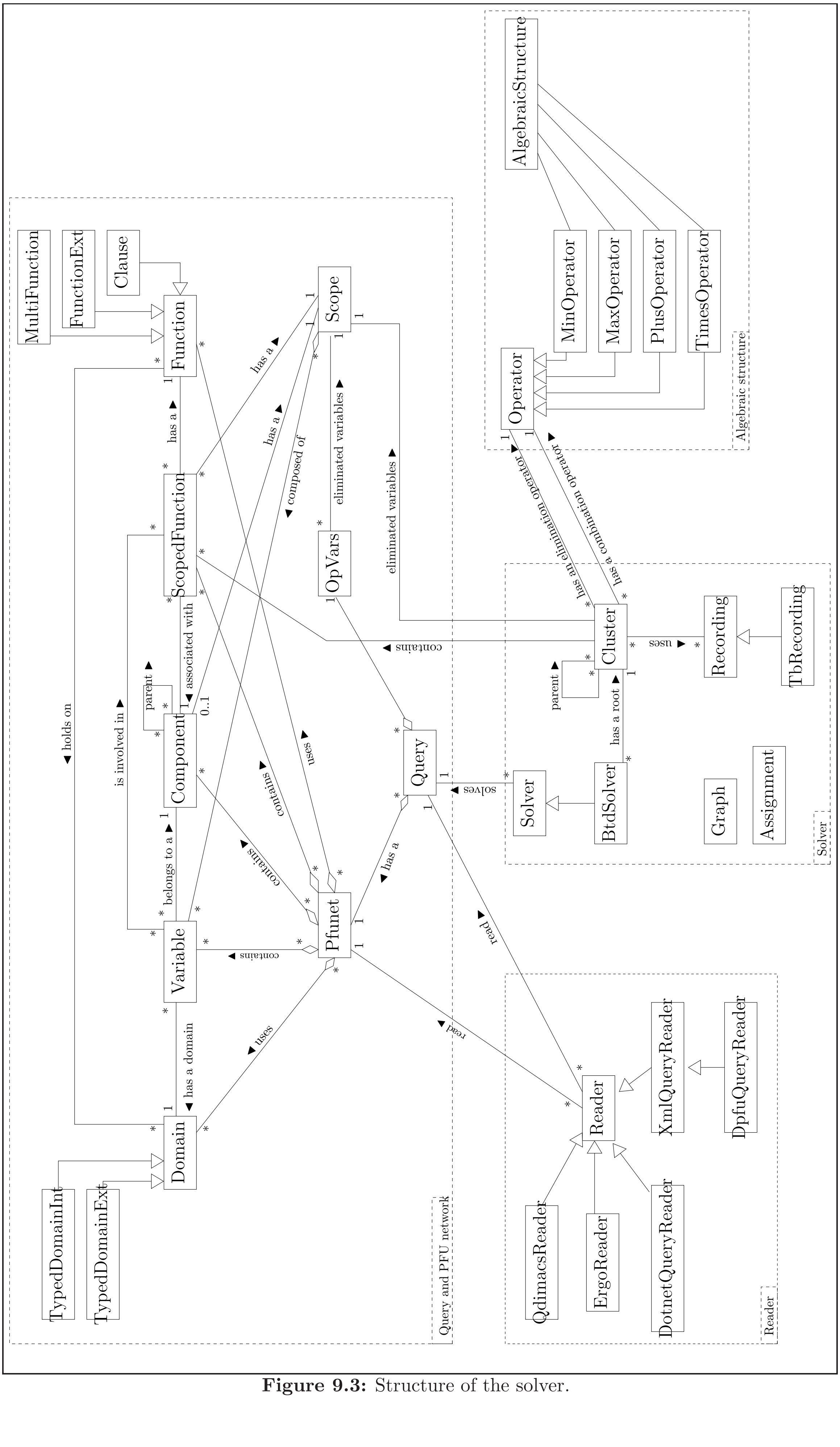


Figure 9.3: Structure of the solver.

of class *Scope* (the scope of the scoped function), and an instance of class *Component* (the component to which the scoped function is associated, if it is a plausibility or a feasibility function).

- A class called *Pfunet*: instances of this class represent PFU networks. In its attributes, this class has a list of domains, a list of variables, a list of components, three lists of scoped functions (one for each type of scoped function), and lists of functions used by the scoped functions.

- A class called *OpVars*, which defines operator-variables pairs. The variables of an operator-variables pair are represented by a scope.

- A class called *Query*, whose instances are queries on PFU networks. In its attributes, this class has an instance of class *Pfunet* and an instance of class *OpVars*.

Readers Several classes enable us to read PFU networks and queries. The corresponding classes are *Reader*, *QdimacsReader* (to read QBFs in the QDIMACS format), *ErgoReader* (to read Bayesian networks in the ERGO format), *DotnetQueryReader* (to read influence diagrams in the ".net" format), *XmlQueryReader* (to read queries specified in the XML format previously described), and *DpfuQueryReader* (in order to read queries expressed in the ".dpfu" format).

Algebraic structure An included file "globaldef.h" contains the type deg_t of the plausibility and utility degrees manipulated (we assume that plausibilities and utilities have the same type). The solver is currently able to deal with $\text{deg\_t} \in \{\text{bool}, \text{int}, \text{float}, \text{double}\}$.

The operators used can be defined in two ways:

- First, operators $\otimes_p$, $\otimes_u$, $\otimes_{pu}$, $\oplus_p$, and $\oplus_u$, as well as 0_p, 1_p, and 0_u, can be explicitly defined as parameterized macros. This enables a user to directly specify a new expected utility structure if needed.

- When the algebraic structure is a totally ordered MCS, we use another representation. A class *AlgebraicStructure* defines algebraic structures. In its attributes, this class has two instances of class *Operator*. These instances define the operators $\oplus$ and $\otimes$ of the MCS. The exact operators used in the executable are defined by a macro called ALGEBRAICSTRUCTURE, involved in the preprocessor conditional compilation directives.

  Macro ALGEBRAICSTRUCTURE refers to an element in a hard-coded list of algebraic structures: (1) probabilistic expected additive utility, (2) probabilistic expected satisfaction, (3) possibilistic optimistic expected utility, (4) possibilistic pessimistic expected utility, (5) expected utility structure with kappa-rankings and only positive utility degrees. Note that when deg_t = bool, algebraic structures (3) and (4) allow boolean optimistic and pessimistic expected conjunctive utilities to be used.

  Class *Operator* has several specializations: *MinOperator*, *MaxOperator*, *PlusOperator*, and *TimesOperator*. Each of these specializations must implement a function *merge*(T a,T b), which combines *a* and *b* with the operator associated with the class (T is a generic type). In fact, *MinOperator*, *MaxOperator*, *PlusOperator*, and *TimesOperator* perform this merging using min, max, +, and × respectively. Extending this list is possible by hard-coding other operators.

Solver The solver itself involves several elements. First, a class *Solver* is defined. It has an instance of class *Query* in its attributes, which corresponds to the query to be solved by the solver. Class *Solver* is able to answer queries in the very general case, i.e. with an algebraic structure which is only an expected utility structure and with feasibilities, thanks to a method implementing algorithm **TreeSearch-answerQ** given in Chapter 6 page 90.

This class is specialized by class *BtdSolver*, which contains methods capable of computing the answer to a query when there are no feasibilities. The algorithms currently available are **TS-mcdag**, **RecTS-mcdag**, **BTD-mcdag**, and **BTD-answerQ** (see previous chapter). Hence, all the algorithms based on tree search are implemented. These methods are valid when the algebraic structure is a totally ordered MCS. Class *BtdSolver* uses a class *Cluster* which enables MCDAG clusters to be represented.

Class *Cluster* has in its attributes a parent cluster, an operator to use as the cluster elimination operator $\oplus^c$, an operator to use as the cluster combination operator $\otimes^c$, and instances of class *Recording*, which enable to record lower and upper bounds over the separator of the cluster with its parents.

Class *Recording* has two specializations, which correspond to a recording performed via tables and via tries respectively. The second data structure is interesting because it is sparse.

A class called *Graph* is also used to perform operations on graphs, like computing cluster-tree decompositions. Cluster-tree decompositions are computed using the so-called *min-fill* heuristic.

The solver can use heuristics for choice points:

- Choice of the next variable to assign inside a given cluster: lexicographic or choice of a variable having a minimal current domain (ties broken lexicographically).

- Choice of a value to assign to a given variable: lexicographic or choice of a value having a minimal or a maximal utility degree obtained by inference.

- Choice of a son cluster to explore: lexicographic or choice of a son of minimum height in the MCDAG.

The unique form of constraint propagation implemented (for the *bound* function) is the propagation of 0_E using backward checking, forward checking, or arc consistency, and a form of valued forward checking [123] restricted to the currently explored cluster.

9.3 Perspectives

Some experiments have been performed, but much more are needed in order to obtain practical results on several points:

- Compare the algorithms previously defined in terms of pratical complexity:
  - quantify the gains in using MCDAGs exploiting the query structure,
  - compare VE algorithms with structured tree search methods,

  - compare structured tree search algorithms for various parameter settings: caching or not, complex or simple bounds, heuristics for variable, value, or cluster choices, and techniques used to compute bounds (soft local consistency, quantifier switching...).

- Compare the implemented methods with existing algorithms designed in a specific formalism.

- Evaluate the complexity given by an expected utility (EU) structure. More precisely, EU structures vary from structures which are more qualitative (such as possibilistic EU) to structures which are more quantitative (such as probabilistic EU) or structures which mix qualitative and quantitative approaches (such as EU based on κ-rankings). We could compare the pratical time and space complexities of these plausibility-utility models, in order to analyze the gains and costs in using a more or less qualitative or quantitative approach.

Conclusion

Synthesis of the contributions

In the last decades, AI has witnessed the design and study of numerous formalisms for reasoning about decision making problems. In this thesis, we have built a generic flexible framework to model sequential decision making problems involving plausibilities, feasibilities, and utilities. This framework covers many existing approaches, including hard, valued, quantified, mixed, and stochastic CSPs, Bayesian networks, Markov random fields, finite horizon probabilistic or possibilistic MDPs, or influence diagrams, as well as unpublished formalisms. The result is an algebraic framework built upon decision-theoretic foundations: the PFU framework. The two facets of the PFU framework are explicit in Theorem 5.9, which states that the operational definition of the answer to a query is equivalent to the decision tree-based semantics. This is the result of a design that accounts both for expressivity and for computational aspects. Compared to related works [127, 32, 75], the PFU framework is the only algebraic framework which directly deals with different types of variables (decision and environment variables), different types of local functions (plausibilities, feasibilities, utilities), and different types of combination and elimination operators.

From an algorithmic point of view, generic algorithms based on tree search and variable elimination have been defined. Decomposability conditions enabling factorizations to be exploited have been identified and used in a generic unified variable elimination algorithm (potentially using so-called potentials). In another direction, a generic approach to query optimization has led to the definition of original architectures for answering queries, called *multi-operator cluster trees* and *multi-operator cluster DAGs*. These architectures have been built thanks to a two-step structuration process using rewriting rules and cluster-tree decomposition techniques, and they lead to an improved width. Based on these architectures, structured tree search algorithms have been designed, using more or less sophisticated mechanisms such as recording or bounds. The main difficulty has lain in handling the multi-operator nature of PFU queries, both in terms of elimination and combination. Obviously, some assumptions made by the PFU framework could be discussed. But it should be noted that the assumptions made have enabled various algorithmic approaches to be considered. Finally, a generic solver able to answer PFU queries has been developed.

All these contributions are summed up in Figure 9.4

From a more global point of view, the conclusions of this thesis can be stated as follows:

1. Building a generic framework encompassing many existing AI formalisms is possible, and the obtained framework is not intractable. It is just a generic form of *algebraic composite graphical model*.

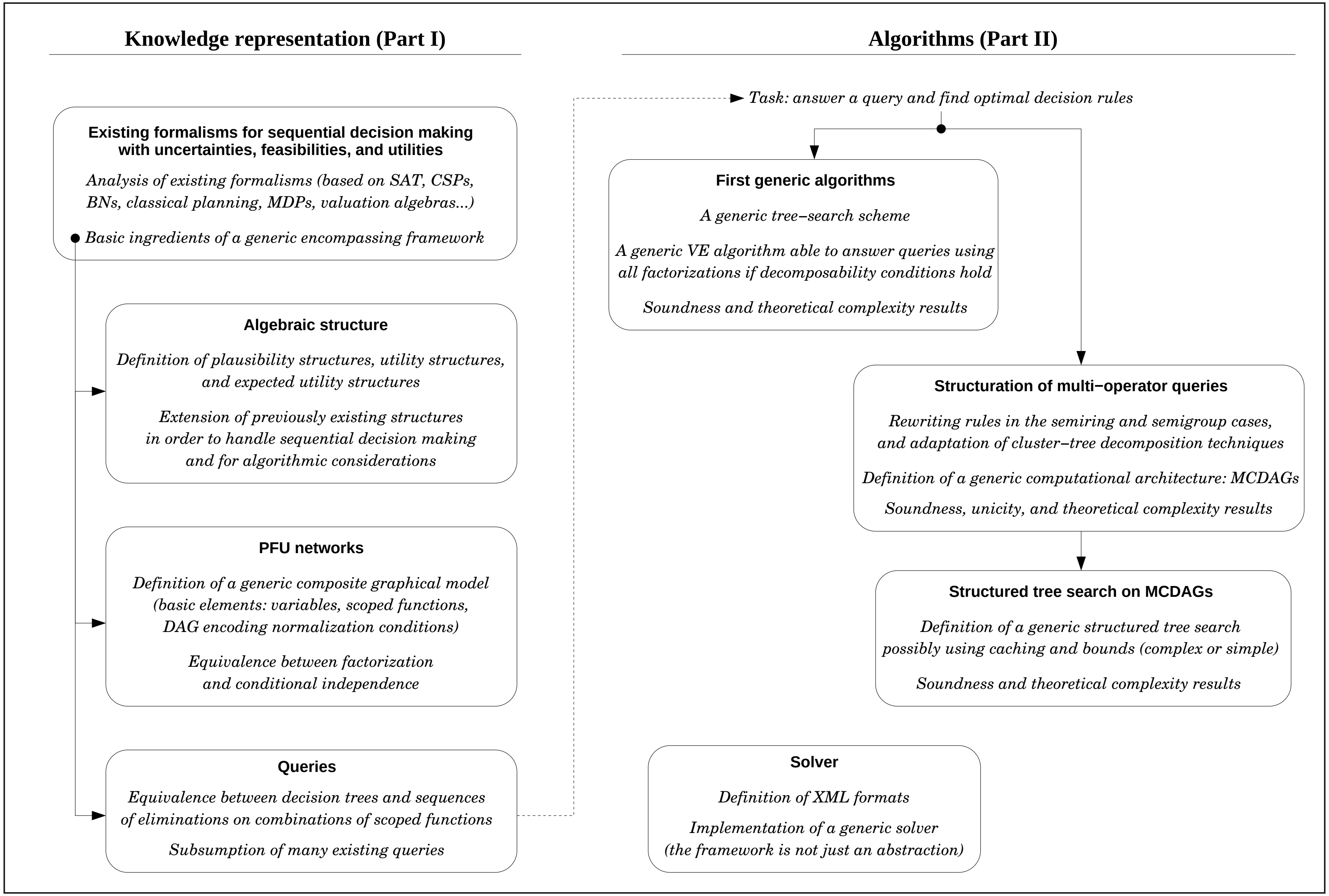

Knowledge representation (Part I)
Algorithms (Part II)
Existing formalisms for sequential decision making with uncertainties, feasibilities, and utilities
Analysis of existing formalisms (based on SAT, CSPs, BNs, classical planning, MDPs, valuation algebras...)
Basic ingredients of a generic encompassing framework
Algebraic structure
Definition of plausibility structures, utility structures, and expected utility structures
Extension of previously existing structures in order to handle sequential decision making and for algorithmic considerations
PFU networks
Definition of a generic composite graphical model (basic elements: variables, scoped functions, DAG encoding normalization conditions)
Equivalence between factorization and conditional independence
Queries
Equivalence between decision trees and sequences of eliminations on combinations of scoped functions
Subsumption of many existing queries
Task: answer a query and find optimal decision rules
First generic algorithms
A generic tree−search scheme
A generic VE algorithm able to answer queries using all factorizations if decomposability conditions hold
Soundness and theoretical complexity results
Structuration of multi−operator queries
Rewriting rules in the semiring and semigroup cases, and adaptation of cluster−tree decomposition techniques
Definition of a generic computational architecture: MCDAGs
Soundness, unicity, and theoretical complexity results
Structured tree search on MCDAGs
Definition of a generic structured tree search possibly using caching and bounds (complex or simple)
Soundness and theoretical complexity results
Solver
Definition of XML formats
Implementation of a generic solver (the framework is not just an abstraction)

Figure 9.4: Summary of contributions.

2. Generic unified algorithms can be defined in this framework, and, as in usual algebraic approaches, topological parameters such as width play an important role in the theoretical time and space complexities. In terms of width, an accurate analysis of the multi-operator queries considered can be helpful.

3. Answering multi-operator queries can be reduced to answering several mono-operator queries organized in a generic architecture called the MCDAG architecture. The latter can be systematically obtained, and once it is, existing methods for the mono-operator case are reusable. The main difficulty yielded by this architecture is the handling of bounds. The reason is that bounds must face the multi-operator nature of queries (both in terms of combination and elimination), because they are used globally in the whole architecture. In fact, MCDAGs can be used whatever the resolution method is (variable elimination, tree search, or local search), because they just express decompositions.

Perspectives

The perspectives of this work are multiple:

- As mentioned at the end of Chapter 9, performing experiments is one of the short term objective, in order to get a better knowledge concerning the algorithms developed.
- The structuration methods reason at the variables level. We could also try to exploit a finer structure, at the function values level, using approaches such as Binary Decision Diagrams (BDDs [1, 21]) or Negation Normal Forms (NNFs [28]).
- Also, we could study more precisely the results provided by the structuration methods for PFU queries and networks replicated from one step to another, as in factored Markov decision processes.
- A lot of work remains to be performed concerning bounds, in order to develop a kind of generalized quantified soft local consistency.
- At a higher level, two opposite attitudes can be adopted concerning the framework itself. These attitudes are not incompatible, and correspond respectively to a generalization and a specialization strategy:
  - We can continue the quest for genericity, in order to be more expressive. Also, we could define kinds of "multi-queries" allowing several queries to be asked simultaneously as is done in BNs to compute several marginal probability distributions simultaneously.
  - Or we can identify some basic problems and focus on them. More precisely, the MCDAG architecture shows that the elementary problems to be solved often consist of computing quantities such as $\sum_S(\prod_{\varphi\in\Phi}\varphi)$, $\max_S(\sum_{\varphi\in\Phi}\varphi)$, or $\max_S(\min_{\varphi\in\Phi}\varphi)$. These elementary problems correspond to the kind of computations performed in BNs [96], weighted CSPs [80], and fuzzy CSPs [42] respectively. In order to justify this specialization approach, we must exhibit morphisms between generic MCDAGs and MCDAGs using just the three elementary problems listed above [25]. Provided that this algebraic step is performed, one can see the MCDAG architecture as a melting pot of these three elementary problems, at the frontier between BNs and soft CSPs.

In the next years, the PFU framework will maybe enable other algorithmic ideas to be integrated in an efficient and flexible generic solver. This would be an opportunity to gather many efforts performed in different communities, and to benefit from the fertile links between algebra, graphical models, and combinatorial optimization.

List of Tables

List of Figures

Bibliography

[1] S.B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, 27(6), 1978.

[2] S.A. Arnborg. Efficient Algorithms for Combinatorial Problems on Graphs with Bounded Decomposability - A Survey. *BIT*, 25:2–23, 1985.

[3] F. Bacchus and A. Grove. Graphical Models for Preference and Utility. In *Proc. of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 3–10, Montréal, Canada, 1995.

[4] B.W. Ballard. The *-Minimax Search Procedure for Trees Containing Chance Nodes. *Artificial Intelligence*, 21(3):327–350, 1983.

[5] R.J. Bayardo and D.P. Miranker. On the Space-Time Trade-off in Solving Constraint Satisfaction Problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 558–562, Montréal, Canada, 1995.

[6] M. Benedetti. Quantifier Trees for QBF. In *Proc. of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, St. Andrews, Scotland, 2005.

[7] U. Bertelé and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

[8] S. Bistarelli, P. Codognet, and F. Rossi. Abstracting Soft Constraints: Framework, Properties, Examples. *Artificial Intelligence*, 139:175–211, 2002.

[9] S. Bistarelli and F. Gadducci. Enhancing Constraints Manipulation in Semiring-based Formalisms. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, Riva del Garda, Italy, 2006.

[10] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 624–630, Montréal, Canada, 1995.

[11] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Satisfaction and Optimization. *Journal of ACM*, 44(2):201–236, 1997.

[12] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-Based CSPs and Valued CSPs: Frameworks, Properties and Comparison. *Constraints*, 4(3):199–240, 1999.

[13] H. L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[14] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree. *Journal of Algorithms*, 18:238–255, 1995.

[15] L. Bordeaux and E. Monfroy. Beyond NP: Arc-consistency for Quantified Constraints. In *Proc. of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*, Ithaca, New York, USA, 2002.

[16] F. Boussemart, F. Hemery, and C. Lecoutre. Description and Representation of the Problems selected for the First International Constraint Satisfaction Solver Competition, 2005.

[17] C. Boutilier, R. Brafman, H. Hoos, and D. Poole. Reasoning With Conditional Ceteris Paribus Preference Statements. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, Stockholm, Sweden, 1999.

[18] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[19] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[20] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-Specific Independence in Bayesian Networks. In *Proc. of the 12th International Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 115–123, Portland, Oregon, USA, 1996.

[21] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[22] R. Chellappa and A. Jain. Markov Random Fields: Theory and Applications. Academic Press, 1993.

[23] F. Chu and J. Halpern. Great Expectations. Part I: On the Customizability of Generalized Expected Utility. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

[24] F. Chu and J. Halpern. Great Expectations. Part II: Generalized Expected Utility as a Universal Decision Rule. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 291–296, Acapulco, Mexico, 2003.

[25] M. Cooper and T. Schiex. Arc Consistency for Soft Constraints. *Artificial Intelligence*, 154(1-2):199–227, 2004.

[26] A. Darwiche. Recursive Conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[27] A. Darwiche and M.L. Ginsberg. A Symbolic Generalization of Probability Theory. In *Proc. of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 622–627, San Jose, CA, USA, 1992.

[28] A. Darwiche and P. Marquis. A Knowledge Compilation Map. *Artificial Intelligence*, 17:229–264, 2002.

[29] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of the 21st National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA, USA, 2006.

[30] S. de Givry, G. Verfaillie, and T. Schiex. Bounding the Optimum of Constraint Optimization Problems. In *Proc. of the 3rd International Conference on Principles and Practice of Constraint Programming (CP-97)*, Schloss Hagenberg, Austria, 1997.

[31] T. Dean and K. Kanazawa. A Model for Reasoning about Persistence and Causation. *Computational Intelligence*, 5(3):142–150, 1989.

[32] R. Dechter. Bucket Elimination: a Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[33] R. Dechter. A New Perspective on Algorithms for Optimizing Policies under Uncertainty. In *Proc. of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, pages 72–81, Breckenridge, CO, USA, 2000.

[34] R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[35] R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125(1-2):93–118, 2001.

[36] R. Dechter and D. Larkin. Hybrid Processing of Beliefs and Constraints. In *Proc. of the 17th International Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 112–119, Seattle, WA, USA, 2001.

[37] R. Dechter and R. Mateescu. Mixtures of Deterministic-Probabilistic Networks and their AND/OR Search Space. In *Proc. of the 20th International Conference on Uncertainty in Artificial Intelligence (UAI-04)*, Banff, Canada, 2004.

[38] R. Dechter and R. Mateescu. AND/OR Search Spaces for Graphical Models. *To appear in Artificial Intelligence Journal*, 2006.

[39] R. Dechter, I. Meiry, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.

[40] R. Dechter and I. Rish. Mini-Buckets: A General Scheme for Bounded Inference. *Journal of the ACM*, 50(2):107 – 153, 2003.

[41] R. Demirer and P.P. Shenoy. Sequential Valuation Networks: A New Graphical Technique for Asymmetric Decision Problems. In *Proc. of the 6th European Conference on Symbolic and Quantitavive Approaches to Reasoning with Uncertainty (ECSQARU-01)*, pages 252–265, London, UK, 2001.

[42] D. Dubois, H. Fargier, and H. Prade. The Calculus of Fuzzy Restrictions as a Basis for Flexible Constraint Satisfaction. In *Proc. of the 2nd IEEE Conference on Fuzzy Sets*, pages 1131–1136, San Francisco, CA, 1993.

[43] D. Dubois and H. Prade. Possibility Theory as a Basis for Qualitative Decision Theory. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1925–1930, Montréal, Canada, 1995.

[44] H. Fargier and J. Lang. Uncertainty in Constraint Satisfaction Problems: A Probabilistic Approach. In *Proc. of the European Conference on Symbolic and Quantitavive Approaches of Reasoning under Uncertainty (ECSQARU-93)*, pages 97–104, Grenade, Spain, 1993.

[45] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. Mixed Constraint Satisfaction : a Framework for Decision Problems under Uncertainty. In *Proc. of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, Montréal, Canada, 1995.

[46] H. Fargier, J. Lang, and T. Schiex. Selecting Preferred Solutions in Fuzzy Constraint Satisfaction Problems. In *Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies (EUFIT-93)*, Germany, 1993.

[47] H. Fargier, J. Lang, and T. Schiex. Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 175–180, Portland, OR, USA, 1996.

[48] H. Fargier and P. Perny. Qualitative Models for Decision Under Uncertainty without the Commensurability Assumption. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 188–195, Stockholm, Sweden, 1999.

[49] R. Fikes and N. Nilsson. STRIPS: a New Approach to the Application of Theorem Proving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[50] P.C. Fishburn. *The Foundations of Expected Utility.* D. Reidel Publishing Company, Dordrecht, 1982.

[51] P. Fonk. *Réseaux d'Inférence pour le Raisonnement Possibiliste.* PhD thesis, Université de Liège, Belgique, Faculté des sciences, 1994.

[52] E. Freuder and R. Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[53] E.C. Freuder and M.J. Quinn. Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems. In *Proc. of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1076–1078, Los Angeles, CA, USA, 1985.

[54] N. Friedman and J. Halpern. Plausibility Measures: A User's Guide. In *Proc. of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 175–184, Montréal, Canada, 1995.

[55] M. Frydenberg. The Chain Graph Markov Property. *Scandinavian Journal of Statistics*, 17:333–353, 1990.

[56] L. Garcia and R. Sabbadin. Possibilistic Influence Diagrams. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, pages 372–376, Riva del Garda, Italy, 2006.

[57] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, 1979.

[58] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[59] P.H. Giang and P.P. Shenoy. A Qualitative Linear Utility Theory for Spohn's Theory of Epistemic Beliefs. In *Proc. of the 16th International Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 220–229, Stanford, California, USA, 2000.

[60] R.P. Goldman and M.S. Boddy. Expressive Planning and Explicit Knowledge. In *Proc. of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 110–117, Edinburgh, Scotland, 1996.

[61] G.Verfaillie, F.Garcia, and L.Peret. Deployment and Maintenance of a Constellation of Satellites: a Benchmark. In *Workshop on Planning under Uncertainty and Incomplete Information (ICAPS'03)*, pages 119–127, Trento, Italie), 2003.

[62] J. Halpern. Conditional Plausibility Measures and Bayesian Networks. *Journal of Artificial Intelligence Research*, 14:359–389, 2001.

[63] J.M. Hammersley and P. Clifford. Markov Fields on Finite Graphs and Lattices. Unpublished, 1971.

[64] R. Howard and J. Matheson. Influence Diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, Menlo Park, CA, USA, 1984.

[65] P. Jégou and C. Terrioux. Hybrid Backtracking bounded by Tree-decomposition of Constraint Networks. *Artificial Intelligence*, 146(1):43–75, 2003.

[66] F. Jensen, F.V. Jensen, and S. Dittmer. From Influence Diagrams to Junction Trees. In *Proc. of the 10th International Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 367–373, Seattle, WA, USA, 1994.

[67] F.V. Jensen, T.D. Nielsen, and P.P. Shenoy. Sequential Influence Diagrams: A Unified Asymmetry Framework. In *Proceedings of the Second European Workshop on Probabilistic Graphical Models (PGM-04)*, pages 121–128, Leiden, Netherlands, 2004.

[68] F.V. Jensen and M. Vomlelova. Unconstrained Influence Diagrams. In *Proc. of the 18th International Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 234–241, Seattle, WA, USA, 2002.

[69] J.Gebhardt and R.Kruse. Background and Perspectives of Possibilistic Graphical Models. In *Proc. of the European Conference on Symbolic and Quantitavive Approaches of Reasoning under Uncertainty (ECSQARU-97)*, pages 108–121, Bad Honnef, Germany, 1997.

[70] C. Jordan. Sur les Assemblages de Lignes. *Journal für die Reine und angewandte Mathematik*, 70:185–190, 1869.

[71] L. Kaelbling, M. Littman, and A. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[72] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal Constraint Reasoning with Preferences. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, WA, USA, 2001.

[73] U. Kjaerulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical Report Tech. Report. R 90-09, Dept. of Mathematics and Computer Science, Aalborg University, Denmark, 1990.

[74] D. Knuth and R. Moore. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 8(4):293–326, 1975.

[75] J. Kolhas. *Information Algebras: Generic Structures for Inference*. Springer, 2003.

[76] A.M.C.A. Koster, H.L. Bodlaender, and S.P.M. Van Hoesel. Treewidth: Computational Experiments. Technical report, Zentrum für Informationstechnik, Berlin, 2001.

[77] N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[78] J. Larrosa. On the Time Complexity of Bucket Elimination Algorithms. Technical report, An ICS technical report, 2001.

[79] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

[80] J. Larrosa and T. Schiex. In the Quest of the Best Form of Local Consistency for Weighted CSP. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 239–244, Acapulco, Mexico, 2003.

[81] S. Lauritzen and D. Nilsson. Representing and Solving Decision Problems with Limited Information. *Management Science*, 47(9):1235–1251, 2001.

[82] M. Littman, S. Majercik, and T. Pitassi. Stochastic Boolean Satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[83] W. Lovejoy. A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes. *Annals of Operations Research*, 28(1-4):47–66, 1991.

[84] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[85] A. Madsen and F.V. Jensen. Lazy Evaluation of Symmetric Bayesian Decision Problems. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 382–390, Stockholm, Sweden, 1999.

[86] D. McDermott. PDDL, the Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control, 1998.

[87] N. Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44, 1949.

[88] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58:160–205, 1992.

[89] G. Monahan. A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science*, 28(1):1–16, 1982.

[90] U. Montanari and F. Rossi. Constraint Relaxation may be Perfect. *Artificial Intelligence*, 48:143–170, 1991.

[91] P. Ndilikilikesha. Potential Influence Diagrams. *International Journal of Approximated Reasoning*, 10:251–285, 1994.

[92] T.D. Nielsen and F.V. Jensen. Representing and solving asymmetric decision problems. *International Journal of Information Technology and Decision Making*, 2:217–263, 2003.

[93] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[94] J. Park and A. Darwiche. Complexity Results and Approximation Strategies for MAP Explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004.

[95] J. Pearl. Fusion, Propagation and Structuring in Belief Networks. *Artificial Intelligence*, 29:241–288, 1986.

[96] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[97] P. Perny, O. Spanjaard, and P. Weng. Algebraic Markov Decision Processes. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, Scotland, 2005.

[98] M.S. Pini, F. Rossi, K.B. Venable, and S. Bistarelli. Bipolar Preference Problems. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, Riva del Garda, Italy, 2006.

[99] C. Pralet, T. Schiex, and G. Verfaillie. Algorithmes et Complexités Génériques pour Différents Cadres de Décision Séquentielle dans l'Incertain. *Revue d'Intelligence Artificielle, à paraître.*

[100] C. Pralet, T. Schiex, and G. Verfaillie. Decomposition of Multi-Operator Queries on Semiring-based Graphical Models. In *Proc. of the 12th International Conference on Principles and Practice of Constraint Programming (CP-06)*, pages 437–452, Nantes, France, 2006.

[101] C. Pralet, T. Schiex, and G. Verfaillie. From Influence Diagrams to Multioperator Cluster DAGs. In *Proc. of the 22nd International Conference on Uncertainty in Artificial Intelligence (UAI-06)*, Cambridge, MA, USA, 2006.

[102] C. Pralet, T. Schiex, and G. Verfaillie. Une Nouvelle Architecture de Calcul pour Résoudre des Diagrammes d'Influence. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA-06)*, Toulouse, France, 2006.

[103] C. Pralet, G. Verfaillie, and T. Schiex. An Algebraic Graphical Model for Decision with Uncertainties, Feasibilities, and Utilities. *Journal of Artificial Intelligence Research, to appear.*

[104] C. Pralet, G. Verfaillie, and T. Schiex. Un Cadre Graphique et Algébrique pour les Problèmes de Décision incluant Incertitudes, Faisabilités et Utilités. *Revue d'Intelligence Artificielle, à paraître.*

[105] C. Pralet, G. Verfaillie, and T. Schiex. Composite Graphical Models for Reasoning about Uncertainties, Feasibilities, and Utilities. In *Proc. of the CP-05 International Workshop on "Preferences and Soft Constraints"*, Sitges, Spain, 2005.

[106] C. Pralet, G. Verfaillie, and T. Schiex. Requêtes Complexes sur des Réseaux de Croyance-Faisabilité-Désir. In *Journées Francophones de Programmation par Contraintes (JFPC-05)*, Lens, France, 2005.

[107] C. Pralet, G. Verfaillie, and T. Schiex. Décision avec Incertitudes, Faisabilités et Utilités: vers un Cadre Algébrique Unifié. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA-06)*, Toulouse, France, 2006.

[108] C. Pralet, G. Verfaillie, and T. Schiex. Decision with Uncertainties, Feasibilities, and Utilities: Towards a Unified Algebraic Framework. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, pages 427–431, Riva del Garda, Italy, 2006.

[109] C. Pralet, G. Verfailllie, and T. Schiex. Belief and Desire Networks for Answering Complex Queries. In *Proc. of the CP-04 Workshop on "Constraint Solving under Change and Uncertainty"*, Toronto, Canada, 2004.

[110] R.C. Prim. Shortest Connection Networks and some Generalisations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[111] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.

[112] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *IEEE*, volume 77(2), pages 257–286, 1989.

[113] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and Their Applications. In *IEEE /ACM International Conference on CAD*, pages 188–191, Santa Clara, California, USA, 1993. IEEE Computer Society Press.

[114] C.P. Robert and G. Casella. *Monte Carlo Statistical Methods.* Springer-Verlag, second edition, 2004.

[115] N. Robertson and P.D. Seymour. Graph Minors ii: Algorithmic Aspects of Treewidth. *Journal of Algorithms*, 7:309–322, 1986.

[116] D.J. Rose. Triangulated Graphs and the Elimination Process. *Journal of Mathematical Analysis and Applications*, 32, 1970.

[117] F. Rossi, B. Venable, and N. Yorke-Smith. Simple Temporal Problems with Preferences and Uncertainty. In *Proc. of the CP-03 Workshop on "Handling Change and Uncertainty"*, Cork, Ireland, 2003.

[118] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach (second edition).* Prentice-Hall, 2003.

[119] R. Sabbadin. A Possibilistic Model for Qualitative Sequential Decision Problems under Uncertainty in Partially Observable Environments. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 567–574, Stockholm, Sweden, 1999.

[120] H. Samulowitz and F. Bacchus. Using SAT in QBF. In *Proc. of the 11th International Conference on Principles and Practice of Constraint Programming (CP-05)*, pages 578–592, Sitges, Spain, 2005.

[121] T. Sang, P. Beame, and H. Kautz. Solving Bayesian Networks by Weighted Model Counting. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 475–482, Pittsburgh, PA, USA, 2005.

[122] T. Schiex. Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints ?". In *Proc. of the 8th International Conference on Uncertainty in Artificial Intelligence (UAI-92)*, Stanford, CA, USA, 1992.

[123] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 631–637, Montréal, Canada, 1995.

[124] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.

[125] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

[126] L. Shapiro and R. Haralick. Structural Descriptions and Inexact Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[127] P. Shenoy. Valuation-based Systems for Discrete Optimization. *Uncertainty in Artificial Intelligence*, 6:385–400, 1991.

[128] P. Shenoy. Valuation-based Systems for Bayesian Decision Analysis. *Operations Research*, 40(3):463–484, 1992.

[129] P. Shenoy. Conditional Independence in Valuation-Based Systems. *International Journal of Approximated Reasoning*, 10(3):203–234, 1994.

[130] P.P. Shenoy. Valuation Network Representation and Solution of Asymmetric Decision Problems. *European Journal of Operational Research*, 121:579–608, 2000.

[131] J.E. Smith, S. Holtzman, and J.E. Matheson. Structuring Conditional Relationships in Influence Diagrams. *Operations Research*, 41:280–297, 1993.

[132] E.J. Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971.

[133] W. Spohn. A General Non-Probabilistic Theory of Inductive Reasoning. In *Proc. of the 6th International Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pages 149–158, Cambridge, MA, USA, 1990.

[134] T.Vidal and M.Ghallab. Dealing with Uncertain Durations in Temporal Constraint Networks dedicated to Planning. In *Proc. of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, Hungary, 1996.

[135] G. Verfaillie and C. Pralet. The Basic Ingredients of a Constraint-based Framework for Decision-making under Uncertainty. In *Proc. of the CP-05 International Workshop on "Constraint solving under Change and Uncertainty"*, Sitges, Spain, 2005.

[136] T. Vidal and H. Fargier. Handling Contingency in Temporal Constraint Networks: From Consistency to Controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[137] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behaviour.* Princeton University Press, 1944.

[138] T. Walsh. Stochastic Constraint Programming. In *Proc. of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 111–115, Lyon, France, 2002.

[139] P. Weng. Axiomatic Foundations for a Class of Generalized Expected Utility: Algebraic Expected Utility. In *Proc. of the 22nd International Conference on Uncertainty in Artificial Intelligence (UAI-06)*, Cambridge, MA, USA, 2006.

[140] E. Weydert. General Belief Measures. In *Proc. of the 10th International Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 575–582, 1994.

[141] N. Wilson. Decision-Making with Belief Functions and Pignistic Probabilities. In *Proc. of the European Conference on Symbolic and Quantitavive Approaches of Reasoning under Uncertainty (ECSQARU-93)*, pages 364–371, Grenade, Spain, 1993.

[142] N. Wilson. An Order of Magnitude Calculus. In *Proc. of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 548–555, Montréal, Canada, 1995.

[143] H.L.S. Younes and M.L. Littman. PPDDL: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, Pittsburgh, PA, 2004.

[144] N.L. Zhang, R. Qi, and D. Poole. A computational theory of decision networks. *International Journal of Approximated Reasoning*, 2(11):83–158, 1994.

Appendix A

Notations

Symbol	*Meaning*
$\oplus_p$	Elimination operator on plausibilities
$\oplus_u$	Elimination operator on utilities
$\otimes_p$	Combination operator for plausibilities
$\otimes_u$	Combination operator for utilities
$\otimes_{pu}$	Combination operator between plausibilities and utilities
$\preceq_p$	Partial order on plausibilities
$\preceq_u$	Partial order on utilities
$\star$	Truncation operator
$\Diamond$	Unfeasible value
V_E	Environment variables
V_D	Decision variables
$dom(x)$	Domain of values of a variable x
$dom(S)$	$\prod_{x \in S} dom(x)$
G	Directed Acyclic Graph (DAG)
$pa_G(x)$	Parents of x in the DAG G
$nd_G(x)$	Non-descendant of x in the DAG G
$\mathcal{C}_E(G)$	Set of environment components of G
$\mathcal{C}_D(G)$	Set of decision components of G
P_i	Plausibility function
F_i	Feasibility function
U_i	Utility function
$Fact(c)$	P_i or F_i factors associated with a component c
$sc(L_i)$	Scope of a local function L_i
$\mathcal{P}_S$	Plausibility distribution over S
$\mathcal{P}_{S_1 \mid S_2}$	Conditional plausibility distribution of S_1 given S_2
$\mathcal{F}_S$	Feasibility distribution over S
$\mathcal{F}_{S_1 \mid S_2}$	Conditional feasibility distribution of S_1 given S_2
Sov	Sequence of operator-variable(s) pairs
Sem-Ans(Q)	Semantic answer to a query Q (decision trees)
Op-Ans(Q)	Operational answer to a query Q
Ans(Q)	Answer to a query Q

Table A.1: Notations.

Appendix B

Proofs

B.1 Proofs of Chapter 3

Proof of Proposition 3.10 (page 59). It is sufficient to verify each of the required axioms successively. □

Proof of Proposition 3.7 (page 56). Given that $\oplus_p$ is associative and commutative, $\oplus_{p_{S'}} \mathcal{P}_{S'} = \oplus_{p_{S'}} (\oplus_{p_{S-S'}} \mathcal{P}_S) = \oplus_{p_S} \mathcal{P}_S = 1_p$. Thus, $\mathcal{P}_{S'} : dom(S') \to E_p$ is a plausibility distribution over S'. □

B.2 Proofs of Chapter 4

Proof of Theorem 4.3 (page 64). Let $\mathcal{P}_S$ be a plausibility distribution over S. For all S_1, S_2 disjoint subsets of S and for all $A \in dom(S_1 \cup S_2)$ satisfying $\mathcal{P}_{S_2}(A) \neq 0_p$, let us define $\mathcal{P}_{S_1 \,|\, S_2}(A) = \max\{p \in E_p \,|\, \mathcal{P}_{S_1,S_2}(A) = p \otimes_p \mathcal{P}_{S_2}(A)\}$. We must show that the $\mathcal{P}_{S_1 \,|\, S_2}$ functions satisfy axioms a, b, c, d, e of Definition 4.2.

(a) By definition of $\mathcal{P}_{S_1 \,|\, S_2}$ and by distributivity of $\otimes_p$ over $\oplus_p$, one can write

$$\mathcal{P}_{S_2} = \oplus_{p_{S_1}} \mathcal{P}_{S_1,S_2} = \oplus_{p_{S_1}} (\mathcal{P}_{S_1 \,|\, S_2} \otimes_p \mathcal{P}_{S_2}) = (\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2}) \otimes_p \mathcal{P}_{S_2}.$$

As $\mathcal{P}_{S_2} \preceq_p \mathcal{P}_{S_2}$, one can infer that $\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2} \preceq_p 1_p$. Let A_2 be an assignment of S_2 satisfying $\mathcal{P}_{S_2}(A_2) \neq 0_p$. Assume that the hypothesis (H): "$\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2}(A_2) \prec_p 1_p$" holds.

Then, for all $A_1 \in dom(S_1)$, $\mathcal{P}_{S_1,S_2}(A_1.A_2) \prec_p \mathcal{P}_{S_2}(A_2)$, since if $\mathcal{P}_{S_1,S_2}(A_1.A_2) = \mathcal{P}_{S_2}(A_2)$, then $\mathcal{P}_{S_1 \,|\, S_2}(A_1.A_2) = 1_p$, which implies that $\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2}(A_2) \succeq_p 1_p$ by monotonicity of $\oplus_p$.

Moreover, (H) implies that there exists a unique $p \in E_p$ satisfying $(\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2}(A_2)) \oplus_p p = 1_p$. Combining this equation by $\mathcal{P}_{S_2}(A_2)$ gives $\mathcal{P}_{S_2}(A_2) \oplus_p \mathcal{P}_{S_2}(A_2) \otimes_p p = \mathcal{P}_{S_2}(A_2)$, i.e. $\mathcal{P}_{S_2}(A_2) \otimes_p (1_p \oplus_p p) = \mathcal{P}_{S_2}(A_2)$. This implies that $1_p \oplus_p p \preceq_p 1_p$. Given that $1_p \oplus_p p \succeq_p 1_p$ (by monotonicity of $\oplus_p$), we obtain $1_p \oplus_p p = 1_p$. We analyze two cases.

- If $p \prec_p 1_p$, there exists a unique p' satisfying $p' \oplus_p p = 1_p$. As both $(\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2}(A_2)) \oplus_p p = 1_p$ and $1_p \oplus_p p = 1_p$, this entails that $\oplus_{p_{S_1}} \mathcal{P}_{S_1 \,|\, S_2}(A_2) = 1_p$, which contradicts (H).
- If $p = 1_p$, then $1_p \oplus_p 1_p = 1_p$. This entails that $\oplus_p$ is idempotent. Let dom' be a subset of $dom(S_1)$ such that $\oplus_{p_{A_1 \in dom'}} \mathcal{P}_{S_1,S_2}(A_1.A_2) = \mathcal{P}_{S_2}(A_2)$. Let $A'_1 \in dom'$. One can

write:

$$\begin{cases} \mathcal{P}_{S_1,S_2}(A_1'.A_2) \oplus_p (\oplus_{p_{A_1 \in dom' - \{A_1'\}}} \mathcal{P}_{S_1,S_2}(A_1.A_2)) = \mathcal{P}_{S_2}(A_2) \\ \mathcal{P}_{S_1,S_2}(A_1'.A_2) \oplus_p (\oplus_{p_{A_1 \in dom'}} \mathcal{P}_{S_1,S_2}(A_1.A_2)) = \mathcal{P}_{S_2}(A_2) \text{ (as } \oplus_p \text{ is idempotent)} \end{cases}.$$

As $\mathcal{P}_{S_1,S_2}(A_1'.A_2) \prec_p \mathcal{P}_{S_2}(A_2)$, there exists a unique $p'' \in E_p$ such that $\mathcal{P}_{S_1,S_2}(A_1'.A_2) \oplus_p p'' = \mathcal{P}_{S_2}(A_2)$. Therefore, $\oplus_{p_{A_1 \in dom'}} \mathcal{P}_{S_1,S_2}(A_1.A_2) = \oplus_{p_{A_1 \in dom' - \{A_1'\}}} \mathcal{P}_{S_1,S_2}(A_1.A_2)$, which gives $\oplus_{p_{A_1 \in dom' - \{A_1'\}}} \mathcal{P}_{S_1,S_2}(A_1.A_2) = \mathcal{P}_{S_2}(A_2)$.

The assumption $\oplus_{p_{A_1 \in dom'}} \mathcal{P}_{S_1,S_2}(A_1.A_2) = \mathcal{P}_{S_2}(A_2)$ holds for $dom' = dom(S_1)$. Recursively applying the previous mechanism by removing one assignment in dom' at each iteration leads to $\oplus_{p_{A_1 \in dom'}} \mathcal{P}_{S_1,S_2}(A_1.A_2) = \mathcal{P}_{S_2}(A_2)$ with $|dom'| = 1$, i.e. it leads to $\mathcal{P}_{S_1,S_2}(A_1''.A_2) = \mathcal{P}_{S_2}(A_2)$ with $dom' = \{A_1''\}$. As a result, we obtain a contradiction.

In both cases, a contradiction with (H) is obtained, whereby $\oplus_{p_{S_1}} \mathcal{P}_{S_1 \mid S_2}(A_2) = 1_p$.

(b) $\mathcal{P}_{S_1} = \mathcal{P}_{S_1 \mid \emptyset} \otimes_p \mathcal{P}_{\emptyset} = \mathcal{P}_{S_1 \mid \emptyset} \otimes_p (\oplus_{p_S} \mathcal{P}_S) = \mathcal{P}_{S_1 \mid \emptyset} \otimes_p 1_p = \mathcal{P}_{S_1 \mid \emptyset}$.

(d) Let $A \in dom(S_1 \cup S_2 \cup S_3)$ satisfying $\mathcal{P}_{S_2,S_3}(A) \neq 0_p$. Then, $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = \mathcal{P}_{S_1 \mid S_2,S_3}(A) \otimes_p \mathcal{P}_{S_2 \mid S_3}(A)$ holds, because:

- If $\mathcal{P}_{S_1,S_2,S_3}(A) \prec_p \mathcal{P}_{S_3}(A)$, then, there exists a unique $p \in E_p$ such that $\mathcal{P}_{S_1,S_2,S_3}(A) = p \otimes_p \mathcal{P}_{S_3}(A)$. As both $\mathcal{P}_{S_1,S_2,S_3}(A) = \mathcal{P}_{S_1,S_2 \mid S_3}(A) \otimes_p \mathcal{P}_{S_3}(A)$ (by definition of $\mathcal{P}_{S_1,S_2 \mid S_3}$) and $\mathcal{P}_{S_1,S_2,S_3}(A) = \mathcal{P}_{S_1 \mid S_2,S_3}(A) \otimes_p \mathcal{P}_{S_2 \mid S_3}(A) \otimes_p \mathcal{P}_{S_3}(A)$ (by definition of $\mathcal{P}_{S_1 \mid S_2,S_3}$ and $\mathcal{P}_{S_2 \mid S_3}$), this implies that $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = \mathcal{P}_{S_1 \mid S_2,S_3}(A) \otimes_p \mathcal{P}_{S_2 \mid S_3}(A)$.
- Otherwise, $\mathcal{P}_{S_1,S_2,S_3}(A) = \mathcal{P}_{S_3}(A)$. This implies that $1_p \preceq_p \mathcal{P}_{S_1,S_2 \mid S_3}(A)$ and, as $\mathcal{P}_{S_1,S_2 \mid S_3}(A) \preceq_p 1_p$, that $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = 1_p$. Similarly, this entails that $\mathcal{P}_{S_2 \mid S_3}(A) = 1_p$ and $\mathcal{P}_{S_1 \mid S_2,S_3}(A) = 1_p$ (the monotonicity of $\oplus_p$ implies that $\mathcal{P}_{S_1,S_2,S_3}(A) = \mathcal{P}_{S_2,S_3}(A) = \mathcal{P}_{S_3}(A)$). As $1_p = 1_p \otimes_p 1_p$, we get $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = \mathcal{P}_{S_1 \mid S_2,S_3}(A) \otimes_p \mathcal{P}_{S_2 \mid S_3}(A)$.

(c)
$$\begin{aligned} \oplus_{p_{S_1}} \mathcal{P}_{S_1,S_2 \mid S_3} &= \oplus_{p_{S_1}} (\mathcal{P}_{S_1 \mid S_2,S_3} \otimes_p \mathcal{P}_{S_2 \mid S_3}) \text{ (using (d))} \\ &= (\oplus_{p_{S_1}} \mathcal{P}_{S_1 \mid S_2,S_3}) \otimes_p \mathcal{P}_{S_2 \mid S_3} \text{ (because } \otimes_p \text{ distributes over } \oplus_p) \\ &= \mathcal{P}_{S_2 \mid S_3} \text{ (using (a))} \end{aligned}$$

(e) Assume that $\mathcal{P}_{S_1,S_2,S_3} = \mathcal{P}_{S_1 \mid S_3} \otimes_p \mathcal{P}_{S_2 \mid S_3} \otimes_p \mathcal{P}_{S_3}$. Let $A \in dom(S_1 \cup S_2 \cup S_3)$ such that $\mathcal{P}_{S_3}(A) \neq 0_p$. Then, $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = \mathcal{P}_{S_1 \mid S_3}(A) \otimes_p \mathcal{P}_{S_2 \mid S_3}(A)$ holds, because:

- If $\mathcal{P}_{S_1,S_2,S_3}(A) \prec_p \mathcal{P}_{S_3}(A)$, there exists a unique $p \in E_p$ such that $\mathcal{P}_{S_1,S_2,S_3}(A) = p \otimes_p \mathcal{P}_{S_3}(A)$, and therefore $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = \mathcal{P}_{S_1 \mid S_3}(A) \otimes_p \mathcal{P}_{S_2 \mid S_3}(A)$.
- Otherwise, one can write $\mathcal{P}_{S_1 \mid S_3}(A) = \mathcal{P}_{S_2 \mid S_3}(A) = \mathcal{P}_{S_1,S_2 \mid S_3}(A) = 1_p$ by using a reasoning similar to the one of (d), and therefore $\mathcal{P}_{S_1,S_2 \mid S_3}(A) = \mathcal{P}_{S_1 \mid S_3}(A) = \mathcal{P}_{S_2 \mid S_3}(A)$.

□

Proof of Proposition 4.5 (page 65).

1. *Symmetry axiom*: directly satisfied by commutativity of $\otimes_p$.

2. *Decomposition axiom*: Assume that $I(S_1, S_2 \cup S_3 \,|\, S_4)$ holds. Then,
$$\begin{aligned} \mathcal{P}_{S_1,S_2 \,|\, S_4} &= \oplus_{p_{S_3}} \mathcal{P}_{S_1,S_2,S_3 \,|\, S_4} \\ &= \oplus_{p_{S_3}} (\mathcal{P}_{S_1 \,|\, S_4} \otimes_p \mathcal{P}_{S_2,S_3 \,|\, S_4}) \text{ (since } I(S_1, S_2 \cup S_3 \,|\, S_4)) \\ &= \mathcal{P}_{S_1 \,|\, S_4} \otimes_p (\oplus_{p_{S_3}} \mathcal{P}_{S_2,S_3 \,|\, S_4}) \text{ (by distributivity of } \otimes_p \text{ over } \oplus_p) \\ &= \mathcal{P}_{S_1 \,|\, S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_4} \end{aligned}$$
It proves that $I(S_1, S_2 \,|\, S_4)$ holds.

3. *Weak union axiom*: Assume that $I(S_1, S_2 \cup S_3 \,|\, S_4)$ holds. The decomposition axiom entails that $I(S_1, S_3 \,|\, S_4)$ is also satisfied. Then,
$$\begin{aligned} \mathcal{P}_{S_1,S_2,S_3,S_4} &= \mathcal{P}_{S_1,S_2,S_3 \,|\, S_4} \otimes_p \mathcal{P}_{S_4} \text{ (chain rule)} \\ &= \mathcal{P}_{S_1 \,|\, S_4} \otimes_p \mathcal{P}_{S_2,S_3 \,|\, S_4} \otimes_p \mathcal{P}_{S_4} \text{ (since } I(S_1, S_2 \cup S_3 \,|\, S_4)) \\ &= \mathcal{P}_{S_1 \,|\, S_4} \otimes_p \mathcal{P}_{S_3 \,|\, S_4} \otimes_p \mathcal{P}_{S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_3,S_4} \text{ (chain rule)} \\ &= \mathcal{P}_{S_1,S_3 \,|\, S_4} \otimes_p \mathcal{P}_{S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_3,S_4} \text{ (since } I(S_1, S_3 \,|\, S_4)) \\ &= \mathcal{P}_{S_1 \,|\, S_3,S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_3,S_4} \otimes_p \mathcal{P}_{S_3,S_4} \text{ (chain rule)} \end{aligned}$$
From axiom (e) in Definition 4.2, one can infer that $\mathcal{P}_{S_1,S_2 \,|\, S_3,S_4} = \mathcal{P}_{S_1 \,|\, S_3,S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_3,S_4}$, i.e. $I(S_1, S_2 \,|\, S_3 \cup S_4)$ holds.

4. *Contraction axiom* Assume that $I(S_1, S_2 \,|\, S_4)$ and $I(S_1, S_3 \,|\, S_2 \cup S_4)$ hold. Then,
$$\begin{aligned} \mathcal{P}_{S_1,S_2,S_3 \,|\, S_4} &= \mathcal{P}_{S_1,S_3 \,|\, S_2,S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_4} \text{ (chain rule)} \\ &= \mathcal{P}_{S_1 \,|\, S_2,S_4} \otimes_p \mathcal{P}_{S_3 \,|\, S_2,S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_4} \text{ (since } I(S_1, S_3 \,|\, S_2 \cup S_4)) \\ &= \mathcal{P}_{S_1,S_2 \,|\, S_4} \otimes_p \mathcal{P}_{S_3 \,|\, S_2,S_4} \text{ (chain rule)} \\ &= \mathcal{P}_{S_1 \,|\, S_4} \otimes_p \mathcal{P}_{S_2 \,|\, S_4} \otimes_p \mathcal{P}_{S_3 \,|\, S_2,S_4} \text{ (since } I(S_1, S_2 \,|\, S_4)) \\ &= \mathcal{P}_{S_1 \,|\, S_4} \otimes_p \mathcal{P}_{S_2,S_3 \,|\, S_4} \text{ (chain rule)} \end{aligned}$$
It proves that $I(S_1, S_2 \cup S_3 \,|\, S_4)$ holds.

□

Proof of Theorem 4.8 (page 66).

(a) First, if $|\mathcal{C}(G)| = 1$, G contains a unique component c_1. Then, $\otimes_{p_{c \in \mathcal{C}(G)}} \mathcal{P}_{c \,|\, pa_G(c)} = \mathcal{P}_{c_1 \,|\, \emptyset} = \mathcal{P}_{c_1}$: the proposition holds for $|\mathcal{C}(G)| = 1$.

Assume that the proposition holds for all DAGs with n components. Let G be a DAG of components compatible with a plausibility distribution $\mathcal{P}_S$ and such that $|\mathcal{C}(G)| = n + 1$. Let c_0 be a component labeling a leaf of G. As G is compatible with $\mathcal{P}_S$, one can write $I(c_0, nd_G(c_0) - pa_G(c_0) \,|\, pa_G(c_0))$. As c_0 is a leaf, $nd_G(c_0) = S - c_0$, and consequently $I(c_0, (S - c_0) - pa_G(c_0) \,|\, pa_G(c_0))$. This means that $\mathcal{P}_{S - pa_G(c_0) \,|\, pa_G(c_0)} = \mathcal{P}_{c_0 \,|\, pa_G(c_0)} \otimes_p \mathcal{P}_{(S - c_0) - pa_G(c_0) \,|\, pa_G(c_0)}$. Combining each side of the equation by $\mathcal{P}_{pa_G(c_0)}$ gives
$$\mathcal{P}_S = \mathcal{P}_{c_0 \,|\, pa_G(c_0)} \otimes_p \mathcal{P}_{S - c_0}.$$

Let G' be the DAG obtained from G by deleting the node labeled with c_0. Then, for every component $c \in \mathcal{C}(G')$, $pa_{G'}(c) = pa_G(c)$ (since the deleted component c_0 is a leaf). Moreover $nd_{G'}(c)$ equals either $nd_G(c)$ or $nd_G(c) - c_0$ (again, since the deleted component c_0 is a leaf). In the first case ($nd_{G'}(c) = nd_G(c)$), the property $I(c, nd_G(c) - pa_G(c) \,|\, pa_G(c))$ directly implies $I(c, nd_{G'}(c) - pa_{G'}(c) \,|\, pa_{G'}(c))$. In the second case ($nd_{G'}(c) = nd_G(c) - c_0$), the decomposition axiom allows us to write $I(c, nd_{G'}(c) - pa_{G'}(c) \,|\, pa_{G'}(c))$ from $I(c, nd_G(c) - pa_G(c) \,|\, pa_G(c))$. Consequently, G' is a DAG compatible with $\mathcal{P}_{S - c_0}$. As $|\mathcal{C}(G')| = n$, the recurrence assumption gives $\mathcal{P}_{S-c_0} = \otimes_{p_{c\in\mathcal{C}(G')}} \mathcal{P}_{c\,|\,pa_G(c)}$, which implies that $\mathcal{P}_S = \otimes_{p_{c\in\mathcal{C}(G)}} \mathcal{P}_{c\,|\,pa_G(c)}$. This ends the proof by recurrence.

(b) Assume that for every component c, $L_{c,pa_G(c)}(A)$ is a plausibility distribution over c for all assignments A of $pa_G(c)$. For $|\mathcal{C}(G)| = 1$, $\mathcal{C}(G) = \{c_1\}$. Then, $\gamma_S = L_{c_1}$ is a plausibility distribution over c_1. Moreover, as $\gamma_{\emptyset\,|\,\emptyset} = 1_p$, one can write $\gamma_{c_1\cup\emptyset\,|\,\emptyset} = \gamma_{c_1\,|\,\emptyset} \otimes_p \gamma_{\emptyset\,|\,\emptyset}$, i.e. $I(c_1, \emptyset\,|\,\emptyset)$. Therefore, G is compatible with γ_{c_1}: the proposition holds for $|\mathcal{C}(G)| = 1$.

Assume that the proposition holds for all DAGs with n components. Let us consider a DAG G with $n+1$ components. We first show that γ_S is a plausibility distribution over S, i.e. $\oplus_{p_S} (\otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)}) = 1_p$. Let c_0 be a leaf component in G. As c_0 is a leaf, the unique scoped function whose scope contains a variable in c_0 is $L_{c_0,pa_G(c_0)}$. By distributivity of $\otimes_p$ over $\oplus_p$, this implies that

$$\oplus_{p_{c_0}} (\otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)}) = (\oplus_{p_{c_0}} L_{c_0,pa_G(c_0)}) \otimes_p (\otimes_{p_{c\in\mathcal{C}(G)-\{c_0\}}} L_{c,pa_G(c)})$$

Given that $L_{c_0,pa_G(c_0)}(A)$ is a plausibility distribution over c_0 for all assignments A of $pa_G(c_0)$, $\oplus_{p_{c_0}} L_{c_0,pa_G(c_0)} = 1_p$. Consequently,

$$\oplus_{p_{c_0}} (\otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)}) = \otimes_{p_{c\in\mathcal{C}(G)-\{c_0\}}} L_{c,pa_G(c)}$$

Applying the recurrence hypothesis to the DAG with n components obtained from G by deleting c_0, one can infer that $\oplus_{p_{S-c_0}} (\otimes_{p_{c\in\mathcal{C}(G)-\{c_0\}}} L_{c,pa_G(c)}) = 1_p$. This allows us to write $\oplus_{p_{S-c_0}} (\oplus_{p_{c_0}} (\otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)})) = 1_p$, i.e. $\oplus_{p_S} \gamma_S = 1_p$: γ_S is a plausibility distribution over S. It remains to prove that G is a DAG of components compatible with γ_S. Let $c \in \mathcal{C}(G)$. We must show that $I(c, nd_G(c) - pa_G(c)\,|\,pa_G(c))$ holds. Two cases are analyzed.

1. If $c = c_0$, we must prove $\gamma_{c_0,nd_G(c_0)-pa_G(c_0)\,|\,pa_G(c_0)} = \gamma_{c_0\,|\,pa_G(c_0)} \otimes_p \gamma_{nd_G(c_0)-pa_G(c_0)\,|\,pa_G(c_0)}$. First,

$$\begin{aligned}
\gamma_{c_0,pa_G(c_0)} &= \oplus_{p_{S-(c_0\cup pa_G(c_0))}} (\otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)}) \\
&= (\oplus_{p_{S-(c_0\cup pa_G(c_0))}} (\otimes_{p_{c\in\mathcal{C}(G)-\{c_0\}}} L_{c,pa_G(c)})) \otimes_p L_{c_0,pa_G(c_0)} \\
&\quad \text{(because } \otimes_p \text{ distributes over } \oplus_p \text{ and } sc(L_{c_0,pa_G(c_0)}) \subset c_0 \cup pa_G(c_0) \\
&= (\oplus_{p_{S-pa_G(c_0)}} (\otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)})) \otimes_p L_{c_0,pa_G(c_0)} \\
&\quad \text{(because } \otimes_p \text{ distributes over } \oplus_p \text{ and } \oplus_{c_0} L_{c_0,pa_G(c_0)} = 1_p) \\
&= \gamma_{pa_G(c_0)} \otimes_p L_{c_0,pa_G(c_0)}
\end{aligned}$$

From this, it is possible to write:

$$\begin{aligned}
&\gamma_{nd_G(c_0)-pa_G(c_0)\,|\,pa_G(c_0)} \otimes_p \gamma_{c_0\,|\,pa_G(c_0)} \otimes_p \gamma_{pa_G(c_0)} \\
&= \gamma_{nd_G(c_0)-pa_G(c_0)\,|\,pa_G(c_0)} \otimes_p \gamma_{c_0,pa_G(c_0)} \\
&= \gamma_{nd_G(c_0)-pa_G(c_0)\,|\,pa_G(c_0)} \otimes_p \gamma_{pa_G(c_0)} \otimes_p L_{c_0,pa_G(c_0)} \\
&= \gamma_{nd_G(c_0)} \otimes_p L_{c_0,pa_G(c_0)} \\
&= \gamma_{S-\{c_0\}} \otimes_p L_{c_0,pa_G(c_0)} \text{ (because } c_0 \text{ is a leaf in } G) \\
&= (\otimes_{p_{c\in\mathcal{C}(G)-\{c_0\}}} L_{c,pa_G(c)}) \otimes_p L_{c_0,pa_G(c_0)} \\
&= \otimes_{p_{c\in\mathcal{C}(G)}} L_{c,pa_G(c)} \\
&= \gamma_S
\end{aligned}$$

Using axiom (e) of Definition 4.2, this entails that $\gamma_{nd_G(c_0)-pa_G(c_0)\,|\,pa_G(c_0)} \otimes_p \gamma_{c_0\,|\,pa_G(c_0)} = \gamma_{S-pa_G(c_0)\,|\,pa_G(c_0)}$, i.e., as $S = c_0 \cup nd_G(c_0)$, that $I(c_0, nd_G(c_0) - pa_G(c_0)\,|\,pa_G(c_0))$.

2. Otherwise, $c \neq c_0$. Let G' be the DAG obtained from G by deleting c_0. G' contains n components: the recurrence hypothesis enables us to write $I(c, nd_{G'}(c) - pa_{G'}(c)\,|\,pa_{G'}(c))$.

As c_0 is a leaf in G, $c_0 \notin pa_G(c)$, which implies $pa_{G'}(c) = pa_G(c)$. Thus, $I(c, nd_{G'}(c) - pa_G(c) \,|\, pa_G(c))$.

(i) If $nd_{G'}(c) = nd_G(c)$, then $I(c, nd_G(c) - pa_G(c) \,|\, pa_G(c))$ directly holds.

(ii) Otherwise, $nd_{G'}(c) \neq nd_G(c)$. As c_0 is a leaf in G, this is equivalent to say that $nd_G(c) = nd_{G'}(c) \cup c_0$. This means that c is not an ancestor of c_0, and a fortiori $c \notin pa_G(c_0)$. In the following, the four semigraphoid axioms are used to prove the required result. From the decomposition axiom, from $I(c_0, nd_G(c_0) - pa_G(c_0) \,|\, pa_G(c_0))$, and from $(c \cup nd_{G'}(c)) \subset nd_G(c_0)$ (because $nd_G(c_0) = S - c_0$), it is possible to infer that $I(c_0, (c \cup nd_{G'}(c)) - pa_G(c_0) \,|\, pa_G(c_0))$, or, in other words, as $c \cap pa_G(c_0) = \emptyset$, that $I(c_0, c \cup (nd_{G'}(c) - pa_G(c_0)) \,|\, pa_G(c_0))$. Using the weak union axiom leads to $I(c_0, c \,|\, (nd_{G'}(c) - pa_G(c_0)) \cup pa_G(c_0))$ and, using the symmetry axiom, to $I(c, c_0 \,|\, (nd_{G'}(c) - pa_G(c_0)) \cup pa_G(c_0))$. As shown previously, $I(c, nd_{G'}(c) - pa_G(c) \,|\, pa_G(c))$. Together with $I(c, c_0 \,|\, (nd_{G'}(c) - pa_G(c_0)) \cup pa_G(c_0))$, the contraction axiom allows us to infer $I(c, (nd_{G'}(c) - pa_G(c)) \cup c_0 \,|\, pa_G(c))$. As $c_0 \notin pa_G(c)$ and $nd_G(c) = nd_{G'}(c) \cup c_0$, this means that $I(c, nd_G(c) - pa_G(c)) \,|\, pa_G(c))$.

We have proved that G is compatible with γ_S. Consequently, the proposition holds if there are $n+1$ components in G, which ends the proof by recurrence.

□

Proof of Proposition 4.10 (page 67). Let $n \in \mathbb{N}^*$. If $\oplus_{p_{i \in [1,n]}} 1_p = 1_p$, then $p_0 = 1_p$ satisfies the required property. Moreover, in this case, the distributivity of $\otimes_p$ over $\oplus_p$ implies that for all $p \in E_p$, $\oplus_{p_{i \in [1,n]}} p = p$. Therefore, if $\oplus_{p_{i \in [1,n]}} p = 1_p$, then $p = 1_p$, which shows that p_0 is unique.

Otherwise, $\oplus_{p_{i \in [1,n]}} 1_p \neq 1_p$. In this case, as $1_p \preceq_p \oplus_{p_{i \in [1,n]}} 1_p$ by monotonicity of $\oplus_p$, one can write $1_p \prec_p \oplus_{p_{i \in [1,n]}} 1_p$. The second item of Theorem 4.3 then implies that there exists a unique $p_0 \in E_p$ such that $1_p = p_0 \otimes_p (\oplus_{p_{i \in [1,n]}} 1_p)$, i.e. such that $1_p = \oplus_{p_{i \in [1,n]}} p_0$. □

Proof of Proposition 4.12 (page 68). $\mathcal{P}_{V_E, V_D} = \mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0$, where p_0 is the element of E_p such that $\oplus_{p_{i \in [1, |dom(V_D)|]}} p_0 = 1_p$. Then,

$$\begin{aligned}
\oplus_{p_{V_E \cup V_D}} \mathcal{P}_{V_E, V_D} &= \oplus_{p_{V_E \cup V_D}} (\mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0) \\
&= \oplus_{p_{V_D}} ((\oplus_{p_{V_E}} \mathcal{P}_{V_E \,||\, V_D}) \otimes_p p_0) \\
&= \oplus_{p_{V_D}} p_0 \\
&= \oplus_{p_{i \in [1, |dom(V_D)|]}} p_0 \\
&= 1_p
\end{aligned}$$

This proves that $\mathcal{P}_{V_E, V_D}$ is a plausibility distribution over $V_E \cup V_D$.

As $\mathcal{P}_{V_E, V_D} = \mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0$ and $\mathcal{P}_{V_E, V_D} = \mathcal{P}_{V_E \,|\, V_D} \otimes_p \mathcal{P}_{V_D}$, one can write $\mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0 = \mathcal{P}_{V_E \,|\, V_D} \otimes_p \mathcal{P}_{V_D}$. Moreover, $\mathcal{P}_{V_D} = \oplus_{p_{V_E}} \mathcal{P}_{V_E, V_D} = \oplus_{p_{V_E}} (\mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0) = p_0$. Thus, $\mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0 = \mathcal{P}_{V_E \,|\, V_D} \otimes_p p_0$. Summing this equation $|dom(V_D)|$ times with $\oplus_p$ gives $\mathcal{P}_{V_E \,||\, V_D} = \mathcal{P}_{V_E \,|\, V_D}$. □

Proof of Proposition 4.14 (page 68). The result is proved only for $\mathcal{P}_{V_E \,|\, V_D}$ (the proof for $\mathcal{F}_{V_D \,|\, V_E}$ is similar). The completion of $\mathcal{P}_{V_E \,||\, V_D}$ looks like $\mathcal{P}_{V_E, V_D} = \mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0$. G_p being compatible with this completion, Theorem 4.8a entails that $\mathcal{P}_{V_E, V_D} = \otimes_{p_{c \in \mathcal{C}(G_p)}} \mathcal{P}_{c \,|\, pa_{G_p}(c)}$. As the decision components are roots in G_p, one can infer, by successively eliminating the environment components, that $\mathcal{P}_{V_D} = \oplus_{p_{V_E}} \mathcal{P}_{V_E, V_D} = \otimes_{p_{c \in \mathcal{C}_D(G_p)}} \mathcal{P}_c$.

On the other hand, $\mathcal{P}_{V_D} = \oplus_{p\,V_E} \left(\mathcal{P}_{V_E \,||\, V_D} \otimes_p p_0\right) = p_0$. This proves that $\otimes_{p\,c \in \mathcal{C}_D(G_p)} \mathcal{P}_c = p_0$. Therefore, $\mathcal{P}_{V_E,V_D} = \mathcal{P}_{V_E \,|\, V_D} \otimes_p p_0 = (\otimes_{p\,c \in \mathcal{C}_E(G_p)} \mathcal{P}_{c \,|\, pa_{G_p}(c)}) \otimes_p p_0$. Summing this equation $|dom(V_D)|$ times with $\oplus_p$ gives $\mathcal{P}_{V_E \,|\, V_D} = \otimes_{p\,c \in \mathcal{C}_E(G_p)} \mathcal{P}_{c \,|\, pa_{G_p}(c)}$. As $\mathcal{C}_E(G_p) = \mathcal{C}_E(G)$ and $pa_{G_p}(c) = pa_G(c)$ for every $c \in \mathcal{C}_E(G)$, this entails that $\mathcal{P}_{V_E \,|\, V_D} = \otimes_{p\,c \in \mathcal{C}_E(G)} \mathcal{P}_{c \,|\, pa_G(c)}$. □

B.3 Proofs of Chapter 5

Proof of Proposition 5.3 (page 77). Proposition 5.3 is entailed by the DAG structure: indeed, as variables are organized in a DAG, it is sufficient to build a sequence Sov as follows. At the beginning, $Sov = \emptyset$ and G is the DAG of the PFU network. While the DAG G is not empty, (1) select a leaf component c in G; (2) if c is a decision component, then $Sov \leftarrow (\max, c).Sov$; otherwise, $Sov \leftarrow (\oplus_u, c).Sov$; (3) delete c from G. □

Proof of Proposition 5.5 (page 78). We denote by p_0 the element in E_p such that the completion of $\mathcal{P}_{V_E \,||\, V_D}$ equals $\mathcal{P}_{V_E \,||\, V_D} \otimes p_0$. Note that $p_0 \neq 0_p$, since it must satisfy $\oplus_{p\,i \in [1,|dom(V_D)|]}\, p_0 = 1_p$.

Lemma B.1. *Let $(E_p, \oplus_p, \otimes_p)$ be a conditionable plausibility structure. Then, $(p_1 \otimes_p p_2 = 0_p) \leftrightarrow ((p_1 = 0_p) \vee (p2 = 0_p))$.*

Proof of Lemma B.1. First, if $p_1 = 0_p$ or $p_2 = 0_p$, then $p_1 \otimes_p p_2 = 0_p$. Conversely, assume that $p_1 \otimes_p p_2 = 0_p$. Then, if $p_1 \succ_p 0_p$, the conditionability of the plausibility structure together with $p_1 \otimes_p 0_p = 0_p$ entails that $p_2 = 0_p$. Similarly, if $p_2 \succ_p 0_p$, then $p_1 = 0_p$. Therefore $(p_1 \otimes_p p_2 = 0_p) \rightarrow ((p_1 = 0_p) \vee (p_2 = 0_p))$. □

Lemma B.2. *Assume that the plausibility structure is conditionable. Let S_1, S_2 be disjoint subsets of V_E. Then, $\mathcal{P}_{S_1 \,|\, S_2 \,||\, V_D} = \mathcal{P}_{S_1 \,|\, S_2, V_D}$.*

Proof of Lemma B.2. On one hand, $\mathcal{P}_{S_1,S_2 \,|\, V_D} = \mathcal{P}_{S_1 \,|\, S_2,V_D} \otimes_p \mathcal{P}_{S_2 \,|\, V_D}$. On the other hand, $\mathcal{P}_{S_1,S_2 \,|\, V_D} = \mathcal{P}_{S_1,S_2 \,||\, V_D} = \mathcal{P}_{S_1 \,|\, S_2 \,||\, V_D} \otimes_p \mathcal{P}_{S_2 \,||\, V_D} = \mathcal{P}_{S_1 \,|\, S_2 \,||\, V_D} \otimes_p \mathcal{P}_{S_2 \,|\, V_D}$.

Let A be an assignment of V. If $\mathcal{P}_{S_1,S_2 \,|\, V_D}(A) \prec_p \mathcal{P}_{S_2 \,|\, V_D}(A)$, then the conditionability of the plausibility structure entails that $\mathcal{P}_{S_1 \,|\, S_2,V_D}(A) = \mathcal{P}_{S_1 \,|\, S_2 \,||\, V_D}(A)$. Otherwise, $\mathcal{P}_{S_1,S_2 \,|\, V_D}(A) = \mathcal{P}_{S_2 \,|\, V_D}(A)$, which also entails that $\mathcal{P}_{S_1,S_2 \,||\, V_D}(A) = \mathcal{P}_{S_2 \,||\, V_D}(A)$. In this case, $\mathcal{P}_{S_1 \,|\, S_2,V_D}(A) = \mathcal{P}_{S_1 \,||\, S_2,V_D}(A) = 1_p$. Therefore, $\mathcal{P}_{S_1 \,|\, S_2,V_D} = \mathcal{P}_{S_1 \,|\, S_2 \,||\, V_D}$. □

(1) Assume that $V_E \neq \emptyset$. Let S_i be the leftmost set of environment variables appearing in Sov and let $A \in dom(l(S_i))$. Using $l(S_i) \cap V_E = \emptyset$, one can write $\mathcal{P}_{l(S_i)}(A) = \oplus_{p\,V - l(S_i)} \mathcal{P}_{V_E,V_D}(A) = \oplus_{p\,V_D - l(S_i)}(\oplus_{p\,V_E} \mathcal{P}_{V_E,V_D}(A)) = \oplus_{p\,V_D - l(S_i)}\, p_0 \neq 0_p$. Therefore, $\mathcal{P}_{S_i \,|\, l(S_i)}(A)$ is well-defined.

(4) Let $l_E(S_i) = l(S_i) \cap V_E$ and $l_D(S_i) = l(S_i) \cap V_D$. For a set of variables S, we denote by $d_G(S)$ the set of variables in V which are descendant in the DAG G of at least one variable in S.

First, $\mathcal{P}_{S_i,l_E(S_i) \,||\, V_D} = \oplus_{p\,V_E - (S_i \cup l_E(S_i))} \mathcal{P}_{V_E \,||\, V_D} = \oplus_{p\,V_E - (S_i \cup l_E(S_i))} (\otimes_{p\,P_j \in P} P_j)$. By definition of a query, variables in $V_E \cap d_G(V_D - l_D(S_i))$ do not belong to $S_i \cup l_E(S_i)$ (the environment variables that are descendant of as-yet-unassigned decision variables are not assigned yet, either).

Thus, $\mathcal{P}_{S_i,l_E(S_i) \,||\, V_D} = \oplus_{p\,V_E - (S_i \cup l_E(S_i) \cup d_G(V_D - l_D(S_i)))} (\otimes_{p\,P_j \notin Fact(c), c \subset V_E \cap d_G(V_D - l_D(S_i))} P_j)$. This equality is obtained by successively eliminating (using the normalization conditions) the environment components included in $d_G(V_D - l_D(S_i))$. As the scope of a plausibility function $P_j \in Fact(c)$ is included in $c \cup pa_G(c)$, this equality entails that $\mathcal{P}_{S_i,l_E(S_i)\,||\,V_D}$ does not depend on the assignment of $V_D - l_D(S_i)$. Morever, $\mathcal{P}_{l_E(S_i)\,||\,V_D} = \oplus_{S_i} \mathcal{P}_{S_i,l_E(S_i)\,||\,V_D}$ does not depend on the assignment of V_D too. As $\mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D} = max\{p \in E_p \,|\, \mathcal{P}_{S_i,l_E(S_i)\,||\,V_D} = p \otimes_p \mathcal{P}_{l_E(S_i)\,||\,V_D}\}$, this also entails that $\mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D}$ does not depend on the assignment of V_D. It can be denoted $\mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,l_D(S_i)}$.

Let us show that $\mathcal{P}_{S_i\,|\,l(S_i)} = \mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,l_D(S_i)}$. First,

$$\begin{aligned}
\mathcal{P}_{S_i,l(S_i)} &= \oplus_{p_{V_D - l_D(S_i)}} \mathcal{P}_{S_i,l_E(S_i),V_D} = \oplus_{p_{V_D - l_D(S_i)}}(\mathcal{P}_{S_i\,|\,l_E(S_i),V_D} \otimes_p \mathcal{P}_{l_E(S_i),V_D}) \\
&= \oplus_{p_{V_D - l_D(S_i)}}(\mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D} \otimes_p \mathcal{P}_{l_E(S_i),V_D}) \text{ (using Lemma B.2)} \\
&= \mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D} \otimes_p (\oplus_{p_{V_D - l_D(S_i)}} \mathcal{P}_{l_E(S_i),V_D}) \\
&\qquad \text{(since } \mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D} \text{ does not depend on the assignment of } V_D - l(S_i)) \\
&= \mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D} \otimes_p \mathcal{P}_{l(S_i)}
\end{aligned}$$

Let A be an assignment of V.

- If $\mathcal{P}_{S_i,l(S_i)}(A) \prec_p \mathcal{P}_{l(S_i)}(A)$, then the conditionability of the plausibility structure directly entails that $\mathcal{P}_{S_i\,|\,l(S_i)}(A) = \mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D}(A)$.
- Otherwise, $\mathcal{P}_{S_i,l(S_i)}(A) = \mathcal{P}_{l(S_i)}(A)$. In this case, $\mathcal{P}_{S_i\,|\,l(S_i)}(A) = 1_p$. Next, on one hand, $\mathcal{P}_{l(S_i)} = \oplus_{p_{V - l(S_i)}}(\mathcal{P}_{V_E\,||\,V_D} \otimes_p p_0) = \oplus_{p_{V_D - l_D(S_i)}}(\mathcal{P}_{l_E(S_i)\,||\,V_D} \otimes_p p_0)$. On the other hand, $\mathcal{P}_{S_i,l(S_i)} = \oplus_{p_{V-(S_i \cup l(S_i))}}(\mathcal{P}_{V_E\,||\,V_D} \otimes_p p_0) = \oplus_{p_{V_D - l_D(S_i)}}(\mathcal{P}_{S_i,l_E(S_i)\,||\,V_D} \otimes_p p_0)$. As $\mathcal{P}_{S_i,l(S_i)}(A) = \mathcal{P}_{l(S_i)}(A)$, one can infer that $\oplus_{p_{V_D - l_D(S_i)}}(\mathcal{P}_{l_E(S_i)\,||\,V_D}(A) \otimes_p p_0) = \oplus_{p_{V_D - l_D(S_i)}}(\mathcal{P}_{S_i,l_E(S_i)\,||\,V_D}(A) \otimes_p p_0)$. As neither $\mathcal{P}_{l_E(S_i)\,||\,V_D}$ nor $\mathcal{P}_{S_i,l_E(S_i)\,||\,V_D}$ depends on the assignment of $V_D - l_D(S_i)$, this entails that $\mathcal{P}_{l_E(S_i)\,||\,V_D}(A) \otimes_p (\oplus_{p_{V_D - l_D(S_i)}} p_0) = \mathcal{P}_{S_i,l_E(S_i)\,||\,V_D}(A) \otimes_p (\oplus_{p_{V_D - l_D(S_i)}} p_0)$. Summing this equation $|dom(l_D(S_i))|$ times gives $\mathcal{P}_{S_i,l_E(S_i)\,||\,V_D}(A) = \mathcal{P}_{l_E(S_i)\,||\,V_D}(A)$, and thus $\mathcal{P}_{S_i\,|\,l_E(S_i)\,||\,V_D}(A) = 1_p = \mathcal{P}_{S_i\,|\,l(S_i)}(A)$.

The results can be extended to feasibilities.

(2) Let $i, j \in [1, k]$ such that $i < j$, $S_i \subset V_E$, $S_j \subset V_E$, and $r(S_i) \cap l(S_j) \subset V_D$ (S_j is the first set of environment variables appearing at the right of S_i in *Sov*). Let $(A, A') \in dom(l(S_i)) \times dom(S_i)$ such that $\mathcal{P}_{S_i\,|\,l(S_i)}(A)$ is well-defined (i.e. $\mathcal{P}_{l(S_i)}(A) \neq 0_p$) and $\mathcal{P}_{S_i\,|\,l(S_i)}(A.A') \neq 0_p$. Let A'' be an extension of $A.A'$ over $l(S_j)$. We must show that $\mathcal{P}_{S_j\,|\,l(S_j)}(A'')$ is well-defined, i.e. that $\mathcal{P}_{l(S_j)}(A'') \neq 0_p$. As $\mathcal{P}_{S_i\,|\,l(S_i)}(A.A') \neq 0_p$ and $\mathcal{P}_{l(S_i)}(A) \neq 0_p$, Lemma B.1 implies that $\mathcal{P}_{S_i,l(S_i)}(A.A') \neq 0_p$. Similarly to the proof of point (4), it is possible to show that $\mathcal{P}_{l(S_j)}$ does not depend on the assignment of $l(S_j) - (S_i \cup l(S_i))$. Therefore, for every A'' extending $A.A'$ over $l(S_j)$, $\oplus_{p_{l(S_j)-(S_i \cup l(S_i))}} \mathcal{P}_{l(S_j)}(A'') \neq 0_p$, which implies that $\mathcal{P}_{l(S_j)}(A'') \neq 0_p$.

(3) Proof similar to point (2), except that plausibilities are replaced by feasibilities and decision variables are replaced by environment ones.

□

Proof of Theorem 5.9 (page 81). Let A_{fr} be an assignment of the set of free variables V_{fr} such that $\mathcal{F}_{V_{fr}}(A_{fr}) = f$. The semantic based definition gives $(\textit{Sem-Ans}(Q))(A_{fr}) = \Diamond$. Given that $\mathcal{F}_{V_{fr}}(A_{fr}) = \vee_{V - V_{fr}} \mathcal{F}_{V_E,V_D}(A_{fr}) = \vee_{V - V_{fr}} \mathcal{F}_{V_D\,||\,V_E}(A_{fr}) = \vee_{V - V_{fr}}(\wedge_{F_i \in F} F_i(A_{fr}))$ (since the completion of $\mathcal{F}_{V_D\,||\,V_E}$ gives $\mathcal{F}_{V_D\,||\,V_E} = \mathcal{F}_{V_D,V_E}$), one can infer that for every complete assignment

A'' extending A_{fr}, $\wedge_{F_i \in F} F_i(A'') = f$ and $(\wedge_{F_i \in F} F_i(A''))\star(\otimes_{p P_i \in P} P_i(A''))\otimes_{pu}(\otimes_{u U_i \in U} U_i(A'')) = \Diamond$. As $\min(\Diamond, \Diamond) = \max(\Diamond, \Diamond) = \Diamond \oplus_u \Diamond = \Diamond$, this entails that $(Op\text{-}Ans(Q))(A_{fr}) = \Diamond$ too.

We now analyze the case $\mathcal{F}_{V_{fr}}(A_{fr}) = t$. We use A'' to denote a complete assignment which must be considered with the semantic definition. Using the properties:

- $p \otimes_{pu} \min(u_1, u_2) = \min(p \otimes_{pu} u_1, p \otimes_{pu} u_2)$ (right monotonicity of $\otimes_{pu}$),
- $p \otimes_{pu} \max(u_1, u_2) = \max(p \otimes_{pu} u_1, p \otimes_{pu} u_2)$ (right monotonicity of $\otimes_{pu}$),
- $p \otimes_{pu} (u_1 \oplus_u u_2) = (p \otimes_{pu} u_1) \oplus_u (p \otimes_{pu} u_2)$ (distributivity of $\otimes_{pu}$ over $\oplus_u$),
- $p_1 \otimes_{pu} (p_2 \otimes_{pu} u) = (p_1 \otimes_p p_2) \otimes_{pu} u$,

one can "move" all the $\mathcal{P}_{S_i \mid l(S_i)}(A.A')$ to get, starting from the semantic definition,

$$(\otimes_{p i \in [1,k], S_i \subset V_E} \mathcal{P}_{S_i \mid l(S_i)})(A'') \otimes_{pu} \mathcal{U}_V(A'')$$

on the right of the elimination operators.

Let us prove that this quantity equals $\mathcal{P}_{V_E \mid V_D}(A'') \otimes_{pu} \mathcal{U}_V(A'')$. Let S be the rightmost set of quantified environment variables. The chain rule enables us to write $\mathcal{P}_{V_E \mid V_D} = \mathcal{P}_{S \mid l_E(S), V_D} \otimes_p \mathcal{P}_{l_E(S) \mid V_D}$, where $l_E(S) = l(S) \cap V_E$. Moreover, using Lemma B.2 and Proposition 5.5(4), one can write $\mathcal{P}_{S \mid l_E(S), V_D} = \mathcal{P}_{S \mid l_E(S) \mid\mid V_D} = \mathcal{P}_{S \mid l(S)}$. Therefore, $\mathcal{P}_{V_E \mid V_D} = \mathcal{P}_{S \mid l(S)} \otimes_p \mathcal{P}_{l_E(S) \mid V_D}$. Recursively applying this mechanism leads to: $\mathcal{P}_{V_E \mid V_D} = \otimes_{p i \in [1,k], S_i \subset V_E} \mathcal{P}_{S_i \mid l(S_i)}$. Therefore, we obtain $\mathcal{P}_{V_E \mid V_D}(A'') \otimes_{pu} \mathcal{U}_V(A'')$ on the right of the elimination operators.

The semantic definition of the query meaning can be updated a bit, thanks to Lemma B.1. This lemma implies that conditions like $\mathcal{P}_{S \mid l(S)}(A.A') \neq 0_p$, which are used only when $\mathcal{P}_{l(S)}(A) \neq 0_p$, are equivalent to $\mathcal{P}_{S, l(S)}(A.A') \neq 0_p$, since $\mathcal{P}_{S, l(S)}(A.A') = \mathcal{P}_{S \mid l(S)}(A.A') \otimes_p \mathcal{P}_{l(S)}(A)$. As a result, the operators $\oplus_{u A' \in dom(S), \mathcal{P}_{S \mid l(S)}(A.A') \neq 0_p}$ can be replaced by $\oplus_{u A' \in dom(S), \mathcal{P}_{S, l(S)}(A.A') \neq 0_p}$.

Similarly, in the eliminations $\min_{A' \in dom(S), \mathcal{F}_{S \mid l(S)}(A.A') = t}$, the conditions $\mathcal{F}_{S \mid l(S)}(A.A') = t$ can be replaced by $\mathcal{F}_{S, l(S)}(A.A') = t$. The same holds for the eliminations $\max_{a \in dom(x_i), \mathcal{F}_{S \mid l(S)}(A.A') = t}$.

We now start from the operational definition and show that it can be reformulated as above. The operational definition applies a sequence of eliminations over the variables domains, on the global function $(\wedge_{F_i \in F} F_i) \star (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\wedge_{U_i \in U} U_i)$, which also equals $\mathcal{F}_{V_D \mid V_E} \star \mathcal{P}_{V_E \mid V_D} \otimes_{pu} \mathcal{U}_V$. Let S be the leftmost set of quantified decision variables. Let A be an assignment of $l(S)$. Assume that S is quantified by min. Let $A_0 \in dom(S)$ such that $\mathcal{F}_{S, l(S)}(A.A_0) = f$. It can be inferred that for all complete assignment A'' extending $A.A_0$, $\mathcal{F}_{V_E, V_D}(A'') = f$, and consequently $\mathcal{F}_{V_D \mid V_E}(A'') = f$. This implies that $\mathcal{F}_{V_D \mid V_E}(A'') \star \mathcal{P}_{V_E \mid V_D}(A'') \otimes_{pu} \mathcal{U}_V(A'') = \Diamond$. Given that $\min(\Diamond, \Diamond) = \max(\Diamond, \Diamond) = \Diamond \oplus_u \Diamond = \Diamond$, we obtain $Qo_r(\mathcal{N}, Sov, A.A_0) = \Diamond$. As $\min(d, \Diamond) = d$, this entails that $\min_{A' \in dom(S)} Qo_r(\mathcal{N}, Sov, A.A') = \min_{A' \in dom(S) - \{A_0\}} Qo_r(\mathcal{N}, Sov, A.A')$. Thus, $\min_{A' \in dom(S)}$ can be replaced by $\min_{A' \in dom(S), \mathcal{F}_{S, l(S)}(A.A') = t}$ (as $\mathcal{F}_{V_{fr}}(A) = t$, there exists at least one assignment $A' \in dom(S)$ such that $\mathcal{F}_{S, l(S)}(A.A') = t$). The same result holds if S is quantified by max. Applying this mechanism to each set of quantified decision variables from the left to the right of Sov, we obtain that $\min_{A' \in dom(S)}$ and $\max_{A' \in dom(S)}$ can be replaced by $\min_{A' \in dom(S), \mathcal{F}_{S, l(S)}(A.A') = t}$ and $\max_{A' \in dom(S), \mathcal{F}_{S, l(S)}(A.A') = t}$ respectively. Moreover, it can be shown that for every complete assignment A'' which is now considered, $\mathcal{F}_{V_D \mid V_E}(A'') = t$. It is then possible to replace $\mathcal{F}_{V_D \mid V_E}(A'') \star \mathcal{P}_{V_E \mid V_D}(A'') \otimes_{pu} \mathcal{U}_V(A'')$ by $\mathcal{P}_{V_E \mid V_D}(A'') \otimes_{pu} \mathcal{U}_V(A'')$.

We now update each $\oplus_{u A' \in dom(S)} Qo_r(\mathcal{N}, Sov, A.A')$. Let S be the leftmost set of quantified environment variables. Let A be an assignment of $l(S)$. Let $A_0 \in dom(S)$ such that $\mathcal{P}_{S,l(S)}(A.A_0) = 0_p$. Then, for all complete assignments A'' extending $A.A_0$, $\mathcal{P}_{V_E \mid V_D}(A'') = 0_p$, and thus $\mathcal{P}_{V_E \mid V_D}(A'') \otimes_{pu} \mathcal{U}_V(A'') = 0_u$. As $\min(0_u, 0_u) = \max(0_u, 0_u) = 0_u \oplus_u 0_u = 0_u$, we obtain $Qo_r(\mathcal{N}, Sov, A.A_0) = 0_u$. As $d \oplus_u 0_u = d$, computing $\oplus_{u A' \in dom(S)} Qo_r(\mathcal{N}, Sov, A.A')$ is equivalent to computing $\oplus_{u A' \in dom(S) - \{A_0\}} Qo_r(\mathcal{N}, Sov, A.A')$. Thus, $\oplus_{u A' \in dom(S)}$ can be replaced by $\oplus_{u A' \in dom(S), \mathcal{P}_{S,l(S)}(A.A') \neq 0_p}$ (as $\mathcal{P}_{l(S)}(A) \neq 0_p$, there exists at least one assignment $A \in dom(S)$ satisfying $\mathcal{P}_{S,l(S)}(A.A') \neq 0_p$). Applying this mechanism, considering each set of quantified environment variables from the left to the right of Sov, enables us to get $\oplus_{u A' \in dom(S), \mathcal{P}_{S,l(S)}(A,A') \neq 0_p}$ instead of $\oplus_{u A' \in dom(S)}$.

Consequently, we have found a function Φ such that $Sem\text{-}Ans(Q) = \Phi$ and $Op\text{-}Ans(Q) = \Phi$. Moreover, the optimal policies for the decisions for $Sem\text{-}Ans(Q)$ are optimal policies for decisions for Φ. Indeed, the transformation rules used preserve the set of optimal policies. The same holds for $Op\text{-}Ans(Q)$ and Φ. It entails that $Sem\text{-}Ans(Q) = Op\text{-}Ans(Q)$, and that the optimal policies for $Sem\text{-}Ans(Q)$ are the same as those for $Op\text{-}Ans(Q)$. □

Proof of Theorem 5.12 (page 82).

Lemma B.3. *Let $(E_p, E_u, \oplus_u, \otimes_{pu})$ be an expected utility structure such that E_u is totally ordered by $\preceq_u$. Let γ_{S_1,S_2} be a local function on E_u, whose scope is $S_1 \cup S_2$. Then,*

$$\max_{\phi: dom(S_2) \to dom(S_1)} \bigoplus_{u\, A \in dom(S_2)} \gamma_{S_1,S_2}(\phi(A).A) = \oplus_{u\, S_2} \max_{S_1} \gamma_{S_1,S_2}$$

Moreover, $\psi : dom(S_2) \to dom(S_1)$ satisfies $(\max_{S_1} \gamma_{S_1,S_2})(A) = \gamma_{S_1,S_2}(\psi(A).A)$ for all $A \in dom(S_2)$ iff $\max_{\phi: dom(S_2) \to dom(S_1)} \oplus_{u A \in dom(S_2)} \gamma_{S_1,S_2}(\phi(A).A) = \oplus_{u A \in dom(S_2)} \gamma_{S_1,S_2}(\psi(A).A)$. In other words, the two sides of the equality have the same set of optimal policies for S_1.

Proof of Lemma B.3 (page 203). Let $\phi_0 : dom(S_2) \to dom(S_1)$ be a function such that

$$\max_{\phi: dom(S_2) \to dom(S_1)} \oplus_{u A \in dom(S_2)} \gamma_{S_1,S_2}(\phi(A).A) = \oplus_{u A \in dom(S_2)} \gamma_{S_1,S_2}(\phi_0(A).A).$$

Given that for all $A \in dom(S_2)$, $\gamma_{S_1,S_2}(\phi_0(A).A) \preceq_u \max_{A' \in dom(S_1)} \gamma_{S_1,S_2}(A'.A)$, the monotonicity of $\oplus_u$ entails that $\oplus_{u A \in dom(S_2)} \gamma_{S_1,S_2}(\phi_0(A).A) \preceq_u \oplus_{u A \in dom(S_2)} \max_{A' \in dom(S_1)} \gamma_{S_1,S_2}(A'.A)$. Thus,

$$\max_{\phi: dom(S_2) \to dom(S_1)} \oplus_{u A \in dom(S_2)} \gamma_{S_1,S_2}(\phi(A).A) \preceq_u \oplus_{u S_2} \max_{S_1} \gamma_{S_1,S_2}.$$

On the other hand, let $\psi_0 : dom(S_2) \to dom(S_1)$ be a function such that $\forall A \in dom(S_2)$, $(\max_{S_1} \gamma_{S_1,S_2})(A) = \gamma_{S_1,S_2}(\psi_0(A).A)$. Then,

$$\oplus_{u S_2} \max_{S_1} \gamma_{S_1,S_2} = \bigoplus_{u\, A \in dom(S_2)} \gamma_{S_1,S_2}(\psi_0(A).A) \preceq_u \max_{\phi: dom(S_2) \to dom(S_1)} \bigoplus_{u\, A \in dom(S_2)} \gamma_{S_1,S_2}(\phi(A).A).$$

The antisymmetry of $\preceq_u$ implies the required equality. The equality of the set of optimal policies over S_1 is directly implied by the equality. □

We now give the proof of the theorem, which uses for some cases the previous lemma.

1. (*CSP based problems [84]*)

   Let us consider a CSP over a set of variables V and with a set of constraints $\{C_1, \ldots, C_m\}$.

(a) (*Consistency, solution finding*) Consistency can be checked with the query $Q = (\mathcal{N}, (\max, V))$, where $\mathcal{N} = (V, G, \emptyset, \emptyset, U)$ (all variables in V are decision variables, G is reduced to a unique decision component containing all variables, and $U = \{C_1, \dots, C_m\}$), and where the expected utility structure is boolean optimistic expected conjunctive utility (row 6 in Table 3.1). Computing $Ans(Q) = \max_V (C_1 \wedge \ldots \wedge C_m)$ is equivalent to checking consistency, because $Ans(Q) = t$ iff there exists an assignment of V satisfying $C_1 \wedge \ldots \wedge C_m$, i.e. iff the CSP is consistent. In order to get a solution when $Ans(Q) = t$, it suffices to record an optimal decision rule for V. Integer Linear Programming [124] with finite domain variables can be formulated as a CSP.

(b) (*Counting the number of solutions*) The expected utility structure considered for this task is probabilistic expected satisfaction (row 2 in Table 3.1). The PFU network is $\mathcal{N} = (V, G, P, \emptyset, U)$, where all variables in V are environment variables, G is a DAG with a unique component $c_0 = V$, $P = \{L_0\}$, L_0 being a constant factor equal to $1/|dom(V)|$ such that $Fact(c_0) = \{L_0\}$, and $U = \{C_1, \dots, C_m\}$. Implicitly, L_0 specifies that the complete assignments are equiprobable. It enables the normalization condition "for all $c \in \mathcal{C}_E(G)$, $\oplus_{p_c} \otimes_{p P_i \in Fact(c)} P_i = 1_p$" to be satisfied, since $\sum_V (1/|dom(V)|) = 1$. The query to consider is then $Q = (\mathcal{N}, (+, V))$. It is not hard to check that this satisfies the conditions imposed on queries and $Ans(Q) = \sum_V (1/L_0 \times (C_1 \times \ldots \times C_m))$ gives the percentage of solutions of the CSP. $L_0 \times Ans(Q)$ gives the number of solutions.

2. (*Solving a Valued CSP (VCSP [123])*)
In order to model this problem, the only difficulty lies in the definition of an expected utility structure. In a VCSP, a triple $(E, \circledast, \succ)$ called a valuation structure is introduced. It satisfies properties such as $(E, \circledast)$ is a commutative semigroup, $\succ$ is a total order on E, and E has a minimum element denoted $\top$. The expected utility structure to consider is the following one: $(E_p, \oplus_p, \otimes_p) = (\{t, f\}, \vee, \wedge)$, $(E_u, \otimes_u) = (E, \circledast)$, and the expected utility structure is $(E_p, E_u, \oplus_u, \otimes_{pu})$, with $\oplus_u = \min$ and $\otimes_{pu}$ defined by "$false \otimes_{pu} u = \top$ and $true \otimes_{pu} u = u$" (it is not hard to verify that this structure is an expected utility structure). Next, the PFU network is $\mathcal{N} = (V, G, \emptyset, \emptyset, U)$, where V is the set of variables of the VCSP, G is a DAG with only one decision component containing all the variables, and U contains the soft constraints. The query $Q = (\min, V)$ enables us to find the minimum violation degree of the soft constraints. A solution for the VCSP is an optimal (argmin) decision rule for V.

3. (*Problems from the SAT framework [82]*)
In the SAT framework, queries on a conjunctive normal form boolean formula ϕ over a set of variables $V = \{x_1, \dots, x_n\}$ are asked.

Let us first prove that an *extended SSAT* formula can be evaluated with a PFU query. An extended SSAT formula is defined by a triple (ϕ, θ, q) where ϕ is a boolean formula in conjunctive normal form, θ is a threshold in $[0, 1]$, and $q = (q_1 x_1) \ldots (q_n x_n)$ is a sequence of quantifier/variable pairs (the quantifiers are $\exists$, $\forall$, or Я; the meaning of Я appears below). If one takes $f \prec t$, the value of ϕ under the quantification sequence q, $val(\phi, q)$, is defined recursively by: (i) $val(\phi, \emptyset) = 1$ if ϕ is t, 0 otherwise; (ii) $val(\phi, (\exists x)\, q') = \max_x val(\phi, q')$; (iii) $val(\phi, (\forall x)\, q') = \min_x val(\phi, q')$; (iv) $val(\phi, (\text{Я} x)\, q') = \sum_x 0.5 \cdot val(\phi, q')$. Intuitively, the last case means that Я quantifies boolean variables taking equiprobable values. An extended SSAT formula (ϕ, θ, q) is t iff $val(\phi, q) \geq \theta$. If S denotes the set of variables quantified by Я, an equivalent definition of $val(\phi, q)$ is: (i') $val(\phi, \emptyset) = 0.5^{|S|}$ if ϕ is t, 0 otherwise; (ii') $val(\phi, (\exists x)\, q') = \max_x val(\phi, q')$; (iii') $val(\phi, (\forall x)\, q') = \min_x val(\phi, q')$; (iv') $val(\phi, (Я x)\, q') = \sum_x val(\phi, q')$.

This second definition proves that $val(\phi, q)$ can be computed with the PFU query defined by: (a) expected utility structure: probabilistic expected satisfaction (row 2 in Table 3.1); (b) PFU network: $\mathcal{N} = (V, G, P, \emptyset, U)$, with V the set of variables of the formula ϕ (the decision variables are the variables quantified by $\exists$ or $\forall$), G a DAG without arcs, with one decision component per decision variable and a unique environment component containing all variables quantified by Я, $P = \{L_0\}$, L_0 being a constant factor equal to $0.5^|V_E|$, and U the set of clauses of ϕ; (c) query: $Q = (\mathcal{N}, Sov)$, Sov being obtained from q by replacing $\exists$, $\forall$, and Я by max, min, and + respectively. Then, $Ans(Q) = val(\phi, q)$, which implies that the value of an extended SSAT formula (ϕ, θ, q) is the value of the bounded query $(\mathcal{N}, Sov, \theta)$.

SSAT is a particular case of extended-SSAT and is therefore covered. *SAT, MAJSAT, E-MAJSAT, QBF* are also particular cases of extended SSAT. As a result, they are covered by PFU bounded queries. More precisely, SAT corresponds to a bounded query of the form $Q = (\mathcal{N}, (\max, V), 1)$; MAJSAT ("given a boolean formula over a set of variables V, is it satisfied for at least half of the assignments of V") corresponds to a bounded query of the form $(\mathcal{N}, (+, V), 0.5)$; E-MAJSAT ("given a boolean formula over $V = V_E \cup V_D$, does there exist an assignment of V_D such that the formula is satisfied for at least half of the assignments of V_E?") corresponds to a bounded query of the form $(\mathcal{N}, (\max, V_D).(+, V_E), 0.5)$; QBF corresponds to a bounded query in which max over existentially quantified variables and min over universally quantified variables alternate.

4. (*Solving a Quantified CSP (QCSP [15])*)

   A QCSP represents a formula of the form $Q_1x_1 \ldots Q_nx_n\ (C_1 \wedge \ldots \wedge C_m)$, where each Q_i is a quantifier ($\forall$ or $\exists$) and each C_i is a constraint. The value of a QCSP is defined recursively as follows: the value of a QCSP without variables (i.e. containing only t, f, and connectives) is given by the definition of the connectives. A QCSP $\exists x\ qcsp$ is t iff either $qcsp((x,t)) = t$ or $qcsp((x,f)) = t$. Assuming $f \prec t$, it gives that $\exists x\ qcsp$ is t iff $\max_x qcsp = t$. A QCSP $\forall x\ qcsp$ is t iff $qcsp((x,t)) = t$ and $qcsp((x,f)) = t$. Equivalently, $\forall x\ qcsp$ is t iff $\min_x qcsp = t$. It implies that the value of a QCSP is actually given by the formula $op(Q_1)_{x_1} \ldots op(Q_n)_{x_n}\ (C_1 \wedge \ldots \wedge C_m)$, with $op(\exists) = \max$ and $op(\forall) = \min$. It corresponds to the answer to the query $(\mathcal{N}, (op(Q_1), x_1). \ldots .(op(Q_n), x_n))$, where $\mathcal{N} = (V, G, \emptyset, \emptyset, U)$ (V is the set of variables of the QBF, G is a DAG with only one decision component containing all variables, and U is the set of constraints), and where the expected utility structure is boolean optimistic expected conjunctive utility (row 6 in Table 3.1).

5. (*Solving a mixed CSP or a probabilistic mixed CSP [47]*)

   A *probabilistic mixed CSP* is defined by (i) a set of variables partitioned into a set W of *contingent* variables and a set X of *decision* variables; an assignment A_W of W is called a world and an assignment A_X of X is called a decision; (ii) a set $C = \{C_1, \ldots, C_m\}$ of constraints involving at least one decision variable; (iii) a probability distribution P_W over the worlds; a possible world A_W (i.e. such that $P_W(A_W) > 0$) is covered by a decision A_X iff the assignment $A_W.A_X$ satisfies all the constraints in C.

On one hand, if a decision must be made without knowing the world, the task is to find an optimal *non-conditional decision*, i.e. to find an assignment A_X of the decision variables that maximizes the probability that the world is covered by A_X. This probability is equal to $\sum_{A_W \,|\, (C_1 \times \ldots \times C_m)(A_X, A_W)=1} P_W(A_W) = \sum_W (P_W \times C_1 \times \ldots \times C_m)$. As a result, an optimal non-conditional decision can be found by recording an optimal decision rule for X for the formula $\max_X \sum_W (P_W \times C_1 \times \ldots \times C_m)$. The previous formula actually specifies how to solve such a problem with PFUs. The algebraic structure is probabilistic expected utility (row 2 in Table 3.1), the PFU network is $\mathcal{N} = (V, G, P, \emptyset, U)$, with $V_D = X$, $V_E = W$, G a DAG without arc, with one decision component X and a set of environment components that depends on how P_W is specified, P is the set of factors that define P_W, and finally $U = \{C_1, \ldots, C_m\}$. The query is then $Q = (\mathcal{N}, (\max, X).(+, W))$.

On the other hand, if the world is known when the decision is made, the task is to look for an optimal *conditional decision*, i.e. to look for a decision rule $\phi_0 : dom(W) \rightarrow dom(X)$ which maximizes the probability that the world is covered. In other words, the goal is to compute $\max_{\phi:dom(W)\rightarrow dom(X)} \sum_{A_W \in dom(W) \,|\, (C_1 \times \ldots \times C_m)(A_W.\phi(A_W))=1} P_W(A_W) =$
$\max_{\phi:dom(W)\rightarrow dom(X)} \sum_{A_W \in dom(W)} (P_W \times C_1 \times \ldots \times C_m)(A_W.\phi(A_W))$. Due to Lemma B.3, it also equals $\sum_W \max_X (P_W \times C_1 \times \ldots \times C_m)$, and ϕ_0 can be found by recording an optimal decision rule for X. It proves that the query $Q = (\mathcal{N}, (+, W).(\max, X))$ enables us to compute an optimal conditional decision.

With *Mixed CSP*, P_W is replaced by a set K of constraints defining the possible worlds. The goal is then to look for a decision, either conditional or non-conditional, that maximizes the number of covered worlds. This task is equivalent, ignoring a normalizing constant, to find a decision that maximizes the percentage of covered worlds. This can be solved using the set of plausibility functions $P = K \cup \{N_0\}$, with N_0 a normalizing constant ensuring that the normalization condition on plausibilities holds. N_0 is the number of possible worlds, but it does actually not need to be computed, since it is a constant factor and we are only interested in optimal decisions.

6. (*Stochastic CSP (SCSP) and stochastic COP (SCOP) [138]*)
   Formally, a SCSP is a tuple (V, S, P, C, θ), where V is a list of variables (each variable x having a finite domain $dom(x)$), S is the set of stochastic variables in V, $P = \{P_s \,|\, s \in S\}$ is a set of probability distributions (in a more advanced version of SCSP, probabilities over S may be defined by a Bayesian network; the subsumption result is still valid for this case), $C = \{C_1, \ldots, C_m\}$ is a set of constraints, and θ is a threshold in $[0, 1]$.

   A SCSP-policy is a tree with internal nodes labeled with variables. The root is labeled with the first variable in V, and the parents of the leaves are labeled with the last variable in V. Nodes labeled with a decision variable have only one child, whereas nodes labeled with a stochastic variable s have $|dom(s)|$ children. Leaf nodes are labeled with 1 if the complete assignment they define satisfies all the constraints in C, and with 0 otherwise. With each leaf node can be associated a probability $\prod_{s\in S} P_s(A_S)$, where A_S stands for the assignment of S implicitly defined by the path from the root to the leaf. The satisfaction of a SCSP-policy is the sum of the values of the leaves weighted by their probabilities. A SCSP is satisfiable iff there exists a SCSP-policy with a satisfaction of at least θ. The optimal satisfaction of a SCSP is the maximum satisfaction of all SCSP-policies.

For the subsumption proof, we first consider the problem of looking for the optimal satisfaction of a SCSP. In a SCSP-policy, each decision variable x can take one value per assignment of the set $pred_s(x)$ of stochastic variables which precede x in the list of variables V. Instead of being described as a tree, a SCSP-policy can be viewed as a set of functions $\Delta = \{\phi^x : dom(pred_s(x)) \rightarrow dom(x)), x \in V - S\}$, and its value is $val(\Delta) = \sum_{A_S \in dom(S)} (\prod_{s \in S} P_s \times \prod_{C_i \in C} C_i)(A_S.(\underset{x \in V-S}{\cdot} \phi^x(A_S)))$. The goal is to maximize the previous quantity among the sets Δ. Let y be the last decision variable in V, and let Φ^y be the set of local functions $\phi^y : dom(pred_s(y)) \rightarrow dom(y)$ defining a decision rule for y. Then,

$$\max_{\phi^y \in \Phi^y} val(\Delta) = \max_{\phi^y \in \Phi^y} \sum_{A_S \in dom(pred_s(y))} (\sum_{S - pred_s(y)} \prod_{s \in S} P_s \times \prod_{C_i \in C} C_i)(A_S.(\underset{x \in V-S}{\cdot} \phi_x(A_S))).$$

By Lemma B.3, the previous quantity also equals:
$\sum_{pred_s(y))} \max_y \sum_{S - pred_s(y)} \left(\prod_{s \in S} P_s \times \prod_{C_i \in C} C_i\right)$. A recursive application of this mechanism shows that the answer $Ans(Q)$ to the query Q described below is equal to the optimal satisfaction of a SCSP:

- expected utility structure: row 2 in Table 3.1 (probabilistic expected satisfaction)
- PFU network: $\mathcal{N} = (V', G, P, \emptyset, U)$, with V' the set of variables of the SCSP; $V_E = S$ and $V_D = V' - S$; G is a DAG without arcs, with one component per variable; $P = \{P_s \,|\, s \in S\}$; $Fact(\{s\}) = \{P_s\}$; U is the set of constraints of the SCSP;
- query: Q =($\mathcal{N}, Sov$), with Sov=$t(V)$ (V is the list of variables of the SCSP), $t(V)$ being recursively defined by $t(\emptyset) = \emptyset$ and $t(x.V'') = \left\{ \begin{array}{l} (+, \{x\}).t(V'') \text{ if } x \in S \\ (\max, \{x\}).t(V'') \text{ otherwise} \end{array} \right.$.

An optimal SCSP-policy can be recorded during the evaluation of $Ans(Q)$. The satisfiability of a SCSP can be answered with the bounded query $(\mathcal{N}, Sov, \theta)$. Again, a corresponding SCSP-policy can be obtained by recording optimal decision rules.

With Stochastic Constraint Optimization Problem (SCOP), the constraints in C are additive soft constraints. The subsumption proof is similar.

7. (*Classical planning problems (STRIPS-like planning [49, 58])*)
In order to search for a plan of length lesser than k, one can simply model a classical planning problem as a CSP. Such a transformation is already available in the literature [58]. However, one can also model a classical planning problem more directly in the PFU framework. More precisely, the state at one step is described by a set of boolean environment variables, one per ground atom. For each step, there is a unique decision variable whose set of values corresponds to the name of all ground instances of operators. Plausibility functions are deterministic functions which link variables in step t to variables in step $t+1$ (these functions simply specify the positive and negative effects of ground operators). The initial state is also represented by a plausibility function linking variables in step 1. Feasibility functions define preconditions for an action to be feasible. They link variables in a step t to the decision variable of that step. Utility functions are boolean functions describing the goal state. They

hold over variables in step k. In order to search for a plan of length lesser than k, the sequence of elimination is a max-elimination on all variables. The expected utility structure used is the boolean optimistic expected disjunctive utility.

8. (*Influence diagrams [64]*)
   We start from the definition of influence diagrams of Section **??**. With each decision variable d, one can associate a decision rule $\delta^d : dom(pa_G(d)) \rightarrow dom(d)$. An influence diagram policy (ID-policy) is a set $\Delta = \{\delta^d \,|\, d \in D\}$ of decision rules (one for each decision variable). The value $val(\Delta)$ of an ID-policy Δ is given by the probabilistic expectation of the utility:
   $$val(\Delta) = \sum_{A_S \in dom(S)} ((\prod_{s \in S} P_{s \,|\, pa_G(s)}) \times (\sum_{U_i \in U} U_i))(A_S.(\underset{d \in D}{\cdot} \delta^d(A_S))).$$
   To solve an influence diagram, one must compute the maximum value of the previous quantity and find an associated optimal ID-policy. Using Lemma B.3 and the DAG structure, it is possible to show, using the same ideas as in the SCSP subsumption proof, that the optimal expected utility is given by the answer to the query Q below (associated optimal decision rules can be recorded during the evaluation of $Ans(Q)$):

   - expected utility structure: row 1 in Table 3.1 (probabilistic expected utility);
   - PFU network: $\mathcal{N} = (V, G', P, \emptyset, U)$; V is the set of variables of the influence diagram, G' is the DAG obtained from the DAG of the influence diagram by removing utility nodes and arcs into decision nodes; in G', there is one component per variable; $P = \{P_{s \,|\, pa_G(s)}, s \in V_E\}$ and $Fact(\{s\}) = \{P_{s \,|\, pa_G(s)}\}$; U is the set of utility functions associated with utility nodes.
   - PFU query: $Q = (\mathcal{N}, Sov)$, with Sov obtained from the DAG of the influence diagram as follows. Initially, $Sov = \emptyset$. In the DAG of an influence diagram, the decisions are totally ordered. Let d be the first decision variable in the DAG G of the influence diagram (i.e. the decision variable with no parent decision variable). Then, repeatedly update Sov by $Sov \leftarrow Sov.(+, pa_G(d)).(\max, \{d\})$ and delete d and the variables in $pa_G(d)$ from G until no decision variable remains. Then, perform $Sov \leftarrow Sov.(+, S)$, where S is the set of chance variables that have not been deleted from G.

9. (*Finite horizon MDP [111, 89, 119, 19, 18]*) In order to prove that the encoding in the PFU framework given in Sections 4.6 and 5.6 actually enables us to solve a T time-steps probabilistic MDP, we start by reminding the algorithm used to compute an optimal MDP-policy. Usually, a decision rule for d_T is chosen by computing $V^*_{s_T} = \max_{d_T} R_{s_T,d_T}$. $V^*_{s_T}$ is the optimal reward which can be obtained in state s_T. At a time-step $i \in [1, T[$, a decision rule for d_i is chosen by computing $V^*_{s_i} = \max_{d_i} (R_{s_i,d_i} + \sum_{s_{i+1}} P_{s_{i+1} \,|\, s_i,d_i} \times V^*_{s_{i+1}})$. Last, the optimal expected value of the reward, which depends on the initial state s_1, is $V^*_{s_1}$.

   Let us prove by recurrence that for all $i \in [1, T-1]$,
   $$V^*_{s_1} = \max_{d_1} \sum_{s_2} \ldots \max_{d_i} \sum_{s_{i+1}} ((\prod_{k \in [1,i]} P_{s_{k+1} \,|\, s_k,d_k}) \times ((\sum_{k \in [1,i]} R_{s_k,d_k}) + V^*_{s_{i+1}})).$$
   This proposition holds for $i = 1$, since
   $$\begin{aligned} V^*_{s_1} &= \max_{d_1} (R_{s_1,d_1} + \sum_{s_2} P_{s_2 \,|\, s_1,d_1} \times V^*_{s_2}) \\ &= \max_{d_1} \sum_{s_2} (P_{s_2 \,|\, s_1,d_1} \times (R_{s_1,d_1} + V^*_{s_2})) \text{ (since } \sum_{s_2} P_{s_2 \,|\, s_1,d_1} = 1) \end{aligned}$$
   Moreover, if the proposition holds at step $i-1$ (with $i > 1$), then
   $$V^*_{s_1} = \max_{d_1} \sum_{s_2} \ldots \max_{d_{i-1}} \sum_{s_i} ((\prod_{k \in [1,i-1]} P_{s_{k+1} \,|\, s_k,d_k}) \times ((\sum_{k \in [1,i-1]} R_{s_k,d_k}) + V^*_{s_i})).$$

Given that

$$\begin{aligned}(\textstyle\sum_{k\in[1,i-1]} R_{s_k,d_k}) + V^*_{s_i} &= (\textstyle\sum_{k\in[1,i-1]} R_{s_k,d_k}) + \max_{d_i} (R_{s_i,d_i} + \sum_{s_{i+1}} P_{s_{i+1}\,|\,s_i,d_i} \times V^*_{s_{i+1}}) \\ &= \max_{d_i} ((\textstyle\sum_{k\in[1,i]} R_{s_k,d_k}) + \sum_{s_{i+1}} P_{s_{i+1}\,|\,s_i,d_i} \times V^*_{s_{i+1}}) \\ &= \max_{d_i} \textstyle\sum_{s_{i+1}} P_{s_{i+1}\,|\,s_i,d_i} \times ((\sum_{k\in[1,i]} R_{s_k,d_k}) + V^*_{s_{i+1}})\end{aligned}$$

(the last equality holds since $\sum_{s_{i+1}} P_{s_{i+1}\,|\,s_i,d_i} = 1$), it can be inferred that

$$\begin{aligned}&(\textstyle\prod_{k\in[1,i-1]} P_{s_{k+1}\,|\,s_k,d_k}) \times ((\sum_{k\in[1,i-1]} R_{s_k,d_k}) + V^*_{s_i}) \\ &= \max_{d_i} \textstyle\sum_{s_{i+1}} ((\prod_{k\in[1,i]} P_{s_{k+1}\,|\,s_k,d_k}) \times ((\sum_{k\in[1,i]} R_{s_k,d_k}) + V^*_{s_{i+1}}))\end{aligned}$$

which proves that the proposition holds at step i. This proves that the proposition holds at step T, and therefore $V^*_{s_1} = Ans(Q)$. Furthermore, as each step in the proof preserves the set of optimal decision rules, an optimal MDP-policy can be recorded during the evaluation of $Ans(Q)$.

We now study the case of partially observable finite horizon MDP (finite horizon POMDP). In a POMDP, one adds for each time step $t > 1$ a conditional probability distribution $P_{o_t\,|\,s_t}$ of making observation o_t at time step t given the state s_t. The value of s_t remains unobserved. We also assume that a probability distribution P_{s_1} over the initial state is available. The subsumption proof for this case is more difficult. We consider the approach of POMDP which consists in finding an optimal *policy tree*. This approach is equivalent to compute, for each decision variable d_t, a decision rule for d_t depending on the observations made so far, i.e. a function $\phi^{d_t} : dom(\{o_2, \ldots, o_t\}) \rightarrow dom(d_t)$. The set of such functions is denoted Φ^{d_t}. A set $\Delta = \{\phi^{d_1}, \ldots, \phi^{d_T}\}$ is called a POMDP-policy. The value of a POMDP-policy is recursively defined as follows. First, the value of the reward at the last decision step, which depends on the assignment A_{s_T} of s_T and on the observations $O_{2\rightarrow T}$ made from the beginning, is $V(\Delta)_{s_T,o_2,\ldots,o_t}(A_{s_T}.O_{2\rightarrow T}) = R_{s_T,d_T}(A_{s_T}, \phi^{d_T}(O_{2\rightarrow T}))$. At a time step i, the obtained reward depends on the actual state A_{s_i} and on the observations $O_{2\rightarrow i}$ made so far. Its expression is:

$$\begin{aligned}&V(\Delta)_{s_i,o_2,\ldots,o_i}(A_{s_i}.O_{2\rightarrow i}) \\ &= (R_{s_i,d_i} + \textstyle\sum_{s_{i+1}} P_{s_{i+1}\,|\,s_i,d_i} \times \sum_{o_{i+1}} P_{o_{i+1}\,|\,s_{i+1}} \times V(\Delta)_{s_{i+1},o_1,\ldots,o_{i+1}})(A)\end{aligned}$$

where $A = A_{s_i}.\phi^{d_i}(O_{2\rightarrow i}).O_{2\rightarrow i}$ (this equation is equivalent to the recursive formula used to define the value of a policy tree for a POMDP; see [71] for a more complete presentation of policy trees for finite horizon POMDP). Finally, the expected reward of the POMDP-policy Δ is $V(\Delta) = \sum_{s_1} P_{s_1} \times V(\Delta)_{s_1}$. To solve a finite horizon POMDP consists in computing the optimal expected reward among all POMDP-policies (i.e. in computing $V^* = \max_{\phi^{d_1},\ldots,\phi^{d_T}} V(\{\phi^{d_1}, \ldots, \phi^{d_T}\})$), as well as associated optimal decision rules.

Using a recurrence as in the observable MDP case, it is first possible to prove that for a problem with T steps,

$$\begin{aligned}&V^* = \max_{\phi^{d_1},\ldots,\phi^{d_T}} \textstyle\sum_{o_2,\ldots,o_T} \sum_{s_1,\ldots,s_T} \beta_V \\ &\text{with } \beta_V = (P_{s_1} \times \textstyle\prod_{i\in[1,T[} P_{s_{i+1}\,|\,s_i,d_i} \times \prod_{i\in[1,T[} P_{o_{i+1}\,|\,s_{i+1}}) \times (\sum_{i\in[1,T]} R_{s_i,d_i})\end{aligned}$$

From this, a recursive use of Lemma B.3 enables us to infer that

$$V^* = \max_{d_1} \textstyle\sum_{o_2} \max_{d_2} \sum_{o_3} \max_{d_3} \ldots \sum_{o_T} \max_{d_T} \sum_{s_1,\ldots,s_T} \beta_V.$$

It proves that the query defined below enables us to compute V^* as well as an optimal policy:

- algebraic structure: probabilistic expected utility (row 1 in Table 3.1);
- PFU network: $\mathcal{N} = (V, G, P, \emptyset, U)$; V equals $\{s_i \,|\, i \in [1, T]\} \cup \{o_i \,|\, i \in [2, T]\} \cup \{d_i \,|\, i \in$

$[1, T]\}$, with $V_D = \{d_i \,|\, i \in [1, T]\}$; G is a DAG with one variable per component; a decision component does not have any parents, an environment component $\{o_i\}$ has $\{s_i\}$ as parent, and a component $\{s_{i+1}\}$ has $\{s_i\}$ and $\{d_i\}$ as parents; $P = \{P_{s_1}\} \cup \{P_{s_{i+1} \,|\, s_i, d_i}, i \in [1, T-1]\} \cup \{P_{o_i \,|\, s_i} \,|\, i \in [2, T]\}$; $Fact(\{s_1\}) = \{P_{s_1}\}$, $Fact(\{s_{i+1}\}) = \{P_{s_{i+1} \,|\, s_i, d_i}\}$, and $Fact(\{o_i\}) = \{P_{o_i \,|\, s_i}\}$; last, $U = \{R_{s_i, d_i} \,|\, i \in [1, T]\}$;

- PFU query: based on the DAG, a necessary condition for a query to be defined is that each decision d_i must appear at the left of the variables in $\{s_k \,|\, k \in [i+1, T]\} \cup \{o_k \,|\, k \in [i+1, T]\}$; the query considered is $Q = (\mathcal{N}, Sov)$, with
$$Sov = (\max, d_1).(+, o_2).(\max, d_2). \dots .(+, o_T).(\max, d_T).(+, \{s_1, \dots, s_T\}).$$

The proofs for finite horizon (PO)MDP based on possibilities or on κ-rankings are similar. As for the subsumption of factored MDP, one can first argue that every factored MDP can be represented as a usual MDP, and therefore as a PFU query on a PFU network. Even if this is a sufficient argument, we can define a better representation of factored MDPs in the PFU framework: it corresponds to a representation where the variables describing states are directly used together with the local plausibility functions and rewards, which can be modeled by scoped functions (defined as decision trees, binary decision diagrams. . .).

10. (*Queries on Bayesian networks, Markov random fields, and chain graphs [96, 55]*)
It suffices to consider chain graphs, since Bayesian networks and Markov random fields are particular cases of chain graphs. The subsumption proofs are provided for the general case of plausibility distributions defined on a totally ordered conditionable plausibility structure.

(a) (*MAP, MPE, and probability of an evidence*) As MPE (Most Probable Explanation) and the computation of the probability of an evidence are particular cases of MAP (Maximum A Posteriori hypothesis), it suffices to prove that MAP is subsumed. The probabilistic MAP problem consists in finding, given a probability distribution $\mathcal{P}_V$, a Maximum A Posteriori explanation to an assignment of a subset O of V which has been observed (also called evidence). More formally, let D denote the set of variables on which an explanation is sought and let e denote the observed assignment of O. The MAP problem consists in finding an assignment A^* of D such that $\max_{A \in dom(D)} P_{D \,|\, O}(A.e) = P_{D \,|\, O}(A^*.e)$. As $P_{D \,|\, O} = P_{D,O}/P_O$, one can write:

$$\begin{aligned} \max\nolimits_{A \in dom(D)} P_{D \,|\, O}(A.e) &= (\max\nolimits_{A \in dom(D)} P_{D,O}(A.e))/P_O(e) \\ &= (\max\nolimits_{A \in dom(D)} \textstyle\sum_{A' \in dom(V-(D \cup O))} P_V(A.e.A'))/P_O(e) \end{aligned}$$

Thus, computing $\max_D \sum_{V-(D \cup O)} P_V(e)$ is sufficient (the difference lies only in a normalizing constant). This result can be generalized to all totally ordered conditionable plausibility structures.

Indeed, as $\otimes_p$ is monotonic, $\max_{A \in dom(D)} \mathcal{P}_{D,O}(A.e) = (\max_{A \in dom(D)} \mathcal{P}_{D \,|\, O}(A.e)) \otimes_p \mathcal{P}_O(e)$. If $\max_{A \in dom(D)} \mathcal{P}_{D,O}(A.e) \prec_p \mathcal{P}_O(e)$, then there exists a unique $p \in E_p$ such that $\max_{A \in dom(D)} \mathcal{P}_{D,O}(A.e) = p \otimes_p \mathcal{P}_O(e)$. This gives us $p = \max_{A \in dom(D)} \mathcal{P}_{D \,|\, O}(A.e)$. Otherwise, if $\max_{A \in dom(D)} \mathcal{P}_{D,O}(A.e) = \mathcal{P}_O(e)$, then one can infer that there exists $A^* \in dom(D)$ such that $\mathcal{P}_{D,O}(A^*.e) = \mathcal{P}_O(e)$, and therefore $\mathcal{P}_{D \,|\, O}(A^*.e) = 1_p$. Thus, $\max_{A \in dom(D)} \mathcal{P}_{D \,|\, O}(A.e) = 1_p$ too. This shows that determining $\max_{A \in dom(D)} \mathcal{P}_{D,O}(A.e)$ gives $\max_{A \in dom(D)} \mathcal{P}_{D \,|\, O}(A.e)$.

Moreover, if $A^* \in \text{argmax}\{\mathcal{P}_{D,O}(A'.e), A' \in dom(D)\}$, then $\max\{p \in E_p \,|\, \mathcal{P}_{D,O}(A^*.e) = p \otimes_p \mathcal{P}_O(e)\} \succeq_p \max\{p \in E_p \,|\, \mathcal{P}_{D,O}(A.e) = p \otimes_p \mathcal{P}_O(e)\}$ for all $A \in dom(D)$. Therefore, an optimal assignment of D for $\max_D \mathcal{P}_{D,O}(e)$ is also an optimal assignment of D for $\max_D \mathcal{P}_{D\,|\,O}(e)$. As a result, the MAP problem can be reduced to the computation of

$$\max_D \mathcal{P}_{D,O}(e) = \max_D \oplus_{pV-(D\cup O)} \mathcal{P}_V(e) = \max_D \oplus_{pV-D} (\mathcal{P}_V \otimes_p \delta_O)$$

where δ_O is the scoped function with scope O such that $\delta_O(e') = 1_p$ if $e' = e$, 0_p otherwise. We define a PFU query whose answer is $Ans(Q) = \max_D \oplus_{pV-D}(\mathcal{P}_V \otimes_p \delta_O)$:

- the plausibility structure is $(E_p, \oplus_p, \otimes_p)$, the utility structure is $(E_u, \otimes_u) = (E_p, \otimes_p)$, and the expected utility structure is $(E_p, E_u, \oplus_u, \otimes_{pu}) = (E_p, E_p, \oplus_p, \otimes_p)$;
- PFU network: the difficulty in the definition of the PFU network lies in the fact that normalization conditions on components must be satisfied. The idea is that only the components in which a variable in $D \cup O$ is involved have to be modified. The PFU network is $\mathcal{N} = (V, G, P, \emptyset, U)$; V the set of variables of the chain graph; $V_D = D$ and $V_E = V - D$; G is a DAG of components obtained from the DAG G' of the chain graph by splitting every component c in which a variable in $D \cup O$ is involved: such a component c is transformed into $|c|$ components containing only one variable; all these $|c|$ components become parents of the child components of c; for a component $\{x_0\}$ included in one of these $|c|$ components, if $x_0 \in D$, then $\{x_0\}$ is a decision component; otherwise, $\{x_0\}$ is an environment component, and one creates a plausibility function P_i, equal to a constant $p_0(x_0)$ such that $\oplus_{p i\in[1,|dom(x_0)|]} p_0(x_0) = 1_p$, and such that $Fact(\{x_0\}) = \{p_0(x_0)\}$; P contains first the constants defined above, and second the factors expressing $P_{c\,|\,pa_{G'}(c)}$ in the chain graph for the components c satisfying $c \cap (D \cup O) = \emptyset$; last, U contains the factors expressing $P_{c\,|\,pa_{G'}(c)}$ in the chain graph for the components c such that $c \cap (D \cup O) \neq \emptyset$, and a constant factor $p_1(x_0)$ satisfying $p_1(x_0) \otimes_p p_0(x_0) = 1_p$ for each component $\{x_0\}$ created in the splitting process described above, and hard constraints representing δ_O; with this PFU network, the local normalization conditions are satisfied, and the combination of the local functions equals $\mathcal{P}_V \otimes_p \delta_O$;
- PFU query: the query is simply $Q = (\mathcal{N}, (\max, D).(\oplus_u, V - D))$.

An optimal decision rule for D can be recorded during the computation of $Ans(Q)$.

(b) (*Plausibility distribution computation task*) Given a plausibility distribution $\mathcal{P}_V$ expressed as a combination of plausibility functions as in chain graphs, the goal is to compute the plausibility distribution $\mathcal{P}_S$ over a set $S \subset V$. The basic formula $\mathcal{P}_S = \oplus_{pV-S} \mathcal{P}_V$ proves that the query defined below actually computes $\mathcal{P}_S$. This query shows the usefulness of free variables.

- the plausibility structure is $(E_p, \oplus_p, \otimes_p)$, the utility structure is $(E_u, \otimes_u) = (E_p, \otimes_p)$, and the expected utility structure is $(E_p, E_u, \oplus_u, \otimes_{pu}) = (E_p, E_p, \oplus_p, \otimes_p)$;
- PFU network: $\mathcal{N} = (V, G, P, \emptyset, U)$, with $V_E = V - S$, $V_D = S$, and with the DAG G and the sets P, U obtained similarly as for the MAP case;
- PFU query: $Q = (\mathcal{N}, (\oplus_u, V - S))$

11. (*Hybrid networks [36]*)

A hybrid network is a triple (G, P, F), where G is a DAG on a set of variables V partitioned into R and D, P is a set of probability distributions expressing $P_{r\,|\,pa_G(r)}$ for all $r \in R$, and F is a set of functions $f_{pa_G(d)}$ for all $d \in D$ (variables in D are deterministic, in the sense that their value is completely determined by the assignment of their parents). The most general task on hybrid networks is the task of belief assessment conditioned on a formula ϕ in conjunctive normal form. It consists of computing the probability distribution of a variable x given a complex evidence ϕ (complex because it may involve several variables). Ignoring a normalizing constant, it requires to compute, for all assignments (x, a) of x, $\sum_{A \in dom(V-\{x\})\,|\,\phi(A.(x,a))=t} P_V(A.(x, a))$. If $C = \{C_1, \ldots, C_m\}$ denotes the set of clauses of ϕ, it also equals $(\sum_{V-\{x\}} (\prod_{r\in R} P_{r\,|\,pa_G(r)}) \times (\prod_{d\in D} f_{pa_G(d)}) \times (\prod_{C_i\in C} C_i))((x, a))$.

The query corresponding to this computation uses the probabilistic expected satisfaction structure (row 2 in Table 3.1), and the PFU network $\mathcal{N} = (V, G, P, \emptyset, U)$, with $V_E = V$, $V_D = \{x\}$, $P = \{P_{r\,|\,pa_G(r)} \,|\, r \in R - \{x\}\} \cup \{f_{pa_G(d)} \,|\, d \in D - \{x\}\}$, and either $U = C \cup \{P_{x\,|\,pa_G(x)}\}$ if $x \in R$ or $U = C \cup \{f_{pa_G(x)}\}$ if $x \in D$. The query is $Q = (\mathcal{N}, (+, V - \{x\}))$.

□

B.4 Proofs of Chapter 6

Proof of Proposition 6.1 (page 92). First, for all $f_1, f_2 \in \{t, f\}$ and for all $u \in E_u$, $(f_1 \wedge f_2) \star u = f_1 \star (f_2 \star u)$: indeed, if $f_1 = f$ or $f_2 = f$, then $(f_1 \wedge f_2) \star u$ and $f_1 \star (f_2 \star u)$ both equal $\Diamond$, and otherwise ($f_1 = t$ and $f_2 = t$), they both equal u. This enables us to write $op_x(F \star P \otimes_{pu} U) = op_x((F^{-x} \wedge F^{+x}) \star P \otimes_{pu} U) = op_x(F^{-x} \star (F^{+x} \star P \otimes_{pu} U))$. Then, $op_x(F \star P \otimes_{pu} U) = F^{-x} \star op_x(F^{+x} \star P \otimes_{pu} U)$, because for all assignment A of $V - \{x\}$,

- If $F^{-x}(A) = f$, then, $F^{-x}(A) \star (op_x(F^{+x} \star P \otimes_{pu} U))(A) = \Diamond$. Moreover, for all $a \in dom(x)$, $(F^{-x} \star (F^{+x} \star P \otimes_{pu} U))(A.(x, a)) = \Diamond$, which implies that $(op_x(F^{-x} \star (F^{+x} \star P \otimes_{pu} U)))(A) = \Diamond$ too.
- Otherwise, $F^{-x}(A) = t$. In this case, $(op_x(F^{-x} \star (F^{+x} \star P \otimes_{pu} U)))(A) = (op_x(F^{+x} \star P \otimes_{pu} U))(A) = F^{-x}(A) \star (op_x(F^{+x} \star P \otimes_{pu} U))(A)$.

Next, $op_x(F^{+x} \star P \otimes_{pu} U) = P^{-x} \otimes_{pu} op_x(F^{+x} \star P^{+x} \otimes_{pu} U)$, because for all $A \in dom(V - \{x\})$,

- If $F^{+x}(A.(x, a)) = f$ for all $a \in dom(x)$, then $(op_x(F^{+x} \star P \otimes_{pu} U))(A) = \Diamond = (P^{-x} \otimes_{pu} (op_x(F^{+x} \star P^{+x} \otimes_{pu} U)))(A)$.
- Otherwise, one can write

$$\begin{aligned}
& op_x(F^{+x} \star P \otimes_{pu} U)(A) \\
= \;& op_{a \in dom(x), F^{+x}(A.(x,a)) \neq f}(P \otimes_{pu} U)(A.(x, a)) \\
= \;& op_{a \in dom(x), F^{+x}(A.(x,a)) \neq f}(P^{-x}(A) \otimes_{pu} (P^{+x}(A.(x, a)) \otimes_{pu} U(A.(x, a)))) \\
= \;& P^{-x}(A) \otimes_{pu} op_{a \in dom(x), F^{+x}(A.(x,a)) \neq f)}(P^{+x}(A.(x, a)) \otimes_{pu} U(A.(x, a))) \\
& \text{by right monotonicity of } \otimes_{pu} \text{ for } op \in \{\min, \max\} \text{ and by distributivity} \\
& \text{of } \otimes_{pu} \text{ over } \oplus_u \text{ when } op = \oplus_u \\
= \;& (P^{-x} \otimes_{pu} op_x(F^{+x} \star P^{+x} \otimes_{pu} U))(A)
\end{aligned}$$

In the end, this proves that $op_x(F \star P \otimes_{pu} U) = F^{-x} \star P^{-x} \otimes_{pu} op_x(F^{+x} \star P^{+x} \otimes_{pu} U)$. Moreover, if $P^{+x} = \emptyset$ and $op \in \{\min, \max\}$, then, for all assignment A of $V - \{x\}$,

- If $F^{+x}(A.(x,a)) = f$ for all $a \in dom(x)$, then
$$(op_x(F^{+x} \star U))(A) = \Diamond = (U^{-x} \otimes_u (op_x\,(F^{+x} \star U^{+x})))(A).$$

- Otherwise,
$$\begin{array}{ll} & op_x(F^{+x} \star U)(A) \\ = & op_{a\in dom(x), F^{+x}(A.(x,a))\neq f}\, U(A.(x,a)) \\ = & op_{a\in dom(x), F^{+x}(A.(x,a))\neq f}(U^{-x}(A) \otimes_u U^{+x}(A.(x,a))) \\ = & U^{-x}(A) \otimes_u op_{a\in dom(x), F^{+x}(A.(x,a))\neq f}\, U^{+x}(A.(x,a)) \text{ (by monotonicity of } \otimes_u) \\ = & (U^{-x} \otimes_u (op_x\,(F^{+x} \star U^{+x})))(A) \end{array}$$

□

Proof of Proposition 6.2 (page 92). A decision variable x appears in the scope of a plausibility function P_i iff x is a parent of one environment component having P_i as a factor. If a rightmost eliminated variable x is in V_D, then no plausibility function can involve x in its scope: otherwise, x should be a parent of an environment component, and the variables of this component should then appear at the right of x in *Sov* by definition of queries. The case $x \in V_E$ is proved similarly. □

Proof of Proposition 6.3 (page 93). $(E_u, \oplus_u)$ and $(E_u, \otimes_u)$ are commutative monoids by definition of an utility structure and of an expected utility structure. Then, for all $u \in E_u$, $0_u \otimes_u u = (0_p \otimes_{pu} 1_u) \otimes_u u = 0_p \otimes_{pu} (1_u \otimes_u u) = 0_u$ (the next to last equality holds because $p \otimes_{pu} (u_1 \otimes_u u_2) = (p \otimes_{pu} u_1) \otimes_u u_2$). Last, $\otimes_u$ distributes over $\oplus_u$. □

Proof of Proposition 6.4 (page 93). If Ax^{SR} holds, then:

$$\begin{array}{lll} \oplus_{ux}(P^{+x} \otimes_{pu} U) & = & \oplus_{ux}(P^{+x} \otimes_{pu} (U^{+x} \otimes_u U^{-x}) \\ & = & \oplus_{ux}((P^{+x} \otimes_{pu} U^{+x}) \otimes_u U^{-x}) \text{ (since } p \otimes_{pu} (u_1 \otimes_u u_2) = (p \otimes_{pu} u_1) \otimes_u u_2) \\ & = & (\oplus_{ux}(P^{+x} \otimes_{pu} U^{+x})) \otimes_u U^{-x} \text{ (since } \otimes_u \text{ distributes over } \oplus_u) \end{array}$$

If Ax^{SG} holds, then:

$$\begin{array}{lll} \oplus_{ux}(P^{+x} \otimes_{pu} U) & = & \oplus_{ux}(P^{+x} \otimes_{pu} (U^{-x} \otimes_u U^{+x})) \\ & = & \oplus_{ux}(P^{+x} \otimes_{pu} (U^{-x} \oplus_u U^{+x})) \\ & = & \oplus_{ux}((P^{+x} \otimes_{pu} U^{-x}) \oplus_u (P^{+x} \otimes_{pu} U^{+x})) \\ & = & (\oplus_{ux}(P^{+x} \otimes_{pu} U^{-x})) \oplus_u (\oplus_{ux}(P^{+x} \otimes_{pu} U^{+x})) \\ & = & ((\oplus_{p_x} P^{+x}) \otimes_{pu} U^{-x}) \oplus_u (\oplus_{ux}(P^{+x} \otimes_{pu} U^{+x})) \end{array}$$

□

Proof of Theorem 6.5 (page 94). Theorem 6.5(a) holds because if $Ax^{SR'}$ holds, then first, $\otimes_u$ distributed over $\oplus_u$ since $\otimes_p$ distributed over $\oplus_p$, and second, $p \otimes_{pu} (u_1 \otimes_u u_2) = p \otimes_{pu} (u_1 \otimes_{pu} u_2) = (p \otimes_{pu} u_1) \otimes_{pu} u_2 = (p \otimes_{pu} u_1) \otimes_u u_2$.

As for Theorem 6.5(b), let us assume that Ax^{SR} holds.

- Proposition 6.3 entails that $(E_u, \oplus_u, \otimes_u)$ is a commutative semiring. Moreover, $E = E_u$ is equipped with a total order $\preceq_u$. If $0_u \preceq_u 1_u$, let us take $\preceq = \preceq_u$ and if $1_u \preceq_u 0_u$, let us take $\preceq$ defined by $(u_1 \preceq u_2) \leftrightarrow (u_2 \preceq_u u_1)$. In all cases, $0_u \preceq 1_u$ holds. As $\otimes_u$ and $\oplus_u$ are monotonic with respect to $\preceq_u$, they are also monotonic with respect to $\preceq$. Using $0_u \preceq 1_u$, one can infer that, for all $u \in E_u$, $0_u \otimes u \preceq 1_u \otimes u$, i.e. $0_u \preceq u$ (we have $0_u \otimes u = 0_u$ since $(E_u, \oplus_u, \otimes_u)$ is a commutative semiring). This implies that $0_u = \min(E)$. Consequently, $(E, \oplus, \otimes)$ is a plausibility structure. Next, $(E, \otimes)$ is a utility structure because $(E_u, \otimes_u)$ is one. Last, $(E, E, \oplus, \otimes)$ is a totally ordered expected utility structure with $(E, \oplus, \otimes)$ as a plausibility structure and $(E, \otimes)$ as a utility structure, since it easily satisfies all properties of Definition 3.3 page 53: indeed, $(E, \oplus)$ is a commutative monoid, $\otimes$ distributes over $\oplus$, $e_1 \otimes (e_2 \otimes e_3) = (e_1 \otimes e_2) \otimes e_3$, $0_u \otimes e = 0_u$, and $1_u \otimes e = e$.

- Let $\mathcal{N} = (V, G, P, F, U)$ be a PFU network on S. Let $\mathcal{N}' = (V, G, \{\phi(P_i) \mid P_i \in P\}, F, U)$. In order to prove that $\mathcal{N}'$ is a PFU network on S', it suffices to prove that for every environment component c, $\oplus_c(\otimes_{P_i \in Fact(c)} \phi(P_i)) = 1_E$. This holds because on one hand, $\phi(1_p) = 1_p \otimes_{pu} 1_u = 1_u = 1_E$, and on the other hand, for every environment component c, $\phi(1_p) = \phi(\oplus_{p_c}(\otimes_{p\, P_i \in Fact(c)} P_i))) = \oplus_c(\otimes_{P_i \in Fact(c)} \phi(P_i))$. The last equality holds because $\phi(p_1 \oplus_p p_2) = (p_1 \oplus_p p_2) \otimes_{pu} 1_u = (p_1 \otimes_{pu} 1_u) \oplus_u (p_2 \otimes_{pu} 1_u) = \phi(p_1) \oplus_u \phi(p_2)$, and $\phi(p_1 \otimes_p p_2) = (p_1 \otimes_p p_2) \otimes_{pu} 1_u = p_1 \otimes_{pu} (p_2 \otimes_{pu} 1_u) = p_1 \otimes_{pu} \phi(p_2) = p_1 \otimes_{pu} (1_u \otimes_u \phi(p_2)) = (p_1 \otimes_{pu} 1_u) \otimes_u \phi(p_2) = \phi(p_1) \otimes_u \phi(p_2)$.

- Let $Q = (Sov, \mathcal{N})$ be a query on a PFU network $\mathcal{N}$ defined on S. Let $Q' = (Sov, \phi(\mathcal{N}))$. First, Q' is a query on $\phi(\mathcal{N})$ by definition of a query and because Q is a query. Then, as $p \otimes_{pu} u = p \otimes_{pu} (1_u \otimes_u u) = (p \otimes_{pu} 1_u) \otimes_u u = \phi(p) \otimes u$, and as $\phi(p_1 \otimes_p p_2) = \phi(p_1) \otimes \phi(p_2)$, one can write $(\wedge_{F_i \in F} F_i) \star (\otimes_{p\, P_i \in P} P_i) \otimes_{pu} (\otimes_{u\, U_i \in U} U_i) = (\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} \phi(P_i)) \otimes (\otimes_{U_i \in U} U_i)$. This implies that $Ans(Q) = Ans(Q')$ and that the set of optimal policies are the same with Q and Q'.

□

Proof of Proposition 6.7 (page 95). On one hand, if $(E_p, E_u, \oplus_u, \otimes_{pu})$ is a totally ordered expected utility structure satisfying $Ax^{SR'}$ (the underlying plausibility and utility structures being $(E_p, \oplus_p, \otimes_p)$ and $(E_u, \otimes_u)$), then $(E, \oplus, \otimes) = (E_u, \oplus_u, \otimes_u)$ is a commutative semiring by Proposition 6.3. It is equipped with a total order $\preceq_u$, and $\otimes$ and $\oplus$ are monotonic with respect to $\preceq_u$. Hence, $(E, \oplus, \otimes)$ is a totally ordered MCS. On the other hand, assume that $(E, \oplus, \otimes)$ is a totally ordered MCS. There is no difficulty in checking that all the properties of a plausibility structure are satisfied by $(E, \oplus, \otimes)$, that all the properties of a utility structure are satisfied by $(E, \otimes)$, and that all the properties of an expected utility structure are satisfied by $(E, E, \oplus, \otimes)$. □

Proof of Proposition 6.8 (page 95). First, $\oplus$ and $\otimes$ remain commutative and associative on $E \cup \{\Diamond\}$. $\oplus$ has $\Diamond$ as an identity and $\Diamond$ is an annihilator for $\otimes$. $\otimes$ has 1_E has an identity (notably using $1_E \otimes \Diamond = 1_E$). Last, $\otimes$ distributes over $\oplus$ on $E \cup \{\Diamond\}$, because first, $\otimes$ distributes over $\oplus$ on E, second, $u_1 \otimes (\Diamond \oplus u_3) = u_1 \otimes u_3 = (u_1 \otimes \Diamond) \oplus (u_1 \otimes u_3)$, and third, $\Diamond \otimes (u_2 \oplus u_3) = \Diamond = (\Diamond \otimes u_2) \oplus (\Diamond \otimes u_3)$. Therefore, $(E \cup \{\Diamond\}, \oplus, \otimes)$ is a commutative semiring.

Let us show that $(E \cup \{\Diamond\}, \max, \otimes)$ is a commutative semiring too. max is commutative and associative, and as max considered as an elimination operator satisfies $\max(u, \Diamond) = u$ for all $u \in E_u$, one can infer that $\Diamond$ is an identity for max. Last, $\otimes$ distributes over max, i.e. $u_1 \otimes \max(u_2, u_3) =$

$\max(u_1 \otimes u_2, u_1 \otimes u_3)$. Indeed, this holds if $(u_1, u_2, u_3) \in E^3$, because $\otimes$ is monotonic on E, this holds if u_2 or u_3 equals $\Diamond$, and this holds if $u_1 = \Diamond$. Therefore, $(E \cup \{\Diamond\}, \max, \otimes)$ is a commutative semiring. The proof for $(E \cup \{\Diamond\}, \min, \otimes)$ is similar. □

Proof of Corollary 6.9 (page 95). Entailed by Proposition 6.8. □

Proof of Proposition 6.10 (page 95). Entailed by Corollary 6.9. □

Proof of Proposition 6.12 (page 96). Let $\mathcal{N} = (V, G, P, F, U)$ be a PFU network. Assume that a component $c \in \mathcal{C}_E(G)$ is not connected. Let c_1 and c_2 be two disjoint subsets forming a partition of c such that there is no plausibility function in $Fact(c)$ involving both one variable in c_1 and one variable in c_2. This entails that the normalization condition on c can be written as

$$\begin{aligned}
&(\oplus_{p_{c_1}}(\otimes_{p_{P_i \in Fact(c), sc(P_i) \cap c_1 \neq \emptyset}} P_i))\\
&\otimes_p(\oplus_{p_{c_2}}(\otimes_{p_{P_i \in Fact(c), sc(P_i) \cap c_2 \neq \emptyset}} P_i))\\
&\otimes_p(\otimes_{p_{P_i \in Fact(c), sc(P_i) \subset pa_G(c)}} P_i)\\
&= 1_p
\end{aligned}$$

If one updates the DAG G of the PFU network $\mathcal{N}$ in order to get the DAG G' such that

- every component c' in G except from c is in G' too, and has parents such that $pa_{G'}(c') = pa_G(c)$;
- component c in G is replaced in G' by the two components c_1 and c_2, which both get $pa_G(c)$ as a set of parents. Moreover, we take
  - $Fact(c_1) = \{P_i \in Fact(P) \mid sc(P_i) \cap c_1 \neq \emptyset\} \cup \{\oplus_{p_{c_2}}(\otimes_{p_{P_i \in Fact(c), sc(P_i) \cap c_2 \neq \emptyset}} P_i)\} \cup \{P_i \in Fact(c), sc(P_i) \subset pa_G(c)\}$;
  - $Fact(c_2) = \{P_i \in Fact(P) \mid sc(P_i) \cap c_2 \neq \emptyset\} \cup \{\oplus_{p_{c_1}}(\otimes_{p_{P_i \in Fact(c), sc(P_i) \cap c_1 \neq \emptyset}} P_i)\} \cup \{P_i \in Fact(c), sc(P_i) \subset pa_G(c)\}$.

  With such settings, the normalization conditions on c_1 and c_2 are satisfied and (1) for every $P_i \in Fact(c_1)$, $sc(P_i) \subset c_1 \cup pa_G(c_1)$; (2) for every $P_i \in Fact(c_2)$, $sc(P_i) \subset c_2 \cup pa_G(c_2)$; (3) the global expressed plausibility function $\otimes_{p_{P_i \in P}} P_i$ does not vary.

This mechanism can be recursively applied until every environment component is connected.

The same "non-connected component splitting technique" can be used for decision component because the feasibility structure is a particular case of plausibility structure. However, in the case of decision components, the updating is easier, since if c_1 and c_2 are two disjoint subsets forming a partition of a decision component c such that there is no feasibility function in $Fact(c)$ involving both one variable in c_1 and one variable in c_2, then one can write

$$\left(\vee_{c_1}(\wedge_{F_i \in Fact(c), sc(F_i) \cap c_1 \neq \emptyset} F_i)\right) \wedge \left(\vee_{c_2}(\wedge_{F_i \in Fact(c), sc(F_i) \cap c_2 \neq \emptyset} F_i)\right) \wedge \left(\vee_{F_i \in Fact(c), sc(F_i) \subset pa_G(c)} F_i\right) = t$$

hence

$$\begin{cases} \vee_{c_1}(\wedge_{F_i \in Fact(c), sc(F_i) \cap c_1 \neq \emptyset} F_i) = t, \\ \vee_{c_2}(\wedge_{F_i \in Fact(c), sc(F_i) \cap c_2 \neq \emptyset} F_i)) = t, \\ \vee_{F_i \in Fact(c), sc(F_i) \subset pa_G(c)} F_i) = t \end{cases}$$

The modification of the PFU network finally simply looks like $Fact(c_1) = \{P_i \in Fact(P) \mid sc(P_i) \cap c_1 \neq \emptyset\}$ and $Fact(c_2) = \{P_i \in Fact(P) \mid sc(P_i) \cap c_2 \neq \emptyset\}$. Moreover, the feasibility functions $F_i \in Fact(c)$ such that $sc(F_i) \subset pa_G(c)$ can be removed. This does not modify the global feasibility degree. □

Proof of Proposition 6.13 (page 97). It is not hard to show that $\boxtimes$ and $\boxplus$ are commutative and associative. As $\oplus_u = \otimes_u$ holds, $0_u = 1_u$ holds too. This entails that for every plausibility functions P_1, P_2, $(P_1, 1_u) \boxtimes (P_2, 1_u) = (P_1 \otimes_p P_2, (P_1 \otimes_{pu} 1_u) \otimes_u (P_2 \otimes_{pu} 1_u)) = (P_1 \otimes_p P_2, (P_1 \otimes_{pu} 0_u) \otimes_u (P_2 \otimes_{pu} 0_u)) = (P_1 \otimes_p P_2, 0_u) = (P_1 \otimes_p P_2, 1_u)$. This implies that $\boxtimes_{P_i \in P}(P_i, 1_u) = (\otimes_{p P_i \in P} P_i, 1_u)$.

In another direction, for every utility functions U_1, U_2, one can write $(1_p, U_1) \boxtimes (1_p, U_2) = (1_p, U_1 \otimes_u U_2)$, which entails that $\boxtimes_{U_i \in U}(1_p, U_i) = (1_p, \otimes_{u U_i \in U} U_i)$.

Therefore, $(\boxtimes_{P_i \in P}(P_i, 1_u)) \boxtimes (\boxtimes_{U_i \in U}(1_p, U_i)) = (\otimes_{p P_i \in P} P_i, (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U}(1_p, U_i)))$ (namely using $0_u = 1_u$). This implies that $(\boxtimes_{F_i \in F} F_i) \boxtimes (\boxtimes_{P_i \in P}(P_i, 1_u)) \boxtimes (\boxtimes_{U_i \in U}(1_p, U_i)) = (\boxtimes_{F_i \in F} F_i) \boxtimes (\otimes_{p P_i \in P} P_i, (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i))$.

Let S be the rightmost set of decision variables in Sov, let $x \in S$, and let S' be the union of the sets of environment variables appearing at the right of S in Sov. We assume that x in quantified with max. The elimination of the environment variables in S' gives

$$\begin{aligned} &(\underset{F_i \in F}{\boxtimes} F_i) \boxtimes \boxplus_{S'}(\underset{P_i \in P}{\otimes_p} P_i, (\underset{P_i \in P}{\otimes_p} P_i) \otimes_{pu} (\underset{U_i \in U}{\otimes_u} U_i)) \\ &= (\underset{F_i \in F}{\boxtimes} F_i) \boxtimes (\underset{S'}{\oplus_p}(\underset{P_i \in P}{\otimes_p} P_i), \underset{S'}{\oplus_u}((\underset{P_i \in P}{\otimes_p} P_i) \otimes_{pu} (\underset{U_i \in U}{\otimes_u} U_i))) \end{aligned}$$

- If x is not the parent of any environment variable, then for all $P_i \in P$, $x \notin sc(P_i)$, and a fortiori $x \notin sc(\oplus_{p S'}(\otimes_{p P_i \in P} P_i))$.

  This implies that $\max_x \boxplus_{S'}((\boxtimes_{F_i \in F} F_i) \boxtimes (\otimes_{p P_i \in P} P_i, (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i)))$ is defined and it can be shown to equal $(\max_x (\boxtimes_{F_i \in F} F_i)) \boxtimes (\oplus_{p S'}(\otimes_{p P_i \in P} P_i), \max_x \oplus_{u S'}((\wedge_{F_i \in F} F_i) \star (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i)))$. The $\max_x (\boxtimes_{F_i \in F} F_i)$ factor is a trick ensuring that if no assignment of x is feasible, then the answer is $\Diamond$ and not $(1_p, \Diamond)$.

- Otherwise, x is the parent of at least one environment component c. c is included in S' by definition of a query. Hence $\oplus_{p S'}(\otimes_{p P_i \in P} P_i) = \oplus_{p S' - (c \cup desc(c)} \oplus_{p c \cup desc(c)}(\otimes_{p P_i \in P} P_i) = \oplus_{p S' - (c \cup desc(c)}(\otimes_{p P_i \in P, P_i \notin \cup_{c' \subset c \cup desc(c)} Fact(c')} P_i)$ (by recursively using the normalization conditions on c and its descendants $desc(c)$).

  Doing so, every environment component whose x is a parent can be considered, in order to obtain $\oplus_{p S'}(\otimes_{p P_i \in P} P_i) = \oplus_{p S' - \cup_{x \in pa_G(c)}(c \cup desc(c))}(\otimes_{p P_i \in P, P_i \notin \cup_{x \in pa_G(c)} \cup_{c' \subset c \cup desc(c)} Fact(c')} P_i)$. As the only environment components c having plausibility functions involving x can be the ones such that $x \in pa_G(c)$, we obtain that for all $a, a' \in dom(x)$, $(\oplus_{p S'}(\otimes_{p P_i \in P} P_i))((x, a)) = (\oplus_{p S'}(\otimes_{p P_i \in P} P_i))((x, a'))$,

  This implies that $\max_x \boxplus_{S'}(\boxtimes_{F_i \in F} F_i) \boxtimes (\otimes_{p P_i \in P} P_i, (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i))$ is defined and equals $(\max_x \boxtimes_{F_i \in F} F_i) \boxtimes (\oplus_{p S'}(\otimes_{p P_i \in P} P_i), \max_x \oplus_{u S'}(\wedge_{F_i \in F} F_i) \star (\otimes_{p P_i \in P} P_i) \otimes_{pu}$

$(\otimes_{u U_i \in U} U_i)))$.

This mechanism can be applied recursively when eliminating variables in the order given by Sov. In the end, we get $(\max_{V_D - V_{fr}} \boxtimes_{F_i \in F} F_i) \boxtimes (\oplus_{p V_E} (\otimes_{p P_i \in P} P_i), Ans(Q))$, i.e. a function ψ such that

- $\psi(A) = (1_p, Ans(Q)(A))$ if $Ans(Q)(A) \neq \Diamond$, because $Ans(Q)(A) \neq \Diamond$ implies that there exists an assignment A' of $V - V_{fr}$ s.t. $\wedge_{F_i \in F} F_i(A.A') = t$ and therefore $\max_{V_D - V_{fr}}(\boxtimes_{F_i \in F} F_i) = 1_{\boxtimes} = (1_p, 1_u)$.

- $\psi(A) = \Diamond$ if $Ans(Q)(A) = \Diamond$, because $Ans(Q)(A) = \Diamond$ implies that for all assignments A' of $V - V_{fr}$, $\wedge_{F_i \in F} F_i(A.A') = f$ and therefore $\max_{V_D - V_{fr}}(\boxtimes_{F_i \in F} F_i) = \Diamond$.

□

Proof of Lemma 6.14 (page 97). Let us first show that $\boxtimes$ distributes over $\boxplus$ on $(E_p \times E_u) \cup \{\Diamond\}$. Let (p_1, u_1), (p_2, u_2), and (p_3, u_3) be elements of $E_p \times E_u$. Then,

$$\begin{aligned}
&(p_1, u_1) \boxtimes ((p_2, u_2) \boxplus (p_3, u_3)) \\
&= (p_1, u_1) \boxtimes (p_2 \oplus_p p_3, u_2 \oplus_u u_3) \\
&= (p_1 \otimes_p (p_2 \oplus_p p_3), (p_1 \otimes_{pu} (u_2 \oplus_u u_3)) \otimes_u ((p_2 \oplus_p p_3) \otimes_{pu} u_1)) \\
&= ((p_1 \otimes_p p_2) \oplus_p (p_1 \otimes_p p_3), (p_1 \otimes_{pu} u_2) \oplus_u (p_1 \otimes_{pu} u_3) \oplus_u (p_2 \otimes_{pu} u_1) \oplus_u (p_3 \otimes_{pu} u_1)) \\
&= (p_1 \otimes_p p_2, (p_1 \otimes_{pu} u_2) \oplus_u (p_2 \otimes_{pu} u_1)) \boxplus (p_1 \otimes_p p_3, (p_1 \otimes_{pu} u_3) \oplus_u (p_3 \otimes_{pu} u_1)) \\
&= ((p_1, u_1) \boxtimes (p_2, u_2)) \boxplus ((p_1, u_1) \boxtimes (p_3, u_3))
\end{aligned}$$

Next, $(p_1, u_1) \boxtimes (\Diamond \boxplus (p_3, u_3)) = (p_1, u_1) \boxtimes (p_3, u_3) = ((p_1, u_1) \boxtimes \Diamond) \boxplus ((p_1, u_1) \boxtimes (p_3, u_3))$ and $\Diamond \boxtimes ((p_2, u_2) \boxplus (p_3, u_3)) = \Diamond = (\Diamond \boxtimes (p_2, u_2)) \boxplus (\Diamond \boxtimes (p_3, u_3))$. All these results prove that $\boxtimes$ distributes over $\boxplus$ on $(E_p \times E_u) \cup \{\Diamond\}$. Hence for every set of potentials Π, $\boxplus_x(\Pi) = \Pi^{-x} \boxtimes \boxplus_x(\Pi^{+x})$.

Let us show that $\boxtimes$ also satisfies a kind of restricted distributivity over max.

$$\begin{aligned}
&\max((p_1, u_1) \otimes (p, u_2), (p_1, u_1) \otimes (p, u_3)) \\
&= \max((p_1 \otimes_p p, (p_1 \otimes_{pu} u_2) \otimes_u (p \otimes_{pu} u_1)), (p_1 \otimes_p p, (p_1 \otimes_{pu} u_3) \otimes_u (p \otimes_{pu} u_1))) \\
&= (p_1 \otimes_p p, \max((p_1 \otimes_{pu} u_2) \otimes_u (p \otimes_{pu} u_1), (p_1 \otimes_{pu} u_3) \otimes_u (p \otimes_{pu} u_1))) \\
&= (p_1 \otimes_p p, \max(p_1 \otimes_{pu} u_2, p_1 \otimes_{pu} u_3) \otimes_u (p \otimes_{pu} u_1)) \\
&= (p_1 \otimes_p p, (p_1 \otimes_{pu} \max(u_2, u_3)) \otimes_u (p \otimes_{pu} u_1)) \\
&= (p_1, u_1) \boxtimes (p, \max(u_2, u_3))
\end{aligned}$$

Moreover, when max is used as an elimination operator, $\max((p_1, u_1) \boxtimes \Diamond, (p_1, u_1) \boxtimes (p, u_3)) = (p_1, u_1) \otimes (p, u_3) = (p_1, u_1) \boxtimes \max(\Diamond, (p, u_3))$, and $\max(\Diamond \boxtimes (p, u_2), \Diamond \boxtimes (p, u_3)) = \Diamond = \Diamond \boxtimes (p, \max(u_2, u_3))$. All these results prove that $\boxtimes$ distributes over max when the plausibility part does not vary, and therefore if $x \notin sc(P_0)$ for all $(P_0, U_0) \in \Pi$, then $\max_x(\Pi)$ exists and $\max_x(\Pi) = \Pi^{-x} \boxtimes \min_x(\Pi^{+x})$. The proof for min is similar. □

Proof of Proposition 6.15 (page 97). Let us show that at each step i, property (H_i) is satisfied:

$$(H_i) : \text{“}\boxtimes_{\pi \in \Pi_{i+1}} \pi \text{ is defined and equals } op(x_i)_{x_i} \ldots op(x_1)_{x_1}(\boxtimes_{\pi \in \Pi_1} \pi)\text{”}.$$

If H_i holds for all $i \in \{0, |Sov|\}$, then, using Proposition 6.13, we directly obtain the required result.

First, it is straightforward that H_0 holds. Let $k = \max\{j \in \{0, \ldots, |Sov|\} \mid \{x_1, \ldots, x_j\} \subset V_E\}$. According to Lemma 6.14, H_i also holds for all $i \in \{1, \ldots, k\}$.

If $k = |Sov|$, then the result is obtained. Otherwise, $k < |Sov|$. According to Lemma 6.14 again, H_{k+1} holds iff $op(x_{k+1})_{x_{k+1}} \Pi_{k+1}^{+x_{k+1}}$ is defined. By definition of queries, all environment components in $desc(c(x))$ are included in $\{x_1, \ldots, x_k\}$. As we work on refined PFU networks, there is exactly one potential in Π_{k+1} whose plausibility part can be written $\oplus_{p\,S}(\otimes_{p\,P_i \in \Phi} P_i)$, with $desc(c(x)) \subset S$ and $\cup_{c' \subset desc(c(x))}\{Fact(c')\} \subset \Phi$. Therefore, by using the normalization conditions, the plausibility part can also be written $\oplus_{p\,S-desc(c(x))}(\otimes_{p\,P_i \in \Phi - \cup_{c' \subset desc(c(x))}\{Fact(c')\}} P_i)$. This entails that the plausibility part does not depend on x. A similar reasoning can be made for the other decision variables, which proves that H_i holds for all $i \in \{1, \ldots, |Sov|\}$. □

Proof of Proposition 6.16 (page 98). Directly entailed by Proposition 6.15. □

Proof of Proposition 6.18 (page 99). First, $\otimes_u^+$ is an operator on E_u^+, since if $u_1, u_2 \in E_u^+$, then $u_1 \otimes_u^+ u_2 = u_1 \otimes_u u_2 = u_1 \oplus_u u_2 \succeq 0_u \oplus_u 0_u = 0_u$. Similarly, $\oplus_u^+ = \otimes_u^+$ is closed on E_u^+, and if $(p, u) \in E_p \times E_u^+$, then $p \otimes_{pu}^+ u = p \otimes_{pu} u \succeq p \otimes_{pu} 0_u = 0_u$ by right monotonicity of $\otimes_{pu}$.

As $\otimes_u = \oplus_u$ is associative, commutative, and monotonic, $\otimes_u^+$ and $\oplus_u^+$ are associative, commutative, and monotonic too. Moreover, as $0_u = 1_u \in E_u^+$, $\otimes_u^+$ and $\oplus_u^+$ both have an identity in E_u^+. It is not hard to check that all the axioms of expected utility structures are satisfied by $(E_p, E_u^+, \oplus_u^+, \otimes_{pu}^+)$.

The proof for $(E_p, E_u^-, \oplus_u^-, \otimes_{pu}^-)$ is similar. □

Proof of Proposition 6.19 (page 99). We prove only the first item, when (H^+) holds, since the proof for (H^-) is similar.

$\mathcal{N}^+$ is a PFU network because the transformation from $\mathcal{N}$ to $\mathcal{N}^+$ only changes the value taken by utility functions. It is also straightforward that Q^+ is a query. Last,

$$\begin{aligned}
Ans(Q) &= Sov(F \star P \otimes_{pu} (\otimes_{u\,U_i \in U} U_i)) \\
&= Sov(F \star P \otimes_{pu} (\otimes_{u\,U_i \in U} \text{translate}(U_i) \otimes_u U_i^-)) \\
&= Sov((F \star P \otimes_{pu} (\otimes_{u\,U_i \in U} \text{translate}^+(U_i))) \otimes_u (F \star P \otimes_{pu} (\otimes_{u\,U_i \in U} U_i^-)) \\
&= Sov((F \star P \otimes_{pu} (\otimes_{u\,U_i \in U} \text{translate}^+(U_i))) \otimes_u (\otimes_{u\,U_i \in U} U_i^-) \\
& \text{(thanks to the normalization conditions)} \\
&= Ans(Q^+) \otimes_u (\otimes_{u\,U_i \in U} U_i^-)
\end{aligned}$$

The formula obtained also shows that the set of optimal policies for Q^+ is included in the set of optimal policies for Q. □

Proof of Proposition 6.20 (page 100). It is first straightforward that Proposition 6.20a) is satisfied. Assume now that S satisfies Ax^{SG} and that all conditions of Proposition 6.20b) hold. Then,

- $\phi(1_p)$ is an identity for $\otimes$ since for all $u \in E$, $\phi(1_p) \otimes u = 1_p \otimes_{pu} u = u$. Also, for all $u \in E$, $\phi(0_p) \otimes u = 0_p \otimes u = 0_u$, hence we have first $\phi(0_p) = \phi(0_p) \otimes 1_E = 0_u$, and second $0_u \otimes u = 0_u$, hence $\otimes$ has $0_u = \phi(0_p)$ as a neutral element.

  Given the other conditions required on $\otimes$ and given that $\oplus = \oplus_u$, $(E, \oplus, \otimes)$ is a monotonic commutative semiring. Moreover, given that the expected utility structure is non bipolar, we can assume $0_u = \min(E_u)$, i.e. $0_E = \min(E)$ (either it is already satisfied, or we can inverse $\preceq_u$). This proves that $(E, \oplus, \otimes)$ is a plausibility structure. All conditions are satisfied for $(E, \oplus)$ to be a utility structure, and last all conditions are satisfied for $(E, E, \oplus, \otimes)$ to be an expected utility structure satisfying Ax^{SG}.

- In order to show that $\mathcal{N}'$ is a PFU network, it suffices to show that for every environment component c, $\oplus_c(\otimes_{P_i \in Fact(c)} \phi(P_i)) = 1_E$. This holds because $\phi(p_1 \oplus_p p_2) = \phi(p_1) \oplus \phi(p_2)$ and $\phi(p_1 \otimes_p p_2) = \phi(p_1) \otimes \phi(p_2)$ for all $p_1, p_2 \in E_p$, and because $\phi(1_p) = 1_E$.
- $Ans(Q) = Ans(Q')$ simply because $p \otimes_{pu} u = \phi(p) \otimes u$. The set of optimal policies does not vary since the global combined plausibility-feasibility-utility function does not vary either.

□

Proof of Proposition 6.21 (page 100). If $(E, \oplus, \otimes)$ is a plausibility structure, then $(E, \oplus, \otimes)$ is a MCS. Conversely, if $(E, \oplus, \otimes)$ is a MCS, then it is not hard to check that $(E, \oplus)$ is a utility structure, that $(E, \oplus, \otimes)$ is a plausibility structure, and that $(E, E, \oplus, \otimes)$ is an expected utility structure (it suffices to check each axiom successively). □

Proof of Proposition 6.30 (page 104). Let o^* be an elimination order such that $w_{\mathcal{G}}(\preceq_{Sov}) = w_{\mathcal{G}}(o^*)$. Let us eliminate variables in the order given by o^*. When a variable x is eliminated, $nbv \leq 1 + w_{\mathcal{G}}(o^*)$ variables are considered. For each of the d^{nbv} assignments of these variables, one must combine the value given by r scoped functions. In the end, the time complexity of a variable elimination step is $O(d^{nbv} \cdot r) \leq O(d^{1+w_{\mathcal{G}}(o)} \cdot r)$. Summing on all the elimination steps gives a time complexity $O(|\Phi| \cdot d^{1+w_{\mathcal{G}}(o)})$. Similarly, the space complexity is $O(|\Phi| \cdot d^{1+w_{\mathcal{G}}(o)})$ too. □

Proof of Proposition 6.31 (page 105). If $\preceq_2$ is weaker than $\preceq_1$, then $lin(\preceq_1) \subset lin(\preceq_2)$ and therefore

$$\min_{o \in lin(\preceq_2)} w_{\mathcal{G}}(o) \leq \min_{o \in lin(\preceq_1)} w_{\mathcal{G}}(o).$$

□

Proof of Proposition 6.32 (page 107). If $\circledast = \odot$, then $\circledast_x (\varphi_1 \circledast \varphi_2) = (\circledast_x \varphi_1) \circledast (\circledast_x \varphi_2)$ by commutativity and associativity of $\circledast$.

Conversely, assume that for all scoped functions φ_1, φ_2, $\circledast_x (\varphi_1 \odot \varphi_2) = (\circledast_x \varphi_1) \odot (\circledast_x \varphi_2)$. The identity of $\circledast$ in E is denoted $1_\circledast$ and the identity of $\odot$ in E is denoted $1_\odot$. Let us consider a boolean variable x and two scoped functions φ_1, φ_2 of scope x, s.t. $\varphi_1((x,t)) = a$, $\varphi_1((x,f)) = \varphi_2((x,t)) = 1_\odot$, $\varphi_2((x,f)) = b$. Then, the initial assumption implies that $(a \odot 1_\odot) \circledast (1_\odot \odot b) = (a \circledast 1_\odot) \odot (1_\odot \circledast b)$, i.e. $a \circledast b = (a \circledast 1_\odot) \odot (1_\odot \circledast b)$. Taking $a = b = 1_\circledast$ gives $1_\circledast = 1_\odot$. Consequently, for all $a, b \in E$, $a \circledast b = (a \circledast 1_\odot) \odot (1_\odot \circledast b) = (a \circledast 1_\circledast) \odot (1_\circledast \circledast b) = a \odot b$, i.e. $\circledast = \odot$. □

Proof of Proposition 6.33 (page 107). Let φ_1, φ_2 be scoped functions such that $(\varphi_1(A) = \Diamond) \leftrightarrow (\varphi_2(A) = \Diamond)$. Let A be an assignment of $(sc(\varphi_1) \cup sc(\varphi_2)) - \{x\}$. If $\varphi_1(A.(x,a)) = \Diamond$ for all $a \in dom(x)$, then $\circledast_x (\varphi_1 \odot \varphi_2)(A) = \Diamond = \Diamond \odot \Diamond = (\circledast_x \varphi_1)(A) \odot (\circledast_x \varphi_2)(A)$. Otherwise, if $\circledast = \odot$, then

$$\begin{aligned}
\circledast_x (\varphi_1 \odot \varphi_2)(A) &= \circledast_{a \in dom(x), \varphi_1(A.(x,a)) \neq \Diamond} (\varphi_1 \odot \varphi_2)(A) \\
&= (\circledast_{a \in dom(x), \varphi_1(A.(x,a)) \neq \Diamond} \varphi_1(A)) \odot (\circledast_{a \in dom(x), \varphi_1(A.(x,a)) \neq \Diamond} \varphi_2(A)) \\
&= (\circledast_{a \in dom(x), \varphi_1(A.(x,a)) \neq \Diamond} \varphi_1(A)) \odot (\circledast_{a \in dom(x), \varphi_2(A.(x,a)) \neq \Diamond} \varphi_2(A)) \\
&= (\circledast_x \varphi_1(A)) \odot (\circledast_x \varphi_2(A))
\end{aligned}$$

□

Proof of Proposition 6.34 (page 107). For all $A \in dom(S_1 \cup \ldots \cup S_m)$, computing

$$\circledast_x (\phi_{x,S_1}(A) \circledast \ldots \circledast \phi_{x,S_m}(A))$$

requires $|dom(x)|(m-1)$ operations to compute the function $\phi_{x,S_1}(A) \circledast \ldots \circledast \phi_{x,S_m}(A)$ ($(m-1)$ operations for each assignment of x) and $(|dom(x)|-1)$ operations to perform $\circledast_x$. Therefore, the raw computation of $\circledast_x\,(\phi_{x,S_1} \circledast \ldots \circledast \phi_{x,S_m})$ requires

$$n_1 = |dom(S_1 \cup \ldots \cup S_m)|(m|dom(x)|-1) \text{ operations.}$$

For each assignment $A \in dom(S_i)$, the raw computation of $\circledast_x\,\phi_{x,S_i}(A)$ requires $|dom(x)|-1$ operations. It entails that the raw computation of $\phi_i = \circledast_x\,\phi_{x,S_i}$ requires $|dom(S_i)|(|dom(x)|-1)$ operations and that the raw computation of m quantities in the set $\{\circledast_x\,\phi_{x,S_i} \mid 1 \leq i \leq m\}$ requires

$$n_2 = \sum_{1 \leq i \leq m} |dom(S_i)|(|dom(x)|-1) \text{ operations.}$$

Then, for each assignment $A \in dom(S_1 \cup \ldots \cup S_m)$, the raw computation of $\phi_1(A) \circledast \ldots \circledast \phi_m(A)$ requires $(m-1)$ operations. In the end, the raw computation of $(\circledast_x\,\phi_{x,S_1}) \circledast \ldots \circledast (\circledast_x\,\phi_{x,S_m})$ requires

$$n_3 = n_2 + |dom(S_1 \cup \ldots \cup S_m)|(m-1) \text{ operations.}$$

$n_1 - n_3$ equals $(|dom(x)|-1)(m|dom(S_1 \cup \ldots \cup S_m)| - \sum_{1 \leq i \leq m} |dom(S_i)|$, which is always positive. Consequently, the raw computation of $\circledast_x(\phi_{x,S_1} \circledast \ldots \circledast \phi_{x,S_m})$ always requires more operations than the raw computation of $(\circledast_x\,\phi_{x,S_1}) \circledast \ldots \circledast (\sum_x \phi_{x,S_m})$.

Furthermore, $n_1 = O(m \cdot d^{1+|S_1 \cup \ldots \cup S_m|})$ and $n_2 = O(m \cdot d^{1+\max_{i \in [1,m]} |S_i|})$. □

B.5 Proofs of Chapter 7

Proof of Proposition 7.5 (page 114). SR cannot be applied an infinite number of times because each computation node involves a finite number of variables.

If n uses an operator different from $\oplus$, then SR cannot be applied on n, hence $n_1 = n_2 = n$. Otherwise, n equals $(\oplus_S, N)$. Let $c_i^{(1)}$ be the i-th component eliminated in order to get n_1 and let $c_j^{(2)}$ be the j-th component eliminated in order to get n_2. Let $nc^{(1)}$ be the number of components eliminated from n to n_1 and let $nc^{(2)}$ be the number of components eliminated from n to n_2. Let us prove by recurrence that if $0 \leq k \leq nc^{(1)}$, then for all $i \in [1,k]$, there exists $j \in [1, nc^{(2)}]$ which satisfies $c_i^{(1)} = c_j^{(2)}$:

- The property obviously holds for $k = 0$.

- Assume that the property holds for $k < nc^{(1)}$. Is it satisfied at step $k+1$?

  Due to the recurrence hypothesis, there exists a step $jmax \in [1, nc^{(2)}]$ such that
  $$\{c_1^{(1)}, \ldots, c_k^{(1)}\} \subset \{c_1^{(2)}, \ldots, c_{jmax}^{(2)}\}$$

  – If $c_{k+1}^{(1)} \in \{c_1^{(2)}, \ldots, c_{jmax}^{(2)}\}$, then the property holds at step $k+1$.

  – Otherwise, $c_{k+1}^{(1)} \notin \{c_1^{(2)}, \ldots, c_{jmax}^{(2)}\}$. Assume that $c_{k+1}^{(1)} \notin \{c_{jmax}^{(2)}, \ldots, c_{nc^{(2)}}^{(2)}\}$. Then, as $c_{k+1}^{(1)}$ has been removed from n_1, one can infer that $Fact(c_{k+1}^{(1)}) \subset N$, $c_{k+1}^{(1)} \subset S$, $c_{k+1}^{(1)} \in \mathcal{C}_E(G)$, and $c_{k+1}^{(1)} \cap sc(N - \cup_{1 \leq l \leq k} Fact(c_l^{(1)})) = \emptyset$. This implies that $c_{k+1}^{(1)} \cap sc(N - \cup_{1 \leq l \leq nc^{(2)}} Fact(c_l^{(2)})) = \emptyset$, which leads to a contradiction with the fact that SR cannot be applied anymore on n_2. Hence, $c_{k+1}^{(1)} \in \{c_{jmax}^{(2)}, \ldots, c_{nc^{(2)}}^{(2)}\}$.

  In both cases, there exists a step $j \in [1, nc^{(2)}]$ such that $c_{k+1}^{(1)} = c_j^{(2)}$. Therefore, the property holds at step $k+1$.

For $k = nc^{(1)}$, this implies that for all $i \in [1, nc^{(1)}]$, there exists $j \in [1, nc^{(2)}]$ satisfying $c_i^{(1)} = c_j^{(2)}$. In other words, $\{c_1^{(1)}, \ldots, c_{nc^{(1)}}^{(1)}\} \subset \{c_1^{(2)}, \ldots, c_{nc^{(2)}}^{(2)}\}$. Similarly, it is possible to prove that $\{c_1^{(2)}, \ldots, c_{nc^{(2)}}^{(2)}\} \subset \{c_1^{(1)}, \ldots, c_{nc^{(1)}}^{(1)}\}$, hence $\{c_1^{(2)}, \ldots, c_{nc^{(2)}}^{(2)}\} = \{c_1^{(1)}, \ldots, c_{nc^{(1)}}^{(1)}\}$. As the same set of components are removed from n to n_1 and from n to n_2, one can infer that $n_1 = n_2$. □

Proof of Lemma 7.7 (page 116). In the following, we denote $(N^{-x})^{-y}$ by N^{-x-y}, $(N^{-x})^{+y}$ by N^{-x+y}, $(N^{+x})^{-y}$ by N^{+x-y}, $(N^{+x})^{+y}$ by N^{+x+y}, and $N^{-x+y} \cup N^{+x-y} \cup N^{+x+y}$ by $N^{+\{x,y\}}$.

Assume first that $op \neq \otimes$. Then, $rewrite(CNT) = (sov \cdot op_x, N^{-y} \cup \{n\})$, where $n = (op_{\{y\} \cup V_e(N^{+y}[op])}, N^{+y}[\neg op] \cup Sons(N^{+y}[op]))$.

- If $N^{+x+y} = \emptyset$, then $N^{+y} = N^{-x+y}$ and $x \notin sc(n)$. In this case, the expression obtained after the second rewriting step is $rewrite^2(CNT) = (sov, N^{-x-y} \cup \{n, n'\})$, with
$$\begin{cases} n = (op_{\{y\} \cup V_e(N^{-x+y}[op])}, N^{-x+y}[\neg op] \cup Sons(N^{-x+y}[op])) \\ n' = (op_{\{x\} \cup V_e(N^{+x-y}[op])}, N^{+x-y}[\neg op] \cup Sons(N^{+x-y}[op])) \end{cases}$$
The expression obtained for $rewrite^2(CNT)$ being symmetric in x/y, one can infer that $rewrite^2(CNT) = rewrite^2(CNT')$.

- Otherwise, $N^{+x+y} \neq \emptyset$. In this case, $x \in sc(n)$, and $rewrite^2(CNT) = (sov, N^{-x-y} \cup \{n'\})$, where $n' = (op_{\{x\} \cup V_e(N^{+x-y}[op]) \cup V_e(n)}, N^{+x-y}[\neg op] \cup Sons(N^{+x-y}[op]) \cup Sons(n))$.

  Given that first,
$$\begin{aligned} &\{x\} \cup V_e(N^{+x-y}[op]) \cup V_e(n) \\ &= \{x\} \cup V_e(N^{+x-y}[op]) \cup \{y\} \cup V_e(N^{+y}[op]) \\ &= \{x, y\} \cup V_e(N^{+x-y}[op] \cup N^{+y}[op]) \\ &= \{x, y\} \cup V_e(N^{+\{x,y\}}[op]) \end{aligned}$$
  and second,
$$\begin{aligned} &N^{+x-y}[\neg op] \cup Sons(N^{+x-y}[op]) \cup Sons(n) \\ &= N^{+x-y}[\neg op] \cup Sons(N^{+x-y}[op]) \cup N^{+y}[\neg op] \cup Sons(N^{+y}[op]) \\ &= N^{+\{x,y\}}[\neg op] \cup Sons(N^{+x-y}[op] \cup N^{+y}[op]) \\ &= N^{+\{x,y\}}[\neg op] \cup Sons(N^{+\{x,y\}}[op]) \end{aligned},$$
  the expression of n' is symmetric in x/y. This implies that $rewrite^2(CNT) = rewrite^2(CNT')$.

In the case $op = \otimes$, $rewrite(CNT) = (sov \cdot op_x, N^{-y} \cup \{(op_{\{y\} \cup V_e(n)}, Sons(n)), n \in N^{+y}[op]\} \cup \{(op_{\{y\}}, \{n\}), n \in N^{+y}[\neg op]\})$.

The second rewriting step gives:
$$rewrite^2(CNT) = \left(sov, \begin{array}{c} N^{-x-y} \\ \cup\{(op_{\{x\} \cup V_e(n)}, Sons(n)), n \in N^{+x-y}[op]\} \\ \cup\{(op_{\{x\}}, \{n\}), n \in N^{+x-y}[\neg op]\} \\ \cup\{(op_{\{x,y\} \cup V_e(n)}, Sons(n)), n \in N^{+x+y}[op]\} \\ \cup\{(op_{\{x,y\}}, \{n\}), n \in N^{+x+y}[\neg op]\} \\ \cup\{(op_{\{y\} \cup V_e(n)}, Sons(n)), n \in N^{-x+y}[op]\} \\ \cup\{(op_{\{y\}}, \{n\}), n \in N^{-x+y}[\neg op]\} \end{array} \right).$$
As this expression is symmetric in x/y, one can infer that $rewrite^2(CNT) = rewrite^2(CNT')$.
□

Proof of Lemma 7.8 (page 116). Let o, o' be two elimination orders on a set of variables S (without constraints on the elimination order). Then, one can obtain o' by successive permutations of eliminations in o. Indeed, this obviously holds if $|S| = 0$. Assume that the property holds for any elimination order on a set of variables of cardinal k. Does it hold at step $k+1$? Let o, o' be two elimination orders on S with $|S| = k+1$. Let x be the first variable eliminated in o' ($x = o'(1)$). By successive permutations, o can be transformed into an elimination order $t(o)$ such that $t(o)$ and o' eliminate the same first variable. Then, the recurrence assumption allows us to transform, by successive permutations, the elimination order $t(o)$ restricted over $S - \{x\}$ into o' restricted over $S - \{x\}$ Therefore, the property holds for $|S| = k+1$, hence the proof by recurrence.

Assume that $Sov = (op_1, S_1) \cdot (op_2, S_2) \cdots (op_q, S_q)$. Let o, o' be two elimination orders in $lin(\preceq_{Sov})$. o can be transformed into o' by using the previous recurrence for each set of variable S_i. □

Proof of Theorem 7.9 (page 116). Lemma 7.8 allows us to recursively apply Lemma 7.7 and to obtain $CNT(Q, o) = CNT(Q, o')$ (also by using the fact the simplification rule is applied at the end of each block of variables eliminated using the same operator, hence the step where SR^* is applied does not vary between o and o', which are both in $lin(\preceq_{Sov})$). □

Proof of Lemma 7.10 (page 116). Because of the MCS structure of $(E, \oplus, \otimes)$, $\otimes$ distributes over every $op \in \{\min, \max, \oplus\}$. Then,

$$\begin{aligned} val((sov \cdot op_x, N)) &= sov \cdot op_x\,(\otimes_{n \in N} val(n)) \\ &= sov((\otimes_{n \in N^{-x}} val(n)) \otimes op_x(\otimes_{n \in N^{+x}} val(n))) \end{aligned} \qquad \text{(eq1)}$$

- If $op = \otimes$, Proposition 6.32 implies that
$$\begin{aligned} op_x\,(\otimes_{n \in N^{+x}} val(n)) &= \otimes_{n \in N^{+x}}\,(op_x\, val(n)) \\ &= val(\{(op_x, \{n\}) \,|\, n \in N^{+x}\}) \end{aligned}$$
Therefore, using (eq1),
$$val\,((sov \cdot op_x, N)) = val\,((sov, N^{-x} \cup \{(op_x, n) \,|\, n \in N^{+x}\})).$$

- Otherwise ($op \neq \otimes$), one can just write
$$op_x\,(\otimes_{n \in N^{+x}} val(n)) = val\,((op_x, N^{+x}))$$
This means that (eq1) can be written as
$$val\,((sov \cdot op_x, N)) = val\,((sov, N^{-x} \cup \{(op_x, N^{+x})\}))$$

□

Proof of Lemma 7.11 (page 116). Given that $\otimes$ distributes over op and $S' \cap sc(N_1) = \emptyset$, one can write

$$\begin{aligned} val((op_S, N_1 \cup \{(op_{S'}, N_2)\})) &= op_S\,((\otimes_{n \in N_1} val(n)) \otimes op_{S'}\,(\otimes_{n \in N_2} val(n))) \\ &= op_S \cdot op_{S'}\,((\otimes_{n \in N_1} val(n)) \otimes (\otimes_{n \in N_2} val(n))) \end{aligned}$$

As $N_1 \cap N_2 = \emptyset$ and $S \cap S' = \emptyset$, the latter quantity also equals $op_{S \cup S'}\,(\otimes_{n \in N_1 \cup N_2} val(n))$, i.e. $val((op_{S \cup S'}, N_1 \cup N_2))$. □

Proof of Lemma 7.12 (page 116). It suffices to recursively apply Lemma 7.11 to get the required result. □

Proof of Lemma 7.13 (page 117). The property holds for $k = 0$ since $CNT_0(Q, o) = (Sov, P \cup U)$ and $V_e(n) = \emptyset$ for all $n \in P \cup U$. If it holds at step k, then it holds at $k+1$ because if the elimination operator used is different from $\otimes$, then DR splits the nodes with x in their scopes and those without x in their scopes. Moreover, it is straightforward that variables whose elimination has not been considered yet (variables in $V_e(CNT_k(Q,o))$) are not eliminated in an internal node of the tree of computation nodes, i.e. for all (sov, N) in $CNT_k(Q,o)$, $V_e(CNT_k(Q,o)) \cap V_e(N) = \emptyset$. □

Proof of Lemma 7.14 (page 117). Assume that $c \in \mathcal{C}_E(G)$ and $c \cap (S \cup sc(N)) = \emptyset$. Then,

$$\begin{aligned}
val((\oplus_{S\cup c}, N \cup Fact(c))) &= \oplus_{S\cup c}\big((\otimes_{n\in N} val(n)) \otimes \big(\otimes_{\varphi\in Fact(c)}\varphi\big)\big)\\
&= \oplus_S\left(\oplus_c\big((\otimes_{n\in N} val(n)) \otimes \big(\otimes_{\varphi\in Fact(c)}\varphi\big)\big)\right) \text{ (since } c\cap S=\emptyset)\\
&= \oplus_S\big((\otimes_{n\in N} val(n)) \otimes \left(\oplus_c\left(\otimes_{\varphi\in Fact(c)}\varphi\right)\right)\big) \text{ (since } c\cap sc(N)=\emptyset)\\
&= \oplus_S((\otimes_{n\in N} val(n)) \otimes 1_E)\\
&= val((\oplus_S, N))
\end{aligned}$$

□

Proof of Lemma 7.15 (page 117). Let $k \in \{0,\dots,|Sov|-1\}$. $CNT_{k+1}(Q,o)$ is obtained from $CNT_k(Q,o)$ by using rewriting rules DR, RR, and SR only.

Thanks to Lemma 7.13 and the fact that all computation nodes are distinct, the hypotheses of Lemma 7.12 hold when RR is applied.

As DR and SR are sound too (cf Lemmas 7.10 and 7.14),

$$val(CNT_{k+1}(Q,o)) = val(CNT_k(Q,o))$$

□

Proof of Theorem 7.16 (page 117). Follows from Lemma 7.15 and from $val(CNT_0(Q,o)) = Ans(Q)$ for all $o \in lin(\preceq_{Sov})$. □

Proof of Proposition 7.17 (page 118). At each rewriting step and for each son n' of the root node, tests like "$x \in sc(n')$" and operations like "$sc(n) \leftarrow sc(n) \cup sc(n')$" or "$sc(n') \leftarrow sc(n') - \{x\}$" are $O(|V|)$, since a scope is represented as a table of size $|V|$. Operations like "$Sons(root) \leftarrow Sons(root) - \{n'\}$", "$Sons(root) \leftarrow Sons(root) \cup \{n\}$", "$V_e(n) \leftarrow V_e(n) \cup V_e(n')$" (with $V_e(n) \cap V_e(n') = \emptyset$), or "$V_e(n) \leftarrow V_e(n) \cup \{x\}$" are $O(1)$, since V_e and $Sons$ are represented as lists. Therefore, the operations performed for each rewriting step and for each son of the root are $O(|V|)$. As at each step, $|Sons(root)| \leq |P \cup U|$, and as there are $|V|$ rewriting steps, the algorithm is time $O(|V|^2 \cdot |P \cup U|)$.

As for the space complexity, given that only the scopes of the root sons are used, we need a space $O(|V| \cdot |P \cup U|)$ for the scopes. As it can be shown that the number of nodes in the tree of computation nodes is always $O(|V| + |P \cup U|)$, recording $op(n)$ and $Sons(n)$ for all nodes n is $O(|V| + |P \cup U|)$ too. Last, recording $V_e(n)$ for all nodes n is $O(|V| \cdot |P \cup U|)$ because the sum of the number of variables eliminated in all nodes is lesser than $|V| \cdot |P \cup U|$ (the worst case occurs when all variables are duplicated). Hence, the overall space complexity is $O(|V| \cdot |P \cup U|)$. □

Proof of Proposition 7.20 (page 120). The result obviously holds if the cluster-tree decomposition contains one cluster c_0, since in this case, $V(c_0) = V$, $\Phi(c_0) = \Phi$, and $Sons(c_0) = \emptyset$.

Assume that the property holds if there are k clusters in the cluster-tree decomposition. Let us consider a cluster-tree decomposition of a graphical model (V, Φ) given S, such that this decomposition contains $k+1$ clusters. Let c be a leaf cluster in this tree-decomposition. Then, for all $\varphi \notin \Phi(c)$, $sc(\varphi) \cap (V(c) - V(pa(c)) = \emptyset$. Indeed, if $\varphi \notin \Phi(c)$, then there exists a cluster c' such that $\varphi \in \Phi(c')$, and hence $sc(\varphi) \subset V(c')$. The running intersection property allows us to infer that $\forall x \in V(c) - V(pa(c)), x \notin V(c')$ (otherwise, as $pa(c)$ is necessarily on the path from c to c', we should have $(V(c) - V(pa(c))) \cap V(pa(c)) \neq \emptyset$). This entails that $V(c') \cap (V(c) - V(pa(c)) = \emptyset$, and therefore $sc(\varphi)) \cap (V(c) - V(pa(c)) = \emptyset$.

For all $\varphi \notin \Phi(c)$, $sc(\varphi) \cap (V(c) - V(pa(c)) = \emptyset$, one can write:

$$\begin{aligned}\oplus_{V-S}(\otimes_{\varphi\in\Phi}\varphi) &= \oplus_{(V-S)-(V(c)-V(pa(c)))}\oplus_{V(c)-V(pa(c))}(\otimes_{\varphi\in\Phi}\varphi)\\ &= \oplus_{(V-S)-(V(c)-V(pa(c)))}((\otimes_{\varphi\notin\Phi(c)}\varphi)\otimes(\oplus_{V(c)-V(pa(c))}(\otimes_{\varphi\in\Phi(c)}\varphi)))\\ &= \oplus_{(V-S)-(V(c)-V(pa(c)))}((\otimes_{\varphi\notin\Phi(c)}\varphi)\otimes val(c))\end{aligned}$$

The result is then obtained by using the recurrence hypothesis on the graphical model $(V - (V(c) - V(pa(c))), (\Phi - \Phi(c)) \cup \{val(c)\})$. □

Proof of Theorem 7.24 (page 122). Entailed by the soundness of the macrostructuration process (Theorem 7.16 page 117), and by Proposition 7.20 (page 120) concerning cluster-tree decompositions. As for the policies, the macrostructuration process guarantees the result concerning policies, because if $x \notin sc(\varphi_0)$, then

$$\operatorname{argmax}_x\varphi \subset \operatorname{argmax}_x(\varphi_0 \otimes \varphi)$$

(and such a form is the only decomposition used for non-duplicated decision variables). As for duplicated decision variables, we know for example that if max $= \otimes$ and φ_1, φ_2 are two scoped functions, then $((\operatorname{argmax}_x\varphi_1) \cup (\operatorname{argmax}_x\varphi_2)) \cap \operatorname{argmax}_x(\varphi_1 \otimes \varphi_2) \neq \emptyset$. Indeed, let $A \in dom(sc(\varphi_1 \otimes \varphi_2) - \{x\})$. Let $a_1 \in \operatorname{argmax}_x\varphi_1(A)$ and $a_2 \in \operatorname{argmax}_x\varphi_2(A)$. Then, for all $a \in dom(x)$, $\varphi_1(a_1.A) \succeq \varphi_1(a.A)$ and $\varphi_2(a_2.A) \succeq \varphi_2(a.A)$. Therefore, for all $a \in dom(x)$, $\max(\varphi_1(a_1.A), \varphi_2(a_2.A)) \succeq \max(\varphi_1(a.A), \varphi_2(a.A))$, which implies that either $(\varphi_1 \otimes \varphi_2)(a_1.A) \succeq (\varphi_1 \otimes \varphi_2)(a.A)$, or $(\varphi_1 \otimes \varphi_2)(a_2.A) \succeq (\varphi_1 \otimes \varphi_2)(a.A)$. As a result, $a_1 \in \operatorname{argmax}_x(\varphi_1 \otimes \varphi_2)(A)$ or $a_2 \in \operatorname{argmax}_x(\varphi_1 \otimes \varphi_2)(A)$. □

Proof of Proposition 7.26 (page 123). Let c be a cluster of a MCTree. According to the definition of $val(c)$, computing the value of c given the values of its sons is time $O(d^{1+w_{CNT(Q)}}) \cdot (|\Phi(c)| + |Sons(c)| - 1))$. Hence, computing the value of all clusters $c \in C$ of a MCTree is time $O(d^{1+w_{CNT(Q)}}) \cdot \sum_{c\in C}(|\Phi(c)| + |Sons(c)| - 1))$.

It can be shown that $\sum_{c\in C}(|\Phi(c)| + |Sons(c)| - 1)) \leq 2 \cdot |P \cup U|$:

- First, $\sum_{c\in C} |\Phi(c)| \leq |P \cup U|$.

- Second, given a tree having nl leaves, it can easily be shown (by recurrence) that the sum of the number of sons of each node minus 1 equals $nl - 1$. Therefore, as the MCTree has at most $|P \cup U|$ leaves, one can infer that $\sum_{c\in C}(|Sons(c)| - 1) \leq |P \cup U| - 1 \leq |P \cup U|$,

Therefore, the time complexity is $O(2 \cdot |P \cup U| \cdot d^{1+w_{CNT(Q)}}) = O(|P \cup U| \cdot d^{1+w_{CNT(Q)}})$.

The space complexity is also $O(|P \cup U| \cdot d^{1+w_{CNT(Q)}})$ because the functions which must be manipulated always have a scope of size lesser than $1 + w_{CNT(Q)}$. □

Proof of Theorem 7.27 (page 123). Let o^* be an elimination order s.t. $w_{\mathcal{G}}(\preceq_{Sov}) = w_{\mathcal{G}}(o^*)$. The idea is to apply the rewriting rules on $CNT_0(Q, o^*)$. Let $\mathcal{G}_0 = \mathcal{G}$ and, if $\mathcal{G}_k = (V_k, H_k)$ and $x = o^*(k)$ is eliminated, then $\mathcal{G}_{k+1} = (V_k - \{x\}, (H_k - H_k^{+x}) \cup \{h_{k+1}\})$, where $h_{k+1} = \cup_{h\in H_k^{+x}} h - \{x\}$ is the hyperedge created from step k to $k + 1$. It can be proved that for all $k \in \{0, \ldots, |Sov| - 1\}$, if $CNT_k(Q, o^*) = (sov \cdot op_x, N)$, then for all $n \in N$, there exists $h \in H_k$ s.t. $sc(n) \subset sc(h)$. Indeed, this property easily holds at step 0, and if it holds at step k, then $sc((op_x, N^{+x})) \subset sc(h_{k+1})$. Moreover, if duplication is used, then for all $n \in N^{+x}$, $sc((op_x, \{n\})) \subset sc(h_{k+1})$. Rewriting rules RR and SR can be shown to be always advantageous in terms of induced-width. This entails the required result. □

Proof of Lemma 7.29 (page 129). Let us start from $CNDAG_k(Q, o)$.

Case $op = \oplus$ As the elimination at the left of $\oplus_y$ is a $\oplus$-elimination too, we have

$$\begin{array}{rcl} CNDAG_{k+1}(Q,o) & = & (sov\cdot\oplus_x, \oplus, \{rewrite((\oplus_y, \otimes, N)), N\in\mathfrak{N}\} \\ & = & (sov\cdot\oplus_x, \oplus, \{(\emptyset,\otimes, N^{-y}\cup\{RR((\oplus_y,\otimes,N^{+y}))\}), N\in\mathfrak{N}\}) \end{array}.$$

If the elimination at the left of $\oplus_x$ is a $\oplus$ elimination, we get:

$$CNDAG_{k+2}(Q,o) = (sov, \oplus, \{rewrite((\oplus_x, \otimes, N^{-y}\cup\{RR((\oplus_y,\otimes,N^{+y}))\})), N\in\mathfrak{N}\}).$$

As $(\oplus_x, \otimes, N^{-y}\cup\{RR((\oplus_y,\otimes,N^{+y}))\}) = rewrite((\oplus_x\oplus_y,\otimes,N))$, one can write:

$$CNDAG_{k+2}(Q,o) = (sov, \oplus, \{rewrite^2((\oplus_x\oplus_y,\otimes,N)), N\in\mathfrak{N}\}).$$

Similarly,

$$CNDAG_{k+2}(Q,o') = (sov, \oplus, \{rewrite^2((\oplus_y\oplus_x,\otimes,N)), N\in\mathfrak{N}\}).$$

Lemma 7.7 enables us to conclude that $CNDAG_{k+2}(Q,o) = CNDAG_{k+2}(Q,o')$. If the elimination at the left of $\oplus_x$ is not a $\oplus$ elimination, then we get:

$$CNDAG_{k+2}(Q,o) = (sov, \oplus, \{simplify(rewrite^2((\oplus_x\oplus_y,\otimes,N))), N\in\mathfrak{N}\}).$$

and similarly, we have $CNDAG_{k+2}(Q,o) = CNDAG_{k+2}(Q,o')$.

Case $op = \max$ **(when** $\max \neq \oplus$**)** In this case,

$$CNDAG_k(Q,o) = (sov\cdot\max_x\cdot\max_y, \oplus, \{(\emptyset,\otimes,N), N\in\mathfrak{N}\})$$

First, if $\mathfrak{N}^{+x+y} = \emptyset$, then the result obtained for $CNDAG_{k+2}(Q,o)$ is symmetric in x/y. Indeed, when $\max_y$ is considered, structural modifications are made on the part depending on y only, i.e. on the nodes associated with $\mathfrak{N}^{+y} = \mathfrak{N}^{-x+y}$, and when $\max_x$ is considered, structural modifications are made on the part depending on x only, i.e. on the nodes associated with $\mathfrak{N}^{+x-y}$. This implies that $CNDAG_{k+2}(Q,o) = CNDAG_{k+2}(Q,o')$.

Otherwise, we have $\mathfrak{N}^{+x+y} \neq \emptyset$. The application of $DR_{\max}$ on $CNDAG_k(Q,o)$ gives

$$(sov.\max_x, \oplus, \{(\emptyset,\otimes,N), N\in\mathfrak{N}^{-y}\}\cup\{(\emptyset,\otimes,N_1\cup\{(\max_y,\oplus,N_2)\})\})$$

where $N_1 = \cap_{N\in\mathfrak{N}^{+y}} N^{-y}$ and $N_2 = \{(\emptyset,\otimes,N-N_1), N\in\mathfrak{N}^{+y}\}$.

N_1 does not involve any max node, because when $\max \neq \oplus$, the definition of $DR_{\max}$ and $RR_{\max}$ implies that variables eliminated with max appear exactly once in the structure. Therefore, $(N-N_1)[\max] = N[\max]$.

Using this result, the application of $RR_{\max}$ transforms $(\max_y, \oplus, N_2)$ into: $(\max_{S_a}, \oplus, N_a)$, where

$$\left\{\begin{array}{rcl} S_a & = & \{y\}\cup V_e(\cup_{N\in\mathfrak{N}^{+y}}(N-N_1)[\max]) \\ N_a & = & \{(\emptyset,\otimes,N-N_1), (N\in\mathfrak{N}^{+y})\wedge(N[\max]=\emptyset)\} \\ & & \cup\{(\emptyset,\otimes,((N-N_1)-N[\max])\cup N'), \\ & & \qquad (N\in\mathfrak{N}^{+y})\wedge(N[\max]\neq\emptyset)\wedge((\emptyset,\otimes,N')\in Sons(N[\max]))\} \end{array}\right.$$

Therefore, we get

$$CNDAG_{k+1}(Q,o) = (sov.\max_x, \oplus, \{(\emptyset,\otimes,N), N\in\mathfrak{N}^{-y}\}\cup\{(\emptyset,\otimes,N_1\cup\{(\max_{S_a},\oplus,N_a)\})\})$$

After these steps, the elimination of x is considered. After the application of $DR_{\max}$, we obtain

$$(sov, \oplus, \{(\emptyset,\otimes,N), N\in\mathfrak{N}^{-x-y}\}\cup\{(\emptyset,\otimes,N'_1\cup\{(\max_x,\oplus,N'_2)\})\})$$

where

- $N'_1 = (\bigcap_{N\in\mathfrak{N}^{+x-y}} N^{-x}) \cap (N_1 \cup \{(\max_{S_a}, \oplus, N_a)\})^{-x}$. As y is eliminated exactly once in the structure, $(\bigcap_{N\in\mathfrak{N}^{+x-y}} N^{-x}) \cap \{(\max_{S_a}, \oplus, N_a)\} = \emptyset$. This allows us to write $N'_1 = (\bigcap_{N\in\mathfrak{N}^{+x-y}} N^{-x}) \cap N_1^{-x}$. Therefore,
$$\begin{aligned} N'_1 &= (\textstyle\bigcap_{N\in\mathfrak{N}^{+x-y}} N^{-x}) \cap (\bigcap_{N\in\mathfrak{N}^{+y}} N^{-y})^{-x} \quad . \\ &= (\textstyle\bigcap_{N\in\mathfrak{N}^{+x-y}} N^{-x-y}) \cap (\bigcap_{N\in\mathfrak{N}^{+y}} N^{-x-y}) \\ &= \textstyle\bigcap_{N\in\mathfrak{N}^{+\{x,y\}}} N^{-x-y} \end{aligned}$$
Hence, the expression of N'_1 is symmetric in x/y.

- $N'_2 = \{(\emptyset, \otimes, N - N'_1), N \in \mathfrak{N}^{+x-y}\} \cup \{(\emptyset, \otimes, (N_1 - N'_1) \cup \{(\max_{S_a}, \oplus, N_a)\})\}$.

After the application of $RR_{\max}$, $(\max_x, \oplus, N'_2)$ is transformed into $(\max_{S_b}, \oplus, N_b)$, where (we use the fact that $N_1[\max] = N'_1[\max] = \emptyset$):

- $S_b = \{x\} \cup V_e(\bigcup_{N\in\mathfrak{N}^{+x-y}} N[\max]) \cup \{y\} \cup V_e(\bigcup_{N\in\mathfrak{N}^{+y}} N[\max]) = \{x, y\} \cup V_e(\bigcup_{N\in\mathfrak{N}^{+\{x,y\}}} N[\max])$. This shows that the expression of S_b is symmetric in x/y.

- $$\begin{aligned} N_b = \; & \{(\emptyset, \otimes, N - N'_1), (N \in \mathfrak{N}^{+x-y}) \wedge (N[\max] = \emptyset)\} \\ & \cup \{(\emptyset, \otimes, ((N - N'_1) - N[\max]) \cup N'), \\ & \qquad (N \in \mathfrak{N}^{+x-y}) \wedge (N[\max] \neq \emptyset) \wedge ((\emptyset, \otimes, N') \in \mathit{Sons}(N[\max]))\} \\ & \cup \{(\emptyset, \otimes, (N_1 - N'_1) \cup (N - N_1)), (N \in \mathfrak{N}^{+y}) \wedge (N[\max] = \emptyset)\} \\ & \cup \{(\emptyset, \otimes, (N_1 - N'_1) \cup ((N - N_1) - N[\max]) \cup N'), \\ & \qquad (N \in \mathfrak{N}^{+y}) \wedge (N[\max] \neq \emptyset) \wedge ((\emptyset, \otimes, N') \in \mathit{Sons}(N[\max]))\} \\ = \; & \big\{(\emptyset, \otimes, N - N'_1), (N \in \mathfrak{N}^{+\{x,y\}}) \wedge (N[\max] = \emptyset)\big\} \\ & \cup \{(\emptyset, \otimes, ((N - N'_1) - N[\max]) \cup N'), \\ & \qquad (N \in \mathfrak{N}^{+\{x,y\}}) \wedge (N[\max] \neq \emptyset) \wedge ((\emptyset, \otimes, N') \in \mathit{Sons}(N[\max]))\big\} \end{aligned}$$
This expression is symmetric in x/y.

As a result,
$$CNDAG_{k+2}(Q, o) = (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}^{-x-y}\} \cup \{(\emptyset, \otimes, N'_1 \cup \{(\max_{S_b}, \oplus, N_b)\})\})$$
As the expressions of N'_1, S_b, and N_b are symmetric in x/y. this entails that $CNDAG_{k+2}(Q, o) = CNDAG_{k+2}(Q, o')$.

Case $op = \min$ **(when** $\min \neq \oplus$**)** This case is dealt with exactly as the case $op = \max$. □

Proof of Theorem 7.30 (page 129). Lemma 7.8 established in the semiring case allows us to recursively apply Lemma 7.29 and to obtain $CNDAG(Q, o) = CNDAG(Q, o')$. □

Proof of Lemma 7.31 (page 129).

$$\begin{aligned} val((sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})) &= sov. \bigoplus_x \left(\bigoplus_{N\in\mathfrak{N}} \left(\bigotimes_{n\in N} val(n) \right) \right) \\ &= sov \left(\bigoplus_{N\in\mathfrak{N}} \left(\bigoplus_x \left(\bigotimes_{n\in N} val(n) \right) \right) \right) \\ &= val((sov, \oplus, \{(\emptyset, \otimes, \{(\oplus_x, \otimes, N)\}), N \in \mathfrak{N}\})) \end{aligned}$$

□

Proof of Lemma 7.32 (page 129). The property holds for $k = 0$, because

$$CNDAG_0(Q, o) = (Sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) \text{ with } \mathfrak{N} = \{P \cup \{U_i\}, U_i \in U\},$$

and therefore, (1) for all $N \in \mathfrak{N}$, for all $n \in N$, $V_e(n) = \emptyset$ and $Sons(n) = \emptyset$, and (2) for all $N \in \mathfrak{N}$, $N[\max] = \emptyset$.

Assume that the property holds for $k < |Sov|-1$ and that $CNDAG_k(Q, o) = (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$. This recurrence assumption is denoted (RA). Does the property hold at step $k + 1$?

Case $sov = sov'.\oplus_x$ After the application of $DR_\oplus$ and *rewrite*, we obtain $CNDAG_{k+1}(Q, o) = (sov', \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}'\})$, with $\mathfrak{N}' = \{N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}, N \in \mathfrak{N}\}$.

Let $N' \in \mathfrak{N}'$, i.e. $N' = N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}$ for some $N \in \mathfrak{N}$. Let $(n_1, n_2) \in N'^2$ such that $n_1 \neq n_2$.

- If $(n_1, n_2) \in (N^{-x})^2$, then $(n_1, n_2) \in N^2$: (RA) directly implies that $V_e(n_1) \cap V_e(n_2) = \emptyset$ and $V_e(n_1) \cap sc(n_2) = \emptyset$. Similarly, if $(n_1, n_2) \in (N^{-x}[\oplus])^2$, then $(n_1, n_2) \in (N[\oplus])^2$, hence (RA) implies that $Sons(n_1) \cap Sons(n_2) = \emptyset$.

- If $n_1 \in N^{-x}$ and $n_2 = RR((\oplus_x, \otimes, N^{+x}))$

  Then, as $V_e(n_2) \subset \{x\} \cup (\cup_{n \in N^{+x}} V_e(n))$, as $x \notin V_e(n_1)$ (because x had not been considered before step k), and as $V_e(n_1) \cap V_e(n) = \emptyset$ for every $n \in N^{+x}$ (thanks to (RA)), this entails that $V_e(n_1) \cap V_e(n_2) = \emptyset$.

  Similarly, as $sc(n_2) \subset \cup_{n \in N^{+x}} sc(n)$, (RA) enables us to infer that $V_e(n_1) \cap sc(n_2) = \emptyset$.

  Next, assume that $(n_1, n_2) \in N'[\oplus]$. This means that $n_1 \in N^{-x}[\oplus] \subset N[\oplus]$. We have $Sons(n_2) = N^{+x}[\neg\oplus] \cup (\cup_{n \in N^{+x}[\oplus]} Sons(n))$. According to (RA), we have, for all $n \in N^{+x}[\oplus]$, $Sons(n_1) \cap Sons(n) = \emptyset$ and $Sons(n_1) \cap N^{+x}[\neg\oplus] = \emptyset$ (since $Sons(n_1) \cap N[\neg\oplus] = \emptyset$). This enables us to infer that $Sons(n_1) \cap Sons(n_2) = \emptyset$.

- If $n_1 = RR((\oplus_x, \otimes, N^{+x}))$ and $n_2 \in N^{-x}$

  Then, it has already been shown (previous item) that $V_e(n_1) \cap V_e(n_2) = \emptyset$ and that if $(n_1, n_2) \in N'[\oplus]$, $(n_1 \neq n_2) \to (Sons(n_1) \cap Sons(n_2) = \emptyset)$.

  As $V_e(n_1) \subset \{x\} \cup (\cup_{n \in N^{+x}} V_e(n))$, as $x \notin sc(n_2)$ (because $n_2 \in N^{-x}$), and as $V_e(n) \cap sc(n_2) = \emptyset$ for every $n \in N^{+x}$ (due to the recurrence assumption), it is possible to infer that $V_e(n_1) \cap sc(n_2) = \emptyset$.

Let $n \in N'[\oplus]$. If $n \in N^{-x}[\oplus]$, then (RA) directly implies that $Sons(n) \cap N^{-x}[\neg\oplus] = \emptyset$, and therefore that $Sons(n) \cap N'[\neg\oplus] = \emptyset$. Otherwise, $n = RR((\oplus_x, \otimes, N^{+x}))$. In this case, $Sons(n) = N^{+x}[\neg\oplus] \cup (\cup_{n' \in N^{+x}[\oplus]} Sons(n'))$. First, $N^{+x}[\neg\oplus] \cap N^{-x}[\neg\oplus] = \emptyset$. Second, for every $n' \in N^{+x}[\oplus]$, $Sons(n') \cap N[\neg\oplus] = \emptyset$ thanks to (RA), and hence $Sons(n') \cap N^{-x}[\neg\oplus] = \emptyset$. Therefore, $Sons(n) \cap N^{-x}[\neg\oplus] = \emptyset$, i.e. $Sons(n) \cap N'[\neg\oplus] = \emptyset$.

As $N' = N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}$, we can write $N'[\max] = N^{-x}[\max] \subset N[\max]$, hence $|N'[\max]| \leq 1$. Let $(\emptyset, \otimes, N_s) \in Sons(N'[\max])$. Then, $(\emptyset, \otimes, N_s) \in Sons(N[\max])$. From this, the recurrence assumption entails that $N_s \cap N[\neg\max] = \emptyset$, and consequently that $N_s \cap N^{-x}[\neg\max] = \emptyset$. Moreover, it is straightforward that $RR((\oplus_x, \otimes, N^{+x})) \notin N_s$. Hence, $N_s \cap N'[\neg\max] = \emptyset$.

Let $(N'_1, N'_2) \in \mathfrak{N}'$ such that $N'_1 \neq N'_2$. This entails that $N'_1 = N_1^{-x} \cup \{RR((\oplus_x, \otimes, N_1^{+x}))\}$ and $N'_2 = N_2^{-x} \cup \{RR((\oplus_x, \otimes, N_2^{+x}))\}$ for some $(N_1, N_2) \in \mathfrak{N}^2$ such that $N_1 \neq N_2$. Then, $N'_1[\max] =$

$N_1^{-x}[\max] \subset N_1[\max]$ and $N_2'[\max] = N_2^{-x}[\max] \subset N_2[\max]$. The recurrence assumption then directly entails that $V_e(N_1'[\max]) \cap V_e(N_2'[\max]) = \emptyset$. Moreover, as $sc(N_2') = sc(N_2) - \{x\}$, this also entails that $V_e(N_1'[\max]) \cap sc(N_2') = \emptyset$.

All these results show that the property holds at step $k+1$. The property still holds if simplification rule SR is applied, since SR can only reduce the set of eliminated variables, the scopes of nodes, and the sets of sons.

Case $sov = sov'.\max_x$ **(when** $\max \neq \oplus$**)** If $\mathfrak{N}^{+x} = \emptyset$, then the structure is unchanged at step $k+1$, hence the property is still satisfied.

Otherwise, the application of $DR_{\max}$ and $RR_{\max}$ gives $(sov', \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}'\})$, with $\mathfrak{N}' = \mathfrak{N}^{-x} \cup \{N_a \cup \{RR_{\max}((\max_x, \oplus, N_b))\}\}$, where $N_a = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$ and $N_b = \{(\emptyset, \otimes, N - N_a), N \in \mathfrak{N}^{+x}\}$.

Let $N' \in \mathfrak{N}'$.

- Either $N' \in \mathfrak{N}^{-x}$. In this case, (RA) directly implies that for all $(n_1, n_2) \in N'$ such that $n_1 \neq n_2$, $V_e(n_1) \cap V_e(n_2) = \emptyset$ and $V_e(n_1) \cap sc(n_2) = \emptyset$, that for all $(n_1, n_2) \in N'[\oplus]$ such that $n_1 \neq n_2$, $Sons(n_1) \cap Sons(n_2) = \emptyset$, and that for all $n \in N'[\oplus]$, $Sons(n) \cap N[\neg\oplus] = \emptyset$.

- Or $N' = N_a \cup \{RR_{\max}(\max_x, \oplus, N_b)\}$.

  Let $(n_1, n_2) \in N'$ such that $n_1 \neq n_2$.

  – If $(n_1, n_2) \in N_a^2$, then there exists $N \in \mathfrak{N}^{+x}$ such that $(n_1, n_2) \in N^2$. In this case, the recurrence assumption directly implies that $V_e(n_1) \cap V_e(n_2) = \emptyset$ and $V_e(n_1) \cap sc(n_2) = \emptyset$, and that if $(n_1, n_2) \in N'[\oplus]$, then $Sons(n_1) \cap Sons(n_2) = \emptyset$.

  – If $n_1 \in N_a$ and $n_2 = RR_{\max}((\max_x, \oplus, N_b))$.

  We have $V_e(n_2) \subset \{x\} \cup (\cup_{N \in \mathfrak{N}^{+x}} V_e(N[\max]))$. Given that $x \notin V_e(n_1)$ and for all $N \in \mathfrak{N}^{+x}$, for all $n \in N[\max]$, $V_e(n) \cap V_e(n_1) = \emptyset$ (because $n_1 \in N$ and $n \neq n_1$), we obtain $V_e(n_1) \cap V_e(n_2) = \emptyset$.

  Next, $sc(n_2) \subset \cup_{n \in N_b} sc(n) = \cup_{N \in \mathfrak{N}^{+x}} sc(N - N_a)$. We know that for all $N \in \mathfrak{N}^{+x}$, for all $n \in N - N_a$, $n \neq n_1$ and consequently, as $n_1 \in N$, $V_e(n_1) \cap sc(n) = \emptyset$. This entails that $V_e(n_1) \cap sc(n_2) = \emptyset$.

  Moreover, $n_2 \notin N'[\oplus]$ (because $\max \neq \oplus$).

  – If $n_1 = RR((\max_x, \oplus, N_b))$ and $n_2 \in N_a$.

  It has already been shown (see previous item), that $V_e(n_1) \cap V_e(n_2) = \emptyset$. Moreover, $V_e(n_1) = \{x\} \cup (\cup_{N \in \mathfrak{N}^{+x}} V_e(N[\max]))$. We know that $x \notin sc(n_2)$ (since $n_2 \in N_a$), and, thanks to (RA), that for all $N \in \mathfrak{N}^{+x}$, if $N[\max] \neq \emptyset$, then $N[\max] = \{n_1\}$ and $V_e(N_1) \cap sc(n_2) = \emptyset$. Hence, $V_e(n_1) \cap sc(n_2) = \emptyset$.

Let $n \in N'[\oplus]$. We then have $n \in N_a[\oplus]$. Together with (RA), this implies that $Sons(n) \cap N_a[\neg\oplus] = \emptyset$. Moreover, it is straightforward that $RR_{\max}(\max_x, \oplus, N_b) \notin Sons(n)$. Therefore, $Sons(n) \cap N'[\neg\oplus] = \emptyset$.

Let $(N_1, N_2) \in (\mathfrak{N}')^2$ such that $N_1 \neq N_2$.

- If $(N_1, N_2) \in (\mathfrak{N}^{-x})^2$, then $(N_1, N_2) \in \mathfrak{N}^2$, and (RA) implies that $V_e(N_1[\max]) \cap V_e(N_2[\max]) = \emptyset$ and $V_e(N_1[\max]) \cap sc(N_2) = \emptyset$.

- If $N_1 \in \mathfrak{N}^{-x}$ and $N_2 = N_a \cup \{RR_{\max}((\max_x, \oplus, N_b))\}$

  We know that $V_e(N_2[\max]) = \{x\} \cup (\cup_{N \in \mathfrak{N}^{+x}} V_e(N[\max]))$. Due to (RA), one can write that for all $N \in \mathfrak{N}^{+x}$, $V_e(N_1[\max]) \cap V_e(N[\max]) = \emptyset$. Furthermore, $x \notin V_e(N_1[\max])$. This entails that $V_e(N_1[\max]) \cap V_e(N_2[\max]) = \emptyset$.

  Similarly, $sc(N_2) \subset \cup_{N \in \mathfrak{N}^{+x}} sc(N[\max])$. (RA) implies that for every $N \in \mathfrak{N}^{+x}$, $V_e(N_1[\max]) \cap sc(N) = \emptyset$. Hence, $V_e(N_1[\max] \cap sc(N_2) = \emptyset$.

- If $N_1 = N_a \cup \{RR_{\max}((\max_x, \oplus, N_b))\}$ and $N_2 \in \mathfrak{N}^{-x}$

  It has already been shown (previous item) that $V_e(N_1[\max]) \cap V_e(N_2[\max]) = \emptyset$.

  $V_e(N_1[\max]) = \{x\} \cup (\cup_{N \in \mathfrak{N}^{+x}} V_e(N[\max]))$. Due to (RA), one can write, for every $N \in \mathfrak{N}^{+x}$, $V_e(N[\max]) \cap sc(N_2) = \emptyset$. Moreover, $x \notin sc(N_2)$. Thus, $V_e(N_1[\max]) \cap sc(N_2) = \emptyset$.

Let $N' \in \mathcal{N}'$.

- If $N' \in \mathfrak{N}^{-x}$, then $N' \in \mathfrak{N}$ and consequently, thanks to (RA), $|N'[\max]| \leq 1$ and for all $(\emptyset, \otimes, N_s) \in Sons(N'[\max])$, $N_s \cap N'[\neg \max] = \emptyset$.

- Otherwise, $N' = N_a \cup \{RR_{\max}((\max_x, \oplus, N_b))\}$.

  In this case, $N'[\max] = \{RR_{\max}((\max_x, \oplus, N_b))\}$. This implies that $|N'[\max]| = 1$.

  Let $(\emptyset, \otimes, N_s) \in Sons(N'[\max])$, i.e. $(\emptyset, \otimes, N_s) \in Sons(RR_{\max}((\max_x, \oplus, N_b)))$.

  We know that $Sons(RR_{\max}((\max_x, \oplus, N_b)) = \{(\emptyset, \otimes, N - N_a), (N \in \mathfrak{N}^{+x}) \wedge (N[\max] = \emptyset)\} \cup \{(\emptyset, \otimes, (N[\neg \max] - N_a) \cup N''), (N \in \mathfrak{N}^{+x}) \wedge (N[\max] = \emptyset) \wedge (\emptyset, \otimes, N'') \in Sons(N[\max])\}$.

  We know that for every $N \in \mathfrak{N}^{+x}$, $(N - N_a) \cap N_a = \emptyset$. Therefore, if $(\emptyset, \otimes, N_s) \in \{(\emptyset, \otimes, N - N_a), (N \in \mathfrak{N}^{+x}) \wedge (N[\max] = \emptyset)\}$, then $N_s \cap N_a = \emptyset$, i.e. $N_s \cap N'[\neg \max] = \emptyset$.

  Otherwise, there exists $N \in \mathfrak{N}^{+x}$ such that $N[\max] \neq \emptyset$ and $(\emptyset, \otimes, N_s) = (\emptyset, \otimes, (N[\neg \max] - N_a) \cup N'')$ with $(\emptyset, \otimes, N'') \in Sons(N[\max])$. This means that $N_s = (N[\neg \max] - N_a) \cup N''$ with $(\emptyset, \otimes, N'') \in Sons(N[\max])$. Then, $N_s \cap N'[\neg \max] = N_s \cap N_a = ((N[\neg \max] - N_a) \cup N'') \cap N_a = ((N[\neg \max] - N_a) \cap N_a) \cup (N'' \cap N_a) = N'' \cap N_a$. We have $(\emptyset, \otimes, N'') \in Sons(N[\max])$ and $N_a \in N[\neg \max]$. (RA) enables us to infer that $N'' \cap N_a = \emptyset$, and consequently $N_s \cap N'[\neg \max] = \emptyset$.

All these results show that the property also holds at step $k+1$ if $sov = sov'.\max_x$.

Case $sov = sov'.\min_x$ **(when** $\min \neq \oplus$**)** Similar to the case $sov = sov'.\max_x$. □

Proof of Lemma 7.33 (page 130). The result follows from Lemmas 7.31 and 7.32.

Indeed, if simplification is not applied, then we have

$$CNDAG_{k+1}(Q, o) = (sov, \oplus, \{rewrite((\oplus_x, \otimes, N)), N \in \mathfrak{N}\})$$

As function *rewrite* gives a sound result (thanks to Proposition 7.10, Proposition 7.12, and Lemma 7.32), this implies that

$$\begin{aligned} val(CNDAG_{k+1}(Q, o)) &= val((sov, \oplus, \{rewrite((\oplus_x, \otimes, N)), N \in \mathfrak{N}\})) \\ &= val((sov, \oplus, \{(\oplus_x, \otimes, N), N \in \mathfrak{N}\}) \end{aligned}$$

The result is still valid if *simplify* is applied, because simplification rule SR is sound.

As $DR_\oplus$ is sound (thanks to Lemma 7.31), this also entails that $val(CNDAG_{k+1}(Q,o)) = val((sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\}) = val(CNDAG_k(Q,o))$

□

Proof of Lemma 7.34 (page 130).

$$val((sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})) = sov \max_x \left(\bigoplus_{N \in \mathfrak{N}} \left(\bigotimes_{n \in N} val(n) \right) \right)$$

If $\mathfrak{N}^{+x} = \emptyset$, then one can infer:

$$\begin{aligned} val((sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})) &= sov \left(\bigoplus_{N \in \mathfrak{N}} \left(\bigotimes_{n \in N} val(n) \right) \right) \\ &= val((sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})) \end{aligned}$$

Otherwise, $\mathfrak{N}^{+x} \neq \emptyset$. In this case, the monotonicity of $\oplus$ enables us to write:

$$\begin{aligned} & val((sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})) \\ = \; & sov \left(\left(\bigoplus_{N \in \mathfrak{N}^{-x}} \left(\bigotimes_{n \in N} val(n) \right) \right) \oplus \max_x \left(\bigoplus_{N \in \mathfrak{N}^{+x}} \left(\bigotimes_{n \in N} val(n) \right) \right) \right) \end{aligned}$$

Furthermore, if $N_1 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$ and $N_2 = \{(\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\}$, then

$$\begin{aligned} & \max_x \left(\bigoplus_{N \in \mathfrak{N}^{+x}} \left(\bigotimes_{n \in N} val(n) \right) \right) \\ = \; & \max_x \left(\bigoplus_{N \in \mathfrak{N}^{+x}} \left(\left(\bigotimes_{n \in N_1} val(n) \right) \otimes \left(\bigotimes_{n \in N - N_1} val(n) \right) \right) \right) \\ = \; & \max_x \left(\left(\bigotimes_{n \in N_1} val(n) \right) \otimes \bigoplus_{N \in \mathfrak{N}^{+x}} \left(\bigotimes_{n \in N - N_1} val(n) \right) \right) \\ = \; & \left(\bigotimes_{n \in N_1} val(n) \right) \otimes \max_x \left(\bigoplus_{N \in \mathfrak{N}^{+x}} \left(\bigotimes_{n \in N - N_1} val(n) \right) \right) \text{ (since } \otimes \text{ is monotonic and } x \notin sc(N_1)) \\ = \; & val((\emptyset, \otimes, N_1 \cup \{(\max_x, \oplus, N_2)\})) \end{aligned}$$

Consequently, if $\mathfrak{N}^{+x} \neq \emptyset$,

$$\begin{aligned} & val((sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})) \\ = \; & sov \left(\left(\bigoplus_{N \in \mathfrak{N}^{-x}} \left(\bigotimes_{n \in N} val(n) \right) \right) \oplus val((\emptyset, \otimes, N_1 \cup \{(\max_x, \oplus, N_2)\})) \right) \\ = \; & val\big((sov, \oplus, \big\{(\emptyset, \otimes, N), N \in \mathfrak{N}^{-x}\big\} \cup \{(\emptyset, \otimes, N_1 \cup \{(\max_x, \oplus, N_2)\})\}\big)\big) \end{aligned}$$

□

Proof of Lemma 7.35 (page 130). Assume that $S' \cap (S \cup sc(N_1) \cup sc(N_2)) = \emptyset$ and $\forall N_3 \in \mathfrak{N}$, $N_2 \cap$

$N_3 = \emptyset$. Then,

$$\begin{aligned}
& val((\max_S, \oplus, N_1 \cup \{(\emptyset, \otimes, N_2 \cup \{(\max_{S'}, \oplus, \{(\emptyset, \otimes, N_3), N_3 \in \mathfrak{N}\})\})\})) \\
= \;& \max_S \left(\left(\bigoplus_{n \in N_1} val(n) \right) \oplus \left(\bigotimes_{n' \in N_2} val(n') \right) \otimes \max_{S'} \left(\bigoplus_{N_3 \in \mathfrak{N}} \left(\bigotimes_{n'' \in N_3} val(n'') \right) \right) \right) \\
= \;& \max_S \left(\left(\bigoplus_{n \in N_1} val(n) \right) \oplus \max_{S'} \left(\left(\bigotimes_{n' \in N_2} val(n') \right) \otimes \bigoplus_{N_3 \in \mathfrak{N}} \left(\bigotimes_{n'' \in N_3} val(n'') \right) \right) \right) \\
& \text{(since } \otimes \text{ is monotonic and } S' \cap sc(N_2) = \emptyset) \\
= \;& \max_S \left(\left(\bigoplus_{n \in N_1} val(n) \right) \oplus \max_{S'} \bigoplus_{N_3 \in \mathfrak{N}} \left(\left(\bigotimes_{n' \in N_2} val(n') \right) \otimes \left(\bigotimes_{n'' \in N_3} val(n'') \right) \right) \right) \\
= \;& \max_S \left(\left(\bigoplus_{n \in N_1} val(n) \right) \oplus \max_{S'} \bigoplus_{N_3 \in \mathfrak{N}} \left(\bigotimes_{n' \in N_2 \cup N_3} val(n') \right) \right) \\
& \text{(since } \forall N_3 \in \mathfrak{N},\ N_2 \cap N_3 = \emptyset) \\
= \;& \max_S \max_{S'} \left(\left(\bigoplus_{n \in N_1} val(n) \right) \oplus \bigoplus_{N_3 \in \mathfrak{N}} \left(\bigotimes_{n' \in N_2 \cup N_3} val(n') \right) \right) \\
& \text{(since } \oplus \text{ is monotonic and } S' \cap sc(N_1) = \emptyset) \\
= \;& \max_{S \cup S'} \left(\left(\bigoplus_{n \in N_1} val(n) \right) \oplus \bigoplus_{N_3 \in \mathfrak{N}} \left(\bigotimes_{n' \in N_2 \cup N_3} val(n') \right) \right) \\
& \text{(since } S \cap S' = \emptyset) \\
= \;& val((\max_{S \cup S'}, \oplus, N_1 \cup \{(\emptyset, \otimes, N_2 \cup N_3), N_3 \in \mathfrak{N}\}))
\end{aligned}$$

□

Proof of Lemma 7.36 (page 130). If max $= \oplus$, then the result is implied by Lemma 7.33. Otherwise , max $\neq \oplus$. The result is straightforward if $\mathfrak{N}^{+x} = \emptyset$. Otherwise, $\mathfrak{N}^{+x} \neq \emptyset$.

According to Lemma 7.34 which states that $DR_{\max}$ is sound, one can write
$val(CNDAG_k(Q, o)) = val((sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}^{-x}\} \cup \{(\emptyset, \otimes, N_a \cup \{(\max_x, \oplus, N_b)\})\}))$
where $N_a = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$ and $N_b = \{(\emptyset, \otimes, N - N_a), N \in \mathfrak{N}^{+x}\}$

Let us denote by n the node $n = (\max_x, \oplus, N_b)$. In order to prove that $val(CNDAG_{k+1}(Q, o)) = val(CNDAG_k(Q, o))$, it suffices to prove that $val(n) = val(RR_{\max}(n))$.

Let us denote by $\mathfrak{N}_0$ the set of sets of nodes $\mathfrak{N}_0 = \{N - N_a, N \in \mathfrak{N}^{+x}\}$. n can then be written as $n = (\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}_0\})$.

Let $\mathfrak{N}_0 = \{N_1, \ldots, N_r\}$. Let us define, for all $i \in \{0, \ldots, r\}$,
$n_i = (\max_{\{x\} \cup V_e(\cup_{N \in \{N_1, \ldots, N_i\}} N[\max])}, \oplus,$.
$\{(\emptyset, \otimes, N), (N \in \{N_1, \ldots, N_i\}) \wedge (N[\max] = \emptyset)\}$
$\cup \{(\emptyset, \otimes, N[\neg\max] \cup N'), (N \in \{N_1, \ldots, N_i\}) \wedge (N[\max] \neq \emptyset) \wedge ((\emptyset, \otimes, N') \in Sons(N[\max]))\}$
$\cup \{(\emptyset, \otimes, N), N \in \{N_{i+1}, \ldots, N_r\}\})$
Let us show that for all $i \in \{0, \ldots, r\}$, $val(n) = val(n_i)$.

The property holds for $i = 0$, because $n_0 = n$. Assume that the property holds for $i < r$. Let us show that it holds at step $i + 1$.

If $N_{i+1}[\max] = \emptyset$, then the result is obvious because in this case, $V_e(N_{i+1}[\max]) = \emptyset$.

Otherwise, $N_{i+1}[\max] \neq \emptyset$. This means that N_{i+1} can be written as $N_{i+1} = N_{i+1}[\neg\max] \cup \{(\max_{V_e(N_{i+1}[\max])}, \oplus, Sons(N_{i+1}[\max]))\}$. Hence, n_i can be written as:

$n_i = (\max_{S \cup V_e(\cup_{N \in \{N_1,\ldots,N_i\}} N[\max])}, \oplus,$.
$\{(\emptyset, \otimes, N), (N \in \{N_1, \ldots, N_i\}) \wedge (N[\max] = \emptyset)\}$
$\cup \{(\emptyset, \otimes, N[\neg\max] \cup N'), (N \in \{N_1, \ldots, N_i\}) \wedge (N[\max] \neq \emptyset) \wedge ((\emptyset, \otimes, N') \in \mathit{Sons}(N[\max]))\}$
$\cup\{(\emptyset, \otimes, N), N \in \{N_{i+2}, \ldots, N_r\}\}$
$\cup\{(\emptyset, \otimes, N_{i+1}[\neg\max] \cup \{(\max_{V_e(N_{i+1}[\max])}, \oplus, \mathit{Sons}(N_{i+1}[\max]))\})$

According to Lemma 7.35, in order to show that $val(n_i) = val(n_{i+1})$, it suffices to prove that:

1. $V_e(N_{i+1}[\max]) \cap (S \cup V_e(\cup_{N \in \{N_1,\ldots,N_i\}} N[\max])) = \emptyset$,
2. for every $N \in \{N_1, \ldots, N_i\}$ such that $N[\max] = \emptyset$, $V_e(N_{i+1}[\max]) \cap sc((\emptyset, \otimes, N)) = \emptyset$,
3. for every $N \in \{N_1, \ldots, N_i\}$ such that $N[\max] \neq \emptyset$, and for every $(\emptyset, \otimes, N') \in \mathit{Sons}(N[\max])$, $V_e(N_{i+1}[\max]) \cap sc((\emptyset, \otimes, N[\neg\max] \cup N')) = \emptyset$,
4. for all $N \in \{N_{i+2}, \ldots, N_r\}$, $V_e(N_{i+1}[\max]) \cap sc((\emptyset, \otimes, N)) = \emptyset$,
5. $V_e(N_{i+1}[\max]) \cap sc(N_{i+1}[\neg\max]) = \emptyset$,
6. if $(\emptyset, \otimes, N''_{i+1}) \in \mathit{Sons}(N_{i+1}[\max])$, then $N_{i+1}[\neg\max] \cap N''_{i+1} = \emptyset$.

In order to show these properties, we use Lemma 7.32. We know that there exists $N'_{i+1} \in \mathfrak{N}$ such that $N_{i+1} = N'_{i+1} - N_a$.

1. Point 1 holds because thanks to Lemma 7.32. Indeed, let $j \in \{1, \ldots, i\}$. We have $N_j = N'_j - N_a$ for one $N'_j \in \mathfrak{N}$. Moreover, as $N_a[\max] = \emptyset$, $N_j[\max] = N'_j[\max]$. Lemma 7.32 enables us to write $V_e(N'_j[\max]) \cap V_e(N'_{i+1}[\max]) = \emptyset$, i.e. $V_e(N_j[\max]) \cap V_e(N_{i+1}[\max]) = \emptyset$. Therefore, $V_e(N_{i+1}[\max]) \cap V_e(\cup_{N \in \{N_1,\ldots,N_i\}} N[\max]) = \emptyset$. Moreover, $x \notin V_e(N_{i+1}[\max])$ (because x had not been considered yet). Thus, point 1 holds.
2. For point 2, let $N \in \{N_1, \ldots, N_i\}$ such that $N[\max] = \emptyset$. We have $N = N' - N_a$ for one $N' \in \mathfrak{N}$. Lemma 7.32 enables us to write $V_e(N'_{i+1}[\max]) \cap sc(N') = \emptyset$, hence $V_e(N_{i+1}[\max]) \cap sc((\emptyset, \otimes, N)) = \emptyset$. As this holds for every $N \in \{N_1, \ldots, N_i\}$, point 2 is satisfied.
3. For point 3, let $N \in \{N_1, \ldots, N_i\}$ such that $N[\max] \neq \emptyset$, and let $(\emptyset, \otimes, N') \in \mathit{Sons}(N[\max])$. We have $N = N'' - N_a$ for one $N'' \in \mathfrak{N}$. Lemma 7.32 enables us to write $V_e(N'_{i+1}[\max]) \cap sc(N'') = \emptyset$ and $V_e(N'_{i+1}[\max]) \cap V_e(N''[\max]) = \emptyset$. Therefore, $V_e(N'_{i+1}[\max]) \cap (sc(N'') \cup V_e(N''[\max])) = \emptyset$. This entails that $V_e(N_{i+1}[\max]) \cap (sc(N[\neg\max]) \cup sc(N[\max]) \cup V_e(N[\max])) = \emptyset$, i.e. $V_e(N_{i+1}[\max]) \cap (sc(N[\neg\max]) \cup sc(\mathit{Sons}(N[\max]))) = \emptyset$, and hence, $V_e(N_{i+1}[\max]) \cap sc((\emptyset, \otimes, N[\neg\max] \cup N')) = \emptyset$. As this holds for every $N \in \{N_1, \ldots, N_i\}$, point 3 is satisfied.
4. Point 4 directly holds thanks to the Lemma 7.32. Indeed, for every $N \in \{N_{i+2}, \ldots, N_r\}$, $N = N' - N_a$ for one $N' \in \mathfrak{N}$, and this lemma enables us to write $V_e(N'_{i+1}[\max]) \cap sc(N') = \emptyset$, which implies that $V_e(N_{i+1}[\max]) \cap sc((\emptyset, \otimes, N) = \emptyset$ too.
5. For point 5, we use the following property, given by Lemma 7.32, that for all $(n_t, n_u) \in N'_{i+1}$, $(n_t \neq n_u) \rightarrow (V_e(n_t) \cap sc(n_u) = \emptyset)$. For n_t such that $\{n_t\} = N_{i+1}[\max]$, this leads to: for all $n_u \in N'_{i+1} - \{n_t\}$, $V_e(n_t) \cap sc(n_u) = \emptyset$, i.e. $V_e(N_{i+1}[\max]) \cap sc(N'_{i+1}[\neg\max]) = \emptyset$, which implies that $V_e(N_{i+1}[\max]) \cap sc(N_{i+1}[\neg\max]) = \emptyset$.

6. Finally, point 6 is also entailed by Lemma 7.32. Indeed, Lemma 7.32 says that for all $(\emptyset, \otimes, N_s) \in \mathit{Sons}(N'_{i+1}[\max])$, $N_s \cap N'_{i+1}[\neg\max] = \emptyset$. As $N'_{i+1}[\max] = N_{i+1}[\max]$ and $N_{i+1}[\neg\max] \subset N'_{i+1}[\neg\max]$, this implies that for all $(\emptyset, \otimes, N_s) \in \mathit{Sons}(N_{i+1}[\max])$, $N_s \cap N_{i+1}[\neg\max] = \emptyset$.

As a result, Lemma 7.35 allows us to transform n_i into the following computation node, while ensuring that the node value is preserved

$$\begin{aligned}
&(\max\nolimits_{S\cup V_e(\cup_{N\in\{N_1,\ldots,N_i\}}N[\max])\cup V_e(N_{i+1}[\max])}, \oplus, \\
&\quad \{(\emptyset,\otimes,N), (N \in \{N_1,\ldots,N_i\}) \wedge (N[\max]=\emptyset)\} \\
&\quad \cup \{(\emptyset,\otimes,N[\neg\max]\cup N'), (N \in \{N_1,\ldots,N_i\}) \wedge (N[\max]\neq\emptyset) \wedge ((\emptyset,\otimes,N') \in \mathit{Sons}(N[\max]))\} \\
&\quad \cup\{(\emptyset,\otimes,N), N \in \{N_{i+2},\ldots,N_r\}\} \\
&\quad \cup\{(\emptyset,\otimes,N_{i+1}[\neg\max]\cup N'), N' \in \mathit{Sons}(N_{i+1}[\max])\})
\end{aligned} .$$

i.e. it enables us to transform n_i into n_{i+1} while ensuring that $val(n_i) = val(n_{i+1})$. As $val(n_i) = val(n)$ thanks to the recurrence hypothesis, we get $val(n_{i+1}) = val(n)$, i.e. the property holds at step $i+1$.

Consequently, the property holds for every $i \in \{0,\ldots,r\}$. For $i = r$, it provides us with $val(RR_{\max}(n)) = val(n)$.

The case of a min-elimination is similar. □

Proof of Lemma 7.37 (page 130). Follows directly from Lemmas 7.33 and 7.36. □

Proof of Theorem 7.38 (page 130). Follows from Lemma 7.37 and from $val(CNDAG_0(Q,o)) = Ans(Q)$ for all $o \in lin(\preceq_{Sov})$. □

Proof of Proposition 7.39 (page 130). The macrostructure of a query is obtained by using algorithm **MacroStruct**(sov, V, P, U), which calls auxiliary functions. We detail the time and space complexities of each of these functions. All elements are recorded as lists, except for the scope of each computation node, which is recorded as a table of $|V|$ booleans. Moreover, in order to explicitly handle a DAG of computation nodes, the sons of a computation node are represented by pointers to computation nodes instead of computation nodes. Given a node n, $\&n$ denotes the memory address where n is stored. The instruction $newNode(op, V_e, \circledast, Sons, sc)$ creates a computation node $(op_{V_e}, \circledast, Sons)$ and set its scope to sc.

```
begin
    $(root, PTRP) \leftarrow initialize()$
    while $(sov = sov' \cdot op_x)$ do
        $sov \leftarrow sov'$
        if $op = \oplus$ then $(root, PTRP) \leftarrow$ structure_$\oplus$()
        else $root \leftarrow$ structure_n$\oplus$()
        return $(root)$
end
```

Figure B.1: MacroStruct(sov, V, P, U).

We can assume that $|V| \neq 0$, since if $|V| = 0$, then the time and space complexities are directly 0.

```
Initialize()
begin
    $root \leftarrow$ newNode($\emptyset, \emptyset, \oplus, \emptyset, \emptyset$)
    $PTRP \leftarrow \emptyset$
    $scp \leftarrow \emptyset$
    foreach $\varphi \in P$ do
        $PTRP \leftarrow PTRP \cup \{\&\varphi\}$
        $scp \leftarrow scp \cup \{sc(\varphi)\}$
    foreach $\varphi \in U$ do
        $n \leftarrow newNode(\emptyset, \emptyset, \otimes, PTRP \cup \{\&\varphi\}, scp \cup sc(\varphi))$
        $Sons(root) \leftarrow Sons(root) \cup \{\&n\}$
    return $((root, PTRP))$
end
```

Figure B.2: Function which builds $CNDAG_0(Q, o)$.

Complexity of the initialization As adding an element to a list is $O(1)$, as computing the union of two scopes is $O(|V|)$, and as the instruction newNode(...) is $O(|P| + 1 + |V|)$, the initialization is time $|P| \cdot (O(1) + O(|V|)) + |U| \cdot (O(|P| + 1 + |V|) + O(1)) = O((|P| + |U|) \cdot |V| + |U| \cdot |P|)$.

The space complexity is $O(|P| \cdot |V| + |U| \cdot (1 + |P| + |V|))$.

Complexity of structure_$\oplus$ The two first instructions are $O(1)$.

Each iteration of the first foreach loop is time $O(|V|)$, since the only operations performed are (1) union of scopes or tests to know whether a variable is in the scope; these operations are $O(|V|)$; (2) removal of an element of a list or concatenation of two lists; these operations are $O(1)$. As it is applied at most $|P|$ times, the first foreach loop is time $O(|P| \cdot |V|)$

Let us now analyze the second foreach loop. Let us consider one iteration of this second foreach loop. As each son of the root has itself at most $1 + |P|$ sons, the internal foreach loop is time $O(|V| \cdot (1 + |P|))$. Then, the test "$PTR_{tmp} = PTRP_x$" is $O(1 + |P|)$, mainly because the list of pointers can be handled so that all pointers appear in the same order in all nodes. The instructions performed after this test can be shown to be $O(|V| \cdot (1 + |P|))$. Last, the updating of $sc(*ptr)$ is $O(|V|)$. Hence, each iteration of the second foreach loop is time $O(|V| \cdot (1 + |P|) + |V| \cdot (1 + |P|) + |V|) = O(|V| \cdot (1 + |P|))$. As the second foreach loop is performed at most $|U|$ times, the time complexity of function structure_$\oplus$ is $O(|P| \cdot |V| + |U| \cdot |V| \cdot (1 + |P|)) = O(|U| \cdot |V| \cdot (1 + |P|))$.

The space complexity of the creation of *np* is $O(|V|)$ because the space required to record a scope as a table of $|V|$ booleans is $O(|V|)$. Then, the instruction of the first foreach loop are $O(1)$, because they just correspond to concatenation of already existing lists. Hence, the first foreach loop is space $O(|P|)$. In the second foreach loop, the instructions requiring a space not $O(1)$ are the creation of n (space complexity $O(|V|)$), and the instruction $Sons(n) \leftarrow Sons(n) \cup \{ptr'\}$, which is $O(1)$ but which may be performed at most $1 + |P|$ times. This implies that the space complexity of the second foreach loop, which is performed lesser than $|U|$ times, is $O(|U| \cdot (|V| + 1 + |P|))$.

Complexity of structure_n$\oplus$ The first foreach loop is time $O(|U| \cdot |V|)$, since the root has at most $|U|$ sons, the test "$x \in sc(*ptr)$" is $O(|V|)$, and the other operations are $O(1)$. Its space complexity is 0.

The computation of *commonPTR* is time $O(|P| \cdot |U|)$ (we assume that the lists of pointers

```
structure_$\oplus$()
begin
    $np \leftarrow newNode(\oplus, \{x\}, \otimes, \emptyset, \emptyset)$
    $PTRP_x \leftarrow \emptyset$
    foreach $ptrp \in PTRP$ do
        if $x \in sc(*ptrp)$ then
            $PTRP \leftarrow PTRP - \{ptrp\}$
            $PTRP_x \leftarrow PTRP_x \cup \{ptrp\}$
            $V_e(np) \leftarrow V_e(np) \cup V_e(*ptrp)$
            $Sons(np) \leftarrow Sons(np) \cup Sons(*ptrp)$
            $sc(np) \leftarrow sc(np) \cup sc(*ptrp)$
    $PTRP \leftarrow PTRP \cup \{\&np\}$
    foreach $ptr \in Sons(root)$ do
        $PTR_{tmp} \leftarrow \emptyset$
        foreach $ptr' \in Sons(*ptr)$ do
            if $x \in sc(*ptr')$ then
                $Sons(*ptr) \leftarrow Sons(*ptr) - \{ptr'\}$
                $PTR_{tmp} \leftarrow PTR_{tmp} \cup \{ptr'\}$
        if $PTR_{tmp} = PTRP_x$ then
            $Sons(*ptr) \leftarrow Sons(*ptr) \cup \{\&np\}$
        else
            $n \leftarrow newNode(\oplus, \{x\}, \otimes, \emptyset, \emptyset)$
            foreach $ptr' \in PTR_{tmp}$ do
                if $op(*ptr') = \oplus$ then
                    $Sons(n) \leftarrow Sons(n) \cup Sons(*ptr')$
                    $V_e(n) \leftarrow V_e(n) \cup V_e(*ptr')$
                else
                    $Sons(n) \leftarrow Sons(n) \cup \{ptr'\}$
                $sc(n) \leftarrow sc(n) \cup sc(*ptr')$
            $Sons(*ptr) \leftarrow Sons(*ptr) \cup \{\&n\}$
        $sc(*ptr) \leftarrow sc(*ptr) - \{x\}$
    return $((root, PTRP))$
end
```

Figure B.3: Function implementing the rewriting for an elimination $\oplus_x$.

are ordered), and the computation of *newsc* is time $O((1+|P|) \cdot |U| \cdot |V|)$. The initialization of *newopnode* is $O(|V|)$ and the initialization of *newrootson* is $O(|P|+1+|V|)$. The space complexity of all these operations can be shown to be $O(|P|+|V|)$.

Hence, the instructions from the beginning to the second foreach loop are time $O((1+|P|) \cdot |U| \cdot |V|)$ and space $O(|P|+|V|)$.

Let us consider an iteration of the second foreach loop. The first instruction is time $O(1+|P|)$. The time complexity to test whether there is a node performing an elimination with *op* is $O(1+|P|)$. If the answer is no, the operation performed is time $O(1)$. Otherwise, the time complexity to get *ptrop* and $PTRnop$ is $O(1+|P|)$. The concatenation of the variables to eliminate is $O(1)$. Then, there are at most $1+|P|$ elements in $Sons(*ptrop)$, and for each of these elements, the operations performed are time $O(1+|P|)+O(1) = O(1+|P|)$, hence a time complexity $O((1+|P|)^2)$. Therefore, one iteration of the second foreach loop is $O((1+|P|)^2)$. As this second foreach loop is performed at most $|U|$ times, the time complexity is $O(|U| \cdot (1+|P|)^2)$. The space complexity can also be shown to be $O(|U| \cdot (1+|P|)^2)$ (the instruction which requires the more space is $n \leftarrow newNode(...)$;

```
structure_n⊕()
begin
    foreach $ptr \in Sons(root)$ do
        if $x \in sc(*ptr)$ then
            $Sons(root) \leftarrow Sons(root) - \{ptr\}$
            $PTR_{tmp} \leftarrow PTR_{tmp} \cup \{ptr\}$
    if $PTR_{tmp} \neq \emptyset$ then
        $commonPTR \leftarrow \cap_{ptr \in PTR_{tmp}} Sons(*ptr)$
        $newsc = \cup_{ptr \in PTR_{tmp}} sc(*ptr)$
        $newopnode \leftarrow newNode(op, \{x\}, \oplus, \emptyset, newsc - \{x\})$
        $newrootson \leftarrow newNode(\emptyset, \emptyset, \otimes, commonPTR \cup \{\&newopnode\}, newsc)$
        $Sons(root) \leftarrow Sons(root) \cup \{\&newrootson\}$
        foreach $ptr \in PTR_{tmp}$ do
            $Sons(*ptr) \leftarrow Sons(*ptr) - commonPTR$
            if $Sons(*ptr)[op] = \emptyset$ then
                $Sons(newopnode) \leftarrow Sons(newopnode) \cup \{ptr\}$
            else
                $\{ptrop\} \leftarrow Sons(*ptr)[op]$
                $PTRnop \leftarrow Sons(*ptr)[\neg op]$
                $V_e(newopnode) \leftarrow V_e(newopdnode) \cup V_e(*ptrop)$
                foreach $ptropson \in Sons(*ptrop)$ do
                    $n \leftarrow newNode(\emptyset, \emptyset, \otimes, PTRnop \cup \{ptropson\}, \emptyset)$
                    $Sons(newopnode) \leftarrow Sons(newopnode) \cup \{\&n\}$
    return $(root)$
end
```

Figure B.4: Function implementing the rewriting for an elimination with an operator distinct from $\oplus$.

each of such instructions is $O(|P|+1)$, and it can be performed $|PTR_{tmp}| \cdot |Sons(*ptrpop)|$ times, which is lesser than $|U| \cdot (1+|P|)$).

As a result, the time and space complexities of function structure_n⊕ are $O((1+|P|) \cdot |U| \cdot |V| + |U| \cdot (1+|P|)^2)$ and $O(|P| + |V| + |U| \cdot (1+|P|)^2)$ respectively, i.e. $O((1+|P|) \cdot |U| \cdot (|P|+|V|))$ and $O(|V| + |U| \cdot (1+|P|)^2)$.

Global complexities It suffices to sum the complexities obtained to have the global time and space complexities:

- Time complexity: $O(((|P|+|U|) \cdot |V| + |U| \cdot |P|) + |V| \cdot |U| \cdot |V| \cdot (1+|P|) + |V| \cdot (1+|P|) \cdot |U| \cdot (|P|+|V|)) = O(|U| \cdot |V| \cdot (|P|+|V|) \cdot (1+|P|))$;
- Space complexity:

  $O(|P| \cdot |V| + |U| \cdot (1+|P|+|V|) + |V| \cdot |U| \cdot (|V|+1+|P|) + |V| \cdot (|V| + |U| \cdot (1+|P|)^2)) = O(|U| \cdot |V| \cdot (|V| + |P|^2))$.

□

Proof of Proposition 7.44 (page 133). Let o be an elimination order in $lin(\preceq_{Sov})$, where Sov is the sequence of eliminations used by the query.

The property holds in $CNDAG_0(Q,o)$. Indeed, $CNDAG_0(Q,o) = (Sov(o), \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$ with $\mathfrak{N} = \{P \cup \{U_i\}, U_i \in U\}$. Therefore, for every $N \in \mathfrak{N}$, there exists a unique n such that

$t(n) = u$. If max $\neq \oplus$, then one can infer that $S \cap sc(P) = \emptyset$, hence for all $N \in \mathfrak{N}$, none of the variables eliminated in Sov are in $sc(P(N))$. Moreover, if $N, N' \in \mathfrak{N}$, then $P(N) = P(N') = P$, hence $((n \in N) \wedge (t(n) = p)) \rightarrow (n \in N')$. obviously hold.

Assume that the property holds in $CNDAG_k(Q, o)$, for $k \in \{0, \ldots, |Sov| - 1\}$. Does it hold at step $k + 1$?

If the sequence of remaining eliminations in $CNDAG_k(Q, o)$ is of the form $sov.\oplus_x$, then no new max computation node is created and the existing max computation nodes are unchanged, because rules $DR_\oplus$, DR, RR, and SR, which can be applied for the elimination of x, do not modify the max computation nodes.

If the sequence of remaining eliminations in $CNDAG_k(Q, o)$ is of the form $sov. \min_x$, then the same conclusion can be derived.

The only case which requires more work is the case where the sequence of remaining eliminations in $CNDAG_k(Q, o)$ is of the form $sov. \max_x$. The new max node created is $RR_{\max}((\max_x, \oplus, \{(\emptyset, \otimes, N - N_1), N \in \mathcal{N}^{+x}\}))$, where $N_1 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$. Let us denote by $\mathfrak{N}_a$ the set of sets of computation nodes $\mathfrak{N}_a = \{N - N_1, N \in \mathcal{N}^{+x}\}$. Hence, the max node created is $RR_{\max}((\max_x, \oplus, \{(\emptyset, \otimes, N_a), N_a \in \mathcal{N}_a\}))$. Does it satisfy the required property?

Let $N_a \in \mathfrak{N}_a$. Then, there exists $N \in \mathfrak{N}$ such that $N_a = N - N_1$. If $u(N) \in N_1$, then this means that $\mathfrak{N}^{+x} = \{N\}$ (because if $\mathfrak{N}^{+x}$ contains another element N' , then $(N \neq N') \rightarrow (u(N) \neq u(N'))$). This implies that $x \notin sc(u(N))$. As $x \notin sc(P(N))$, thanks to the recurrence assumption, this implies that $x \notin sc(N)$, which is a contradiction because $N \in \mathfrak{N}^{+x}$. Therefore, the initial hypothesis $u(N) \in N_1$ is false, i.e. $u(N) \in N - N_1$. This proves that there exists a unique computation node of type u in N_a.

Do we have $S \cap sc(P(N)) = \emptyset$?

Let $CNDAG_k(Q, o) = (sov. \max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$. If max $\neq \oplus$, then for all $N \in \mathfrak{N}$, for all $n \in N$, $(t(n) = p) \rightarrow (x \notin sc(n))$

Indeed, assume that $t(n) = p$ and $x \in sc(n)$. Then, by connectivity of the components and thanks to the updating of the definition of N^{+x}, we know that $n = (\oplus_S, \otimes, N)$, where S contains at least one environment component c_0 in the descendants of $c(x)$ and that N contains $Fact(c_0)$. Moreover, c_0 can be chosen the deeper as possible, so that for all $n \in N - Fact(c_0)$, $c_0 \cap sc(n) = \emptyset$. This leads to a contradiction because c_0 should have been eliminated. Therefore, $(t(n) = p) \rightarrow (x \notin sc(n))$.

Let us show that for all computation nodes $(\emptyset, \otimes, N)$ in $CNDAG_k(Q, o)$, there exists a unique computation node n in N such that $t(n) = u$.

The property holds for $k = 0$ since the $(\emptyset, \otimes, N)$ nodes involved in the initial DAG of computation nodes are of the form $(\emptyset, \otimes, P \cup \{U_i\})$ with $U_i \in U$, hence the only node of type u is U_i.

Assume that the property holds at step k. We must show that the $(\emptyset, \otimes, N)$ nodes created from $CNDAG_k(Q, o)$ to $CNDAG_{k+1}(Q, o)$ satisfy the required property.

- If $CNDAG_k(Q, o) = (sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, then, if no simplification is used, $CNDAG_{k+1}(Q, o) = (sov, \oplus, \{(\emptyset, \otimes, N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}), N \in \mathfrak{N}\})$. The unique computation nodes of the form $(\emptyset, \otimes, N)$ which differ from $CNDAG_k(Q, o)$ to $CNDAG_{k+1}(Q, o)$ are the nodes of the form $(\emptyset, \otimes, N^{-x} \cup \{RR((\oplus_x.\otimes, N^{+x}))\})$ for $N \in \mathfrak{N}$.

Given $N \in \mathfrak{N}$, let $N' = N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}$. It it is straightforward that either $u(N) \in N^{-x}$, and hence $u(N') = u(N)$, or $u(N) \in N^{+x}$ and hence $u(N') = RR((\oplus_x, \otimes, N^{+x}))$.

If function *simplify* is used, then the result still holds because this function does neither modify the type of a node, nor remove nodes of type u.

Hence, the property holds at step $k+1$.

- If $\max \neq \oplus$ and $CNDAG_k(Q,o) = (sov.\max_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, then, either $\mathfrak{N}^{+x} = \emptyset$ and the property is directly satisfied at step $k+1$, or $CNDAG_k(Q,o) = (sov.\oplus_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}^{-x}\}) \cup \{(\emptyset, \otimes, N_1 \cup \{RR_{\max}((\max_x, \oplus, \{((\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\}))\})\}$, with $N_1 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$. In this case, we know that for each $n \in N_1$, $t(n) = p$. As $t((\emptyset, \otimes, N)) = u$, this implies that $t((\emptyset, \otimes, N - N_1)) = u$. As $\mathfrak{N}^{+x} \neq \emptyset$, this implies that $t(RR_{\max}((\max_x, \oplus, \{((\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\}))) = u$, and therefore the node created from step k to step $k+1$, which is $(\emptyset, \otimes, N_1 \cup \{RR_{\max}((\max_x, \oplus, \{((\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\}))\})$, satisfies the required property.

  But some nodes are updated, due to the recomposition rule $RR_{\max}$, which transforms $(\max_x, \oplus, \{((\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\})$ into another node $(\max_S, \oplus, \{(\emptyset, \otimes, N'), N' \in \mathfrak{N}'\})$. We must show that for every $N' \in \mathfrak{N}'$, $(\emptyset, \otimes, N')$ satisfies the required property.

  Let $N' \in \mathfrak{N}'$. Then, N' can be of the form $N''[\neg \max] \cup N_s$ with $N'' = N - N_1$ for some $N \in \mathfrak{N}^{+x}$ and $(\emptyset, \otimes, N_s) \in Sons(N[\max])$. Due to the recurrence assumption, we know that there exists a unique $n \in N_s$ such that $t(n) = u$. This implies that $t(N''[\max]) = u$, and therefore, by unicity, for all $n \in N''[\neg \max]$, $t(n) = p$. This entails that there exists a unique $n \in N''[\neg \max] \cup N_s$ such that $t(n) = u$.

  But N'' can also be of the form $(\emptyset, \otimes, N - N_1)$ with $N \in \mathfrak{N}^{+x}$. As $t(n) = p$ for every $n \in N_1$, this implies that the unique node of type u which was is N is now in $N - N_1$, and it is still unique.

  Consequently, the property holds at step $k+1$.

- Idem for an elimination $\min_x$ when $\oplus \neq \min$.

Hence the proof by recurrence that if $(op_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$ is in $CNDAG(Q)$, then for all $N \in \mathfrak{N}$, there exists a unique $n \in N$ such that $t(n) = u$.

Given that all max computation nodes are of the form $(\max_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, this implies that, at each step k, all max-nodes in $CNDAG_k(Q,o)$ are of type u, and therefore given a computation node $(\emptyset, \otimes, N)$, if $N[\max] \neq \emptyset$, then $N[\max] = \{u(N)\}$.

Let us show an invariant for the sons of the root: let us show that if $CNDAG_k(Q,o) = (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$, then the following properties hold: for all $N_1, N_2 \in \mathfrak{N}$,

(C1) If $N_1[\max] = N_2[\max] = \emptyset$, then

$$[(n \in N_1) \wedge (t(n) = p)] \rightarrow [(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))]$$

(C2) If $N_1[\max] = \emptyset$ and $N_2[\max] \neq \emptyset$, then, for all $(\emptyset, \otimes, N_{s2}) \in Sons(N_2[\max])$,

$$[(n \in N_1) \wedge (t(n) = p)] \rightarrow [(n \in N_2[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_2)))]$$

(C3) If $N_1[\max] \neq \emptyset$ and $N_2[\max] = \emptyset$, then, for all $(\emptyset, \otimes, N_{s1}) \in Sons(N_1[\max])$,

$$[(n \in N_1[\neg \max] \cup N_{s1}) \wedge (t(n) = p)] \rightarrow [(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))]$$

(C4) If $N_1[\max] \neq \emptyset$ and $N_2[\max] \neq \emptyset$, then, for all $(\emptyset, \otimes, N_{s1}) \in Sons(N_1[\max])$, for all $(\emptyset, \otimes, N_{s2}) \in Sons(N_2[\max])$,

$$[(n \in N_1[\neg \max] \cup N_{s1}) \wedge (t(n) = p)] \rightarrow [(n \in N_2[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_2)))]$$

The property holds at step $k = 0$, because if $N_1, N_2 \in \mathfrak{N}$, then $N_1 = P \cup \{U_1\}$ and $N_2 = P \cup \{U_2\}$, with $U_1, U_2 \in U$, and therefore we have first, $N_1[\max] = N_2[\max] = \emptyset$, and second $(n \in N_1) \wedge (t(n) = p)$ implies that $n \in P$, and therefore $n \in N_2$.

Assume that the property holds at step k. Let us show that it holds at step $k + 1$.

Let $CNDAG_k(Q, o) = (sov.op_x, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}_k\})$. We study several cases depending on op.

- Case $op_x = \oplus_x$

  Assume that function *simplify* is not used. In this case, we have

  $$CNDAG_{k+1} = (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}_{k+1}\})$$

  with

  $$\mathfrak{N}_{k+1} = \{N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}, N \in \mathfrak{N}_k\}$$

  Let $N_1, N_2 \in \mathfrak{N}_{k+1}$. There exist $N, N' \in \mathfrak{N}_k$ such that

  $$N_1 = N^{-x} \cup \{RR((\oplus_x, \otimes, N^{+x}))\}$$
  $$N_2 = N'^{-x} \cup \{RR((\oplus_x, \otimes, N'^{+x}))\}$$

  We analyze the four cases corresponding to (C1), (C2), (C3), and (C4).

  1. If $N[\max] = N'[\max] = \emptyset$, then $N_1[\max] = N_2[\max] = \emptyset$.

     Let $n \in N_1$ such that $t(n) = p$.

     – Either $n \in N^{-x}$.

       This means that $n \in N$ and $x \notin sc(n)$. As $n \in N$, the recurrence assumption implies that (a) either $n \in N'$, and hence $n \in N'^{-x}$, which implies that $n \in N_2$; (b) or $sc(n) \subset sc(u(N'))$, and in this case, it is not hard to see that $sc(u(N')) \subset sc(u(N_2)) \cup \{x\}$, which implies that $sc(n) \subset sc(u(N_2)) \cup \{x\}$, and, as $x \notin sc(n)$, that $sc(n) \subset sc(u(N_2))$.

     – Or $n = RR((\oplus_x, \otimes, N^{+x}))$.

       In this case, as $t(n) = p$, we know that for all $n_a \in N^{+x}$, $t(n_a) = p$, and hence for all $n_a \in N^{+x}$, we have $(n_a \in N'^{+x}) \vee (sc(n_a) \subset sc(u(N')))$. This notably implies that $sc(N^{+x}) \subset sc(N'^{+x})$

       If there exists $n_a \in N^{+x}$ such that $sc(n_a) \subset sc(u(N'))$, then we can infer that $t(RR((\oplus_x, \otimes, N'^{+x}))) = u$. Moreover, as $sc(N^{+x}) \subset sc(N'^{+x})$, this implies that $sc(n) \subset sc(u(N_2))$

       Otherwise, for all $n_a \in N^{+x}$, we have $n_a \in N'^{+x}$. This implies that $N^{+x} \subset N'^{+x}$. In another direction, if $n_b \in N'^{+x}$, then we can write $(n_b \in N^{+x}) \vee (sc(n_b) \subset sc(u(N)))$. As $x \in sc(n_b)$ and $x \notin sc(u(N))$ (otherwise n would not be of type p), this implies that $n_b \in N^{+x}$. Therefore, $N'^{+x} \subset N^{+x}$ also holds, which implies that $N^{+x} = N'^{+x}$, and consequently $n = RR((\oplus_x, \otimes, N^{+x}) = RR((\oplus_x, \otimes, N'^{+x})) \in N_2$.

     Hence, we have $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$

2. If $N[\max] = \emptyset$ and $N'[\max] \neq \emptyset$.

   Then, we have $N_1[\max] = \emptyset$. Let $n \in N_1$ such that $t(n) = p$.

   Let us first analyze the case $N_2[\max] = \emptyset$. In this case, we have $u(N') \in N'^{+x}$ (because the max node, which is necessarily of type u, has disappeared in N_2). Then,

   - Either $n \in N^{-x}$.

     In this case, we know that for all $(\emptyset, \otimes, N'_s) \in \mathit{Sons}(N'[\max])$,
     $$(n \in N'[\neg \max] \cup N'_s) \vee (sc(n) \subset sc(u(N')))$$
     If $n \in N'[\neg \max]$, then, as $x \notin sc(n)$, we have $n \in N'^{-x}$, hence $n \in N_2$.

     Otherwise, if $n \in N'_s$, then $sc(n) \subset sc(u(N') \cup V_e(N'[\max]))$. As $t(n) = p$, one can infer that $sc(n) \cap V_e(N'[\max]) = \emptyset$. Therefore, $sc(n) \subset sc(u(N'))$. As $u(N') \in N'^{+x}$, we can infer that $sc(n) \subset sc(N'^{+x})$. As $x \notin sc(n)$, this entails that $sc(n) \subset sc(RR((\oplus_x, \otimes, N'^{+x})))$, i.e. $sc(n) \subset sc(u(N_2))$.
   - Or $n = RR((\oplus_x, \otimes, N^{+x}))$.

     The recurrence assumption implies that for all $n_a \in N^{+x}$, for all $(\emptyset, \otimes, N'_s) \in \mathit{Sons}(N'[\max])$,
     $$(n_a \in N'[\neg \max] \cup N'_s) \vee (sc(n_a) \subset sc(u(N')))$$
     In both cases, as $x \in sc(n_a)$, we can infer that $sc(n_a) \subset sc(N'^{+x})$. Consequently, $sc(N^{+x}) \subset sc(N'^{+x})$. This implies that $sc(n) \subset sc(u(N_2))$.

   Otherwise, $N_2[\max] \neq \emptyset$. In this case, we have $N_2[\max] = \{u(N_2)\} = N'[\max] = u(N')$.

   - Either $n \in N^{-x}$.

     Let $(\emptyset, \otimes, N_{s2}) \in \mathit{Sons}(N_2[\max])$. Then, $(\emptyset, \otimes, N_{s2}) \in \mathit{Sons}(N'[\max])$, which implies, as $n \in N$, that $(n \in N'[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$. Given that $N'[\neg \max] = (N_2[\neg \max] - \{RR((\oplus_x, \otimes, N'^{+x}))\}) \cup N'^{+x}[\neg \max]$ and that $x \notin sc(n)$, this entails that $n \in N_2[\neg \max]$. Therefore, $(n \in N_2[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$.
   - Or $n = RR((\oplus_x, \otimes, N^{+x}))$.

     Let $(\emptyset, \otimes, N_{s2}) \in \mathit{Sons}(N_2[\max])$. Then, $(\emptyset, \otimes, N_{s2}) \in \mathit{Sons}(N'[\max])$, which implies that for all $n_a \in N^{+x}$, $(n_a \in N'[\neg \max] \cup N_{s2}) \vee (sc(n_a) \subset sc(u(N_{s2})))$.

     As $x \in sc(n_a)$ and $x \notin sc(u(N_{s2}))$ (because $N_2[\max] \neq \emptyset$), we can infer that $n_a \in N'[\neg \max]$, and therefore $n_a \in N'^{+x}[\neg \max]$, and therefore $n_a \in N'^{+x}$. This entails that $N^{+x} \subset N'^{+x}$.

     Moreover, if $n_b \in N'^{+x}$, then $n_b \in N'^{+x}[\neg \max]$. The recurrence assumption implies that $n_b \in N^{+x}$ or $sc(n_b) \subset sc(u(N))$. As $x \in sc(n_b)$ and $x \notin sc(u(N))$ (because $t(RR((\oplus_x, \otimes, N^{+x}))) = p$), this entails that $n_b \in N^{+x}$, hence $N'^{+x} \subset N^{+x}$.

     Therefore, $N^{+x} = N^{-x}$, which entails that $n \in N_2$.

   This proves the required result for the case $N[\max] = \emptyset$ and $N'[\max] \neq \emptyset$.

3. If $N[\max] \neq \emptyset$ and $N'[\max] = \emptyset$

   In this case, $N_2[\max] = \emptyset$. We analyze two cases, depending on whether $N_1[\max] = \emptyset$ or not.

   First, if $N_1[\max] = \emptyset$, then, as $N[\max] \neq \emptyset$, we have $N[\max] \subset N^{+x}$, or equivalently $x \in sc(u(N))$. Let $n \in N_1$ such that $t(n) = p$. We must show that $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$.

- Either $n \in N^{-x}$

  In this case, we know that $n \in N[\neg \max]$ (because $x \in N[\max]$). The recurrence assumption therefore implies that $(n \in N') \vee (sc(n) \subset sc(u(N')))$, i.e. $(n \in N'^{-x}) \vee (sc(n) \subset sc(u(N')))$, which entails that $(n \in N_2) \vee (sc(n) \subset sc(u(N')))$. As $x \notin sc(n)$, it is not hard to infer that $sc(n) \subset sc(u(N_2))$. As a result, $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$.
- Or $n = RR((\oplus_x, \otimes, N^{+x}))$

  This node cannot be of type p, otherwise we would not have $N_1[\max] = \emptyset$.

Second, let us assume that $N_1[\max] \neq \emptyset$. Then, $N_1[\max] = N[\max] = \{u(N)\} = \{u(N_1)\}$, and $x \notin sc(u(N))$.

Let $(\emptyset, \otimes, N_{s1}) \in Sons(N_1[\max])$. Then, $(\emptyset, \otimes, N_{s1}) \in Sons(N[\max])$. This implies that if $n \in N[\neg \max] \cup N_{s1}$ and $t(n) = p$, then $(n \in N') \vee (sc(n) \subset sc(u(N')))$.

- If $n \in N^{-x}[\neg \max] \cup N_{s1}$, then $x \notin sc(n)$, and therefore $(n \in N'^{+x}) \vee (sc(n) \subset sc(u(N_2)))$, which implies that $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$.
- Otherwise, if $n \in (N_1[\neg \max] \cup N_{s1}) - (N^{-x}[\neg \max] \cup N_{s1})$, then this means that $n \in N_1[\neg \max] - N^{-x}[\neg \max]$, i.e. $n \in (N_1 - N^{-x})[\neg \max]$, i.e. $n = RR((\oplus_x, \otimes, N^{+x}))$. Does $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$ hold? Given that $N_1[\max] \neq \emptyset$, we know that $N^{+x}[\neg \max] = N^{+x}$. Due to the recurrence assumption, this enables us to infer that for all $n_a \in N^{+x}$, $(n_a \in N'^{+x}) \vee (sc(n_a) \subset sc(u(N')))$.
  - ∗ If $x \in sc(u(N'))$, then one can directly infer that $sc(n) \subset sc(u(n_2))$.
  - ∗ Otherwise, $x \notin sc(u(N'))$. In this case, for all $n_a \in N^{+x}$, $(n_a \in N'^{+x})$, which means that $N^{+x} \subset N'^{+x}$. Conversely, let $n_b \in N'^{+x}$. The recurrence assumption enables us to infer that given $(\emptyset, \otimes, N_s) \in Sons(N[\max])$, i.e. given $(\emptyset, \otimes, N_s) \in Sons(N_1[\max])$, we have $(n_b \in N[\neg \max] \cup N_s) \vee (sc(n_b) \subset sc(u(N)))$. As $x \in sc(n_b)$ and $x \notin sc(u(N))$ (because otherwise, we would have $N_1[\max] = \emptyset$), we have $n_b \in N[\neg \max]$, and therefore $n_b \in N^{+x}[\neg \max]$, hence $n_b \in N^{+x}$. As a result, $N'^{+x} = N^{+x}$.

    As $N^{+x} = N'^{+x}$, we obtain $n \in N_2$.

This shows that the property hold at step $k+1$ when $N[\max] \neq \emptyset$ and $N'[\max] = \emptyset$.

4. If $N[\max] \neq \emptyset$ and $N'[\max] \neq \emptyset$

   In this case, we can have $N_1[\max] = \emptyset$ or not and $N_2[\max] = \emptyset$ or not: we must analyze four cases.

   (a) Case 1: $N_1[\max] = N_2[\max] = \emptyset$

   In this case, we know that $x \in sc(u(N))$ and $x \in sc(u(N))$.

   Let $n \in N_1$ such that $t(n) = p$.

   - Either $n \in N^{-x}$

     In this case, $n \in N[\neg \max]$. The recurrence assumption implies that given $(\emptyset, \otimes, N'_s) \in Sons(N'[\max])$, we have $(n \in N'[\neg \max] \cup N'_s) \vee (sc(n) \subset sc(u(N'_s)))$. As $x \in sc(N[\max])$ and $N[\max] = \{u(N)\}$, this allows us to write, if $n \notin N'[\neg \max]$, that $sc(n) \subset sc(u(N_2))$. Otherwise, if $n \in N'[\neg \max]$, then we can write $n \in N'^{-x}[\neg \max]$, which implies that $n \in N_2$.

As a result, $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$.

- Or $n = RR(\oplus_x, \otimes, N^{+x})$

  This case is impossible because as $N_1[\max] = \emptyset$, we have $N[\max] \in N^{+x}$, and hence $t(n) = u$.

(b) Case 2: $N_1[\max] = \emptyset$ and $N_2[\max] \neq \emptyset$

In this case, $x \in sc(u(N))$ and $N_2[\max] = N'[\max] = \{u(N)\} = \{u(N_2)\}$ and $x \notin sc(u(N))$.

Let $n \in N_1$ such that $t(n) = p$ and let $(\emptyset, \otimes, N_{s2}) \in Sons(N_2[\max])$ (we also have $(\emptyset, \otimes, N_{s2}) \in Sons(N'[\max])$). Does $(n \in N_2[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$ hold?

As $N_1[\max] = \emptyset$ and $N[\max] \neq \emptyset$, one can infer that $t(RR((\oplus_x, \otimes, N^{+x}))) = u$, hence if $n \in N_1$ and $t(n) = p$, then $n \in N^{-x}$, and therefore $n \in N^{-x}[\neg\max]$. The recurrence assumption implies that $(n \in N'[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$. As $x \notin sc(n)$, this entails that $(n \in N'^{-x}[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$, and therefore $(n \in N'^{-x} \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$, and therefore $(n \in N_2 \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$. Hence the required result.

(c) Case 3: $N_1[\max] \neq \emptyset$ and $N_2[\max] = \emptyset$

In this case, $x \in sc(u(N'))$, $N_1[\max] = N[\max] = \{u(N_1)\} = \{u(N_2)\}$, and $x \notin sc(u(N))$.

Let $(\emptyset, \otimes, N_{s1}) \in Sons(N_1[\max])$ and let $n \in N_1[\neg\max] \cup N_{s1}$ such that $t(n) = p$. Does $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$ holds?

- If $n \in N^{-x}$

  In this case, we have $(\emptyset, \otimes, N_{s1}) \in Sons(N[\max])$ and $n \in N^{-x}[\neg\max] \cup N_{s1} \subset N[\neg\max] \cup N_{s1}$. Due to the recurrence assumption, this implies that given $(\emptyset, \otimes, N'_s) \in Sons(N'[\max])$, we have $(n \in N'[\neg\max] \cup N'_s) \vee (sc(n) \subset sc(u(N'_s)))$, i.e. $(n \in N'[\neg\max]) \vee (n \in N'_s) \vee (sc(n) \subset sc(u(N'_s)))$.

  If $n \in N'[\neg\max]$, then $n \in N'^{-x}[\neg\max]$, and therefore $n \in N_2$. Otherwise, $(n \in N'_s) \vee (sc(n) \subset sc(u(N'_s)))$. Hence, $sc(n) \subset sc(N'_s)$. As $u(N_2) = RR((\oplus_x, \otimes, N'^{+x}))$ and $N'[\max] \subset N^{+x}$, this allows us to infer that $sc(n) \subset sc(u(N_2))$.

  As a result, $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$.

- Otherwise, $n \in (N_1[\neg\max] \cup N_{s1}) - N^{-x} = (N_1[\neg\max] - N^{-x}) \cup N_{s1} = \{RR((\oplus_x, \otimes, N^{+x}))\} \cup N_{s1}$.

  As $N_1[\max] \neq \emptyset$, this means that for every $n_a \in N^{+x}$, we have $n_a \in N[\neg\max]$ and $t(n_a) = p$. Due to the recurrence assumption, this entails that given $(\emptyset, \otimes, N'_s) \in Sons(N'[\max])$, we have $(n_a \in N'[\neg\max] \cup N'_s) \vee (sc(n_a) \subset sc(u(N'_s)))$. As $x \in sc(n_a)$, we can have neither $n_a \in N'_s$, nor $sc(n_a) \subset sc(u(N'_s))$ (since otherwise, we would have $N'[\max] = \emptyset$). Therefore, $n_a \in N'[\neg\max]$, and also $n_a \in N'^{+x}[\neg\max]$. This implies that $N^{+x} \subset N'^{+x}$.

  Let $n_b \in N'^{+x}$. Then, as $N'[\max] \neq \emptyset$, we can write $n_b \in N'^{+x}[\neg\max]$. The recurrence assumption entails that given $(\emptyset, \otimes, N_s) \in Sons(N[\max])$, we have $(n_b \in N[\neg\max] \cup N_s) \vee (sc(n_b) \subset sc(u(N_s)))$.

If there exists $n_b \in N'^{+x}$ such that $n_b \in N_s$ or $sc(n_b) \subset sc(u(N_s))$, then we can directly infer that $sc(n) \subset sc(u(N_2))$ (because $u(N_2) = RR((\oplus_x, \otimes, N'^{+x}))$ and $sc(N_s) \subset sc(N'^{+x})$).

Otherwise, we obtain that for all $n_b \in N'^{+x}$, $n_b \in N[\neg \max]$, and therefore $n_b \in N^{+x}[\neg \max]$, and therefore $n_b \in N^{+x}$. In this case, $N^{+x} = N'^{+x}$, which entails that $n \in N_2$.

Hence, $(n \in N_2) \vee (sc(n) \subset sc(u(N_2)))$ is always satisfied.

(d) Case 4: $N_1[\max] \neq \emptyset$ and $N_2[\max] \neq \emptyset$

In this case, $x \notin sc(u(N))$ and $x \notin sc(u(N'))$.

Let $(\emptyset, \otimes, N_{s1}) \in Sons(N_1[\max])$ and let $(\emptyset, \otimes, N_{s2}) \in Sons(N_2[\max])$. Let $n \in N_1[\neg \max] \cup N_{s1}$ such that $t(n) = p$.

- If $n \in N^{-x}$

  Then, $n \in N[\neg \max] \cup N_{s1}$ and $t(n) = p$. As $N_1[\max] = N[\max]$ and $N_2[\max] = N'[\max]$, the recurrence assumption enables us to infer that if $n \in N[\neg \max] \cup N_{s1}$ and $t(n) = p$, then $(n \in N'[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$, which implies that $(n \in N'^{-x}[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$, and therefore $(n \in N_2 \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$.

- Otherwise, $n = RR((\oplus_x, \otimes, N^{+x}))$

  If $n_a \in N^{+x}$, then $n_a \in N[\neg \max]$ (because otherwise, we would have $N_1[\max] = \emptyset$). The recurrence assumption entails that $(n_a \in N'[\neg \max] \cup N_{s2}) \vee (sc(n_a) \subset sc(u(N_{s2})))$. As $x \in sc(n_a)$, neither $n_a \in N_{s_2}$, nor $sc(n_a) \subset sc(u(N_{s2}))$ can be satisfied (otherwise, we should have $N_2[\max] = \emptyset$). Hence, $n_a \in N'[\neg \max]$. This implies that $n_a \in N'^{+x}[\neg \max]$, and therefore $n_a \in N'^{+x}$. As a result, $N^{+x} \subset N'^{+x}$.

  Similarly, it is possible to prove that for all $n_b \in N'^{+x}$, we have $n_b \in N^{+x}$, and hence $N^{+x} = N'^{+x}$. This entails that $n \in N_2$.

As a result, $(n \in N_2[\neg \max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$.

If function *simplify* is used, then the result still holds because as soon as a simplification occurs in a computation node of type u, then the same simplification can be done in computation nodes of type p.

- Case $op_x = \max_x$, with $\max \neq \oplus$

  If $\mathfrak{N}^{+x} = \emptyset$, then the property is obviously satisfied at the next step.

  Otherwise, we have $CNDAG_{k+1} = (sov, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}_{k+1}\})$ with
  $$\mathfrak{N}_{k+1} = \mathfrak{N}_k^{-x} \cup \{N_0 \cup \{RR_{\max}((\max_x, \oplus, \{(\emptyset, \otimes, N - N_0), N \in \mathfrak{N}^{+x}\}))\}\}$$
  where $N_0 = \cap_{N \in \mathfrak{N}^{+x}} N^{-x}$.

  Let $N_1, N_2 \in \mathfrak{N}_{k+1}$.

  - If $N_1, N_2 \in \mathfrak{N}_k^{-x}$, then the property is directly satisfied.
  - If $N_1 \in \mathfrak{N}_k^{-x}$ and $N_2 = N_0 \cup \{RR_{\max}((\max_x, \oplus, \{(\emptyset, \otimes, N - N_0), N \in \mathfrak{N}^{+x}\}))\}$.

    In this case, $N_2[\max] \neq \emptyset$.

$*$ If $N_1[\max] = \emptyset$

Let $n \in N_1$ such that $t(n) = p$. Let $(\emptyset, \otimes, N_{s2}) \in Sons(N_2[\max])$.

- Either $N_{s2} = N - N_0$ with $N \in \mathfrak{N}_k^{+x}$, and $(N - N_0)[\max] = \emptyset$. Then, we have $N[\max] = \emptyset$. According to the recurrence assumption, this implies that $(n \in N) \vee (sc(n) \subset sc(u(N)))$. Therefore, $(n \in N_0 \cup (N - N_0)) \vee (sc(n) \subset sc(u(N)))$. As $N_0 = N_2[\neg\max]$ and $N - N_0 = N_{s2}$, we have $(n \in N_2[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N)))$.

  As $sc(u(N)) = sc(u(N - N_0)) = sc(u(N_{s2}))$, this entails that $(n \in N_2[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$.
- Or $N_{s2} = (N - N_0)[\neg\max] \cup N_s$ with $N[\max] \neq \emptyset$ and $(\emptyset, \otimes, N_s) \in Sons(N[\max])$. The recurrence assumption implies that $(n \in N[\neg\max] \cup N_s) \vee (sc(n) \subset sc(u(N_s)))$. First, we have $u(N_{s2}) = u((N - N_0)[\neg\max] \cup N_s) = u(N_s)$. Second, we have $N[\neg\max] \cup N_s = N_0 \cup N_{s2} = N_2[\neg\max] \cup N_{s2}$. This implies that $(n \in N_2[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$.

Therefore, in both cases, $(n \in N_2[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$.

$*$ Otherwise, $N_1[\max] \neq \emptyset$

Let $(\emptyset, \otimes, N_{s1}) \in Sons(N_1[\max])$ and $(\emptyset, \otimes, N_{s2}) \in Sons(N_2[\max])$. Let $n \in N_1[\neg\max] \cup N_{s1}$.

Does $(n \in N_2[\neg\max] \cup N_{s2}) \vee (sc(n) \subset sc(u(N_{s2})))$ hold?

- Either $N_{s2} = N - N_0$ with $N \in \mathfrak{N}_k^{+x}$ with $(N - N_0)[\max] = \emptyset$ Then, we have $N[\max] = \emptyset$. According to the recurrence assumption, this implies that $(n \in N) \vee (sc(n) \subset sc(u(N)))$. As $N = N_0 \cup N_{s2} = N_2[\neg\max] \cup N_{s2}$ and $u(N) = u(N_{s2})$, this entails the required result.
- Or $N_{s2} = (N - N_0)[\neg\max] \cup N_s$ with $N[\max] \neq \emptyset$ and $(\emptyset, \otimes, N_s) \in Sons(N[\max])$. The recurrence assumption implies that $(n \in N[\neg\max] \cup N_s) \vee (sc(n) \subset sc(u(N_s)))$. In this case, as $N_{s2} = (N - N_0)[\neg\max] \cup N_s = (N[\neg\max] \cup N_s) - N_0$, we have $N[\neg\max] \cup N_s = N_{s2} \cup N_0 = N_{s2} \cup N_2[\neg\max]$. Moreover, as in the previous case, it can be shown that $u(N_s) = u_(N_{s2})$, which implies the required result.

– If $N_1 = N_0 \cup \{RR_{\max}((\max_x, \oplus, \{(\emptyset, \otimes, N - N_0), N \in \mathfrak{N}^{+x}\}))\}$ and $N_2 \in \mathfrak{N}_k^{-x}$. The result is proved in a similar way as the previous case.

- Case $op_x = \min_x$, with $\min \neq \oplus$

  Same proof as in the case $op_x = \max_x$.

As a result, we have prove the invariant for the root. Thanks to this invariant, it is possible to infer that for the internal max node $(\max_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$ (which are recomposed, i.e. which satisfy $N[\max] = \emptyset$), case (C1) holds, i.e. for all $N_1, N_2 \in \mathfrak{N}$, $[(n \in N) \wedge (t(n) = p)] \rightarrow [(n \in N') \vee (sc(n) \subset sc(u(N')))]$. □

Proof of Theorem 7.39 (page 130). The only nodes for which a justification is needed are the max computation nodes (if $\max \neq \oplus$) and the min computation nodes (if $\min \neq \oplus$). We prove the result for max computation nodes only.

Let $n = (\max_S, \oplus, \{(\emptyset, \otimes, N), N \in \mathfrak{N}\})$ be a max computation node.

Let us consider the graphical model $\mathcal{M} = (sc(n) \cup \{S\}, \{val(u(N)), N \in \mathfrak{N}\})$. Let $(T, V(.), \Phi(.))$ be a cluster-tree decomposition of $\mathcal{M}$ given $sc(n) - S$. Let r be the root of this decomposition. We have $val(r) = \max_S(\oplus_{N \in \mathfrak{N}} val(u(N)))$. Let $N \in \mathfrak{N}$ and let $n \in N$ such that $t(n) = p$. We know that $S \cap sc(n) = \emptyset$. Therefore, if we add levels in the cluster-tree decomposition where each $val(u(N))$ is combined with $val(N - \{u(N)\})$, the value of the new root r' is $val(r') = \max_S(\oplus_{N \in \mathfrak{N}}((\otimes_{n' \in N - \{u(N)\}} val(n')) \otimes val(u(N))) = \max_S(\oplus_{N \in \mathfrak{N}}(\otimes_{n' \in N} val(n')) = val(n)$. Then, moving some weights is the structure does not change the result.

Concerning optimal decision rules, the argument is still that $\text{argmax}_x U^{+x} = \text{argmax}_x(U^{-x} \oplus U^{+x})$. □

Proof of Proposition 7.48 (page 135). Let C denote the set of clusters of the MCDAG. Each cluster c of the MCDAG must perform $|Sons(c)| + |\Phi(c)| - 1$ combination operations for each assignment of its variables. Therefore, the computations performed by one cluster c are time $O((|\Phi(c)| + |Sons(c)| - 1) \cdot d^{1+w_{CNDAG}})$. Summing on all clusters of the MCDAG gives a time complexity $O((\sum_{c \in C}(|\Phi(c)| + |Sons(c)| - 1)) \cdot d^{1+w_{CNDAG}})$.

Let us show that $\sum_{c \in C}(|\Phi(c)| + |Sons(c)| - 1) \leq 2 \cdot (1 + |P|) \cdot (1 + |U|)$:

- First, the number of scoped function in the MCDAG is lesser than $|P| \cdot |U| + |U|$, because each utility functions appears exactly once in the MCDAG and each plausibility function can be duplicated $|U|$ times. Hence $\sum_{c \in C} |\Phi(c)| \leq |P| \cdot |U| + |U|$.

- Second, let C_p and C_u denote the sets of clusters of type p and u respectively (a cluster of type p involves only plausibility functions, whereas a cluster c is of type u involves a utility function either in $\Phi(c)$ or in its descendants). Given a cluster c, let us denote $Sons_p(c)$ and $Sons_u(c)$ the sets of sons of c which are of type p and u respectively. Then,
  $\sum_{c \in C}(|Sons(c)| - 1)$
  $= \sum_{c \in C_p}(|Sons(c)| - 1) + \sum_{c \in C_u}(|Sons(c)| - 1)$
  $= \sum_{c \in C_p}(|Sons_p(c)| - 1) + \sum_{c \in C_u}(|Sons(c)| - 1)$
  (because the sons of clusters of type p are of type p)
  $\leq (|P| - 1) + \sum_{c \in C_u}(|Sons(c)| - 1)$
  (because the structure obtained when keeping only clusters of type p is a forest which has at most $|P|$ leafs)
  $\leq (|P| - 1) + \sum_{c \in C_u}(|Sons_u(c)| - 1) + \sum_{c \in C_u} |Sons_p(c)|$
  $\leq (|P| - 1) + (|U| - 1) + \sum_{c \in C_u} |Sons_p(c)|$
  (because the structure obtained when keeping only clusters of type u is a tree which has at most $|U|$ leafs)
  $\leq (|P| - 1) + (|U| - 1) + |P| \cdot |U|$

  The last inequality holds for several reasons. First, if one keeps only the clusters in C_p, then one obtains a forest with at most $|P|$ trees (because there are at most $|P|$ leaves). Second, each of the tree in this forest is connected at most once with each branch of the tree obtained by keeping only clusters in C_u (because a plausibility cluster cannot weight twice the same branch). As the tree obtained by keeping only clusters in C_u has at most $|U|$ different branches (because it has at most $|U|$ leaves), the number of connections between one cluster in C_p and one cluster in C_u is lesser than $|P| \cdot |U|$, which means that $\sum_{c \in C_u} |Sons_p(c)| \leq |P| \cdot |U|$.

As a result, $\sum_{c\in C}(|\Phi(c)|+|Sons(c)|-1)\leq|P|\cdot|U|+|U|+(|P|-1)+(|U|-1)+|P|\cdot|U|$, which implies that

$$\sum_{c\in C}(|\Phi(c)|+|Sons(c)|-1)\leq 2\cdot(1+|P|)\cdot(1+|U|)=O((1+|P|)\cdot(1+|U|))$$

Thus, the time complexity is $O((1+|P|)\cdot(1+|U|)\cdot d^{1+w_{CNDAG}})$.

The space complexity is $O((|P\cup U|)\cdot d^{1+w_{CNDAG}})$ because the scope functions manipulated have a scope of size lesser than $1+w_{CNDAG}$. □

Proof of Theorem 7.49 (page 135). Let $o\in lin(\preceq_{Sov})$. We denote by $\Pi_k(o)$ the set of potentials obtained at step k with the elimination order o. More precisely, $\Pi_0(o)=\{(P_i,1_u)|P_i\in P\}\cup\{(1_p,U_i)|U_i\in U\}$ and if $x=o(k)$ is the kth variable eliminated in o, $\Pi_{k+1}(o)=(\Pi_k(o)-\Pi_k(o)^{+x})\cup\{\pi^c_{k+1}(o)\}$, where $\pi^c_{k+1}(o)$ is the potential created from step k to step $k+1$ and equal to $\pi^c_{k+1}(o)=op(x)(\boxtimes_{\pi\in\Pi_k(o)^{+x}}\pi)$.

Let us show that for all $k\in\{0,\dots,|Sov|\}$, for all $(\emptyset,\otimes,N)\in Sons(CNDAG_k(Q,o))$, and for all $n\in N$, there exists $\pi\in\Pi_k(o)$ such that $sc(n)\subset sc(\pi)$.

The property holds for $k=0$. Indeed, let $(\emptyset,\times,N)\in Sons(CNDAG_0(Q,o))$ and let $n\in N$. Then, either $n=P_i\in P$ or $n=U_i\in U$. In the first case, $sc(n)\subset sc((P_i,1_u))$. In the second case, $sc(n)\subset sc((1_p,U_i))$.

Assume that the property holds at step k. Let $CNDAG_k(Q,o)=(sov.op_x,\oplus,\{(\emptyset,\otimes,N),N\in\mathfrak{N}\})$.

We analyze several cases, depending on the elimination performed at step k:

- Case $op_x=\oplus_x$

  Let $N\in\mathfrak{N}$. If no simplification is used, the computation node created from N is $(\emptyset,\otimes,N^{-x}\cup\{RR((\oplus_x,\otimes,N^{+x}))\})$.

  Let us show that for all $n\in N^{-x}\cup\{RR((\oplus_x,\otimes,N^{+x}))\}$, there exists $\pi\in\Pi_{k+1}(o)$ such that $sc(n)\subset sc(\pi)$.

  - Let $n\in N^{-x}$. Then, $\exists\pi\in\Pi_k(o),\ sc(n)\subset sc(\pi)$ (because the property holds at step k). Given that $x\notin sc(n)$, (1) either $x\notin sc(\pi)$: in this case, $\pi\in\Pi_{k+1}(o)$, (2) or $x\in sc(\pi)$: in this case, π is combined with other potentials to give $\pi^c_{k+1}(o)\in\Pi_{k+1}(o)$, and $(sc(\pi)-\{x\})\subset sc(\pi^c_{k+1}(o))$; as $sc(n)\subset sc(\pi)$ and $x\notin sc(n)$, it follows that $sc(n)\subset sc(\pi^c_{k+1}(o))$.

    In both cases, $\exists\pi\in\Pi_{k+1}(o),\ sc(n)\subset sc(\pi)$.

  - Let $n=RR((\oplus_x,\otimes,N^{+x}))$. For all $n'\in N^{+x}$, there exists $\pi(n')\in\Pi_k(o)$ such that $sc(n')\subset sc(\pi(n'))$ (and namely $x\in sc(\pi(n'))$). The potential created at step $k+1$ looks like $\pi^c_{k+1}(o)=\boxplus_x\left(\boxtimes_{\pi\in\Pi_k(o)^{+x}}\pi\right)$. As $\{\pi(n'),n'\in N^{+x}\}\subset\Pi_k(o)^{+x}$, this entails that $(sc(N^{+x})-\{x\})\subset sc(\pi^c_{k+1}(o))$, i.e. $sc(n)\subset sc(\pi^c_{k+1}(o))$. If the simplification rule is used, then the property still holds because function *simplify* can only remove variables from a scope.

- Case $op_x=\max_x$

Let us first analyze the sons of the root of $CNDAG_k(Q,o)$ non impacted by $DR_{\max}$, which look like $(\emptyset, \otimes, N)$ with $x \notin sc(N)$. Let $n \in N$. Then, $\exists \pi \in \Pi_k(o)$, $sc(n) \subset sc(\pi)$. In $\Pi_{k+1}(o)$, either π is still here or it has been combined with other potentials to give $\pi^c_{k+1}(o)$. In both cases, there exists $\pi' \in \Pi_{k+1}(o)$ such that $sc(n) \subset sc(\pi')$.

Next, we analyze the node which may be created to eliminate x, which looks like $(\emptyset, \times, N_1 \cup \{RR_{\max}((\max_x, \oplus, \{(\emptyset, \otimes, N - N_1), N \in \mathfrak{N}^{+x}\}))\})$.

- If $n \in N_1$, then a reasoning similar to the previous one enables to prove that there exists $\pi \in \Pi_{k+1}(o)$ such that $sc(n) \subset sc(\pi)$.
- If $n = RR_{\max}((\max_x, +, \{(\emptyset, \times, N - N_1), N \in \mathfrak{N}^{+x}\}))$.

  We know that $sc(n) = sc(\{u(N - N_1), N \in \mathfrak{N}^{+x}\}) - \{x\} = sc(\{u(N), N \in \mathfrak{N}^{+x}\}) - \{x\}$, thanks to Proposition 7.44. For each $N \in \mathfrak{N}^{+x}$, there exists $\pi \in \Pi_k(o)$ such that $sc(u(N)) \subset sc(\pi)$, thanks to the recurrence assumption. Moreover, $x \in sc(u(N))$, since otherwise, as $x \notin sc(P(N))$, this would contradict $N \in \mathfrak{N}^{+x}$. This implies that $sc(\{u(N), N \in \mathfrak{N}^{+x}\}) \subset sc(\Pi_k(o)^{+x})$, hence $sc(n) \subset sc(\pi^c_{k+1}(o))$.

Therefore, the property holds at step $k+1$.

Then, let $o^* \in lin(\preceq_{Sov})$ be an elimination order such that $w_{\mathcal{G}}(\preceq_{Sov}) = w_{\mathcal{G}}(o^*)$. If the cluster-tree decompositions transforming $CNDAG(Q) = CNDAG(Q, o^*)$ into a MCDAG use the elimination order given by o^* (which is always possible), then, according to the previous result, we now that the width w of this MCDAG satisfies $w \leq \max_{k \in \{0,\ldots,|Sov|-1\}} |sc(\pi^c_{k+1}(o^*))|$, and therefore that $w \leq w_{\mathcal{G}}(\preceq_{Sov})$. As $w_{CNDAG(Q)} \leq w$, this entails that $w_{CNDAG(Q)} \leq w_{\mathcal{G}}(\preceq_{Sov})$. □

B.6 Proofs of Chapter 8

Proof of Proposition 8.5 (page 144). Item (a) holds because by definition of $val(c, A, V, \Phi)$, we have $val(r, \emptyset, V(r), \Phi(r)) = val(r) = Ans(Q)$.

Let $x \in V$. Then,

$$\begin{aligned} & val(c, A, V, \Phi) \\ = \; & \oplus^c{}_V \left((\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s \in Sons(c)}\, val(s)(A)\right)\right) \\ = \; & \oplus^c{}_x \oplus^c{}_{V-\{x\}} \left((\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s \in Sons(c)}\, val(s)(A)\right)\right) \\ = \; & \oplus^c{}_x \left((\otimes^c{}_{\varphi \in \Phi_0}\, \varphi(A)) \otimes^c \left(\oplus^c{}_{V-\{x\}} \left((\otimes^c{}_{\varphi \in \Phi - \Phi_0}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s \in Sons(c)}\, val(s)(A)\right)\right)\right)\right) \\ = \; & \oplus^c{}_{a \in dom(x)} \left((\otimes^c{}_{\varphi \in \Phi_0}\, \varphi(A.(x,a))) \otimes^c val(c, A.(x,a), V - \{x\}, \Phi - \Phi_0)\right) \end{aligned}$$

Therefore, item (b) holds. Last, item (c) holds because by definition of $val(s)(A)$, we have $val(s)(A) = val(s, A, V(s) - V(c), \Phi(s))$. □

Proof of Proposition 8.6 (page 145). Directly entailed by Proposition 8.5. □

Proof of Proposition 8.7 (page 145). Each cluster c is considered at most $\mu \cdot d^{\alpha_c}$ times, where μ is the number of paths from the root to c and α_c is the maximum number of variables appearing in such paths. The variables in c can be assigned with $d^{|V(c)|}$ assignments. For each of these assignments, $|\Phi(c)| + |Sons(c)| - 1$ combination operations must be performed.

Therefore, the global time complexity is $O(\sum_{c\in C}(\mu \cdot d^{\alpha_c} \cdot d^{|V(c)|} \cdot (|\Phi(c)| + |Sons(c)| - 1)))$. As $\alpha + |V(c)|$ is lesser than the height h of the MCDAG, this time complexity can also be written $O(\sum_{c\in C}(\mu \cdot d^h \cdot (|\Phi(c)| + |Sons(c)| - 1)))$.

As shown in the proofs of Propositions 7.26 and 7.48, $\sum_{c\in C}(|\Phi(c)|+|Sons(c)|-1)) \leq 2\cdot|P\cup U|$ in the semiring case and $\sum_{c\in C}(|\Phi(c)| + |Sons(c)| - 1) \leq 2 \cdot (1 + |P|) \cdot (1 + |U|)$ in the semigroup case, hence the argued time complexity.

The linear space complexity result is straightforward. Indeed, as the MCDAG is of height h, we need to record the current domain of at most h variables simultaneously. Hence, recording the stack of current domains is $O(h \cdot d)$. Recording $V - \{x\}$ or $V(s) - V(c)$ for each recursive call of TS-mcdag is also $O(h)$, and recording $\Phi - \Phi_0$ or $\Phi(s)$ for each recursive call of TS-mcdag is $O(h \cdot m)$. Recording the current assignment is also $O(h)$. As it can be shown that a given cluster has less than m sons, recording the set of unexplored sons of a cluster is $O(h \cdot m)$. In the end, the space complexity of TS-mcdag is $O(h \cdot (d + m))$. □

Proof of Proposition 8.8 (page 147). Directly entailed by Proposition 8.5, and by the fact that given a cluster c and a cluster $s \in Sons(c)$, $val(s)(A) = val(s)(A')$ for all assignments A, A' of c and its ascendants such that $A^{\downarrow c\cap s} = A'^{\downarrow c\cap s}$. □

Proof of Proposition 8.9 (page 147). Thanks to caching, the value of each cluster c is computed only once per assignment of its variables. There are at most d^{w+1} assignments of its variables. For each of these assignments, the cluster must perform $|\Phi(c)| + |Sons(c)| - 1$ combination operations. Therefore, the time complexity is $O((\sum_{c\in C}(|\Phi(c)| + |Sons(c)| - 1)) \cdot d^{w+1})$. The factor $\sum_{c\in C}(|\Phi(c)| + |Sons(c)| - 1)$ can be bounded as in the proof of Propositions 7.26 and 7.48, which provides the given time complexity.

The space complexity is given by the space required for caching. For each separator, at most d^s elements are recorded. Each of these elements takes a space $s+1$ (in order to record the assignment and its value). Finally, if the MCDAG contains N nodes, then there are $N-1$ separators. Therefore, the space complexity is $O(N \cdot s \cdot d^s)$. □

Proof of Lemma 8.12 (page 154). Let us assume that function *bound* is sound and complete, and that function *evalSons* is sound and complete for all clusters c of depth h. Let us assume that *evalClusterMax*$(c, A, V, \Phi, \mathcal{B})$ is called, where c is a cluster of height h. Does it returns an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$?

The answer is yes if $|V| = 0$, because if there are no more variables to assign in the current cluster (test $V = \emptyset$), then *evalClusterMax* returns *evalSons*$(c, A, \emptyset, \Phi, \mathcal{B})$, which is an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$ according to our initial hypothesis.

Assume that the answer is yes for all sets of variables of size k. Let us consider a set of variables V of size $k + 1$. In this case, the set V of unassigned variables is not empty. Let $x \in V$ and let $\Phi_0 = \{\varphi \in \Phi, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$ be the set of scoped functions in Φ whose scope will be assigned when x will be assigned. We can use the following formulas, which hold directly from Definition 8.4:

$$val(c, A, V, \Phi) \quad = \quad \max_{a\in dom(x)} val(c, A.(x, a), V - \{x\}, \Phi)$$

and, for all $a \in dom(x)$,

$$val(c, A.(x,a).V - \{x\}, \Phi) \quad = \quad \left(\underset{\varphi \in \Phi_0}{\otimes^c} \varphi(A,(x,a))\right) \otimes^c val(c, A.(x,a), V - \{x\}, \Phi - \Phi_0)$$

In order to compute an evaluation of $\max_{a \in dom(x)} val(c, A.(x,a).V - \{x\}, \Phi)$ bounded by $\mathcal{B}$, values in $dom(x)$ are considered stepwise. At each iteration of the while loop, d is the set of values of x which have not been considered yet.

Let us consider the following set of properties, denoted PW (properties at the beginning of each iteration of the while loop):

- (lb, ub) is an evaluation of $\max_{a' \in dom(x)-d} val(c, A.(x,a'), V - \{x\}, \Phi)$ bounded by $\mathcal{B}$
- $(LB' \succeq LB) \wedge ((LB' = LB) \vee (LB' = lb_{\otimes} \otimes lb \oplus lb_{\oplus}))$

PW holds before entering the while block, since at that point, we have $LB' = LB$ and $lb = ub = \perp = \max_{a' \in \emptyset} val(c, A.(x,a'), V - \{x\}, \Phi)$.

Assume that PW holds at the beginning of one iteration of the while loop. Let us prove that it holds at the end of the iteration of the while loop, i.e. that

- first, $(\max(lb, val_0 \otimes^c lb'), \max(ub, val_0 \otimes^c ub'))$ is an evaluation of $\max_{a' \in dom(x)-(d \cup \{a\})} val(c,$ $A.(x,a'),$ $V - \{x\},$ $\Phi)$ bounded by $\mathcal{B}$, where a is the value in d chosen during the iteration of the while loop;
- and second, $LB' \succeq LB$ and either $LB' = LB$, or $LB' = lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$.

It is straightforward that the second condition holds at the end of the while loop iteration, because the unique instruction updating LB' is "$LB' \leftarrow \max(LB', lb_{\otimes} \otimes lb \oplus lb_{\oplus})$", and it appears just after the instruction "$lb \leftarrow \max(lb, val_0 \otimes^c lb')$". Therefore, we only have to check whether the first condition is satisfied.

During the iteration of the while loop, $val_0 = \otimes^c{}_{\varphi \in \Phi_0} \varphi(A,(x,a))$ is computed. A lower bound lb' and an upper bound ub' on $val(c, A.(x,a).V - \{x\}, \Phi - \Phi_0)$ are computed thanks to function *bound*, and they can be updated by the call to *evalClusterMax*$(c, A.(x,a), V - \{x\}, \Phi - \Phi_0)$. As function bound is sound and complete and as $|V - \{x\}| = k$, one can infer that $lb' \preceq val(c, A.(x,a).V - \{x\}, \Phi - \Phi_0) \preceq ub'$. As $val(c, A.(x,a), V - \{x\}, \Phi) = val_0 \otimes^c val(c, A.(x,a).V - \{x\}, \Phi - \Phi_0)$, this implies that $val_0 \otimes^c lb'$ and $val_0 \otimes^c ub'$ are lower and upper bounds for $val(c, A.(x,a), V - \{x\}, \Phi)$.

Moreover, $lb \preceq \max_{a \in dom(x)-d} val(c, A, (x,a), V - \{x\}, \Phi) \preceq ub$ because of PW. This makes it possible to infer that $\max(lb, val_0 \otimes^c lb') \preceq \max_{a' \in dom(x)-(d-\{a\})} val(c, A, (x,a'), V - \{x\}, \Phi) \preceq \max(ub, val_0 \otimes^c lb')$. The main conclusion of this is that in order to prove that PW is satisfied at the end of the iteration of the while loop, it suffices to show that one of the following conditions hold:

(BE1) $\max(lb, val_0 \otimes^c lb') = \max(ub, val_0 \otimes^c lb')$;

(BE2) $lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} = ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$;

(BE3) $LB \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$;

(BE4) $UB \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$.

Let us analyze more finely an iteration of the while loop. The algorithm achieves some tests and may perform further computations concerning value a. Just after the "if" block, we have:

(a) if the conditions of the "if" block have not been satisfied, then this means that one of the following conditions holds:

- $val_0 \otimes^c lb' = val_0 \otimes^c ub'$;
- $lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $LB' \succeq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$
- $UB \preceq lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus$;

(b) if the conditions of the "if" block have been satisfied, then (lb', ub') is an evaluation of $val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0)$ bounded by $\mathcal{B}'$, because $|V - \{x\}| = k$.

If $\otimes^c = \otimes$, then $\mathcal{B}' = (LB', UB, val_0 \otimes lb_\otimes, val_0 \otimes ub_\otimes, lb_\oplus, ub_\oplus)$, and therefore one of the following conditions holds:

- $lb' = ub'$;
- $(val_0 \otimes lb_\otimes) \otimes lb' \oplus lb_\oplus = (val_0 \otimes ub_\otimes) \otimes ub' \oplus ub_\oplus$, i.e. $lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $LB' \succeq (val_0 \otimes ub_\otimes) \otimes ub' \oplus ub_\oplus$, i.e. $LB' \succeq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $UB \preceq (val_0 \otimes lb_\otimes) \otimes lb' \oplus lb_\oplus$, i.e. $UB \preceq lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus$.

If $\otimes^c = \oplus$, then $\mathcal{B}' = (LB', UB, lb_\otimes, ub_\otimes, lb_\oplus \oplus lb_\otimes \otimes val_0, ub_\oplus \oplus ub_\otimes \otimes val_0)$, and therefore one of the following conditions holds:

- $lb' = ub'$;
- $lb_\otimes \otimes lb' \oplus lb_\oplus \oplus lb_\otimes \otimes val_0 = ub_\otimes \otimes ub' \oplus ub_\oplus \oplus ub_\otimes \otimes val_0$, which can also be written $lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $LB' \succeq ub_\otimes \otimes ub' \oplus ub_\oplus \oplus ub_\otimes \otimes val_0$, i.e. $LB' \succeq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $UB \preceq lb_\otimes \otimes lb' \oplus lb_\oplus \oplus lb_\otimes \otimes val_0$, i.e. $UB \preceq lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus$.

Thus, in both cases ($\otimes^c = \otimes$ and $\otimes^c = \oplus$), one of the following conditions holds:

- $lb' = ub'$, and hence $val_0 \otimes^c lb' = val_0 \otimes^c ub'$;
- $lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $LB' \succeq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus$;
- $UB \preceq lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus$.

A synthesis of cases (a) and (b) shows that at the end of the "if" block, we have:

$$\begin{array}{l}(val_0 \otimes^c lb' = val_0 \otimes^c ub') \\ \vee(lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus) \\ \vee(LB' \succeq ub_\otimes \otimes (val_0 \otimes^c ub') \oplus ub_\oplus) \\ \vee(UB \preceq lb_\otimes \otimes (val_0 \otimes^c lb') \oplus lb_\oplus)\end{array} \tag{B.1}$$

As said previously, we also have:

$$val_0 \otimes^c lb' \preceq val(c, A.(x,a), V - \{x\}, \Phi) \preceq val_0 \otimes^c ub' \tag{B.2}$$

Moreover, as PW holds at the beginning of the while loop iteration, we have, before the update of lb and ub:

$$\begin{aligned}&(lb = ub)\\ &\vee(lb_\otimes \otimes lb \oplus lb_\oplus = ub_\otimes \otimes ub \oplus ub_\oplus)\\ &\vee(LB \succeq ub_\otimes \otimes ub \oplus ub_\oplus)\\ &\vee(UB \preceq lb_\otimes \otimes lb \oplus lb_\oplus)\end{aligned} \tag{B.3}$$

and

$$lb \preceq \max_{a' \in dom(x)-d} val(c, A.(x,a'), V - \{x\}, \Phi) \preceq ub \tag{B.4}$$

and

$$(LB' \succeq LB) \wedge ((LB' = LB) \vee (LB' = lb_\otimes \otimes lb \oplus lb_\oplus)) \tag{B.5}$$

In order to show that PW holds at the beginning of the next iteration of the while loop, let us prove that the conjunction of Equations B.1 to B.5 implies $BE1 \vee BE2 \vee BE3 \vee BE4$. We analyze different cases (we analyze the different cases provided by Equation B.1, and then subcases are analyzed by following Equation B.3):

1. Case $val_0 \otimes^c lb' = val_0 \otimes^c ub'$:

   Using Equation B.2, this implies that $val_0 \otimes^c lb' = val(c, A.(x,a), V - \{x\}, \Phi) = val_0 \otimes^c ub'$. We analyze the different cases given by Equation B.3:

   (a) If $lb = ub$

   Then, $\max(lb, val_0 \otimes^c lb') = \max(ub, val_0 \otimes^c ub')$, and hence BE1 holds.

   (b) If $lb_\otimes \otimes lb \oplus lb_\oplus = ub_\otimes \otimes ub \oplus ub_\oplus$

   We analyze two cases:

   - If $val_0 \otimes^c ub' \preceq ub$

     Then, one can write $lb_\otimes \otimes lb \oplus lb_\oplus = ub_\otimes \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_\oplus$ This implies that $lb_\otimes \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_\oplus \succeq ub_\otimes \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_\oplus$.

     In another direction, as $lb \preceq ub$, $lb_\otimes \preceq ub_\otimes$, $lb_\oplus \preceq ub_\oplus$, and $lb' \preceq ub'$, one can write $lb_\otimes \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_\oplus \preceq ub_\otimes \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_\oplus$.

     Hence, $lb_\otimes \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_\oplus = ub_\otimes \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_\oplus$, which shows that BE2 holds.

   - Otherwise, $val_0 \otimes^c ub' \succ ub$

     Then, $\max(ub, val_0 \otimes^c ub') = val_0 \otimes^c ub'$. Moreover, we also have $\max(lb, val_0 \otimes^c lb') = val_0 \otimes^c lb'$, because $val_0 \otimes^c lb' = val_0 \otimes^c ub' \succ ub \succeq lb$. This implies that $\max(lb, val_0 \otimes^c lb') = val_0 \otimes^c ub'$ too. In other words, $\max(lb, val_0 \otimes^c lb') = \max(ub, val_0 \otimes^c ub')$, i.e. BE1 holds.

(c) If $UB \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$

Then, $UB \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$, hence BE4 holds.

(d) If $LB \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}$

- If $val_0 \otimes^c ub' \preceq ub$

  Then, $LB \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$, and therefore BE3 holds.

- Otherwise, $val_0 \otimes^c ub' \succ ub$

  Then, $\max(ub, val_0 \otimes^c ub') = val_0 \otimes^c ub'$. Moreover, as $val_0 \otimes^c ub' = val_0 \otimes^c lb'$, one can write $val_0 \otimes^c lb' \succ ub \succeq lb$, hence $\max(lb, val_0 \otimes^c lb') = val_0 \otimes^c lb'$ and $\max(lb, val_0 \otimes^c lb') = \max(ub, val_0 \otimes^c ub')$. This implies that BE1 holds.

2. Case $lb_{\otimes} \otimes (val_0 \otimes^c lb') \oplus lb_{\oplus} = ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}$.

(a) If $lb = ub$

- If $val_0 \otimes^c ub' \preceq ub$

  Then $\max(ub, val_0 \otimes^c ub') = ub$. Moreover, as $val_0 \otimes^c lb' \preceq val_0 \otimes^c ub' \preceq ub = lb$, one can infer that $\max(lb, val_0 \otimes^c ub') = lb$. As $lb = ub$, this implies that $\max(lb, val_0 \otimes^c ub') = \max(ub, val_0 \otimes^c ub')$, hence BE1 holds.

- Otherwise, $val_0 \otimes^c ub' \succ ub$

  Then, $\max(ub, val_0 \otimes^c ub') = val_0 \otimes^c ub'$, and therefore $lb_{\otimes} \otimes (val_0 \otimes^c lb') \oplus lb_{\oplus} = ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. This implies that $lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$.

  As $lb \preceq ub$, $lb' \preceq ub'$, $lb_{\otimes} \preceq ub_{\otimes}$, and $lb_{\oplus} \preceq ub_{\oplus}$, the inverse inequality also holds. Thus, $lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} = ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$, i.e. BE2 holds.

(b) If $lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes ub \oplus lb_{\oplus}$

Then, BE2 holds because

$$\begin{aligned} lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} &= \max(lb_{\otimes} \otimes lb \oplus lb_{\oplus}, lb_{\otimes} \otimes (val_0 \otimes^c lb') \oplus lb_{\oplus}) \\ &= \max(ub_{\otimes} \otimes ub \oplus ub_{\oplus}, ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}) \\ &= ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus} \end{aligned}$$

(c) If $UB \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$

Then, $UB \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$, hence BE4 holds.

(d) If $LB \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}$

- If $val_0 \otimes^c ub' \preceq ub$

  Then, $LB \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$, i.e. BE3 holds.

- Otherwise, $val_0 \otimes^c ub' \succ ub$

  In this case, we have $lb_{\otimes} \otimes (val_0 \otimes^c lb') \oplus lb_{\oplus} = ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. This entails that $lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. As argued is some of the previous cases, the inverse inequality holds. Therefore, $lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} = ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$, hence BE2 holds.

3. Case $UB \preceq lb_{\otimes} \otimes (val_0 \otimes^c lb') \oplus lb_{\oplus}$

In this case, $UB \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$, i.e. BE4 holds.

4. Case $LB' \succeq ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}$

   Note that in this case, if $LB = LB'$ and $val_0 \otimes^c ub' \succeq ub$, then we have $LB \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$, hence BE3 holds.

   (a) If $lb = ub$

   - If $LB = LB'$
     If $val_0 \otimes^c ub' \succeq ub$, we have already proved that BE3 holds. Otherwise, $val_0 \otimes^c ub' \prec ub$. In this case, one can write first $\max(ub, val_0 \otimes^c ub') = ub$, and second $\max(lb, val_0 \otimes^c lb') = lb$, because $lb = ub \succ val_0 \otimes^c ub' \succeq val_0 \otimes^c lb'$. As $lb = ub$, this entails that $\max(ub, val_0 \otimes^c ub') = \max(lb, val_0 \otimes^c lb')$, hence BE1 holds.
   - Otherwise, $LB' = lb_{\otimes} \otimes lb \oplus lb_{\oplus}$
     Then, as $LB' \succeq ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}$, we have $ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus} \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus} \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$.
     If $val_0 \otimes^c ub' \succeq ub$, then we get $ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus} \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$. As argued in some previous cases, the inverse inequality is also satisfied. Therefore, $ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus} = lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$, which prove that BE2 holds.
     Otherwise, $val_0 \otimes^c ub' \prec ub$. In this case, $\max(ub, val_0 \otimes^c ub') = ub$. Moreover, $lb = ub \succ val_0 \otimes^c ub' \succeq val_0 \otimes^c lb'$. Thus, $\max(lb, val_0 \otimes^c lb') = lb$. As $lb = ub$, we get $\max(ub, val_0 \otimes^c ub') = \max(lb, val_0 \otimes^c lb')$, which means that BE1 is satisfied.

   (b) If $lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes ub \oplus ub_{\oplus}$

   - If $LB = LB'$
     If $val_0 \otimes^c ub' \succeq ub$, we have already proved that BE3 holds. Otherwise, $val_0 \otimes^c ub' \prec ub$. In this latter case, one can write $\max(ub, val_0 \otimes^c ub') = ub$, and therefore $lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. This implies that $lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus} \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. As previously, this enables us to conclude that BE2 holds.
   - Otherwise, $LB' = lb_{\otimes} \otimes lb \oplus ub_{\oplus}$
     Then, as $LB' \succeq ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}$, we have $ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus} \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$. Together with $lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes ub \oplus ub_{\oplus}$, this enables us to write: $\max(ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}, ub_{\otimes} \otimes ub \oplus ub_{\oplus}) \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$, i.e. $ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus} \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$, and therefore $ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus} \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$. As previously, this enables us to conclude that BE2 holds.

   (c) If $UB \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$

   Then, $UB \preceq lb_{\otimes} \otimes \max(lb, val_0 \otimes^c lb') \oplus lb_{\oplus}$, hence BE4 holds.

   (d) If $LB \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}$

   - If $LB' = LB$, then we have both $LB \succeq ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}$ and $LB \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}$, and therefore $LB \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. This implies that BE3 holds.

- Otherwise, $LB' = lb_{\otimes} \otimes lb \oplus lb_{\oplus}$. Then, we get $lb_{\otimes} \otimes lb \oplus lb_{\oplus} \succeq ub_{\otimes} \otimes (val_0 \otimes^c ub') \oplus ub_{\oplus}$. Moreover, as $LB' \succeq LB$, we also have $lb_{\otimes} \otimes lb \oplus lb_{\oplus} \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}$. Therefore, $lb_{\otimes} \otimes lb \oplus lb_{\oplus} \succeq ub_{\otimes} \otimes \max(ub, val_0 \otimes^c ub') \oplus ub_{\oplus}$. As previously, this enables us to conclude that BE2 holds.

We have proved that PW holds at the end of the while loop iteration. As there is a finite number of iterations of the while loop (because each variable has a finite domain), we obtain that the stopping conditions are satisfied at one iteration (after $|dom(x)|$ iterations, the test $d \neq \emptyset$ is false).

To conclude, let us prove that if one of the stopping conditions of the while loop is satisfied, then the algorithm returns an evaluation of $val(c, A.(x,a), V - \{x\}, \Phi)$ bounded by $\mathcal{B}$:

- If $LB' \succeq UB$, then $LB' \neq LB$ (because $LB \prec UB$). Hence $LB' = lb_{\otimes} \otimes lb \oplus lb_{\oplus}$, which implies that $UB \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$. Given that
$$\begin{array}{lll} lb & \preceq & \max_{a \in dom(x)-d} val(c, A.(x,a), V - \{x\}, \Phi) \\ & \preceq & \max_{a \in dom(x)} val(c, A.(x,a), V - \{x\}, \Phi) = val(c, A, V, \Phi) \end{array}$$
it suffices to return lb as a lower bound (case 4 of the definition of a bounded evaluation). Moreover, if $d = \emptyset$, then, as PW holds, $\max_{a \in dom(x)} val(c, A.(x,a), V - \{x\}, \Phi) \preceq ub$, i.e. $val(c, A, V, \Phi) \preceq ub$. In this case, the pair (lb, ub) returned by the algorithm is an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. Otherwise, if $d \neq \emptyset$, the algorithm returns $(lb, \top)$, which is also an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$.

- If $lb = \top$, then, as $lb \preceq ub$, one can infer that $lb = ub = \top$. Moreover, as
$$\begin{array}{lll} lb & \preceq & \max_{a \in dom(x)-d} val(c, A, (x,a), V - \{x\}, \Phi) \\ & \preceq & \max_{a \in dom(x)-d} val(c, A, (x,a), V - \{x\}, \Phi) \end{array}$$
this also implies that $\top \preceq val(c, A, V, \Phi)$. As a result, we have $lb = ub = val(c, A, V, \Phi) = \top$, hence the pair (lb, ub) returned by the algorithm is a bounded evaluation of $val(c, A, V, \Phi)$ with $\mathcal{B}$ as a bound (case 1 in the definition of a bounded evaluation).

- If $d = \emptyset$, then the algorithm returns (lb, ub), which is an evaluation of $\max_{a \in dom(x)} val(c,$ $A.(x,a)$, $V - \{x\}$, $\Phi)$ bounded by $\mathcal{B}$ because PW holds.

As a result, *evalClusterMax*$(c, A, V, \Phi, \mathcal{B})$ returns an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$ if $|V| = k{+}1$. By recurrence, this proves that whatever the size of V is, *evalClusterMax*$(c, A, V, \Phi, \mathcal{B})$ returns an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. □

Proof of Lemma 8.13 (page 154). The proof is the similar to the proof concerning *evalClusterMax*. □

Proof of Lemma 8.14 (page 154). Let us assume that function *bound* is sound and complete, and that function *evalSons* is sound and complete for all clusters c of depth h, Let us assume that *evalClusterPlus*$(c, A, V, \Phi, \mathcal{B})$ is called, where c is a cluster of depth h. Does it return an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$?

The answer is yes if $|V| = 0$, because if there are no more variables to assign in the current cluster (test $V = \emptyset$), then *evalClusterPlus* returns *evalSons*$(c, A, \emptyset, \Phi, \mathcal{B})$, which is an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$ according to our initial hypothesis.

Assume that the answer is yes for all sets of variables of size k. Let us consider a set of variables V of size $k+1$. In this case, the set V of unassigned variables is not empty. Let $x \in V$ and let $\Phi_0 = \{\varphi \in \Phi, sc(\varphi) \cap (V - \{x\}) = \emptyset\}$ be the set of scoped functions in Φ whose scope will be assigned when x will be assigned. We can use the following formulas, which hold directly from Definition 8.4:

$$val(c, A, V, \Phi) \quad = \quad \oplus_{a \in dom(x)} val(c, A.(x,a).V - \{x\}, \Phi)$$

and, for all $a \in dom(x)$,

$$val(c, A.(x,a), V - \{x\}, \Phi) \quad = \quad \left(\underset{\varphi \in \Phi_0}{\otimes} \varphi(A, (x,a))\right) \otimes val(c, A.(x,a).V - \{x\}, \Phi - \Phi_0)$$

In order to compute an evaluation of $\oplus_{a \in dom(x)} val(c, A.(x,a).V - \{x\}, \Phi)$ bounded by $\mathcal{B}$, values in $dom(x)$ are considered stepwise. At each iteration of the while loop, d is the set of values of x which have not been considered yet.

Using function *bound*, the algorithm first computes, for each $a \in dom(x)$, lower and upper bounds $tablb[a]$ and $tabub[a]$ such that $tablb[a] \preceq val(c, A.(x,a).V - \{x\}, \Phi) \preceq tabub[a]$. Then, it computes the subset d_0 of values a in $dom(x)$ such that $tablb[a] = tabub[a]$. For each $a \in d_0$, we then have $tablb[a] = tabub[a] = val(c, A.(x,a).V - \{x\}, \Phi)$, hence $val(c, A.(x,a).V - \{x\}, \Phi)$ is known. The other values are gathered in $d = dom(x) - d_0$. After these steps, the algorithm initializes res by $res = \oplus_{a \in dom(x)-d} val(c, A.(x,a).V - \{x\}, \Phi)$, lb by $lb = res \oplus (\oplus_{a \in d} tablb[a]) = \oplus_{a \in dom(x)} tablb[a]$ and ub by $ub = res \oplus (\oplus_{a \in d} tabub[a]) = \oplus_{a \in dom(x)} tabub[a]$. It is straightforward that lb and ub are respectively lower and upper bounds on $val(c, A, V, \Phi)$.

If $d = \emptyset$ before the whole while block is processed, then it is straightforward that $lb = ub = val(c, A, V, \Phi)$. In this case, the while loop is not processed and the pair (lb, ub) returned is a bounded evaluation of $val(c, A, V, \Phi)$.

Otherwise, there is at least one value in d before processing the whole while loop. Let us show that at each iteration of the while loop,

$$\begin{aligned} &((lb, ub) \text{ is an evaluation of } val(c, A, V, \Phi) \text{ bounded by } \mathcal{B}) \\ &\qquad \vee (res = \oplus_{a' \in dom(x)-d} val(c, A.(x,a'), V, \Phi)) \end{aligned} \tag{B.6}$$

This property is denoted PW.

PW holds before entering the while block, because $res = \oplus_{a' \in dom(x)-d} val(c, A.(x,a'), V, \Phi)$.

Assume that PW holds at the beginning of an iteration of the while loop. As an iteration of the while loop is performed, none of its stopping conditions is satisfied. This exactly means that (lb, ub) is not an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. As PW holds, this means that $res = \oplus_{a' \in dom(x)-d} val(c, A.(x,a'), V - \{x\}, \Phi)$ at the beginning of this iteration.

At each iteration of the while loop, d is the set of values in $dom(x)$ which have not been considered yet. Let a be a value in d. As $V - \{x\}$ contains k variables, (lb_a, ub_a) is an evaluation of $val(c, A.(x,a), V - \{x\}, \Phi)$ bounded by $\mathcal{B}'$. This means that first, $lb_a \preceq val(c, A.(x,a), V - \{x\}, \Phi) \preceq ub_a$, and second,

$(lb_a = ub_a)$
$\vee(lb_\otimes \otimes val_0 \otimes lb_a \oplus lb_\oplus \oplus lb_\otimes \otimes lb_{\neg a} = ub_\otimes \otimes val_0 \otimes ub_a \oplus ub_\oplus \oplus ub_\otimes \otimes ub_{\neg a})$
$\vee(UB \preceq lb_\otimes \otimes val_0 \otimes lb_a \oplus lb_\oplus \oplus lb_\otimes \otimes lb_{\neg a})$
$\vee(LB \succeq ub_\otimes \otimes val_0 \otimes ub_a \oplus ub_\oplus \oplus ub_\otimes \otimes ub_{\neg a})$
that is to say
$(lb_a = ub_a)$
$\vee(lb_\otimes \otimes (val_0 \otimes lb_a \oplus lb_{\neg a}) \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes ub_a \oplus ub_{\neg a}) \oplus ub_\oplus)$
$\vee(UB \preceq lb_\otimes \otimes (val_0 \otimes lb_a \oplus lb_{\neg a}) \oplus lb_\oplus)$
$\vee(LB \succeq ub_\otimes \otimes (val_0 \otimes ub_a \oplus ub_{\neg a}) \oplus ub_\oplus)$

The algorithm uses instructions which enable us to write: $val_0 \otimes lb_a \oplus lb_{\neg a} \preceq val(c, A, V, \Phi) \preceq val_0 \otimes lb_a \oplus lb_{\neg a}$. Therefore, at the end of each iteration of the while loop, we have, after the update of lb and ub, $lb \preceq val(c, A, V, \Phi) \preceq ub$.

We then analyze four cases:

1. Case $lb_a = ub_a$

   In this case, we have $lb_a = ub_a = val(c, A.(x, a), V - \{x\}, \Phi - \Phi_0)$, and therefore $val_0 \otimes lb_a = val(c, A.(x, a), V - \{x\}, \Phi)$. Hence, we have
   $$\begin{aligned} res \oplus val_0 \otimes lb_a &= (\oplus_{a' \in dom(x)-d} val(c, A.(x, a'), V, \Phi)) \oplus val(c, A.(x, a), V - \{x\}, \Phi) \\ &= \oplus_{a' \in dom(x)-(d-\{a\})} val(c, A.(x, a'), V, \Phi) \end{aligned}$$
   Thanks to the instruction "$res \leftarrow res \oplus val_0 \otimes lb_a$", this implies that PW holds at the end of the iteration of the while loop.

2. Case $lb_\otimes \otimes (val_0 \otimes lb_a \oplus lb_{\neg a}) \oplus lb_\oplus = ub_\otimes \otimes (val_0 \otimes ub_a \oplus ub_{\neg a}) \oplus ub_\oplus$

   In this case, $(lb, ub) = (val_0 \otimes lb_a \oplus lb_{\neg a}, val_0 \otimes ub_a \oplus ub_{\neg a})$ is directly an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$.

3. Case $UB \preceq lb_\otimes \otimes (val_0 \otimes lb_a \oplus lb_{\neg a}) \oplus lb_\oplus$

   In this case, $(lb, ub) = (val_0 \otimes lb_a \oplus lb_{\neg a}, val_0 \otimes ub_a \oplus ub_{\neg a})$ is directly an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$.

4. Case $LB \succeq ub_\otimes \otimes (val_0 \otimes ub_a \oplus ub_{\neg a}) \oplus ub_\oplus$

   In this case, $(lb, ub) = (val_0 \otimes lb_a \oplus lb_{\neg a}, val_0 \otimes ub_a \oplus ub_{\neg a})$ is directly an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$.

Therefore, PW holds at the end of the iteration of the while loop.

If one of the stopping conditions of the while loop is satisfied, then this exactly means that (lb, ub) is an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$.

Otherwise, assume that none of the stopping conditions is satisfied before the last value a in d is eliminated. As none of the stopping conditions is satisfied before this iteration, (lb, ub) is not an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. As PW holds, this means that $res = \oplus_{a' \in dom(x)-\{a\}} val(c, A.(x, a), V - \{x\}, \Phi)$ at the beginning of this iteration. Then, we get
$(lb_{\neg a}, ub_{\neg a})$
$= (res, res)$
$= (\oplus_{a' \in dom(x)-\{a\}} val(c, A.(x, a'), V - \{x\}, \Phi), \oplus_{a' \in dom(x)-\{a\}} val(c, A.(x, a'), V - \{x\}, \Phi))$

- If $lb_a = ub_a$, then $val_0 \otimes lb_a = val_0 \otimes ub_a = val(c, A.(x,a), V - \{x\}, \Phi)$. We therefore get $(lb, ub) = (lb_{\neg a} \oplus val_0 \otimes lb_a, ub_{\neg a} \oplus val_0 \otimes ub_a) = (\oplus_{a' \in dom(x)} val(c, A.(x,a'), V - \{x\}, \Phi), \oplus_{a' \in dom(x)} val(c, A.(x,a'), V - \{x\}, \Phi))$. This implies that after the treatment of the last value in d, we have $lb = ub$, hence the while loop is stopped at the next iteration.
- In the other cases, the previous part of the proof shows that the while loop is stopped at the next iteration.

This proves that there is a finite number of iterations of the while loop (even if we do not have a test like $d \neq \emptyset$), and therefore the algorithm is complete. It is also sound because as previously said, once one of the conditions of the while loop is not satisfied, (lb, ub) is an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. □

Proof of Lemma 8.15 (page 155). Let c be a cluster of maximal depth. Then, $Sons(c) = \emptyset$, and the initializations of S and S_0 give $S_0 = S = \emptyset$. This implies that lb and ub are initialized with $(lb, ub) = ({\otimes^c}_{\varphi \in \Phi}\, \varphi(A), {\otimes^c}_{\varphi \in \Phi}\, \varphi(A))$.

As $lb = ub$, the while loop is not traversed. Moreover, by definition of $val(c, A, \emptyset, \Phi)$, one can write $val(c, A, \emptyset, \Phi) = {\otimes^c}_{\varphi \in \Phi}\, \varphi(A)$. Therefore $lb = ub = val(c, A, \emptyset, \Phi)$, which proves that (lb, ub) is a bounded evaluation of $val(c, A, \emptyset, \Phi)$. □

Proof of Lemma 8.16 (page 155). Let us assume that function *bound* is sound and complete and that *evalClusterMin*, *evalClusterMax*, *evalClusterPlus*, and *bound* are sound and complete for all clusters c of depth h. Let c be a cluster of depth $h - 1$.

We can use the following formula:

$$val(c, A, \emptyset, \Phi) \quad = \quad \left(\underset{\varphi \in \Phi}{\otimes^c}\, \varphi(A)\right) \otimes^c \left(\underset{s \in Sons(c)}{\otimes^c}\, val(s)(A)\right) \tag{B.7}$$

Clusters in $Sons(s)$ are considered stepwise. The algorithm first computes the set of son clusters S_0 such that for each $s \in S_0$, $val(s)(A)$ is known and equals $LB(s, A^{\downarrow s})$. The other son clusters are gathered in $S = Sons(c) - S_0$. This entails that res is actually initialized by $res = ({\otimes^c}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c ({\otimes^c}_{s \in S_0}\, val(s)(A))$.

Moreover, thanks to Eq. B.7, $lb = res \otimes^c ({\otimes^c}_{s' \in S}\, LB(s, A^{\downarrow s}))$ and $ub = res \otimes^c ({\otimes^c}_{s' \in S}\, UB(s, A^{\downarrow s}))$ are respectively lower and upper bounds on $val(c, A, \emptyset, \Phi)$.

If $S = \emptyset$ before the whole while block is processed, then it is straightforward that $lb = ub = val(c, A, \emptyset, \Phi)$. In this case, the while loop is not processed and the pair (lb, ub) returned is a bounded evaluation of $val(c, A, \emptyset, \Phi)$.

Otherwise, there is at least one son cluster in S before processing the whole while loop. Let us show that at each iteration of the while loop,

$$\begin{gathered}((lb, ub) \text{ is an evaluation of } val(c, A, \emptyset, \Phi) \text{ bounded by } \mathcal{B}) \\ \vee (res = \left(\underset{\varphi \in \Phi}{\otimes^c}\, \varphi(A)\right) \otimes^c \left(\underset{s' \in Sons(c) - S}{\otimes^c}\, val(s')(A)\right))\end{gathered} \tag{B.8}$$

This property is denoted PW.

PW holds before entering the while block, since $res = ({\otimes^c}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \big({\otimes^c}_{s' \in Sons(c) - S}\, val(s)(A)\big)$.

Assume that PW holds at the beginning of an iteration of the while loop. As an iteration of the while loop is performed, none of its stopping conditions is satisfied. This exactly means that (lb, ub) is not an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$. As PW holds, this means that $res = ({\otimes^c}_{\varphi\in\Phi}\,\varphi(A)) \otimes^c ({\otimes^c}_{s'\in Sons(s)-S}\,val(s')(A))$ at the beginning of this iteration. At each iteration of the while loop, S is the set of son clusters of c which have not been considered yet. Let s be a son cluster in S.

As *evalClusterMin*, *evalClusterMax*, *evalClusterPlus*, and *bound* are assumed to be sound and complete for clusters of depth h, (lb_s, ub_s) is an evaluation of $val(s, A, V(s) - V(c), \Phi(s))$ bounded by $\mathcal{B}'$, i.e. (lb_s, ub_s) is an evaluation of $val(s)(A)$ bounded by $\mathcal{B}'$. This means that first, $lb_s \preceq val(s)(A) \preceq ub_s$, and second,

- If $\otimes^c = \otimes$,
  $(lb_s = ub_s)$
  $\vee(lb_{\neg s} \otimes lb_\otimes \otimes lb_s \oplus lb_\oplus = ub_{\neg s} \otimes ub_\otimes \otimes ub_s \oplus ub_\oplus)$
  $\vee(UB \preceq lb_{\neg s} \otimes lb_\otimes \otimes lb_s \oplus lb_\oplus)$
  $\vee(LB \succeq ub_{\neg s} \otimes ub_\otimes \otimes ub_s \oplus ub_\oplus)$
  that is to say
  $(lb_s = ub_s)$
  $\vee(lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus)$
  $\vee(UB \preceq lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus)$
  $\vee(LB \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus)$
- If $\otimes^c = \oplus$,
  $(lb_s = ub_s)$
  $\vee(lb_\otimes \otimes lb_s \oplus lb_\oplus \oplus lb_\otimes \otimes lb_{\neg s} = ub_\otimes \otimes ub_s \oplus ub_\oplus \oplus ub_\otimes \otimes ub_{\neg s})$
  $\vee(UB \preceq lb_\otimes \otimes lb_s \oplus lb_\oplus \oplus lb_\otimes \otimes lb_{\neg s})$
  $\vee(LB \succeq ub_\otimes \otimes ub_s \oplus ub_\oplus \oplus ub_\otimes \otimes ub_{\neg s})$
  that is to say
  $(lb_s = ub_s)$
  $\vee(lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus)$
  $\vee(UB \preceq lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus)$
  $\vee(LB \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus)$

Therefore, in both cases, we have
$(lb_s = ub_s)$
$\vee(lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus)$
$\vee(UB \preceq lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus)$
$\vee(LB \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus)$

After the computation of (lb_s, ub_s), *evalSons* uses instructions which enable us to write:

$$\max(lb_s, LB(s, A^{\downarrow s})) \otimes^c lb_{\neg s} \preceq val(c, A, \emptyset, \Phi) \preceq \min(ub_s, UB(s, A^{\downarrow s})) \otimes^c ub_{\neg s}$$

Therefore, at the end of each iteration of the while loop, we have, after the update of lb and ub, $lb \preceq val(c, A, \emptyset, \Phi) \preceq ub$. Moreover, the update of $LB(s, A^{\downarrow s})$ and $UB(s, A^{\downarrow s})$ is sound because it preserves the property that $LB(s, A^{\downarrow s})$ and $UB(s, A^{\downarrow s})$ are lower and upper bounds for $val(s)(A)$.

We then analyze four cases:

1. Case $lb_s = ub_s$

   In this case, we have $lb_s = ub_s = val(s)(A)$. Moreover, as $LB(s, A^{\downarrow s}) \preceq val(s)(A)$, $\max(lb_s, LB(s, A^{\downarrow s})) = lb_s = val(s)(A)$. Hence, one can write
   $$\begin{aligned} res \otimes^c \max(lb_s, LB(s, A^{\downarrow s})) &= (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c (\otimes^c{}_{s' \in Sons(s)-S}\, val(s')(A)) \otimes^c val(s)(A) \\ &= (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s' \in Sons(c)-(S-\{s\})}\, val(s')(A)\right) \end{aligned}$$
   This implies that PW holds at the end of the iteration of the while loop.

2. Case $lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus$

   - If $ub_s \preceq UB(s, A^{\downarrow s})$, then this implies that $lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))) \oplus ub_\oplus$, and therefore $lb_\otimes \otimes (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s})) \oplus lb_\oplus \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))) \oplus ub_\oplus$. This inverse inequality being straightforwardly satisfied, we get $lb_\otimes \otimes (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s})) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))) \oplus ub_\oplus$.

     Hence, $(lb, ub) = (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s})), ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))$ is an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$.

   - Otherwise, $ub_s \succ UB(s, A^{\downarrow s})$.

     Then, one can infer that $ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c UB(s, A^{\downarrow s})) \oplus ub_\oplus \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c val(s)(A)) \oplus ub_\oplus \succeq lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus$.

     As $lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus$, this implies that $lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))) \oplus ub_\oplus$

     Also, this enables us to infer that $lb_\otimes \otimes (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s}))) \oplus lb_\oplus \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))) \oplus ub_\oplus$. The inverse inequality being easily satisfied, we obtain $lb_\otimes \otimes (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s}))) \oplus lb_\oplus = ub_\otimes \otimes (ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))) \oplus ub_\oplus$, and therefore $(lb, ub) = (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s})), ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))$ is an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$.

3. Case $UB \preceq lb_\otimes \otimes (lb_{\neg s} \otimes^c lb_s) \oplus lb_\oplus$

   In this case, $(lb, ub) = (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s})), ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))$ is directly an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$.

4. Case $LB \succeq ub_\otimes \otimes (ub_{\neg s} \otimes^c ub_s) \oplus ub_\oplus$

   In this case, $(lb, ub) = (lb_{\neg s} \otimes^c \max(lb_s, LB(s, A^{\downarrow s})), ub_{\neg s} \otimes^c \min(ub_s, UB(s, A^{\downarrow s}))$ is directly an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$.

Therefore, PW holds at the end of the iteration of the while loop.

If one of the stopping conditions of the while loop is satisfied, then this exactly means that (lb, ub) is an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$.

Otherwise, assume that none of the stopping conditions is satisfied before the last son $s \in S$ is considered. As none of the stopping conditions is satisfied before this iteration, (lb, ub) is not an evaluation of $val(c, A, \emptyset, \Phi)$ bounded by $\mathcal{B}$. As PW holds, this means that $res = (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s' \in Sons(c)-\{s\}}\, val(s')(A)\right)$ at the beginning of this iteration.

After the instruction $S \leftarrow S - \{s\}$, we get $(lb_{\neg s}, ub_{\neg s}) = (res, res) = ((\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s' \in Sons(c)-\{s\}}\, val(s')(A)\right), (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s' \in Sons(c)-\{s\}}\, val(s')(A)\right))$.

- If $lb_s = ub_s$, then $lb_s = ub_s = val(s)(A)$. We therefore get $(lb, ub) = (lb_{\neg s}\otimes^c lb_s, ub_{\neg s}\otimes^c ub_s) = \left((\otimes^c{}_{\varphi\in\Phi}\,\varphi(A)) \otimes^c \left(\otimes^c{}_{s'\in Sons(c)}\, val(s')(A)\right), (\otimes^c{}_{\varphi\in\Phi}\,\varphi(A)) \otimes^c \left(\otimes^c{}_{s'\in Sons(c)}\, val(s')(A)\right)\right)$.

  This implies that after the treatment of the last son in $Sons(c)$, we have $lb = ub$

- In the other cases, the previous part of the proof shows that one of the stopping conditions of the while is necessarily fulfilled.

This proves that there is a finite number of iterations of the while loop (even if we do not have a test like $d \neq \emptyset$), and therefore the algorithm is complete. It is also sound because as previously said, once one of the conditions of the while loop is not satisfied, (lb, ub) is an evaluation of $val(c, A, V, \Phi)$ bounded by $\mathcal{B}$. □

Proof of Lemma 8.17 (page 155). Let us assume that function *bound* is sound and complete. Thanks to Lemma 8.15, the result holds for clusters of maximal depth.

If *evalSons* is sound and complete for all clusters of depth h, then, using Lemmas 8.12, 8.13, and 8.14, one can infer that *evalClusterMin*, *evalClusterMax*, and *evalClusterPlus* are sound and complete for all clusters c of depth h. Thanks to Lemma 8.16, *evalSons* is sound and complete for all clusters of depth $h-1$. By recurrence, this proves that *evalSons* is sound and complete as soon as function *bound* is sound and complete, . □

Proof of Theorem 8.18 (page 155). Let us assume that function *bound* is sound and complete. Let r be the root of the MCDAG. Thanks to Lemma 8.17, the algorithm returns an evaluation of $val(r, \emptyset, V(r), \Phi(r)) = Ans(Q)$ bounded by $(\bot^-, \top^+, 1_E, 1_E, 0_E, 0_E))$, i.e. it returns a pair $(lb, ub) \in E^2$ such that $lb \preceq Ans(Q) \preceq ub$ and $(lb = ub) \vee (1_E \otimes lb \oplus 0_E = 1_E \otimes ub \oplus 0_E) \vee (\bot^- \succeq 1_E \otimes ub \oplus 0_E) \vee (\top^+ \preceq 1_E \otimes lb \oplus 0_E)$, i.e. such that $(lb = ub) \vee (lb = ub) \vee (\bot^- \succeq ub) \vee (\top^+ \preceq lb)$, i.e., as $(lb, ub) \in E^2$, such that $lb = ub$. Therefore, the algorithm returns $lb = Ans(Q)$. □

Proof of Proposition 8.19 (page 155). Basically, compared to algorithm **TS-mcdag**, using bounds does not change the worst case time complexity, because the values recorded are not the exact values of a cluster, hence a cluster can be revisited several times. As for the space complexity, BTD-mcdag uses twice as much space as RecTS-mcdag (because lower and upper bounds are recorded instead of exact values). But the complexity is still $O(N \cdot s \cdot d^s)$, where N is the number of clusters and s is the maximum size of the separators. □

Proof of Theorem 8.21 (page 158). Similar to the proof of Theorem 8.18. □

Proof of Proposition 8.22 (page 161). These results are quite straightforward.

First, for all $A'' \in dom(S')$, $\max_{A\in dom(S)} \varphi(A.A'') \succeq \max_{A\in dom(S)} \min_{A'\in dom(S')} \varphi(A.A')$, hence $\min_{A''\in dom(S')} \max_{A\in dom(S)} \varphi(A.A'') \succeq \max_{A\in dom(S)} \min_{A'\in dom(S')} \varphi(A.A')$. In other words, one can write $\min_S \max_{S'} \varphi \succeq \max_S \max_{S'} \varphi$.

Second, for all $A'' \in dom(S)$, $\oplus_{A'\in dom(S')}\varphi(A''.A') \preceq \oplus_{A'\in dom(S')} \max_{A\in dom(S)} \varphi(A.A')$, hence $\max_{A''\in dom(S)} \oplus_{A'\in dom(S')}\varphi(A''.A') \preceq \oplus_{A'\in dom(S')} \max_{A\in dom(S)} \varphi(A.A')$. In other words, one can write $\max_S \oplus_{S'}\varphi \succeq \oplus_{S'} \max_S \varphi$.

The proof for $\oplus_S \min_{S'} \varphi \preceq \min_{S'} \oplus_S\varphi$ is similar. □

Proof of Proposition 8.23 (page 162). As $\varphi \preceq \max_c \varphi$ and as $\otimes$ is monotonic, it is possible to write $\oplus_c((\otimes_{P_i \in Fact(c)} P_i) \otimes \varphi) \preceq \oplus_c((\otimes_{P_i \in Fact(c)} P_i) \otimes (\max_c \varphi))$. By distributivity of $\otimes$ over $\oplus$, this implies that $\oplus_c((\otimes_{P_i \in Fact(c)} P_i) \otimes \varphi) \preceq (\oplus_c \otimes_{P_i \in Fact(c)} P_i) \otimes (\max_c \varphi) = 1_E \otimes (\max_c \varphi) = \max_c \varphi$.

Similarly, as $\min_c \varphi \preceq \varphi$, one can infer that $\min_c \varphi \preceq \oplus_c((\otimes_{P_i \in Fact(c)} P_i) \otimes \varphi)$. □

Proof of Proposition 8.24 (page 163). First, as $\otimes$ is monotonic and as $\varphi_2 \preceq \max_S \varphi_2$, one can write $\varphi_1 \otimes \varphi_2 \preceq \varphi_1 \otimes \max_S \varphi_2$. Maximizing over S leads to $\max_S(\varphi_1 \otimes \varphi_2) \preceq (\max_S \varphi_1) \otimes (\max_S \varphi_2)$.

The proofs for $\max_S(\varphi_1 \oplus \varphi_2)$, $\min_S(\varphi_1 \otimes \varphi_2)$, and $\min_S(\varphi_1 \oplus \varphi_2)$ are similar.

Finally, as $0_E = \min(E)$, it is possible to write $\varphi_2 \preceq \oplus_S \varphi_2$. By monotonicity of $\otimes$, this implies that $\varphi_1 \otimes \varphi_2 \preceq \varphi_1 \otimes (\oplus_S \varphi_2)$. Summing over S leads to the required result. □

Appendix C

Concrete problem example: deployment and maintenance of a constellation of satellites

So forth, the PFU framework has been illustrated by toy examples only. We give here the PFU formulation of a concrete real-life planning problem involving plausibilities, feasibilities, and utilities. The description of this problem as well as Figures C.1 and C.2 are directly taken from [61].

Problem description *Whatever its mission is (telecommunication, navigation, or observation), a constellation of satellites is made up of a specified number of spatially distributed satellites. All the satellites or at least a subset of them must be operational for the mission to be filled. If too few satellites are operational, the mission objectives will be only partially met. In general, several launches using various launcher types are necessary to deploy the constellation of satellites. These launches must be organized over time. Failures may also occur at any stage of the deployment, of the maintenance, and of the operational life of the constellation. So, the management of its deployment and of its maintenance must be able to anticipate these possible failures, as well as to react to them when they occur.*

Globally speaking, managing the deployment and the maintenance of a constellation consists in organizing the launches and the orbital transfers in order to deploy it as soon as possible and to maintain it as best as possible in its operational state.

More precisely, the constellations we consider are organized along several orbital planes (see Figure C.1). A specified number of operational satellites is necessary on each orbital plane. On each orbital plane, satellites may be either on an operational orbit, or on a spare orbit. Satellites that are on a spare orbit are drifting in a month from an orbital plane to the following one. Launchers are able to put a specified number of satellites on one of the orbital planes (all the launched satellites on the same orbital plane). These satellites can be either immediately transferred from the spare orbit to the operational one on this orbital plane, or left on the spare orbit to drift from orbital plane to orbital plane. In the later case, when their orbital plane coincides with an operational orbital plane, that is once per month, they may be transferred from the spare orbit to the operational one

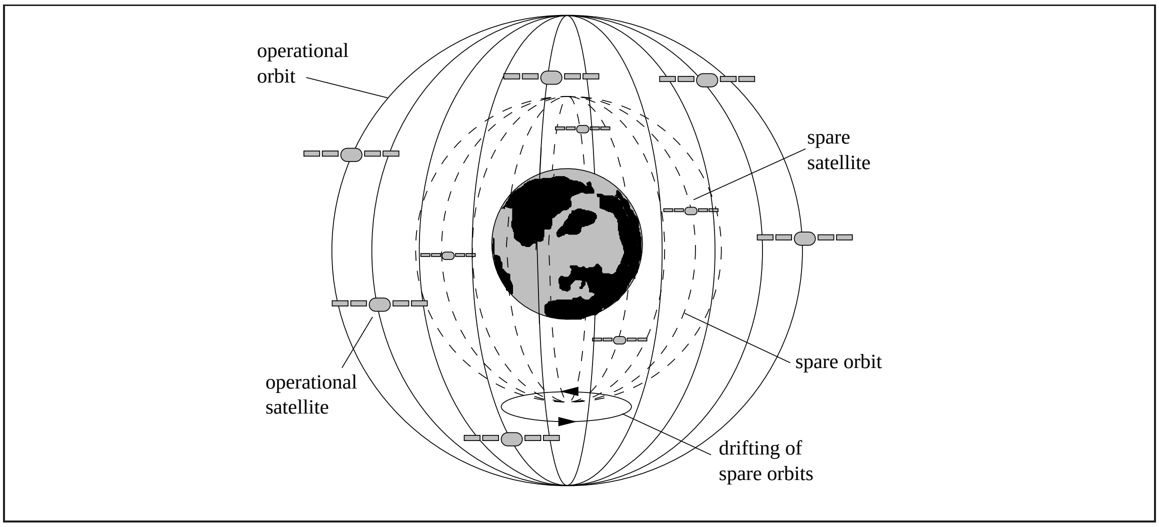


Figure C.1: View of the goal constellation.

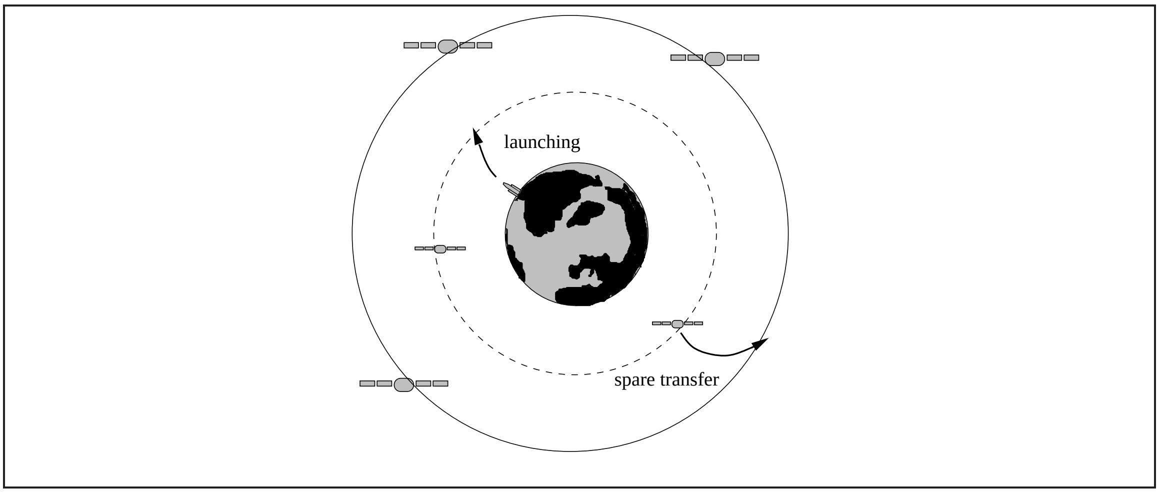


Figure C.2: On an orbital plane, launch of a satellite and transfer of a spare satellite from the spare orbit to the operational one.

on this orbital plane (see Figure C.2).

Launches are not possible at any time. We consider that no more than one launch is possible each month and that there exists a minimum time between two launches of the same type. Moreover, the management of the launch sites imposes that launches must be decided a specified time in advance.

Two types of costs must be considered: first, the cost of the production of launchers and satellites and of the launches; second the cost which may result from a partial or complete unavailability of the constellation.

Failures may occur at any stage and at any time: launcher failure, spare satellite running failure, spare satellite orbital transfer failure, operational satellite running failure, failure of either a spare or an operational satellite.

The global objective of the management is finally to minimize over a given temporal horizon the sum of the production and of the unavailability costs.

At each step i (each month), three types of decisions are successively made:

1. *sub-step $k = 1$: the orbital plane of the launch at i is chosen;*

2. *sub-step $k = 2$: the number of satellites that are transferred from spare to operational on each orbital plane is chosen;*

3. *sub-step $k = 3$: the type of the launch at $i + DH$ is planned (launches must be planned in advance).*

PFU formulation In the following, we use the following notations:

- Cardinalities:
  - NOS = Number of Operational Satellites necessary on an orbital plane,
  - $MNSS$ = Maximum Number of Satellites on a Spare orbit,
  - NOP = Number of Orbital Plane,
  - NTL = Number of Types of Launchers,
  - $NLS[tl]$ = Number of Launchable Satellite for launchers of type tl,
  - $MTL[tl]$ = Minimum Time between two Launches of type tl,
  - DH = Decision Horizon (number of time steps necessary to plan a launch in advance).
- Probabilities of failure:
  - $PFL[tl]$ = Probability of Failure of a Launch of type tl,
  - $PFRSS$ = Probability of Failure when launching a satellite and Running it as a Spare Satellite,
  - $PFROS$ = Probability of Failure when transferring a satellite from a spare orbit to an operational one and Running it as an Operational Satellite,
  - $PFSS$ = Probability of Failure of a Spare Satellite in a month,
  - $PFOS$ = Probability of Failure of an Operational Satellite in a month.
- Costs:
  - $CL[tl]$ = Cost of a Launcher of type tl,
  - CS = Cost of a Satellite,
  - CU = Cost of a partial Unavailability of the constellation (a complete availability is assumed to be required at any moment).

Algebraic structure This problem uses probabilities, additive costs, and probabilistic expected utility. Therefore, we use:

- $S_p = (\mathbb{R}^+, +, \times)$ as a plausibility structure,
- $S_u = (\mathbb{R}^+, +)$ as a utility structure (an utility $u = \alpha$ stands for a cost of α),
- $S_{pu} = (E_p, E_u, +, \times)$ as an expected utility structure.

Variables We introduce environment variables which describe the state of the constellation and decision variables which correspond to the decisions made at each step.

- Environment variables:

  1. $nos[i, k, op]$ = number of operational satellites on orbital plane op, at step i, before the decision made at sub-step k;

     $dom(nos[i, k, op]) = \{0, \ldots, NOS\}$.

  2. $nss[i, k, op]$ = number of spare satellites on orbital plane op, at step i, before the decision made at sub-step k;

     $dom(nss[i, k, op]) = \{0, \ldots, MNSS\}$.

- Decision variables

  1. $lop[i]$ = orbital plane of the launch at step i;

     $dom(lop[i]) = \{0, \ldots, NOP\}$ ($lop[i] = 0$ applies when no launch has been planned).

  2. $nts[i, op]$ = number of spare satellites transferred at step i for orbital plane op;

     $dom(nts[i, op]) = \{0, \ldots, NOS\}$.

  3. $ptl[i]$ = type of launch planned at step i

     $dom(ptl[i]) = \{0, \ldots, NTL\}$ ($ptl[i] = 0$ means that no launch is planned at step i).

Feasibility functions The constraints on the decisions can be modeled using feasibility functions

1. $\forall i$: $(ptl[i] = 0) \rightarrow (lop[i] = 0)$ (this function associates no orbital plane with a null type of launch),

2. $\forall i, \forall op$: $nts[i, op] \leq nss[i, 2, op]$ (on each orbital plane, it is not possible to transfer more satellites than the number of satellites available on the spare orbit),

3. $\forall i, \forall op$: $nts[i, op] + nos[i, 2, op] \leq NOS$ (on each orbital plane, it is not possible to transfer more satellites than necessary),

4. $\forall i, j$: $(i < j < i + MTL[ptl[i]]) \rightarrow (ptl[i] \neq ptl[j])$ (constraints on the minimum time between two launches of the same type).

Plausibility (probability) functions The initial state is described by unary plausibility functions over each variable $nos[1, 1, op]$ and over each variable $nss[1, 1, op]$. Typically, $nos[1, 1, op] = nss[1, 1, op] = 0$ if we start from an empty constellation of satellites. The evolution of the constellation from step to step and from sub-step to sub-step is described by the following set of plausibility functions:

1. $\forall i, \forall op$: $nos[i, 2, op] = nos[i, 1, op]$ (we could merge the two variables)

2. $\forall i, \forall op$: $(lop[i] \neq op) \rightarrow (nss[i, 1, op] = nss[i, 2, op])$

3. $\forall i, \forall op$: let $op = op[i]$, $p_1 = PFL[ptl[i]]$, $p_2 = PFRSS$, $n = NLS[ptl[i]]$. Then,
$$P(nss[i,2,op] = nss[i,1,op] + k) = \begin{cases} p_1 + p_2^n \cdot (1-p_1) & \text{if } k = 0 \\ (1-p_1) \cdot C_n^k \cdot p_2^{n-k} \cdot (1-p_2)^k & \text{if } 0 < k \leq n \\ 0 & \text{otherwise} \end{cases}$$

4. $\forall i, \forall op$: $nss[i,3,op] = nss[i,2,op] - nts[i,op]$

5. $\forall i, \forall op$: let $p = PFROS$ and $n = nts[i,op]$. Then,
$$P(nos[i,3,op] = nos[i,2,op] + k) = \begin{cases} C_n^k \cdot p^{n-k} \cdot (1-p)^k & \text{if } 0 \leq k \leq n \\ 0 & \text{otherwise} \end{cases}$$

6. $\forall i, \forall op$: let $p = PFSS$, $n = nss[i,3,op]$, and $op' = (op \bmod NOP) + 1$. Then,
$$P(nss[i+1,1,op'] = k) = \begin{cases} C_n^k \cdot p^{n-k} \cdot (1-p)^k & \text{if } 0 \leq k \leq n \\ 0 & \text{otherwise} \end{cases}$$

7. $\forall i, \forall op$: let $p = PFOS$ and $n = nos[i,3,op]$. Then,
$$P(nos[i+1,1,op] = k) = \begin{cases} C_n^k \cdot p^{n-k} \cdot (1-p)^k & \text{if } 0 \leq k \leq n \\ 0 & \text{otherwise} \end{cases}$$

Utility (cost) functions In order to model the cost of the launches and of the satellites, and the cost which may result from a partial or complete unavailability of the constellation, we introduce several utility functions:

1. $\forall i$: $cl[i] = CL[ptl[i+DH]] + CS \cdot NLS[ptl[i+DH]]$ (cost of the planned launch)

2. $\forall i, \forall op$: $cu[i,op] = (NOS - nos[i,1,op]) \cdot CU$ (the cost of unavailability of the satellites is proportional to the number of missing satellites)

As we consider a finite horizon T, we need an evaluation of the final state of the constellation at T. Several formulations can be considered, one of them being simply to consider that utility of the state of the constellation at T is proportional to the number of operational satellites unavailable at T.

The PFU network graphical representation for a given step i is provided in Figures C.3 and C.4.

Query In order to deploy or maintain the constellation of satellites, the sequence of variable eliminations to consider is:

$$\ldots \sum_{\substack{\{nos[i,1,op], 1 \leq op \leq NOP\} \\ \cup\{nss[i,1,op], 1 \leq op \leq NOP\}}} \min_{lop[i]} \sum_{\substack{\{nos[i,2,op], 1 \leq op \leq NOP\} \\ \cup\{nss[i,2,op], 1 \leq op \leq NOP\}}} \min_{\{nts[i,op], 1 \leq op \leq NOP\}}$$
$$\sum_{\substack{\{nos[i,3,op], 1 \leq op \leq NOP\} \\ \cup\{nss[i,3,op], 1 \leq op \leq NOP\}}} \min_{ptl[i+DH]} \sum_{\substack{\{nos[i+1,1,op], 1 \leq op \leq NOP\} \\ \cup\{nss[i+1,1,op], 1 \leq op \leq NOP\}}} \ldots$$

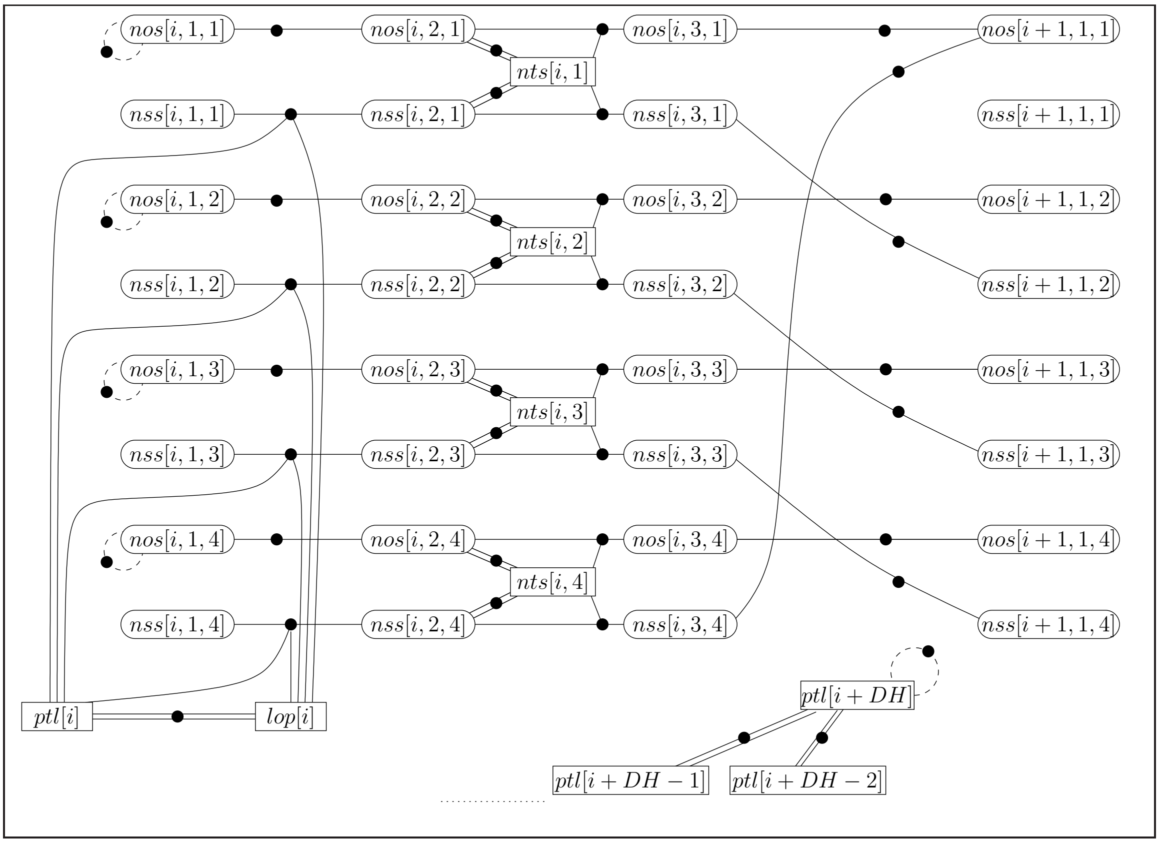


Figure C.3: Network of scoped functions.

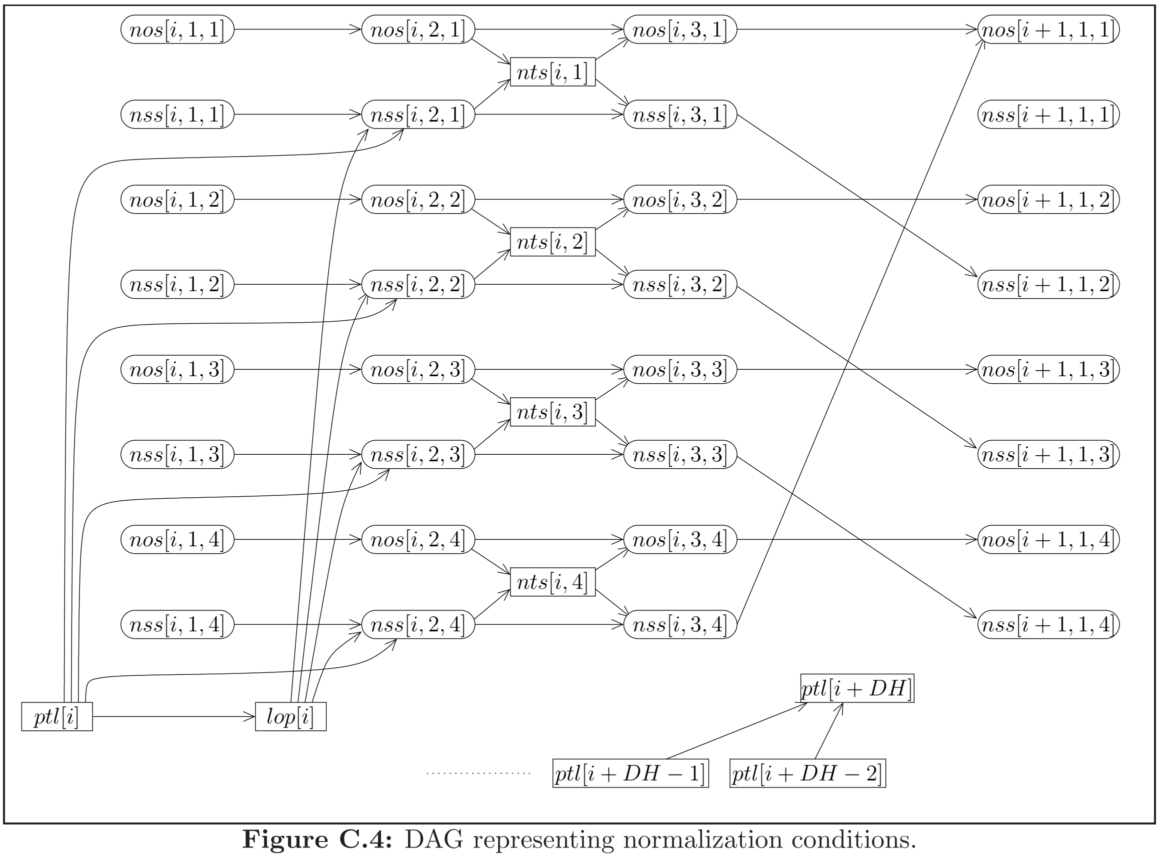


Figure C.4: DAG representing normalization conditions.

Appendix D

DTD of the XML format

```
<!ELEMENT query (name?,author?,date?,description?,pfunet,sov)>
<!ATTLIST query PFUnet (#PCDATA) #REQUIRED
                nbStages (#PCDATA) #REQUIRED
                nbRecords (#PCDATA) #REQUIRED>

<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT date (#PCDATA)>

<!ELEMENT pfunet EMPTY>
<!ATTLIST pfunet file (#PCDATA) #REQUIRED>

<!ELEMENT sov (op_vars_pair)>
<!ATTLIST sov nbstages (#PCDATA) #REQUIRED>

<!ELEMENT op_vars_pair EMPTY>
<!ATTLIST op_vars_pair op (MIN|MAX|PLUS) #REQUIRED
                       vars (#PCDATA) #REQUIRED
                       record (#PCDATA) #IMPLIED>
```

Figure D.1: DTD (Document Type Definition) for the XML representation of queries.

```
<!ELEMENT pfunet (name?,author?,date?,domains,plausfunctions?,feasfunctions?,utilfunctions?,
                  variables,plausibilities?,feasibilities?,utilities?,components)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT domains (domain+)>
<!ATTLIST domains nbDom (#PCDATA) #REQUIRED >
<!ELEMENT domain EMPTY>
<!ATTLIST domain id ID #REQUIRED
                 type (string|int|float|double|bool) #REQUIRED
                 description (extension|intension) #REQUIRED
                 values (#PCDATA) #REQUIRED>
<!ELEMENT plausfunctions (plausfunction+)>
<!ATTLIST plausfunctions nbPlausFunctions (#PCDATA) #REQUIRED>
<!ELEMENT plausfunction (instance*)>
<!ATTLIST plausfunction id ID #REQUIRED
                        domains (#PCDATA) #REQUIRED
                        default_degree (#PCDATA) #REQUIRED
                        nbInst (#PCDATA) #REQUIRED>
<!ELEMENT feasfunctions (feasfunction+)>
<!ATTLIST feasfunctions nbFeasFunctions (#PCDATA) #REQUIRED>
<!ELEMENT feasfunction (instance*)>
<!ATTLIST feasfunction id ID #REQUIRED
                        domains (#PCDATA) #REQUIRED
                        default_degree (#PCDATA) #REQUIRED
                        nbInst (#PCDATA) #REQUIRED>
<!ELEMENT utilfunctions (utilfunction+)>
<!ATTLIST utilfunctions nbUtilFunctions (#PCDATA) #REQUIRED>
<!ELEMENT utilfunction (instance*)>
<!ATTLIST utilfunction id ID #REQUIRED
                        domains (#PCDATA) #REQUIRED
                        default_degree (#PCDATA) #REQUIRED
                        nbInst (#PCDATA) #REQUIRED>
<!ELEMENT instance>
<!ATTLIST instance assignment (#PCDATA) #REQUIRED
                   degree (#PCDATA) #REQUIRED>
<!ELEMENT variables (variable+)>
<!ATTLIST variables nbVar (#PCDATA) #REQUIRED>
<!ELEMENT variable EMPTY>
<!ATTLIST variable id ID #REQUIRED
                   nature (decision|environment) #REQUIRED
                   domain IDREF #REQUIRED
                   description (#PCDATA) #IMPLIED>
<!ELEMENT plausibilities (plausibility+)>
<!ATTLIST plausibilities nbPlaus (#PCDATA) #REQUIRED>
<!ELEMENT plausibility EMPTY>
<!ATTLIST plausibility id ID #REQUIRED
                       scope IDREFS #REQUIRED
                       function IDREFS #REQUIRED>
<!ELEMENT feasibilities (feasibility+)>
<!ATTLIST feasibilities nbFeas (#PCDATA) #REQUIRED>
<!ELEMENT feasibility EMPTY>
<!ATTLIST feasibility id ID #REQUIRED
                       scope IDREFS #REQUIRED
                       function IDREFS #REQUIRED>
<!ELEMENT utilities (utility+)>
<!ATTLIST utilities nbUtil (#PCDATA) #REQUIRED>
<!ELEMENT utility EMPTY>
<!ATTLIST utility id ID #REQUIRED
                       scope IDREFS #REQUIRED
                       function IDREFS #REQUIRED>
<!ELEMENT components (component+)>
<!ATTLIST components nbComp (#PCDATA) #REQUIRED>
<!ELEMENT component EMPTY>
<!ATTLIST component id (#PCDATA) #REQUIRED
                    nature (decision|environment) #REQUIRED
                    vars IDREFS #REQUIRED
                    scoped_f IDREFS #REQUIRED
                    parents IDREFS #REQUIRED>
```

Figure D.2: DTD (Document Type Definition) for the XML representation of PFU networks.

THÈSE

présentée pour obtenir le titre de

DOCTEUR DE l'ÉCOLE NATIONALE SUPÉRIEURE DE L'AÉRONAUTIQUE ET DE L'ESPACE

Spécialité : Informatique

par

Cédric Pralet

Un cadre algébrique général pour représenter et résoudre des problèmes de décision séquentielle avec incertitudes, faisabilités et utilités

Thèse présentée devant le jury composé de:

Malik Ghallab	**LAAS-CNRS, Toulouse**	Examinateur
Patrice Perny	**LIP6, Paris**	Rapporteur
Francesca Rossi	**Université de Padoue (Italie)**	Rapporteur
Thomas Schiex	**INRA, Toulouse**	Directeur de thèse
Gérard Verfaillie	**ONERA, Toulouse**	Directeur de thèse
Nic Wilson	**4C, Cork (Irlande)**	Examinateur

Thèse préparée au LAAS-CNRS et à l'INRA Toulouse

Table des matières

Note de lecture

Ce document est un résumé d'une thèse écrite en anglais. Pour une vision complète et formelle du travail réalisé, nous conseillons au lecteur de se référer à la version anglaise.

Remerciements

Merci tout d'abord à Elyssa de m'avoir toujours supporté (dans les deux sens du terme) pendant ma thèse. Cette thèse est un peu la tienne. Merci aussi à ma famille pour son soutien. Je tiens également à remercier les personnes suivantes, tant sur le plan scientifique que sur le plan humain :

— Thomas Schiex et Gérard Verfaillie, mes deux directeurs de thèse, pour leur disponibilité, l'excellence de leur encadrement, leur ouverture d'esprit, et leur soutien. Merci notamment pour le caractère scientifiquement stimulant de nos réunions, qui, de mon point de vue, ont fait du travail de recherche un pur plaisir.

— Francesca Rossi, de l'université de Padoue, et Patrice Perny, de l'université Paris 6, qui m'ont fait l'honneur de s'intéresser à mon travail en acceptant d'être rapporteurs de cette thèse.

— Malik Ghallab, directeur du LAAS-CNRS, et Nic Wilson, chercheur au Cork Constraint Computation Center, pour avoir accepté de participer à mon jury de thèse. Merci sincèrement à Nic de m'avoir invité à présenter mes travaux à un workshop ECAI'06. Je lui suis réellement reconnaissant de cette belle opportunité.

— Aux membres de mon "comité de thèse" réunis à l'issue de mes premières et deuxièmes années de thèse : Rachid Alami du LAAS-CNRS, Jean-Loup Farges de l'ONERA Toulouse, Jérôme Lang de l'IRIT, et Régis Sabbadin de l'INRA Toulouse. Merci pour leur lecture attentive de mes rapports d'avancement et pour les discussions que j'ai pu avoir avec eux par la suite.

— Plus généralement, merci aux personnes du groupe RIA du LAAS-CNRS et aux personnes de l'INRA pour la bonne ambiance de travail dont j'ai pu bénéficier.

Introduction

Au cours des dernières décennies, de nombreux formalismes ont été développés pour représenter et résoudre des problèmes de décision pouvant correspondre à des problèmes de planification d'actions (comme en ordonnancement de tâches ou en allocation de ressources) ou à des problèmes de recherche d'explications (comme en diagnostic ou en suivi de situation). Ces problèmes peuvent être plus ou moins complexes suivant les paramètres qu'ils intègrent :

1. L'évolution de l'environnement peut être déterministe ou non et on peut avoir des mesures d'incertitudes, appelées *plausibilités*, concernant l'état du monde.
2. Certaines actions peuvent être *faisables* uniquement si certaines préconditions sont satisfaites.
3. Les différents états de l'environnement et les diverses décisions possibles n'ont généralement pas la même valeur du point de vue des décideurs : des préférences peuvent être exprimées pour modéliser des coûts, des gains, des risques, des degrés de satisfaction, des exigences dures… Ces préférences sont appelées ici des *utilités*.
4. Le processus décisionnel peut être *séquentiel*, c'est-à-dire qu'il peut y avoir plusieurs étapes de décision. Certaines informations peuvent être observées entre deux étapes de décision, à l'instar des échecs où deux joueurs jouent à tour de rôle, et où chaque coup est joué après observation du dernier coup adverse.
5. Le problème peut faire intervenir plusieurs agents collaboratifs ou antagonistes. Certaines décisions peuvent être non *contrôlables* par un agent donné.

Cette thèse considère des formes générales de problèmes de décision faisant intervenir tous ces aspects. Plus précisément, étant données les plausibilités sur l'état de l'environnement, les contraintes de faisabilité sur les décisions, les utilités définissant des préférences et la succession des différentes étapes de décision, le but est de fournir à un agent décideur des règles de décision optimales pour les décisions qu'il contrôle, ceci en fonction de l'environnement et des autres agents.

De nombreux formalismes classiques existent pour résoudre des problèmes inclus dans cette classe de problèmes. Parmi ces formalismes, on peut citer :

— le cadre des problèmes de satisfiabilité d'une formule logique propositionnelle (SAT) et ses extensions permettant de prendre en compte un contexte non déterministe (*Quantified Boolean Formulae, QBF*) ou stochastique (Stochastic Satisfiability [62]) ;

— le cadre très proche des problèmes de satisfaction de contraintes (*Constraint Satisfaction Problems*, CSP [63]) et ses extensions permettant de prendre en compte des préférences (*Valued* ou *Semiring-based CSP* [11]) ou un contexte non déterministe (*Quantified CSP* [13], *Mixed CSP* [38]) ou stochastique (*Stochastic CSP* [107]) ;

- le cadre de la représentation de l'incertain, incluant les *réseaux bayésiens* [73], les champs de Markov (*Markov random fields* [18]), les graphes chaînés [42], ainsi que leurs extensions permettant de prendre en compte des contraintes (*Hybrid* et *Mixed networks* [29, 30]), ou alors des décisions, des utilités et/ou des faisabilités (*Influence diagrams* [48, 52, 101, 71, 51], *valuation networks* [98, 100, 34]) ;
- le cadre de la planification (*STRIPS* planning [40, 44], PDDL [65]) et ses extensions permettant de prendre en compte l'incertitude sur l'état courant et sur les effets des actions (*Conformant Planning* [46], *Probabilistic Planning* [58]) ;
- le cadre enfin des processus décisionnels markoviens (*Markov Decision Processes* [86]), avec ses extensions permettant de prendre en compte un contexte d'observabilité partielle (*Partially Observable MDP* [68]), des incertitudes non probabilistes [91, 74], ou encore la structure des états (*Factored MDP* [16]).

Au-delà de leurs nombreuses différences, ces cadres possèdent d'importantes similitudes :

- ils utilisent des variables à domaine souvent fini pour représenter soit l'état de l'environnement (*variables d'environnement*), soit les décisions d'un ou plusieurs agents (*variables de décision*) ;
- ils mettent en jeu des *fonctions locales* qui peuvent représenter des *faisabilités* pesant sur les variables de décision (par exemple, des pré-conditions d'actions), des *plausibilités* portant sur les variables d'environnement (par exemple, des distributions de probabilité conditionnelles) ou encore des *utilités*, fonctions de diverses variables (par exemple, des coûts ou des degrés de satisfaction) ;
- ils font appel à divers opérateurs, soit pour *agréger* les fonctions locales (par exemple, le $\wedge$ logique pour agréger les faisabilités, le $\times$ pour agréger les probabilités, le $+$ pour agréger les utilités additives), soit pour *synthétiser* une information globale (par exemple, le $\vee$ logique pour décider d'une faisabilité, le $+$ pour calculer une distribution de probabilité marginale, le max ou le min pour sélectionner une décision optimale).

Ils peuvent donc tous être vus comme des *modèles graphiques* dans la mesure où ils reposent tous, implicitement ou explicitement, sur un *hyper-graphe* de fonctions locales entre variables à domaine fini. Les différences entre eux tiennent essentiellement à ce que représentent variables et fonctions, ainsi qu'aux opérateurs d'agrégation et de synthèse utilisés.

Cette thèse montre qu'il est possible de les rassembler dans un même cadre générique, graphique et algébrique : *graphique* pour respecter leur nature graphique et *algébrique* pour abstraire les opérateurs d'agrégation et de synthèse utilisés et ne plus considérer que des opérateurs abstraits dotés de certaines propriétés algébriques. La variété des cadres visés, des problèmes de satisfiabilité aux processus décisionnels markoviens, peut cependant faire penser qu'une telle unification est hors d'atteinte ou que le résultat en serait un monstre incompréhensible et ingérable. Cette thèse montre qu'il n'en est rien et que tous ces cadres sont de fait suffisamment proches pour être, grâce à une approche algébrique, réunis dans un seul, dénommé PFU, et dont les composants et les propriétés peuvent être décrits de manière relativement simple et compacte.

Cette thèse montre également qu'il est possible de ramener de nombreux problèmes de décision à des calculs de séquences d'*éliminations* de variables sur une *combinaison* de fonctions locales.

Motivations Construire un cadre générique pour représenter et résoudre des problèmes de décision variés est utile pour diverses raisons :

- *Unification et meilleure compréhension des formalismes existants* : construire un cadre générique présente tout d'abord un intérêt théorique et pédagogique. Cette démarche peut permettre de mieux comprendre des relations souvent ignorées entre des cadres spécifiques développés par des communautés qui souvent se méconnaissent ;
- *Expressivité accrue* : un cadre générique peut permettre, par la variété de la structure proposée, de considérer des nouveaux cadres spécifiques non encore explorés ;
- *Intérêt algorithmique* : il devrait être possible de définir des algorithmes de résolution génériques, dont on sait qu'ils se révèlent souvent aussi performants, sinon plus, que les algorithmes spécifiques développés dans tel ou tel cadre au prix d'efforts non négligeables. Cet objectif est dans son esprit relié à une démarche globale d'identification d'approches algorithmiques communes développées pour résoudre différents problèmes d'intelligence artificielle. Il peut également permettre à un cadre donné de bénéficier des avancées algorithmiques réalisées dans d'autres cadres.

Organisation de la thèse Cette thèse est découpée en deux parties :

1. La première partie se concentre sur des aspects représentation de la connaissance. Elle introduit un nouveau cadre général de représentation pour la décision séquentielle avec incertitudes, faisabilités et utilités.

   Après avoir défini certaines notations et certaines notions (chapitre 1), nous commençons par montrer informellement au chapitre 2, via un catalogue de formalismes existants, pourquoi et comment un cadre générique peut être construit.

   Ce cadre générique, appelé le cadre Plausibilité-Faisabilité-Utilité (PFU), est ensuite formellement introduit en trois temps :

   - Des structures algébriques permettant d'exprimer des formes générales d'incertitudes, de faisabilités et d'utilités sont tout d'abord définies au chapitre 3. Ces structures spécifient comment combiner et synthétiser des informations.
   - Sur ces structures algébriques, nous introduisons au chapitre 4 une forme de modèle graphique faisant intervenir des variables et un réseau de fonctions locales entre ces variables.
   - Enfin, la notion de requête est introduite au chapitre 5. Les requêtes permettent de formuler des problèmes de décision variés sur un réseau de fonctions locales.

2. La second volet de cette thèse a pour objet la définition d'algorithmes génériques permettant de répondre à des requêtes définies dans le cadre PFU.

   - Les premiers algorithmes génériques présentés au chapitre 6 sont des algorithmes de recherche arborescente et d'élimination de variables qui essaient d'exploiter au mieux le fait que les informations sont exprimées par des fonctions *locales*. Leur complexité est fonction d'un paramètre appelé largeur induite contrainte.
   - Des techniques plus sophistiquées analysant la structure d'une requête de manière plus fine sont ensuite introduites au chapitre 7. Cette analyse structurelle nous conduit à une architecture de calcul générale, appelée l'architecture des DAG de clusters multi-opérateurs (DAG = Directed Acyclic Graph). Cette architecture exprime de manière

explicite une décomposition des calculs à réaliser pour répondre à une requête.

- Partant de cette architecture, le chapitre 8 définit des algorithmes de recherche arborescente structurée qui peuvent être plus ou moins sophistiqués suivant s'ils utilisent des techniques de mémorisation ou des bornes pour élaguer l'espace de recherche.
- Enfin, le chapitre 9 présente très brièvement un outil de résolution générique permettant de répondre à des requêtes sur un réseau PFU. Cet outil prouve notamment que le cadre développé n'est pas juste une abstraction.

Première partie

Un nouveau cadre générique de représentation de problèmes de décision : le cadre PFU

Chapitre 1

Notations et définitions

Ce court chapitre introduit quelques objets mathématiques utilisés intensivement par la suite. Nous manipulons notamment les notions de variables, domaines, fonctions locales, modèles graphiques, opérateurs de combinaison, opérateurs d'élimination, règles de décision et certains éléments de vocabulaire relatifs aux graphes. Certaines de ces notions sont illustrées par un exemple jouet qui précise également ce que nous entendons par "plausibilité", "faisabilité", "utilité", "observabilité partielle" ou "controlabilité".

1.1 Quelques définitions

Définition 1.1. *Le* domaine *de valeurs d'une variable x est noté $dom(x)$ et pour tout $a \in dom(x)$, (x, a) représente l'affectation de x avec la valeur a.*

Par extension, étant donné un ensemble de variables S, nous notons $dom(S)$ le produit cartésien des domaines des variables de S, c'est-à-dire $dom(S) = \prod_{x \in S} dom(x)$. Un élément $A \in dom(S)$ est appelé une affectation de S.[1]

Si A_1, A_2 sont deux affectations d'ensembles disjoints S_1, S_2, alors la concaténation de A_1 et A_2, notée $A_1.A_2$, est l'affectation de $S_1 \cup S_2$ dans laquelle les variables de S_1 prennent la même valeur que dans A_1 et les variables de S_2 prennent la même valeur que dans A_2.

Si A est une affectation d'un ensemble de variables S, alors la projection de A sur un ensemble de variables S', notée $A^{\downarrow S'}$, est l'affectation de $S \cap S'$ donnant à chaque variable la même valeur que dans A.

Définition 1.2. *(Fonction locale et portée d'une fonction locale) Une* fonction locale *est un couple (S, φ) tel que S est un ensemble de variables et φ est une fonction associant à chaque élément de $dom(S)$ un élément dans un ensemble E donné.*

Par la suite, nous considérons souvent que l'ensemble de variables S est implicite. Ainsi, une fonction locale (S, φ) peut être notée simplement φ. L'ensemble de variables S est appelée la portée *de φ et est noté $sc(\varphi)$ (sc comme "scope"). Si A est une affectation d'un sur-ensemble de $sc(\varphi)$, alors $\varphi(A)$ vaut $\varphi(A^{\downarrow sc(\varphi)})$.*

[1] Techniquement, une affectation de $S = \{x_1, \ldots, x_k\}$ devrait être un ensemble de paires variable-valeur $\{(x_1, a_1), \ldots, (x_k, a_k)\}$. Nous supposons ici que les variables sont implicites lorsqu'un tuple de valeurs $(a_1, \ldots, a_k) \in dom(S)$ est utilisé.

Par exemple, une fonction locale φ associant à chaque affectation de $sc(\varphi)$ un élément dans le treillis booléen $\mathbb{B} = \{t, f\}$ est analogue à une contrainte décrivant le sous-ensemble des affectations de $sc(\varphi)$ qui satisfont la contrainte en question.

A partir de la notion de fonction locale, la notion de modèle graphique peut être définie :

Définition 1.3. *(Modèle graphique) Un* modèle graphique *est un couple (V, Φ) tel que $V = \{x_1, \ldots, x_n\}$ est un ensemble fini de variables et $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ est un ensemble fini de fonctions locales dont la portée est incluse dans V.*

Le terme modèle *graphique* est utilisé simplement car un ensemble de fonctions locales peut être représenté par un hypergraphe dont les hyper-arêtes sont les portées des fonctions locales. Comme nous le verrons, cet hypergraphe représente une certaine forme d'indépendance conditionnelle et induit des paramètres influençant la complexité algorithmique. La définition adoptée ici généralise la définition classique utilisée en statistique selon laquelle un modèle graphique est un graphe (orienté ou non) dont les nœuds représentent des variables aléatoires et dont la structure modélise des relations d'indépendances conditionnelles probabilistes.

Les fonctions locales d'un modèle graphique expriment de manière compacte une fonction globale portant sur toutes les variables du modèle graphique. Cette fonction globale est obtenue en agrégeant toutes les fonctions locales. Par exemple, un réseau bayésien [73] représente une distribution de probabilité jointe globale $P_{x,y,z}$ sous la forme d'un produit de fonctions locales qui peuvent être les fonctions locales de l'ensemble $\{P_x, P_{y|x}, P_{z|x}\}$.

Afin de raisonner sur un modèle graphique (V, Φ), il est nécessaire de pouvoir synthétiser l'information qu'il exprime sur un sous-ensemble des variables de V. Par exemple, pour calculer une distribution de probabilité marginale $\mathcal{P}_{y,z}$ à partir du réseau bayésien précédent, nous devons calculer la quantité : $\sum_x P_{x,y,z} = \sum_x (P_x \times P_{y|x} \times P_{z|x})$. Partant d'une information portant sur $\{x, y, z\}$, l'opérateur $\sum$ permet d'obtenir une information sur $\{y, z\}$ en "éliminant" la variable x. Les opérateurs utilisés pour combiner des fonctions locales sont appelés des opérateurs de *combinaison* et les opérateurs utilisés pour synthétiser des informations sont appelés des opérateurs d'*élimination*.

Définition 1.4. *(Combinaison) Soit φ_1, φ_2 deux fonctions locales à valeurs dans E_1 et E_2 respectivement. Soit $\otimes : E_1 \times E_2 \to E$ un opérateur binaire. La combinaison de φ_1 et φ_2, notée $\varphi_1 \otimes \varphi_2$, est la fonction locale à valeurs dans E dont la portée est $sc(\varphi_1) \cup sc(\varphi_2)$, et qui satisfait, pour toute affectation A de cette portée, $(\varphi_1 \otimes \varphi_2)(A) = \varphi_1(A) \otimes \varphi_2(A)$. L'opérateur $\otimes$ est appelé un* opérateur de combinaison *de φ_1 et φ_2.*

Définition 1.5. *(Elimination) Soit φ une fonction locale à valeurs dans E. Soit op un opérateur associatif et commutatif sur E. L'élimination d'une variable x sur φ avec un opérateur op est la fonction locale de portée $sc(\varphi) - \{x\}$ qui satisfait, pour toute affectation A de cette portée, $(op_x\, \varphi)(A) = op_{a \in dom(x)}\, \varphi(A.(x, a))$. L'opérateur op est dans ce cas appelé* opérateur d'élimination *de la variable x*

De manière analogue, l'élimination d'un ensemble de variables $S = \{x_1, \ldots, x_k\}$ sur φ est une fonction locale de portée $sc(\varphi) - S$ definie par $(op_S\, \varphi)(A) = op_{A' \in dom(S)}\, \varphi(A.A')$.

Ainsi, dans l'expression $\sum_x (P_x \times P_{y|x} \times P_{z|x})$, des fonctions locales sont agrégées via l'opérateur de combinaison $\otimes = \times$ et l'information est synthétisée via une élimination de x avec l'opérateur d'élimination $+$. Dans toute la suite de la partie I, $\otimes$ représente des opérateurs de combinaison et $\oplus$ représente des opérateurs d'élimination. Notons que la notion d'opérateur de combinaison ou d'élimination n'est pas une propriété intrinsèque d'un opérateur donné. Elle dépend de l'usage qui est fait de cet opérateur : par exemple, l'opérateur $+$ a le statut d'opérateur de combinaison s'il est utilisé pour agréger des gains et des coûts alors qu'il a le statut d'opérateur d'élimination s'il sert à calculer une distribution de probabilité marginale.

Dans certains cas, l'élimination d'un ensemble de variables S avec un opérateur op sur une fonction locale φ doit être réalisée uniquement sur un sous-ensemble de $dom(S)$ contenant les affectations qui satisfont une certaine propriété représentée par une fonction booléenne F. Nous devons alors calculer, pour chaque affectation $A \in dom(sc(\varphi)-S)$, la quantité $op_{A' \in dom(S), F(A')=t}\, \varphi(A.A')$. Pour des raisons de simplicité et d'homogénéité et pour n'utiliser que des éliminations sur $dom(S)$, on peut de manière équivalente tronquer la fonction φ pour que les éléments de $dom(S)$ violant la propriété définie par F soient associés à un élément spécial (noté $\Diamond$) qui est lui-même un élément neutre de op.

Définition 1.6. *(Opérateur de troncature) L'*élément infaisable $\Diamond$ *est un nouvel élément spécial et tout opérateur d'élimination op est étendu de manière à satisfaire $op(\Diamond, e) = op(e, \Diamond) = e$ pour tout élément e du domaine de définition de op.*

Soit $\{t, f\}$ le treillis booléen. Pour chaque booléen b et chaque élément e, nous définissons l'opérateur $\star$ tel que $b \star e$ vaut e si $b = t$ et $\Diamond$ sinon. $\star$ est appelé l'opérateur de troncature.

Etant donnée une fonction locale booléenne F, l'élément infaisable $\Diamond$ et l'opérateur de troncature $\star$ permettent d'écrire des quantités telles que $op_{A' \in dom(S), F(A')=t}\, \varphi$ sous la forme $op_S\, (F \star \varphi)$. Dans cette dernière forme, une élimination est réalisée sur tout le domaine des variables et la fonction locale F a le même statut que la fonction locale φ (c'est pourquoi nous disons que $\star$ et $\Diamond$ permettent d'utiliser des notations plus simples et plus homogènes).

Lorsqu'un problème de décision est résolu, l'objectif est souvent d'obtenir des *règles de décision* indiquant des décisions à prendre en fonction des informations disponibles :

Définition 1.7. *(Règle de décision, politique) Une* règle de décision *pour une variable x sachant un ensemble de variables S' est une fonction $\delta : dom(S') \to dom(x)$ associant à chaque affectation de S' une valeur du domaine de x. Par extension, une règle de décision pour un ensemble de variables S sachant un ensemble de variables S' est une fonction $\delta : dom(S') \to dom(S)$. Un ensemble de règles de décision est appelé une* politique.

Des exemples de règles de décision sont les règles décision qui sont optimales du point de vue d'un certain critère décision. Par exemple, si l'on travaille sur un ensemble totalement ordonné et si l'on doit effectuer le calcul $\sum_{S'} \max_S \varphi$ avec φ une fonction locale, une règle de décision optimale $\delta : dom(S') \to dom(S)$, qui vérifie $\varphi(A.\delta(A)) \succeq \varphi(A.A')$ pour tout $(A, A') \in dom(sc(\varphi)-S) \times dom(S)$, peut être obtenue en utilisant argmax.

Quelques définitions sur les graphes

Définition 1.8. *$\mathcal{G} = (V, H)$ est un hypergraphe si et seulement si V est un ensemble de variables et H est un ensemble d'hyper-arêtes sur V, i.e. un sous-ensemble de 2^V.*

Définition 1.9. *Un graphe $G = (V, E)$ est un arbre si et seulement si G est un graphe connexe, non orienté, et sans cycle. G est un arbre enraciné si et seulement si G est un graphe connexe, orienté et sans cycle. La racine de l'arbre est alors l'unique sommet du graphe sans parent.*

Définition 1.10. *(Graphe acyclique orienté (Directed Acyclic Graph, DAG)) Un graphe orienté G est un DAG si et seulement si G ne contient pas de cycle orienté. Lorsque des variables sont associées aux sommets du graphe, on note $pa_G(x)$ l'ensemble des parents d'une variable x dans G.*

Enfin, le cardinal d'un ensemble fini Γ est noté $|\Gamma|$.

1.2 Un exemple illustratif

Un exemple jouet a été bâti pour donner une vision concrète des notions de "plausibilités", "faisabilités", "utilités", "observabilité", "variable de décision", "variable d'environnement" ou encore de "contrôlabilité". Cet exemple illustre également concrêtement comment variables et fonctions locales peuvent exprimer une information globale de manière compacte. Il introduit enfin le lien entre problèmes de décision et séquences d'éliminations de variables sur des combinaisons de fonctions locales

Exemple *Jean a trois portes en face de lui : A, B, et C de gauche à droite. Derrière l'une de ces portes, se trouve un trésor et derrière une autre, un gangster. Jean doit décider quelle porte ouvrir. Il sait qu'il gagnera 10 000€ s'il ouvre la porte où se trouve le trésor, mais qu'il devra payer 4 000€ s'il ouvre la porte où se trouve le gangster.*

Modélisation Pour modéliser ce problème, nous introduisons trois variables : (1) deux variables représentant l'environnement de Jean, l'une notée tr pour représenter la porte du trésor et l'autre notée ga pour représenter la porte du gangster ; (2) une variable notée do (comme "door") représentant la décision de Jean. Ces trois variables ont toutes le même domaine de valeur $\{A, B, C\}$. Les variables de décision correspondent aux variables dont la valeur est choisie directement par un agent, alors que les variables d'environnement correspondent aux variables dont la valeur n'est pas choisie directement par un agent.

Pour modéliser les coûts et les gains possibles, nous introduisons deux fonctions locales d'utilité : une première U_1 qui exprime que si Jean ouvre la porte du trésor, il gagne 10 000€ (contrainte souple $do = tr$ de poids 10 000, qui renvoie son poids si elle est satisfaite et 0 sinon) et une seconde U_2 qui exprime que si Jean ouvre la porte du gangster, il paye 4 000€ (contrainte souple $do = ga$ de poids -4 000, qui renvoie de même son poids si elle est satisfaite et 0 sinon). Une contrainte souple peut aussi être appelée une fonction de coût.

Requête Quelle est ou quelles sont la(les) décision(s) de Jean qui maximise(nt) son utilité si le gangster est derrière la porte A et le trésor est derrière la porte C ? La réponse est évidemment la décision (do, C).

Ajout d'incertitudes

Dans les problèmes réels, l'environnement peut ne pas être complètement connu : il peut exister des incertitudes sur l'environnement, appelées ici des plausibilités, et certaines observations de l'environnement incertain peuvent éventuellement être réalisées.

Example *Le trésor et le gangster ne sont pas derrière la même porte et toutes les situations possibles sont équiprobables. Jean travaille en équipe avec Pierre et chacun peut choisir une porte où écouter et ainsi tenter de repérer le gangster. On suppose que la probabilité d'entendre le gangster est de* 0.8 *si on écoute à la porte derrière laquelle il se trouve, de* 0.4 *si on écoute à une porte adjacente et de* 0 *sinon.*

Nouvelle modélisation Pour modéliser ces nouveaux éléments, nous introduisons quatre variables supplémentaires :

- deux variables de décisions li_J et li_P (li comme “listen”) de domaine $\{A, B, C\}$ qui représentent respectivement les portes auxquelles Jean et Pierre écoutent ;
- deux variables d'environnement he_J et he_P (he comme “hear”) de domaine $\{yes, no\}$ qui représentent respectivement le fait que Jean ou Pierre entend le gangster ou non.

Nous introduisons également des *fonctions locales de plausibilité* :

- $P_1 : ga \neq tr$ et $P_2 = 1/6$, qui représentent la distribution de probabilité sur les positions du gangster et du trésor
- $P_3 = P_{he_J \,|\, li_J, ga}$, qui spécifie la probabilité que Jean entende du bruit sachant la porte à laquelle il écoute et la porte derrière laquelle se trouve le gangster ;
- P_4, qui correspond de manière analogue à la distribution conditionnelle $P_{he_P \,|\, li_P, ga}$.

Ces fonctions locales de plausibilité satisfont implicitement certaines *conditions de normalisation.* D'une part, étant donné que le gangster et le trésor se trouvent quelque part, on peut écrire $\sum_{ga,tr}(P_1 \times P_2) = 1$. D'autre part, étant donné que Jean et Pierre entendent quelque chose ou non, nous avons $\sum_{he_J} P_3 = 1$ et $\sum_{he_P} P_4 = 1$. Ces normalisations traduisent le fait que la disjonction de toutes les situations possibles est certaine.

Requêtes associées Quelle est ou quelle sont la(les) décision(s) qui maximisent l'utilité espérée, si on suppose que Jean et Pierre choisissent chacun une porte où écouter et ensuite Jean choisit une porte à ouvrir en fonction de ce qui a été entendu ?

Pour répondre à une telle requête, une approche classique consiste à construire un *arbre de décision.* Dans cet arbre, les variables sont considérées dans un ordre cohérent avec l'ordre des décisions et des observations, par exemple dans l'ordre $li_J \to li_P \to he_J \to he_P \to do \to ga \to tr$. Tout nœud n de cet arbre correspond à une variable x et toute arête de n vers un nœud fils correspond à une affectation (x, a) de x. Si x est une variable d'environnement, cette arête est de plus pondérée par la probabilité $P((x, a) \,|\, A)$, où A est l'affectation associée au chemin de la racine à n. Toute feuille de cet arbre correspond à une affectation A de l'ensemble des variables et son utilité est l'utilité globale $(U_1 + U_2)(A)$ associée à A. L'utilité d'un nœud de décision est l'utilité optimale de ses nœuds fils, avec possibilité de mémoriser la(les) décision(s) correspondante(s). L'utilité d'un nœud d'environnement est la somme des utilités de ses nœuds fils, pondérées par le poids des arêtes associées. L'utilité espérée associée à la requête est celle du nœud racine. Il est cependant prouvé [79] que cette approche à base d'arbres de décisions est équivalente au calcul de la formule suivante qui ne fait intervenir que les probabilités présentes dans la définition du problème, et pas des probabilités de type $P((x,a) \mid A)$ dont le calcul est potentiellement complexe :

$$\max_{li_J, li_P} \sum_{he_J, he_P} \max_{do} \sum_{ga,tr} \left(\left(\prod_{i \in [1,4]} P_i \right) \times \left(\sum_{i \in [1,2]} U_i \right) \right)$$

Cet exemple montre que raisonner à partir d'arbres de décision est équivalent à effectuer une séquence d'éliminations de variables sur une combinaison de fonctions locales. Des règles de décision optimales peuvent être mémorisées en utilisant un argmax pendant les calculs.

D'autres scénarios correspondent à d'autres séquences d'éliminations :

— Si Jean pense que Pierre est un traître et s'il le laisse choisir une porte à ouvrir en premier (attitude pessimiste vis-à-vis de Pierre), la séquence d'éliminations devient

$$\min_{li_P} \max_{li_J} \sum_{he_J, he_P} \max_{do} \sum_{ga,tr}$$

Cette séquence élimine li_P avec l'opérateur min.

— Si Pierre ne dit même pas à Jean ce qu'il a entendu, ou autrement si Jean n'observe pas la valeur de la variable he_P, alors la séquence d'éliminations devient

$$\min_{li_P} \max_{li_J} \sum_{he_J} \max_{do} \sum_{he_P} \sum_{ga,tr}$$

c'est-à-dire que l'élimination $\sum_{he_P}$ est placée à droite de l'élimination $\max_{do}$.

Ajout de faisabilités

Il se peut que certaines préconditions soient requises pour que certaines décisions soient faisables. Par exemple, si deux joueurs acceptent de respecter les règles des échecs, alors un coup est dit faisable s'il respecte ces règles. Notons ici que ce qui est infaisable est différent de ce qui est inacceptable, puisque par exemple aucun des joueurs ne peut jouer un coup impossible alors que chacun d'entre eux peut éventuellement jouer un coup inacceptable pour son adversaire en le mettant échec et mat.

Exemple *Jean et Pierre ne peuvent pas écouter à la même porte, et la porte A est fermée.*

Modélisation Ces contraintes sur les décisions peuvent être modélisées en utilisant deux fonctions locales de faisabilité : une première F_1 qui exprime que Jean et Pierre ne peuvent pas écouter à la même porte (contrainte dure $li_J \neq li_P$) et une seconde F_2 qui exprime que la porte C ne peut pas être ouverte (contrainte dure $do \neq C$). De la même façon qu'avec les plausibilités, ces contraintes sont supposées exprimer des distributions de faisabilité normalisées ($\vee_{li_J, li_P} F_1 = t$ et $\vee_{do} F_2 = t$) pour traduire le fait qu'une décision est toujours possible, même s'il s'agit de ne rien faire.

Avec ces données supplémentaires, la procédure classique à base d'arbre de décision est équivalente au calcul de la quantité suivante :

$$\min_{li_P}\max_{li_J}\sum_{he_J}\max_{do}\sum_{he_P}\sum_{ga,tr}((\underset{i\in[1,2]}{\wedge} F_i)\star(\prod_{i\in[1,4]} P_i)\times(\sum_{i\in[1,2]} U_i))$$

dans laquelle l'opérateur de troncature $\star$ permet de ne pas prendre en compte les décisions infaisables. La forme obtenu est à nouveau une séquence d'éliminations de variables sur une combinaison de fonctions locales.

Au terme de cet exemple, nous voyons que les données du problème peuvent être exprimées par l'intermédiaire de variables et de fonctions locales qui forment un modèle graphique *composite* constitué d'un DAG représentant des conditions de normalisation sur les plausibilités et les faisabilités (figure 1.1(a)), [2] et d'un réseau de fonctions locales (figure 1.1(b)). Ce réseau fait intervenir plusieurs types de variables (variables de décision et variables d'environnement) et plusieurs types de fonctions locales (fonctions locales de plausibilité, de faisabilité et d'utilité).

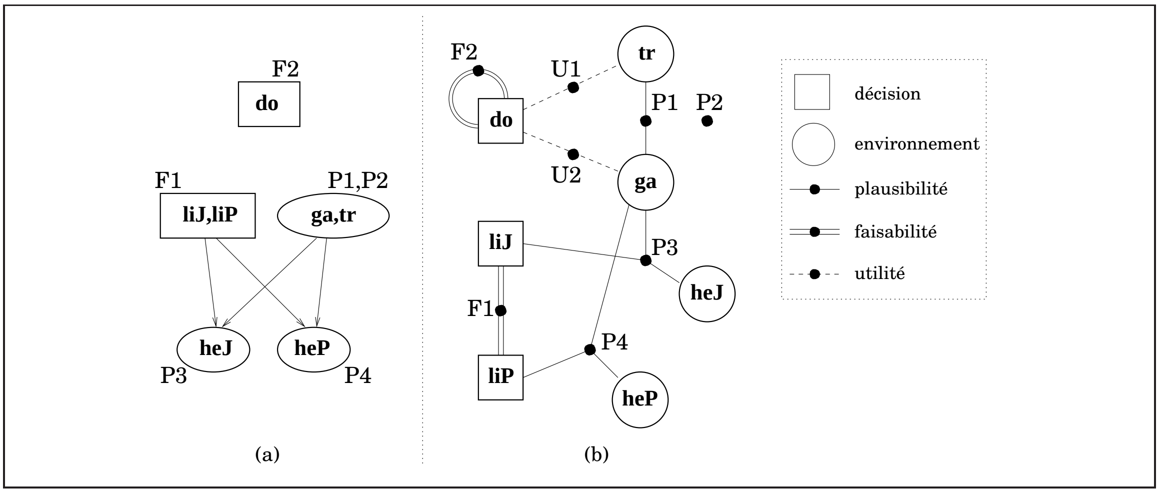


Figure 1.1: Modèle graphique composite (a) DAG représentant des conditions de normalisation ; (b) Réseau de fonctions locales.

En résumé, l'exemple introduit ici illustre la notion de problème de décision séquentielle faisant intervenir des plausibilités, des faisabilités et des utiités. Cette notion est fortement utilisée dans les chapitres à venir.

[2] Si P est l'ensemble des fonctions locales de plausibilité associées à un nœud du DAG étiqueté par un ensemble de variables S, cela signifie que $\sum_S (\prod_{P_i\in P} P_i) = 1$. De même, si F est l'ensemble des fonctions locales de faisabilités associées à un nœud du DAG étiqueté par un ensemble de variables S, cela signifie que $\vee_S(\wedge_{F_i\in F} F_i) = t$.

Chapitre 2

Cadres existants

L'étape prélable à la définition d'un cadre générique pour la décision est une étape de compréhension et d'analyse des formalismes existants. Ces formalismes, issus d'efforts non négligeables réalisés au sein de plusieurs communautés, sont nombreux. Ils peuvent présenter des capacités de représentation plus ou moins sophistiquées. Certains peuvent modéliser des préférences alors que d'autres sont adaptés uniquement pour modéliser des exigences dures. Certains peuvent modéliser des incertitudes alors que d'autres ne le peuvent pas. Certains peuvent modéliser des problèmes de décision séquentielle, d'autres ne le peuvent pas.

Ce chapitre présente un catalogue **non-exhaustif** de certains de ces formalismes. Ce catalogue présente deux caractéristiques principales :

- Il est incrémental, dans le sens où il montre comment des cadres de base que sont le cadre des problèmes de satisfiabilité d'une formule logique propositionnelle (SAT), les problèmes de satisfaction de contraintes [63], les réseaux bayésiens [73], la planification classique [40, 44] ou les processus décisionnels markoviens [86, 68] ont été étendus pour intégrer la notion d'incertitude pour certains ou la notion de préférence et de décision pour d'autres.
- Il analyse les similitudes et les différences entre les formalismes existants en termes de représentation de la connaissance. Cette analyse tend à montrer que beaucoup de formalismes raisonnent à partir de variables et de fonctions locales entre variables et que les problèmes de décision qu'ils permettent de formuler peuvent être réduits à des calculs de séquences d'éliminations de variables sur une combinaison de fonctions locales.

Se référer à la version anglaise de la thèse pour une description plus précise et plus complète des formalismes existants.

2.1 Des CSP aux MDP algèbriques

Le cadre des problèmes de satisfaction de contraintes (*Constraint Satisfaction Problems*, CSP [63]) ou celui de la satisfiabilité d'une formule logique propositionnelle (SAT) peuvent être vus comme des modèles graphiques (V, Φ) dans lesquels l'ensemble des fonctions locales Φ contient des contraintes ou des clauses associant la valeur vrai ou faux à chaque affectation de leur portée. Une requête classique sur les CSP consiste à trouver une affectation des variables qui satisfait toutes les contraintes. En termes purement algébriques, cette requête peut être vue comme une requête d'optimisation binaire s'écrivant sous la forme (en supposant $f \prec t$) :

$$\max_V \big(\bigwedge_{\varphi \in \Phi} \varphi \big) \tag{2.1}$$

Si la quantité précédente vaut t, alors le CSP est dit cohérent et une affectation optimale de V obtenue en utilisant argmax définit une solution, c'est-à-dire une affectation des variables qui satisfait toutes les contraintes. Si la quantité précédente vaut f, alors le CSP est dit incohérent et aucune affectation de V ne satisfait toutes les contraintes. Dans l'équation 2.1, les fonctions locales sont combinées par un $\wedge$ logique et les variables sont toutes éliminées par un max.

En remplaçant les contraintes dures par des contraintes souples pour lesquelles un coût est payé en cas de violation de la contrainte et en remplaçant $\wedge$ par un opérateur de combinaison $\otimes$ qui peut valoir min, max, $+$, $\times$... nous obtenons la forme algébrique associée à la recherche d'une solution pour un CSP valué [94] ou un *semiring-based CSP* [9, 10] totalement ordonné.

Dans une autre direction, un *réseau bayésien* [73] est aussi un modèle graphique (V, Φ), dans lequel les fonctions locales sont des distributions de probabilité conditionnelles : $\Phi = \{P_{x \,|\, pa(x)}\ ,\ x \in V\}$ où $pa(x)$ correspond à l'ensemble des parents de la variable x dans un graphe acyclique orienté (DAG) associé au réseau bayésien. Un tel réseau représente de façon concise une distribution de probabilité jointe P_V sur toutes les variables, qui s'écrit sous la forme $P_V = \prod_{x \in V} P_{x \,|\, pa(x)}$, de la même manière qu'un CSP représente une contrainte globale sur toutes les variables comme une conjonction de contraintes locales. Les requêtes susceptibles d'être formulées sur un réseau bayésien sont multiples. On peut par exemple chercher la distribution de probabilité marginale sur une variable $y \in V$. Algébriquement parlant, le calcul associé à cette requête est

$$P_y = \sum_{V - \{y\}} \big(\prod_{x \in V} P(x \,|\, pa(x)) \big) \tag{2.2}$$

Dans ce cas, les fonctions locales sont combinées par un produit et toutes les variables, hormis y, sont éliminées par une somme.

Utiliser des distributions de probabilité conditionnelles locales n'est pas l'unique moyen d'exprimer une distribution de probabilité globale sous une forme factorisée. En effet, dans des domaines tels que le traitement d'images ou les neurosciences, d'autres modèles sont utilisés, parmi lesquels on trouve notamment le formalisme des champs de Markov (*Markov Random Fields* [18]). Dans un champ de Markov, une distribution de probabilité jointe globale P_V se factorise sous une forme appelée distribution de Gibbs, c'est-à-dire sous la forme $P_V = 1/Z \cdot \prod_{\varphi \in \Phi} e^{-\beta_\varphi \cdot \varphi}$ avec Z une constante de normalisation, Φ un ensemble de fonctions locales appelées des potentiels et β_φ une constante associée à chaque $\varphi \in \Phi$. La portée des potentiels $\varphi \in \Phi$ est définie par les cliques d'un graphe non orienté associé au champ de Markov et les potentiels ne correspondent pas à des distributions de probabilité conditionnelles locales.

Etant donné un champ de Markov modélisant P_V, une affectation la plus probable de V peut être déterminée en calculant :

$$\max_V \big(\frac{1}{Z} \times \prod_{\varphi \in \Phi} exp(-\beta_\varphi \cdot \varphi) \big) \tag{2.3}$$

Les fonctions locales sont combinées par un produit et des éliminations avec max sont réalisées. Intrinsèquement, un champ de Markov exploite des propriétés d'indépendance dans un graphe non orienté alors qu'un réseau bayésien exploite des propriétés d'indépendance dans un graphe orienté. Les deux modèles peuvent fournir des résultats très différents en terme de taille de la portée des fonctions locales utilisées. *Généralement*, les réseaux bayésiens sont plus utilisés pour modéliser des relations de causalité alors que les champs de Markov sont plutôt utilisés pour modéliser des corrélations spatiales.

Les réseaux bayésiens et les champs de Markov sont unifiés par le formalisme des graphes chaînés (*Chain graphs* [42]). Un graphe chaîné utilise un graphe qui contient à la fois des liens orientés et des liens non orientés, tels que les cycles dans ce graphe impliquent uniquement des liens non orientés. L'ensemble $\mathcal{C}$ des composantes connexes obtenues lorsque les liens orientés sont supprimés est appelé l'ensemble des composantes du graphe chaîné. Un graphe chaîné peut alors être vu comme un DAG dont les sommets correspondent aux composantes de $\mathcal{C}$.

Un graphe chaîné représente une distribution de probabilité jointe P_V sous la forme factorisée $P_V = \prod_{c \in \mathcal{C}} P_{c\,|\,pa(c)}$, chaque distribution de probabilité conditionnelle $P_{c\,|\,pa(c)}$ étant elle-même exprimée comme dans un champ de Markov sous une forme du type $P_{c\,|\,pa(c)} = \frac{1}{Z_{pa(c)}} \prod_{\varphi \in \Phi_c} e^{-\beta_\varphi \cdot \varphi}$.

Les équations 2.1, 2.2 et 2.3 utilisent un seul opérateur de combinaison et un seul opérateur d'élimination. Dans d'autres cas, plusieurs opérateurs de combinaison et/ou plusieurs opérateurs d'élimination peuvent être utilisés.

Prenons l'exemple des CSP stochastiques (*stochastic CSP* [107]). Les CSP stochastiques étendent les CSP classiques en ajoutant, en plus des variables de décision contrôlables, des variables dites contingentes qui sont non contrôlables et qui ne peuvent pas être influencées par les décisions prises. Cette dernière donnée est appelée l'hypothèse de contingence. Lorsque les variables contingentes sont mutuellement indépendantes, un ensemble P de distributions de probabilité unaires P_s sur chaque variable contingente s est disponible. Un ensemble de contraintes C est également défini, comme dans un CSP classique.

Considérons le scénario suivant : la valeur de deux variables de décision d_1 et d_2 doit être choisie, puis on observe la valeur d'une variable contingente s_1 et enfin on prend des décisions d_3 et d_4 (potentiellement en fonction de s_1) sans avoir observé une variable contingente s_2. Deux distributions de probabilité P_{s_1} et P_{s_2} sont définies sur s_1 et s_2 et certaines contraintes sont imposées sur les variables. Ces éléments définissent un CSP stochastique dit à deux étages (car il y a deux étages de décision). Chercher des règles de décision maximisant la probabilité que les contraintes soient satisfaites est équivalent à chercher des règles de décision optimales pour la quantité suivante :

$$\max_{d_1} \max_{d_2} \sum_{s_1} \max_{d_3} \max_{d_4} \sum_{s_2} (P(s_1) \times P(s_2)) \times (c_1 \times \ldots \times c_m) \tag{2.4}$$

Ainsi, toutes les fonctions locales sont combinées par un produit et des éliminations utilisant max pour les variables de décision et + pour les variables contingentes sont réalisées.

Un autre formalisme dans lequel plusieurs types d'opérateurs peuvent être utilisés est le formalisme des diagrammes d'influence [48]. Ce formalisme étend les réseaux bayésiens en leur ajoutant les notions de décision et d'utilité. Un diagramme d'influence met en jeu trois types de variables organisées dans une structure de DAG :

- des variables aléatoires ; l'ensemble des variables aléatoires est noté S et pour chaque variable $s \in S$, on spécifie une distribution de probabilité $P_{s \mid pa(s)}$ sur s sachant ses parents dans le DAG ;
- des variables de décision ; l'ensemble des variables de décision est noté $\{d_1, \ldots, d_q\}$, les indices représentant l'ordre dans lequel les décisions sont prises ; pour chaque variable de décision d, $pa(d)$ correspond à l'ensemble des variables dont la valeur est observée lorsque la décision d est prise ;
- des variables d'utilité ; l'ensemble des variables d'utilité est noté Γ ; on associe à chaque variable d'utilité u une fonction locale $U_{pa(u)}$ dont la portée est égale aux parents de u dans le DAG ; ces fonctions locales représentent une utilité globale $U_G = \sum_{u \in \Gamma} U_{pa(u)}$; les variables d'utilité doivent être des feuilles du DAG.

Le problème associé à un diagramme d'influence consiste à trouver des règles de décision maximisant l'utilité espérée. Algébriquement parlant, si on note I_0 l'ensemble des variables aléatoires observées avant de prendre la première décision, I_k l'ensemble des variables aléatoires dont la valeur est observée entre les décisions d_k et d_{k+1} et I_q l'ensemble des variables aléatoires dont la valeur n'est pas observée avant la dernière décision, alors des règles de décision optimales sont définies en utilisant argmax dans la quantité suivante :

$$\sum_{I_0} \max_{d_1} \sum_{I_1} \max_{d_2} \ldots \sum_{I_{q-1}} \max_{d_q} \sum_{I_q} ((\prod_{s \in S} P_{s \mid pa(s)}) \times (\sum_{u \in \Gamma} U_{pa(u)})) \tag{2.5}$$

Dans le calcul ci-dessus, nous combinons les probabilités par un produit, les utilités par une somme et les probabilités avec les utilités par un produit. Les variables de décision sont éliminées par un max et les variables aléatoires sont éliminées par un $+$. L'ordre dans lequel les variables sont éliminées est directement fonction de l'ordre dans lequel les décisions sont prises et les observations sont réalisées.

Il se peut également qu'intervienne un ensemble de fonctions locales de faisabilité décrivant quelles décisions sont possibles et que, comme dans les champs de Markov, les fonctions locales représentant les facteurs multiplicatifs d'une distribution de probabilité globale ne soient pas des distributions de probabilité conditionnelles. Dans ce cas, le formalisme des réseaux de valuation [98, 100, 34] peut être utilisé. Dans ce formalisme ou dans ses extensions, les calculs à effectuer pour trouver des règles de décisions optimales peuvent s'écrire sous la forme suivante :

$$\max_{d_1,d_2} \sum_{s_1} \max_{d_3,d_4} \sum_{s_2} \left(\left(\bigwedge_{F_i \in F} F_i \right) \star \left(\prod_{P_i \in P} P_i \right) \times \left(\sum_{U_i \in U} U_i \right) \right) \tag{2.6}$$

Les fonctions locales de faisabilité sont combinées par un $\wedge$ logique et combinées avec les autres fonctions locales en utilisant l'opérateur de troncature $\star$ (cf. définition 1.6 page 19). A nouveau, une séquence d'éliminations de variables est réalisée. De telles fonctions locales de faisabilités peuvent également être utilisées pour modéliser des préconditions dans un problème de planification classique [40, 44].

Considérons maintenant le cadre des processus décisionnels markoviens (*Markov Decision Processes*, MDP [86, 68, 91]) à horizon fini. La présentation ci-dessous ne correspond pas à la présentation classique des MDP mais est une formulation équivalente.

Un MDP décrit l'évolution de l'environnement par pas de temps. A chaque pas de temps t sont associées une variable non déterministe s_t décrivant l'état de l'environnement à t et une variable de décision d_t décrivant une décision prise à t.

Dans un MDP probabiliste, les incertitudes sur l'évolution de l'environnement sont décrites par des distributions de probabilité conditionnelles locales $P_{s_{t+1}\,|\,s_t,d_t}$ d'être dans l'état s_{t+1} à l'étape $t+1$ sachant l'état s_t à l'instant t et la décision d_t prise à t. Des préférences sur l'environnement et sur les décisions sont exprimées par l'intermédiaire de fonctions locales de récompense additives R_{s_t,d_t} associées à chaque instant t. A chaque instant, l'état de l'environnement s_t est connu avant de prendre la décision d_t. Pour des raisons de clarté, l'état s_1 est supposé connu. S'il y a T étapes de décision, alors trouver des règles de décisions optimales peut se faire en calculant

$$\max_{d_1}\sum_{s_2}\max_{d_2}\ldots\sum_{s_T}\max_{d_T}\left(\prod_{t\in[1,T-1]} P_{s_{t+1}\,|\,s_t,d_t}\right)\times\left(\sum_{t\in[1,T]} R_{s_t,d_t}\right) \tag{2.7}$$

pour obtenir l'utilité espérée optimale et en utilisant argmax pour définir une politique optimale : les probabilités sont combinées par un produit, les récomponses sont combinées par une somme, les probabilités sont combinées avec les récompenses en utilisant un produit, les variables de décisions sont éliminées par un max et les variables d'environnement sont éliminées par une somme.

Dans un MDP possibiliste, le même genre d'équation peut être obtenu. La différence est que les distributions de probabilité sont remplacées par des distributions de possibilité $\pi(s_{t+1}\,|\,s_t,d_t)$, les récompenses additives sont remplacées par des preférences $\mu(s_t,d_t)$ combinées par un min et les opérateurs de combinaison et d'élimination utilisés conduisent à la forme suivante, dans le cas d'un MDP possibiliste dit *pessimiste* :

$$\max_{d_1}\min_{s_2}\max_{d_2}\ldots\min_{s_T}\max_{d_T}\max\left(1-\min_{t\in[1,T-1]}\pi_{s_{t+1}\,|\,s_t,d_t}\ ,\ \min_{t\in[1,T]}\mu_{s_t,d_t}\right) \tag{2.8}$$

Les incertitudes sont combinées par un min, les utilités sont combinées par un min, une plausibilité p et une utilité u sont combinées par $\max(1-p,u)$, les variables de décision sont éliminées par un max et les variables d'environnement sont éliminées par un min.

Ces similarités entre MDP utilisant plusieurs modèles de plausibilité et d'utilité ont été exploitées pour définir les MDP algébriques [74], qui permettent d'exprimer des problèmes qui sont équivalents à calculer des quantités du type :

$$\max_{d_1}\underset{s_2}{\oplus_u}\max_{d_2}\ldots\underset{s_T}{\oplus_u}\max_{d_T}\left(\left(\underset{t\in[1,T-1]}{\otimes_p} P_{s_{t+1}\,|\,s_t,d_t}\right)\otimes_{pu}\left(\underset{t\in[1,T]}{\otimes_u} U_{s_t,d_t}\right)\right) \tag{2.9}$$

où les plausibilités sont combinées par un opérateur abstrait $\otimes_p$, les utilités sont combinées par un opérateur $\otimes_u$, plausibilités et utilités sont combinées par un opérateur $\otimes_{pu}$ et où une séquence d'éliminations de variables est effectuée (max sur les décisions et $\oplus_u$ sur les variables d'environnement).

2.2 Les trois éléments de base d'un cadre générique pour la décision séquentielle avec incertitudes, faisabilités et utilités

Les exemples donnés précédemment montrent que nombre de requêtes classiques formulées dans des cadres existants peuvent être ramenées à un calcul d'une séquence d'éliminations de variables sur une combinaison de fonctions locales. Ces cadres existants peuvent de surcroît tous être vus comme des modèles graphiques qui diffèrent principalement de par les opérateurs d'élimination et de combinaison utilisés et de par ce que les variables et les fonctions locales représentent.

C'est ce genre d'observations qui a conduit à la définition des MDP algébriques [74] ou des algèbres de valuation [97, 98, 56], ce dernier cadre étant un cadre algébrique générique au sein duquel le problème principal est de calculer une séquence d'éliminations de variables sur une combinaison de fonctions locales. Cependant, les algèbres de valuation ne font intervenir qu'un seul opérateur de combinaison, alors que plusieurs opérateurs peuvent être nécessaires pour agréger les différents types de fonctions locales d'un modèle graphique composite. En outre, les algèbres de valuation ne font intervenir qu'un seul type d'élimination, alors que plusieurs opérateurs d'élimination peuvent être nécessaires pour éliminer les différents types de variables. Dans les réseaux de valuation [100], les plausibilités représentent nécessairement des probabilités, et des minimisations ne peuvent pas être réalisées. Tous ces arguments justifient la nécessité d'introduire un cadre plus expressif.

Afin de couvrir les requêtes formulées dans divers formalismes, la forme générale à considérer est la suivante :

$$Sov\left(\left(\bigwedge_{F_i\in F} F_i\right) \star \left(\underset{P_i\in P}{\otimes_p} P_i\right) \otimes_{pu} \left(\underset{U_i\in U}{\otimes_u} U_i\right)\right) \tag{2.10}$$

où (1) $\wedge$, $\otimes_p$, $\otimes_u$ sont utilisés respectivement pour combiner les faisabilités locales, les plausibilités locales et les utilités locales, $\otimes_{pu}$ est utilisé pour combiner plausibilités et utilités, et l'opérateur de troncature $\star$ permet d'ignorer les décisions infaisables sans avoir à gérer des éliminations sur des domaines restreints ; (2) F, P, U sont des ensembles (éventuellement vides) de fonctions locales de faisabilité, de plausibilté et d'utilité respectivement ; (3) Sov est une séquence de couple opérateur-variable(s) qui indique comment les variables doivent être éliminées. Sov fait intervenir les opérateurs d'élimination min ou max sur les variables de décision et un opérateur $\oplus_u$ sur les variables d'environnement.

L'équation 2.10 a été obtenue de manière informelle, par analogie avec les cadres existants. Afin de donner une sémantique claire au calcul purement algébrique qu'elle exprime, il est nécessaire de définir trois éléments principaux :

1. Nous devons définir les divers opérateurs de combinaison $\otimes_p$, $\otimes_u$, $\otimes_{pu}$ ainsi que l'opérateur $\oplus_u$ utilisé pour éliminer les variables d'environnement. Un opérateur d'élimination noté $\oplus_p$

permettant de synthétiser certaines informations sur les plausibilités sera également introduit. Ces divers opérateurs définissent la *structure algébrique* du cadre PFU. Naturellement, certaines propriétés algébriques seront requises. Sémantiquement parlant, ces opérateurs définissent le modèle de plausibilité/utilité.

2. Nous devons ensuite exprimer des informations sous la forme d'un modèle graphique faisant intervenir un ensemble de variables et des ensembles de fonctions locales exprimant des plausibilités, des faisabilités et des utilités (ensembles P, F, U). Ces éléments définiront des *réseaux PFU*. Nous devons également justifier la possibilité d'exprimer de la connaissance sous une telle forme factorisée, notamment via la notion d'indépendance conditionnelle.
3. Enfin, pour formuler des problèmes de décision, nous devons définir la notion de *requêtes sur des réseaux PFU* en introduisant une séquence Sov de couple opérateur-variable(s) appliquée à la combinaison de toutes les fonctions locales, comme dans l'équation 2.10. Ces requêtes doivent permettrent de modéliser des situations variées en termes d'observabilité et de contrôlabilité. Il est également nécessaire de montrer pourquoi le calcul de quantités du type de celle donnée à l'équation 2.10 est intéressant d'un point de vue sémantique, en comparant cette équation avec une approche classique à base d'arbres de décision.

2.3 Résumé

Ce chapitre a montré de manière informelle que de nombreuses requêtes formulées dans des formalismes variés raisonnant sur des plausibilités et/ou des faisabilités et/ou des utilités pouvaient être réduites à des calculs de séquences d'éliminations de variables sur des combinaisons de fonctions locales utilisant des opérateurs variés, sous une forme intuitivement couverte par celle donnée à l'équation 2.10.

Les trois éléments fondamentaux (une structure algébrique, un réseau PFU, et une séquence d'éliminations de variables) nécessaires pour définir formellement cette équation et lui donner du sens sont introduits dans les chapitres 3, 4 et 5 respectivement.

Chapitre 3

Une structure algébrique générique pour la décision séquentielle dans l'incertain

Le premier élément du cadre PFU est une structure algébrique spécifiant comment les informations fournies par les plausibilités, les faisabilités et les utilités sont combinées et synthétisées. La structure algébrique utilisée s'appuie sur les travaux de Friedman, Halpern et Chu [41, 47, 19], qui proposent une généralisation axiomatique et algébrique des notions classiques de probabilité, d'utilité et d'utilité espérée, sous les dénominations de *plausibilité*, d'*utilité* et d'*utilité espérée généralisée* (des approches similaires sont développées dans [108, 22]). La structure proposée s'écarte cependant sur certains points de leurs propositions.

3.1 Quelques définitions algébriques

Définition 3.1. $(E, \circledast)$ *est un* monoïde commutatif *ssi E est un ensemble et $\circledast$ est un opérateur binaire ($E \times E \to E$) qui est associatif ($x \circledast (y \circledast z) = (x \circledast y) \circledast z$), commutatif ($x \circledast y = y \circledast x$), et qui possède un élément neutre $1_E \in E$ ($x \circledast 1_E = 1_E \circledast x = x$).*

Définition 3.2. $(E, \oplus, \otimes)$ *est un* semi-anneau commutatif *ssi*

- *$(E, \oplus)$ est un monoïde commutatif dont l'élément neutre est noté 0_E ;*
- *$(E, \otimes)$ est un monoïde commutatif dont l'élément neutre est noté 1_E,*
- *0_E est absorbant pour $\otimes$ ($x \otimes 0_E = 0_E$),*
- *$\otimes$ est distributif par rapport à $\oplus$ ($x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$).*

Définition 3.3. *Soit E un ensemble partiellement ordonné par $\preceq$. Un opérateur $\circledast$ sur E est dit monotone ssi $(x \preceq y) \to (x \circledast z \preceq y \circledast z)$ pour tous $x, y, z \in E$.*

3.2 Structure de plausibilité

L'exemple de la chasse au trésor introduit au chapitre 1 utilise des *probabilités* pour modéliser l'incertitude. Sous hypothèse d'indépendance, les probabilités sont agrégées via un opérateur $\otimes_p = \times$ et synthétisées via un opérateur $\oplus_p = +$. Mais d'autres théories peuvent être utilisées pour modéliser l'incertitude, comme par exemple la théorie des *possibilités* [36] ou celle des *fonctions de Spohn* [102]. Avec la première, une option possible consiste à utiliser $\otimes_p = \min$ pour combiner les plausibilités et $\oplus_p = \max$ pour synthétiser des plausibilités "marginales", tandis qu'avec la seconde, $\otimes_p = +$ et $\oplus_p = \min$.

Afin de manipuler des formes générales de plausibilités, nous définissons la notion de *structure de plausibilité*. Une structure de plausibilité est définie comme un triplet $(E_p, \oplus_p, \otimes_p)$ où E_p est un ensemble de *degrés de plausibilité* équipé d'un *ordre partiel* $\preceq_p$, $\oplus_p$ est un *opérateur d'élimination* et $\otimes_p$ un *opérateur de combinaison* sur les plausibilités. Nous imposons à cette structure de satisfaire les axiomes de base sur les plausibilités proposés par Friedman et Halpern dans [41, 47]. Nous étendons cependant la structure qu'ils proposent pour que $\oplus_p$ et $\otimes_p$ soient clos sur E_p. Ceci nous conduit à la définition suivante :

Définition 3.4. *Une structure de plausibilité est un triplet* $(E_p, \oplus_p, \otimes_p)$ *tel que :*

— $(E_p, \oplus_p, \otimes_p)$ *est un semi-anneau commutatif ; l'élément neutre de* $\oplus_p$ *est noté* 0_p *et l'élément neutre de* $\otimes_p$ *est noté* 1_p *;*

— E_p *est équipé d'un ordre partiel* $\preceq_p$ *dont* 0_p *est l'élément minimum ;*

— $\oplus_p$ *et* $\otimes_p$ *sont monotones pour* $\preceq_p$*.*

0_p est associé aux événements impossibles et 1_p aux événements certains. A noter qu'alors que cette définition impose que 0_p soit l'élément minimum de E_p, elle n'impose pas que 1_p en soit l'élément maximum. La structure de plausibilité associée aux probabilités est par exemple $(\mathbb{R}^+, +, \times)$, avec $\preceq_p = \leq$, $0_p = 0$ et $1_p = 1$.

3.3 Structure de faisabilité

Du fait qu'une décision est soit faisable, soit infaisable, la *structure de faisabilité* que nous utilisons n'est pas paramétrable. C'est en fait un cas particulier de structure de plausibilité : $(\{t, f\}, \vee, \wedge)$ avec $f \prec_p t$, $0_p = f$ et $1_p = t$.

Ainsi, les fonctions locales exprimant des faisabilités sont combinées par un $\wedge$ logique car une décision est faisable si et seulement si toutes les fonctions de faisabilité la juge faisable. Etant donnée une fonction locale de faisabilité F_i, une affectation A est faisable d'après F_i si et seulement s'il existe une affectation A' des variables de la portée de F_i non affectées par A telle que $F_i(A.A') = t$. Ceci explique pourquoi les éliminations sur les faisabilités se font via un $\vee$ logique.

3.4 Structure d'utilité

L'exemple de la chasse au trésor utilise des *utilités additives* (des coûts et des gains) pour modéliser les préférences. Elles sont agrégées via un opérateur $\otimes_u = +$. Mais si les préférences sont modélisées par des *priorités*, elles sont agrégées via $\otimes_u = \min$. D'autres opérateurs d'agrégation des utilités peuvent également être utilisés.

Une *structure d'utilité* est définie comme une paire $(E_u, \otimes_u)$ où E_u est un ensemble de *degrés d'utilité* équipé d'un *ordre partiel* $\preceq_u$ et $\otimes_u$ un *opérateur de combinaison* sur les utilités. Des axiomes classiques sur les utilités nous conduisent à la définition suivante :

Définition 3.5. *Une structure d'utilité est une paire $(E_u, \otimes_u)$ telle que :*

- *$(E_u, \otimes_u)$ est un monoïde commutatif dont l'élément neutre est noté 1_u ;*
- *E_u est équipé d'un ordre partiel $\preceq_u$ et $\otimes_u$ est monotone.*

1_u est associé aux situations indifférentes du point de vue de l'agrégation des utilités. La structure d'utilité associée aux utilités additives classiques est par exemple $(\mathbb{R}, +)$, avec $\preceq_u = \leq$ et $1_u = 0$.

La distinction entre plausibilités, faisabilités et utilités est importante et peut être justifiée par des arguments purement algébriques. Etant donné que les opérateurs $\otimes_p$ et $\otimes_u$ peuvent être différents (par exemple $\otimes_p = \times$ et $\otimes_u = +$ dans la théorie de l'utilité espérée probabiliste avec utilités additives), il est tout d'abord nécessaire de distinguer plausibilités et utilités.

Il est aussi nécessaire de distinguer les faisabilités des utilités et des plausibilités. En effet, prenons l'exemple d'un jeu de cartes où chaque joueur doit jouer deux cartes prises parmi des valets, des dames et des rois. Jouer un valet (respectivement une dame, un roi) rapporte 5, 10 et 20 points respectivement. La décision la meilleure est alors évidemment de jouer deux fois un roi et la décision la pire consiste à jouer deux fois un valet. Mais supposons que jouer deux fois la même carte soit interdit.

Dans ce cas, si l'on veut calculer des décisions optimales, il est nécessaire de restreindre les opérations d'optimisation à la partie faisable du domaine des variables de décision. Comme aucun élément de l'ensemble des degrés d'utilité E_u ne peut être ignoré à la fois par l'opérateur min et par l'opérateur max, les faisabilités doivent être l'objet d'un traitement spécifique. Intuitivement, l'infaisabilité n'est pas une notion relative (alors que l'utilité l'est) : l'infaisabilité correspond à des règles communément admises par tous les agents décideurs et se situe ainsi hors de toute échelle d'utilité.

Afin d'ignorer les valeurs infaisables des variables de décision, nous utilisons l'opérateur de troncature $\star$ introduit à la définition 1.6 page 19. Afin d'éliminer une variable de décision d'une fonction locale φ tout en ignorant les décisions infaisables spécifiées par une fonction de faisabilité F_i, il suffit alors d'éliminer x sur $(F_i \star \varphi)$ au lieu de φ, de manière à associer la valeur infaisable $\lozenge$ aux décisions infaisables.

3.5 Structure d'utilité espérée

Dans la théorie de l'*utilité espérée probabiliste*, la formule classique $\sum_i p_i \times u_i$ traduit une agrégation des probabilités et des utilités via un opérateur $\otimes_{pu} = \times$ et une synthèse du résultat via un opérateur $\oplus_u = +$. Cette formule se généralise sous la forme $\underset{i}{\oplus_u} (p_i \otimes_{pu} u_i)$, avec par exemple $\otimes_{pu} = \min$ et $\oplus_u = \max$ dans la théorie de l'*utilité espérée possibiliste optimiste* [37] ou $\otimes_{pu} = +$ et $\oplus_u = \min$ dans la théorie de l'*utilité* espérée avec *fonctions de Spohn* [45] (avec des utilités uniquement positives).

Afin de généraliser ces structures existantes, nous définissons la notion de *structure d'utilité espérée*. Une structure d'utilité espérée est définie comme un quadruplet $(E_p, E_u, \oplus_u, \otimes_{pu})$ où $\oplus_u$ est un opérateur d'élimination sur les utilités et $\otimes_{pu}$ un opérateur de combinaison entre plausibilités et utilités. Nous imposons à cette structure de satisfaire les axiomes de base sur l'utilité espérée généralisée proposés dans [19]. Pour prendre en compte l'aspect éventuellement *séquentiel* de la décision, nous imposons cependant des axiomes supplémentaires justifiés par la théorie des loteries [106]. Ceci nous conduit à la définition suivante :

Définition 3.6. *Une structure d'utilité espérée est un quadruplet* $(E_p, E_u, \oplus_u, \otimes_{pu})$ *tel que :*

— $(E_u, \oplus_u, \otimes_{pu})$ *est un semi-module sur* $(E_p, \oplus_p, \otimes_p)$*, ce qui implique que* $(E_p, \oplus_p, \otimes_p)$ *soit un semi-anneau commutatif, que* $(E_u, \oplus_u)$ *soit un monoïde commutatif avec un élément neutre noté* 0_u *et que l'opérateur binaire* $\otimes_{pu}$ $(E_p \times E_u \to E_u)$ *satisfasse les propriétés suivantes :*

- *distributivité par rapport à* $\oplus_p$ *et* $\oplus_u$ *;*
- $\forall p_1, p_2 \in E_p, \forall u \in E_u, p_1 \otimes_{pu} (p_2 \otimes_{pu} u) = (p_1 \otimes_p p_2) \otimes_{pu} u$ *;*
- $\forall u \in E_u, (0_p \otimes_{pu} u = 0_u) \wedge (1_p \otimes_{pu} u = u)$ *;*

— $\oplus_u$ *est monotone et* $\otimes_{pu}$ *monotone à droite pour* $\preceq_u$.

0_u est associé aux situations *indifférentes* du point de vue de l'élimination des utilités. La structure d'utilité espérée associée à l'utilité espérée probabiliste est par exemple $(\mathbb{R}^+, \mathbb{R}, +, \times)$, avec $0_u = 0$. Toutes les structures présentées dans le tableau 3.1 sont des structures d'utilité espérée.

	E_p	$\preceq_p$	$\oplus_p$	$\otimes_p$	$0_p, 1_p$	E_u	$\preceq_u$	$\otimes_u$	$\oplus_u$	$\otimes_{pu}$	$\bot_u, 0_u, 1_u$
1	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 1$	$\mathbb{R} \cup \{-\infty\}$	$\leq$	$+$	$+$	$\times$	$-\infty, 0, 0$
2	$\mathbb{R}^+$	$\leq$	$+$	$\times$	$0, 1$	$\mathbb{R}^+$	$\leq$	$\times$	$+$	$\times$	$0, 0, 1$
3	$[0, 1]$	$\leq$	max	min	$0, 1$	$[0, 1]$	$\leq$	min	max	min	$0, 0, 1$
4	$[0, 1]$	$\leq$	max	min	$0, 1$	$[0, 1]$	$\leq$	min	min	$\max(1{-}p, u)$	$0, 1, 1$
5	$\mathbb{N} \cup \{\infty\}$	$\geq$	min	$+$	$\infty, 0$	$\mathbb{N} \cup \{\infty\}$	$\geq$	$+$	min	$+$	$\infty, \infty, 0$
6	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\wedge$	$\vee$	$\wedge$	f, f, t
7	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\wedge$	$\wedge$	$\to$	f, t, t
8	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\vee$	$\wedge$	f, f, f
9	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	f, t	$\{t, f\}$	$\preceq_{bool}$	$\vee$	$\wedge$	$\to$	f, t, f

TABLE 3.1 – Ensembles et opérateurs utilisés dans plusieurs cadres classiques : (1) utilité espérée probabiliste avec utilités additives (permet de calculer l'espérance d'un gain ou d'un coût), (2) utilité espérée probabiliste avec utilités multiplicatives (permet de calculer la probabilité que des contraintes soient satisfaites), (3) utilité espérée possibiliste optimiste, (4) utilité espérée possibiliste pessimiste, (5) utilité qualitative avec kappa-rankings et utilités uniquement positives, (6) utilité espérée booléenne optimiste avec utilités conjonctives (permet de savoir s'il existe un monde possible dans lequel tous les buts d'un ensemble de buts B sont satisfaits), (7) utilité espérée booléenne pessimiste avec utilités conjonctives (permet de savoir si dans tous les mondes possibles tous les buts d'un ensemble de buts B sont satisfaits), (8) utilité espérée booléenne optimiste avec utilités disjonctives (permet de savoir s'il existe un monde possible dans lequel au moins un but d'un ensemble de buts B est satisfait), (9) utilité espérée booléenne pessimiste avec utilités disjonctives (permet de savoir si dans tous les mondes possibles au moins un but d'un ensemble de buts B est satisfait).

Le problème du repas d'affaire Pour illustrer les définitions précédentes et à venir, considérons l'exemple suivant. *Pierre invite Jean and Marie (un couple divorcé) à un repas d'affaire pour les convaincre d'investir dans son entreprise. Pierre sait que si Jean est présent à la fin du dîner, il* *investira* $10K€$ *et que si Marie est présente à la fin du dîner, elle investira* $50K€$*. Pierre sait que Jean et Marie ne seront pas présents ensemble (car l'un deux doit garder leur fils), qu'au moins l'un d'entre eux viendra et que le cas "Jean vient et Marie ne vient pas" se produit avec une probabilité* 0.6*. Concernant le menu, Pierre peut commander du poisson ou de la viande pour le plat principal et du vin rouge ou du vin blanc pour le vin. Cependant, le restaurant refuse de servir du poisson avec du vin rouge. Jean n'aime pas le vin blanc et Marie n'aime pas la viande. Si le menu ne leur convient pas, alors ils quitteront la table. Si Jean vient, Pierre ne veut pas qu'il parte car il est son meilleur ami.*

Exemple 3.7. *Le problème du dîner utilise la structure d'utilité espérée associée à l'utilité espérée additive probabiliste (ligne 1) : la structure de plausibilité est* $(\mathbb{R}^+, +, \times)$*,* $\oplus_u = +$*,* $\otimes_{pu} = \times$*, et les utilités sont des gains additifs :* $(E_u, \otimes_u) = (\mathbb{R} \cup \{-\infty\}, +)$*, avec la convention que* $u + (-\infty) = -\infty$*.*

Hypothèses implicites et cadres non couverts Les hypothèses faites sur les structures de plausibilité, d'utilité et d'utilité espérée résultent de compromis toujours discutables entre des considérations sémantiques et algorithmiques : pouvoir englober le plus grand nombre possible de situations et d'approches, garder à l'esprit que les problèmes concrets doivent pouvoir être représentés de façon suffisamment compacte et résolus de façon raisonnablement efficace.

Par exemple, le fait de supposer l'existence d'ensembles E_p et E_u de degrés de plausibilité et d'utilité impose que plausibilités et utilités soient *cardinales*. Des approches purement ordinales comme par exemple les réseaux de préférences conditionnelles (*CP-nets* [14]) ne sont pas couvertes par le cadre proposé.

De la même façon, le fait que l'agrégation d'une plausibilité avec une utilité produise une utilité ($\otimes_{pu} : E_p \times E_u \to E_u$) repose sur une hypothèse implicite de *commensurabilité* entre plausibilités et utilités. Des travaux tels que [39], qui ne font pas cette hypothèse, ne sont pas couverts.

Enfin, les axiomes imposés impliquent que seules des plausibilités à caractère *distributionnel* sont couvertes : la plausibilité d'un ensemble E d'affectations des variables est supposée être complètement déterminée par la plausibilité de chacune des affectations complètes couvertes par E. De ce fait, des cadres non distributionnels comme les *fonctions de croyance* de Dempster-Shafer [96] ne sont pas couverts.

3.6 Résumé

Dans ce chapitre, nous avons introduit la notion de *structure d'utilité espérée*, qui constitue le premier élément clé du cadre PFU. Les structures algébriques définies spécifient comment les plausibilités sont combinées et synthétisées (via $\otimes_p$ et $\oplus_p$), comment les utilités sont combinées (via $\otimes_u$) et comment plausibilités et utilités sont prises en compte simultanément (via $\oplus_u$ et $\otimes_{pu}$). Plus précisément, les structures algébriques de base utilisées sont :

— un semi-anneau commutatif $(E_p, \oplus_p, \otimes_p)$ pour manipuler des plausibilités,

— un monoïde commutatif $(E_u, \otimes_u)$ pour manipuler des utilités,

— un semimodule $(E_p, E_u, \oplus_p, \otimes_{pu})$ pour calculer des utilités espérées.

L'ajout d'axiomes de monotonie à ces structures algébriques classiques permet d'obtenir ce que nous appelons structure de plausibilité, structure d'utilité, et structure d'utilité espérée. Ces structures couvrent des modèles variés de modélisation des plausibilités et des utilités. Elles sont inspirées de structures existantes définies par Friedman, Chu et Halpern [41, 47, 19]. Les différences principales tiennent à l'ajout d'axiomes permettant de traiter des problèmes de décision séquentielle (comportant plusieurs étapes de décision) et à l'ajout d'axiomes pour des raisons d'efficacité des algorithmes futurs.

Chapitre 4

Réseaux de Plausibilité-Faisabilité-Utilité

Le second élément du cadre PFU est un réseau de fonctions locales P_i, F_i et U_i (cf. équation 2.10 page 30) portant sur des variables d'un ensemble V. Ce réseau définit une représentation compacte et structurée des quantités globales que sont les plausibilités sur l'état de l'environnement, les faisabilités sur les décisions et les utilités sur l'environnement et les décisions. Ce chapitre définit de tels réseaux PFU et analyse les relations entre d'un côté l'expression de quantités globales par une combinaison de quantités locales et de l'autre la notion d'indépendance conditionnelle.

Nous appellons dorénavant *fonction de plausibilité* une fonction locale à valeurs dans E_p (l'ensemble des degrés de plausibilité), *fonction de faisabilité* une fonction locale à valeurs dans $\{t, f\}$ (l'ensemble des degrés de faisabilité) et *fonction d'utilité* une fonction locale à valeurs dans E_u (l'ensemble des degrés d'utilité).

4.1 Variables de décision et variables d'environnement

Les exemples introduits jusqu'alors mettent en évidence une distinction importante entre variables de décision, contrôlées par un agent décideur, et variables d'environnement, non contrôlées par un agent décideur.

En conséquence, le premier élément de la structure graphique proposée est un ensemble V de *variables à domaines finis*, partitionné en deux sous-ensembles : un ensemble V_D de *variables de décision* et un ensemble V_E de *variables d'environnement.*

Exemple 4.1. *Pour modéliser le problème du repas d'affaire, nous introduisons six variables : bp_J et bp_M (valeur t ou f), qui représentent respectivement les présences de Jean et Marie au début du repas, ep_J et ep_M (valeur t ou f), qui représentent leur présence en fin de repas, mc (valeur fish ou meat), qui représente le choix du plat principal, et w (valeur white ou red), qui représente le choix du vin. Ainsi, nous avons $V_D = \{mc, w\}$ et $V_E = \{bp_J, bp_M, ep_J, ep_M\}$.*

4.2 Vers des fonctions locales de faisabilité et de plausibilité

L'utilisation de fonctions locales pour représenter une quantité globale soulève certaines questions : comment et pourquoi des fonctions locales peuvent-elles être utilisées pour représenter des quantités globales et inversement, lorsque des fonctions locales sont exprimées, quel sens donner à leur agrégation. Nous montrons que ces questions sont fortement liées à une sorte d'équivalence entre factorisation et indépendance conditionnelle.

4.2.1 Une première étape de factorisation utilisant des indépendances conditionnelles

Il est tout d'abord possible d'étendre un résultat central sur les réseaux bayésiens [73] qui fait le lien entre le DAG associé à un réseau bayésien et la factorisation de la distribution de probabilité jointe exprimée par ce réseau. Ce résultat utilise la notion de *compatibilité* entre un DAG et une distribution de probabilité jointe : un DAG G sur un ensemble de variables V est dit compatible avec une distribution de probabilité jointe sur V si et seulement si chaque variable $x \in V$ est conditionnellement indépendante de ses non-descendants dans G, étant donnés ses parents dans G. Ceci conduit au théorème suivant : si G est un DAG sur un ensemble de variables V qui est compatible avec une distribution de probabilité jointe $\mathcal{P}_V$ sur V, alors $\mathcal{P}_V = \prod_{x \in V} \mathcal{P}_{x|pa_G(x)}$.

L'extension de ce résultat probabiliste au cadre plus général des plausibilités nécessite d'étendre la notion d'*indépendance conditionnelle*. Encore une fois, nous nous appuyons sur les travaux de Friedman et Halpern [41, 47], tout en nous en écartant sur certains aspects. Si S est un ensemble de variables, nous définissons une *distribution de plausibilité* sur S comme une fonction $\mathcal{P}_S$ de $dom(S)$ dans E_p, telle que $\underset{A \in dom(S)}{\oplus_p} \mathcal{P}_S(A) = 1_p$. Une distribution de plausibilité sur S induit une distribution de plausibilité sur tout $S' \subseteq S$: $\mathcal{P}_{S'} = \underset{dom(S-S')}{\oplus_p} \mathcal{P}_S$. A partir de là, pour des structures de plausibilités dites *conditionnables*, il est possible d'introduire la notion de *distribution de plausibilité conditionnelle* sur S_1 sachant S_2 (S_1 et S_2 disjoints), qui vérifie certaines propriétés telles que $\mathcal{P}_{S_1,S_2} = \mathcal{P}_{S_1 \mid S_2} \otimes_p \mathcal{P}_{S_2}$. Sur cette base, nous disons que S_1 est *conditionnellement indépendant* de S_2 sachant S_3 (S_1, S_2 et S_3 disjoints) si et seulement si $\mathcal{P}_{S_1,S_2|S_3} = \mathcal{P}_{S_1|S_3} \otimes_p \mathcal{P}_{S_2|S_3}$. Dans le cas des réseaux bayésiens, le DAG défini est un DAG dont les sommets représentent des variables. Il est parfois plus naturel d'utiliser un DAG dont les sommets correspondent à des ensembles de variables, comme ce qui est fait dans les graphes chaînés. Nous disons aussi qu'un DAG G sur un ensemble de composantes C (une composante étant un ensemble de variables) est compatible avec une distribution de plausibilité jointe sur $S = \cup_{c \in C} c$ si et seulement si chaque composante $c \in G$ est conditionnellement indépendante de ses non-descendantes dans G, étant données ses parentes dans G. Il est alors possible d'établir le théorème suivant :

Théorème 4.2. *Si un DAG G sur un ensemble de composantes C est compatible avec une distribution de plausibilité $\mathcal{P}_S$ jointe sur $S = \underset{c \in C}{\cup} c$, alors $\mathcal{P}_S = \underset{c \in C}{\otimes_p} \mathcal{P}_{c|pa_G(c)}$.*

Le théorème 4.2 nous fournit un premier niveau de factorisation composante par composante de la distribution de plausibilité jointe, via quelques étapes techniques permettant de pallier la présence de variables de différentes natures (variables de décision et variables d'environnement).

4.2.2 Etapes de factorisations supplémentaires

Mais il est possible d'aller plus loin puisque, pour chaque composante $c \in C$, la distribution de plausibilité conditionnelle $\mathcal{P}_{c|pa_G(c)}$ peut être elle-même factorisée en fonctions locales de plausibilité P_i associées à c. Si $Fact(c)$ représente l'ensemble des facteurs permettant d'exprimer $\mathcal{P}_{c|pa_G(c)}$, nous pouvons écrire $\mathcal{P}_{c|pa_G(c)} = \underset{P_i \in Fact(c)}{\otimes_p} P_i$. Cette seconde factorisation implique notamment que

$$\underset{c}{\oplus_p} (\underset{P_i \in Fact(c)}{\otimes_p} P_i) = 1_p$$

Ainsi, le résultat de l'agrégation des fonctions locales associées à une composante c est bien une distribution de plausibilité conditionnelle.

Par exemple, dans le cas d'un réseau bayésien, rien ne s'oppose à ce que les probabilités conditionnelles $\mathcal{P}_{x|pa_G(x)}$ s'expriment comme un produit de fonctions plus locales, éventuellement sous forme de contraintes en cas d'information binaire (0 ou 1 ; voir par exemple [29]). Dans certains cas, exprimer une décomposition de $\mathcal{P}_{c\,|\,pa_G(c)}$ est assez naturel. Dans d'autres cas, comme pour les champs de Markov [18], de telles factorisations supplémentaires peuvent être obtenues en utilisant des techniques systématiques. Enfin, on peut envisager d'utiliser d'autres définitions d'indépendance conditionnelle, comme par exemple celle utilisée pour les réseaux de valuation [99]. Ces étapes de factorisation supplémentaires sont intéressantes car réduire la taille des portées des fonctions locales utilisées peut être très avantageux d'un point de vue algorithmique.

Comme la structure de faisabilité est un cas particulier de structure de plausibilité, le même type de résultat peut être établi concernant les faisabilités, c'est-à-dire qu'une distribution de faisabilité globale peut s'exprimer comme une combinaison (plus exactement une conjonction) de faisabilités conditionnelles $\mathcal{F}_{c\,|\,pa_G(c)}$, chaque facteur $\mathcal{F}_{c\,|\,pa_G(c)}$ pouvant lui-même être exprimé comme une combinaison de fonctions locales.

Ceci nous montre finalement une façon naturelle permettant obtenir des fonctions locales de plausibilité et de faisabilité et un DAG de composantes représentant certaines normalisations.

Exemple 4.3. *Considérons le problème du repas d'affaire pour illustrer les deux étapes de factorisation. Afin d'obtenir le DAG, nous pouvons identifier des indépendances conditionnelles telles que "la présence de Jean en fin de repas est conditionnellement indépendante de la présence de Marie en fin de repas sachant la présence de Jean au début et le menu choisi". Les indépendances conditionnelles peuvent nous mener à définir le DAG de composantes donné à la figure 4.1(a). Nous avons par exemple une composante $\{bp_J, bp_M\}$ qui contient deux variables qui sont corrélées (on ne peut pas dire si bp_J a une influence causale sur bp_M ou si bp_M a une influence causale sur bp_J, c'est pourquoi il peut être naturel de mettre ces deux variables dans une même composante).*

Le théorème 4.2, qui définit en quelque sorte la première étape de factorisation, nous assure que les fonctions globales de plausibilité et de faisabilité se factorisent de la manière suivante : $\mathcal{P}_{V_E\,|\,V_D} = \mathcal{P}_{bp_J,bp_M} \times \mathcal{P}_{ep_J\,|\,bp_J,bp_M,mc,w} \times \mathcal{P}_{ep_M\,|\,bp_J,bp_M,mc,w}$ et $\mathcal{F}_{V_D\,|\,V_E} = \mathcal{F}_{mc,w}$.

La seconde étape de factorisation revient à exprimer chacun des facteurs ci-dessus en utilisant une combinaison de fonctions locales. Le facteur $\mathcal{P}_{bp_J,bp_M}$ peut être exprimé comme une combinaison de fonctions locales de plausibilités. La première, P_1, spécifie les probabilités de présences de Jean et Marie en début de repas : P_1 est défini par $P_1((bp_J, t).(bp_M, f)) = 0.6$, $P_1((bp_J, f).(bp_M, t)) = 0.4$, et $P_1((bp_J, t).(bp_M, t)) = P_1((bp_J, f).(bp_M, f)) = 0$. Il est possible d'ajouter une information *déterministe redondante via une seconde fonction de plausibilité* P_2*, définie comme la contrainte* $bp_J \neq bp_M$ *(*$P_2(A) = 1$ *si la contrainte est satisfaite,* 0 *sinon). Nous obtenons alors* $\mathcal{P}_{bp_J,bp_M} = P_1 \otimes_p P_2$ *et* $Fact(\{bp_J, bp_M\}) = \{P_1, P_2\}$*.*

$\mathcal{P}_{ep_J \mid bp_J,bp_M,mc,w}$ *peut également être spécifié comme une combinaison de deux fonctions de plausibilité* P_3 *et* P_4*.* P_3 *exprime que Jean n'est pas présent à la fin du repas s'il ne l'était pas au début :* P_3 *correspond à la contrainte dure* $(bp_J = f) \rightarrow (ep_J = f)$*. La fonction locale* $P_4 : (bp_J = t) \rightarrow ((ep_J = t) \leftrightarrow (w \neq white))$ *est une contrainte dure qui indique que Jean quitte le dîner si et seulement si du vin blanc est choisi. Ainsi, nous obtenons* $\mathcal{P}_{ep_J \mid bp_J,bp_M,mc,w} = P_3 \otimes_p P_4$ *et* $Fact(\{ep_J\}) = \{P_3, P_4\}$*. De manière analogue,* $\mathcal{P}_{ep_M \mid bp_J,bp_M,mc,w} = P_5 \otimes_p P_6$*, avec* P_5*,* P_6 *définies comme des contraintes et* $Fact(\{ep_M\}) = \{P_5, P_6\}$*.*

Concernant les faisabilités, $\mathcal{F}_{mc,w}$ *peut être exprimé par une contrainte dure indiquant que commander du vin rouge avec du poisson n'est pas permis :* $F_1 : \neg((mc = fish) \wedge (w = red))$ *et* $Fact(\{mc, w\}) = \{F_1\}$*. L'association entre fonctions locales et composantes est représentée à la figure 4.1(a).*

4.3 Fonctions locales d'utilité

Nous introduisons enfin des fonctions locales d'utilité portant indifféremment sur les variables de décision ou d'environnement. Soit ces fonctions locales sont directement définies, comme ce qui est fait dans les CSP ou les diagrammes d'influence, soit elles peuvent être obtenues à partir de travaux tels que [3], qui font le lien entre factorisation et indépendance conditionnelle pour les utilités lorsque $\otimes_u = +$.

Exemple 4.4. *Dans l'exemple du repas d'affaire, nous pouvons définir trois fonctions locales d'utilité. Une première fonction d'utilité binaire* U_1 *exprime que Pierre ne veut pas que Jean quitte le dîner :* U_1 *correspond à la contrainte dure* $(bp_J = t) \rightarrow (ep_J = t)$ *(*$U_1(A) = 0$ *si la contrainte est satisfaite,* $-\infty$ *sinon). Deux fonctions d'utilité unaires* U_2 *et* U_3*, définies par* $U_2((ep_J, t)) = 10$ *et* $U_2((ep_J, f)) = 0$ *et* $U_3((ep_M, t)) = 50$ *et* $U_3((ep_M, f)) = 0$ *respectivement, indiquent les gains obtenus en fonction des personnes présentes en fin de repas. Toutes les fonctions locales introduites jusqu'alors sont représentées à la figure 4.1(b).*

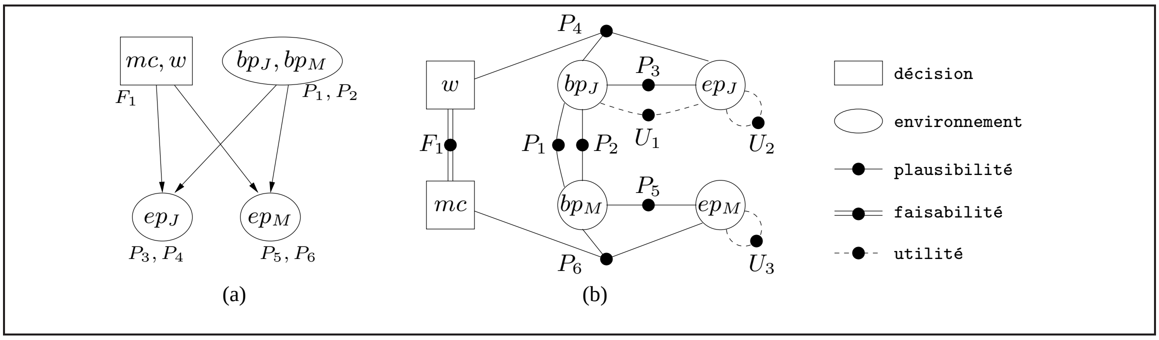


Figure 4.1: (a) DAG de composantes (b) Réseau de fonctions locales.

4.4 Définition formelle d'un réseau PFU

Ceci nous conduit à la définition suivante d'un réseau PFU :

Définition 4.5. *Un* réseau PFU *sur une structure d'utilité espérée est un quintuplet* (V, G, P, F, U) *tel que*

- $V = \{x_1, x_2, \ldots\}$ *est un ensemble fini de variables à domaines finis, partitionné en un ensemble* V_D *de variables de décision et un ensemble* V_E *de variables d'environnement*
- G *est un DAG dont les sommets, appelés composantes, sont une partition de* V*, plus précisément l'union d'une partition* $\mathcal{C}_D$ *de* V_D *et d'une partition* $\mathcal{C}_E$ *de* V_E*.*
- $P = \{P_1, P_2, \ldots\}$ *est un ensemble fini de fonctions de plausibilité. Chaque* $P_i \in P$ *est associée à une unique composante* $c \in \mathcal{C}_E$ *qui satisfait* $sc(P_i) \subset c \cup pa_G(c)$*. L'ensemble des fonctions de plausibilité* $P_i \in P$ *associées à une composante* $c \in \mathcal{C}_E$ *est noté* $Fact(c)$ *et doit satisfaire* $\underset{c}{\oplus_p}\left(\otimes_{p\,P_i \in Fact(c)} P_i\right) = 1_p$*.*
- $F = \{F_1, F_2, \ldots\}$ *est un ensemble fini de fonctions de faisabilité. Chaque* $F_i \in F$ *est associée à une unique composante* $c \in \mathcal{C}_D$ *qui satisfait* $sc(F_i) \subset c \cup pa_G(c)$*. L'ensemble des fonctions de faisabilité* $F_i \in F$ *associées à une composante* $c \in \mathcal{C}_D$ *est noté* $Fact(c)$ *et doit satisfaire* $\underset{c}{\vee}\left(\wedge_{F_i \in Fact(c)} F_i\right) = t$*.*
- $U = \{U_1, U_2, \ldots\}$ *est un ensemble fini de fonctions d'utilité.*

4.5 Liens avec des cadres existants

Le cadre des réseaux PFU couvre de nombreux cadres existant en Intelligence Artificielle.

Par exemple, un *problème de satisfaction de contraintes* (CSP) Pb peut être modélisé comme un PFU $(V, G, \emptyset, \emptyset, U)$ où V contient toutes les variables de Pb, considérées comme des variables de décision, où G contient une seule composante englobant toutes les variables et où U contient toutes les contraintes de Pb, considérées comme des fonctions locales d'utilité (si elles étaient considérées comme des fonctions locales de faisabilité, les conditions de normalisation interdiraient de considérer des CSP incohérents). La démarche est identique pour un *problème de satisfiabilité d'une formule booléenne* (SAT), à la seule différence que les variables sont booléennes et les contraintes des clauses. Il en est de même pour les CSP *valués* et, malgré les apparences, pour les *formules booléennes quantifiées* (QBF) et les CSP *quantifiés* (QCSP) ; les différences apparaîtront au niveau des requêtes. Avec les CSP *mixtes*, la distinction est faite entre variables de décision et variables d'environnement. Avec les *problèmes de satisfiabilité stochastiques* et les CSP *stochastiques*, des composantes apparaissent et des fonctions locales de plausibilité s'ajoutent aux fonctions locales d'utilité.

Un *réseau bayésien* Pb peut être modélisé comme un réseau PFU $(V, G, P, \emptyset, \emptyset)$ où V contient toutes les variables de Pb, considérées comme des variables d'environnement, où une composante est associée à chaque variable, où le DAG G est celui de Pb et où P contient les distributions de probabilité conditionnelles de Pb (une par variable), considérées comme des fonctions locales de plausibilité.

Un *diagramme d'influence* Pb peut aussi être modélisé comme un PFU $(V, G, P, \emptyset, U)$: V contient les variables d'environnement et les variables de décision de Pb ; une composante est associée à chaque variable ; G est le DAG associé à Pb sans les variables d'utilité et sans les arcs vers les variables de décision et d'utilité ; P contient des fonctions locales de plausibilité correspondant aux distributions de probabilité conditionnelles sur les variables d'environnement et U contient des fonctions locales d'utilité (une associée à chaque variable d'utilité u de Pb, dont la portée est l'ensemble des parents de u dans le DAG associé à Pb). Avec les *réseaux de valuation*, apparaissent en plus des fonctions locales de faisabilité.

Un *processus décisionnel markovien* (MDP) à horizon fini Pb peut aussi être modélisé comme un PFU $(V, G, P, \emptyset, U)$: pour chaque instant i, une variable d'environnement s_i est associée à l'état à l'instant i et une variable de décision d_i à la décision à l'instant i ; une composante est associée à chaque variable ; G est le DAG où, pour chaque instant i, s_{i+1} a pour parent s_i et d_i ; P contient des fonctions locales de plausibilité correspondant aux probabilités de transition $P_{s_{i+1}|s_i,d_i}$ et U contient des fonctions locales d'utilité correspondant aux gains locaux U_{s_i,d_i}. Pour les *processus partiellement observables* (POMDP), il est nécessaire d'ajouter, pour chaque instant i, une variable d'environnement o_i associée à l'observation à l'instant i, d'indiquer dans le DAG que pour chaque instant i, o_i a pour parent s_i et d'ajouter les probabilités d'observation $P_{o_i|s_i}$ aux fonctions locales de plausibilité. Avec les MDP *factorisés*, les variables s_i sont décomposées en autant de variables que de facteurs. Avec les MDP *possibilistes* [91], rien ne change, hormis les opérateurs d'agrégation et d'élimination utilisés.

Il en est de même pour un *problème de planification* à horizon fixé h dans le cadre STRIPS. Il peut être modélisé comme un PFU (V, G, P, F, U) : une variable d'environnement s_i est associée à chaque instant i et à chaque composante booléenne de l'état ; une variable de décision d_i est associée à chaque instant ; G est le DAG où, pour chaque instant i, d_i a *a priori* pour parents tous les s_i, et s_{i+1} a pour parents d_i et tous les s_i ; P contient des fonctions locales de plausibilité correspondant aux effets des actions, F contient des fonctions locales de faisabilité correspondant aux pré-conditions des actions et U contient des fonctions d'utilité booléenne décrivant les différents buts et portant sur les variables s_h.

4.6 Résumé

Dans ce chapitre, nous avons introduit le second élément du cadre PFU : un réseau de fonctions locales de plausibilité, de faisabilité et d'utilité, avec un DAG capturant des conditions de normalisation. La factorisation de plausibilités, faisabilités et utilités globales en fonctions locales a été reliée à la notion d'indépendance conditionnelle. Ceci nous permet d'obtenir une méthode constructive pour spécifier des fonctions locales représentant une fonction globale donnée. D'un point de vue purement technique, la définition des réseaux PFU (définition 4.5) est relativement succinte.

Chapitre 5

Requêtes sur un réseau PFU

Une requête correspond à une tâche de raisonnement concernant l'information exprimée par un réseau PFU. Des exemples de requêtes informelles concernant le problème du repas d'affaire sont les suivantes :

1. "Quel est le meilleur choix de menu si Pierre ne sait pas qui est présent au début ?"
2. "Quel est le meilleur choix de menu si Pierre sait exactement qui vient ?"
3. "Comment maximiser l'investissement espéré si le restaurant choisit le plat principal en premier et si Pierre est pessimiste concernant ce choix, si ensuite la présence des convives est observée, et si enfin Pierre choisit le vin ?"

La distinction entre réseaux PFU et requêtes sur un réseau PFU est notamment cohérente avec la démarche actuelle utilisée dans le cadre des diagrammes d'influence qui consiste à relaxer les liens appelés *liens d'information* (*Unconstrained Influence Diagrams* [52], *Limited Memory Influence Diagrams* [61]). Elle rend explicite le fait que les requêtes ne changent pas les relations locales qui existent entre les variables.

Dans ce chapitre, nous définissons une classe de requêtes sur des réseaux PFU. Nous supposons qu'une *séquence de décisions* doit être effectuée et que l'ordre dans lequel les décisions sont prises et les observations sont réalisées est connu. Nous faisons aussi une hypothèse de *no-forgetting*, c'est-à-dire que lorsqu'une décision est prise par un agent, cet agent se rappele de toutes les décisions et de toutes les observations préalablement réalisées. A partir de maintenant, nous supposons également que l'ensemble des degrés d'utilité est *totalement* ordonné. Dans le contexte d'un calcul et d'une exécution automatique de décisions, cette hypothèse d'ordre total, qui est valide dans des formalismes variés, permet de toujours pouvoir identifier des règles de décision optimales. Voir la section 5.6 pour une discussion sur l'extension de ce travail au cas d'un ordre partiel sur les utilités.

Deux définitions de la réponse à une requête sont fournies, la première formalisant un processus de raisonnement fondé sur des arbres de décision et la seconde étant plus opérationnelle. Il est montré que les deux définitions introduites sont équivalentes.

5.1 Modélisation d'une requête

Les exemples de requêtes introduits au chapitre 1 montrent des séquences d'éliminations de variables qui sont à chaque fois cohérentes avec l'ordre des décisions et des observations et qui utilisent des opérateurs d'élimination fonctions de la nature des variables (décision ou environnement) et, pour les variables de décision, fonctions du caractère coopératif ou non des agents. Ceci nous conduit à la définition suivante d'une *requête* sur un réseau PFU :

Définition 5.1. *Une requête sur un réseau PFU* (V, G, P, F, U) *est une séquence Sov de paires (opérateur d'élimination-ensemble de variables)* (op, S) *telle que*

1. *tous les* S *sont disjoints ;*
2. *soit* $S \subseteq V_D$ *(variables de décision) et op* $= \min$ *ou* $\max$, *soit* $S \subseteq V_E$ *(variables d'environnement) et op* $= \oplus_u$ *;*
3. *les variables qui n'apparaissent pas dans Sov sont des variables de décision ;*
4. *pour toute paire* (x, y) *de variables de* V *de natures différentes (l'une est une variable de décision, l'autre est une variable d'environnement) telle que la composante contenant* x *est ascendante de la composante contenant* y *dans* G, x *n'apparaît pas à droite de* y *dans Sov (ce qui implique qu'elle apparaisse à gauche ou qu'elle soit libre).*

La dernière condition est une condition relative à la causalité qui permet d'exclure des requêtes du type $\oplus_{u bp_J, bp_M, ep_J, ep_M} \max_{mc,w}$, dans lesquelles il est implicitement supposé que les présences à la fin sont connues avant le choix du menu, ce qui n'est pas causalement cohérent.

La définition d'une requête n'impose pas que toutes les variables de V apparaissent dans *Sov*. Les variables qui n'y apparaissent pas sont dites *libres*, les autres *liées*. Notons que rien n'interdit que dans certains cadres $\oplus_u = \min$ ou $\oplus_u = \max$, ce qui veut dire que min ou max peuvent parfois être utilisés pour éliminer des variables de décision et des variables d'environnement.

5.2 Réponse à une requête : définition sémantique utilisant les arbres de décision

Il est possible de définir la réponse à une requête en utilisant les concepts d'*arbre de décision* et de *loterie* présentés dans l'exemple du chapitre 1. Le résultat est le suivant sous l'hypothèse que la structure de plausibilité soit conditionnable et que toutes les variables soient liées (si toutes les variables ne sont pas liées, la réponse à une requête est simplement une fonction de l'affectation des variables libres) :

Définition 5.2. *La réponse sémantique Sem-Ans*(Q) *à une requête* $Q = (Sov, (V, G, P, F, U))$ *est définie récursivement de la façon suivante, si* $\mathcal{P}_V$, $\mathcal{F}_V$ *et* $\mathcal{U}_V$ *sont respectivement les fonctions globales de plausibilité, de faisabilité et d'utilité associées au réseau PFU :*

$$\begin{array}{ll}
(1) & \mathit{Sem\text{-}Ans}(Q) = \mathcal{V}_{Sr}(Sov, Pb, \emptyset) \\
(2) & \mathcal{V}_{Sr}(\emptyset, Pb, A) = \mathcal{U}_V(A) \\
(3) & (x \in V_D) \wedge (op = \min) \rightarrow \\
 & \mathcal{V}_{Sr}((op, x).Sov, Pb, A) = \min\limits_{a \in dom(x)} \mathcal{F}_V((x,a)|A) \star \mathcal{V}_{Sr}(Sov, Pb, A.(x,a)) \\
(4) & (x \in V_D) \wedge (op = \max) \rightarrow \\
 & \mathcal{V}_{Sr}((op, x).Sov, Pb, A) = \max\limits_{a \in dom(x)} \mathcal{F}_V((x,a)|A) \star \mathcal{V}_{Sr}(Sov, Pb, A.(x,a)) \\
(5) & (x \in V_E) \wedge (op = \oplus_u) \rightarrow \\
 & \mathcal{V}_{Sr}((op, x).Sov, Pb, A) = \underset{a \in dom(x)}{\oplus_u} \mathcal{P}_V((x,a)|A) \otimes_{pu} \mathcal{V}_{Sr}(Sov, Pb, A.(x,a))
\end{array}$$

Cette définition, sémantiquement justifiée par les concepts d'arbre de décision et de loterie, pose des problèmes au niveau algorithmique. En effet, le calcul de $\mathit{Sem\text{-}Ans}(Q)$ nécessite l'évaluation de quantités telles que $\mathcal{F}_V((x,a)|A)$ et $\mathcal{P}_V((x,a)|A)$ qui ne sont pas présentes telles quelles dans la définition d'un réseau PFU et dont le calcul (de type calcul d'une probabilité marginale) peut être de complexité exponentielle en fonction du nombre de variables non affectées par A.

5.3 Réponse à une requête : définition opérationnelle

Il est cependant possible de considérer une autre définition de la réponse à une requête qu'on peut qualifier d'*opérationnelle* dans la mesure où elle ne présente pas les mêmes difficultés algorithmiques que la précédente. En supposant que toutes les variables sont liées (l'hypothèse d'une structure de plausibilité conditionnable n'est pas ici nécessaire) :

Définition 5.3. *La réponse opérationnelle* $\mathit{Op\text{-}Ans}(Q)$ *à une requête* Q *est définie récursivement de la façon suivante :*

$$\begin{array}{ll}
(1) & \mathit{Op\text{-}Ans}(Q) = \mathcal{V}_{Or}(Sov, Pb, \emptyset) \\
(2) & \mathcal{V}_{Or}(\emptyset, Pb, A) = ((\underset{F_i \in F}{\wedge} F_i) \star ((\underset{P_i \in P}{\otimes_p} P_i) \otimes_{pu} (\underset{U_i \in U}{\otimes_u} U_i)))(A) \\
(3) & \mathcal{V}_{Or}((op, x).Sov, Pb, A) = op_{a \in dom(x)} \mathcal{V}_{Or}(Sov, Pb, A.(x,a))
\end{array}$$

Elle peut aussi s'écrire de la manière suivante :

$$\mathit{Op\text{-}Ans}(Q) = Sov((\underset{F_i \in F}{\wedge} F_i) \star (\underset{P_i \in P}{\otimes_p} P_i) \otimes_{pu} (\underset{U_i \in U}{\otimes_u} U_i)) \tag{5.1}$$

c'est-à-dire qu'elle correspond exactement à l'équation 2.10.

Dans la mesure où elle fait appel uniquement à des quantités présentes dans la définition d'un réseau PFU, cette définition est algorithmiquement plus attrayante. Sa justification sémantique est cependant moins claire.

5.4 Théorème d'équivalence

Il est heureusement possible de lever ces difficultés en établissant le théorème suivant d'équivalence entre les deux définitions proposées :

Théorème 5.4. *Si la structure de plausibilité est conditionnable, les réponses sémantique et opérationnelle à une requête correcte sont identiques : Sem-Ans*(Q) = *Op-Ans*(Q)*. De plus, les règles de décision qui peuvent être obtenues dans les deux cas en mémorisant pour chaque décision la(les) valeur(s) optimisante(s) (l'*argmax *ou l'*argmin*) sont identiques.*

Ce théorème donne une justification sémantique à la définition opérationnelle algorithmiquement intéressante, ce qui ne veut pas dire qu'avec cette définition les choses seront forcément algorithmiquement faciles. Du fait de ce théorème d'équivalence, nous notons la réponse à une requête simplement par $Ans(Q)$.

Si on considère par exemple, pour le problème du trésor, la requête ayant pour séquence d'élimination $\min_{li_P} \max_{li_J} \sum_{he_J} \max_{do} \sum_{he_P} \sum_{ga,tr}$, la réponse est : 2900€ (espérance de gain de Jean). La politique associée de Pierre (faux ami de Jean) est d'écouter indifféremment à la porte A ou B. Si Pierre écoute à la porte A (respectivement B), la politique optimale de Jean est d'écouter à la porte B (respectivement A). En ce qui concerne la porte à ouvrir, Jean n'a le choix qu'entre A et B. S'il n'entend pas le gangster, sa politique optimale est d'ouvrir la porte à laquelle il a écouté. Sinon, sa politique optimale est d'ouvrir l'autre porte.

Il est également possible de définir la notion de *requête bornée*, qui correspond à un problème de décision consistant à déterminer si la réponse à une requête a une valeur supérieure à un certain seuil (et à trouver des règles de décision associées).

5.5 Requêtes classiques dans les cadres couverts

Avec un *problème de satisfaction de contraintes* (CSP) $Pb = (V, C)$, la requête classique utilise $Sov = \max_V$ (avec $\max = \exists$). Il en est de même pour un problème SAT. Pour un CSP *valué*, elle utilise aussi $Sov = \min_V$, avec un min qui dépend de la structure de valuation choisie.

Avec une *formule booléenne quantifiée* (QBF) ou un CSP *quantifié* (QCSP) apparaît une libre alternance entre $\min = \forall$ et $\max = \exists$. Avec un *problème de satisfiabilité stochastique* (SSAT) ou un CSP *stochastique* (SCSP), on trouve aussi une libre alternance entre max pour les variables de décision et $+$ pour les variables d'environnement.

Avec un *réseau bayésien*, le calcul d'une distribution de probabilité sur une variable y se ramène à des requêtes utilisant $Sov = \sum_{V-\{y\}}$. La recherche de l'hypothèse la plus vraisemblable sur la valeur d'une variable y amène à considérer cette variable comme une variable de décision et à considérer une requête utilisant $Sov = \max_y \cdot \sum_{V-\{y\}}$.

Avec un *diagramme d'influence* ou un *réseau de valuation Pb*, on a une alternance entre max pour les variables de décision et $+$ pour les variables d'environnement, sachant que l'ordre d'élimination doit être cohérent avec le DAG associé à Pb.

Avec un *processus décisionnel markovien* (MDP) à horizon fini, on a une alternance, à chaque instant i, entre max pour d_i et $+$ pour s_{i+1}. Avec un *processus partiellement observable* (POMDP), on a la même alternance entre max pour d_i et $+$ pour o_{i+1}, mais tous les s_i (qui sont non directement observables) sont repoussés à la fin de la séquence et éliminés avec $+$. Il est intéressant de remarquer qu'avec un MDP possibiliste pessimiste, on a une alternance entre max pour d_i et min pour s_{i+1}, alors qu'avec un MDP possibiliste optimiste, max est le seul opérateur d'élimination (aussi bien pour d_i que pour s_{i+1}), ce qui donne une structure algébrique identique à celle des VCSP possibilistes.

Avec un *problème de planification* à horizon fixé dans le cadre STRIPS, on se retrouve dans la même situation que dans le cadre CSP, du fait que $\max = \oplus_u = \vee$.

Plus formellement, il est possible d'établir le théorème suivant :

Théorème 5.5. *Les requêtes et les requêtes bornées permettent d'exprimer et de résoudre la liste suivante de problèmes :*

1. *dans le cadre des problèmes de satisfiabilité : SAT, QBF, stochastic SAT (SSAT) et extended-SSAT [62].*
2. *cadre des CSP :*
   - *tester la cohérence d'un CSP [63] ; trouver une solution à un CSP, compter le nombre de solutions d'un CSP.*
   - *Chercher une solution pour un CSP valué [94].*
   - *Résoudre un QCSP [13].*
   - *Trouver des règles de décision conditionnelles ou inconditionnelles pour un CSP mixte ou un CSP mixte probabiliste [38].*
   - *Trouver une politique optimale pour un CSP stochastique ou trouver une politique ayant une valeur supérieure à un certain seuil ; résoudre un problème d'optimisation stochastique [107].*
3. *Programmation linéaire en nombres entiers [95] avec des variables à domaines finis.*
4. *Recherche d'un plan solution de longueur $\leq k$ pour un problème de planification classique (planification à la STRIPS [40, 44]).*
5. *Résoudre des requêtes classiques sur un réseau bayésien [73], sur un champ de Markov [18] ou sur des graphes chaînés [42], avec des plausibiltés exprimées comme des probabilités, des possibilités ou des κ-rankings :*
   - *Calculer des distributions de probabilité.*
   - *Résoudre un problème MAP (Maximum A Posteriori hypothesis).*
   - *Résoudre un problème MPE (Most Probable Explanation).*
   - *Calculer la plausibilité d'une observation ou la plausibilité d'une clause logique dans un réseau hybride [29].*
6. *Résoudre un diagramme d'influence [48].*
7. *Avec un horizon fini, résoudre un MDP probabiliste, un MDP possibiliste, un MDP basé sur les κ-rankings, complètement ou partiellement observable (POMDP), factorisé ou non [86, 68, 91, 16, 15].*

5.6 Extensions à d'autres classes de requêtes

Les requêtes pourraient être rendues plus complexes en relâchant certaines hypothèses :

— Telle quelle, la définition des requêtes s'applique si l'ensemble des degrés d'utilité E_u est totalement ordonné par $\preceq_u$. Etendre les résultats à des ordres *partiels* est quasi-immédiat lorsque $(E_u, \preceq_u)$ est un *treillis*, ce qui permet au cadre PFU de subsumer les *semiring-based CSP* [9, 10]. D'autres extensions à des ordres partiels devraient pouvoir permettre d'englober les MDP algébriques [74].

— L'hypothèse de *no-forgetting* pourrait également être relâchée, dans l'esprit des diagrammes d'influence à mémoire limitée (*limited memory influence diagrams*, LIMIDs [61]), dans lesquels on cherche des règles de décision fonctions uniquement de certaines variables, et non de toutes les variables présentes dans l'historique des observations.

— L'ordre dans lequel les décisions sont prises et les observations sont réalisées a été supposé connu. Dans certaines situations, il peut être utile non seulement de calculer des règles de décisions optimales, mais aussi de déterminer un ordre optimal dans lequel prendre des décisions et faire des observations. Dans cette voie, les travaux sur les diagrammes d'influence avec décisions non totalement ordonnées [52] pourraient être considérés.

— Enfin, relâcher l'hypothèse de variables à domaines finis n'est pas direct car transformer un + en intégrale ou faire des opérations d'optimisation sur des domaines continus requiert quelques propriétés supplémentaires. Dans cette direction, tous les travaux concernant les réseaux temporels simples (*Simple Temporal Problems*, STPs [32]) and leurs extensions [53, 103, 105, 90] peuvent être considérés. Dans ces problèmes, les variables sont des points dans le temps à valeurs dans des intervalles continus et des contraintes de durée entre certains de ces points sont définies. Cependant, dans ces formalismes, la gestion de l'incertain s'applique uniquement à des non déterminismes booléens, qui spécifient quelles sont les évolutions possibles et impossibles de l'environnement.

5.7 Résumé

Ce chapitre a introduit le dernier élément clé du cadre PFU : une classe de requêtes sur des réseaux PFU. La réponse à une requête a été définie dans un premier temps à partir d'arbres de décision. Le premier résultat important de ce chapitre est le théorème 5.4, qui donne un fondement théorique à une seconde définition de la réponse à une requête, équivalente à la première. Cette seconde définition, qui identifie la réponse à une requête à une séquence d'éliminations de variables sur une combinaison de fonctions locales, est mieux adaptée aux besoins des algorithmes futurs étant donnée qu'elle utilise uniquement les fonctions locales définies par un réseau PFU. Le second résultat important est le théorème 5.5 qui montre que nombre de requêtes usuelles peuvent être exprimées comme des requêtes PFU.

Au final, le cadre PFU est entièrement spécifié par les définitions 3.4, 3.5, 3.6 pour la structure algébrique, par la définition 4.5 pour les réseaux PFU et par la définition 5.1 et l'équation 5.1 pour les requêtes.

5.8 Conclusion de la partie I : gains et coûts du cadre PFU

Une meilleure compréhension Le théorème 5.5 montre que de nombreux cadres existants sont des instances du cadre PFU. A travers cette unification, les similitudes et les différences entre les formalismes existants peuvent apparaître plus clairement. Par exemple, une comparaison des VCSP et de la version optimiste des MDP possibilistes, à travers la définition opérationnelle de la réponse à une requête révèle que d'un point de vue purement algébrique, un MDP possibiliste optimiste (complètement ou partiellement observable) n'est qu'une forme spécifique de CSP flou. Il s'ensuit que les librairies disponibles pour résoudre des VCSP peuvent être directement utilisées pour résoudre de tels MDP.

Du point de vue théorique, une étude de la complexité temporelle et spatiale du calcul de l'équation 2.10 (page 30) peut permettre d'obtenir des bornes de complexité à moindre coût sur plusieurs formalismes simultanément. On peut également chercher à caractériser des propriétés induisant une complexité théorique donnée.

Expressivité accrue Le pouvoir d'expression accru des réseaux PFU est le résultat d'une flexibilité présente à plusieurs niveaux : (1) flexibilité du modèle de plausibilité/utilité ; (2) flexibilité en termes de réseaux possibles ; (3) flexibilité des requêtes considérées en termes de scénarios modélisables. Cette expressivité permet aux réseaux PFU de couvrir des formes génériques de problèmes de décision séquentielle avec plausibilités, faisabilités et utilités, avec des agents coopératifs ou antagonistes, des observabilités partielles et des paramètres dans le processus décisionnel via la présence de variables libres.

Aucun des cadres indiqués dans le théorème 5.5 (le théorème de subsumption) ne présente une telle flexibilité. Plus spécifiquement, comparé aux diagrammes d'influence [48, 52, 101, 71, 51] ou aux réseaux de valuations (VN [98, 100, 34]), le cadre PFU peut manipuler autre chose que de l'utilité espérée additive probabiliste, il permet de faire des éliminations avec l'opérateur min pour modéliser la présence d'agents antagonistes. Ainsi, les formules booléennes quantifiées ne peuvent pas être représentées par l'intermédiaire d'un diagramme d'influence ou d'un réseau de valuation, mais elles peuvent l'être par une requête PFU. En outre, les réseaux PFU font intervenir un DAG capturant certaines conditions de normalisation concernant les plausibilités et les faisabilités, alors que les réseaux de valuation n'encapsulent pas cette information. Comparé aux diagrammes d'influence ou aux réseaux de valuation dits séquentiels (*sequential influence diagrams* [51], *sequential valuation networks* [34]), le cadre PFU est aussi capable d'exprimer des problèmes de décision dits *assymétriques*, dans lesquels une variable de décision peut ne pas avoir à être considérée dans le processus décisionnel, via l'ajout de valeurs supplémentaires dans le domaine de certaines variables.

En fait, certains problèmes simples qui peuvent être exprimés comme des requêtes PFU ne pouvent pas être définis directement dans d'autres formalismes. Parmi ces problèmes, on citer des problèmes faisant intervenir "faisabilités *avec conditions de normalisation* + exigences dures" (l'utilisation d'un CSP pour modéliser un tel problème fait perdre l'information fournie par les conditions de normalisation, à moins d'utiliser des variables supplémentaires permettant définir des contraintes sur les contraintes). De même pour les problèmes de "décision séquentielle du type diagramme d'influence, raisonnant à partir d'utilités espérées possibilistes", qui pourraient être baptisés diagrammes d'influence possibilistes.[1] De même pour l'instance "CSPs stochastiques sans hypothèse de contingence", pour l'instance "max-QBF" (analogue au problème max-SAT),

[1] Les diagrammes d'influence possibilistes ont été proposés très récemment dans un travail effectué parallèlement à cette thèse [43]. Ce formalisme est une instanciation simple du cadre PFU.

ou pour l'instance "VCSP quantifiés", qui pourrait correspondre à des VCSP faisant intervenir une alternance de minimisations et de maximisations modélisant la présence de décideurs antagonistes. Ainsi, le cadre PFU couvre de nouveaux formalismes.

Le coût de la flexibilité et de l'expressivité accrue est que le cadre PFU ne peut être décrit aussi succintement que par exemple le cadre des CSP.

Algorithmes génériques La partie II va montrer qu'il est possible de définir des algorithmes génériques pour répondre à une requête sur un réseau PFU. Comme indiqué précédemment, construire des algorithmes génériques peut générer des fertilisations croisées, dans le sens où cela peut permettre à chacun des formalismes couverts de bénéficier de techniques développées dans un autre formalisme couvert. Cette démarche est en adéquation avec les efforts réalisés au cours de ces dernières années pour généraliser des méthodes de résolution utilisées parallèlement pour résoudre des problèmes différents. Par exemple, la propagation de contraintes souples est un outil qui améliore grandement la résolution des VCSP ; intégrer un tel outil dans un solver générique défini dans le cadre PFU permettrait de l'utiliser directement pour résoudre des diagrammes d'influence. L'utilisation d'opérateurs algébriques abstraits peut également permettre d'identifier des propriétés algorithmiquement intéressantes, ou d'induire des conditions nécessaires ou suffisantes pour qu'un algorithme particulier soit utilisable.

Cependant, certaines techniques peuvent être fortement liées à un formalisme spécifique ou à un type de problème, et dans ce cas un algorithme dédié peut être notoirement plus efficace qu'un algorithme générique. Une solution minimisant la portée de cette objection est simplement de dire que le cadre PFU est une opportunité pour généraliser de tels algorithmes dédiés, en caractérisant clairement les propriétés algébriques qui le rendent si efficace. De plus, même si des algorithmes spécialisés sont généralement meilleurs que des algorithmes génériques, il existe des cas dans lesquels des outils génériques s'avèrent plus performants que des algorithmes spécialisés. Voir par exemple [93] ou l'utilisation de solvers SAT pour résoudre des problèmes de planification à la STRIPS.

Deuxième partie

Algorithmes génériques pour répondre à des requêtes sur des réseaux PFU

Chapitre 6

Premiers algorithmes génériques

Le cadre PFU est un cadre flexible qui unifie plusieurs formalismes existants. On pourrait penser qu'un écueil inhérent à cette généricité est que répondre à une requête sur un réseau PFU est hors de portée, du moins si on souhaite le faire de manière efficace. Un des objectifs des chapitres à venir est de montrer qu'il n'en est rien et que la possibilité des répondre efficacement ou non à une requête est une conséquence de la requête considérée elle-même, et non de la généricité.

Initialement, le cadre PFU a été construit non seulement pour ses aptitudes en termes de représentation de la connaissance, mais aussi pour permettre de définir des algorithmes génériques capables de répondre à des requêtes. Certains choix ont même été justifiés par des considérations algorithmiques. En d'autres termes, l'objectif est de pouvoir répondre à des requêtes aussi efficacement que possible, et non uniquement de les exprimer. Dans ce qui suit, nous introduisons des schémas de résolution génériques qui sont soit des généralisations d'algorithmes déjà existants, soit des techniques nouvelles applicables directement à tous les formalismes couverts par le cadre PFU. Ce chapitre présente deux premières approches permettant de répondre à des requêtes PFU dans le cas général, c'est-à-dire sans hypothèses algébriques supplémentaires. Ces approches sont toutes deux developpées à partir de la définition opérationnelle de la réponse à une requête, définie par $Ans(Q) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i))$. Plus précisément, nous introduisons :

- un algorithme générique de recherche arborescente très basique ;
- un algorithme générique utilisant des mécanismes d'élimination de variables [6], afin d'exploiter au mieux la factorisation en fonction locales.

Des résultats de complexité théorique fonctions d'un paramètre appelé largeur induite contrainte sont également fournis.

6.1 Un algorithme naïf de recherche arborescente

Sur la base du cadre proposé et de la définition opérationnelle de la réponse à une requête, il est possible de définir des algorithmes génériques capables de répondre à n'importe quelle requête sur n'importe quel réseau PFU.

Un premier algorithme possible est un algorithme de type *recherche arborescente* qui utilise comme ordre d'affectation des variables un ordre compatible avec la séquence d'élimination Sov utilisée dans la requête : si deux variables x et y appartiennent à deux paires (opérateur d'élimination-ensemble de variables) différentes pov_x et pov_y et si pov_x est située avant pov_y (à gauche) dans Sov, x est affectée avant y ; si elles appartiennent à une même paire, leur ordre d'affectation est indifférent (du point de vue du résultat, pas forcément du point de vue de l'efficacité algorithmique).

Cet algorithme collecte au niveau de chaque feuille f la combinaison des plausibilités, des faisabilités et des utilités pour l'affectation complète associée à f. Il synthétise au niveau de chaque nœud non feuille n les valeurs fournies par les nœuds fils de n en utilisant l'opérateur d'élimination associé à n. La figure 6.1 montre le pseudo-code d'un tel algorithme (avec toujours l'hypothèse que toutes les variables sont liées). La réponse à la requête Sov sur un PFU Pb est fournie par l'appel **RechArbPFU**$(Sov, Pb, \emptyset)$.

```
RechArbPFU$(Sov, (V, G, P, F, U), A)$
début
    si $Sov = \emptyset$ alors
        retourner $((\wedge_{F_i \in F} F_i) \star (\otimes_{p\,P_i \in P} P_i) \otimes_{pu} (\otimes_{u\,U_i \in U} U_i))(A)$
    sinon
        $(op, S).Sov' \leftarrow Sov$
        choisir $x \in S$
        si $S = \{x\}$ alors $Sov \leftarrow Sov'$ sinon $Sov \leftarrow (op, S - \{x\}).Sov'$
        $d \leftarrow dom(x)$
        $res \leftarrow \Diamond$
        tant que $d \neq \emptyset$ faire
            choisir $a \in d$
            $d \leftarrow d - \{a\}$
            $res \leftarrow op(res,$ RechArbPFU$(Sov, Pb, A.(x, a)))$
        retourner $res$
fin
```

Figure 6.1: Un algorithme générique de type recherche arborescente.

Cet algorithme naïf a une complexité *temporelle exponentielle* en fonction du nombre de variables, mais une complexité *spatiale polynomiale*.

Ceci nous fournit au moins une information intéressante sur la *classe de complexité* du problème de réponse à une requête PFU, problème noté AnswerPFU : sachant que le problème QBF (satisfiabilité d'une formule booléenne quantifiée) est *PSPACE-complet* et est un sous-problème du problème AnswerPFU, ce dernier est au moins *PSPACE-dur* ; mais sachant qu'il est *PSPACE* (existence d'un algorithme de complexité spatiale polynomiale), il est forcément *PSPACE-complet* [1].

6.2 Elimination de variables : une première version naïve

Un autre algorithme possible est un algorithme de type *élimination de variables* [6, 97, 25, 56] utilisant comme ordre d'élimination un ordre inverse de celui utilisé par la recherche arborescente : si deux variables x et y appartiennent à deux paires (opérateur d'élimination-ensemble de variables)

[1] En toute rigueur, le problème AnswerPFU n'est pas un problème de décision et il faudrait parler du problème de décision associé, par exemple, le problème consistant à déterminer si la réponse à une requête est supérieure ou égale à une valeur fixée.

différentes pov_x et pov_y et si pov_x est située avant pov_y (à gauche) dans Sov, y est éliminée avant x ; si elles appartiennent à une même paire, leur ordre d'élimination est indifférent.

Cet algorithme commence par combiner toutes les fonctions locales de plausibilité, de faisabilité et d'utilité en une seule fonction globale d'utilité. Il en élimine ensuite les variables les unes après les autres en utilisant à chaque fois l'opérateur d'élimination associé. La figure 6.2 montre le pseudo-code d'un tel algorithme.

```
ElimVarPFU($Sov, Pb$)
début
    $L_0 \leftarrow ((\wedge_{F_i \in F} F_i) \star (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i))$
    tant que $Sov \neq \emptyset$ faire
        $Sov'.(op, S) \leftarrow Sov$
        choisir $x \in S$
        si $S = \{x\}$ alors $Sov \leftarrow Sov'$ sinon $Sov \leftarrow Sov'.(op, S - \{x\})$
        $L_0 \leftarrow op_x\, L_0$
    retourner $L_0$
fin
```

Figure 6.2: Un algorithme générique naïf de type élimination de variables.

Cet algorithme naïf est pire que le précédent puisqu'il a une complexité *temporelle* et *spatiale exponentielle*. De la même façon que le premier algorithme de recherche arborescente, cet algorithme n'exploite pas la *factorisation* en fonctions locales de plausibilité, de faisabilité et d'utilité. Sans propriétés algébriques supplémentaires, il est en fait impossible d'utiliser des algorithmes d'élimination classiques, comme par exemple *bucket elimination* [25], qui exploitent la structure de l'hypergraphe des fonctions locales.

La raison principale est qu'aucun axiome n'est imposé concernant les relations entre l'opérateur $\otimes_u$ et les autres opérateurs. Plus precisément, il est par exemple impossible dans le cas général de calculer la quantité $\underset{x}{\oplus_u} P_x \otimes_{pu} (U_x \otimes_u U_y)$ en ne considérant que les fonctions locales qui dépendent de x.

Nous avons retenu deux ensembles de propriétés supplémentaires, parce qu'ils sont l'un ou l'autre satisfaits par chacun des cadres classiques et parce qu'ils permettent l'un et l'autre l'utilisation de ces algorithmes d'élimination classiques. Dans ce qui suit, les opérateurs de combinaison $\otimes$ sur E sont étendus à $E \cup \{\Diamond\}$ par $e \otimes \Diamond = \Diamond \otimes e = \Diamond$.

6.3 Deux conditions suffisantes de décomposabilité

Les deux axiomes retenues permettant de n'avoir à considérer que les fonctions locales ayant x dans leur portée lorsqu'une variable x est éliminée sont notés Ax^{SG} et Ax^{SA} et s'écrivent de la manière suivante :

$$Ax^{SA} : \begin{cases} \otimes_u \text{ distributif par rapport à } \oplus_u \\ \text{et } p \otimes_{pu} (u_1 \otimes_u u_2) = (p \otimes_{pu} u_1) \otimes_u u_2 \text{ pour tout } (p, u_1, u_2) \in E_p \times E_u \times E_u \end{cases}$$

$$Ax^{SG} : \text{“}\otimes_u = \oplus_u \text{ sur } E_u\text{” (mais pas sur } E_u \cup \{\Diamond\})$$

La première condition suffisante de décomposabilité est notée Ax^{SA} comme “axiome du cas semi-anneau", car il confère à $(E_u, \oplus_u, \otimes_u)$ une structure de semi-anneau. La seconde condition suffisante de décomposabilité est notée Ax^{SG} comme "axiome pour le cas semi-groupe', car il rend la structure $(E_u, \oplus_u, \otimes_u)$ en quelque sorte équivalente à la structure $(E_u, \oplus_u)$, qui est un semi-groupe. Le tableau 6.1 montre que ces deux axiomes disjoints couvrent des structures classiques d'utilité espérée.

	E_p	E_u	$\otimes_u$	$\oplus_u$	$\otimes_{pu}$	Axiome satisfait
1	$\mathbb{R}^+$	$\mathbb{R} \cup \{-\infty\}$	$+$	$+$	$\times$	SG
2	$\mathbb{R}^+$	$\mathbb{R}^+$	$\times$	$+$	$\times$	SA
3	$[0,1]$	$[0,1]$	min	max	min	SA
4	$[0,1]$	$[0,1]$	min	min	$\max(1{-}p, u)$	SG
5	$\mathbb{N} \cup \{\infty\}$	$\mathbb{N} \cup \{\infty\}$	$+$	min	$+$	SA
6	$\{t, f\}$	$\{t, f\}$	$\wedge$	$\vee$	$\wedge$	SA
7	$\{t, f\}$	$\{t, f\}$	$\wedge$	$\wedge$	$\rightarrow$	SG
8	$\{t, f\}$	$\{t, f\}$	$\vee$	$\vee$	$\wedge$	SG
9	$\{t, f\}$	$\{t, f\}$	$\vee$	$\wedge$	$\rightarrow$	SA

Table 6.1 – Structures d'utilité espérées satisfaisant Ax^{SA} ou Ax^{SG} : (1) utilité espérée probabiliste avec utilités additives (permet de calculer l'espérance d'un gain ou d'un coût), (2) utilité espérée probabiliste avec utilités multiplicatives (permet de calculer la probabilité que des contraintes soient satisfaites), (3) utilité espérée possibiliste optimiste, (4) utilité espérée possibiliste pessimiste, (5) utilité qualitative avec kappa-rankings et utilités uniquement positives, (6) utilité espérée booléenne optimiste avec utilités conjonctives, (7) utilité espérée booléenne pessimiste avec utilités conjonctives, (8) utilité espérée booléenne optimiste avec utilités disjonctives, (9) utilité espérée booléenne pessimiste avec utilités disjonctives.

La proposition 6.1 montre que dès que l'un des deux axiomes est satisfait, l'élimination d'une variable peut se faire en ne considérant que les fonctions locales qui dépendent de cette variable. Etant donné un ensemble de fonctions locales Φ et une variable x, nous notons Φ^{+x} l'ensemble des fonctions ayant x dans leur portée ($\Phi^{+x} = \{\varphi \in \Phi \,|\, x \in (\varphi)\}$) et $\Phi^{-x} = \Phi - \Phi^{+x}$. De plus, étant donnés des ensembles P, F et U de fonctions locales de plausibilité, de faisabilité et d'utilité respectivement, nous notons $(\wedge_{F_i \in F} F_i) \star (\otimes_{p P_i \in P} P_i) \otimes_{pu} (\otimes_{u U_i \in U} U_i)$ sous la forme $F \star P \otimes_{pu} U$, c'est-à-dire que nous considérons la combinaison de fonctions du même type comme implicite.

Proposition 6.1. *Soit $(E_p, E_u, \oplus_u, \otimes_{pu})$ une structure d'utilité espérée. Soit P et U des ensembles de fonctions locales de plausibilité et d'utilité respectivement.*

Si l'axiome Ax^{SA} est verifié, alors

$$\underset{x}{\oplus_u} (P^{+x} \otimes_{pu} U) \quad = \quad U^{-x} \otimes_u (\underset{x}{\oplus_u} (P^{+x} \otimes_{pu} U^{+x})) \tag{6.1}$$

Si l'axiome Ax^{SG} est vérifié, alors

$$\underset{x}{\oplus_u} (P^{+x} \otimes_{pu} U) \quad = \quad ((\underset{x}{\oplus_p} P^{+x}) \otimes_{pu} U^{-x}) \otimes_u (\underset{x}{\oplus_u} (P^{+x} \otimes_{pu} U^{+x})) \tag{6.2}$$

Cette proposition peut être illustrée par les structures utilisées pour la satisfaction espérée probabiliste et pour l'utilité additive espérée probabiliste. Dans le premier cas, qui satisfait Ax^{SA}, il est possible d'écrire $\sum_x (P^{+x} \times U) = U^{-x} \times (\sum_x (P^{+x} \times U^{+x}))$, alors que dans le second cas, nous pouvons écrire $\sum_x (P^{+x} \times (U^{-x} + U^{+x})) = ((\sum_x P^{+x}) \times U^{-x}) + (\sum_x (P^{+x} \times U^{+x}))$.

6.4 Algorithme d'élimination de variables amélioré

Comme nous allons le voir, les axiomes Ax^{SA} et Ax^{SG} permettent de calculer la réponse à une requête en utilisant un véritable algorithme d'élimination de variables, c'est-à-dire un algorithme permettant d'éliminer une variable x en ne considérant que les fonctions locales dépendantes de x.

6.4.1 Algorithme amélioré dans le cas semi-anneau

Lorsque l'axiome Ax^{SA} est satisfait, il est possible de se ramener à une structure algébrique beaucoup plus simple : il est possible de montrer que l'on peut transformer l'espace des degrés de plausibilité E_p en l'espace des degrés d'utilité E_u via un morphisme $\phi : p \to p \otimes_{pu} 1_u$. Grâce à ce morphisme, il est possible de montrer qu'il suffit de travailler sur un seul espace E égal à E_u faisant intervenir un seul opérateur de combinaison $\otimes$ égal à $\otimes_u$ et un seul opérateur d'élimination $\oplus$ égal à $\oplus_u$. En d'autres termes, l'axiome Ax^{SA} est équivalent à l'axiome suivant :

$$Ax^{SA'} : \begin{cases} (E_p, \preceq_p) = (E_u, \preceq_u) = (E, \preceq) \\ \otimes_p = \otimes_{pu} = \otimes_u = \otimes \\ \oplus_p = \oplus_u = \oplus \end{cases}$$

La structure algébrique du cadre PFU devient alors beaucoup plus simple.

Définition 6.2. $(E, \oplus, \otimes)$ *is un semi-anneau commutatif monotone totalement ordonné (*totally ordered Monotonic Commutative Semiring (MCS)*) si et seulement si* $(E, \oplus, \otimes)$ *est un semi-anneau commutatif muni d'un ordre total tel que* $\oplus$ *et* $\otimes$ *sont monotones pour cet ordre total.*

Proposition 6.3. $(E_p, E_u, \oplus_u, \otimes_{pu})$ *est une structure d'utilité espérée totalement ordonnée qui satisfait* $Ax^{SA'}$ *(les structures de plausibilité et d'utilité espérée sous-jacente étant* $(E_p, \oplus_p, \otimes_p)$ *et* $(E_u, \otimes_u)$ *respectivement) si et seulement si* $(E_u, \oplus_u, \otimes_u)$ *est un MCS totalement ordonné.*

Ainsi, lorsque $Ax^{SA'}$ est satisfait, la structure algébrique du cadre PFU est tout simplement un MCS totalement ordonné $(E, \oplus, \otimes) = (E_u, \oplus_u, \otimes_u)$. La condition de normalisation sur les composantes d'environnement devient

$$\bigoplus_c \left(\bigotimes_{P_i \in Fact(c)} P_i \right) = 1_E$$

et la réponse opérationnelle à une requête prend la forme

$$Ans(Q) = Sov\left(\left(\bigwedge_{F_i \in F} F_i \right) \star \left(\bigotimes_{\varphi \in P \cup U} \varphi \right) \right) \tag{6.3}$$

En outre, au lieu d'exprimer les faisabilités sur $\{t, f\}$, nous pouvons les exprimer sur $\{1_E, \Diamond\}$ en associant la valeur 1_E à t et la valeur $\Diamond$ à f. Cet artifice technique préserve la valeur d'une requête car $t \star u = 1_E \otimes u$ et $f \star u = \Diamond \otimes u$. La réponse à une requête Q devient $Ans(Q) = Sov(\otimes_{\varphi \in P \cup F \cup U} \varphi)$. Ainsi, le cas semi-anneau peut nécessiter plusieurs opérateurs d'élimination (min, max et $\oplus$) mais ne requiert en réalité qu'un seul opérateur de combinaison ($\otimes$).

Proposition 6.4. *Soit* $(E, \oplus, \otimes)$ *un MCS totalement ordonné. Nous étendons* $\oplus$ *et* $\otimes$ *sur* $E \cup \{\Diamond\}$ *par* $u \oplus \Diamond = \Diamond \oplus u = u$ *et* $u \otimes \Diamond = \Diamond \otimes u = \Diamond$*. Alors, pour tout* $op \in \{\min, \max, \oplus\}$, $(E \cup \{\Diamond\}, op, \otimes)$ *est un semi-anneau commutatif.*

La propriété précédente assure notamment une distributivité de l'opérateur de combinaison $\otimes$ sur tous les opérateurs d'élimination utilisés. Ce résultat est essentiel pour pouvoir utiliser l'algorithme d'élimination de variables donné à la figure 6.3, qui exploite la factorisation en fonctions locales. Le premier appel est **VE-answerQ**$(Sov, \otimes, P \cup F \cup U)$.

```
**VE-answerQ**$(Sov, \circledast, \Phi)$
**début**
    **si** $Sov = \emptyset$ **alors retourner** $\Phi$
    **sinon**
        $Sov'.(op, S) \leftarrow Sov$
        choisir $x \in S$
        **si** $S = \{x\}$ **alors** $Sov \leftarrow Sov'$ **sinon** $Sov \leftarrow Sov'.(op, S - \{x\})$
        $\varphi_0 \leftarrow op_x\left(\circledast_{\varphi \in \Phi^{+x}} \varphi\right)$
        $\Phi \leftarrow (\Phi - \Phi^{+x}) \cup \{\varphi_0\}$
        **retourner** $\textit{VE-answerQ}(Sov, \circledast, \Phi)$
**fin**
```

Figure 6.3: Un algorithme d'élimination de variables générique utilisant les factorisations disponibles (Sov : séquence d'éliminations, $\circledast$: opérateur de combinaison, Φ : ensemble de fonctions locales).

6.4.2 Algorithme amélioré dans le cas semi-groupe

Le cas semi-groupe nécessite un peu plus de travail que le cas semi-anneau. La raison est que l'équation 6.2 ne crée pas uniquement une nouvelle fonction d'utilité résultant de l'élimination de x : elle crée d'une part une nouvelle fonction de plausibilité $\underset{x}{\oplus_p} P^{+x}$ qui doit être combinée avec toutes les fonctions d'utilité de U^{-x}, et d'autre part une nouvelle fonction d'utilité $\underset{x}{\oplus_u} (P^{+x} \otimes_{pu} U^{+x})$. Autrement dit, la quantité obtenue après élimination de x n'est pas directement de la forme $Sov'(F' \star P' \otimes_{pu} U')$, avec Sov' la séquence d'élimination restant à traiter et F', P' et U' des nouveaux ensembles de fonctions locales.

Afin d'exploiter une forme générale qui ne varie pas durant les différentes étapes d'éliminations, une solution consiste à travailler sur des couples (fonction de plausibilité,fonction d'utilité) appelés des *potentiels* [70] (ces potentiels n'ont rien à voir avec les potentiels utilisés dans un champ de Markov).

Définition 6.5. *Un* potentiel *est une paire* (P_0, U_0) *composée d'une fonction de plausibilité* P_0 *et d'une fonction d'utilité* U_0*. Deux opérateurs sont définis sur les paires plausibilité-utilité :*

— *un opérateur de combinaison* $\boxtimes$ *défini par* $(p_1, u_1) \boxtimes (p_2, u_2) = (p_1 \otimes_p p_2, (p_1 \otimes_{pu} u_2) \otimes_u (p_2 \otimes_{pu} u_1))$,

— *un opérateur d'élimination* $\boxplus$ *défini par* $(p_1, u_1) \boxplus (p_2, u_2) = (p_1 \oplus_p p_2, u_1 \oplus_u u_2)$.

Enfin, un ordre partiel est défini sur les paires plausibilité-utilité par $(p, u_1) \preceq (p, u_2)$ *ssi* $u_1 \preceq u_2$.

Dans ce qui suit, nous considérons également chaque fonction de faisabilité comme un potentiel. Comme seul un ordre partiel est défini sur les potentiels, certaines étapes techniques sont requises pour assurer que des opérations de minimisation ou de maximisation soient bien définies. Ces étapes techniques impliquent de travailler sur des réseaux PFU dits raffinés et nécessitent une légère modification de la définition de Φ^{+x} pour un ensemble de fonctions Φ. Ces étapes permettent progressivement d'arriver au résultat suivant :

Proposition 6.6. *Soit $Q = (Sov, \mathcal{N})$ une requête. Soit $T(Sov)$ la séquence de paires opérateur-variables obtenus à partir de Sov en remplaçant les $\oplus_u$ par des $\boxplus$.*

VE-answerQ$(T(Sov), \boxtimes, \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\})$ *renvoie un ensemble de potentiels* Π *tel que* $\boxtimes_{\varphi \in \Pi} \varphi(A) = \begin{cases} (1_p, Ans(Q)(A)) \text{ si } Ans(Q)(A) \neq \Diamond \\ \Diamond \text{ sinon} \end{cases}$

Notons que comme dans le cas semi-anneau, avec le passage de Ax^{SA} à $Ax^{SA'}$, il est possible de transformer l'axiome Ax^{SG} en un axiome $Ax^{SG'}$ plus simple permettant d'utiliser seulement deux opérateurs paramétrables ($\oplus$ et $\otimes$) pour définir la réponse à une requête. Mais cette fois, la transformation n'est pas gratuite, c'est-à-dire qu'elle nécessite certaines conditions telles que la possibilité de translater l'échelle des utilités pour obtenir une structure non bipolaire, c'est-à-dire avec des degrés d'utilité soit tous inférieurs à 0_u soit tous supérieurs à 0_u. Avec l'axiome $Ax^{SG'}$, la réponse à une requête s'écrit :

$$Ans(Q) = Sov((\bigwedge_{F_i \in F} F_i) \star (\bigotimes_{P_i \in P} P_i) \otimes (\bigoplus_{U_i \in U} U_i)) \tag{6.4}$$

avec $(E, \oplus, \otimes)$ un MCS totalement ordonné.

6.4.3 Cas général

Les cas semi-anneau et semi-groupe définissent deux conditions *suffisantes* permettant d'utiliser la factorisation en fonctions locales. Montrer à quel point ces conditions sont nécesssaires est un problème ouvert. Lorsque ni Ax^{SA} ni Ax^{SG} n'est satisfait, on peut se ramener à un calcul du type :

$$Ans(Q) = Sov((\bigwedge_{F_i \in F} F_i) \star (\bigotimes_{P_i \in P} P_i) \otimes U_0) = Sov(\bigotimes_{\varphi \in P \cup F \cup \{U_0\}} \varphi) \tag{6.5}$$

Ainsi, le cas général est un cas particulier du cas semi-anneau à condition d'agrèger toutes les fonctions locales d'utilité pour obtenir une unique fonction globale d'utilité. L'algorithme **VE-answerQ** peut à nouveau être utilisé, avec **VE-answerQ**$(Sov, \otimes, P \cup F \cup \{U_0\})$ comme premier appel.

Le tableau 6.2 résume l'utilisation de l'algorithme **VE-answerQ** pour répondre à une requête PFU. Notons que pour chaque cas, *aucune hypothèse algébrique supplémentaire n'est nécessaire.*

CAS	PREMIER APPEL
semi-anneau (Ax^{SA})	**VE-answerQ**$(Sov, \otimes, P \cup F \cup U)$
semi-groupe (Ax^{SG})	**VE-answerQ**$(T(Sov), \boxtimes, \{(P_i, 1_u), P_i \in P\} \cup F \cup \{(1_p, U_i), U_i \in U\})$
cas général	**VE-answerQ**$(Sov, \otimes, P \cup F \cup \{U_0\})$, with $U_0 = \otimes_{u U_i \in U} U_i$

Table 6.2 – Utilisation de l'algorithme **VE-answerQ**.

6.5 Evaluation de la complexité théorique

Cette section fournit des bornes supérieures sur la complexité temporelle et spatiale de l'algorithme **VE-answerQ**. Les résultats de complexité sont exprimés en fonction d'un paramètre connu sous le nom de largeur induite contrainte [50, 72]. Les bornes données sont valables pour tous les formalismes couverts par le cadre PFU.

6.5.1 Largeur induite

La largeur induite [28, 27] est un paramètre définissant une borne supérieure sur la complexité théorique des algorithmes classiques d'élimination de variables. Elle est aussi connue sous le nom de largeur d'arbre [88] ou de *k-tree number* [2]. Etant donnée une requête mono-opérateur sur un modèle graphique (V, Φ), la largeur induite est définie à partir de l'hypergraphe $\mathcal{G} = (V, \{sc(\varphi) \mid \varphi \in \Phi\})$ associé à ce modèle graphique.

Définition 6.7. *Un ordre d'élimination o sur un ensemble de variables $V = \{x_1, \ldots, x_n\}$ est une bijection de $\{1, \ldots, n\}$ dans V. Pour tout $k \in \{1, \ldots, n\}$, $o(k)$ est appelé la k-ème variable éliminée dans o.*

Un ordre d'élimination o induit un ordre total $\preceq$ sur V, défini par $o(n) \prec \ldots \prec o(2) \prec o(1)$, $x \prec y$ signifiant que y doit être éliminée avant x. Ceci nous permet de considérer de manière abusive que o est un ordre total sur V.

Définition 6.8. *(Largeur induite d'un ordre d'élimination) Soit $\mathcal{G} = (V_{\mathcal{G}}, H_{\mathcal{G}})$ un hypergraphe. Soit o un ordre d'élimination sur $V_{\mathcal{G}}$. o peut être utilisé pour générer une séquence d'hypergraphes $\mathcal{G}_1, \ldots, \mathcal{G}_{n+1}$ (avec $n = |V_{\mathcal{G}}|$), définie par*

- $\mathcal{G}_1 = \mathcal{G}$
- *si $\mathcal{G}_k = (V_k, H_k)$ et si x est la k-ème variable éliminée dans o, alors $\mathcal{G}_{k+1} = (V_k - \{x\}, (H_k - H_k^{+x}) \cup \{h_{k+1}\})$, avec H_k^{+x} l'ensemble des hyper-arêtes de H_k impliquant la variable x et $h_{k+1} = (\cup_{h \in H_k^{+x}} h) - \{x\}$ l'hyper-arête créée de l'étape k à l'étape $k+1$ (étape d'élimination de variable).*

La largeur induite de $\mathcal{G}$ pour l'ordre d'élimination o, notée $w_{\mathcal{G}}(o)$ est égale à la taille maximale des hyper-arêtes créées, c'est-à-dire $w_{\mathcal{G}}(o) = \max_{k \in \{1,\ldots,n\}} |h_{k+1}|$.

Informellement, l'hyper-arête h_{k+1} créée de l'étape k à l'étape $k+1$ est obtenue en considérant l'ensemble H_k^{+x} des hyper-arêtes de $\mathcal{G}_k$ qui "dépendent" de x et en "reliant" toutes les variables impliquées dans H_k^{+x} (sauf x). Ceci signifie que l'élimination de x crée une nouvelle fonction locale de portée h_{k+1}. Le nombre $1 + w_{\mathcal{G}}(o)$ correspond au nombre maximal de variables à considérer simultanément pendant les étapes d'élimination de variables.

Un résultat connu est que les complexités temporelles et spatiales d'un algorithme classique d'élimination de variables sont exponentielles en la largeur induite de l'ordre d'élimination utilisé.

Définition 6.9. *(Largeur d'un hypergraphe $\mathcal{G}$) Soit $\mathcal{G} = (V_{\mathcal{G}}, H_{\mathcal{G}})$ un hypergraphe. La largeur de $\mathcal{G}$, notée $w_{\mathcal{G}}$, est égale à la largeur minimale induite par un ordre d'élimination sur $V_{\mathcal{G}}$: si $\mathcal{O}$ représente l'ensemble de tous les ordres d'élimination possibles sur $V_{\mathcal{G}}$, alors $w_{\mathcal{G}} = \min_{o \in \mathcal{O}} w_{\mathcal{G}}(o)$.*

La largeur induite d'un hypergraphe est égale au nombre minimal de variables à considérer simultanément par un algorithme d'élimination de variables lorsqu'un ordre d'élimination optimal est utilisé. Le problème de décision associé à la recherche d'un ordre d'élimination optimal est cependant un problème NP-complet [2]. Si seulement un sous-ensemble des variables doit être éliminé, alors la largeur induite est définie de manière similaire. Dans ce qui suit, nous considérons que l'ensemble des variables à éliminer est implicite.

6.5.2 Largeur induite contrainte

Les requêtes multi-opérateurs imposent certaines contraintes sur l'ordre d'élimination des variables. La complexité est alors donnée par la *largeur induite contrainte* [50, 72].

Définition 6.10. *Soit $\preceq$ un ordre partiel sur V. L'ensemble des* linéarisations *de $\preceq$, noté $lin(\preceq)$, est l'ensemble des ordres totaux $\preceq'$ sur V qui satisfont $(x \preceq y) \rightarrow (x \preceq' y)$.*

Définition 6.11. *Soit $\mathcal{G} = (V_\mathcal{G}, H_\mathcal{G})$ un hypergraphe et soit $\preceq$ un ordre partiel sur $V_\mathcal{G}$. La largeur induite contrainte $w_\mathcal{G}(\preceq)$ de $\mathcal{G}$ avec contraintes sur l'ordre d'élimination données par $\preceq$ ("$x \prec y$" signifie "y doit être éliminée avant x") est définie par $w_\mathcal{G}(\preceq) = \min_{o \in lin(\preceq)} w_\mathcal{G}(o)$.*

Les contraintes sur l'ordre d'élimination induites par une séquence d'éliminations de variables *Sov* sont formellement définies de la manière suivante :

Définition 6.12. *Soit $Q = (Sov, \mathcal{N})$ une requête sur un réseau PFU tel que $Sov = (op_1, S_1) \cdot (op_2, S_2) \cdots (op_q, S_q)$. L'*ordre partiel *$\preceq_{Sov}$ induit par Sov est donné par $S_1 \prec_{Sov} S_2 \prec_{Sov} \ldots \prec_{Sov} S_q$. Cet ordre partiel force les variables de S_j à être éliminées avant les variables de S_i dès que $i < j$.*

Par exemple, l'ordre partiel induit par la séquence d'élimination $Sov = \min_{x_1,x_2} \sum_{x_3,x_4} \max_{x_5}$ est défini par $\{x_1, x_2\} \prec_{Sov} \{x_3, x_4\} \prec_{Sov} x_5$.

Proposition 6.13. *Soit $\mathcal{G} = (V, \Phi)$ un modèle graphique. Soit Sov une séquence de paires opérateurs-variables sur V. Avec un ordre d'élimination optimal, l'algorithme* ***VE-answerQ***$(Sov, \circledast, \Phi)$ *a une complexité spatiale et temporelle $O(|\Phi| \cdot d^{1+w_\mathcal{G}(\preceq_{Sov})})$, avec d la taille maximale du domaine des variables de V.*

Ainsi, étant donné une requête $Q = (Sov, \mathcal{N})$ sur un réseau PFU $\mathcal{N} = (V, G, P, F, U)$,

— répondre à une requête dans le cas semi-anneau ou semi-groupe est $O(|P \cup F \cup U| \cdot d^{1+w_\mathcal{G}(\preceq_{Sov})})$ en temps et en espace, avec $\mathcal{G} = (V, \{sc(\varphi) \mid \varphi \in P \cup F \cup U\})$ l'hypergraphe associé au réseau PFU ;

— répondre à une requête dans le cas général est $O((|P| + |F| + 1) \cdot d^{1+w_\mathcal{G}(\preceq_{Sov})})$, avec $\mathcal{G} = (V, \{sc(\varphi) \mid \varphi \in P \cup F \cup \{U_0\}\})$ l'hypergraphe associé au réseau PFU dans lequel toutes les fonctions locales d'utilité sont fusionnées pour donner une fonction d'utilité unique U_0.

6.6 Vers une diminution de la largeur induite contrainte

Puisqu'une variation linéaire de la largeur induite engendre une variation exponentielle des complexités temporelle et spatiale, il peut être utile de diminuer la largeur induite.

6.6.1 Relâchement de contraintes sur l'ordre d'élimination

Les contraintes sur l'ordre d'élimination sont définies par l'ordre partiel $\preceq_{Sov}$ induit par la séquence d'élimination Sov. Ce choix peut être inutilement contraignant car certaines éliminations avec des opérateurs distincts peuvent parfois être interverties si l'on tient compte des portées des fonctions locales en jeu.

En effet, si l'on considère par exemple un calcul du type

$$\max_{x_1,\dots,x_q} \sum_y \max_{x_{q+1}} \left(P_y \times c_{y,x_1} \times \prod_{i\in\{1,\dots,q\}} c_{x_i,x_{q+1}} \right).$$

alors la largeur induite contrainte vaut q car dans ce cas, $\preceq_{Sov}$ force la variable x_{q+1}, qui est liée avec q autres variables, à être éliminée en premier. Cependant, les portées des fonctions locales garantissent que le calcul suivant donne le même résultat :

$$\max_{x_1} \left(\left(\sum_y P_y \times c_{y,x_1} \right) \times \left(\max_{x_2,\dots,x_{q+1}} \left(\prod_{i\in\{1,\dots,q\}} c_{x_i,x_{q+1}} \right) \right) \right).$$

Dans ce cas, la largeur induite devient égale à 1, par exemple en utilisant l'ordre d'élimination $x_{q+1} \prec x_q \prec \dots \prec x_2 \prec x_1 \prec y$. Ainsi, la complexité théorique passe de $O(d^{q+1})$ à $O(d^2)$.

Par conséquent, définir les contraintes sur l'ordre d'élimination uniquement à partir de Sov est trop contraignant et potentiellement exponentiellement sous-optimal. Ainsi, il est possible de révéler certaines libertés dans l'ordre d'élimination en tenant compte des portées des fonctions locales.

6.6.2 Travail sur l'hypergraphe des fonctions locales

Tout d'abord, les conditions de normalisation peuvent être utilisées pour ne pas effectuer de calculs inutiles du type $\sum_x P_{x \mid pa(x)}$ lorsque $P_{x \mid pa(x)}$ est une distribution de probabilité conditionnelle sur x sachant $pa_G(x)$.

Ensuite, il peut exister des décompositions qui utilisent plus que la distributivité des opérateurs d'élimination sur les opérateurs de combinaison. Par exemple, si l'on considère un diagramme d'influence associé au calcul

$$\max_{x_1,\dots,x_q} \sum_y P_y \cdot \left(U_{y,x_1} + \dots + U_{y,x_q} \right)$$

alors la largeur induite contrainte vaut q. Cependant, étant donné que $\sum_S (U_1 + U_2) = (\sum_S U_1) + (\sum_S U_2)$, on peut écrire

$$\max_{x_1,\dots,x_q} \sum_y P_y \cdot \left(U_{y,x_1} + \dots + U_{y,x_q} \right) = (\max_{x_1} \sum_y P_y \cdot U_{y,x_1}) + \dots + (\max_{x_q} \sum_y P_y \cdot U_{y,x_q})$$

Cette *duplication* de la variable y permet d'obtenir un calcul de largeur 1. Le processus de duplication est utilisable dès que l'opérateur d'élimination est égal à l'opérateur de combinaison. Cette technique peut s'appliquer aux éliminations avec $\forall$ pour les QBF et les QCSP, aux éliminations avec min pour les MDP possibilistes ou avec + sur les diagrammes d'influence.

6.7 Résumé

Dans ce chapitre, nous avons introduit un algorithme d'élimination de variable nommé **VE-answerQ**. Cet algorithme peut répondre à une requête tout en utilisant les factorisations de quantités globales en fonctions locales, pourvu qu'une des deux conditions de décomposabilité (axiomes Ax^{SA} et Ax^{SG}) soit satisfaite. L'utilisation de cet algorithme est résumée par le tableau 6.2, qui montre que dans le cas semi-anneau, son utilisation est plutôt naturelle, que dans le cas semi-groupe, elle requiert l'utilisation d'éléments appelés potentiels, et que dans le cas général, elle nécessite de combiner toutes les fonctions d'utilité locales en une fonction d'utilité globale.

Le principe de cet algorithme est d'éliminer les variables dans un ordre compatible avec la séquence d'éliminations multi-opérateurs Sov. Ses complexités temporelle et spatiale sont exponentielles en la largeur induite contrainte. Comme indiqué dans la dernière partie de ce chapitre, cet algorithme ne profite cependant pas toujours de la *structure réelle* d'une requête multi-opérateur, et ce pour plusieurs raisons :

1. Premièrement, il est restrictif de définir des contraintes sur l'ordre d'élimination uniquement à partir de la séquence d'éliminations de variables : certaines libertés dans l'ordre des éliminations peuvent apparaître si la portée des fonctions locales est prise en compte.
2. Ensuite, l'algorithme **VE-answerQ** utilise essentiellement la distributivité d'un opérateur de combinaison par rapport à des opérateurs d'élimination, alors qu'il peut exister des décompositions supplémentaires utilisant le mécanisme de duplication précédemment introduit.
3. Enfin, les réseaux PFU contiennent certaines conditions de normalisation qui n'ont jusqu'alors pas été utilisées. C'est un tort, car les conditions de normalisation peuvent complètement masquer la complexité réelle d'un problème.

Utiliser ces trois mécanismes peut diminuer la largeur induite contrainte et conduire à des gains exponentiels en termes de complexité théorique. Ce constat nous amène à définir des techniques plus sophistiquées permettant de faire apparaître la structure réelle des requêtes multi-opérateurs de manière automatique.

Chapitre 7

Structuration des requêtes multi-opérateurs

La largeur induite contrainte peut être améliorée en analysant la structure des requêtes multi-opérateurs, afin de révéler les contraintes réelles sur l'ordre d'élimination, de faire apparaître les décompositions possibles et de supprimer des calculs inutiles. Ce faisant, des gains exponentiels en termes de complexité théorique peuvent être obtenus.

Les techniques introduites dans ce chapitre visent à automatiser le processus de structuration d'une requête. Elles ne sont pas juste des généralisations de techniques déjà existantes définies dans des formalismes couverts par le cadre PFU et peuvent par conséquent contribuer à la résolution des formules booléennes quantifiées, des problèmes de satisfiabilité stochastique, des CSP quantifiés ou stochastiques, des diagrammes d'influence probabilistes ou possibilistes, ou encore des MDP factorisés.

Les étapes de structuration d'une requête vont nous mener progessivement à une nouvelle architecture de calcul appelée l'architecture des *DAGs de clusters multi-opérateurs*. Cette dernière permet de répondre à une requête plus efficacement (en termes de largeur induite) qu'avec l'algorithme **VE-answerQ** introduit au chapitre précédent.

7.1 Retour sur les requêtes multi-opérateurs considérées

Par la suite, nous considérons qu'un des deux axiomes $Ax^{SR'}$ ou $Ax^{SG'}$ est satisfait (notons à nouveau que le cas général est un sous-cas du cas semiring dès lors que l'on agrège toutes les fonctions d'utilité). Ainsi, nous supposons que :

— Au lieu d'avoir une structure de plausibilité, une structure d'utilité et une structure d'utilité espérée, nous considérons plus simplement un MCS $(E, \oplus, \otimes)$ totalement ordonné.

— Les conditions de normalisation sur les composantes d'environnement c d'un réseau PFU (V, G, P, F, U) deviennent $\oplus_c(\otimes_{P_i \in Fact(c)} P_i) = 1_E$.

— La définition opérationnelle de la valeur d'une requête $Q = (Sov, (V, G, P, F, U))$ devient :
  - $Ans(Q) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} P_i) \otimes (\otimes_{U_i \in U} U_i))$ dans le cas semi-anneau ($Ax^{SR'}$),
  - $Ans(Q) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} P_i) \otimes (\oplus_{U_i \in U} U_i))$ dans le cas semi-groupe ($Ax^{SG'}$).

7.2 D'une requête à des nœuds de calcul

Le processus de structuration raisonne à partir d'éléments appelés des *nœuds de calcul*. Ces nouveaux éléments de représentation sont introduits car les éléments considérés jusqu'alors nous empêche d'utiliser certains mécanismes : par exemple, l'utilisation des potentiels dans le cas semi-groupe nous empêche d'utiliser le mécanisme de duplication.

Définition 7.1. *Un* nœud de calcul *sur un ensemble E est :*

- *soit une fonction locale φ à valeurs dans E (nœud de calcul* atomique*) ;*
- *soit un triplet $(sov, \circledast, N)$ tel que $(E, \circledast)$ est un monoïde commutatif, N est un ensemble de nœuds de calcul et sov est une séquence de couples opérateur-variable(s) utilisant des opérateurs op tels que (E, op) soit un monoïde commutatif sur E.*

Par exemple, si P_1, P_2 sont deux fonctions locales de plausibilités et si U_1, U_2 sont deux fonctions locales d'utilité, alors P_1, P_2, U_1, U_2 sont des nœuds de calcul atomiques. Les triplets $n_1 = (\sum_x, \times, \{P_1\})$ et $n_2 = (\sum_{y,z,t}, \times, \{P_2, U_2\})$ sont aussi des nœuds de calcul, tout comme $n_3 = (\min_q \max_r, +, \{n_1, n_2, U_1\})$. Implicitement, un nœud de calcul représente un calcul à réaliser. Ceci est rendu explicite par l'intermédiaire de la définition de la valeur d'un nœud de calcul.

Définition 7.2. *Soit n un nœud de calcul. La* valeur de n*, notée $val(n)$, est définie par*

$$val(n) = \begin{cases} n \text{ si } n \text{ est atomique} \\ sov(\circledast_{n' \in N}\, val(n')) \text{ si } n = (sov, \circledast, N) \end{cases}$$

L'ensemble des variables éliminées par n, noté $V_e(n)$, est vide si n est atomique et est égal à l'ensemble des variables apparaissant dans sov si $n = (sov, \circledast, N)$.

La portée de *n, notée $sc(n)$, est définie par* $sc(n) = \begin{cases} sc(\varphi) \text{ si } n = \varphi \text{ est atomique} \\ (\cup_{n' \in N}\, sc(n')) - V_e(n) \text{ si } n = (sov, \circledast, N) \end{cases}$

L'ensemble des fils de *n, noté $Sons(n)$, est un ensemble de nœuds de calcul qui est vide si n est atomique et qui vaut N si $n = (sov, \circledast, N)$.*

Par exemple, la valeur de n_1 est $val(n_1) = \sum_x P_1$, la valeur de n_2 est $val(n_2) = \sum_{y,z,t}(P_2 \times U_2)$ et la valeur de n_3 est $val(n_3) = \min_q \max_r(val(n_1) + val(n_2) + U_1)$. Ainsi, un nœud $(sov, \circledast, N)$ définit une séquence d'éliminations sov sur une combinaison de nœuds de calcul via l'opérateur $\circledast$. Un nœud de calcul peut être représenté comme la racine d'un arbre de nœuds de calcul (cf. figure 7.1).

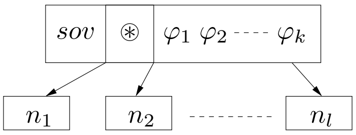


Figure 7.1: Un nœud de calcul $(sov, \circledast, N)$, avec $\{\varphi_1, \ldots, \varphi_k\}$ (respectivement $\{n_1, \ldots, n_l\}$) l'ensemble de nœuds de calcul atomiques (respectivement non atomiques) de N.

Les définitions précédentes sont étendues à des ensembles de nœuds de calcul N par $sc(N) = \cup_{n' \in N} sc(n')$, $V_e(N) = \cup_{n' \in N} V_e(n')$ et $Sons(N) = \cup_{n' \in N} Sons(n')$.

De plus, pour tout $op \in \{\min, \max, \oplus\}$, nous définissons l'ensemble de nœuds de N réalisant des éliminations uniquement avec un opérateur op par $N[op] = \{n \in N \,|\, n = (op_S, \circledast, N')\}$. L'ensemble $N - N[op]$ est quant à lui noté $N[\neg op]$. Par exemple, pour $N = \{n_1, n_2, n_3\}$ avec

$n_1 = (\sum_x, \times, \{P_1\})$, $n_2 = (\sum_{y,z,t}, \times, \{P_2, U_2\})$ et $n_3 = (\min_q \max_r, +, \{n_1, n_2, U_1\})$, nous avons $N[+] = \{n_1, n_2\}$ et $N[\neg +] = \{n_3\}$.

Enfin, étant donné un ensemble de nœuds de calcul N, nous définissons N^{+x} (respectivement N^{-x}) comme l'ensemble des nœuds de N dont la portée contient (resp. ne contient pas) la variable x : $N^{+x} = \{n \in N \mid x \in sc(n)\}$ (resp. $N^{-x} = \{n \in N \mid x \notin sc(n)\}$).

La valeur d'un nœud de calcul peut facilement être reliée à la réponse à une requête $Q = (Sov, (V, G, P, F, U))$:

— Dans le cas semi-anneau, $Ans(Q) = val(n_0)$ avec $n_0 = (Sov, \otimes, P \cup F \cup U)$.

— Dans le cas semi-groupe, $Ans(Q) = val(n_0)$ avec $n_0 = (Sov, \oplus, \{(\emptyset, \otimes, P \cup F \cup \{U_i\}), U_i \in U\})$. En effet, $val(n_0) = Sov(\oplus_{U_i \in U}(\otimes_{\varphi \in P \cup F \cup \{U_i\}} \varphi)) = Sov((\wedge_{F_i \in F} F_i) \star (\otimes_{P_i \in P} P_i) \otimes (\oplus_{U_i \in U} U_i))$.

Nous définissons également de manière explicite la notion d'ordre d'élimination compatible avec une séquence d'éliminations.

Définition 7.3. *Un ordre d'élimination o sur V est* compatible avec *une séquence d'élimination Sov portant sur V ssi $o \in lin(\preceq_{Sov})$. Si $op(x)$ correspond à l'opérateur d'élimination de x dans Sov, alors $Sov(o)$ représente la séquence d'éliminations suivante ($o(k)$ est la k-ème variable eliminée dans o) :*

$$Sov(o) = op(o(n))_{o(n)} \cdots op(o(2))_{o(2)} \cdot op(o(1))_{o(1)}$$

Exemple 7.4. *Soit $Sov = \min_{x_1,x_2} \sum_{x_3,x_4} \max_{x_5}$. L'ordre d'élimination o : $x_1 \prec x_2 \prec x_4 \prec x_3 \prec x_5$ est compatible avec Sov et $Sov(o) = \min_{x_1} \min_{x_2} \sum_{x_4} \sum_{x_3} \min_{x_5}$. L'ordre d'élimination o' : $x_4 \prec x_2 \prec x_1 \prec x_3 \prec x_5$ n'est pas compatible avec Sov car $x_4 \prec x_2$ alors que $x_2 \prec_{Sov} x_4$.*

Une structuration en deux temps

Structurer une requête est synonyme de réécrire le nœud de calcul initial n_0. Pour ce faire, nous utilisons un mécanisme de structuration en deux temps :

1. Nous cherchons d'abord ce que nous appelons la *macrostructure* d'une requête multi-opérateur. Cette première étape revient à révéler les libertés disponibles dans l'ordre d'élimination des variables et à déterminer les décompositions possibles (elle ne correspond pas à déterminer un ordre d'élimination optimal).

   Etant donné un ordre d'élimination o compatible avec la séquence d'élimination Sov d'une requête, cette macrostructure est obtenue par l'intermédiaire de règles de réécriture qui permettent de *simuler* les décompositions induites par l'élimination des variables de droite à gauche de $Sov(o)$. Une règle de réécriture $R : n_1 \rightsquigarrow n_2$ permet de transformer un nœud de calcul n_1 en un nœud de calcul $n_2 = R(n_1)$. Plus précisément, trois types de règles sont introduites :

   — des règles de *décomposition*, qui décomposent la structure en utilisant notamment le mécanisme de duplication ;

   — des règles de *recomposition*, qui permettent de réveler des libertés dans l'ordre d'élimination des variables ;

   — des règles de *simplification*, qui permettent de supprimer des calculs inutiles du fait des normalisations.

2. Une fois la macrostructure obtenue, la seconde étape de structuration, plus fine que la précédente, consiste à exploiter au mieux les libertés dans l'ordre d'élimination qui ont été révélées par la première étape. Cette seconde phase est réalisée en utilisant des techniques classiques de décomposition arborescente.

Nous supposons ici qu'il n'y a pas de faisabilité. Si des faisabilités interviennent, alors des mécanismes de structuration peuvent également être définis (voir la version anglaise de la thèse). Le processus de structuration diffère suivant si le cas considéré est le cas semi-groupe ou le cas semi-anneau.

7.3 Structuration des requêtes multi-opérateurs : le cas semi-anneau

Pour rendre les règles de réécriture plus lisibles, nous notons les nœuds de calcul $(sov, \otimes, N)$ simplement par (sov, N) car dans le cas semi-anneau, l'opérateur de combinaison utilisé est à chaque fois $\otimes$.

7.3.1 Macrostructuration d'une requête par règles de réécriture

Soit o un ordre d'élimination compatible avec la séquence d'élimination Sov d'une requête. Notre point de départ est le nœud de calcul non structuré obtenu directement à partir de la requête. Ce nœud s'écrit $n_0 = (Sov(o), \otimes, P \cup U)$, ou autrement dit $(Sov(o), P \cup U)$ dans le cas semi-anneau. Il peut être vu comme un arbre de nœuds de calcul (*Tree of Computation Nodes*, CNT) et est de ce fait noté $CNT_0(Q, o)$. Par exemple, à la figure 7.2, $CNT_0(Q, o)$ correspond au premier nœud. L'application de règles de réécriture va générer une séquence d'arbres de nœuds de calcul : pour tout $k \in \{0, \dots, |Sov| - 1\}$, la macrostructure à l'étape $k+1$, notée $CNT_{k+1}(Q, o)$, est obtenue à partir de la structure $CNT_k(Q, o)$ à l'étape k en considérant l'élimination la plus à droite encore non traitée dans $Sov(o)$ et en appliquant trois règles de réécriture différentes :

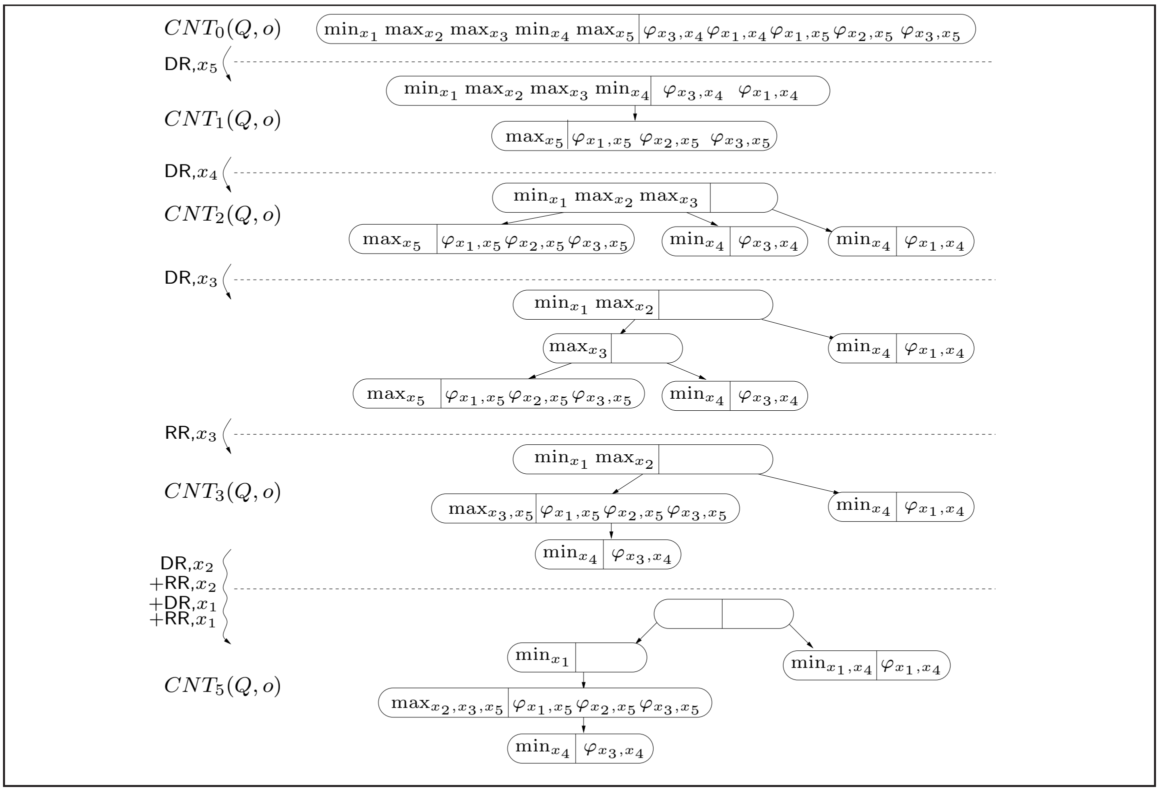
1. Une première règle dite de décomposition, notée DR (*Decomposition Rule*), utilise d'une part la distributivité de $\otimes$ par rapport aux opérateurs d'élimination (pour faire en sorte que l'élimination d'une variable x fasse intervenir seulement les fonctions locales ayant x dans leur portée) et d'autre part le procédé de duplication. La règle de réécriture DR met en œuvre ces deux types de décompositions. Elle s'écrit de la manière suivante :

$$\boxed{DR} \qquad \left(sov.\underset{x}{op}, N \right) \rightsquigarrow \begin{cases} (sov, N^{-x} \cup \{(op_x, \{n\}) \mid n \in N^{+x}\}) \text{ si } op = \otimes \\ (sov, N^{-x} \cup \{(op_x, N^{+x})\}) \text{ sinon} \end{cases}$$

Dans la figure 7.2, DR transforme la structure initiale $CNT_0(Q, o) = (\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4} \max_{x_5}, \{\varphi_{x_3,x_4}, \varphi_{x_1,x_4}, \varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})$ en la structure

$CNT_1(Q, o) = (\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4}, \{\varphi_{x_3,x_4}, \varphi_{x_1,x_4}, (\max_{x_5}, \{\varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})\})$ (cas $op \neq \otimes$ utilisant juste la distributivité de $\wedge$ par rapport à max).

L'élimination de x_4 avec l'opérateur min transforme ensuite $CNT_1(Q, o)$ en $CNT_2(Q, o) = (\min_{x_1} \max_{x_2} \max_{x_3}, \{(\min_{x_4}, \{\varphi_{x_3,x_4}\}), (\min_{x_4}, \{\varphi_{x_1,x_4}\}), (\max_{x_5}, \{\varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})\})$ (cas $op = \otimes = \min$, utilisant une duplication de x_4).



$CNT_0(Q,o)$
$\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4} \max_{x_5}$ | $\varphi_{x_3,x_4} \varphi_{x_1,x_4} \varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
DR,x_5
$\min_{x_1} \max_{x_2} \max_{x_3} \min_{x_4}$ | φ_{x_3,x_4} φ_{x_1,x_4}
$CNT_1(Q,o)$
$\max_{x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
DR,x_4
$\min_{x_1} \max_{x_2} \max_{x_3}$
$CNT_2(Q,o)$
$\max_{x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_3,x_4}
$\min_{x_4}$ | φ_{x_1,x_4}
DR,x_3
$\min_{x_1} \max_{x_2}$
$\max_{x_3}$
$\min_{x_4}$ | φ_{x_1,x_4}
$\max_{x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_3,x_4}
RR,x_3
$\min_{x_1} \max_{x_2}$
$CNT_3(Q,o)$
$\max_{x_3,x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_1,x_4}
$\min_{x_4}$ | φ_{x_3,x_4}
DR,x_2
+RR,x_2
+DR,x_1
+RR,x_1
$\min_{x_1}$
$\min_{x_1,x_4}$ | φ_{x_1,x_4}
$CNT_5(Q,o)$
$\max_{x_2,x_3,x_5}$ | $\varphi_{x_1,x_5} \varphi_{x_2,x_5} \varphi_{x_3,x_5}$
$\min_{x_4}$ | φ_{x_3,x_4}

Figure 7.2: Application des règles de réécriture sur CSP quantifié : $\min_{x_1} \max_{x_2,x_3} \min_{x_4} \max_{x_5} (\varphi_{x_3,x_4} \wedge \varphi_{x_1,x_4} \wedge \varphi_{x_1,x_5} \wedge \varphi_{x_2,x_5} \wedge \varphi_{x_3,x_5})$, avec l'ordre d'élimination $o : x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$.

2. Une seconde règle dite de recomposition, notée *RR* (*Recomposition Rule*), a pour but de révéler des libertés dans l'ordre d'élimination pour les nœuds créés par la règle *DR*.

$$\boxed{RR} \qquad \left(\underset{x}{op}, N\right) \rightsquigarrow \left(\underset{\{x\}\cup V_e(N[op])}{op}, N[\neg op] \cup Sons(N[op])\right)$$

   La formulation de *RR* signifie que si un nœud de calcul est chargé de faire une élimination op_x et a des fils qui doivent faire des éliminations du type op_S en utilisant le même opérateur *op*, alors il n'y a aucune raison d'imposer aux variables de S d'être éliminées avant x. La règle *RR* rend cela explicite en fusionnant les nœuds de calcul correspondant. Sur l'exemple de la figure 7.2, *RR* transforme le nœud $(\max_{x_3}, \{(\min_{x_4}, \{\varphi_{x_3,x_4}\}), (\max_{x_5}, \{\varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})\})$ créé par $DR(CNT_2(Q,o))$, en le nœud "recomposé"

$$(\max_{x_3,x_5}, \{(\min_{x_4}, \{\varphi_{x_3,x_4}\}), \varphi_{x_1,x_5}, \varphi_{x_2,x_5}, \varphi_{x_3,x_5}\})$$

   qui apparaît au sein de la structure $CNT_3(Q,o)$. Ainsi, la règle *RR* révèle que même si $x_3 \prec_{Sov} x_5$, rien n'empêche d'éliminer x_3 avant x_5.

3. Une troisième règle dite de simplification, notée *SR* (*Simplification Rule*), permet d'exploiter les conditions de normalisation $\oplus_c(\otimes_{P_i \in Fact(c)} P_i) = 1_E$:

$$\boxed{SR} \qquad [\text{Préconditions : } (c \in \mathcal{C}_E(G)) \wedge (c \cap (S \cup sc(N)) = \emptyset)]$$

$$(\underset{S\cup c}{\oplus}, N \cup Fact(c)) \rightsquigarrow (\underset{S}{\oplus}, N)$$

Par exemple, SR transforme un nœud $n = (\sum_{x,y,z}, \{P_{x\,|\,y,z}, P_y, P_z, c_y\})$ en le nœud simplifié $n' = (\sum_{y,z}, \{P_y, P_z, c_y\})$ grâce à la condition de normalisation $\sum_x P_{x\,|\,y,z} = 1$. Une autre application de SR génère un nœud de calcul encore plus simple, $n'' = (\sum_y, \{P_y, c_y\})$. SR ne peut alors plus être appliquée sur n''. Intrinsèquement, la raison pour laquelle des simplifications peuvent être disponibles est qu'il peut être ardu pour la personne spécifiant une requête d'identifier toutes les indépendances conditionnelles disponibles.

Il est important de noter que la règle SR peut elle-même révéler de nouvelles décompositions et de nouvelles libertés dans l'ordre d'élimination. C'est pourquoi les techniques utilisées peuvent restructurer un nœud qui a été simplifié (voir la version anglaise pour les détails techniques de la chose).

Partant d'une requête $Q = (Sov, \mathcal{N})$ et d'un ordre d'élimination o compatible avec Sov, l'arbre de nœuds de calcul obtenu après utilisation des règles de réécriture DR, RR et DR en traitant les éliminations de droite à gauche de $Sov(o)$ est noté $CNT(Q, o)$.

Quelques propriétés satisfaites par la macrostructure obtenue

Le théorème 7.5 montre que l'arbre de nœuds de calcul $CNT(Q, o)$ obtenu à partir d'une requête $Q = (Sov, \mathcal{N})$ et d'un ordre d'élimination o est en réalité indépendant de l'ordre d'élimination $o \in lin(\preceq_{Sov})$ choisi au départ.

Théorème 7.5. *Soit $Q = (Sov, \mathcal{N})$ une requête. Alors, pour tous $o, o' \in lin(\preceq_{Sov})$, $CNT(Q, o) = CNT(Q, o')$.*

Ce résultat nous permet de noter $CNT(Q, o)$ simplement sous la forme $CNT(Q)$.

Comme indiqué par le théorème 7.6, la structure finale obtenue est correcte, c'est-à-dire que sa valeur est bien égale à la réponse $Ans(Q)$ à la requête Q considérée.

Théorème 7.6. *Soit $Q = (Sov, \mathcal{N})$ une requête. Alors, $val(CNT(Q)) = Ans(Q)$.*

Enfin, il est enfin possible de montrer que le processus de structuration a une complexité polynomiale en temps et en espace.

7.3.2 Une seconde étape de structuration utilisant des décompositions arborescentes

La macrostructure obtenue est un arbre de nœuds de calcul mono-opérateurs de la forme $(\min_S, \otimes, N)$, $(\max_S, \otimes, N)$ ou $(\oplus_S, \otimes, N)$. Nous pouvons maintenant rechercher des structures plus fines en utilisant un processus de structuration interne pour chaque nœud de calcul obtenu. L'objectif est ici d'utiliser au mieux les libertés dans l'ordre d'élimination révélées par le processus de macrostructuration.

Pour ce faire, des techniques de décomposition arborescente sont utilisées (voir la version anglaise pour une présentation complète de la notion de décomposition arborescente). Ces techniques sont des outils génériques utilisés pour traiter notamment des CSPs ou des réseaux bayésiens. Elles exploitent les propriétés topologiques des modèles graphiques considérés de manière à structurer un problème donné en un arbre de problèmes élémentaires à résoudre [89, 2, 88, 54, 12, 57]. La complexité du problème global est alors fonction de la complexité du problème élémentaire le plus dur à résoudre, d'où l'intérêt de chercher de bonnes décompositions du problème initial. Les techniques de décomposition arborescente sont basiquement utilisées pour des problèmes ne faisant intervenir qu'un seul opérateur d'élimination et qu'un seul opérateur de combinaison, ce qui est le cas de tous les nœuds de calcul mono-opérateurs obtenus après la phase de macrostructuration.

Une fois les techniques de décomposition arborescente utilisées, la structure obtenue contient d'une part une macrostructure donnée par les nœuds de calcul et d'autre part une structure plus fine à l'intérieur de chaque nœud. Nous obtenons ainsi une architecture de calcul générale appelée *arbre de clusters multi-opérateur*.

Définition 7.7. *Un arbre de clusters multi-opérateur (*Multi-operator Cluster Tree*, MCTree) est un arbre enraciné* (C, E) [1] *dont chaque sommet* $c \in C$*, appelé un cluster, est étiqueté par trois éléments :*

— *un ensemble de variables* $V(c)$*,*

— *un ensemble de fonctions locales* $\Phi(c)$ *à valeurs dans un ensemble* E*,*

— *et un couple* $(\oplus^c, \otimes^c)$ *d'opérateurs sur* E *tels que* $(E, \oplus^c, \otimes^c)$ *est un semi-anneau commutatif.*

La largeur d'un MCTree est définie par $w = \max_{c \in C} |V(c)| - 1$*.*

Nous spécifions explicitement un opérateur d'élimination et un opérateur de combinaison à utiliser dans chaque cluster de manière à gérer proprement la nature multi-opérateur des requêtes considérées. La figure 7.3 montre un exemple de MCTree qui peut être obtenu à partir d'un problème de satisfiabilité stochastique étendue [62].

Définition 7.8. *La valeur d'un cluster* c *d'un MCTree est définie par*

$$val(c) = \underset{V(c)-V(pa(c))}{\oplus^c} \left(\left(\underset{\varphi \in \Phi(c)}{\otimes^c} \varphi \right) \otimes^c \left(\underset{s \in Sons(c)}{\otimes^c} val(s) \right) \right)$$

La valeur $val(M)$ *d'un MCTree* M *est égale à la valeur de son cluster racine.*

Théorème 7.9. *Soit* Q *une requête. Soit* M *un MCTree que l'on peut obtenir à partir de* $CNT(Q)$*. Alors,* $val(M) = Ans(Q)$*. De plus, chaque règle de décision optimale dans* $val(M)$ *pour une variable non dupliquée est aussi une règle de décision optimale dans* $Ans(Q)$ *et pour toute variable de décision dupliquée, il existe au moins une règle de décision optimale dans* $val(M)$ *qui est aussi optimale dans* $Ans(Q)$*.*

En fait, les règles de décision optimales peuvent être mémorisées sur les séparateurs du MCTree (le séparateur entre deux clusters c et $s \in Sons(c)$ est l'ensemble de variables $V(c) \cap V(s)$).

7.3.3 Comparaison avec une approche non structurée

Une analyse fine de la structure d'une rêquete peut engendrer des gains exponentiels, comme l'ont montré les exemples introduits à la section 6.6. Un résultat plus fort peut être établi. Ce résultat est qu'en termes de largeur induite (ou largeur d'arbre), l'approche structurée est *toujours au moins aussi bonne* qu'une approche non structurée du type de celle utilisée dans l'algorithme **VE-answerQ**.

[1] C est l'ensemble des sommets de l'arbre, appelés des clusters, et E est l'ensemble des arêtes de l'arbre.

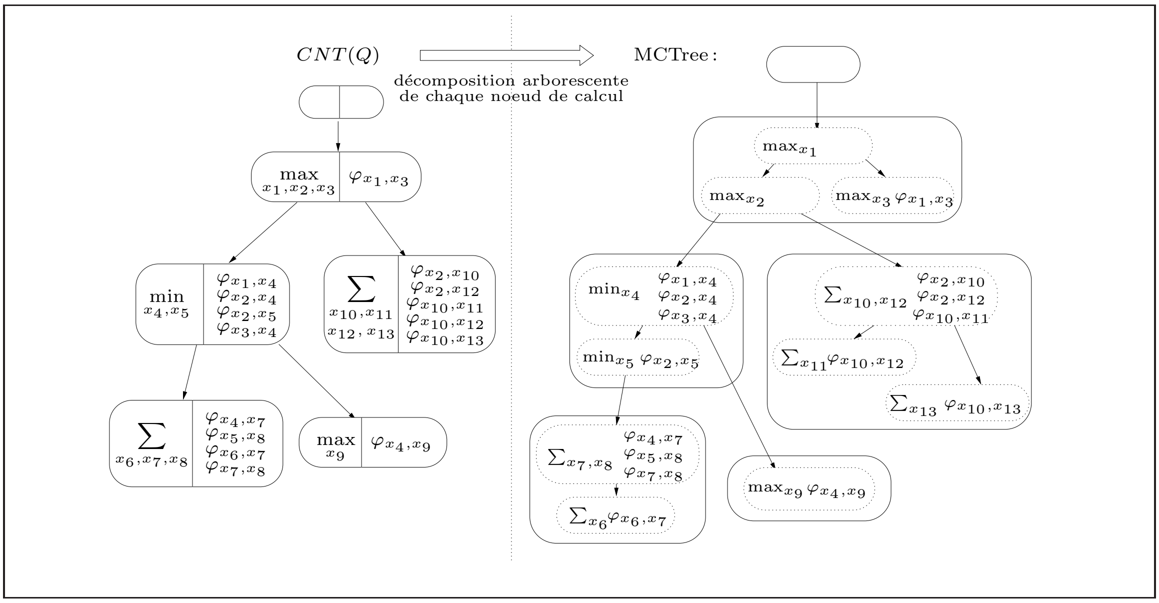

$CNT(Q)$
décomposition arborescente
de chaque noeud de calcul
MCTree :
$\max_{x_1,x_2,x_3}$ φ_{x_1,x_3}
$\min_{x_4,x_5}$ φ_{x_1,x_4} φ_{x_2,x_4} φ_{x_2,x_5} φ_{x_3,x_4}
$\sum_{x_{10},x_{11},x_{12},x_{13}}$ $\varphi_{x_2,x_{10}}$ $\varphi_{x_2,x_{12}}$ $\varphi_{x_{10},x_{11}}$ $\varphi_{x_{10},x_{12}}$ $\varphi_{x_{10},x_{13}}$
$\sum_{x_6,x_7,x_8}$ φ_{x_4,x_7} φ_{x_5,x_8} φ_{x_6,x_7} φ_{x_7,x_8}
$\max_{x_9}$ φ_{x_4,x_9}
$\max_{x_1}$
$\max_{x_2}$
$\max_{x_3}\varphi_{x_1,x_3}$
$\min_{x_4}$ φ_{x_1,x_4} φ_{x_2,x_4} φ_{x_3,x_4}
$\min_{x_5}\varphi_{x_2,x_5}$
$\sum_{x_{10},x_{12}}$ $\varphi_{x_2,x_{10}}$ $\varphi_{x_2,x_{12}}$ $\varphi_{x_{10},x_{11}}$
$\sum_{x_{11}}\varphi_{x_{10},x_{12}}$
$\sum_{x_{13}}\varphi_{x_{10},x_{13}}$
$\sum_{x_7,x_8}$ φ_{x_4,x_7} φ_{x_5,x_8} φ_{x_7,x_8}
$\sum_{x_6}\varphi_{x_6,x_7}$
$\max_{x_9}\varphi_{x_4,x_9}$

Figure 7.3: Exemple d'arbre de clusters multi-opérateur obtenue à partir de $CNT(Q)$. Notons qu'un cluster c est représenté par (1) l'ensemble de variables $V(c) - V(pa(c))$ qu'il élimine, son opérateur d'élimination $\oplus^c$ et l'ensemble $\Phi(c)$ des fonctions qui lui sont associées ; dans le cas semi-anneau, nous avons toujours $\otimes^c = \otimes$; (2) l'ensemble de ses fils.

Définition 7.10. *La largeur d'un arbre de nœuds de calcul CNT, notée w_{CNT}, est égale à la largeur minimale d'un MCTree qui peut être obtenu à partir d'une décomposition arborescente de CNT.*

Théorème 7.11. *Soit $Q = (Sov, \mathcal{N})$ une requête sur un réseau PFU $\mathcal{N} = (V, G, P, \emptyset, U)$. Soit $\mathcal{G} = (V, \{sc(\varphi), \varphi \in P \cup U\})$ l'hypergraphe associé à $\mathcal{N}$. Alors, $w_{CNT(Q)} \leq w_{\mathcal{G}}(\preceq_{Sov})$.*

Pour le QCSP pris en exemple à la figure 7.2, $w_{CNT(Q)} = 1$ alors que la largeur induite contrainte vaut $w_{\mathcal{G}}(\preceq_{Sov}) = 3$: ainsi, la complexité théorique passe de $O(|\Phi| \cdot d^4)$ à $O(|\Phi| \cdot d^2)$.

Des différences plus importantes peuvent être observées entre la largeur avant structuration ($w_{\mathcal{G}}(\preceq_{Sov})$) et la largeur après structuration ($w_{CNT(Q)}$) sur des problèmes de plus grande taille. Des expérimentations ont par exemple été réalisées sur des instances de la librairie QBF, pour lesquels des gains d'un facteur pouvant aller jusqu'à 10 *en termes de largeur* ont pu être observés (rappelons que la complexité théorique est exponentielle en la largeur induite).

Comparaison avec les approches existantes Les règles de réécriture que nous avons définies peuvent être reliées avec l'approche des arbres de quantificateurs (*quantifier tree* [5]) récemment introduite pour résoudre des formules booléennes quantifiées. Le principe de cette approche est d'analyser les structures cachées de QBF exprimées en forme prénexe normale conjonctive par l'intermédiaire de mécanismes de structuration. Cette analyse génère des gains importants en termes de temps de résolution d'une QBF. Les techniques de structuration utilisées pour construire les arbres de quantificateurs correspondent exactement à l'application de la règle de décomposition *DR* instanciée pour la structure algébrique utilisée par les QBF, i.e. pour $\oplus = \vee$ et $\otimes = \wedge$.

Les MCTrees fournissent une explication théorique aux résultats expérimentaux constatés avec les arbres de quantificateur, via la largeur d'arbre. De plus, étant définie dans un cadre algébrique général, notre approche étend et généralise entièrement l'approche des arbres de quantificateurs.

Elle est en effet applicable à d'autres formalismes que les QBF, incluant les QCSP, SSAT ou les CSP stochastiques. En outre, les arbres de quantificateurs n'utilisent ni de règle de recomposition RR, ni de décomposition en arbre de clusters minimisant la largeur d'arbre, ni de règle de simplification (ce dernier point étant relativement normal puisqu'une formule booléenne quantifiée ne fait pas intervenir de normalisations).

7.4 Structuration des requêtes multi-operateurs : le cas semi-groupe

7.4.1 Processus de structuration

La structuration des requêtes dans le cas semi-anneau nous conduit à une architecture de calcul générale, appelée l'architecture MCTree, qui fait intervenir plusieurs opérateurs d'élimination et un seul opérateur de combinaison. Les choses sont techniquement plus complexes dans le cas semi-groupe, qui fait intervenir plusieurs opérateurs de combinaison ($\otimes$ et $\oplus$). A nouveau, nous utilisons un mécanisme de structuration en deux temps avec une phase de macrostructuration et une phase de décomposition en arbres de clusters.

Cette fois, la phase de macrostructuration crée une séquence de DAGs de nœuds de calcul au lieu d'une séquence d'arbres de nœud de calcul. La raison intuitive est que dans le cas semi-groupe, les mêmes plausibilités s'appliquent sur toutes les fonctions locales d'utilité, puisque par exemple on peut écrire $\sum_S(P \times (U_1 + U_2 + U_3)) = (\sum_S(P \times U_1)) + (\sum_S(P \times U_2)) + (\sum_S(P \times U_3))$. Ceci explique que certains calculs fait sur les plausibilités soient partagés par plusieurs nœuds de calcul, d'où la structure de DAG.

Une fois toutes les éliminations considérées, nous obtenons un DAG de nœuds de calcul noté $CNDAG(Q)$, sur lequel nous pouvons alors utiliser des techniques de décomposition arborescente. Ceci nous mène finalement à une architecture de calcul général appelée un DAG de clusters multi-opérateur (*Multi-operator Cluster DAG*, MCDAG).

Définition 7.12. *Un* DAG de clusters multi-opérateur *est un DAG dans lequel chaque sommet c, appelé un cluster, est étiqueté par trois éléments :*

- *un ensemble de variables $V(c)$,*
- *un ensemble $\Phi(c)$ de fonctions locales à valeurs dans E,*
- *et un couple $(\oplus^c, \otimes^c)$ d'opérateurs sur E tels que $(E, \oplus^c, \otimes^c)$ est un semi-anneau commutatif.*

La largeur d'un MCDAG est définie par $w = \max_{c \in C} |V(c)| - 1$. La hauteur d'un MCDAG est égale au nombre maximal de variables qui apparaissent sur un chemin allant de la racine vers une feuille du MCDAG.

Définition 7.13. *La valeur d'un cluster c d'un MCDAG est définie par*

$$val(c) = \bigoplus^c_{V(c)-V(pa(c))} \left(\left(\bigotimes^c_{\varphi \in \Phi(c)} \varphi \right) \otimes^c \left(\bigotimes^c_{s \in Sons(c)} val(s) \right) \right)$$

La valeur d'un MCDAG est la valeur de son nœud racine.

La figure 7.4 donne un exemple de MCDAG qui peut être obtenu à partir d'un DAG de nœuds de calcul $CNDAG(Q)$.

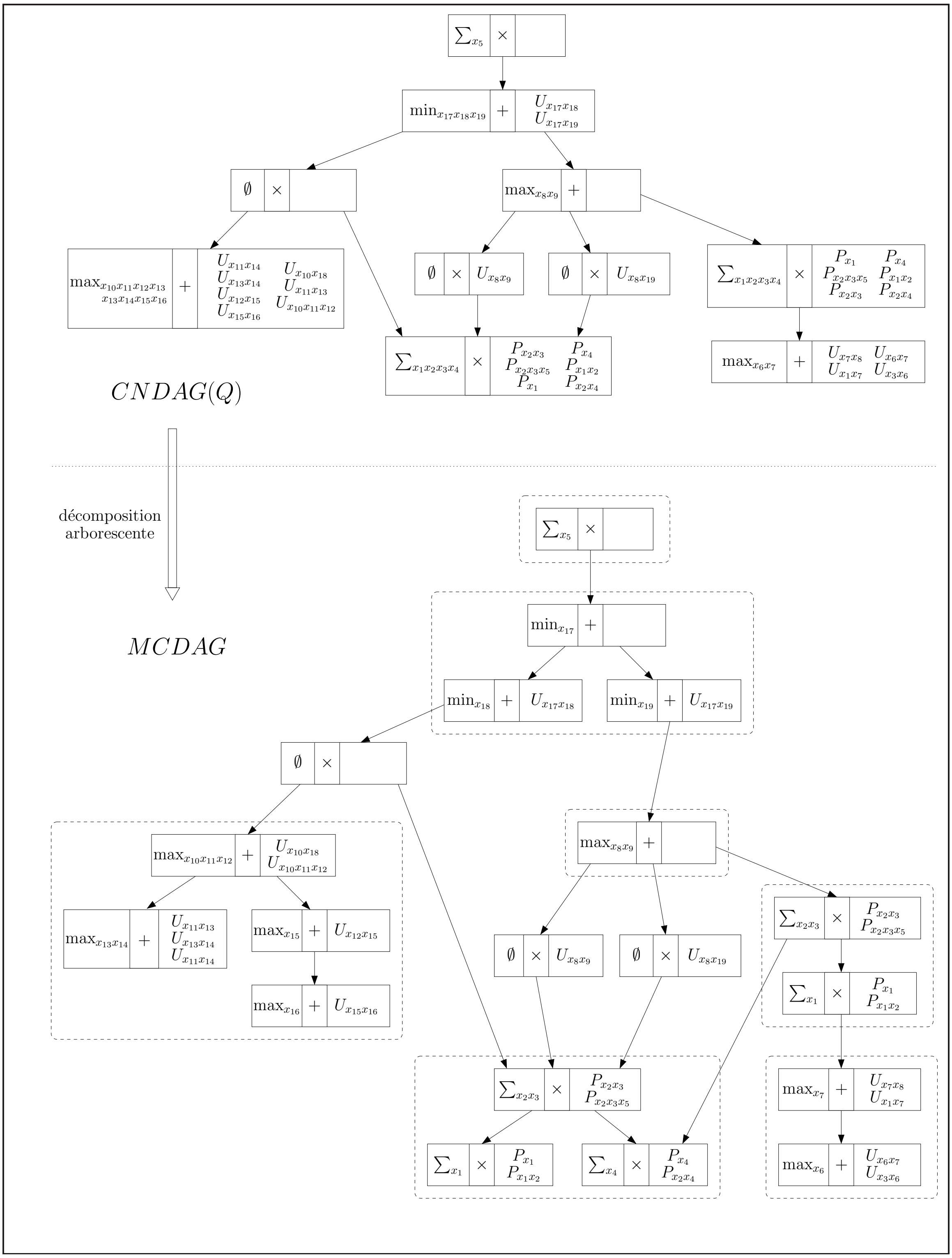

$\sum_{x_5}$ ×
$\min_{x_{17}x_{18}x_{19}}$ + $U_{x_{17}x_{18}}$ $U_{x_{17}x_{19}}$
∅ ×
$\max_{x_8x_9}$ +
$\max_{x_{10}x_{11}x_{12}x_{13}x_{14}x_{15}x_{16}}$ + $U_{x_{11}x_{14}}$ $U_{x_{13}x_{14}}$ $U_{x_{12}x_{15}}$ $U_{x_{15}x_{16}}$ $U_{x_{10}x_{18}}$ $U_{x_{11}x_{13}}$ $U_{x_{10}x_{11}x_{12}}$
∅ × $U_{x_8x_9}$
∅ × $U_{x_8x_{19}}$
$\sum_{x_1x_2x_3x_4}$ × P_{x_1} $P_{x_2x_3x_5}$ $P_{x_2x_3}$ P_{x_4} $P_{x_1x_2}$ $P_{x_2x_4}$
$\sum_{x_1x_2x_3x_4}$ × $P_{x_2x_3}$ $P_{x_2x_3x_5}$ P_{x_1} P_{x_4} $P_{x_1x_2}$ $P_{x_2x_4}$
$\max_{x_6x_7}$ + $U_{x_7x_8}$ $U_{x_1x_7}$ $U_{x_6x_7}$ $U_{x_3x_6}$
$CNDAG(Q)$
décomposition arborescente
$MCDAG$
$\sum_{x_5}$ ×
$\min_{x_{17}}$ +
$\min_{x_{18}}$ + $U_{x_{17}x_{18}}$
$\min_{x_{19}}$ + $U_{x_{17}x_{19}}$
∅ ×
$\max_{x_8x_9}$ +
$\max_{x_{10}x_{11}x_{12}}$ + $U_{x_{10}x_{18}}$ $U_{x_{10}x_{11}x_{12}}$
$\max_{x_{13}x_{14}}$ + $U_{x_{11}x_{13}}$ $U_{x_{13}x_{14}}$ $U_{x_{11}x_{14}}$
$\max_{x_{15}}$ + $U_{x_{12}x_{15}}$
$\max_{x_{16}}$ + $U_{x_{15}x_{16}}$
$\sum_{x_2x_3}$ × $P_{x_2x_3}$ $P_{x_2x_3x_5}$
∅ × $U_{x_8x_9}$
∅ × $U_{x_8x_{19}}$
$\sum_{x_1}$ × P_{x_1} $P_{x_1x_2}$
$\sum_{x_2x_3}$ × $P_{x_2x_3}$ $P_{x_2x_3x_5}$
$\max_{x_7}$ + $U_{x_7x_8}$ $U_{x_1x_7}$
$\sum_{x_1}$ × P_{x_1} $P_{x_1x_2}$
$\sum_{x_4}$ × P_{x_4} $P_{x_2x_4}$
$\max_{x_6}$ + $U_{x_6x_7}$ $U_{x_3x_6}$

Figure 7.4: Exemple de MCDAG obtenu à partir de $CNDAG(Q)$ en utilisant des décompositions arborescentes et en fusionnant des clusters faisant exactement le même calcul.

7.4.2 Comparaison avec une approche non structurée

Définition 7.14. *Soit Q une requête. La largeur de $CNDAG(Q)$ est la largeur minimale d'un MCDAG qui peut être obtenu à partir de $CNDAG(Q)$ en utilisant des décompositions en arbre de clusters.*

Le théorème 7.15 ci-dessous montre que structurer des requêtes multi-opérateurs ne peut être que bénéfique en termes de largeur, et ainsi la largeur d'arbre "subie" par un algorithme d'élimination de variables sur un MCDAG n'est jamais plus grande que la largeur d'arbre subie par l'algorithme **VE-answerQ** qui utilise une approche non structurée.

Théorème 7.15. *Soit $Q = (Sov, \mathcal{N})$ une requête sur un réseau PFU $\mathcal{N} = (V, G, P, \emptyset, U)$. Soit $\mathcal{G} = (V, \{sc(\varphi), \varphi \in P \cup U\})$ l'hypergraphe associé à $\mathcal{N}$. Alors, $w_{CNDAG(Q)} \leq w_{\mathcal{G}}(\preceq_{Sov})$.*

Comparaison avec des approches existantes Comparés aux architectures de calcul existantes pour résoudre des diagrammes d'influence, les MCDAGs peuvent donner des résultats potentiellement exponentiellement meilleurs étant donné qu'ils peuvent faire décroître linéairement la largeur d'arbre en utilisant :

- le mécanisme de duplication, qui a en fait déjà été utilisé dans la littérature, où il était appliqué à la volée durant le processus d'élimination [26] ; dans notre cas, la duplication est utilisée en preprocessing et en synergie avec d'autres règles de réécriture, ce qui peut accroître son impact. L'idée principale derrière la duplication est qu'un diagramme d'influence exprime deux types d'indépendances : l'une pour la distribution globale de probabilité qu'il définit, et l'autre pour la fonction globale d'utilité. Dans l'architecture MCDAG, ces deux aspects sont clairement utilisés alors que toutes les architectures à base de potentiels utilisent une forme plus faible d'indépendance, qui mixe les deux formes énoncées ci-dessus ;
- le relâchement des contraintes sur l'ordre d'élimination ; ce mécanisme peut être relié avec la notion d'information pertinente (*relevant information*) utilisée classiquement pour réduire la complexité spatiale de stockage des règles de décision. Dans notre cas, l'affaiblissement des contraintes sur l'ordre d'élimination a aussi un intérêt en termes de complexité temporelle ;
- l'utilisation des conditions de normalisation : classiquement, ces conditions sont utilisées à la volée par un mécanisme connu sous le nom de Lazy Propagation [64]. Les conditions de normalisation sont plus exploitées dans notre cas car elles peuvent modifier l'architecture de calcul elle-même en révélant des libertés cachées dans l'ordre d'élimination.

Enfin, l'architecture MCDAG n'utilise aucune opération de division.

Les MCDAGs peuvent également être utilisés pour des diagrammes d'influence possibilistes pessimistes ou des problèmes de planification classique dans lesquels il s'agit de trouver un plan permettant d'arriver à au moins un but parmi un ensemble de buts.

7.5 Conclusion : l'architecture MCDAG, une architecture de calcul générique

Ce chapitre a présenté comment structurer des requêtes multi-opérateurs de manière systématique. Le processus de structuration met en jeu deux étapes majeures :

- Une étape de macrostructuration, à base de règles de réécriture.
- Une étape de décomposition en arbres de clusters, exploitant les libertés révélées par la première étape.

Ceci nous conduit à l'architecture MCTree dans le cas semi-anneau (cf. définition 7.7 page 73), et à l'architecture MCDAG dans le cas semi-groupe (cf. définition 7.12 page 75). Ces deux architectures, qui satisfont des propriétés de correction et d'unicité, permettent d'obtenir une meilleure largeur induite (ou largeur d'arbre). Comparée à d'autres architectures de calculs faisant de l'élimination de variables, l'architecture MCDAG est la seule à utiliser à la fois plusieurs opérateurs d'élimination et plusieurs opérateurs de combinaison.

L'architecture MCTree étant un cas particulier de l'architecture MCDAG, nous pouvons en fait raisonner uniquement sur l'architecture MCDAG et ne plus dissocier les cas semi-anneau et semi-groupe par la suite, comme illustré à la figure 7.5.

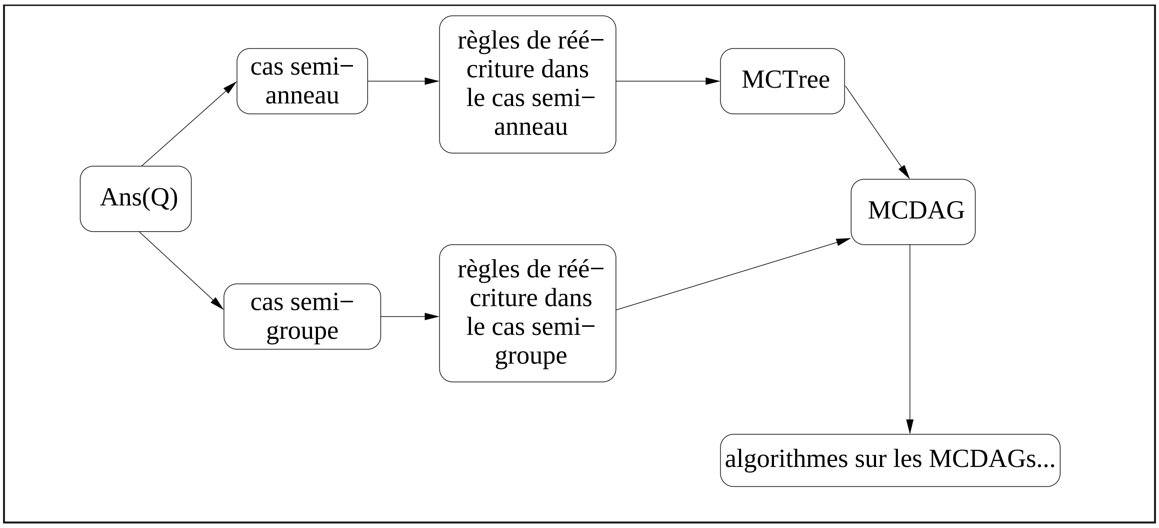

cas semi−anneau
règles de réé−criture dans le cas semi−anneau
MCTree
Ans(Q)
MCDAG
cas semi−groupe
règles de réé−criture dans le cas semi−groupe
algorithmes sur les MCDAGs...

Figure 7.5: Vers une architecture de calcul unique.

Une autre manière de formuler cette conclusion consiste à dire que calculer la réponse $Ans(Q)$ à une requête Q ainsi que des règles de décision optimales correspond à résoudre le problème suivant :

Soit $(E, \oplus, \otimes)$ un MCS totalement ordonné.
Soit m un MCDAG mettant en jeu des fonctions locales à valeurs dans E et des clusters utilisant $(\oplus^c, \otimes^c) \in \{(\min, \oplus), (\max, \oplus), (\min, \otimes), (\max, \otimes), (\oplus, \otimes)\}$
Calculer la valeur de m et des règles de décision optimales.

Une fois l'architecture MCDAG obtenue, un nouvel algorithme générique d'élimination de variables consiste tout simplement à dire que dès qu'un cluster c a reçu la valeur $val(s)$ de chacun de ses fils $s \in Sons(c)$, il peut calculer sa propre valeur

$$val(c) = {\oplus^c}_{V(pa(c))-V(c)} \left(\left({\otimes^c}_{\varphi\in\Phi(c)}\, \varphi\right) \otimes^c \left({\otimes^c}_{s\in Sons(c)}\, val(s)\right)\right)$$

puis la transmettre à chacun de ses parents. La valeur du cluster racine est alors égale à la réponse à la requête.

Pour chaque cluster c, $val(c)$ peut être calculé en éliminant une à une les variables de $V(pa(c))-V(c)$, comme fait précédemment, ou en considérant toutes les variables de $V(pa(c)) - V(c)$ simultanément. La seconde approche, connue sous le nom d'élimination de clusters (*Cluster-Tree Elimination*, CTE [6]), généralise l'algorithme d'élimination de variables et fournit d'une part la même complexité théorique et d'autre part une meilleure complexité spatiale, exponentielle en la taille du plus grand séparateur entre deux clusters du MCDAG. De telles méthodes sont également connues dans la littérature sous le nom de programmation dynamique (non sérielle), algorithme d'arbre de jonction ou de relaxation parfaite [69].

Même si tous ces algorithmes peuvent répondre à des requêtes, ils n'utilisent ni des mécanismes de retour arrière (*backtrack*), ni des mécanismes d'élagage de l'espace de recherche. Partant de ce constat, la prochaine étape consiste à utiliser de tels outils au sein de l'architecture MCDAG.

Chapitre 8

Algorithme de recherche arborescente structurée sur l'architecture MCDAG

Répondre à une requête PFU est équivalent à calculer la valeur d'un MCDAG. Cette seconde tâche peut être réalisée de manière relativement naturelle en utilisant un algorithme d'élimination de variables (VE) ou d'élimination de clusters (CTE). Ces algorithmes calculent la valeur d'un MCDAG en propageant les informations des feuilles vers la racine. L'algorithme VE fournit une complexité temporelle exponentielle en la largeur du MCDAG, au prix d'une complexité spatiale aussi exponentielle en la largeur du MCDAG. L'algorithme CTE donne la même complexité temporelle, tout en assurant une meilleure complexité spatiale, exponentielle seulement en la taille maximale d'un séparateur entre deux clusters du MCDAG.

Parallèlement, une méthode de recherche telle que la recherche arborescente en profondeur d'abord offre une complexité spatiale linéaire. De plus, malgré une complexité spatiale théorique plus élevée, une recherche arborescente est souvent meilleure qu'un algorithme d'élimination de variables en pratique, notamment lorsque des techniques d'élagage de l'espace de recherche par des bornes sont utilisées.

Afin de bénéficier à la fois de l'efficacité pratique de la recherche arborescente et de la bonne complexité temporelle théorique des algorithmes d'élimination de variables, nous introduisons un algorithme générique de recherche arborescente structurée qui tire partie des décompositions structurelles exprimées par l'architecture MCDAG. Dans son principe, une telle idée n'est pas nouvelle : en particulier, plusieurs méthodes de recherche arborescente structurée ont été récémment définies [21, 49, 31].

Cependant, ces méthodes existantes permettent basiquement de calculer une séquence mono-opérateur d'éliminations de variables sur une combinaison mono-opérateur de fonctions locales. Cette nature mono-opérateur facilite grandement l'utilisation de bornes. De plus, les schémas existants fonctionnent soit avec des opérateurs bien spécifiques, soit avec une structure algébrique faisant des hypothèses plus fortes que celles faites en considérant un MCS (*Monotonic Commutative Semiring*) totalement ordonné.

Ainsi, nous avons besoin de définir des algorithmes de recherche arborescente structurée capable de gérer la nature multi-opérateur des requêtes considérées (ou, de manière équivalente, des MCDAGs considérés). Comme précédemment indiqué cela soulève de nouvelles questions concernant l'utilisation de bornes dans le contexte d'une alternance d'éliminations utilisant les opérateurs min, max et $\oplus$.

8.1 Algorithmes de recherche arborescente structurée existants

Différents algorithmes de recherche arborescente structurée ont déjà été définis dans la littérature. Parmi eux, on peut entre autres trouver les algorithmes de recherche AND/OR [31], la méthode *recursive conditioning* [21] et la méthode BTD (Backtrack bounded by Tree Decomposition [49]). La méthode de recherche que nous proposons s'inspire de l'algorithme BTD. Voir la version anlgaise de la thèse pour une présentation de ces schémas existants.

Recherche arborescente structurée sur un MCDAG Nous présentons de manière incrémentale un algorithme qui utilise la structure des MCDAGs et qui généralise l'algorithme BTD utilisé sur les CSP et les CSP valués. Nous partons d'une version de base qui correspond à une recherche arborescente structurée sans mémorisation et sans utilisation de bornes, pour arriver à une recherche arborescente structurée utilisant à la fois des bornes pour élaguer l'espace de recherche et des techniques de mémorisation permettant d'éviter des calculs redondants.

Dans ce qui suit, nous supposons qu'il n'y a ni variables libres dans la requête, ni faisabilités. Il est possible d'étendre les mécanismes proposés au cas avec variables libres et avec faisabilités.

8.2 Un premier algorithme de recherche arborescente structurée

Le premier algorithme que nous définissons correspond à un parcours du MCDAG de la racine vers les feuilles (alors qu'un algorithme d'élimination de variables ou de clusters propage de l'information des feuilles du MCDAG vers la racine). Nous introduisons tout d'abord une définition essentielle pour la compréhension du reste de ce chapitre.

Définition 8.1. *Soit c un cluster du MCDAG. Soit $V \subset V(c) - V(pa(c))$ un sous-ensemble des variables à éliminer dans c. Soit $\Phi \subset \Phi(c)$ un sous-ensemble des fonctions locales associées à c. Soit A une affectation des variables de $V(c) - V$ et des variables impliquées dans les ascendants de c dans le MCDAG. Nous définissons $val(c, A, V, \Phi)$ par*

$$val(c, A, V, \Phi) = \bigoplus^{c}_{V} \left(\left(\bigotimes^{c}_{\varphi \in \Phi} \varphi(A) \right) \otimes^{c} \left(\bigotimes^{c}_{s \in Sons(c)} val(s)(A) \right) \right)$$

avec $val(s)(A)$ la valeur donnée par la définition 7.8 page 73.

En d'autres termes, $val(c, A, V, \Phi)$ correspond à l'élimination des variables de V sur la combinaison des fonctions locales dans Φ et des valeurs des clusters fils de c. Les valeurs sont prises pour une affectation A et les opérateurs d'élimination et de combinaison utilisés sont les opérateurs $\oplus^c$ et $\otimes^c$ du cluster c.

Proposition 8.2. *Soit M un MCDAG associé à une requête Q. Soit c un cluster de M.*

(a) Soit r le cluster racine de M. Alors, $Ans(Q) = val(r, \emptyset, V(r), \Phi(r))$.

(b) $\forall x \in V,\ val(c, A, V, \Phi) = \oplus^c{}_{a \in dom(x)} \left((\otimes^c{}_{\varphi \in \Phi_0} \varphi(A.(x,a))) \otimes^c val(c, A.(x,a), V - \{x\}, \Phi - \Phi_0)\right)$, avec $\Phi_0 = \{\varphi \in \Phi \mid sc(\varphi) \cap (V - \{x\}) = \emptyset\}$

(c) $val(c, A, \emptyset, \Phi) = (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s \in Sons(c)}\, val(s, A, V(s) - V(c), \Phi(s))\right)$

La proposition 8.2 permet de définir directement un algorithme de recherche arborescente structurée. Plus précisément, la proposition 8.2(a) montre que pour calculer la réponse $Ans(Q)$ à une requête Q associée à un MCDAG ayant pour racine r, il suffit de calculer $val(r, \emptyset, V(r), \Phi(r))$. Une utilisation récursive de la proposition 8.2(b) donne alors une méthode pour calculer cette quantité $val(r, \emptyset, V(r), \Phi(r))$, en affectant récursivement les variables de $V(r)$. Dès que toutes les variables de $V(r)$ sont affectées, des quantités telles que $val(r, A, \emptyset, \Phi)$ doivent être calculées. La proposition 8.2(c), nous permet alors d'écrire

$$val(r, A, \emptyset, \Phi) = (\otimes^c{}_{\varphi \in \Phi}\, \varphi(A)) \otimes^c \left(\otimes^c{}_{s \in Sons(r)}\, val(s, A, V(s) - V(r), \Phi(s))\right).$$

Chaque $val(s, A, V(s) - V(r), \Phi(s))$ peut ensuite être calculé en utilisant à nouveau la proposition 8.2(b). Ainsi, une application récursive et alternée des propositions 8.2(b) et 8.2(c) permet de calculer $Ans(Q)$.

L'algorithme associé au mécanisme précédemment décrit, nommé **TS-mcdag** (comme "Tree Search on MCDAG"), est donné à la figure 8.1. Comme il utilise la structure du problème, cet algorithme sera probablement plus performant que l'algorithme de recherche arborescente non structurée donné à la section 6.1.

```
TS-mcdag($c, A, V, \Phi$)
début
    si ($V = \emptyset$) alors
        $\mathcal{S} \leftarrow Sons(c)$
        $val \leftarrow \otimes^c{}_{\varphi \in \Phi}\, \varphi(A)$
        tant que $\mathcal{S} \neq \emptyset$ faire
            choisir $s \in \mathcal{S}$
            $\mathcal{S} \leftarrow \mathcal{S} - \{s\}$
            $val \leftarrow val \otimes^c$ TS-mcdag($s, A, V(s) - V(c), \Phi(s)$)
        retourner ($val$)
    sinon
        choisir $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi(c)\,,\ sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $val \leftarrow \Diamond$
        tant que $d \neq \emptyset$ faire
            choisir $a \in d$
            $d \leftarrow d - \{a\}$
            $val \leftarrow val \oplus^c (\otimes^c{}_{\varphi \in \Phi_0}\, \varphi(A.(x,a))) \otimes^c$ TS-mcdag($c, A.(x,a), V - \{x\}, \Phi - \Phi_0$)
        retourner ($val$)
fin
```

Figure 8.1: Un algorithme de recherche arborescente structurée sur un MCDAG.

La premier appel est **TS-mcdag**$(r, \emptyset, V(r), \Phi(r))$. Il est en fait possible de montrer que **TS-mcdag**(c, A, V, Φ) calcule la quantité $val(c, A, V, \Phi)$.

Proposition 8.3. *L'algorithme **TS-mcdag** est correct et complet, c'est-à-dire qu'il renvoie $Ans(Q)$.*

Proposition 8.4. *Soit M un MCDAG associé à une requête $Q = (Sov, (V, G, P, \emptyset, U))$. La complexité spatiale de l'algorithme **TS-mcdag** est $O(h \cdot (d + m))$ et sa complexité spatiale est*

$$O(m \cdot \mu \cdot d^h),$$

avec d la taille maximale des domaines des variables, h la hauteur du MCDAG, μ le nombre maximal de parents pour un cluster du MCDAG ($\mu = 1$ si le MCDAG est un arbre) et $m = |P \cup U|$ dans le cas semi-anneau et $m = (1 + |P|)(1 + |U|)$ dans le cas semi-groupe.

La proposition 8.4 montre que la largeur induite n'est pas le seul critère permettant de juger la qualité d'un MCDAG : la hauteur peut aussi être un critère de choix de MCDAG.

8.3 Ajout de techniques de mémorisation

L'algorithme **TS-mcdag** peut être amené à faire de nombreuses fois le même calcul. Mémoriser le résultat de certaines opérations permet d'éviter de telles redondances dans les calculs, en troquant de la complexité spatiale pour un gain de complexité temporelle. Le point clé est qu'étant donnés un cluster c et un fils s de c, l'algorithme précédent calcule la valeur $val(s)(A)$ du fils s pour de nombreuses affectations A, alors que certaines affectations mènent nécessairement au même résultat du fait de la structure du problème. La raison est que $val(s)(A) = val(s)(A')$ dès que A et A' coïncident sur le séparateur entre c et s, c'est-à-dire dès que $A^{\downarrow c \cap s} = A'^{\downarrow c \cap s}$. Afin d'éviter ces calculs redondants, nous introduisons une structure de mémorisation : la valeur mémorisée pour $val(s)(A^{\downarrow c \cap s})$ est notée $rec(s, A^{\downarrow c \cap s})$ et vaut **nil** si aucune valeur n'est mémorisée pour $val(s)(A^{\downarrow c \cap s})$. L'algorithme actualisé, appelé **RecTS-mcdag** comme *TS-mcdag with Recording*, est donné à la figure 8.2. Par rapport à l'algorithme précédent, seule une ligne est ajoutée.

Proposition 8.5. *L'algorithme **RecTS-mcdag** est correct et complet, c'est-à-dire qu'il renvoie $Ans(Q)$.*

Proposition 8.6. *Soit M un MCDAG associé à une requête $Q = (Sov, (V, G, P, F, U))$. Soit w la largeur du MCDAG. Calculer $Ans(Q)$ avec l'algorithme **RecTS-mcdag** sur le MCDAG M est $O(m \cdot d^{w+1})$ en temps, avec $m = |P \cup U|$ dans le cas semi-anneau et $m = (1 + |P|) \cdot (1 + |U|)$ dans le cas semi-groupe. La complexité spatiale est $O(N \cdot s \cdot d^s)$ avec N le nombre de clusters dans le MCDAG et s la taille du plus grand séparateur.*

Notons également que l'algorithme **RecTS-mcdag** peut être facilement transformé en une version *any-space*, dans laquelle certaines mémorisations peuvent être oubliées lorsque la mémoire disponible devient trop faible.

8.4 Utilisation de bornes

Un des intérêts de la recherche arborescente est de pouvoir élaguer certaines branches de l'espace de recherche. Cet élagage induit généralement un gain important en termes de complexités

```
RecTS-mcdag($c, A, V, \Phi$)
début
    si ($V = \emptyset$) alors
        $\mathcal{S} \leftarrow Sons(c)$
        $val \leftarrow \otimes^c{}_{\varphi \in \Phi}\, \varphi(A)$
        tant que $\mathcal{S} \neq \emptyset$ faire
            choisir $s \in \mathcal{S}$
            $\mathcal{S} \leftarrow \mathcal{S} - \{s\}$
            si $\underline{(\mathbf{rec}(\mathbf{s}, \mathbf{A}^{\downarrow \mathbf{c} \cap \mathbf{s}}) = \mathbf{nil})}$ alors $\underline{\mathbf{rec}(\mathbf{s}, \mathbf{A}^{\downarrow \mathbf{c} \cap \mathbf{s}}) \leftarrow \textbf{RecTS-mcdag}(\mathbf{s}, \mathbf{A}, \mathbf{V}(\mathbf{s}) - \mathbf{V}(\mathbf{c}), \boldsymbol{\Phi}(\mathbf{s}))}$
            $val \leftarrow val \otimes^c rec(s, A^{\downarrow c \cap s})$
        retourner ($val$)
    sinon
        choisir $x \in V$
        $d \leftarrow dom(x)$
        $\Phi_0 \leftarrow \{\varphi \in \Phi\,,\ sc(\varphi) \cap (V - \{x\}) = \emptyset\}$
        $val \leftarrow \Diamond$
        tant que $d \neq \emptyset$ faire
            choisir $a \in d$
            $d \leftarrow d - \{a\}$
            $val \leftarrow val \oplus^c (\otimes^c{}_{\varphi \in \Phi_0}\, \varphi(A.(x,a))) \otimes^c$ RecTS-mcdag($c, A.(x,a), V - \{x\}, \Phi - \Phi_0$)
        retourner ($val$)
fin
```

Figure 8.2: Algorithme de recherche arborescente structurée utilisant de la mémorisation.

temporelle et spatiale en pratique car il permet d'éviter d'avoir à considérer des parties entières de l'espace de recherche.

Nous supposons dans ce qui suit que le MCS totalement ordonné $(E, \oplus, \otimes)$ admet un élément minimum $\bot$ égal à 0_E et un élément maximum $\top$. Cette hypothèse supplémentaire est gratuite dès que le MCS considéré satisfait l'axiome $(x \otimes y = 0_E) \rightarrow ((x = 0_E) \vee (y = 0_E))$, qui est satisfait par toutes les structures classiques d'utilité espérée.

8.4.1 Utilisation de bornes en présence de plusieurs opérateurs d'élimination

Une première difficulté dans l'adaptation des techniques de branch-and-bound sur les MCDAGs réside dans la présence de plusieurs opérateurs d'élimination. Pour remédier à cette difficulté, nous adaptons des idées venant de l'algorithme alpha-beta [55] utilisé en théorie des jeux (cet algorithme a d'ailleurs été étendu au cas des jeux stochastiques [4] où les opérateurs d'élimination min, max et + peuvent alterner).

L'adaptation des techniques alpha-beta nous conduit à utiliser une borne inférieur LB et une borne supérieure UB qui spécifient que la résultat d'un calcul local doit être compris strictement entre LB et UB pour être intéressant du point de vue du calcul global. Informellement, les clusters utilisant max comme opérateur d'élimination renforceront l'exigence donnée par LB (de manière à toujours chercher un résultat meilleur que le meilleur résultat connu) et les clusters utilisant min comme opérateur d'élimination renforceront l'exigence donnée par UB (de manière à toujours chercher un résultat pire que le pire résultat connu). Nous utilisons également deux valeurs spéciales $\bot^-$ et $\top^+$, avec $\bot^-$ plus petit que tout élément de E et $\top^+$ plus grand que tout élément de $E \cup \{\bot^-\}$. Imposer les exigences $LB = \bot^-$ et $UB = \top^+$ signifie qu'aucune exigence n'est imposée sur le résultat cherché.

8.4.2 Utilisation de bornes sans inverse pour les opérateurs de combinaison

Une seconde difficulté réside dans le fait que la structure algébrique de MCS totalement ordonné ne permet pas de supposer l'existence d'un opérateur de différence $\oslash$ inverse de $\otimes$ ou l'existence d'un opérateur de différence $\ominus$ inverse de $\oplus$.

Du fait de la factorisation en fonctions locales, il se peut que l'on souhaite imposer des exigences telles que $e_{\otimes} \otimes val \prec UB$ sur une valeur val à calculer, avec $e_{\otimes}$ un facteur qui doit être combiné avec val en utilisant $\otimes$. Comme nous ne supposons pas l'existence d'un opérateur de division $\oslash$, nous ne pouvons pas directement imposer une exigence du type $val \prec UB \oslash e_{\otimes}$ et prendre $UB' = UB \oslash e_{\otimes}$ comme nouvelle borne supérieure imposée sur val.

La même remarque s'applique pour des exigences du type $val \oplus e_{\oplus} \prec UB$ avec $e_{\oplus}$ un facteur à combiner avec val en utilisant $\oplus$, car nous ne supposons pas l'existence d'un opérateur de différence $\ominus$ inverse de $\oplus$.

C'est pourquoi nous allons avoir besoin d'imposer des exigences complexes de la forme $e_{\otimes} \otimes val \oplus e_{\oplus} \prec UB$ ou $LB \prec e_{\otimes} \otimes val \oplus e_{\oplus}$. De plus, les facteurs $e_{\otimes}$ et $e_{\oplus}$ peuvent eux-mêmes ne pas être connus précisément : il se peut que seulement une borne inférieure $lb_{\otimes}$ et une borne supérieure $ub_{\otimes}$ sur $e_{\otimes}$ soient disponibles, et que seulement une borne inférieure $lb_{\oplus}$ et une borne supérieure $ub_{\oplus}$ sur $e_{\oplus}$ soient disponibles. Afin de manipuler uniquement des exigences à une inconnue, nous aurons besoin d'imposer des exigences de la forme :

$$(LB \prec ub_{\otimes} \otimes val \oplus ub_{\oplus}) \wedge (lb_{\otimes} \otimes val \oplus lb_{\oplus} \prec UB) \tag{8.1}$$

Ceci nous conduit à définir la notion de *borne complexe*.

Définition 8.7. *Une borne complexe est un sextuplet* $(LB, UB, lb_{\otimes}, ub_{\otimes}, lb_{\oplus}, ub_{\oplus})$ *tel que* $LB \prec UB$*,* $lb_{\otimes} \preceq ub_{\otimes}$ *et* $lb_{\oplus} \preceq ub_{\oplus}$*.*

Informellement, imposer une borne complexe $(LB, UB, lb_{\otimes}, ub_{\otimes}, lb_{\oplus}, ub_{\oplus})$ sur une quantité val à calculer signifie imposer l'équation 8.1. Grâce aux bornes complexes, certaines branches de l'espace de recherche vont pouvoir être coupées. Ainsi, si une branche de l'espace de recherche est chargée de calculer la valeur $val(c, A, V, \Phi)$ tout en satisfaisant une exigence complexe $\mathcal{B}$, alors la valeur exacte de $val(c, A, V, \Phi)$ n'est pas requise s'il est prouvé que l'exigence complexe $\mathcal{B}$ est nécessairement violée. Afin de représenter cela, nous introduisons la notion d'*évaluation bornée*.

Définition 8.8. *Soit* $\mathcal{B} = (LB, UB, lb_{\otimes}, ub_{\otimes}, lb_{\oplus}, ub_{\oplus})$ *une borne complexe. Une évalution de* $val(c, A, V, \Phi)$ *bornée par* $\mathcal{B}$ *est un couple* $(lb, ub) \in E^2$ *tel que* $lb \preceq val(c, A, V, \Phi) \preceq ub$ *et tel que* $(lb = ub) \vee (lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes ub \oplus ub_{\oplus}) \vee (LB \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}) \vee (UB \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus})$*.*

En d'autres termes, une évaluation de $val(c, A, V, \Phi)$ bornée par $\mathcal{B}$ doit fournir des bornes inférieures et supérieures lb et ub sur $val(c, A, V, \Phi)$ telles que l'une des conditions suivantes est satisfaite

1. $lb = ub$, i.e. la valeur exacte de $val(c, A, V, \Phi)$ est connue ;

2. $lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes ub \oplus ub_{\oplus}$: dans ce cas, quelle que soit la valeur exacte de $val(c, A, V, \Phi)$, nous aurons $e_{\otimes} \otimes val(c, A, V, \Phi) \oplus e_{\oplus} = lb_{\otimes} \otimes lb \oplus lb_{\oplus} = ub_{\otimes} \otimes ub \oplus ub_{\oplus}$. Cela signifie que quelle que soit la valeur exacte de $val(c, A, V, \Phi)$, lb et ub garantissent qu'un unique degré global sera obtenu après combinaison avec le reste du problème ;
3. $LB \succeq ub_{\otimes} \otimes ub \oplus ub_{\oplus}$, i.e. la borne supérieure ub prouve que $val(c, A, V, \Phi)$ ne satisfait pas les exigences imposées par $\mathcal{B}$;
4. $UB \preceq lb_{\otimes} \otimes lb \oplus lb_{\oplus}$, i.e. la borne inférieure lb prouve que $val(c, A, V, \Phi)$ ne satisfait pas les exigences imposées par $\mathcal{B}$.

8.4.3 Définition de l'algorithme

Partant de là, il est possible d'ajouter l'utilisation de bornes dans un algorithme de recherche arborescente structurée. Cet algorithme utilise plusieurs fonctions :

- Une fonction *bound*(c, A, V, Φ), qui renvoie une borne inférieure lb et une borne supérieure ub sur $val(c, A, V, \Phi)$.
- Trois fonctions nommées *evalClusterMin*$(c, A, V, \Phi, \mathcal{B})$, *evalClusterMax*$(c, A, V, \Phi, \mathcal{B})$ et *evalClusterPlus*$(c, A, V, \Phi, \mathcal{B})$ qui calculent, pour un cluster éliminant les variables en utilisant min, max et $\oplus$ respectivement, une évaluation de $val(c, A, V, \Phi)$ bornée par la borne complexe $\mathcal{B}$. Lorsque l'ensemble V des variables à éliminer est vide, cette fonction fait appel à la fonction d'évaluation des fils de c (cf. ci-dessous).
- Une fonction *evalSons*$(c, A, \Phi, \mathcal{B})$, qui calcule une évaluation de $val(c, A, \emptyset, \Phi)$ bornée par $\mathcal{B}$. Cette fonction calcule une combinaison de la valeur des fils s de c. La valeur de chacun des fils (ou une évaluation de cette valeur) est obtenue en utilisant une des trois fonctions ci-dessus, en fonction de l'opérateur d'élimination utilisé par s.
- Une fonction principale, appelée *BTD-mcdag*(), qui renvoie la réponse $Ans(Q)$ à une requête Q. Si r correspond à la racine du MCDAG considéré, cette fonction appelle simplement *evalSons*$(r, \emptyset, \Phi(r), \mathcal{B}_0)$ avec $\mathcal{B}_0 = (\bot^-, \top^+, 1_E, 1_E, 0_E, 0_E)$ une borne complexe “vide” qui n'impose aucune exigence.

La définition formelle de ces fonctions est laissée à la version anglaise de la thèse. Notons que du fait de l'élagage de l'espace de recherche, il se peut que l'on “rentre” dans un cluster c et que l'on en ressorte sans connaître la valeur exacte de c. De ce fait, la structure de mémorisation utilisée n'enregistre pas la valeur exacte d'un cluster pour une affectation donnée, mais seulement des bornes inférieures et supérieures sur cette valeur. La valeur exacte du cluster est connue lorsque ces bornes inférieures et supérieures sont égales. Notons églement que la structure utilisée pour la mémorisation peut être creuse : la structure de mémorisation peut être une table de hachage, une structure du type diagramme de décision binaire [1, 17]... et non forcément une table.

Théorème 8.9. *Si la fonction* bound *est correcte et complète, alors l'algorithme **BTD-mcdag** est correct et complet, c'est-à-dire qu'il renvoie $Ans(Q)$.*

Proposition 8.10. *Soit M un MCDAG associé à une requête $Q = (Sov, (V, G, P, \emptyset, U))$. La complexité temporelle de l'algorithme **BTD-mcdag** est $O(m \cdot \mu \cdot d^h)$, avec h la hauteur du MCDAG, μ le nombre maximal de parents pour un cluster du MCDAG ($\mu = 1$ si le MCDAG est un arbre) et $m = |P \cup U|$ dans le cas semi-anneau et $m = (1 + |P|)(1 + |U|)$ dans le cas semi-goupe. La* *complexité spatiale est $O(N \cdot s \cdot d^s)$, avec N le nombre de clusters dans le MCDAG et s la taille maximale des séparateurs.*

La complexité théorique de l'algorithme **BTD-mcdag** est pire que celle de l'algorithme **RecTS-mcdag** et les deux algorithmes ont la même complexité spatiale. Cependant, cela ne signifie pas que l'algorithme **RecTS-mcdag** est meilleur que l'algorithme **BTD-mcdag** en pratique, car les complexités fournies ici sont uniquement théoriques. En pratique, une recherche arborescente qui utilise des bornes est bien plus efficace qu'une recherche sans bornes malgré une moins bonne complexité théorique.

L'algorithme ainsi défini est applicable pour résoudre des problèmes aussi variés que des problèmes de satisfiabilité stochastique, des CSP stochastiques, des QBF, des QCSP, des MDP factorisés, des diagrammes d'influence probabilistes ou possibilistes, des problèmes MAP (Maximum A Posteriori hypothesis) ou encore des problèmes de planification stochastique ou de conformant planning. Ceci montre l'intérêt de définir des algorithmes génériques dans un cadre générique.

8.5 Utilisation d'opérateurs de différence et de division

Les bornes complexes permettent de définir un algorithme de recherche arborescente structurée utilisant des bornes. Cependant, utiliser des bornes complexes n'est pas gratuit, car par exemple pour chaque test impliquant LB une opération $\otimes$ et une opération $\oplus$ sont réalisées, en plus du test de comparaison pour savoir si LB est satisfaite. Lorsque la structure algébrique est plus riche, il est possible de revenir à des bornes simples (LB, UB) imposant des exigences simples du type $(LB \prec ub) \wedge (lb \prec UB)$, au lieu d'utiliser des bornes complexes $(LB, UB, lb_\otimes, ub_\otimes, lb_\oplus, ub_\oplus)$ et des comparaisons complexes telles que $(LB \prec ub_\otimes \otimes ub \oplus ub_\oplus) \wedge (lb_\otimes \otimes lb \oplus lb_\oplus \prec UB)$.

Les hypothèses algébriques supplémentaires qui permettent d'utiliser des bornes simples sont essentiellement liées à l'existence d'opérations inverses pour $\otimes$ et $\oplus$. Certaines de ces hypothèses se rapprochent des hypothèses faites dans [20] pour les *fair* VCSP ou dans [8] pour les *semiring-based CSP*. Ces hypothèses supplémentaires s'écrivent de la manière suivante :

— Hypothèse supplémentaire sur $\oplus$, notée "$Ax^\ominus$"
  Pour tous $x, y \in E$ tels que $x \preceq y$, l'ensemble $\{z \in E \mid y = z \oplus x\}$ admet un élément maximum noté $y \ominus x$. En d'autres termes, nous supposons l'existence d'une différence maximale entre y et x.

— Hypothèse supplémentaire sur $\otimes$, notée "$Ax^\oslash$", avec deux versions disjointes
  - $Ax_1^\oslash$: ou bien $1_E = \top$ et pour tous $x, y \in E$ tels que $x \preceq y$, l'ensemble $\{z \in E \mid x = z \otimes y\}$ admet un élément maximum noté $x \oslash y$ (c'est-à-dire qu'il existe une différence maximale entre x et y).
  - $Ax_2^\oslash$: ou bien $1_E \neq \top$ et $\top = \top^+$ (ce qui signifie que $\top$ a été ajouté à la structure de départ) et pour tous $x, y \in E$ tels que $y \notin \{0_E, \top\}$, il existe un unique $z \in E$, noté $x \oslash y$, tel que $x = y \otimes z$.

Ces axiomes sont satisfaits dans de nombreux cas classiques. Par exemple, l'axiome $Ax^\ominus$ est satisfait pour $(E, \preceq, \oplus) = ([0, +\infty], \leq, +)$, $(E, \preceq, \oplus) = ([0, +\infty], \geq, \min)$ et $(E, \preceq, \oplus) = ([0, 1], \leq, \max)$. L'hypothèse supplémentaire sur $\otimes$ est satisfaite avec $(E, \preceq, \otimes) = ([0, +\infty], \geq, +)$ et $(E, \preceq, \otimes) = ([0, 1], \leq, \min)$ dans le premier cas ($1_E = \top$) et avec $(E, \preceq, \otimes) = ([0, +\infty], \leq, \times)$ dans le second cas ($1_E \neq \top$).

Comme indiqué dans le tableau 8.1, dès que $Ax^{\ominus}$ et $Ax^{\oslash}$ sont vérifiés, il est possible d'utiliser des bornes simples. Ce tableau montre que si val est une quantité à calculer, des exigences telles que $\alpha \oplus val \prec UB$, $\alpha \oplus val \succ LB$, $\alpha \otimes val \prec UB$ ou $\alpha \otimes val \succ LB$ peuvent être transformées en des exigences pour lesquelles il suffit de comparer val avec une borne inférieure actualisée LB' ou avec une borne supérieure actualisée UB'.

Cas	Exigence complexe	Condition	Exigence simple
$Ax^{\ominus}$	$\alpha \oplus val \prec UB$	$UB \preceq \alpha$	$val \prec \bot^-$
		$\alpha \prec UB$	$val \prec UB \ominus \alpha$
	$LB \prec \alpha \oplus val$	$LB \prec \alpha$	$val \succ \bot^-$
		$\alpha \preceq LB$	$val \succ LB \ominus \alpha$
$Ax_1^{\oslash}$	$\alpha \otimes val \prec UB$	$UB \preceq \alpha$	$val \prec UB \oslash \alpha$
		$\alpha \prec UB$	$val \prec \top^+$
	$LB \prec \alpha \otimes val$	$LB \prec \alpha$	$val \succ LB \oslash \alpha$
		$\alpha \preceq LB$	$val \succ \top^+$
$Ax_2^{\oslash}$	$\alpha \otimes val \prec UB$	$\alpha \notin \{0_E, \top\}$	$val \prec UB \oslash \alpha$
		$\alpha = 0_E$	$val \prec \top^+$
	$LB \prec \alpha \otimes val$	$\alpha \notin \{0_E, \top\}$	$val \succ LB \oslash \alpha$
		$(\alpha = 0_E) \wedge (LB \neq \bot^-)$	$val \succ \top^+$
		$(\alpha = 0_E) \wedge (LB = \bot^-)$	$val \succ \bot^-$
		$\alpha = \top$	$val \succ \bot^-$

TABLE 8.1 – De bornes complexes à des bornes simples en utilisant des opérations de différence et de division, avec $(\alpha, val) \in E^2$ et $LB \prec UB$. Pour $Ax_2^{\oslash}$, le cas de l'exigence $\alpha \otimes val \prec UB$ combinée avec $\alpha = \top$ n'est pas considére car il n'apparaîtra jamais en pratique (informellement, lorsqu'une exigence de la forme $\alpha \otimes val \prec UB$ sera imposée, α correspondra à une borne inférieure sur une quantité dans $E - \{\top\}$, et donc $\alpha \neq \top$).

De nouvelles versions des fonctions *evalClusterMin*, *evalClusterMax*, *evalClusterPlus* et *evalSons* peuvent alors être définies. Ces nouvelles versions utilisent uniquement des bornes simples (LB, UB) et doivent calculer des évaluations de quantités $val(c, A, V, \Phi)$ bornées par des bornes simples, comme défini ci-dessous.

Définition 8.11. *Une évaluation de $val(c, A, V, \Phi)$ bornée par une borne simple (LB, UB) est un couple $(lb, ub) \in E^2$ tel que $lb \preceq val(c, A, V, \Phi) \preceq ub$ et $(lb = ub) \vee (UB \preceq lb) \vee (ub \preceq LB)$.*

En d'autres termes, une évaluation de $val(c, A, V, \Phi)$ bornée par (LB, UB) est tout simplement une couple de bornes inférieures et supérieures sur $val(c, A, V, \Phi)$ qui soit donnent la valeur exacte de $val(c, A, V, \Phi)$, soit prouvent qu'une des exigences imposées par LB et UB est violée.

La nouvelle fonction principale est appelée **BTD-answerQ**().

8.6 Calcul de bornes

Les algorithmes précédemment introduits utilisent une fonction nommée *bound* qui permet de calculer des bornes sur certaines valeurs. Afin de ne pas retourner des bornes naïves telles que $(\bot, \top)$, de nombreuses techniques peuvent être utilisées :

— *Calcul de bornes par propagation de contraintes* [63, 13, 63, 10, 20, 59].

— *Inversion de quantificateurs* : inverser certains quantificateurs peut donner une borne inférieure ou supérieure et le calcul à réaliser après inversion de quantificateurs peut être très simple en termes de largeur induite, même si la quantité à calculer avant inversion est compliquée à calculer en termes de largeur.

— *Changement de quantificateurs* : une autre technique peut être de remplacer des min par des max, des max par des min, des $\oplus$ par des max... Pourvu que les opérations de remplacement soit diminuent toujours la quantité à calculer, soit l'augmentent toujours, la quantité obtenue après changement de quantificateurs peut être très facile à calculer et fournir une borne inférieure ou supérieure. Ce type de techniques a déjà été utilisé dans [92] sur des QBF : dans ce travail, l'idée consiste à remplacer des quantificateurs $\forall$ par des quantificateurs $\exists$ pour obtenir une forme ne faisant intervenir que des quantificateurs $\exists$, et donc pour laquelle aucune contrainte sur l'ordre d'élimination n'est imposée (d'où un calcul potentiellement beaucoup plus simple).

— *Mini-buckets* [33] : l'idée des techniques *mini-bucket* est de conserver des fonctions locales dont la portée a une taille inférieure à un certain seuil. La largeur induite est ainsi contrôlée et même si le résultat obtenu n'est pas exact, il peut fournir des bornes sur la quantité à calculer.

— *Simplification de la structure algébrique* : on peut travailler sur une formulation du problème utilisant une structure algébrique plus simple, résoudre ce problème plus simple et enfin transformer le résultat obtenu en bornes sur le problème initial [7, 24]

8.7 Résumé et perspectives

Ce chapitre a montré comment un algorithme générique de recherche arborescente structurée utilisant des bornes pouvait être défini pour calculer la valeur d'un MCDAG. Les points centraux sont la gestion de l'aspect multi-opérateur pour les combinaisons et les éliminations et la gestion des bornes élaguant l'espace de recherche. Des résultats de complexité théorique ont également été établis en fonction de paramètres divers que sont la largeur du MCDAG, sa hauteur, ou encore la taille maximale d'un séparateur entre deux clusters.

D'un autre côté, on pourrait envisager de définir des algorithmes de résolution approchée utilisant des techniques d'échantillonnage [66, 87] ou de recherche locale [67] : échantillonnage pour des éliminations avec l'opérateur + (+, et non $\oplus$) et recherche locale pour des éliminations par les opérateurs min et max. Ceci est une des voies à explorer pour définir de nouveaux algorithmes génériques.

D'un point de vue plus pragmatique, les algorithmes définis dans ce chapitre font en fait intervenir plusieurs paramètres dont l'influence reste à étudier :

— Heuristiques pour le choix de la prochaine variable à affecter dans le cluster courant, pour le choix de la valeur à affecter à une variable, pour le choix du prochain cluster fils à considérer...

— Calcul de bornes : quelques pistes ont été données concernant le calcul de bornes, mais beaucoup de travail reste à accomplir pour trouver de bons réglages (par exemple concernant le degré de cohérence locale à établir lors d'une propagation de contraintes).

Beaucoup d'études sur tous ces paramètres ont déjà été réalisées dans chacun des formalismes couverts par le cadre PFU. Afin d'acquérir une meilleure connaissance concernant leur “influence générique” et afin de tester l'efficacité pratique des algorithmes définis, il est nécessaire d'obtenir des résultats expérimentaux. C'est pourquoi un outil de résolution générique permettant de répondre à des requêtes PFU génériques a été développé.

Chapitre 9

Un outil générique pour répondre à des requêtes PFU

Ce chapitre décrit brièvement l'outil de résolution générique qui a été implémenté pour résoudre des requêtes sur des réseaux PFU. Il présente d'abord les formats utilisés pour décrire des réseaux PFU et des requêtes, avant de présenter l'outil générique développé. Le but principal de ce chapitre est de montrer que le cadre PFU n'est pas juste une abstraction. Voir la version anglaise de la thèse pour plus de détails.

Quelques résultats expérimentaux préliminaires ont été obtenus sur un problème réel de déploiement et de maintien d'une constellation de satellites, mais poursuivre les expérimentations sur d'autres problèmes est nécessaire pour approfondir les points suivants :

- Comparer les algorithmes définis précédemment en termes de complexité pratique :
  - quantifier les gains obtenus grâce aux techniques de structuration des requêtes (qui fournissant l'architecture MCDAG),
  - comparer les algorithmes d'élimination de variable avec les méthodes de recherche arborescente structurée,
  - comparer les algorithmes de recherche arborescente structurée pour plusieurs réglages : mémorisation ou non, bornes simples ou bornes complexes, heuristiques pour les choix de variables, de valeurs ou de cluster fils à explorer, techniques utilisées pour calculer des bornes (cohérence locale souple, inversion de quantificateurs...)
- Comparer les algorithmes implémentés avec des algorithmes développés dans des cadres spécifiques
- Evaluer la complexité induite par l'utilisation d'une structure d'utilité espérée donnée, pour quantifier par exemple le coût engendré par l'utilisation de modèles de plausibilités/utilités plus ou moins qualitatifs ou quantitatifs (par exemple, pour une même forme de réseau PFU et pour une même requête, comparer les temps de calcul obtenus si l'on utilise un modèle d'utilité espérée additive probabiliste, un modèle de satisfation espérée, un modèle d'utilité espérée possibiliste...).

Conclusion

Synthèse des contributions

Au cours des dernières décennies, de nombreux formalismes ont été définis pour modéliser et résoudre des problèmes de décision. Dans cette thèse, nous avons introduit un nouveau cadre de représentation générique pour des problèmes de décision séquentielle mettant en jeu des incertitudes, des infaisabilités et des utilités. Ce cadre flexible couvre non seulement de nombreuses approches existantes, telles que les CSP durs, valués, quantifiés, mixtes et stochastiques, les réseaux bayésiens, les champs de Markov, les processus décisionnels markoviens probabilistes ou possibilistes, ou encore les diagrammes d'influence, mais il permet aussi de définir directement de nouveaux formalismes correspondant à certaines de ses instanciations.

Au final, le cadre obtenu, appelé le cadre PFU, est à la fois algébrique et justifié d'un point de vue de la théorie de la décision. Ces deux facettes, qui sont le résultat d'une conception prenant en compte à la fois des enjeux en termes d'expressivité et des enjeux algorithmiques, apparaissent clairement dans le théorème 5.4, qui établit une équivalence formelle entre d'une part la définition de la réponse à une requête à partir d'arbre de décision et d'autre part la définition opérationnelle de la valeur d'une requête. Comparée aux approches algébriques génériques existantes [97, 25, 56], le cadre PFU est le seul qui manipule explicitement plusieurs types de variables (variables de décision et variables d'environnement), plusieurs types de fonctions locales (plausibilités, faisabilités et utilités) et plusieurs types d'opérateurs de combinaison et d'élimination.

D'un point de vue opérationnel, des algorithmes génériques à base de recherche arborescente ou d'élimination de variables ont également été définis. Certaines conditions dites de décomposabilité, permettant d'utiliser toutes les factorisations de quantités globales en fonctions locales, ont été identifiées et utilisées au sein d'un algorithme unifié d'élimination de variables (utilisant éventuellement des éléments appelés potentiels). Une autre voie a ensuite été explorée avec des mécanismes d'optimisation de la structure d'une requête. Cette voie nous a conduit à définir deux nouvelles architectures de calcul permettant de répondre à des requêtes de manière plus efficace, l'architecture des *arbre de clusters multi-opérateurs* (MCTree) et l'architecture des *DAGs de clusters multi-opérateurs* (MCDAG). Ces architectures ont été construites en utilisant un mécanisme de structuration en deux temps utilisant d'une part des règles de réécriture et d'autre part des techniques de décomposition en arbres de clusters. Cette structuration permet d'améliorer un paramètre connu sous le nom de largeur induite (ou largeur d'arbre) qui influence les complexités spatiales et temporelles de résolution d'une requête. A partir de ces architectures, nous avons défini des algorithmes de recherche arborescente plus ou moins sophistiqués suivant s'ils utilisent des techniques de mémorisation ou d'élagage par des bornes. La principale difficulté a été de réussir à gérer la nature multi-opérateur des requêtes PFU, à la fois en terme d'élimination et de combinaison. Naturellement, certaines hypothèses faites par le cadre PFU sont discutables. Mais il est nécessaire de garder à l'esprit que ces mêmes hypothèses ont permis d'explorer des approches algorithmiques variées. Enfin, un outil générique de résolution a été implémenté.

Toutes ces contributions sont résumées à la figure 9.1

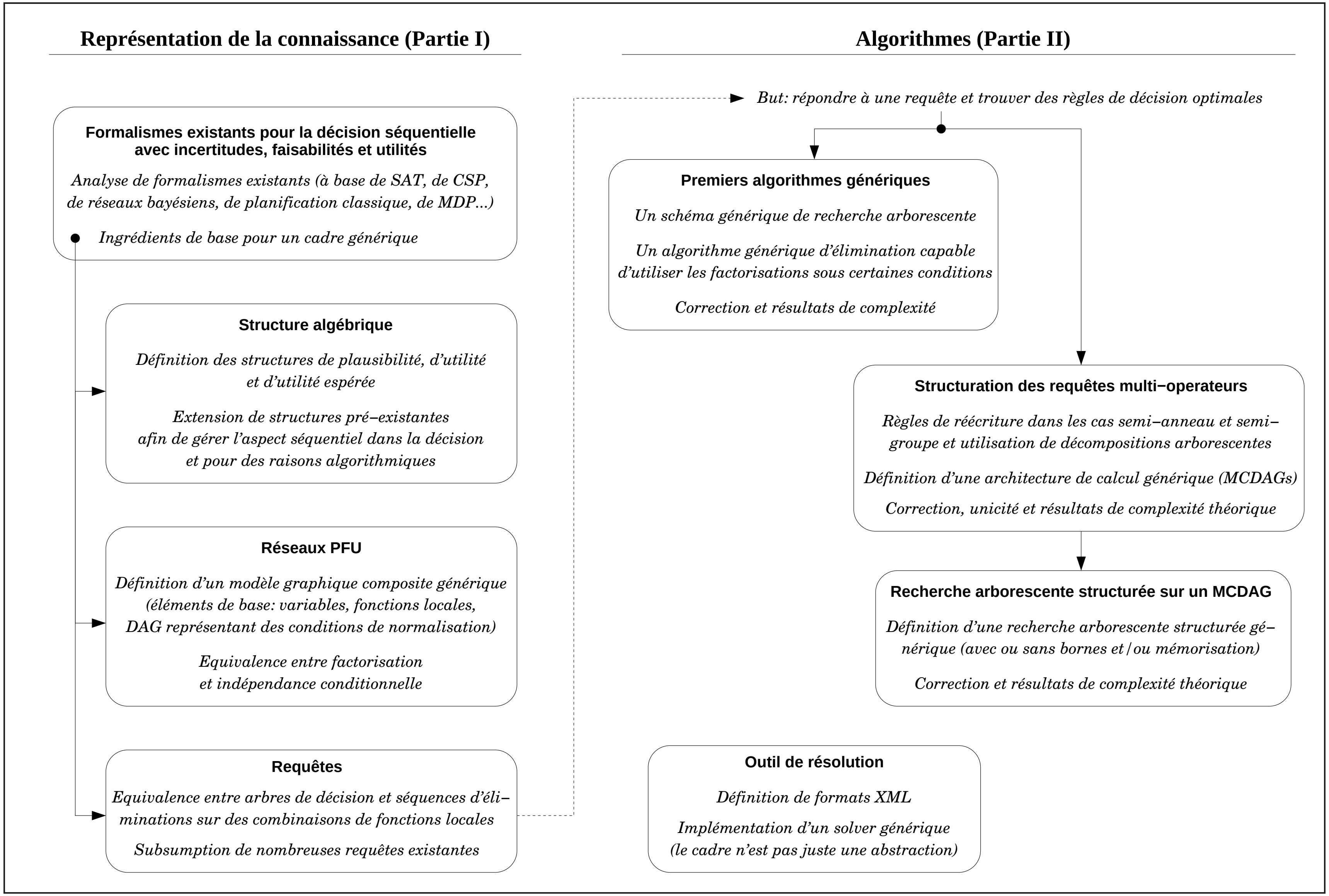
D'un point de vue plus général, les conclusions de cette thèse sont les suivantes :

1. Construire un cadre générique englobant de nombreux formalismes existants est possible, et le cadre obtenu n'est pas un monstre incompréhensible et ingérable. Il correspond juste à une forme générique de *modèle graphique algébrique composite*.
2. Des algorithmes génériques unifiés peuvent être définis dans ce cadre et, comme pour les approches algébriques existantes, ces algorithmes offrent des complexité théoriques spatiales et temporelles fonctions de paramètres topologiques tels que la largeur d'arbre. En termes de largeur, une analyse précise de la structure des requêtes multi-opérateur considérées peut être fortement bénéfique.
3. Répondre à des requêtes multi-opérateurs est équivalent à répondre à plusieurs requêtes mono-opérateurs organisées dans une architecture générique appelée l'architecture MCDAG. Cette dernière peut être obtenue de manière systématique et dès qu'elle est disponible, les algorithmes de résolution du cas mono-opérateur peuvent être réutilisés. La principale difficulté concerne la gestion des bornes, qui doivent être utilisés en synergie avec les multiples opérateurs de combinaison et d'élimination. En réalité, les MCDAGs peuvent être utilisés quelle que soit la méthode de résolution utilisée (élimination de variables, recherche arborescente ou algorithmes approchés), car ils expriment juste des décompositions.

Perspectives

Les perspectives de ce travail sont multiples :

- Comme indiqué au chapitre 9, obtenir des résultats expérimentaux est l'un des objectifs à court terme, ce pour pouvoir obtenir une meilleure connaissance concernant les algorithmes proposés.
- Les méthodes de structuration raisonnent en fonction des variables. Nous pourrions aussi tenter d'exploiter des structures présentes à des niveaux plus fins, par example au niveau de la valeur des fonctions locales, en utilisant des approches telles que les diagrammes de décision binaires (*Binary Decision Diagrams*, BDDs [1, 17]) ou les *Negation Normal Forms* (NNFs [23]).
- Nous pourrions également étudier plus précisément les résultats fournis par les méthodes de structuration pour des requêtes et des réseaux répliqués sur plusieurs pas de temps, comme dans un processus décisionnel markovien.
- Beaucoup de choses restent à faire concernant l'utilisation de bornes, une des idées maîtresses pouvant être de définir une sorte d'*arc cohérence quantifiée généralisée*.
- D'un point de vue plus général, deux attitudes opposées peuvent être adoptées concernant le cadre lui-même. Ces deux attitudes ne sont pas incompatibles et correspondent respectivement à une démarche de généralisation et à une démarche de spécialisation :



Représentation de la connaissance (Partie I)

Formalismes existants pour la décision séquentielle avec incertitudes, faisabilités et utilités

Analyse de formalismes existants (à base de SAT, de CSP, de réseaux bayésiens, de planification classique, de MDP...)

Ingrédients de base pour un cadre générique

Structure algébrique

Définition des structures de plausibilité, d'utilité et d'utilité espérée

Extension de structures pré−existantes afin de gérer l'aspect séquentiel dans la décision et pour des raisons algorithmiques

Réseaux PFU

Définition d'un modèle graphique composite générique (éléments de base: variables, fonctions locales, DAG représentant des conditions de normalisation)

Equivalence entre factorisation et indépendance conditionnelle

Requêtes

Equivalence entre arbres de décision et séquences d'éli−minations sur des combinaisons de fonctions locales

Subsumption de nombreuses requêtes existantes

Algorithmes (Partie II)

But: répondre à une requête et trouver des règles de décision optimales

Premiers algorithmes génériques

Un schéma générique de recherche arborescente

Un algorithme générique d'élimination capable d'utiliser les factorisations sous certaines conditions

Correction et résultats de complexité

Structuration des requêtes multi−operateurs

Règles de réécriture dans les cas semi−anneau et semi−groupe et utilisation de décompositions arborescentes

Définition d'une architecture de calcul générique (MCDAGs)

Correction, unicité et résultats de complexité théorique

Recherche arborescente structurée sur un MCDAG

Définition d'une recherche arborescente structurée gé−nérique (avec ou sans bornes et/ou mémorisation)

Correction et résultats de complexité théorique

Outil de résolution

Définition de formats XML

Implémentation d'un solver générique (le cadre n'est pas juste une abstraction)

Figure 9.1: Résumé des contributions.

- Une première approche consiste à poursuivre la quête de la généricité, de manière à accroître le pouvoir d'expression du cadre PFU. Nous pourrions aussi définir des sortes de "multi-requêtes" permettant de poser plusieurs requêtes simultanément. C'est ce qui est par exemple implicitement fait dans les réseaux bayésiens pour calculer plusieurs distributions de probabilités marginales simultanément.
- La seconde approche consiste à identifier certains problèmes de base à résoudre et à se concentrer sur ces problèmes. Plus précisément, l'architecture MCDAG montre que les problèmes élémentaires à résoudre consistent souvent à calculer des quantités du type $\sum_S(\prod_{\varphi\in\Phi}\varphi)$, $\max_S(\sum_{\varphi\in\Phi}\varphi)$, ou $\max_S(\min_{\varphi\in\Phi}\varphi)$. Ces trois problèmes élémentaires correspondent aux types de calculs réalisés dans des réseaux bayésiens [73], des CSP pondérés [60], et des CSPs flous [35] respectivement. Une méthode possible pour justifier cette démarche de spécialisation est d'exhiber des morphismes entre des MCDAGs génériques et des MCDAGs utilisant juste les trois problèmes élémentaires mentionnés précédemment [20]. Une fois ce pré-requis algébrique effectué (s'il est possible de l'effectuer), l'architecture MCDAG peut être vue comme un entrelacement de ces trois problèmes élémentaires, à la frontière entre réseaux bayésiens et CSPs valués.

Dans les prochaines années, le cadre PFU permettra peut-être d'intégrer de nouvelles idées algorithmiques dans un outil de résolution flexible et générique. Il représente une opportunité de rassembler les nombreux efforts réalisés dans différentes communautés et de bénéficier des liens féconds entre algèbre, modèles graphiques et optimisation combinatoire.

Liste des tableaux

Table des figures

Bibliographie

[1] S.B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, 27(6), 1978.

[2] S.A. Arnborg. Efficient Algorithms for Combinatorial Problems on Graphs with Bounded Decomposability - A Survey. *BIT*, 25 :2–23, 1985.

[3] F. Bacchus and A. Grove. Graphical Models for Preference and Utility. In *Proc. of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 3–10, Montréal, Canada, 1995.

[4] B.W. Ballard. The *-Minimax Search Procedure for Trees Containing Chance Nodes. *Artificial Intelligence*, 21(3) :327–350, 1983.

[5] M. Benedetti. Quantifier Trees for QBF. In *Proc. of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, St. Andrews, Scotland, 2005.

[6] U. Bertelé and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

[7] S. Bistarelli, P. Codognet, and F. Rossi. Abstracting Soft Constraints : Framework, Properties, Examples. *Artificial Intelligence*, 139 :175–211, 2002.

[8] S. Bistarelli and F. Gadducci. Enhancing Constraints Manipulation in Semiring-based Formalisms. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, Riva del Garda, Italy, 2006.

[9] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 624–630, Montréal, Canada, 1995.

[10] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Satisfaction and Optimization. *Journal of ACM*, 44(2) :201–236, 1997.

[11] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-Based CSPs and Valued CSPs : Frameworks, Properties and Comparison. *Constraints*, 4(3) :199–240, 1999.

[12] H. L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11 :1–21, 1993.

[13] L. Bordeaux and E. Monfroy. Beyond NP : Arc-consistency for Quantified Constraints. In *Proc. of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*, Ithaca, New York, USA, 2002.

[14] C. Boutilier, R. Brafman, H. Hoos, and D. Poole. Reasoning With Conditional Ceteris Paribus Preference Statements. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, Stockholm, Sweden, 1999.

[15] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning : Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11 :1–94, 1999.

[16] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence*, 121(1-2) :49–107, 2000.

[17] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8) :677–691, 1986.

[18] R. Chellappa and A. Jain. Markov Random Fields : Theory and Applications. Academic Press, 1993.

[19] F. Chu and J. Halpern. Great Expectations. Part I : On the Customizability of Generalized Expected Utility. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

[20] M. Cooper and T. Schiex. Arc Consistency for Soft Constraints. *Artificial Intelligence*, 154(1-2) :199–227, 2004.

[21] A. Darwiche. Recursive Conditioning. *Artificial Intelligence*, 126(1-2) :5–41, 2001.

[22] A. Darwiche and M.L. Ginsberg. A Symbolic Generalization of Probability Theory. In *Proc. of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 622–627, San Jose, CA, USA, 1992.

[23] A. Darwiche and P. Marquis. A Knowledge Compilation Map. *Artificial Intelligence*, 17 :229–264, 2002.

[24] S. de Givry, G. Verfaillie, and T. Schiex. Bounding the Optimum of Constraint Optimization Problems. In *Proc. of the 3rd International Conference on Principles and Practice of Constraint Programming (CP-97)*, Schloss Hagenberg, Austria, 1997.

[25] R. Dechter. Bucket Elimination : a Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2) :41–85, 1999.

[26] R. Dechter. A New Perspective on Algorithms for Optimizing Policies under Uncertainty. In *Proc. of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, pages 72–81, Breckenridge, CO, USA, 2000.

[27] R. Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[28] R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125(1-2) :93–118, 2001.

[29] R. Dechter and D. Larkin. Hybrid Processing of Beliefs and Constraints. In *Proc. of the 17th International Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 112–119, Seattle, WA, USA, 2001.

[30] R. Dechter and R. Mateescu. Mixtures of Deterministic-Probabilistic Networks and their AND/OR Search Space. In *Proc. of the 20th International Conference on Uncertainty in Artificial Intelligence (UAI-04)*, Banff, Canada, 2004.

[31] R. Dechter and R. Mateescu. AND/OR Search Spaces for Graphical Models. *To appear in Artificial Intelligence Journal*, 2006.

[32] R. Dechter, I. Meiry, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49 :61–95, 1991.

[33] R. Dechter and I. Rish. Mini-Buckets : A General Scheme for Bounded Inference. *Journal of the ACM*, 50(2) :107 – 153, 2003.

[34] R. Demirer and P.P. Shenoy. Sequential Valuation Networks : A New Graphical Technique for Asymmetric Decision Problems. In *Proc. of the 6th European Conference on Symbolic and Quantitavive Approaches to Reasoning with Uncertainty (ECSQARU-01)*, pages 252–265, London, UK, 2001.

[35] D. Dubois, H. Fargier, and H. Prade. The Calculus of Fuzzy Restrictions as a Basis for Flexible Constraint Satisfaction. In *Proc. of the 2nd IEEE Conference on Fuzzy Sets*, pages 1131–1136, San Francisco, CA, 1993.

[36] D. Dubois and H. Prade. Possibility Theory : An Approach to Computerized Processing of Uncertainty. Plenum Press, 1988.

[37] D. Dubois and H. Prade. Possibility Theory as a Basis for Qualitative Decision Theory. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1925–1930, Montréal, Canada, 1995.

[38] H. Fargier, J. Lang, and T. Schiex. Mixed Constraint Satisfaction : a Framework for Decision Problems under Incomplete Knowledge. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 175–180, Portland, OR, USA, 1996.

[39] H. Fargier and P. Perny. Qualitative Models for Decision Under Uncertainty without the Commensurability Assumption. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 188–195, Stockholm, Sweden, 1999.

[40] R. Fikes and N. Nilsson. STRIPS : a New Approach to the Application of Theorem Proving. *Artificial Intelligence*, 2(3-4) :189–208, 1971.

[41] N. Friedman and J. Halpern. Plausibility Measures : A User's Guide. In *Proc. of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 175–184, Montréal, Canada, 1995.

[42] M. Frydenberg. The Chain Graph Markov Property. *Scandinavian Journal of Statistics*, 17 :333–353, 1990.

[43] L. Garcia and R. Sabbadin. Possibilistic Influence Diagrams. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, pages 372–376, Riva del Garda, Italy, 2006.

[44] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning : Theory and Practice.* Morgan Kaufmann, 2004.

[45] P.H. Giang and P.P. Shenoy. A Qualitative Linear Utility Theory for Spohn's Theory of Epistemic Beliefs. In *Proc. of the 16th International Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 220–229, Stanford, California, USA, 2000.

[46] R.P. Goldman and M.S. Boddy. Expressive Planning and Explicit Knowledge. In *Proc. of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 110–117, Edinburgh, Scotland, 1996.

[47] J. Halpern. Conditional Plausibility Measures and Bayesian Networks. *Journal of Artificial Intelligence Research*, 14 :359–389, 2001.

[48] R. Howard and J. Matheson. Influence Diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, Menlo Park, CA, USA, 1984.

[49] P. Jégou and C. Terrioux. Hybrid Backtracking bounded by Tree-decomposition of Constraint Networks. *Artificial Intelligence*, 146(1) :43–75, 2003.

[50] F. Jensen, F.V. Jensen, and S. Dittmer. From Influence Diagrams to Junction Trees. In *Proc. of the 10th International Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 367–373, Seattle, WA, USA, 1994.

[51] F.V. Jensen, T.D. Nielsen, and P.P. Shenoy. Sequential Influence Diagrams : A Unified Asymmetry Framework. In *Proceedings of the Second European Workshop on Probabilistic Graphical Models (PGM-04)*, pages 121–128, Leiden, Netherlands, 2004.

[52] F.V. Jensen and M. Vomlelova. Unconstrained Influence Diagrams. In *Proc. of the 18th International Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 234–241, Seattle, WA, USA, 2002.

[53] L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal Constraint Reasoning with Preferences. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, WA, USA, 2001.

[54] U. Kjaerulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical Report Tech. Report. R 90-09, Dept. of Mathematics and Computer Science, Aalborg University, Denmark, 1990.

[55] D. Knuth and R. Moore. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 8(4) :293–326, 1975.

[56] J. Kolhas. *Information Algebras : Generic Structures for Inference*. Springer, 2003.

[57] A.M.C.A. Koster, H.L. Bodlaender, and S.P.M. Van Hoesel. Treewidth : Computational Experiments. Technical report, Zentrum für Informationstechnik, Berlin, 2001.

[58] N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *Artificial Intelligence*, 76(1-2) :239–286, 1995.

[59] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.

[60] J. Larrosa and T. Schiex. In the Quest of the Best Form of Local Consistency for Weighted CSP. In *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 239–244, Acapulco, Mexico, 2003.

[61] S. Lauritzen and D. Nilsson. Representing and Solving Decision Problems with Limited Information. *Management Science*, 47(9) :1235–1251, 2001.

[62] M. Littman, S. Majercik, and T. Pitassi. Stochastic Boolean Satisfiability. *Journal of Automated Reasoning*, 27(3) :251–296, 2001.

[63] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1) :99–118, 1977.

[64] A. Madsen and F.V. Jensen. Lazy Evaluation of Symmetric Bayesian Decision Problems. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 382–390, Stockholm, Sweden, 1999.

[65] D. McDermott. PDDL, the Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control, 1998.

[66] N. Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44, 1949.

[67] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing Conflicts : a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58 :160–205, 1992.

[68] G. Monahan. A Survey of Partially Observable Markov Decision Processes : Theory, Models, and Algorithms. *Management Science*, 28(1) :1–16, 1982.

[69] U. Montanari and F. Rossi. Constraint Relaxation may be Perfect. *Artificial Intelligence*, 48 :143–170, 1991.

[70] P. Ndilikilikesha. Potential Influence Diagrams. *International Journal of Approximated Reasoning*, 10 :251–285, 1994.

[71] T.D. Nielsen and F.V. Jensen. Representing and solving asymmetric decision problems. *International Journal of Information Technology and Decision Making*, 2 :217–263, 2003.

[72] J. Park and A. Darwiche. Complexity Results and Approximation Strategies for MAP Explanations. *Journal of Artificial Intelligence Research*, 21 :101–133, 2004.

[73] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[74] P. Perny, O. Spanjaard, and P. Weng. Algebraic Markov Decision Processes. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, Scotland, 2005.

[75] C. Pralet, T. Schiex, and G. Verfaillie. Algorithmes et Complexités Génériques pour Différents Cadres de Décision Séquentielle dans l'Incertain. *Revue d'Intelligence Artificielle, à paraître.*

[76] C. Pralet, T. Schiex, and G. Verfaillie. Decomposition of Multi-Operator Queries on Semiring-based Graphical Models. In *Proc. of the 12th International Conference on Principles and Practice of Constraint Programming (CP-06)*, pages 437–452, Nantes, France, 2006.

[77] C. Pralet, T. Schiex, and G. Verfaillie. From Influence Diagrams to Multioperator Cluster DAGs. In *Proc. of the 22nd International Conference on Uncertainty in Artificial Intelligence (UAI-06)*, Cambridge, MA, USA, 2006.

[78] C. Pralet, T. Schiex, and G. Verfaillie. Une Nouvelle Architecture de Calcul pour Résoudre des Diagrammes d'Influence. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA-06)*, Toulouse, France, 2006.

[79] C. Pralet, G. Verfaillie, and T. Schiex. An Algebraic Graphical Model for Decision with Uncertainties, Feasibilities, and Utilities. *Journal of Artificial Intelligence Research, to appear.*

[80] C. Pralet, G. Verfaillie, and T. Schiex. Un Cadre Graphique et Algébrique pour les Problèmes de Décision incluant Incertitudes, Faisabilités et Utilités. *Revue d'Intelligence Artificielle, à paraître.*

[81] C. Pralet, G. Verfaillie, and T. Schiex. Composite Graphical Models for Reasoning about Uncertainties, Feasibilities, and Utilities. In *Proc. of the CP-05 International Workshop on "Preferences and Soft Constraints"*, Sitges, Spain, 2005.

[82] C. Pralet, G. Verfaillie, and T. Schiex. Requêtes Complexes sur des Réseaux de Croyance-Faisabilité-Désir. In *Journées Francophones de Programmation par Contraintes (JFPC-05)*, Lens, France, 2005.

[83] C. Pralet, G. Verfaillie, and T. Schiex. Décision avec Incertitudes, Faisabilités et Utilités : vers un Cadre Algébrique Unifié. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA-06)*, Toulouse, France, 2006.

[84] C. Pralet, G. Verfaillie, and T. Schiex. Decision with Uncertainties, Feasibilities, and Utilities : Towards a Unified Algebraic Framework. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI-06)*, pages 427–431, Riva del Garda, Italy, 2006.

[85] C. Pralet, G. Verfailllie, and T. Schiex. Belief and Desire Networks for Answering Complex Queries. In *Proc. of the CP-04 Workshop on "Constraint Solving under Change and Uncertainty"*, Toronto, Canada, 2004.

[86] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.

[87] C.P. Robert and G. Casella. *Monte Carlo Statistical Methods.* Springer-Verlag, second edition, 2004.

[88] N. Robertson and P.D. Seymour. Graph Minors ii : Algorithmic Aspects of Treewidth. *Journal of Algorithms*, 7 :309–322, 1986.

[89] D.J. Rose. Triangulated Graphs and the Elimination Process. *Journal of Mathematical Analysis and Applications*, 32, 1970.

[90] F. Rossi, B. Venable, and N. Yorke-Smith. Simple Temporal Problems with Preferences and Uncertainty. In *Proc. of the CP-03 Workshop on "Handling Change and Uncertainty"*, Cork, Ireland, 2003.

[91] R. Sabbadin. A Possibilistic Model for Qualitative Sequential Decision Problems under Uncertainty in Partially Observable Environments. In *Proc. of the 15th International Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 567–574, Stockholm, Sweden, 1999.

[92] H. Samulowitz and F. Bacchus. Using SAT in QBF. In *Proc. of the 11th International Conference on Principles and Practice of Constraint Programming (CP-05)*, pages 578–592, Sitges, Spain, 2005.

[93] T. Sang, P. Beame, and H. Kautz. Solving Bayesian Networks by Weighted Model Counting. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 475–482, Pittsburgh, PA, USA, 2005.

[94] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 631–637, Montréal, Canada, 1995.

[95] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.

[96] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

[97] P. Shenoy. Valuation-based Systems for Discrete Optimization. *Uncertainty in Artificial Intelligence*, 6 :385–400, 1991.

[98] P. Shenoy. Valuation-based Systems for Bayesian Decision Analysis. *Operations Research*, 40(3) :463–484, 1992.

[99] P. Shenoy. Conditional Independence in Valuation-Based Systems. *International Journal of Approximated Reasoning*, 10(3) :203–234, 1994.

[100] P.P. Shenoy. Valuation Network Representation and Solution of Asymmetric Decision Problems. *European Journal of Operational Research*, 121 :579–608, 2000.

[101] J.E. Smith, S. Holtzman, and J.E. Matheson. Structuring Conditional Relationships in Influence Diagrams. *Operations Research*, 41 :280–297, 1993.

[102] W. Spohn. A General Non-Probabilistic Theory of Inductive Reasoning. In *Proc. of the 6th International Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pages 149–158, Cambridge, MA, USA, 1990.

[103] T.Vidal and M.Ghallab. Dealing with Uncertain Durations in Temporal Constraint Networks dedicated to Planning. In *Proc. of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, Hungary, 1996.

[104] G. Verfaillie and C. Pralet. The Basic Ingredients of a Constraint-based Framework for Decision-making under Uncertainty. In *Proc. of the CP-05 International Workshop on "Constraint solving under Change and Uncertainty"*, Sitges, Spain, 2005.

[105] T. Vidal and H. Fargier. Handling Contingency in Temporal Constraint Networks : From Consistency to Controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1) :23–45, 1999.

[106] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behaviour*. Princeton University Press, 1944.

[107] T. Walsh. Stochastic Constraint Programming. In *Proc. of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 111–115, Lyon, France, 2002.

[108] E. Weydert. General Belief Measures. In *Proc. of the 10th International Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 575–582, 1994.

Résumé

De nombreux formalismes existent pour modéliser et résoudre des problèmes de décision séquentielle. Certains, comme les réseaux de contraintes, permettent de formuler des problèmes de décision "simples" alors que d'autres peuvent prendre en compte des données plus complexes telles que des incertitudes, des infaisabilités sur les décisions et des utilités. Diverses extensions d'un même formalisme sont de plus souvent introduites de manière à représenter l'incertain et les préférences sous des formes variées (probabilités, possibilités... ; utilités additives ou non...). Chacun de ces formalismes est généralement équipé d'algorithmes dédiés. La première partie de cette thèse définit un cadre de représentation général qui englobe de nombreux formalismes de décision séquentielle dans l'incertain. Ce cadre, nommé cadre PFU pour "Plausibilité-Faisabilité-Utilité", repose sur trois éléments clés : (1) une structure algébrique spécifiant comment combiner et synthétiser des informations ; (2) des fonctions locales portant sur certaines variables et exprimant des incertitudes, des faisabilités ou des utilités ; (3) une classe de requêtes sur ces fonctions locales, qui permet de modéliser des scénarios décisionnels variés en termes d'observabilité et de controlabilité. Ce travail de représentation de la connaissance est complété, dans la seconde partie de la thèse, par un travail algorithmique. Les deux types d'algorithmes développés sont des algorithmes de type élimination de variables et de type recherche arborescente avec bornes et techniques de mémorisation. Nous montrons également qu'il est possible d'utiliser une architecture de calcul générale qui exploite la structure des requêtes considérées pour les décomposer en calcul locaux. En unifiant des formalismes variés, le cadre PFU apporte une meilleure compréhension des liens entre certains formalismes. Il n'est pas qu'un cadre unificateur étant donné que certaines de ces intanciations correspondent à de nouveaux formalismes. Enfin, il permet de définir des algorithmes génériques qui sont soit des généralisations d'algorithmes existants soit des techniques nouvelles applicables directement aux formalismes couverts.

Mots clés : Décision séquentielle, Incertitudes, Préférences, Contraintes, Modèles graphiques algébriques, Architecture de calcul, Décomposition en arbre, Elimination de variables, Programmation dynamique, Recherche arborescente.

Abstract

Numerous formalisms and dedicated algorithms have been designed to model and solve decision making problems. Some formalisms, such as constraint networks, can express "simple" decision problems, while others can take into account uncertainties, unfeasible decisions, and utilities. Even in a single formalism, several variants are often proposed to model different types of uncertainty (probability, possibility...) or utility (additive or not). In the first part of this thesis, we introduce a generic algebraic framework that encompasses a large number of such formalisms: (1) we first adapt existing algebraic structures for representing uncertainty and expected utility in order to deal with generic forms of sequential decision making; (2) on these structures, we introduce composite graphical models that express information via variables linked by "local" functions; (3) on these graphical models, we define a simple class of queries which can represent various scenarios in terms of observabilities and controllabilities. These three elements define the *Plausibility-Feasibility-Utility* (PFU) framework. This work on knowledge representation is completed by the second part of this thesis, which focuses on algorithms for answering PFU queries. Two types of algorithms are introduced: variable elimination algorithms, which can be more or less sophisticated depending on whether they finely analyze the structure of the queries they answer to, and tree search algorithms, which can be more or less advanced depending on whether they use bounds or recording. We also show that queries can be answered using a generic architecture of local computations called the multi-operator cluster DAG architecture. Theoretical complexity results based on tree-width are also provided, as well as a generic solver that answers PFU queries. The PFU framework provides a better understanding of the links between existing formalisms, it covers yet unpublished frameworks, and it unifies formalisms such as quantified booleans formulas and influence diagrams. The algorithms proposed are either generalizations of existing algorithms, or new techniques directly applicable to all subsumed formalisms.

Keywords: Sequential decision-making, Uncertainties, Preferences, Constraints, Algebraic graphical models, Computational architecture, Cluster-tree decomposition, Variable elimination, Dynamic programming, Tree search.

SUPAERO

10, avenue Édouard-Belin - BP 54032 - 31055 Toulouse cedex 4 - France
Tél : 33 (0)5 62 17 80 80 - Fax : 33 0(5) 62 17 83 30 - http://www.supaero.fr
Établissement public à caractère administratif sous tutelle du Ministère de la Défense