N° d'ordre : 341

Année 2002

THÈSE

présentée en vue de
l'obtention du titre de

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

SPÉCIALITÉ : **Intelligence Artificielle**

par

Maria del Pilar POZOS PARRA

Modélisation en logique de systèmes complexes constitués d'agents partiellement autonomes

Soutenue le 3 juin 2002 devant la Commission d'Examen :

Mme	C.	CAYROL	Présidente
MM.	R.	DEMOLOMBE	Directeur de thèse
	L.	FARIÑAS DEL CERRO	
	A.	HERZIG	
Mmes	A.	ILLARRAMENDI	Rapporteur
	C.	SCHWIND	Rapporteur
M.	M.	GHALLAB	Membre invité

Thèse préparée au sein du Département Traitement de l'Information et Modélisation de l'ONERA-centre de Toulouse.

à Rodo

Remerciements

Je voudrais trouver une phrase pour décrire la présence de M. DEMOLOMBE dans ma vie et je n'arrive pas à discerner entre : protecteur de cauchemar de thèse, père scientifique, directeur de rêve, … Pardon, j'envie de finir la liste mais je n'ose pas continuer car mis à part toutes les qualités que M. DEMOLOMBE possède, il est de plus modeste. Merci M. DEMOLOMBE pour toutes vos connaissances offertes avec patience, gentillesse et bonne humeur, votre rigueur subtile et bénéfique sera un grand souvenir.

Concernant la correction de la grammaire française j'ai eu la chance de compter sur des personnes très sympathiques : Patrice Cros, Thérèse Loraschi, Marion Richard, Marc Boyer et Robert Demolombe qui ont amélioré énormément ce travail. Merci beaucoup.

Un grand merci à Mme Cholvy et à Christophe Garion pour résoudre plein des petites questions.

Je tiens à remercier les rapporteurs de thèse : Mme Schwind et Mme Illarramendi qui ont bien voulu accepter la tâche de relecture de ce mémoire malgré la courte période assignée.

Je remercie aussi les membres du jury, en particulier Mme Cayrol pour toutes ses remarques faites au sujet de ce manuscrit.

Je tiens à remercier également les membres de l'équipe DTIM en particulière Noëlle, Christiane et Jeanot. Ainsi que Annie et Marisa de SUPAERO et le personnel du centre d'information de l'IRIT, de l'ONERA et de SUPAERO. Merci aussi à Anna Maneta de SFERE.

Je tiens à exprimer ma profonde gratitude à tous les amis que j'ai pu rencontrer pendant mon séjour en France : Marthita, Cali, Edgardo, Almita et Chuy, Richi et Amparo, Serge et Marilyne (y compris leur famille et amis), Anibal et famille, Sami, Joe, Paul, Bernadette, Thérèse et famille, Irina, Marion et Alex, Douglas et ses enfants, Arely et Nicky, Lety, Stéf (y compris sa famille et ses amis), Serge, Marie-Claire et nos amis du marché bio, sans oublier la "banda de mexicanos".

Comme d'habitude, merci à ma très chère famille et mes amis de toujours qui sont un point de repère quand la vie n'a plus de sens.

Merci à toutes les personnes qui m'ont offert un petit sourire.

Cette thèse a pu être réalisée grâce au soutien financier de l'institution mexicaine CONACyT-SEP (Consejo Nacional de Ciencia y Tecnología).

MODÉLISATION EN LOGIQUE DE SYSTÈMES COMPLEXES CONSTITUÉS D'AGENTS PARTIELLEMENT AUTONOMES

Résumé : Cette thèse concerne la formalisation en logique de systèmes complexes. Notre intérêt porte sur la modélisation de deux aspects des systèmes : son caractère dynamique et sa décomposition en plusieurs entités actives (agents). La modélisation de ce type de systèmes est basée sur le comportement de l'être humain, et pour cela la représentation des notions telles qu'action, croyance, intention, capacité, génération de plans, etc. est nécessaire. Après une courte introduction intuitive de ces notions nous présentons le calcul des situations sur lequel sont basées les théories que nous proposons.

La version classique du calcul des situations ne considère qu'un seul agent. Une des contributions importantes de notre recherche est l'introduction de deux extensions du calcul des situations pour représenter l'évolution des croyances de plusieurs agents. La première en termes de relations d'accessibilité et la deuxième en termes d'opérateurs modaux.

Dans le calcul des situations, l'égalité joue un rôle important. Une autre contribution est celle de l'introduction d'un démonstrateur de théorèmes pour des théories avec égalité appelé SOLE-résolution qui est une extension de la SOL-résolution. Le résultat le plus important dans ce cas est la démonstration de la complétude de la SOLE-résolution. Finalement deux versions d'implémentations des théories exprimées à l'aide d'opérateurs modaux sont présentées. La première accepte seulement des théories initiales complètes dans lesquelles on fait l'hypothèse du monde fermé. La seconde utilise la SOLE-résolution pour laquelle on accepte aussi des théories initiales incomplètes.

Mots clés : Intelligence Artificielle - logique - systèmes multiagents - croyance - calcul des situations - frame problem - théories avec égalité - déduction automatique - SOL-résolution.

LOGICAL MODELLING OF COMPLEX SYSTEM MADE OF PARTIALLY AUTONOMOUS AGENTS

Abstract: The aim of this thesis is complex system formalisation. The main interest focuses on the modelling of two aspects of systems: its dynamic property and its decomposition in several active entities (agents). Modelling of this type of systems is based on human being behaviour, and the representation of notions such that action, belief, intention, capacity, plan generation, etc. is required. After an intuitive introduction to these notions we present the situation calculus which is the framework for the theories we have proposed.

In standard situation calculus only one agent is considered. A significant contribution of our work is the introduction of two situation calculus extensions to represent the evolution of several agents. The first extension is in terms of accessibility relation, the second extension is in terms of modal operators.

In situation calculus equality plays a significant role. Another contribution is the introduction of a theorem prove for theories with equality called SOLE-resolution, which is an extension of SOL-resolution. In that case the most significant result is the proof of SOLE-resolution completeness. Finally, two versions of the implementation of theories expressed with modal operators are presented. The first one only accepts complete initial theories with closed world assumption. The second one makes use of SOLE-resolution and accepts incomplete initial theories.

Keywords: Artificial Intelligence - logics - multi agent system - belief - situation calculus -frame problem - theories with equality - automated deduction - SOL-resolution.

Table des matières

Introduction

Le développement scientifique de ces dernières années, notamment dans le domaine de la micro-électronique et de l'informatique permettent d'envisager de plus en plus l'automatisation d'une vaste gamme d'applications. Le problème qui se pose dans la construction d'un système est le manque de modèle général qui prenne en compte tous les concepts issus d'une application.

Depuis une dizaine d'années, une grande quantité de recherches se sont intéressées à modéliser les applications en prenant la notion d'agent comme notion de base pour décrire un système. Leur objectif, envisageable peut-être à long terme, est la construction de systèmes qui puissent agir avec du sens commun tel que le fait l'être humain. C'est-à-dire posséder une base d'informations éventuellement incomplètes, observer son environnement, raisonner à partir de l'information de la base et des observations, prendre des décisions et agir pour modifier l'environnement selon ses buts.

La modélisation de systèmes complexes (mondes en constante évolution) constitués d'un ensemble d'agents consiste souvent à abstraire les propriétés significatives du système et à les exprimer de façon formelle. On rencontre de nombreuses applications dont l'analyse peut être décomposée de cette manière, notamment dans le commerce électronique, la gestion du trafic aérien, la résolution de problèmes par une communauté de robots en interaction, etc.

En général, les systèmes considérés sont constitués par deux types d'entités : celles qui peuvent agir pour modifier l'environnement (les agents) et celles qui sont passives (les objets). On associe un ensemble de caractéristiques à chaque entité et à chaque agent on associe un nombre fini d'actions. Les caractéristiques des entités peuvent être changées seulement à l'issue de l'action réalisée par un agent. Par exemple, dans un monde constitué d'une personne (agent) et d'une porte (objet), la propriété de la porte d'être ouverte est changée par l'action de fermer la porte, qui peut être effectuée par la personne. Désormais pour simplifier, nous considérons les objets comme un type spécial d'agents dont le nombre d'actions associé est égal à zéro.

L'autonomie est un concept important pour caractériser les agents. Si un agent ne possède pas d'autonomie, une relation de dépendance est créée entre cet agent et d'autres agents. Cette dépendance peut concerner l'aspect externe, c'est-à-dire la capacité de réaliser certaines actions. Par exemple, un robot qui ne possède pas de capteur dépend d'autres agents pour percevoir la présence d'un obstacle. La dépendance peut concerner aussi l'aspect interne, c'est-à-dire la capacité de prendre des décisions. Par exemple, pour un robot qui avance automatiquement c'est le pilote qui prend la décision de l'arrêter. Dans

ces conditions notre objectif est de *modéliser des systèmes complexes composés d'un ensemble d'agents avec un certain degré d'autonomie.* Les agents peuvent être des agents humains, des agents artificiels ou ce qui est plus intéressant, une combinaison d'agents humains et d'agents artificiels.

Dans la construction de ce type de systèmes une étape importante est celle de l'identification des concepts décrivant le problème à résoudre. On étudie les propriétés qu'ils doivent satisfaire et on propose une définition formelle des concepts.

Le concept identifié comme la clef de la dynamique est celui d'interaction qui est défini en termes d'actions et d'agents. Dans une interaction il y a toujours un agent qui réalise l'action et un agent, éventuellement lui-même qui subit les effets de l'action. Une analyse plus fine permet de déterminer les nombreux concepts que l'interaction fait intervenir : action, croyance, connaissance, intention, capacité, opportunité, temps, préférence, obligation, lois de l'environnement, attitudes sociales, confiance, réputation, négociation, etc.

Afin de modéliser ces concepts, acceptés par une large communauté comme descripteurs des systèmes dynamiques, de nombreux chercheurs utilisent la logique mathématique. C'est dans cette discipline que se situent les travaux de recherche que nous présentons. La finalité de ces travaux peut donc se résumer en la définition d'un système formel basé sur la logique, qui permette de modéliser à la fois des agents et leurs interactions.

L'une des caractéristiques demandées au système est la flexibilité. En effet dans un système réel il y a des entités qui peuvent changer l'état du système et qui ne sont pas connues au moment de la conception du système artificiel. Dans l'exemple de la porte et de la personne, il pourrait exister une troisième entité non inclue dans le modèle initial, qui peut fermer la porte, par exemple, le vent.

Si cette entité est repérée, on demande que son introduction puisse se faire de manière très naturelle et sans (ou presque pas) altérer le modèle déjà construit pour les autres agents. Cette flexibilité est exigée aussi dans la mesure du possible pour l'introduction de nouvelles capacités (actions) autorisées aux agents.

Un autre objectif est d'automatiser le raisonnement fait sur le modèle et de réaliser un programme qui permette de tester divers scénarios de systèmes complexes.

L'intérêt de la modélisation en logique des concepts caractérisant les systèmes concerne d'abord l'unification du langage. La logique propose une grande variété de définitions de concepts abstraits tels que l'intention et l'obligation, dont les définitions formelles peuvent, dans la plupart des cas, être justifiées de façon intuitive. Ces concepts sont difficiles à accepter universellement à la différence d'autres concepts concrets tels que la position d'un objet ou la vitesse d'une voiture. Donc il est important de proposer des définitions en logique mathématique qui garantissent l'homogénéité de la terminologie utilisée par les différents concepteurs de systèmes, même si ces définitions ne prétendent pas avoir de valeur intrinsèque ni une caractérisation fermée.

Un deuxième intérêt présenté par la logique est l'importance que d'autres disciplines lui attachent. C'est le cas, par exemple de la philosophie analytique

qui depuis une cinquantaine d'années propose de nombreux travaux concernant l'analyse des concepts qui sont exprimés en logique mathématique. Donc ce type de modélisation profite des travaux faits par les philosophes qui essaient de caractériser les lois du comportement humain, et qui proposent des définitions de concepts tels que l'action, la croyance, l'obligation etc. Il existe une vaste littérature consacrée à la philosophie de la logique parmi laquelle on peut citer : Acta Philosophica Fennica, Journal of Applied Non-Classical Logics, Journal of Logic and Computation, Journal of Logic Language and Computation, Journal of Symbolic Logic, Journal of Philosophical Logic, Logique et Analyse, Mind, Nordic Journal of Philosophical Logic, Notre Dame Journal of Formal Logic, Philosophical Studies, Studia Logica, Synthese, Theoria.

Un autre argument pour utiliser la logique est le fait que les raisonnements sur le modèle exprimé en logique puissent dans de nombreux cas être automatisé. On peut donc analyser les propriétés d'un système en utilisant des outils de démonstration automatique.

Le projet d'avoir un cadre en logique qui englobe l'ensemble des concepts caractérisant les interactions est très ambitieux. A présent on se contente de formalisations détaillées de certains concepts isolés, tels que: le temps [Pri67, Tho84], la croyance [Hin63, Che88, FH88, Gär88, Lev84, Kon86, Dem99], l'action [SC96, Har84, Hil97, Seg89, Seg92, Rei99], la capacité [Bro88, HB95, vvM98], l'intention [CL90a, CL90b, Colss, GK96, CMM90, DMWK96, Jon90, Sad92], et l'obligation [BC96, Me99, DH00], mais les cadres formels réunissant plusieurs concepts sont bien moins explorés [Sin94, DvL96, SRG98, Woo92, WJ95, vL96], en général leur définition présuppose de fortes limitations.

En fonction du problème auquel on s'intéresse on peut ignorer certains concepts, mais il y a des notions qui apparaissent quelque soit le problème. C'est le cas de la notion d'action. Un système qui n'autorise pas d'action est constitué seulement par des objets, donc il devient statique. Les caractéristiques des objets sont inchangées et ce type de cadre est déjà très étudié.

Nous considérons deux types d'actions : les actions modifiant l'état physique des agents ou de l'environnement, par exemple, modifier la position d'un robot. Et les actions modifiant les croyances des agents, par exemple, informer un robot de la position d'un obstacle.

L'étude des actions soulève de nombreux problèmes, un des plus difficiles à résoudre est celui du "frame problem" qui permet de caractériser l'évolution du monde. Le Cognitive Robotic Group de l'Université de Toronto a proposé une solution très intéressante à ce problème, aussi bien au niveau formel qu'au niveau implémentation. La solution est basée sur le calcul des situations qui est une logique classique du premier ordre avec égalité (avec quelques axiomes exprimés dans la logique du second ordre).

Ce groupe a proposé également une extension pour raisonner sur l'évolution des croyances. Mais cette extension n'a pas été implémentée. Puisque notre but est d'aboutir à une implémentation, dans [DP00] nous avons proposé une autre extension basée sur l'idée originale de la résolution proposée pour le frame problem pour l'évolution du monde physique, et qui a été étendue aux cas de l'évolution du monde des croyances. Cette extension ne permet pas d'imbrications des croyances mais en contre partie elle a été partiellement implémentée.

L'implémentation permet de déduire, à partir de la description d'une situation initiale, les propriétés qui sont crues par les agents après avoir réalisé une séquence d'actions.

La validité des résultats produits par l'implémentation, si l'on donne une description complète de la situation initiale, est assurée. Dans le cas d'une description incomplète nous avons analysé la démarche à suivre, et même si nous n'avons pas abouti à une implémentation finale, par manque de temps, nous avons résolu quelques problèmes aussi bien au niveau de la démonstration automatique qu'au niveau de l'implémentation partielle.

Notre préoccupation, portant aussi bien sur l'aspect théorique que sur celui de l'implémentation, nous a conduit à diviser notre travail en deux parties. Dans la première partie on s'intéresse à la modélisation en logique et dans la deuxième on considère l'aspect démonstration automatique.

Nous n'avons pas abordé la formalisation de "tous" les concepts. Notre recherche a été dirigée vers la formalisation de la notion d'action ainsi que celle de la croyance. Cependant avant de formaliser ces notions, il nous a paru utile d'analyser de façon informelle l'ensemble des concepts qui interviennent directement ou indirectement dans la modélisation des systèmes complexes. C'est le but du premier chapitre qui ne suppose pas de connaissances approfondies sur le sujet de la part du lecteur. Aussi dans la dernière partie du chapitre nous décrivons de façon brève et informelle quelques modélisations en logique significatives dans le domaine.

Dans le second chapitre nous faisons un rappel du calcul des situations qui propose une représentation de l'évolution du monde physique.

Dans le troisième chapitre, le calcul des situations est étendu aux croyances. Nous faisons un rappel théorique des extensions proposées par l'équipe de Toronto concernant des systèmes mono agent. Nous présentons aussi notre contribution qui a été d'élargir le cadre formel aux systèmes multiagents. Nous proposons deux façons originales de résoudre les problèmes rencontrés pour le cas de plusieurs agents.

Le chapitre quatre, le premier de la partie concernant la démonstration automatique, présente la réalisation d'un démonstrateur général qui prend en compte l'égalité. Dans toutes les applications réelles l'utilisation de l'égalité joue un rôle important, elle permet de représenter des conditions arithmétiques, l'unicité et les axiomes de complétion. On peut utiliser la règle de paramodulation pour faire des inférences avec l'égalité mais l'inconvénient est que le nombre d'inférences inutiles est relativement grand. Une contribution originale est d'avoir proposé une extension de la SOL-résolution qui considère les théories avec l'égalité. Ainsi nous introduisons la SOLE-résolution qui est une procédure efficace pour la déduction dans des théories avec l'égalité.

Parmi les nombreuses implémentations de démonstration automatique la SOL-résolution a l'avantage d'une part d'être efficace et d'autre part de permettre de caractériser les conséquences générées. C'est-à-dire qu'elle permet de générer des conséquences qui satisfont une certaine propriété. Dans l'implémentation partielle que nous proposons cette propriété permet d'isoler les conditions arithmétiques.

Un des résultats les plus significatifs de ce travail est le fait d'avoir démontré

la complétude et la validité de la SOLE-résolution.

Le but du chapitre cinq est de montrer l'implémentation qui a été faite en Prolog du formalisme défini pour le cas des théories complètes ainsi que de garantir que cette implémentation est correcte. L'implémentation que nous proposons est basée sur une implémentation existante réalisée par le groupe de Toronto. Pour les théories incomplètes nous avons implémenté cette résolution originale qu'est la SOLE-résolution. Mais à cause du manque de temps l'implémentation reste incomplète, au même titre que les théories qu'elle doit traiter ! Cependant nous présentons les idées que nous avons développées pour aboutir à une implémentation.

Finalement nous présentons les conclusions et une brève description de trois problèmes que nous laissons ouverts.

Dans les annexes nous faisons des rappels de la logique classique et modale y compris celle du premier ordre. Notre objectif ici est d'avoir une représentation homogène des concepts qui sont exprimés avec des notations variées dans la littérature. Le lecteur peut ainsi avoir une référence homogène de la définition des différentes logiques.

Les lecteurs familiers du domaine pourront trouver inutiles quelques démonstrations. Notre objectif est aussi d'attirer l'attention de ceux qui ont peu de connaissances en logique et de leur offrir quelques références aussi bien formelles que pratiques pour la compréhension de ce domaine.

Partie I

Formalisation en logique

Chapitre 1

Logiques des Systèmes multiagents

On vit dans un monde complexe, nous-mêmes, nous sommes des êtres complexes. Des molécules de toutes les variétés se rejoignent dans le métabolisme pour faire des cellules. Des cellules interagissent avec d'autres cellules pour former l'organisme. Des organismes interagissent pour former des systèmes, des économies, des sociétés. Les lois qui régissent cette complexité ne sont pas totalement connues, cependant l'interaction entre les éléments est la source du chaos et de la complexité. Les effets des interactions peuvent apparaître totalement aléatoires bien qu'ils soient produits par des règles déterministes. Ils peuvent aussi générer une organisation naturelle et spontanée, qui est la source d'un certain "ordre". Atlan définit la complexité comme "un ordre dont on ignore le code".

La construction de systèmes artificiels fondés sur des agents autonomes vise à abstraire des propriétés formelles analogues aux systèmes naturels complexes. Les applications sont variées : robotique, théorie de la décision, contrôle, simulation, automatisation de fabrication, animation, langage naturel, langage de programmation, gestion du trafic aérien, réalité virtuelle, bases de données, etc.

Pendant les trois derniers siècles, la science a pris un chemin réductionniste pour analyser les systèmes complexes. Ceux-ci sont divisés en plusieurs systèmes "simples". Ces systèmes simples, s'il le faut, sont divisés en sous-systèmes et ainsi de suite. Ce processus de réduction a eu un grand succès.

Pourtant cette méthode, en général, pose des problèmes pour prendre en compte l'ensemble global. Par exemple : comment utiliser l'information des parties pour construire une théorie commune ? La difficulté est due au fait que le système complet peut manifester des propriétés qui ne sont pas explicables par la compréhension séparée des parties.

Une approche analytique n'est pas exclue, pourtant ce genre d'approche est souvent difficile à mettre en œuvre. Avec les développements en microélectronique et l'exploration synthétique des ressources informatiques, les approches réductionnistes paraissent alors plus appropriées. Nous reprendrons cette méthode d'analyse.

Au cours de ce chapitre, une approche réductionniste, celle des systèmes

multiagents, est présentée comme un outil pour la construction de systèmes artificiels complexes. Nous présentons aussi quelques formalismes exprimés en logique, qui permettent de représenter, raisonner et valider des propriétés des systèmes multiagents.

1.1 Systèmes multiagents

Les systèmes réels sont ouverts et leur analyse est limitée. Ce sont des systèmes qui sont trop vastes à caractériser ou qui présentent un comportement imprédictible. Quel que soit leur modèle, de nouvelles informations causent des résultats non attendus. Cependant, l'approche des systèmes multiagents (SMA)[1] est de plus en plus acceptée pour la modélisation des systèmes réels. Ceci est illustré aussi bien par le nombre croissant d'articles parus dans les revues, que par les nombreuses communications sur ce thème dans les Congrès tels que: International Conference on Multi-Agent Systems (ICMAS) et International Conference on Autonomous Agents (ICAA).

Cette sorte de modélisation consiste à regarder un système complexe comme un ensemble d'agents hétérogènes partiellement autonomes. Chaque agent joue un rôle particulier et interagit avec les autres dans un environnement pour résoudre un problème global ou pour surveiller un processus.

Les concepts clefs dans cette approche sont: agent (identifie une entité *individuelle* composante du système) et interaction (identifie la mise en relation *collective* des agents à travers des actions).

L'approche repose sur la représentation des interactions entre les agents et des effets de ces interactions sur les agents et leur environnement.

Cette nouvelle approche, malgré son côté prometteur, a soulevé des problèmes technologiques et scientifiques ayant rapport avec les langages et protocoles d'interaction, avec la représentation des croyances des agents par les autres agents, avec la prise de décisions, avec les informations incomplètes et inconsistantes, avec la négociation entre les agents en conflit, et d'autres problèmes peu connus.

Avant d'analyser quelques formalismes en logique concernant des notions utilisées dans cette approche, il nous paraît important de décrire de façon informelle chacune des notions et leur intérêt dans les SMA. Il est difficile de diviser les SMA en deux parties (l'*agent* et l'*interaction*) car l'interrelation qui existe entre les deux est subtile. Ainsi nous définirons ces deux notions et celles qui s'ensuivent telles que: action, croyance, obligation, etc.

1.1.1 Agent

Une définition standard de cette notion n'existe pas. Ici, nous considérons un agent comme une entité qui a la capacité de réaliser certaines actions, qui possède certaines croyances à propos du monde, qui peut raisonner sur les informations que lui ou d'autres agents possèdent, pour prendre des décisions et

[1] Pour simplifier la lecture, nous utilisons les termes: systèmes complexes, systèmes multiagents, systèmes dynamiques, mondes dynamiques, mondes, comme étant des synonymes.

agir pour transformer le monde. Une personne ou un agent artificiel hautement sophistiqué (systèmes, robots) ont ces caractéristiques.

Pour définir une telle entité, il faut considérer le fait que le contrôle dans un SMA est décentralisé, i.e. chaque agent possède un "morceau" du contrôle. Dans ces conditions on peut dire que les agents possèdent la caractéristique d'être *autonomes*.

Formaliser l'autonomie d'un agent attire l'attention sur les *actions* qu'il peut effectuer, c'est-à-dire l'autonomie est en rapport avec l'ensemble des actions que l'agent est capable de réaliser. L'autonomie peut être vue aussi par rapport aux *prises de décision* que l'agent peut effectuer, i.e. l'amplitude du choix des actions et des agents qui les réaliseront.

De plus, le degré d'autonomie est en rapport avec une propriété très importante des systèmes dynamiques, l'*adaptation* de leurs composants (dans ce cas, les agents) aux changements d'état de l'environnement ou d'autres composants. Par exemple, le degré d'autonomie de l'être humain à la naissance est insuffisant pour s'adapter à son nouvel environnement. S'il n'est pas assisté par d'autres agents (ses parents, par exemple), la possibilité de subsister est faible. Au fur et à mesure qu'il apprend, il devient autonome et son adaptation aux changements d'environnement ne dépend plus des autres agents.

Malgré la similitude de sens, on utilise les notions d'autonomie et d'indépendance d'une façon différente. On suppose qu'un agent dans un SMA n'a pas d'indépendance car il est toujours soumis aux effets des actions des autres agents ; par contre on suppose que l'agent est autonome dans la mesure où 'il décide de faire telle ou telle action. Donc pour modéliser un SMA constitué d'agents partiellement autonomes, on suppose que les agents composant le système ont la capacité de décider, sous certaines conditions, les actions à effectuer.

Le niveau d'abstraction est un paramètre à considérer dans la définition d'un agent. Par exemple, dans l'automatisation d'une usine, un agent peut être un capteur, un robot qui possède le capteur, le système de transport qui possède le robot, le système général qui possède le système de transport, etc. Le choix est fait en fonction du niveau de détail auquel on s'intéresse.

Les agents peuvent jouer des rôles différents. Pour chaque rôle l'agent possède certaines caractéristiques normatives, telles que : des obligations, des permissions, des interdictions, etc. Par exemple, dans une équipe de football il y a des personnes qui jouent le rôle de gardien de but, d'attaquant, de défenseur. Dépendant du rôle assigné, le joueur doit respecter certaines règles. Mais la notion de rôle est indépendante du joueur, ainsi par exemple, le gardien peut changer de rôle ou un défenseur peut devenir gardien.

Nous ne considérons pas la dégradation naturelle subie par les agents à travers le temps, ni le processus inverse, la gradation. Par exemple, pour le corps humain : vieillir et grandir, respectivement.

Ce domaine est couvert par une vaste littérature, mais nous ne considérons pas ici "tous" les concepts associés aux agents. Nous nous concentrons sur certains concepts déterminés tels que l'action, la croyance, l'obligation.

1.1.2 Interaction

Pour considérer un agent comme partie du système, il faut qu'il interagisse avec les autres agents constituant le système, sinon il n'y a pas d'intérêt à le prendre en compte comme élément de l'ensemble.

Dans une interaction, l'agent interprète les actions des autres agents et agit de telle façon que lui-même puisse interpréter ses propres actions.

L'interaction est un concept complexe, sa définition englobe des concepts perceptibles tels que les agents et les *actions* réalisées par ceux-ci, et des concepts abstraits (communément connus comme les états mentaux des agents) tels que les *croyances, connaissances, intentions, désirs* et *obligations* attribués aux agents.

Dans une interaction il y a un but à accomplir (éventuellement partiel), un agent qui réalise une action déterminée (soit parce qu'il a réfléchi et croit qu'en faisant cette action le but pourrait être accompli, soit parce qu'il a reçu un ordre) et les effets de cette action sont subis, soit par un agent (éventuellement lui-même), soit par l'environnement.

On fait la différence entre un agent réactif et un agent cognitif, et en général on fait la différence entre un système réactif ou cognitif. Par exemple la robotique classique s'occupe des déplacements et des perceptions des agents à la différence de la robotique cognitive qui étudie les processus de raisonnement et de prise de décision de tels agents, et qui a pour but de formaliser le comportement d'agents autonomes.

De cette manière on peut dire qu'un système (société, individu) est intelligent si on peut lui attribuer des concepts cognitifs, tels que: croyances, connaissances, intentions, désirs, qui lui permettent d'atteindre un but. Par exemple, l'un des buts de l'être humain est de survivre et si possible avec commodité, paix, amusement, etc. Le but d'une usine de chaussures est de créer des chaussures et si possible de bonne qualité, jolies, confortables, etc. Le but d'une équipe de robcup est de gagner le championnat et si possible avec beaucoup de buts, sans commettre de fautes, en donnant envie de regarder le match, etc.

Pour atteindre le but on se place dans un monde qui peut évoluer, ce qui est vrai peut devenir faux, et vice versa. On suppose que le but est faux au départ et l'objectif est de le rendre vrai.

En effet, le comportement d'un système réel est dynamique et ceci est dû aux interactions entre les composants. La représentation d'un tel système conduit à la représentation du temps, qui est défini en termes des changements qui ont lieu dans le système. Ces changements sont généralement causés par l'exécution d'une action, laquelle affecte des relations, des propriétés ou bien des fonctions. L'information, elle aussi, change et on peut exprimer ses changements par des règles.

1.1.3 Action

La notion d'action est fortement liée à la notion de *changement.* Il est intéressant de noter, par exemple que les actions telles que percevoir ou communiquer ont pour effet de modifier les croyances des agents (changement des

croyances ou, plus précisément, changement dans le monde des croyances). Les actions physiques modifient l'état matériel des agents ou de l'environnement (changement dans le monde réel). Les actions telles que proclamer, ordonner ou interdire modifient les obligations (changement dans les mondes idéaux).

La formalisation de la signification de *l'exécution d'une action* soulève un problème plus général, celui de représenter l'état du monde après chaque exécution. Dans la section suivante nous décrivons plus en détail ce problème appelé *frame problem*[2].

La façon la plus classique de représenter les actions est de décrire leurs préconditions et leurs effets. Sous certaines conditions, il existe une solution au frame problem basée sur le calcul des situations, qui possède des propriétés souhaitables (cf. Chapitre 2).

L'action elle-même est un concept difficile à caractériser dans l'approche multiagents. Il ne faut pas oublier que c'est elle qui est la source du changement. Dans ces conditions elle doit être introduite forcément dans la modélisation des autres notions telles que les croyances (qu'est ce qui fait changer les croyances d'un agent? Les actions telles que communiquer, observer, vérifier, sentir, etc.), les intentions (qu'est ce qui fait changer les intentions d'un agent? Les actions telles que communiquer, observer, vérifier, sentir, réfléchir, etc.), les obligations (qu'est ce qui fait changer la loi qui doit être respectée par l'agent, les actions telles que proclamer, interdire, etc.).

Bref, dans une approche multiagents on a des états du monde qui décrivent soit une situation réelle (monde réel), soit une situation hypothétique (monde des croyances des agents), soit une situation idéale (mondes idéaux dans le contexte juridique), et l'action est une description d'un processus dont l'exécution cause le changement d'un état du monde vers un autre état.

1.1.4 Croyance / connaissance

Les croyances et les connaissances sont des termes, qui sont en général compris de façon intuitive, mais qui sont compliqués à caractériser, décrire et analyser.

Les expressions de la forme $B_i p$ dans un langage L signifient 'i croit que p', 'i sait que p', 'i a l'opinion p', 'i est certain que p'. Pour notre approche nous regroupons ces interprétations comme 'i accepte p comme vraie'. Il y a des auteurs qui font la différence entre croire et connaître, et qui utilisent des opérateurs différents pour chaque notion, ainsi $B_i p$ exprime 'i croit que p' et $K_i p$ exprime 'i sait que p'.

Les connaissances sont des informations que l'agent a sur le monde et qui sont "vraies[3] ". Formellement cela signifie que l'agent connaît des formules qui sont vraies. Les croyances sont des informations que l'agent a sur le monde et qui

[2] En français ce problème est connu comme le problème de la rémanence, de la persistance, du cadre ou du décor. Nous ne voulons pas utiliser une traduction littérale du terme, alors nous avons décidé de garder le nom en anglais.

[3] Vraies du point de vue du modélisateur car il ne faut pas oublier qu'un SMA n'est qu'une représentation d'un système réel et que cette représentation peut contenir des informations ne correspondant pas à la réalité.

peuvent être erronées. Formellement cela signifie que l'agent croit des formules qui sont vraies ou fausses. L'agent peut avoir des introspections positives et négatives, l'introspection positive signifie que l'agent sait que lui-même connaît une chose, quand il connaît cette chose, et l'introspection négative signifie que l'agent sait que lui-même ne connaît pas une chose, quand il ne connaît pas cette chose.

Au sens large, parler des croyances ou des connaissances revient à parler de toutes les attitudes mentales (intention, choix, préférences, désirs, buts, obligations) [Lon99], puisque l'agent doit être capable de signaler quels sont ses buts, intentions, choix, obligations, etc., et que cette capacité est due à la conscience de l'existence de telles attitudes. C'est-à-dire que l'agent croit (ou sait) qu'il les a. C'est ce qui est décrit par les axiomes suivants :

$$\begin{aligned} B_i B_i p &\leftrightarrow B_i p \\ B_i Goal_i p &\leftrightarrow Goal_i p \\ B_i Intend_i p &\leftrightarrow Intend_i p \\ &\vdots \end{aligned}$$

où la signification intuitive de $Goal_i p$ est 'i a comme but p' et celle de $Intend_i p$ est 'i a comme intention p'.

Dans [WJ95] les auteurs utilisent le terme "attitude d'information" pour désigner les croyances et les connaissances et le reste est appelé "pro-attitudes". Ces dernières sont celles qui guident les actions des agents. Parmi celles-ci l'intention joue un rôle important pour la prise de décision.

1.1.5 Intention

L'intention est une notion importante pour les SMA car elle permet de lier les croyances avec les buts et les engagements. La plupart des théories sur l'intention sont issues des travaux du philosophe Bratman [Bra87], repris plus tard par Cohen et Levesque [CL90b] qui ont donné la base des théories formelles du comportement intentionnel.

Bratman dans [Bra87] dit que les philosophes essaient fréquemment de définir l'intention à partir des croyances et des désirs (buts à atteindre). Il fait remarquer que le comportement rationnel ne peut pas être analysé uniquement en termes de croyances et de désirs, et qu'il est nécessaire d'introduire un troisième état mental, l'intention.

Bratman dans [Bra84] signale qu'avoir une intention (vouloir faire quelque chose) et faire quelque chose intentionnellement ne partagent pas la même notion d'intention. Et que c'est plutôt la première qui concerne les plans d'un agent. Cohen et Levesque sont d'accord avec Bratman à ce propos et proposent un modèle.

Le modèle de l'intention de Cohen et Levesque est inspiré du travail de Bratman, le modèle donne la signification d'engagement aux intentions des agents. Ce modèle est formalisé par une logique modale du premier ordre temporel, qui a un opérateur modal qui satisfait des propriétés souhaitables pour un modèle de l'intention : les agents ont une tolérance limitée à la frustration.

Ainsi, même si les buts peuvent être flexibles, les agents ne les abandonnent pas assez facilement ; autrement dit, les auteurs décrivent comment l'agent est engagé vers ses buts. Ils modélisent l'intention avec des buts persistants qui sont basés sur quatre opérateurs primitifs, BEL (croyance), GOAL (but), HAPPENS (l'événement prochain) et DONE (l'événement qui vient d'avoir lieu). Cohen et Levesque analysent l'intention d'une façon différente des autres états mentaux. Par exemple, ils font remarquer que les connaissances et croyances contiennent des propositions tandis que le contenu d'une intention est généralement une action.

Les attitudes propositionnelles peuvent être analysées toutes seules tandis que des attitudes comme l'intention sont en relation avec d'autres attitudes mentales telles que les croyances, ainsi qu'avec d'autres concepts tels que le temps et l'action. L'approche explore d'abord les conséquences d'un engagement qui est persistant jusqu'à que l'on croie que son but est accompli ou bien que l'on croie qu'il est infaisable. Basés sur ces fondements une théorie de l'interaction rationnelle et de la communication a été construite [CL90b].

L'intention suppose une analyse raisonnée de la situation présente et des situations futures possibles après le choix d'une action. Les agents sont capables de prévoir les conséquences de leurs choix, et plus précisément leur décision dépend de la relation de causalité existant entre les événements. Autrement dit ils prévoient le futur. Le fait que l'agent base son comportement sur cette capacité d'anticipation du futur permet de dire que l'agent "intentionnel" est nécessairement cognitif, c'est-à-dire qu'il a un modèle du monde sur lequel il est capable de spéculer pour évaluer les conséquences de ses actes et choisir l'action à tenter. D'ailleurs cette prise de décision est supposée "rationnelle", il choisit la meilleure solution en fonction des buts et des façons possibles de l'atteindre.

1.1.6 Aspects sociaux

On considère que toutes les notions, que nous avons décrites, caractérisent un agent comme une entité autonome mais non isolée. Effectivement, ces notions caractérisent la capacité rationnelle de l'agent vis-à-vis de la collectivité. D'une façon naïve, on peut dire que l'on a décrit les notions internes (subjectives) d'un agent dans une société. Nous présentons ici, de façon brève, quelques aspects externes (sociaux) qui jouent un rôle important pour l'itération des agents.

On a vu qu'un SMA est constitué par un groupe d'agents, qui sont, d'une façon ou d'une autre, restreints mutuellement par leurs interactions. En général, les contraintes sont imposées pour chaque agent par rapport au rôle qu'il joue dans le groupe. Donc, en principe, leur comportement et leurs interactions dépendent du rôle imposé. Dans le groupe, les agents coopèrent et s'engagent avec d'autres agents pour atteindre leurs buts. Il y a des travaux qui se concentrent sur la formalisation de la résolution coopérative des problèmes [WJ97]. Parfois, chaque membre du groupe est contraint à avoir le même but commun, et le groupe constitue ainsi une équipe. Quand les agents n'ont pas le même but, peuvent surgir des conflits, et en ces circonstances il faut caractériser la façon de résoudre le conflit. Ainsi des notions telles que la négociation apparais-

sent. Il faut remarquer que ce type de conflit est d'une nature différente (sociale ou externe) de ceux introduits au moment de la prise de décision d'un agent (individuel ou subjectif), i.e. les conflits subjectifs doivent tenir compte, d'un côté de l'intérêt particulier, et des exigences normatives sociales d'un autre. Il y des notions qui permettent de caractériser cette prise de décision, telles que le choix que l'agent fait pour l'adoption des buts et la sélection des plans dirigés vers l'atteinte de ceux-ci.

Croyances mutuelles

L'un des premiers concepts qui permet de passer d'un simple agent à une société d'agents est celui de *croyance mutuelle*. On dit qu'un groupe d'agents a une croyance mutuelle p si (1) chaque agent croit p, (2) chaque agent croit que chaque agent croit p, (3) chaque agent croit que chaque agent croit que chaque agent croit p, et ainsi de suite. De cette façon la croyance mutuelle caractérise un état mental partagé par la société. Cette notion peut être utilisée pour spécifier des aspects de la communication.

Intention commune

Une autre notion qui est étendue pour prendre en compte la société est celle de l'intention. Dans [LCN90], la notion d'*intention commune* est introduite. La description de la théorie proposée est complexe. On se limite à faire une description approximative. On dit qu'un groupe d'agents partage une intention commune pour atteindre p si (1) chaque agent a comme but p, (2) chaque agent s'engage à satisfaire son but jusqu'à avoir la croyance mutuelle que, soit p a été accompli, soit p ne peut être accompli, et (3) les conditions (1) et (2) sont des croyances mutuelles.

Il y a des auteurs qui ont développé des théories où les agents partagent la création des plans en introduisant la notion de coopération des activités des agents.

Des notions telles que *engagement social*, où un agent s'engage vis-à-vis d'un autre agent, ont été aussi développées [Cas95]. Cette sorte d'engagement est en rapport avec la notion d'obligation (cf. logique déontique, page 36). Dans cette approche, un agent spécial nommé le médiateur est introduit. Il certifie la fin de la tâche engagée. L'engagement social peut être utilisé pour spécifier comment doivent être les interactions des agents. Il entraîne la description des notions associées telles que les *rôles* qui existent au sein du groupe des agents.

Prendre en compte des groupes soulève de nombreux problèmes d'organisation, de détermination des actions individuelles et en collectif, de détermination des croyances, de détermination des motivations des agents en ce qui concerne leur façon d'agir et d'analyser l'entourage social (cela détermine le comportement des agents autonomes cognitifs), de détermination des normes et conventions sociales (les lois), de détermination de comment l'agent répondra aux ordres et nécessités des autres agents et comment il obéira aux lois.

Le problème d'obéissance aux lois est intéressant du point de vue organisationnel, car la spécification d'une structure peut s'imposer à travers les règles

du comportement des agents.

Aspect normatif

Dans les spécifications des systèmes, la distinction entre le comportement normatif (tel qu'*il devrait être*[4]) et celui non normatif (tel qu'*il est*) doit se faire afin de distinguer le comportement illégal (non-normatif) mais néanmoins possible. Le comportement illégal est seulement réglementé par une spécification, bien qu'il soit intéressant de décrire "qu'il devrait avoir lieu" si jamais un tel comportement illégal (mais possible) a lieu. Autrement dit, quelle sanction il faut appliquer quand un agent viole une loi. La logique déontique essaie de caractériser cela en utilisant des opérateurs modaux qui donnent le statut du comportement, i.e. soit il est légal (normatif) ou non. Autrement dit la logique déontique formalise les réglementations (cf. Section 1.3.6).

1.2 Autour de l'action

Comme nous l'avons vu, l'action est le concept clé pour la modélisation des systèmes dynamiques. Nous consacrons cette section à son étude, nous présentons certains problèmes liés aux changements, ainsi que des méthodes en rapport avec ceux-ci, tel que la planification, et finalement nous présentons la révision et la mise-à-jour, deux problèmes liés aux changements des croyances à l'issue d'une action.

Nous utilisons l'exemple suivant afin d'illustrer les problèmes que nous allons présenter.

On suppose que le monde réel que nous voulons représenter[5] est constitué par une table, sur laquelle il y a un verre et une coupe. Les propriétés[6] auxquelles on s'intéresse concernent la position des objets et la question de savoir si les objets sur la table sont, ou non, cassés et quel est leur contenu. Supposons aussi qu'il y a deux agents : Nathalie et son fils.

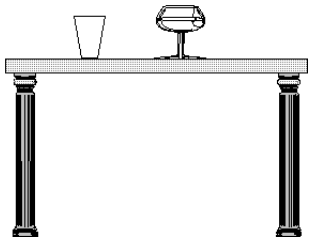

Figure 1.1: Monde réel

Ce cadre peut être exprimé formellement par les prédicats suivants :

$$position(table, X_1, Y_1)$$

[4] De l'anglais should be.

[5] Il faut noter que cette représentation devient une croyance du modélisateur, c'est-à-dire qu'elle peut ne pas correspondre à la réalité mais qu'elle représente la réalité du modélisateur.

[6] Le nombre et le type de propriétés sont fixés pour caractériser le problème à résoudre.

$$position(verre, X_2, Y_2)$$
$$position(coupe, X_3, Y_3)$$
$$\neg cassé(verre)$$
$$\neg cassé(coupe)$$
$$dedans(verre, Z_1)$$
$$dedans(coupe, Z_2)$$

où la signification intuitive de $position(O, X, Y)$ est 'l'objet O a la position (X, Y)', celle de $cassé(O)$ est 'l'objet O est cassé' et celle de $dedans(O, Z)$ est 'l'objet O contient Z'.

1.2.1 Frame Problem

Supposons que nous voulions représenter les faits et changements suivants :

[t_1.] Nathalie enlève la coupe.
[t_2.] Sur la table il n'y a qu'un verre.
[t_3.] Nathalie pose la coupe dans l'évier.
[t_4.] Nathalie prend une tasse de café.
[t_5.] Nathalie met la tasse sur la table.
[t_6.] Sur la table il y a un verre et une tasse.

Il ne suffit pas de décrire l'état suivant en décrivant les résultats de l'exécution de l'action. Il faut aussi être capable de décrire les formules, qui étaient vraies avant l'exécution et le restent à l'état suivant. Par exemple, quand Nathalie prend un verre sur la table, la position de cette table ne change pas, ni celle des autres objets sur la table. L'adresse de Nathalie ne change pas quand Nathalie change de petit ami. Ce qui est vrai avant et n'est pas concerné par l'action exécutée reste vrai après l'exécution.

Nous venons de décrire le frame problem, qui consiste à exprimer dans un langage formel l'idée suivante : après avoir exécuté une action, aucune proposition (fait) ne change sa valeur de vérité à l'exception des propositions qui sont en relation avec les effets de l'action exécutée. Plusieurs solutions ont été étudiées depuis la création de l'Intelligence Artificielle [Bro87], la plupart ont la caractéristique de sortir du cadre de la logique classique du premier ordre et proposent un mécanisme qui permet de déduire les propriétés inchangées.

Le but des solutions au frame problem est de spécifier de façon naturelle les lois du changement. Il y a au moins cinq sous-problèmes qui se posent:

1. Décrire toutes les propriétés inchangées.

2. Spécifier les conditions sous lesquelles les actions sont exécutables.

3. Spécifier l'influence ou les effets des actions.

4. Spécifier éventuellement les conséquences indirectes des actions.

5. Spécifier l'occurrence des événements externes.

McCarthy et Hayes [MH69] ont été les premiers à observer ce problème, ils proposent une solution qui introduit les frame axioms[7]. Ces axiomes décrivent chaque propriété qui n'est pas affectée par une action particulière, par exemple, si $dedans(o, z, s)$ veut dire 'l'objet o contient z dans l'état[8] s', le frame axiom : $\forall s\ (dedans(o, z, s) \rightarrow dedans(o, z, résultat(déplacer(o), s)))$ veut dire 'le contenu de l'objet n'est pas affecté par l'action de déplacer l'objet'. Cela ne suffit pas, il faut aussi écrire le frame axiom indiquant que la propriété d'être cassé n'est pas affectée par la même action : $\forall s\ (cassé(o, s) \rightarrow cassé(o, résultat(déplacer(o), s)))$, où $cassé(o, s)$ signifie 'l'objet o est cassé dans l'état s'. Cette solution est théoriquement correcte mais le fait de gérer un grand nombre de frame axioms rend difficile sa mise en œuvre, parfois même impossible si leur nombre est infini.

En effet, donner la liste exhaustive des axiomes pour toutes les actions et pour toutes propriétés qui ne sont pas affectées par chaque action semble déraisonnable, si on ajoute la complexité qui existe au moment d'introduire une nouvelle action ou propriété (car l'ensemble complet des axiomes doit être réécrit) on constate que cette solution est quasi impossible à mettre en œuvre. L'inefficacité en ce qui concerne la gestion d'un grand nombre de propriétés (faits) est un autre inconvénient de cette solution. En effet, les propriétés ne sont pas transmises d'un état à l'autre et donc pour chaque nouvel état il faut prouver que les propriétés sont vraies ou fausses.

Jusqu'à présent, on a supposé que les actions exécutées par les agents transforment l'état externe du monde. Mais il ne faut pas oublier que les actions de perception, telles que percevoir, sentir ou observer, modifient les états mentaux des agents. Par exemple, exécuter l'action d'observer le contenu de la coupe donne à Nathalie la connaissance du contenu de cette coupe. Tester la distance au mur donne au robot la connaissance de l'éloignement du mur. Il y a des auteurs qui donnent le nom d'*actions qui produisent des connaissances* (knowledge-producing actions) à cette sorte d'actions [SL93].

Alors on vise à introduire des formalismes qui peuvent être étendus pour représenter cette sorte d'actions et leurs effets sur les états mentaux des agents. Ainsi on a besoin d'un mécanisme pour représenter les croyances et aussi pour résoudre le frame problem dans le contexte des croyances.

Il y a d'autres problèmes liés à la représentation des actions et leurs effets sur le monde réel, nous évoquons par la suite la ramification (cf. Section 1.2.2) et la qualification.

1.2.2 Problème de la ramification

Proche du frame problem, le problème de la ramification consiste à essayer de représenter "toutes" les conséquences à l'issue de l'exécution d'une action, y compris les conséquences secondaires. Il est possible de déduire qu'une formule reste vraie lors de l'exécution d'une action en utilisant des axiomes. Mais le problème de la ramification est encore plus compliqué s'il s'agit des changements indirects causés par l'exécution de l'action et qui ne sont pas décrits dans la

[7] De façon similaire au frame problem, nous utilisons ici le terme en anglais.

[8] Les auteurs utilisent le terme *situation* et non état.

formule décrivant le résultat de l'action. Par exemple, si Nathalie pousse la table pleine de verres, la table et les verres posés sur la table ont changé de position et s'il y a d'autres verres sur les verres qui sont sur la table, ces verres sur les verres ont aussi changé de position, etc. Il ne faut pas oublier que la solution proposée à ce problème doit inclure aussi une solution au frame problem. Dans cet exemple, il faut exprimer aussi que les autres objets dans la pièce restent inchangés après avoir poussé la table qui supporte les verres.

Le problème de la ramification peut être vu sous deux angles : le premier concerne la représentation (le nombre de conséquences peut être très grand). Le second concerne la définition des conséquences complexes (elles peuvent être le résultat des relations qui ne sont pas visibles de manière évidente).

1.2.3 Problème de la qualification

Avant l'exécution de toute action un certain nombre de conditions doivent être satisfaites. Ces conditions sont généralement appelées *préconditions* de l'action. Le nombre de préconditions pour chaque action est considérable pour ne pas dire infini. Imaginons toutes les choses qui peuvent empêcher Nathalie de pousser la table pleine de verres : Nathalie peut être trop faible pour réaliser le mouvement, la table peut être très lourde, la table peut être collée au sol, Nathalie peut avoir son fils dans les bras, Nathalie peut tenir une pile d'assiettes, Nathalie peut avoir entre les mains un rhinocéros rose (en peluche), etc. Au niveau pratique, on ne peut pas énoncer toutes les possibilités explicitement.

Le choix des actions se fait sur les préconditions. Il y a des préconditions vraies avant qui restent vraies après et il y a aussi des préconditions qui ne sont plus vraies après. Le problème est de savoir si l'on a explicité "toutes" les préconditions. Les préconditions pourront porter sur l'environnement ou sur les agents.

De façon similaire au problème de la ramification, celui de la qualification peut être vu sous deux angles : la longueur de la liste de préconditions et la complexité de sa définition. Les solutions proposées ne sont pas entièrement satisfaisantes.

1.2.4 Planification

La planification étudie la façon de générer des plans, où un plan est une séquence d'actions dont l'exécution permet de résoudre un problème donné. On peut dire que la planification permet de rechercher une solution totale ou partielle avant l'exécution des actions.

Dans ce contexte on se place dans un monde qui est descriptible, où sa représentation peut aider à trouver des solutions. La vue que l'on a du monde est discrète, chaque intervalle est représenté par un état du monde. Un agent a les moyens (actions) de faire changer (évoluer) le monde. L'état de départ est appelé état initial, le but à atteindre est l'état final. Il faut alors trouver la suite d'actions pour passer de l'état initial à l'état final. Il y a des approches qui considèrent l'état initial complètement connu, ce qui est une hypothèse très

forte. La méthode de planification doit aussi considérer une solution au frame problem à chaque changement d'état.

A titre d'information, dans la programmation, les effets des instructions sont connus tandis que dans la planification, les conséquences d'une action sont complexes à définir étant donné les relations entre les objets manipulés. En programmation, on suppose que la machine marche et que l'exécution d'une instruction est réussie, tandis qu'en planification il est possible d'avoir des échecs des opérations. Parfois aussi il émerge un nouveau problème et l'adaptation du plan doit être considérée.

La plupart des planificateurs utilisés dans la communauté de l'Intelligence Artificielle sont une variante de STRIPS.

Système de planification STRIPS

STRIPS (STanford Research Institute Problem Solver) est un formalisme du début des années 70's [FN71], dont il existe plusieurs versions. Toutes possèdent un ensemble d'opérateurs et leurs descriptions correspondantes, ainsi qu'une structure de données (base de données) qui est une représentation instantanée du monde (état actuel[9] du monde). Les opérateurs représentent les actions. Formellement, ils transforment la base de données. Puisqu'il n'y a qu'une seule représentation du monde, la base est renouvelée à chaque application d'un opérateur. Intuitivement un opérateur permet de saisir les effets de l'action qu'il représente. Quand un opérateur est appliqué à la base de données, l'opérateur propose une nouvelle base de données qui est une représentation hypothétique de comment serait le monde après l'exécution (application) de cette action (opérateur).

Le postulat de STRIPS dit qu'à la suite de l'exécution d'une action (l'application d'un opérateur), ce qui ne change pas explicitement (ce qui n'est pas dans la liste d'ajouts ou de retraits) ne change pas. Il y a une seule représentation du monde, qui est non monotone[10]. On ajoute ce qui devient vrai et on enlève ce qui devient faux. Donc chaque action est décrite en trois parties : les préconditions, la liste d'éléments à ajouter et la liste d'éléments à enlever. Une action ne peut être effectuée que si la base actuelle satisfait les préconditions d'une telle action. Les effets de l'action sont estimés en ajoutant (en effaçant) de la base les éléments de la liste d'ajouts (de retraits). Lorsqu'une action est exécutée, la base actuelle est mise à jour et le même raisonnement est fait sur la nouvelle base obtenue qui devient la base actuelle. Le processus continue jusqu'à trouver une séquence d'actions constituant le plan. Plus on complique les préconditions plus la recherche du plan va être lente.

Cette approche apporte, d'une manière simple, une solution au frame problem mais ne donne pas une solution satisfaisante aux autres problèmes liés à l'action tels que la ramification. STRIPS fonctionne à l'aide des opérateurs et ne

[9] Il ne faut pas oublier qu'il s'agit d'une planification, donc l'état actuel n'est pas un état physique, mais plutôt un état hypothétique : c'est l'état de référence pour continuer la méthode de planification.

[10] 'Non monotone' au sens où une formule qui est vraie à un moment donné peut ne plus l'être plus tard, puis redevenir vraie après l'exécution de quelques actions, et ainsi de suite.

permet pas de déduire des effets secondaires des actions. Envisager de prendre en compte les effets secondaires revient à estimer tous les effets secondaires de chaque action pour chaque contexte possible et les intégrer aux listes des éléments d'ajouts et de retraits.

Formellement, une base de données est un ensemble de formules fermées (cf. Définition 26) avec une syntaxe restreinte, généralement des formules atomiques. Par exemple, supposons maintenant que sur la table il y a quatre verres et que l'on veut savoir s'ils sont sur la table ou sur un autre verre, supposons le cadre suivant :

$$\{surtable(B),\ sur(A,B),\ libre(A),\ surtable(D),\ sur(C,D),\ libre(C)\} \quad (1.1)$$

qui veut dire que les verres B et D sont sur la table, les verres A et C sont sur les verres B et D respectivement et que les verres A et C n'ont pas de verre au-dessus.

On a fait l'hypothèse du monde fermé (cf. Section 5.1.2), c'est-à-dire que tout ce qui n'est pas explicitement décrit est considéré faux. Ici, puisqu'ils ne sont pas donnés explicitement on a implicitement les littéraux :

$$\{\neg surtable(C),\ \neg sur(D,C),\ \ldots\}$$

Chaque opérateur a quatre composants :

(1) Le nom de l'opérateur éventuellement avec des paramètres (des variables).

(2) Les préconditions ci-dessus étiquetées par P et représentées par une formule.

(3) La liste d'ajouts ci-dessus étiquetée par A et représentée par une formule.

(4) La liste de retraits ci-dessus étiquetée par R et représentée par une formule.

La description d'un opérateur doit exprimer la façon de transformer la base de données. Par exemple l'action *déplacer* est décrite comme suit :

$$\begin{aligned} &déplacer(x,y,z)\\ &\quad P : libre(x) \wedge libre(z) \wedge sur(x,y) \wedge \neg(x=y) \wedge \neg(x=z) \wedge \neg(y=z)\\ &\quad A : libre(y) \wedge sur(x,z)\\ &\quad R : libre(z) \wedge sur(x,y) \end{aligned}$$

On peut voir cette définition comme un schéma, dont les occurrences sont celles qui s'appliquent à la base de données. Par exemple l'occurrence $déplacer(A, D, C)$ ne peut pas être appliquée car elle ne satisfait pas les préconditions puisque A n'est pas sur D. Par contre l'occurrence $déplacer(A, B, C)$ satisfait toutes les préconditions et elle peut être appliquée à la base (1.1) donnant comme résultat la base suivante :

$$\{surtable(B),\ sur(A,C),\ libre(A),\ surtable(D),\ sur(C,D),\ libre(B)\}$$

La génération des plans dans le cadre de STRIPS se fait de façon naturelle. On suppose une base de données initiale D_0 et une formule β. Une séquence des occurrences d'opérateurs $a_1, a_2, a_3, \ldots, a_n$ est un plan pour le but β si et seulement s'il existe une séquence de bases de données $D_0, D_2, D_3, \ldots, D_n$ telles que :

(a) toutes les préconditions pour appliquer a_i sont satisfaites dans D_{i-1} $(1 \leq i \leq n)$,

(b) D_i est la base qui résulte de l'application de a_i à D_{i-1} $(1 \leq i \leq n)$,

(c) à partir de D_n on peut déduire β.

Un exemple de définition d'un système STRIPS est la base de données (1.1) et l'opérateur $déplacer(x, y, z)$. Si le but est, par exemple $\exists x\ sur(x, D) \wedge sur(A, x)$, il n'y a qu'un plan $déplacer(A, B, C)$. Maintenant, si l'on ajoute les actions de déplacement d'un verre de la table sur un autre verre et d'un autre verre sur la table, comme suit :

$$
\begin{aligned}
&déplacer_{depuis\_table}(x, y)\\
&\quad P : libre(x) \wedge libre(y) \wedge surtable(x) \wedge \neg(x = y)\\
&\quad A : sur(x, y)\\
&\quad R : libre(y) \wedge surtable(x)
\end{aligned}
$$

$$
\begin{aligned}
&déplacer_{sur\_table}(x, y)\\
&\quad P : libre(x) \wedge sur(x, y) \wedge \neg(x = y)\\
&\quad A : libre(y) \wedge surtable(x)\\
&\quad R : sur(x, y)
\end{aligned}
$$

le nombre de plans que l'on peut trouver est illimité, par exemple :
$déplacer(A, B, C)$, $déplacer_{sur\_table}(A, B)$, $déplacer_{depuis\_table}(A, C)$,
$déplacer_{sur\_table}(A, B)$, $déplacer_{depuis\_table}(A, B)$, $déplacer(A, B, C)$, ...
$déplacer_{sur\_table}(A, B,)$, $déplacer_{sur\_table}(C, D)$, $déplacer_{depuis\_table}(B, D)$,
$déplacer_{depuis\_table}(A, B)$, ...

Il n'existe pas une procédure efficace de planification pour les systèmes STRIPS. En général les bases de données sont générées en appliquant des occurrences d'opérateurs et en vérifiant la satisfaction ou non du but.

Un inconvénient de cette sorte de systèmes est qu'ils n'ont pas une axiomatique en logique, ils n'ont pas non plus d'axiomatique des actions et de leurs effets et la description de l'opérateur n'est pas une formule logique. La solution au frame problem est donnée comme un mécanisme d'ajouts et d'effacements des éléments plutôt que comme un formalisme explicite. Les plans ne sont pas des conséquences des axiomes. D'ailleurs, STRIPS est un planificateur dépendant du domaine d'application. Au niveau implémentation on a le problème du calcul de l'état. Si les ressources sont limitées, on met à jour la même base, sinon on peut utiliser une méthode pour marquer et démarquer les éléments de la base.

Les intentions dans un système de planification ne sont que le contenu du plan. Des telles intentions sont représentées comme des actions possibles pour atteindre le but.

Le raisonnement est difficile quand on utilise des définitions opérationnelles, il semble que la caractérisation déclarative est plus appropriée.

Jusqu'à présent, on a parlé des problèmes posés par les actions et leurs effets modifiant l'état du monde réel, mais les problèmes se posent aussi sur les états mentaux des agents. Par exemple, pour les croyances, on rencontre les problèmes de révision et mise-à-jour.

1.2.5 Révision et mise-à-jour

La prise en compte de l'évolution des croyances des agents amène de nouveaux problèmes. En effet les croyances ont un caractère subjectif, chaque agent a une image du monde réel représentée par ses croyances. Dans ces conditions il faut distinguer deux sortes de changements : les changements des croyances pour faire en sorte que les croyances correspondent à un nouvel état du monde réel (en considérant le monde réel en changement). Ceux-ci sont désignés comme la mise-à-jour des croyances. Et les changements des croyances pour faire en sorte que les croyances "erronées" soient remises en cause (en considérant le monde réel fixé). Ceux-ci sont appelés la révision des croyances. Dans les deux cas le changement des croyances pose le problème de la représentation des croyances inchangées. C'est-à-dire on retrouve le frame problem dans la représentation de l'évolution des croyances.

Considérons le cadre suivant : Nathalie croit que le verre est cassé tandis que son fils croit que celui-ci n'est pas cassé (voir Figure 1.2).

Forcément l'un des deux a une croyance "erronée" en ce qui concerne le verre. Supposons, pour sortir de l'ordinaire, que c'est le fils qui a raison. Une révision des croyances (due par exemple à l'action d'observation) permettra à Nathalie de changer sa croyance par "le verre n'est pas cassé". Maintenant si l'on considère que le monde est sujet à des évolutions correspondant à des actions telles que : faire tomber un verre, réparer un verre, etc. Il faut considérer le changement non seulement dans le monde réel mais aussi éventuellement dans la représentation du monde chez les agents. Ainsi, par exemple, si l'enfant laisse tomber le verre et le casse, une mise-à-jour permettra à l'enfant de changer sa croyance par "le verre est cassé". Etant donné que ce n'est pas Nathalie qui exécute l'action de laisser tomber le verre, la mise-à-jour peut éventuellement ne pas affecter ses croyances.

Dans une théorie logique, la révision permet de valider la nouvelle information par rapport aux croyances précédentes, en particulier par rapport à celles qui sont contradictoires, afin de l'intégrer dans la théorie. Carlos Alchourrón, Peter Gärdenfors et David Makinson [AGM85] ont proposé un ensemble de postulats, connus sous le nom de postulats d'AGM, que tout opérateur de révision doit satisfaire. Hirofumi Katsuno et Alberto Mendelzon montrent que ces postulats caractérisent bien le processus d'intégration d'une nouvelle information au sujet d'un monde statique, mais qu'ils ne sont pas adaptés à un monde en évolution [KM91]. Ils introduisent alors un nouvel ensemble de postulats KM

Figure 1.2: Nathalie a tort en ce qui concerne le verre.

caractérisant la mise-à-jour.

Il y a des approches permettant de conserver l'état précédent. A chaque changement, la base de croyances n'est pas écrasée par une nouvelle base. Le but est de construire la mémoire de l'agent avec différents états de la base qui sont accessibles depuis son état mental actuel [CL90b, Per90]. Ces approches ont pour but de définir le comportement rationnel des agents. Avec des "souvenirs", on a un point de référence pour agir de telle ou telle façon dans le futur en se référant aux événements déjà vécus et gardés dans la mémoire.

Après avoir mis en évidence les problèmes auxquels nous souhaitons proposer des solutions, c'est-à-dire la représentation des notions et le frame problem, nous avons besoin de faire un état de l'art des différentes approches existantes. En particulier, nous nous intéressons aux approches logiques.

1.3 Approches logiques

1.3.1 Introduction

Il existe dans la littérature de nombreux formalismes proposés pour prendre en compte les raisonnements, dits de "sens commun", attribués à l'être humain en société (cf. les sections précédentes qui décrivent quelques notions et problèmes associés). Certains purement symboliques, d'autres basés sur des théories numériques.

Il semble illusoire qu'un formalisme unique puisse être capable de représenter

simultanément les multiples aspects du raisonnement de sens commun. Ce défi a donné naissance à de nombreux travaux, notamment concernant le développement des logiques. Mais chaque logique traite un aspect particulier du raisonnement humain et en conséquence n'offre qu'une solution partielle.

La modélisation de l'interaction pose le problème de la mise en accord sur les concepts. Les agents peuvent avoir des perceptions différentes du même objet, et leurs points de vue sur le même sujet peut varier d'agent à agent. Chacun peut utiliser un lexique différent et les relations et concepts employés ne sont pas nécessairement les mêmes. Une façon d'approcher une solution est d'essayer de réduire et si possible d'éliminer les dissensions entre terminologies et entre définitions conceptuelles. La logique offre un cadre idéal partageant un vocabulaire de référence donné pour tous les domaines d'applications.

Nous avons alors opté pour une approche de ce type malgré son caractère réducteur, car la logique fournit une base méthodologique incomparable pour la formalisation du raisonnement. Le plus grand avantage de l'approche logique, pour modéliser l'Intelligence Artificielle ou le sens commun, est sa capacité à représenter les informations de façon déclarative (au moyen de formules logiques). On obtient ainsi des sémantiques déclaratives claires.

La logique est un système formel qui se compose d'un langage, d'un ensemble de schémas d'axiomes, de règles d'inférence et d'une description de la façon dont ces axiomes et ces règles sont utilisés pour produire des théorèmes. Par ailleurs, une sémantique est associée au système formel, définie au moyen d'une fonction qui assigne une valeur de vérité à toute formule.

Dans cette section nous analysons quelques travaux importants concernant les logiques non-classiques en rapport avec l'approche des systèmes multiagents.

1.3.2 Logiques non-classiques

Malgré l'ambiguïté résultant d'une définition d'une logique non-classique comme la négation de la logique classique (logique des propositions ou des prédicats du premier ordre), nous utilisons, de façon naïve, l'expression "logique non-classique" pour désigner toutes les logiques mentionnées dans ce travail sauf la logique classique.

A cause de sa conception réductionniste, la logique classique porte une limitation. Si le calcul des prédicats est bien adapté aux mathématiques, il permet difficilement de prendre en compte certains aspects du raisonnement de sens commun (dans l'incertain, le temporel, etc.). Ceci conduit à la définition d'un grand nombre de logiques non-classiques, qui veulent décrire de tels aspects du raisonnement. En général, ces logiques sont construites en élargissant le cadre de la logique classique sur trois points :

- En étendant le langage, soit en rajoutant des prédicats particuliers, soit en introduisant des opérateurs. Par exemple, "le possible" et "le nécessaire" de la logique modale ou le "je sais" de la logique épistémique.

- En rajoutant des nouveaux schémas d'axiomes. Par exemple, les schémas d'axiomes de l'égalité de la logique égalitaire.

- En définissant une nouvelle notion de déduction, à laquelle peut correspondre une nouvelle notion de sémantique. Par exemple, l'univers des mondes possibles dans la logique modale. Pour celle-ci, on n'impose pas certaines propriétés des systèmes déductifs classiques, notamment la monotonie de l'inférence. On ne se limite pas à la règle de modus ponens, on autorise d'autres règles d'inférence comme la règle de nécessitation de la logique modale ou les règles d'inférence de la logique des défauts [Rei80].

1.3.3 Logiques modales

Parmi ces logiques non-classiques, les logiques modales sont l'un des meilleurs outils d'analyse pour l'étude de diverses notions, principalement celles de la nécessité, de la possibilité, de l'obligation, de la permission, du futur, du passé, du temps en général, de l'action, du savoir, de la croyance, de l'intention, etc.

La formalisation des logiques modales a été proposée au début du vingtième siècle par Clarence Lewis [Lew18]. Lewis a exploré une variété de logiques, où un nouvel opérateur ("box") est utilisé. Au début, l'interprétation de $\Box F$ était "il est nécessaire que F" et $\neg\Box\neg F$ était "il est possible que F", noté aussi $\Diamond F$.

D'autres interprétations ont, par la suite, été proposées. Si nous interprétons $\Box F$ comme "i sait que F est vrai", la logique qui en résulte est appelée logique des connaissances. Si $\Box F$ signifie "dans tous les futurs possibles, F" ou "à partir de maintenant, F", cette logique permet de modéliser le raisonnement temporel.

Il y a des propriétés de la logique classique qui ne sont plus vérifiées dans ce genre de logiques, par exemple, le théorème de la déduction. En effet, l'implication ($\rightarrow$) et la conséquence ($\vdash$) ne coïncident plus dans les logiques modales. Ainsi, pour la conséquence, si nous avons prouvé F dans un système modal, nous pourrons en déduire $\Box F$, intuitivement, si F est prouvé, alors il est toujours vrai (c'est la raison pour laquelle $\Box$ se lit "nécessairement"). D'autre part, pour l'implication, la formule $F \rightarrow \Box F$ n'est en général pas vraie, par exemple, dans l'interprétation des connaissances, $F \rightarrow \Box F$ signifie que "si F est vrai alors i sait que F est vrai", cela veut dire que l'agent connaît tout ce qui est vrai. Ce qui en général n'est pas réaliste.

Cependant les logiques modales ont des langages suffisamment expressifs, certaines d'entre elles sont décidables ou semi-décidables. Elles ont été utilisées dans la formalisation des logiques non monotones, dont le but est de modéliser des affirmations comme "en général F est vrai mais il y a des cas particuliers où F n'est pas vrai".

Formellement les logiques modales étendent les axiomes, les règles d'inférences, les théorèmes et un langage L de la logique propositionnelle ou des prédicats du premier ordre. Le langage modal L_M est l'extension d'un langage L par des opérateurs modaux. Au niveau axiomatique l'extension est définie par la donnée supplémentaire d'axiomes et de règles d'inférences régissant l'emploi des opérateurs modaux.

Les systèmes modaux sont des systèmes formels dont les axiomes et les règles d'inférences ne dépendent pas des domaines d'applications dans lesquels

ils sont utilisés, différemment, par exemple, aux logiques de défaut de Reiter qui dépendent du domaine d'application [Rei80].

L'interprétation sémantique des logiques modales utilise les modèles de Kripke. Ceux-ci définissent des contextes pour les formules. Un modèle de Kripke peut être vu comme un ensemble de modèles classiques (ensemble de mondes possibles) reliés par une relation binaire (relation d'accessibilité). Chaque monde possible est relié, par la relation d'accessibilité, aux mondes possibles qui définissent son contexte. C'est-à-dire, la relation caractérise pour chaque monde possible w, l'ensemble des autres mondes possibles qui doivent être pris en compte pour l'interprétation des formules dans le monde w. Ainsi un modèle de Kripke représente un ensemble de mondes possibles (modèles classiques) pris dans un contexte possible. Autrement dit, l'ensemble des mondes possibles définit le sens de la modalité et peut être interprété comme un contexte. Prenons, par exemple, le contexte des croyances de l'agent i. Ce qui est nécessairement vrai dans ce contexte représente ce que croit i, et peut être dénoté par l'opérateur modal B_i.

La relation d'accessibilité caractérise l'opérateur modal et a des propriétés particulières qui dépendent du type de contexte traité. Dans cette sorte d'interprétation, une formule sera valide, si elle est vraie non seulement dans tous les mondes possibles (modèles classiques) mais aussi dans tous les contextes possibles (modèles de Kripke).

Nous ne rentrons pas dans les détails formels, ils pourront être consultés dans [Che88, HC72]. Notre intérêt est de montrer que les logiques modales sont des systèmes bien adaptés pour représenter des notions des systèmes multiagents. Elles permettent en particulier de modéliser les notions d'action, de croyance et d'obligation. Celles que nous souhaitons modéliser dans ce travail. Nous voulons représenter les actions qui font évoluer le monde "réel", les mondes des "croyances" des agents et les mondes "idéaux" choisis par les agents, tout ça en faisant cohabiter les contextes de chacune de ces notions.

Dans le reste de la section nous présentons quelques approches de logiques modales formalisant chacune de ces notions. Remarquons que pour modéliser ces notions nous avons choisi l'approche du calcul des situations. En effet, nous souhaitions avoir une implémentation du formalisme et dans cette dernière cela est plus faisable.

1.3.4 Logiques des actions

Il ne faut pas oublier que le but est de créer des systèmes constitués d'agents autonomes. Or, dans la section précédente nous avons vu que la notion d'action joue un rôle important dans la définition de l'autonomie car celle-ci peut être considérée comme la capacité de l'agent à réaliser des actions désirées[11]. Plus on demande d'autonomie pour les agents, plus on s'intéresse à considérer des aspects liés aux actions.

Il existe des formalismes modélisant l'action qui considèrent des aspects tels que : ce qu'a réalisé l'agent est *la cause* du fait que la proposition p est vraie,

[11] Sans prendre en compte le temps.

l'agent a *la capacité* ou la compétence de faire l'action, l'agent a *l'intention* de faire l'action, l'agent a *le pouvoir* de faire faire aux autres agents des actions, l'agent a *le savoir* qui permet de réaliser des actions, l'agent a *l'obligation* de faire des actions, etc.

Nous présentons ici la formalisation de l'action représentant l'aspect de la causalité, c'est-à-dire l'action comme étant *la cause* qui produit qu'une proposition[12] devient vraie ou fausse.

Remarquons que dans le calcul des situations, l'action n'est pas la seule cause du changement de valeur de vérité d'une proposition, il y a aussi des valeurs de vérité des propositions avant l'exécution d'une telle action. Nous estimons que la liaison entre action et proposition, dans ce formalisme, est plus réaliste, au moins de façon intuitive car on peut lier d'autres paramètres à la cause du changement permettant ne pas laisser toute la "responsabilité" du changement à l'action. En outre, dans cette approche une façon de résoudre le frame problem a été proposée.

Si nous nous plaçons dans le monde réel, une action effectuée par un agent permet de définir la relation qui existe entre l'agent et un certain état du monde. L'agent est la cause du changement de valeur de vérité d'une proposition si son action effectuée a influencé l'évolution "naturelle" du fait représenté par cette proposition.

L'approche de Saint Anselme

L'un des premiers à avoir caractérisé l'aspect de causalité de l'action a été Saint Anselme. Il a introduit la modalité $\boxed{Faire}_i$, où l'interprétation de $\boxed{Faire}_i p$ est "l'agent i a fait en sorte que p soit vraie". Ainsi, il a déterminé quatre attitudes possibles d'un agent i par rapport à un fait du monde représenté par la formule p. Par exemple si p représente le fait : "la télévision est en panne", les quatre attitudes sont :

(1) $\boxed{Faire}_i p$: l'agent i a fait en sorte que la télévision soit en panne.

(2) $\boxed{Faire}_i \neg p$: l'agent i a fait en sorte que la télévision ne soit pas en panne (qu'elle soit en état de fonctionnement).

(3) $\neg\boxed{Faire}_i p$: l'agent i n'a pas fait en sorte que la télévision soit en panne.

(4) $\neg\boxed{Faire}_i \neg p$: l'agent i n'a pas fait en sorte que la télévision ne soit pas en panne.

Il est clair que dans les attitudes (1) et (2) l'agent a un rôle actif. On peut traduire (1) par "i a fait tomber en panne la télévision" dont l'interprétation est : avant l'action de i, la télévision fonctionnait, après l'action de i, la télévision

[12] Nous utilisons ici indistinctement les termes proposition et fait, sans perdre de vue qu'une proposition représente un fait du monde réel. Ainsi, dire qu'une proposition est vraie revient à dire aussi qu'il existe un fait du monde réel représenté par cette proposition. De même, dire qu'une proposition est fausse veut dire que le fait représenté par la proposition n'existe pas dans le monde réel.

est en panne. De la même manière, (2) peut se reformuler en "i a réparé la télévision" dont l'interprétation est : avant l'action de i, la télévision était en panne, après l'action de i, la télévision fonctionne. On note dans ces deux cas, la dépendance qui existe entre le fait et l'action de l'agent. Il a réalisé une *action* pour transformer le *fait*.

Le rôle de l'agent dans (3) et (4) n'est pas clair concernant la causalité car l'agent n'agit pas directement, mais i s'est abstenu de faire en sorte que quelque chose soit vraie. Par exemple, (4) peut se reformuler en "i n'a pas fait fonctionner la télévision". Notez que cette affirmation dans la vie courante peut amener à la conclusion "la télévision est en panne", tandis qu'au sens strict, l'affirmation ne dit rien concernant l'état de fonctionnement de la télévision. Donc il y a deux interprétations possibles de cette affirmation. La première est que la télévision était en panne avant l'action et qu'elle est restée en panne après l'action. La seconde est que la télévision fonctionnait avant l'action et qu'elle fonctionne toujours après l'action. Si l'on garde la première interprétation, on voit que l'état du monde après l'action est le même que dans (1), la télévision est en panne. Cependant, il est évident que l'attitude de l'agent n'est pas la même dans les deux cas. Dans (1), l'agent a agit (il l'a fait tomber, l'a mouillée, lui a donné un coup de pied, etc.) tandis que dans (4) il n'a rien fait pour qu'elle soit en panne. Supposons maintenant que l'agent a la capacité et l'opportunité de réparer la télévision et qu'il s'abstient de le faire, alors l'agent a fait quelque chose pour maintenir l'état de la télévision en panne, il s'est abstenu de la réparer. Donc "il n'a pas fait en sorte que" est ambigu, soit il exprime "n'a rien fait", soit il exprime "s'est abstenu de faire".

Si l'on considère la deuxième interprétation, l'état du monde après l'action est le même que dans (2), c'est-à-dire la télévision est en état de fonctionnement. Bien que l'attitude de l'agent ne soit pas la même dans les deux cas. Dans (2), l'agent a agit (il l'a réparée) tandis que dans (4) il n'a rien fait pour qu'elle soit en état de fonctionner. Supposons maintenant que l'agent ait réalisé une action pour maintenir la télévision en état de fonctionnement. Dans cette interprétation il y a aussi une ambiguïté pour définir "il n'a pas fait en sorte que" : soit il exprime "ne rien faire", soit il exprime "faire pour maintenir".

Malgré des ambiguïtés, dans ce formalisme, on peut exprimer la différence entre faire en sorte qu'une proposition ne soit pas vraie, et ne pas faire en sorte que la proposition soit vraie.

Notons aussi que : $\boxed{Faire}_i p \models p$ et $\boxed{Faire}_i \neg p \models \neg p$. Donc si l'on a, à la fois, $\boxed{Faire}_i p$ et $\boxed{Faire}_i \neg p$, on aboutit à la contradiction $p \wedge \neg p$. Pour exprimer que l'agent ne peut pas, à la fois, faire en sorte que p soit vraie et faire en sorte que le contraire $\neg p$ soit aussi vrai, on a le schéma d'axiome suivant : $\neg(\ \boxed{Faire}_i p \ \wedge \ \boxed{Faire}_i \neg p\ )$.

L'approche de Georg Henrik von Wright

Une deuxième approche, que nous présentons, est celle proposée par Georg Henrik von Wright [vW63]. Pour caractériser la causalité, ce formalisme propose deux modalités $\boxed{Br}_i$ et $\boxed{Ss}_i$ représentant des opérateurs d'action, et

aussi un opérateur $(\overset{om}{\neg})$ applicable uniquement à des modalités. On peut intuitivement interpréter $\boxed{Br}_i p$ par "l'agent i a fait en sorte que p soit vraie"[13] et $\boxed{Ss}_i p$ veut dire "l'agent i a fait en sorte que p soit maintenue vraie"[14], et l'opérateur $(\overset{om}{\neg})$ signifie "omettre de". Il définit huit attitudes possibles d'un agent i par rapport à une proposition p, quatre pour chaque modalité. En reprenant l'exemple précédent où p est "la télévision est en panne", on a :

(1) $\boxed{Br}_i p$: l'agent i a fait en sorte que la télévision soit en panne.

(2) $\boxed{Br}_i \neg p$: l'agent i a fait en sorte que la télévision ne soit pas en panne (qu'elle soit en état de fonctionnement).

(3) $\overset{om}{\neg}\boxed{Br}_i p$: l'agent i a omis de faire en sorte que la télévision soit en panne.

(4) $\overset{om}{\neg}\boxed{Br}_i \neg p$: l'agent i a omis de faire en sorte que la télévision ne soit pas en panne.

(5) $\boxed{Ss}_i p$: l'agent i a fait en sorte que la télévision soit maintenue en panne.

(6) $\boxed{Ss}_i \neg p$: l'agent i a fait en sorte que la télévision ne soit pas maintenue en panne (qu'elle soit maintenue en état de fonctionner).

(7) $\overset{om}{\neg}\boxed{Ss}_i p$: l'agent i a omis de faire en sorte que la télévision soit maintenue en panne.

(8) $\overset{om}{\neg}\boxed{Ss}_i \neg p$: l'agent i a omis de faire en sorte que la télévision ne soit pas maintenue en panne.

Notons que l'opérateur $(\overset{om}{\neg})$ n'est pas une simple négation. En effet, il exprime non seulement le fait que l'agent n'ait pas fait l'action qui est dans le champ de $(\overset{om}{\neg})$ mais aussi qu'il aurait pu la faire et qu'il s'est abstenu de la faire, dans le sens où il en a l'opportunité et la capacité mais ne l'accomplit pas.

Dans ce formalisme, pour caractériser les modalités il faut considérer pour chaque état initial (monde avant l'action), deux états finaux (monde après l'action) : l'état résultant de l'action réalisée par l'agent, et l'état dans lequel serait le monde si l'agent n'avait pas agit, i.e. s'il avait laissé agir "la nature".

Dans les attitudes (1), (2), (5) et (6) l'agent joue un rôle actif. On peut traduire (1) par "i a mis la télévision en panne" dont l'interprétation est : avant l'action de i, la télévision fonctionnait, après l'action de i, la télévision est en panne (comme dans le formalisme de Saint Anselme); et si i n'avait pas accompli l'action, la télévision serait toujours en état de fonctionner (ce qui correspond au cours naturel des choses).

De façon similaire, (2) peut se reformuler en "i a réparé la télévision" dont l'interprétation est : avant l'action de i, la télévision était en panne, après

[13] Pour "to bring it about that".
[14] Pour "to sustain".

l'action de i, la télévision fonctionne; et si i n'avait pas agit la télévision serait en panne.

Pour (5) la traduction est "i a fait en sorte que la télévision reste en panne" dont l'interprétation est : avant l'action de i, la télévision était en panne, après l'action de i la télévision est toujours en panne; et si i n'avait pas agit, la télévision serait en état de fonctionner. Par exemple, si un électricien a essayé de réparer la télévision et i l'a empêché de le faire.

Pour (6) la traduction est "i a fait en sorte que la télévision reste en état de fonctionnement" dont l'interprétation est : avant l'action de i, la télévision fonctionnait, après l'action de i, la télévision fonctionne encore, et si i n'avait pas agit, la télévision serait en panne. Par exemple, si la foudre est tombée et i a débranché la télévision.

Notons dans les quatre cas, la dépendance existant entre l'état du monde et l'action de l'agent. Il a agit, soit pour le transformer, soit pour le maintenir dans un état qui est détourné du cours naturel des choses. L'agent a alors un rôle actif.

Le rôle de l'agent dans les autres cas est moins clair. En effet, si l'on accepte que le fait de s'abstenir de réaliser une action donnée a pour conséquence que p prend une certaine valeur de vérité, alors pour déterminer la cause d'un fait, il faudrait expliciter toutes les abstentions pour tous les agents possibles. Cela est un problème similaire à celui de la qualification (cf. page 20). Par exemple, pour déterminer la cause du fait "la télévision est en état du fonctionner" il faudrait expliciter toutes les abstentions, tels que tous ceux qui sont potentiellement capables de la faire tomber en panne s'en sont abstenus. La notion d'abstention contient implicitement d'autres notions telles que l'intention, la capacité et l'opportunité. S'abstenir peut se décomposer en vouloir faire, avoir la capacité et l'opportunité de faire et ne pas faire. Dans [HB95], Horty et Belnap ont démontré la différence entre "ne pas faire" et "omettre de faire" et l'équivalence entre "faire" ou "se retenir de faire" et "être capable de faire".

(3) peut se reformuler en "i a eu l'opportunité de mettre la télévision en panne, mais il ne l'a pas mise en panne" dont l'interprétation est : avant l'action de i, la télévision fonctionnait, après l'action de i, la télévision fonctionne encore, et si i n'avait pas agit, la télévision serait toujours en état de fonctionner. Alors ce qu'a fait i ne concerne pas l'état de fonctionnement de la télévision, mais en tout cas i aurait pu mettre la télévision en panne.

Par contre, dans (8), i ne peut pas causer la panne de télévision car elle tombe en panne "toute seule". (8) peut se reformuler "i a eu l'opportunité de maintenir la télévision en état de fonctionnement, mais il ne l'a pas maintenue en état de fonctionner" dont l'interprétation est : avant l'action de i, la télévision fonctionnait, après l'action de i, la télévision est en panne, et si i n'avait pas agit, la télévision serait de toute façon en panne. Alors i n'avait pas l'opportunité de la mettre en panne car elle se met en panne indépendamment de l'action de i.

On peut donc distinguer trois types d'attitudes :

- celles où l'agent est actif, où l'effet de l'évolution naturelle du monde et l'effet de l'action de l'agent sont différents vis-à-vis de p.

- les attitudes où l'agent est passif mais il a l'opportunité d'influencer l'évolution du monde. Dans ce cas, l'évolution naturelle du monde ne change pas la valeur de p. L'agent a l'opportunité d'influencer l'évolution du monde mais il ne le fait pas, il omet de faire une action qui change la valeur de p.

- les attitudes où l'agent est passif mais il n'a pas l'opportunité d'influencer l'évolution du monde. Dans ce cas, l'évolution naturelle du monde change la valeur de p. L'agent a l'opportunité de maintenir le monde inchangé mais il ne le fait pas, il omet de maintenir le monde dans un état donné.

Dans ce formalisme on rencontre le problème du non-déterminisme. D'une part, l'évolution de la nature est indéfinie, elle peut évoluer de plusieurs façons différentes. D'autre part, il y a différentes actions qu'un agent peut réaliser pour faire en sorte que l'on ait p.

L'approche de Kanger et Pörn

Le troisième formalisme que nous considérons est celui de Stig Kanger et Ingmar Pörn. Dans leur approche, une sorte d'équivalence entre une action A réalisée par un agent et les effets p que l'agent désire produire par A y est définie. On peut dire que les actions caractérisent ce que font les agents.

On peut même aller plus loin et considérer que c'est cette façon de décrire les actions qui donne la signification aux opérations réalisées par les agents. Si, par exemple, on définit une action réalisée par un agent comme l'action qui consiste à faire en sorte que la porte soit ouverte, et que l'on compare cette définition à une autre définition, qui serait la description des mouvements réalisés par l'agent (le mouvement qui consiste à s'approcher de la porte, à tourner la poignée, et à faire tourner la porte sur ses gongs), alors on peut remarquer que l'on pourrait attribuer une signification différente au mouvement. Par exemple, rafraîchir la température de la pièce en laissant entrer de l'air frais, c'est-à-dire faire en sorte que la température de la pièce soit plus basse. Mais si cela était le sens donné à cette action, on peut noter aussi que cette même action pourrait être réalisée par un mouvement tout à fait différent, tel que, s'approcher de la fenêtre, et l'ouvrir pour laisser entrer de l'air frais venant de l'extérieur.

Cet exemple montre que la description du *même mouvement* peut correspondre à deux actions différentes : faire que la porte soit ouverte, ou bien faire que la température de la pièce soit plus basse. D'autre part, la *même action* : faire que la température de la pièce soit plus basse, peut être réalisée par deux mouvements différents, soit aller ouvrir la porte, soit aller ouvrir la fenêtre. On peut donc conclure de cet exemple que la notion d'action, au sens de "faire en sorte que" le monde soit dans un certain état, et la notion d'action, au sens de mouvement ou plus généralement, d'ensemble d'opérations réalisées, ne peuvent pas être confondues. Il n'est pas possible de déduire l'une à partir de l'autre, même si elles ne sont pas indépendantes.

Une autre idée importante, qui vise à exprimer la notion de causalité, est que si l'agent n'avait pas agit comme il l'a fait, on n'aurait pas eu p.

Ces deux idées sont exprimées formellement à l'aide de structures M qui sont des quadruples $M =< W, R, R', T >$ où R et R' sont deux relations d'accessibilité définies comme pour les logiques modales normales[15].

Dans ces approches on remarque l'intérêt de définir la notion de causalité mais elles ne proposent pas de solution au frame problem.

1.3.5 Logique des croyances / connaissances

A part les représentations de type logique, Thayse dans [T+90] dénombre deux autres types de formalismes de représentation des connaissances[16] pour des applications informatiques : les représentations de type réseau et les représentations de type objet.

L'avantage des représentations de type réseau est d'offrir un cadre structuré. On peut citer ici les graphes conceptuels, réseaux sémantiques, les réseaux de Petri et les réseaux Bayésiens. On ne décrira pas ces formalismes puisque cela nous éloigne des objectifs.

La logique a l'avantage d'être non seulement une technique de représentation des connaissances et du raisonnement, mais aussi d'avoir un cadre où les aspects mathématiques et formels sont bien définis. La logique permet de valider le raisonnement mené dans n'importe quelle autre technique de représentation et d'analyser la sémantique des connaissances manipulées.

C'est pourquoi les approches de type logique constituent une référence à laquelle la plupart des approches alternatives se rattachent, notamment en ce qui concerne la formalisation du raisonnement.

Logiques modales des croyances

Le cadre des logiques modales est bien adapté à la modélisation des croyances d'un agent. La logique épistémique[17], appelée aussi logique des connaissances ou logique des croyances, est la logique qui gère des concepts tels que "je sais que", "je sais qu'il sait que", "je crois que" etc. Ses applications sont multiples, notamment dans l'Intelligence Artificielle. Elle permet de distinguer ce que l'agent croit possible d'être vrai (croyance incertaine) et ce qu'il croit nécessairement vrai (croyance certaine).

Formellement, les formules possiblement vraies sont vraies dans certains mondes possibles mais pas dans tous. Tandis que les formules qui sont nécessairement vraies sont vraies dans tous les mondes possibles. Ces dernières correspondent aux croyances dont l'agent ne peut pas douter, or étant donné que le mode de raisonnement est non-monotone, il peut remettre en cause le fait que ces formules soient indubitables.

Il fait une hypothèse non réaliste : "tout ce qui est cru est vrai" qui s'oppose par nature à la révision des croyances.

---

[15] Si on considère plusieurs agents i,j,k... on aura autant de relations R_i,R_j,R_k,... et R'_i,R'_j,R'_k,... qu'il y a d'agents.

[16] Nous utilisons les termes connaissances et croyances indistinctement malgré les différences faites par quelques auteurs.

[17] Quelques auteurs donnent le nom de logique doxastique à la logique des croyances, ne réservant le nom de logique épistémique qu'aux logiques des connaissances.

Moore dans [Moo84] propose une méthode fondée sur la logique modale pour formaliser le raisonnement non-monotone d'un agent introspectif idéalement rationnel : la logique autoépistémique. Il s'agit de construire toutes les extensions légales que l'agent rationnel peut croire à partir d'un ensemble de prémisses données comme les croyances de base de l'agent. La notion de légalité remplace la notion classique de consistance et impose de plus que les extensions construites soient cohérentes avec l'ensemble de prémisses, c'est-à-dire que l'ensemble des modèles de toute extension légale de la théorie doit contenir tous les modèles de l'ensemble de prémisses. Ainsi, les croyances de l'agent rationnel sont vraies dès que ses croyances de base le sont.

Ce nouveau cadre revient à reformuler la logique de McDermott [MD80, McD82] abandonnant l'axiome de la connaissance, tout en conservant la technique de point fixe. Le gain obtenu par rapport à la logique de McDermott est non seulement d'obtenir un schéma non-monotone mais aussi de restreindre les points fixes obtenus à partir d'un ensemble de prémisses. Contrairement à la logique de McDermott, on ne rajoute pas la proposition dans les extensions possibles à chaque fois qu'elle n'est pas affirmée.

Il faut préciser que sont également abandonnées la généralité et l'expressivité du langage prédicatif du premier ordre : le langage de la logique autoépistémique est purement propositionnel.

On constate en fin de compte que l'on oscille entre une représentation implicite élégante et formelle mais qui risque d'être indécidable et une représentation qui s'interprète dans le cadre totalement explicite de l'énumération de tous les mondes possibles (lorsque cela est possible et que l'ensemble des mondes possibles est fini).

D'autres approches encore tentent d'utiliser des conventions implicites qui ne sont pas directement inscrites dans le système formel : c'est le cas des approches fondées sur l'hypothèse de monde clos.

La sémantique des mondes possibles des croyances considère chaque monde comme une collection de propositions, et les croyances sont modélisées par une relation d'accessibilité B liant les mondes. Une formule est crue dans un monde si et seulement si elle est vraie dans tous les mondes accessibles par B.

Le problème posé par la révision des connaissances peut être résumé par la question suivante : étant donnée une base de connaissances représentant des connaissances à propos du monde et une nouvelle information à propos de ce monde (pouvant remettre en cause une partie des connaissances), quels changements cette nouvelle information va-t-elle produire dans la base de connaissances ?

Alchourron, Gärdenfors et Makinson (AGM) ont proposé un ensemble de postulats qu'un opérateur de révision "sensé" doit satisfaire [AGM85]. Cette caractérisation logique n'est bien sûr pas universelle et on peut lui adresser certaines critiques. Mais de nombreux théorèmes de représentation montrant l'équivalence entre les opérateurs vérifiant ces propriétés et des méthodes de révision "naturelles" ne permettent pas de remettre en doute leur rationalité. De plus il a également été montré que la théorie de la révision AGM entretenait des liens étroits avec les relations d'inférences non-monotones et la logique possibiliste.

Les critiques que l'on peut adresser aux postulats AGM ne les remettent donc pas en question mais suggèrent des adaptations, des généralisations de ce cadre de travail.

1.3.6 Logique déontique

La logique déontique est une branche de la logique modale qui raisonne sur les normes et le comportement normatif en opposition au comportement réel. Les modalités déontiques ou normatives telles qu'interdiction, permission et obligation (*forbidden*, *permitted*, *ought*) sont définies en donnant une signification particulier aux opérateurs modaux [MW93].

Des applications sont variées, par exemple, elles sont utilisées pour décrire les contraintes d'intégrité des systèmes d'information, systèmes du bureautique, systèmes de sécurité et traitement du langage naturel. Toutes ces applications ont un dénominateur commun "l'aspect normatif", ce sont des systèmes normatifs. C'est-à-dire que les normes jouent un rôle important dans le comportement de tels systèmes, et ils ont besoin de concepts normatifs pour les décrire et les spécifier.

Par exemple, une proposition non normative est "Jean a du diabète" et une proposition normative est "Jean doit faire attention avec les sucreries".

La logique déontique est introduite en 1951 par von Wright dans [vW51]. Depuis son introduction, le nombre de publications est considérable. Le système original a reçu de nombreuses citriques, von Wright, lui-même propose des variantes afin de surmonter les déficiences des premières idées. Néanmoins il continue à être considéré comme le noyau dans ce domaine et il est considéré un point de référence standard. L'auteur remarque que c'est un abus de traiter les normes comme une sorte de propositions qui peuvent être évaluées comme vraies ou fausses. Par exemple, la norme 'On doit doubler à gauche' est-elle *vraie* ou *fausse*? Si on s'intéresse à l'existence d'une tel norme, alors l'évaluation est *vraie* si l'on se place en France, cela ne sera pas le cas si l'on se situe en Angleterre. Les valeurs de vérité des propositions normatives dépendent des actions normatives effectuées par les autorités compétentes, telles que promulguer ou déroger une loi.

La logique déontique standard comporte les axiomes et les règles d'inférence suivants :

(SDL0) Toutes les tautologies du calcul des propositions

(SDL1) $Op \wedge O(p \rightarrow q) \rightarrow Oq$ axiome K

(SDL2) $Op \rightarrow Pq$ axiome D

(SDL3) Si l'on a $\vdash p$ et $\vdash p \rightarrow q$ alors on a $\vdash q$ Modus Ponens

(SDL4) Si l'on a $\vdash p$ alors on a $\vdash Op$ Nécessitation pour O

la signification intuitive de Op est 'il est obligatoire que p', on a aussi les deux définitions suivantes : $Pp \leftrightarrow \neg O \neg p$ et $Fp \leftrightarrow \neg P \neg p$, où Pp signifie 'il est permis que p' et Fp signifie 'il est interdit que p'. Donc la permission est le dual de l'obligation et interdire est ne pas permettre.

Cette logique est un système KD pour O (cf. Annexe B.4), où (SDL1) garantit la fermeture des obligations pour la conséquence logique.

Dans ce formalisme on trouve des formules (qui sont des théorèmes dans la logique KD) qui ne correspondent pas à l'intuition et qui sont dénommées paradoxes, telles que $Op \rightarrow O(p \vee q)$ le paradoxe de Ross, $P(p \vee q) \leftrightarrow Pp \vee Pq$ le paradoxe du libre choix de la permission, $Fp \rightarrow F(p \wedge q)$ le paradoxe du bon samaritain, $\neg p \rightarrow (p \rightarrow Oq)$ le paradoxe de l'obligation conditionnelle, $Op \wedge O(p \rightarrow q) \wedge (\neg p \rightarrow O\neg q) \wedge \neg p$ le paradoxe de Chisholm. D'autres approches ont été proposées. Par exemple, l'approche réductionniste [And58] qui essaie de réduire la logique déontique à la logique modale aléthique sans aucun contenu déontique sauf celui d'un atome propositionnel spécial V indiquant la notion de violation ou sanction. La formalisation se fait en utilisant un système KT pour $\Box$. Dans cette logique les notions déontiques sont définies en fonction de $\Box$ et V. Par exemple, on a $Op \leftrightarrow \Box(\neg p \rightarrow V)$ qui veut dire intuitivement 'il est nécessaire que si p n'est pas réalisé alors il y a une violation'. La permission est définie comme $Pp \leftrightarrow \Diamond(p \wedge \neg V)$ qui veut dire 'il est possible que p soit réalisé et qu'il n'y ait pas de violation'. Et l'interdiction est définie comme $Fp \leftrightarrow \Box(p \rightarrow V)$ qui veut dire 'il est nécessaire que si p est réalisé alors il y a une violation'. Quelques théorèmes qui s'ensuivent sont sujets à controverse. En plus dans ce formalisme on retrouve quelques paradoxes rencontrés dans la logique déontique standard.

1.3.7 Modèles BDI (Belief / Desire / Intention)

L'intention joue un rôle important dans la détermination du comportement rationnel des agents cherchant à atteindre leurs buts. En général, les formalismes qui concèdent une importance principale à l'intention sont nommés architectures BDI et essaient de mettre en rapport les croyances, buts et intentions d'un agent avec ses actions.

Dans [RG91], les auteurs proposent une formalisation des intentions basée sur un modèle des mondes possibles avec une représentation des temps arborescents. Chaque monde possible est une structure temporelle arborescente avec une seule branche pour le passé et plusieurs branches optionnelles pour le futur. Chaque noeud d'un arbre est reliée à un autre par une action (ou événement). Une situation définit un moment spécifique d'un monde (time-point). La situation à partir de laquelle toutes les branches divergent représente le présent. Ils introduisent des opérateurs représentant la différence entre ce qui est inévitable (vérifié dans toutes les branches représentant le futur) et ce qui est optionnel (vérifié au moins dans une branche représentant le futur). Ainsi que des opérateurs différenciant ce qui est toujours vrai (vérifié dans tous les points d'une branche représentant le futur) de ce qui est possiblement vrai (vérifie au moins dans un point représentant une branche du futur). On peut définir les formules : optionnel possiblement vraies, optionnel toujours vraies, inévitable possiblement vraies et inévitable toujours vraies.

Leur modèle comprend plusieurs éléments importants de la théorie de l'intention de Bratman. En particulier la notion d'intention est définie comme une primitive ($Intend_i$) de même que les notions de croyance (Bel_i) et de but

($Goal_i$). Les relations entre ces trois attitudes sont introduites par des axiomes. Différentes sortes d'agents rationnels sont modélisées en imposant des conditions sur la persistance des agents aux croyances, buts (désirs choisis qui sont consistants)[18] et intentions.

En ce qui concerne la sémantique dans [CL90a], Cohen et Levesque introduisent la propriété du réalisme pour l'adoption des buts que l'agent croit qu'il peut accomplir. Cette propriété n'est pas satisfaite par les désirs.

Exprimer un haut niveau de spécification cognitive pour les agents revient à décrire comment l'agent adopte une intention et comment l'agent adopte des croyances. Un tel niveau de spécification permet de faire abstraction des agents d'une façon scientifique et naturelle. Mais pour utiliser les notions cognitives, il faut les définir d'une façon rigoureuse.

Les logiques modélisant les notions de croyances, désirs et intentions (approches BDI) peuvent être utilisées pour raisonner sur des agents et sur la façon dont laquelle leurs croyances, intentions et actions sont déterminées par la satisfaction des buts.

Pour comprendre le comportement rationnel, le processus de révision des croyances, buts et intentions ont été introduits dans [RG91].

Il y a plusieurs façons d'envisager le côte pratique des théories concernant les concepts BDI.

1.3.8 Une logique basée sur les topiques

Afin de modéliser les changements des croyances des agents dans [HL00] les auteurs proposent une théorie du changement. Cette approche est basée sur la notion de topique d'information. Les axiomes basés sur les topiques pour l'adoption de croyances ainsi que pour la préservation de celles-ci constituent le noyau de cette théorie.

Dans le langage on a : AGT un ensemble des agents, Bel_i les croyances de l'agent i, $Bel_{i,j}$ les croyances mutuelles des agents i et j, ACT un ensemble d'actes de communication. Les actes de communication sont les seules actions possibles. Chaque acte de communication a la forme $< Force_{i,j}, A >$ où $Force$ est le nom de la force illocutoire (exemple, informer, demander, affirmer, etc.), i désigne l'émetteur, j le récepteur de l'acte, et A est le contenu propositionnel. Par exemple, $< informer_{voyageur,guichet}, Destination(Paris) >$ qui veut dire intuitivement 'le voyageur informe au guichet automatique sa destination, Paris'.

La logique de changement des croyances [Lon99] est une logique multimodale dynamique dont chaque attitude mentale d'agent est représentée explicitement par un opérateur modal où la sémantique est celle de mondes possibles.

Les opérateurs dynamiques sont $Done$ et $Feasible$, où $Done_\alpha A$ signifie intuitivement 'l'acte α vient d'être accompli, avant quoi A étais vrai' et $Feasible_\alpha A$ signifie 'l'acte α peut être accompli, après quoi A sera vrai'.

On suppose que chaque agent est coopératif (il aide à achever les buts d'autres participants) et sincère (les affirmations correspondent à son état men-

[18] Les désirs peuvent être inconsistants mais pas les buts.

tal).

Intuitivement l'agent adopte les croyances de l'émetteur seulement s'il juge (crois) que l'émetteur est compétent à ce sujet. La compétence de l'émetteur est donc un critère clef pour déterminer quelle partie doit être acceptée (adoptée) par l'agent récepteur.

En ce qui concerne la préservation des croyances, s'il n'y a pas de relation de dépendance (influence) entre l'acte de communication et les croyances, ces croyances peuvent être préservées dans le prochain état.

D'abord on définit les axiomes d'adoption et de préservation des croyances sans inclure la notion topique, en exprimant la relation entre agents, actes et propositions comme une primitive. Puis on présente les mêmes axiomes en exprimant les relations en termes de topiques.

L'intérêt porte sur la compétence d'un agent par rapport à une proposition, par exemple 'l'agent i est compétent sur A". Car c'est la compétence de l'émetteur sur une certaine croyance qui va influencer l'adoption de cette croyance par le récepteur. Pour exprimer la compétence on utilise une relation de dépendance C entre un agent et une formule, où $C(i, A)$ signifie 'i influence l'adoption de A par la compétence qu'il a à ce sujet'. L'adoption de croyance par dépendance est définie par l'axiome suivant : Soient i dans AGT et A une formule sans opérateurs modaux, on a :

$$Bel_i A \rightarrow A \text{ si } C(i, A).$$

La préservation des croyances doit considérer les croyances qui n'ont aucun lien avec l'acte accompli. Ce lien s'exprime par la relation d'indépendance P entre un acte de communication et une formule A, où $P(\alpha, A)$ signifie 'α n'a aucune influence sur A'. La préservation par indépendance est définie par l'axiome suivant : Soient α dans ACT et A une formule sans opérateurs modaux, on a :

$$Done_\alpha A \rightarrow A \text{ si } P(\alpha, A)$$

La notion de topique est introduite afin de définir la compétence d'un agent et l'influence d'un acte de communication sur les croyances.

Les relations de compétence et préservation sont donc définies de façon non primitive en termes de topiques. Ceux-ci permettent de justifier les liens entre les agents, les formules et les actes de communication.

De plus parler de topiques est plus intuitif que de parler de relation de dépendance, car le topique veut dire thème, sujet (ce dont on parle).

Soit $\mathbb{T}$ l'ensemble contenant tous les topiques. On associe un ensemble de topiques à chaque formule, agent et acte de communication comme suit :

- Le *sujet d'une formule* est l'ensemble des topiques à propos desquels est la formule. On suppose qu'il existe une fonction sujet qui fait correspondre à toute formule du langage un ensemble de topiques de $\mathbb{T}$. Cette fonction est caractérisée de façon récursive en fonction des connecteurs logiques et des quantificateurs.

- La *compétence d'un agent* est l'ensemble des topiques qui sont du "domaine" de l'agent. On suppose qu'il existe une fonction competence qui fait correspondre à chaque agent un ensemble de topiques de $\mathbb{T}$.

- Le *scope d'un acte de communication* est l'ensemble des topiques associés à l'acte de communication (spécifiquement au contenu propositionnel de l'acte). On suppose qu'il existe une fonction $\mathfrak{scope}$ qui fait correspondre à chaque acte de communication un ensemble de topiques de $\mathbb{T}$.

Intuitivement un agent i est compétent pour A si et seulement si l'ensemble de topiques associés à A est un sous-ensemble de l'ensemble de topiques associés à i. La relation C peut être exprime en termes de topiques comme suit $\mathfrak{sujet}(A) \subseteq \mathfrak{competence}(i)$. L'axiome d'adoption des croyances est donc reformulé de la façon suivante. Soient i dans AGT et A une formule sans opérateurs modaux, on a:

$$Bel_i A \rightarrow A \text{ si } \mathfrak{sujet}(A) \subseteq \mathfrak{competence}(i) \tag{1.2}$$

L'information A est préservée après un acte de communication α si A et α n'ont pas de topique commun. Formellement la relation P peut être exprime en termes de topiques comme $\mathfrak{scope}(\alpha) \cap \mathfrak{sujet}(A) \neq \emptyset$. L'axiome de préservation de croyances est donc reformulé comme suit. Soient α, β dans ACT et A une formule, tel que $Done_\beta$ n'apparaît pas dans A, on a:

$$Done_\alpha A \rightarrow A \text{ si } \mathfrak{scope}(\alpha) \cap \mathfrak{sujet}(A) = \emptyset \tag{1.3}$$

On remarque que les modalités ne contribuent pas à la définition de l'ensemble de topiques associés à une formule, un agent ou un acte de communication. D'ailleurs une limitation est l'interdiction des opérateurs modaux dans la formule qu'un agent cherche à adopter.

Le système ne permet pas de traiter des requêtes d'un certain domaine de compétence de l'émetteur. Pour résoudre cela, la notion de topique est affinée. Ainsi un topique est défini en termes de thèmes et de contextes.

On définit un ensemble $\mathcal{T}$ contenant les thèmes primitifs, et aussi on définit un ensemble $\mathcal{C}$ contenant les contextes. Les contextes sont des séquences de contextes atomique, et les contextes atomiques ont la forme ma_i (attitude mentale de l'agent i) pour i dans AGT.

Donc un topique est un thème associé à un contexte dénoté par $c : t$, où $c \in \mathcal{C}$ et $t \in \mathcal{T}$.

La fonction $\mathfrak{sujet}$ est aussi affinée et certaines propriétés découlent de ce raffinement, par exemple $\{ma_i : t \mid t \in \mathcal{T}\} \subseteq \mathfrak{competence}(\mathrm{i})$.

Grâce à cette nouvelle définition de topique, les axiomes d'adoption de croyances et de préservation de croyances (cf. Formules (1.2) et (1.3)) sont définis en termes d'attitudes mentales. En plus l'axiome d'adoption de croyances n'est plus restreint aux formules sans opérateurs modaux.

Dans cette approche, le problème de la ramification (cf. Section 1.2.2) est résolu en caractérisant l'adoption des croyances et le frame problem est résolu en caractérisant la préservation des croyances. Une critique de cette approche est que le nombre d'informations à fournir pour caractériser les topiques est grand et il est difficile de juger les conséquences de ces informations par rapport à la dépendance.

Comme les seuls changements qui peuvent être effectués, à l'aide des actes de communication, concernent les croyances, cette approche ne pourrait pas

s'appliquer à un domaine plus général tel que la robotique où on a d'autres types d'actions.

1.3.9 Implémentations

Nous remarquons que l'analyse des théories formelles des systèmes BDI (Cohen et Levesque, et Rao et Georgeff) montre que les contraintes exigées, i.e. consistance des intentions et connaissances complètes sur les états courants du monde, sont difficiles à atteindre au niveau pratique du point de vue des agents autonomes. Cela conduit à chercher un compromis entre la formalisation et l'implémentation des théories formelles.

Les concepts mentionnés sont repris par Shoham [Sho97], qui propose un langage de programmation orienté agents dénommé AGENT0 et qui utilise des notions telles que croyances, engagements et compétences. Dans [LRL+97] les auteurs présentent GOLOG, un langage de programmation logique qui permet de raisonner sur des actions. L'interprète de GOLOG a l'avantage d'éviter les chemins sans issue qui ne sont pas évités par l'interprète BDI.

Les outils et méthodologies utilisés pour la conception des SMA ne doivent pas seulement supporter une ample variété de concepts abstraits mais aussi être fondés sur des traitements rigoureux des abstractions. Les méthodes formelles des SMA concernent implicitement des structures mathématiques expliquant ces notions. Malgré leur courte vie, les méthodes formelles ont donné des résultats intéressants. Des techniques formelles ont été utilisées et ont influencé des systèmes pratiques.

Chapitre 2

Calcul des situations

D'abord nous présentons le calcul des situations (CS) de façon intuitive, en montrant, à partir d'un exemple, comment on peut représenter dans ce langage les actions et leurs effets. Aussi nous présentons la façon de résoudre le frame problem et le problème de qualification dans ce formalisme. Puis nous donnons les définitions formelles et nous concluons avec GOLOG, un langage et interpréteur du calcul des situations.

2.1 Introduction : un exemple

Le calcul des situations est un langage du premier ordre mais certains aspects nécessitent une extension au second ordre. Il a été conçu pour représenter des environnement dynamiques. Il permet de représenter les conditions nécessaires pour effectuer une action, ainsi que les changements provoqués par l'action dans l'environnement.

2.1.1 Situations et fluents

On fait l'hypothèse que tous les changements sont provoqués par des *actions*. Celles-ci sont représentées par des termes du langage (e.g. $avancer$, $déplacer(x, y)$). Ces actions sont des arguments d'une fonction "spéciale" appelée do, qui permet de représenter une situation (histoire), c'est-à-dire une séquence d'actions à partir d'un état initial.

L'évaluation de $do(a, s)$ donne comme résultat une situation s' qui représente l'exécution de l'action a à partir de la situation précédente s , ainsi $s' = do(a, s)$. La construction d'une situation avec do s'effectue de façon récursive sur le deuxième argument qui est aussi une situation. Il existe une constante S_0 qui représente la situation au début de l'histoire.

Un prédicat qui peut éventuellement changer sa valeur de vérité après l'exécution d'une action est appelé *fluent*. Les fluents ont un nom, comme n'importe quel prédicat. La principale caractéristique des fluents est qu'ils sont munis d'un argument de type *situation*[1]. Par exemple, supposons que l'on ait

[1]Par convention, l'argument de type *situation* est toujours placé à la fin.

le prédicat $position(x)$ représentant la position d'un robot dans un monde statique (le robot ne peut pas se déplacer, aucune action n'est permise). Dans un monde dynamique (le robot peut se déplacer : actions d'*avancer* ou de *reculer*), on utilise le fluent $position(x, s)$ pour représenter la position du robot x à l'instant[2] s. Ainsi on peut représenter le changement de position du robot après son déplacement. Intuitivement, s représente l'instant dans lequel le prédicat est évalué. Considérons, par exemple, que le robot est dans la position 5 à l'instant initial (instant considéré comme le premier, i.e. l'instant avant l'exécution de la "première" action), représenté par :

$$position(5, S_0).$$

Maintenant, si le robot avance d'une unité, la position du robot après l'exécution de l'action *avancer* à partir de l'instant initial est représentée par :

$$position(6, do(avancer, S_0)).$$

Autrement dit, la position est 6 dans ce nouvel instant. D'ailleurs le robot n'est plus dans la position 5, donc on doit pouvoir démontrer :

$$\neg position(5, do(avancer, S_0)).$$

La représentation d'une situation grandit au fur et à mesure que les actions sont exécutées, en même temps cette représentation garde la trace des actions exécutées. Par exemple, la situation du fluent suivant donne le nombre des actions exécutées (5), leurs noms (*avancer*, *observer_pos* et *observer_obs*) et l'ordre dans lequel elles ont été exécutées.

$$position(8, do(avancer, do(avancer, \\ do(observer\_obs, do(avancer, do(observer\_pos, S_0)))))).$$

L'intérêt d'utiliser le calcul des situations pour représenter les actions est que, contrairement à d'autres formalismes, il donne une axiomatique concise, qui permet de résoudre le frame problem. En plus il permet de quantifier sur les actions et les situations. Ainsi, par exemple, la représentation de l'affirmation 'la seule possibilité de changer la position du robot revient à exécuter l'une des deux actions, *avancer* ou *reculer*' est donnée par[3] :

$$\forall s \forall a \forall x \; [(position(x, s) \wedge \neg position(x, do(a, s))) \rightarrow (a = avancer \vee a = reculer)]$$

2.1.2 Solution du frame problem

Dans [Rei91], Reiter propose une solution *simple* au frame problem basée sur le calcul des situations. Le frame problem, largement connu dans la modélisation des environnement dynamiques (cf. Section 1.2.1), est résolu ici en représentant

[2]Dans cet exemple, nous utilisons le mot "instant" pour désigner, de façon intuitive, le moment occupant une place déterminée dans la suite des événements (actions). Par la suite nous utiliserons, de façon formelle, le mot "situation" pour désigner ce moment.

[3]Par la suite, nous supposons implicite la quantification universelle de toutes les variables.

l'information persistante (information qui ne change pas après l'exécution d'une action) par des axiomes. Ces axiomes sont appelés *axiomes de changement d'état*. A chaque fluent on associe un axiome de changement d'état, qui spécifie aussi les effets des actions sur le fluent. Par exemple, pour le fluent $position(x, s)$, on a l'axiome suivant :

$$\begin{aligned} &position(x, do(a, s)) \leftrightarrow \\ &\quad [a = avancer \wedge position(x-1, s) \vee a = reculer \wedge position(x+1, s)] \\ &\quad \vee position(x, s) \wedge \neg[(a = avancer \vee a = reculer) \wedge position(x, s)] \end{aligned} \tag{2.1}$$

En fait, cet axiome décrit comment interviennent les actions sur la valeur du fluent, cette intervention ou *loi causale* peut être divisée en deux, les effets positifs et les effets négatifs des actions. On peut définir pour chaque fluent, deux axiomes représentant ces effets. Par exemple, pour le fluent $position(x, s)$, l'exécution de l'action *avancer* (respectivement *reculer*) quand le robot est dans la position $x-1$ (respectivement $x+1$) dans la situation s, *provoque* que la position du robot soit x dans la situation $do(a, s)$. L'axiome représentant les effets positifs est donc le suivant :

$$\begin{aligned} [a = avancer \wedge position(x-1, s) \vee a = reculer \wedge position(x+1, s)] \rightarrow \\ position(x, do(a, s)) \end{aligned} \tag{2.2}$$

L'axiome des effets négatifs décrit que si le robot est dans la position x dans la situation s et si l'action exécutée est celle d'*avancer* ou celle de *reculer* alors la position du robot n'est plus x dans la situation $do(a, s)$, ce qui s'exprime comme suit :

$$\begin{aligned} (a = avancer \vee a = reculer) \wedge position(x, s) \rightarrow \\ \neg position(x, do(a, s)) \end{aligned} \tag{2.3}$$

2.1.3 Hypothèse de la complétude des causes

Pour résoudre le frame problem dans cette approche, on utilise l'hypothèse de la complétude des causes [Rei99], laquelle affirme que *toutes* les conditions sous lesquelles l'action a peut rendre $vrai$ le fluent (respectivement $faux$) dans la situation suivante s, sont caractérisées par l'axiome des effets positifs (respectivement l'axiome des effets négatifs). Alors si l'on accepte cette hypothèse, on peut en déduire l'axiome général (2.1) à partir de (2.2) et (2.3).

2.1.4 Solution du problème de la qualification

En fait, les actions ne peuvent être exécutées que si certaines préconditions sont satisfaites. Ces préconditions sont exprimées en utilisant un fluent spécial, nommé $Poss$, ce fluent possède deux arguments, une action a et une situation s. Intuitivement, $Poss(a, s)$ signifie qu'il est possible d'exécuter l'action a dans la situation s. Par exemple, la précondition pour exécuter l'action d'*avancer*

est qu'il n'existe pas d'obstacle dans la position qui est devant le robot $(x+1)$. Formellement on a:

$$Poss(avancer, s) \leftrightarrow \neg\exists x \exists y \ (position(x, s) \wedge obstacle(y, s) \wedge y = x+1)$$

On accepte cette représentation des préconditions qui n'est pas complète. En effet, l'emploi de l'équivalence permet d'omettre certaines conditions considérées d'importance "mineure" dans la modélisation. Supposons, par exemple, que l'on veuille savoir si l'exécution de l'action *avancer* est possible dans s, en principe il suffit de prouver $\neg\exists x \exists y \ (position(x, s) \wedge obstacle(y, s) \wedge y = x + 1)$, mais soyons réalistes, l'affirmation 'le robot peut avancer s'il n'existe aucun obstacle qui l'en empêche' est idéaliste. Il faut considérer aussi la condition physique du robot: qu'il ne soit pas cassé $(\neg(cassé(x, s)))$, qu'il ne soit pas brûlé $(\neg(brûlé(x, s)))$, etc. Il faut considérer aussi les conditions météorologiques: qu'il ne fasse pas un vent très fort $(\neg(vent(fort)))$, qu'il ne fasse pas une forte chaleur $(\neg(chaud(fort)))$, etc. On ne peut pas qualifier *toutes* les conditions qui peuvent empêcher le robot d'avancer (cf. Section 1.2.3), cependant on part de l'hypothèse que la définition de $Poss(a, s)$ considère les conditions les plus "importantes", et que celles-ci constituent "toutes" les conditions à accomplir.

2.2 Définitions formelles

Dans la section précédente nous avons introduit le calcul des situations de façon intuitive et nous avons mis en valeur les principaux concepts à prendre en compte dans ce formalisme, les situations, les fluents, les préconditions. Nous considérons dans cette section quelques aspects formels, ensuite, dans la section suivante, nous donnons un langage et une axiomatisation constituant une logique du calcul des situations.

2.2.1 Le calcul des situations, un langage typé

Dans un langage typé, les variables et en général les termes ont un type spécifique.

La syntaxe d'un langage typé est la même que celle d'un langage non typé (cf. Annexe B.2, pag. 146) mis à part que pour chaque type i on a des symboles de variables $x_1^i, x_2^i, \ldots$, un quantificateur universel $(\forall_i)$ et un quantificateur existentiel $(\exists_i)$. A chaque symbole de prédicat et de fonction n-aires on associe un ensemble de types $< i_1, i_2, \ldots i_n >$ qui définit le type de chaque argument, le type de l'argument qui occupe la position j est donc i_j. A chaque symbole de fonction on associe un type i.

Par exemple, les affirmations 'x est un enfant en 1972' et 'x est un adulte en 2002' peuvent être représentées par les prédicats typés $enfant(x, 1972)$ et $adulte(x, 2002)$ respectivement où le premier argument est de type 'personne' et le deuxième de type 'année'. L'implication 'si x est un enfant en y alors x est adulte en $y + 30$' peut être représenté par la formule

$$\forall_p x \, \forall_a y \, [enfant(x, y) \rightarrow adulte(x, addition(y, 30))],$$

où la fonction *addition* a deux arguments de type 'année' et son évaluation donne aussi une valeur de type 'année' *addition* : $année \times année \mapsto année$.

Le langage typé de premier ordre $\mathcal{L}_{\mathcal{P}}{}^t$ est similaire à celui de $\mathcal{L}_{\mathcal{P}'}$ (cf. Annexe B.2), à l'exception de la règle SYN-4., qui est donnée comme suit :

SYN-4^t. Soit x^i une variable de type i et $\psi \in \mathcal{L}_{\mathcal{P}}{}^t$ alors $\forall_i x^i\ \psi$, $\exists_i x^i\ \psi \in \mathcal{L}_{\mathcal{P}}{}^t$

Cette règle veut dire intuitivement que les quantificateurs doivent être appliqués aux variables de même type, ainsi les formules qui contiennent $\forall_i x^j$, tel que $i \neq j$, ne sont pas admises.

Au niveau sémantique, ici, à la différence de la logique non typée, le domaine est divisé en sous-domaines disjoints, chaque type d'objets a son sous-domaine respectif. Une variable prend ses valeurs dans son sous-domaine et un terme dénote un élément du sous-domaine qui lui correspond. De même les arguments des prédicats et des fonctions sont typés. La valeur donnée par chaque fonction est aussi d'un type spécifique. Les règles permettant de définir une formule satisfaisable (vraie) sur une interprétation donnée sont les mêmes que celles du langage $\mathcal{L}_{\mathcal{P}'}$ sauf qu'ici la définition d'une interprétation est adaptée pour tenir compte des types. Au lieu d'avoir une seule structure $M_{P'}$, on a ici autant de structures M_{P^i} que de types dans le langage. On associe à chaque structure M_{P^i} son quantificateurs ($\forall_i$). Chaque structure possède un ensemble non vide $\mathbb{D}_i$ appelé domaine du type i. Les domaines des différents types doivent être disjoints. Chaque $p_i^{M_{P^{j_1}} \times M_{P^{j_2}} \ldots \times M_{P^{j_{n_i}}}} \subseteq \mathbb{D}_{j_1} \times \mathbb{D}_{j_2} \ldots \mathbb{D}_{j_{n_i}}$ définit l'ensemble de n_i-uplets qui sont associés au symbole de prédicat p_i. Chaque $f_i^{M_{P^{j_1}} \times M_{P^{j_2}} \ldots \times M_{P^{j_{n_i}}}} : \mathbb{D}_{j_1} \times \mathbb{D}_{j_2} \ldots \mathbb{D}_{j_{n_i}} \mapsto \mathbb{D}_{j_{n_i+1}}$ est une fonction n_i-aire sur $\mathbb{D}_{j_{n_i+1}}$.

Pour chaque type i, on a une fonction $v_i : V^i \mapsto \mathbb{D}_i$ qui assigne à chaque variable du type i un élément du domaine $\mathbb{D}_i$.

Pour chaque type i, on a une extension de la fonction v_i, $\bar{v_i} : T^i \mapsto \mathbb{D}_i$ qui assigne à chaque terme du type i un élément du domaine $\mathbb{D}_i$.

Une interprétation est constituée d'un ensemble de structures M_{P^i}, $1 \leq i \leq n$ (n est le nombre des types dans le langage) et d'un ensemble de fonctions d'assignation de variables v_i, $1 \leq i \leq n$. La satisfaction d'une formule sous une interprétation $(M_{P^1}, v_1), \ldots, (M_{P^n}, v_n)$ est définie récursivement comme suit :

SEM-4^t. $(M_{P^1}, v_1), \ldots, (M_{P^n}, v_n) \models p(t_1^{i_1}, t_2^{i_2}, \ldots, t_m^{i_m})$ ssi
$< \bar{v_{i_1}}(t_1^{i_1}), \bar{v_{i_2}}(t_2^{i_2}), \ldots, \bar{v_{i_m}}(t_m^{i_m}) > \in p^{M_{P^{i_1}} \times M_{P^{i_2}} \ldots \times M_{P^{i_m}}}$

SEM-5^t. SEM-2 et SEM-3 avec $(M_{P^1}, v_1), \ldots, (M_{P^n}, v_n)$ comme interprétation

SEM-6^t. $(M_{P^1}, v_1), \ldots, (M_{P^i}, v_i), \ldots, (M_{P^n}, v_n) \models \forall_i x\ \psi$ ssi
pour toute $d \in \mathbb{D}_i$, $(M_{P^1}, v_1), \ldots, (M_{P^i}, v_{i,(x|d)}), \ldots, (M_{P^n}, v_n) \models \psi$,
où $v_{i,(x|d)}$ est une fonction exactement égale à v_i sauf pour la variable x, laquelle prend la valeur d

Remarque : les logiques typées ont le même pouvoir d'expression que les logiques non typées. En introduisant un prédicat T_i pour chaque type i, où $T_i(t)$ signifie que t est de type i, on peut transformer une logique typée en une logique ordinaire non typée.

Dans le calcul des situations il y a trois types d'objets ou termes : les situations, les actions et les objets.

Par commodité, nous n'utilisons pas les quantificateurs typés dans le calcul des situations, les types peuvent être déduits à partir du contexte.

2.2.2 Le calcul des situations, une logique du second ordre

Dans une logique de second ordre les quantificateurs portent aussi sur des prédicats et des fonctions à la différence des logiques de premier ordre permettant de quantifier seulement sur les éléments du domaine.

La syntaxe d'un langage de second ordre est celle de la logique du premier ordre (cf. Annexe B.2), augmentée des symboles de variables de prédicats n-aires $X_1^n, X_2^n, \ldots$, et des variables de fonctions n-aires $F_1^n, F_2^n, \ldots$ Une formule atomique est de la forme : $P(t_1, t_2, \ldots, t_n)$, où P est un prédicat ou une variable de prédicat n-aires, où t_i est un terme pour $1 \leq i \leq n$. La définition d'un terme inclut celle d'un terme de la logique du premier ordre, à laquelle on ajoute des variables de fonction possédant aussi des termes comme arguments.

Par exemple, l'axiome de l'induction de Peano est formulé comme suit :

$$\forall P \, (\, (P(0) \wedge \forall x \, (P(x) \rightarrow P(succ(x)))) \rightarrow \forall x (P(x)) \,) \tag{2.4}$$

Cet type de système est approprié pour la caractérisation de propriétés mathématiques [Göd81].

Le langage de second ordre $\mathcal{L}_{\mathcal{S}}$ est celui de $\mathcal{L}_{\mathcal{P}'}$ auquel on ajoute les règles suivantes :

SYN-5. Soit X une variable de prédicat et $\psi \in \mathcal{L}_{\mathcal{S}}$ alors $\forall X \, \psi$, $\exists X \, \psi \in \mathcal{L}_{\mathcal{S}}$

SYN-6. Soit F une variable de fonction et $\psi \in \mathcal{L}_{\mathcal{S}}$ alors $\forall F \, \psi$, $\exists F \, \psi \in \mathcal{L}_{\mathcal{S}}$

Une formule fermée est une formule bien formée qui ne possède pas de variables[4] libres.

Au niveau sémantique, on garde la structure de $\mathcal{L}_{\mathcal{P}'}$ (cf. Annexe B.2), dénotée ici par $M_{\mathcal{PS}}$. Par contre, on définit une nouvelle fonction d'assignation des variables v comme suit : (1) si x est une variable comme celles de la logique du premier ordre, alors $v(x)$ est un élément du domaine, (2) si X^n est une variable de prédicat, alors $v(X^n)$ est un ensemble de n-uplets du domaine, (3) si F^n est une variable de fonction, alors $v(F^n)$ est une fonction qui va de l'ensemble de n-uplets du domaine vers le domaine. L'extension de v, $\bar{v}$ reste la même que celle de la logique du premier ordre sauf qu'ici, on ajoute $\bar{v}(F(t_1, t_2, \ldots, t_n)) = \bar{v}(F)(\bar{v}(t_1), \bar{v}(t_2), \ldots, \bar{v}(t_n))$ pour tenir compte de l'évaluation des termes de la forme : $F(t_1, t_2, \ldots, t_n)$ où F est un symbole de variable de fonction.

Une interprétation est constituée d'une structure $M_{\mathcal{PS}}$ et d'une fonction d'assignation de variables v. La satisfaction d'une formule sous l'interprétation $(M_{\mathcal{PS}}, v)$ est définie récursivement comme suit :

SEM-7. Les règles de $\mathcal{L}_{\mathcal{P}'}$ avec $(M_{\mathcal{PS}}, v)$ comme interprétation

[4] D'aucune des trois sortes : des éléments du domaine, des prédicats ou des fonctions.

SEM-8. $M_{PS}, v \models X(t_1, t_2, \ldots, t_n)$ ssi $< \bar{v}(t_1), \bar{v}(t_2), \ldots, \bar{v}(t_n) > \in \bar{v}(X)$

SEM-9. $M_{PS}, v \models \forall X^n\ \psi$ ssi pour toute relation n-aire $R \subseteq \mathbb{D}^n$, $M_{PS}, v_{(X^n|R)} \models \psi$, où $v_{(X^n|R)}$ est une fonction exactement égale à v sauf que la variable de prédicat X^n prend la valeur R

SEM-10. $M_{PS}, v \models \forall F^n\ \psi$ ssi pour toute fonction $f : \mathbb{D}^n \mapsto \mathbb{D}$, $M_{PS}, v_{(F^n|f)} \models \psi$, où $v_{(F^n|f)}$ est une fonction exactement égale à v sauf que la variable de fonction F^n prend la valeur f

Intuitivement, la règle SEM-9 exprime que si une formule telle que $\forall X^n\ \psi$ est *vraie*, alors ψ est *vraie* sous l'interprétation donnée sans prendre en compte l'extension du prédicat X^n, cela est intéressant quand ψ contient des occurrences des variables X^n.

La définition du quantificateur ($\exists$) est la même que celle de la logique du premier ordre, i.e. $\exists x\ \psi \stackrel{\text{def}}{=} \neg\forall x\ (\neg\psi)$, pour une variable x de quelque nature (commune, de prédicat ou de fonction).

Les définitions de validité et de satisfaisabilité de la logique du premier ordre (cf. Annexe C.1) peuvent être généralisées de façon naturelle pour la logique du second ordre.

La logique de second ordre n'est pas une logique complète, il n'existe pas une axiomatique "convenable" pour déduire toutes les formules valides de la logique de second ordre.

2.2.3 Axiomes de changement d'état

Soient F un fluent quelconque, $\Upsilon^+_F(a,s)$ et $\Upsilon^-_F(a,s)$ deux formules du premier ordre dépendant d'une action a, d'une situation s et éventuellement des autres variables[5]. L'axiome de changement d'état de F est :

$$F(do(a,s)) \leftrightarrow \Upsilon^+_F(a,s) \vee [F(s) \wedge \neg\Upsilon^-_F(a,s)] \tag{2.5}$$

où $\Upsilon^+_F(a,s)$ (respectivement $\Upsilon^-_F(a,s)$) décrit les conditions qui permettent de changer la valeur de vérité de F de $faux$ à $vrai$ (respectivement de $vrai$ à $faux$) dans $do(a,s)$, i.e. dans la situation résultant de l'exécution de l'action a dans s. A la différence d'autres formalismes, le changement d'état est ici exprimé, non seulement comme une implication (les conditions sont *suffisantes*) mais plutôt comme une équivalence (les conditions sont aussi *nécessaires* pour le changement de valeur du fluent).

La longueur de l'axiome de changement d'état dépend du nombre d'actions qui modifient la valeur de vérité du fluent. En principe, le nombre d'actions, qui modifient le fluent est relativement petit par rapport au nombre d'actions qui ne le modifient pas. L'axiome de changement d'état est donc relativement court.

Si $\Upsilon^+_F(a,s)$ et $\Upsilon^-_F(a,s)$ sont inconsistantes, l'axiome de changement d'état est équivalent aux quatre axiomes suivants :

$$\Upsilon^+_F(a,s) \rightarrow F(do(a,s)) \tag{2.6}$$

[5] Par convention on omet toutes les variables à l'exception de a et s.

$$\Upsilon_F^-(a,s) \rightarrow \neg F(do(a,s)) \tag{2.7}$$
$$F(s) \wedge \neg F(do(a,s)) \rightarrow \Upsilon_F^-(a,s) \tag{2.8}$$
$$\neg F(s) \wedge F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \tag{2.9}$$

Preuve
($\Rightarrow$)

1. $F(do(a,s)) \leftrightarrow \Upsilon_F^+(a,s) \vee [F(s) \wedge \neg\Upsilon_F^-(a,s)]$ hyp
2. $\Upsilon_F^+(a,s) \vee [F(s) \wedge \neg\Upsilon_F^-(a,s)] \rightarrow F(do(a,s))$ de 1
3. $(\Upsilon_F^+(a,s) \rightarrow F(do(a,s))) \wedge (F(s) \wedge \neg\Upsilon_F^-(a,s) \rightarrow F(do(a,s)))$ de 2 et $A \vee B \rightarrow C \leftrightarrow (A \rightarrow C) \wedge (B \rightarrow C)$
4. $\Upsilon_F^+(a,s) \rightarrow F(do(a,s))$ de 3
5. $F(s) \wedge \neg\Upsilon_F^-(a,s) \rightarrow F(do(a,s))$ de 3
6. $F(s) \wedge \neg F(do(a,s)) \rightarrow \Upsilon_F^-(a,s)$ de 5
7. $F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee [F(s) \wedge \neg\Upsilon_F^-(a,s)]$ de 1
8. $F(do(a,s)) \rightarrow (\Upsilon_F^+(a,s) \vee F(s)) \wedge (\Upsilon_F^+(a,s) \vee \neg\Upsilon_F^-(a,s))$ de 7
9. $(F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee F(s)) \wedge (F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee \neg\Upsilon_F^-(a,s))$ de 8 et $A \rightarrow (B \wedge C) \leftrightarrow (A \rightarrow B) \wedge (A \rightarrow C)$
10. $F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee F(s)$ de 9
11. $\neg F(s) \wedge F(do(a,s)) \rightarrow \Upsilon_F^+(a,s)$ de 10
12. $F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee \neg\Upsilon_F^-(a,s)$ de 9
13. $\neg\Upsilon_F^+(a,s) \wedge \Upsilon_F^-(a,s) \rightarrow \neg F(do(a,s))$ contraposée de 12
14. $\Upsilon_F^-(a,s) \rightarrow \neg\Upsilon_F^+(a,s)$ inconsistance de $\Upsilon_F^-(a,s)$ et $\Upsilon_F^+(a,s)$
15. $\Upsilon_F^-(a,s) \rightarrow \neg\Upsilon_F^+(a,s) \wedge \Upsilon_F^-(a,s)$ de 14
16. $\Upsilon_F^-(a,s) \rightarrow \neg F(do(a,s))$ de 15 et 13

Alors à partir de $F(do(a,s)) \leftrightarrow \Upsilon_F^+(a,s) \vee [F(s) \wedge \neg\Upsilon_F^-(a,s)]$ on peut déduire les conséquences 4, 16, 6 et 11.

($\Leftarrow$)

1. $\neg F(s) \wedge F(do(a,s)) \rightarrow \Upsilon_F^+(a,s)$ axiome 2.9
2. $F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee F(s)$ de 1
3. $\Upsilon_F^-(a,s) \rightarrow \neg F(do(a,s))$ axiome 2.7
4. $F(do(a,s)) \rightarrow \neg\Upsilon_F^-(a,s)$ contraposé de 3
5. $F(do(a,s)) \rightarrow (\Upsilon_F^+(a,s) \vee F(s)) \wedge \neg\Upsilon_F^-(a,s)$ de 2 et 4

6. $F(do(a,s)) \rightarrow (\Upsilon_F^+(a,s) \wedge \neg\Upsilon_F^-(a,s)) \vee (F(s) \wedge \neg\Upsilon_F^-(a,s))$ de 5
7. $\Upsilon_F^+(a,s) \wedge \neg\Upsilon_F^-(a,s) \leftrightarrow \Upsilon_F^+(a,s)$ inconsistance de $\Upsilon_F^-(a,s)$ et $\Upsilon_F^+(a,s)$
8. $F(do(a,s)) \rightarrow \Upsilon_F^+(a,s) \vee (F(s) \wedge \neg\Upsilon_F^-(a,s))$ de 6 et 7
9. $\Upsilon_F^+(a,s) \rightarrow F(do(a,s))$ axiome 2.6
10. $F(s) \wedge \neg F(do(a,s)) \rightarrow \Upsilon_F^-(a,s)$ axiome 2.8
11. $F(s) \wedge \neg\Upsilon_F^-(a,s) \rightarrow F(do(a,s))$ de 10
12. $\Upsilon_F^+(a,s) \vee (F(s) \wedge \neg\Upsilon_F^-(a,s)) \rightarrow F(do(a,s))$ de 9 et 11
13. $F(do(a,s)) \leftrightarrow \Upsilon_F^+(a,s) \vee [F(s) \wedge \neg\Upsilon_F^-(a,s)]$ de 8 et 12

Alors à partir des axiomes 2.6, 2.7, 2.8 et 2.9, on peut déduire $F(do(a,s)) \leftrightarrow \Upsilon_F^+(a,s) \vee [F(s) \wedge \neg\Upsilon_F^-(a,s)]$. ■

Dans la démonstration on utilise implicitement les axiomes d'unicité des noms d'actions, i.e. pour deux actions différentes A et B on a $\neg(A = B)$.

Les axiomes[6] (2.6) et (2.7) représentent les effets positifs de F et les effets négatifs de F respectivement. Les axiomes[7] (2.8) et (2.9) représentent l'hypothèse de la complétude des causes. Intuitivement, (2.8) veut dire que si F est *vrai* dans la situation s et s'il change sa valeur de vérité dans la situation qui suit s (la situation[8] $do(a,s)$), alors la condition $\Upsilon_F^-(a,s)$ qui rend *faux* le fluent doit être satisfaite. De façon similaire (2.9) veut dire que si F est *faux* dans la situation s, et s'il devient *vrai* dans la situation qui suit s (la situation $do(a,s)$), alors la condition $\Upsilon_F^+(a,s)$ qui rend *vrai* le fluent doit être satisfaite.

L'hypothèse de la complétude des causes, utilisée dans cette solution du frame problem, caractérise le fait que les actions qui peuvent changer la valeur de vérité du fluent F sont connues à l'avance et qu'il n'existe aucune autre action qui puisse le faire changer. Par ailleurs l'hypothèse garantit que *toutes* les actions qui peuvent changer la valeur de vérité du fluent F sont connues.

Les contraposées de (2.8) et (2.9) sont deux propriétés intéressantes (cf. (2.10) et (2.11) ci-dessous). De façon intuitive, on peut dire que ces propriétés donnent une solution au frame problem, i.e. représentent formellement l'information persistante sous la forme d'axiomes. (2.10) signifie que si F est *vrai* avant l'exécution de a et si les effets de a ne sont pas en relation avec F, alors F continue à être *vrai* après l'exécution de a, autrement dit F reste invariante. De façon similaire (2.11) signifie que si F est *faux* avant l'exécution de a et si les effets de a ne sont pas en relation avec F, alors F continue à être *faux* après l'exécution de a.

$$\neg\Upsilon_F^-(a,s) \rightarrow [F(s) \rightarrow F(do(a,s))] \quad (2.10)$$

$$\neg\Upsilon_F^+(\vec{x},a,s) \rightarrow [\neg F(\vec{x},s) \rightarrow \neg F(\vec{x},do(a,s))] \quad (2.11)$$

[6] Dans [Rei99], l'auteur appelle ces axiomes formes normales des axiomes des effets.

[7] Dans [Rei99], l'auteur appelle ces axiomes des explications fermées.

[8] On fait l'hypothèse que la situation qui suit s est toujours conséquence de l'exécution d'une action a, cette nouvelle situation est donc représentée par $do(a,s)$.

L'avantage de ces propriétés, par rapport aux autres approches représentant les informations invariantes, est sa *concision*. Considérons, par exemple, l'approche des frame axioms. Soit $position(x, s)$ un fluent, quelques frame axioms positifs sont : $position(x, s) \rightarrow position(x, do(observer\_pos, s))$, $position(x, s) \rightarrow position(x, do(casser(y), s))$, $position(x, s) \rightarrow position(x, do(brûler(y), s))$, etc., qui dénotent intuitivement que le robot reste dans la même position après l'exécution des actions 'observer la position du robot', 'casser un objet y' ou 'brûler un objet y' respectivement. Quelques frame axioms négatifs sont :
$\neg position(x, s) \rightarrow \neg position(x, do(peindre(y), s))$,
$\neg position(x, s) \rightarrow \neg position(x, do(observer\_obs, s))$,
$\neg position(x, s) \rightarrow \neg position(x, do(casser(y), s))$, etc., qui dénotent intuitivement que si le robot n'était pas dans la position x avant l'exécution des actions de 'peindre un objet y', 'observer la présence d'un obstacle' ou 'casser un objet y' respectivement, alors le robot continue à ne pas être dans la position x après l'exécution de ces actions respectivement. L'utilisation des frame axioms oblige à introduire un grand nombre de tels axiomes puisque le nombre d'actions qui modifient la valeur d'un fluent est en général relativement petit par rapport à celles qui laissent le fluent invariant. Au niveau implémentation le problème est évident, on perd en efficacité dès que l'on cherche à gérer et raisonner sur un nombre relativement grand d'axiomes. Par exemple, pour 100 actions et 50 fluents on a autour de 10000 frame axioms (100 actions $\times$ 50 fluents $\times$ 2 frame axioms, l'un positif et l'autre négatif) sur lesquels on doit faire des déductions. En fait, le frame problem a été énoncé, pour la première fois, de la façon suivante : trouver une procédure qui permet de générer tous les frame axioms et si c'est possible une représentation concise de ces axiomes [MH69].

Les axiomes (2.10) et (2.11) proposent une représentation compacte des frame axioms. Etant donné la quantification universelle sur les actions, le nombre des axiomes est réduit. Dans l'exemple précédent on n'a que 100 axiomes (50 fluents $\times$ 2 frame axioms compactes).

La solution proposée par Reiter considère que les actions sont toutes déterministes, c'est-à-dire que l'effet d'une action est *bien déterminé*. Des effets, tels que : $\exists x\ malade(x, do(manger(poisson\_empoisonné), s))$ ou $pile(do(lancer\_pièce, s)) \vee face(do(lancer\_pièce, s))$ ne sont pas admis.

Jusqu'à présent on a considéré seulement les actions primitives, dans GOLOG, un langage de programmation basé sur le calcul des situations, on peut définir des actions complexes à partir des actions primitives et des constructeurs.

2.3 La logique du calcul des situations

2.3.1 Langage du calcul des situations

Le langage du calcul des situation $\mathcal{L}_{\mathcal{CS}}$ est un langage du second ordre typé avec l'égalité (cf. Annexe B.3). Les trois types d'objets manipulés sont : *action*, *situation* et *objet* pour dénoter les actions, les situations et les éléments quelconques (sauf des actions ou des situations). En général pour dénoter les variables de type *action*, on utilise le symbole a avec des indices, pour dénoter les variables de type *situation*, on utilise le symbole s avec des indices, pour le

reste des objets il n'y a pas de contrainte sauf que l'on n'emploie, ni le symbole a, ni le symbole s.

Le langage est formé par :

1. Deux symboles de fonction de type *situation*, une fonction d'arité zéro ou constante S_0 et une fonction d'arité deux do : $action \times situation \mapsto situation$. Intuitivement, $do(a, s)$ dénote la situation qui résulte de l'exécution de a à partir de s.

2. Un symbole de prédicat d'arité deux $=$: $(action \cup situation \cup objet)^2$ définissant la relation d'identité sur les trois domaines. Intuitivement, $t_1 = t_2$ signifie que t_1 est égal à t_2, évidemment t_1 et t_2 doivent être du même type.

3. Un symbole de prédicat d'arité deux $\prec$: $situation \times situation$ définissant une relation d'ordre sur les situations. Ici, on interprète une situation comme une séquence d'actions ou histoire, où $s_1 \prec s_2$ signifie que la situation s_1 est une sous-histoire de s_2. On peut le voir comme un ordre partiel "temporel".

4. Un symbole de prédicat d'arité deux $Poss$: $action \times situation$. $Poss(a, s)$ signifie que l'action a peut être exécutée dans la situation s.

5. Un ensemble de symboles de prédicats d'arité n et de type $(action \cup objet)^n$. Ces prédicats ne dépendent pas des situations, on peut dire que ce sont des prédicats qui maintiennent leur extension "constante", en ne changeant pas après l'exécution des actions.

6. Un ensemble de symboles de fonctions d'arité n et de type $(action \cup objet)^n \mapsto (action \cup objet)$. Comme dans le cas des prédicats, ces fonctions ne dépendent pas des situations, et ne changent pas après l'exécution des actions. En particulier, les fonctions de type $(action \cup objet)^n \mapsto action$ sont appelées *fonctions d'action*. Celles-ci sont considérées de façon spéciale dans l'axiomatique. On associe un *axiome de préconditions d'action* à chaque fonction d'action.

7. Un ensemble de symboles de prédicats d'arité $n + 1$ et de type $(action \cup objet)^n \times situation$. Ces prédicats sont appelés *fluents*. Les fluents dépendent des situations, on peut dire que ce sont des prédicats qui "changent" leur extension éventuellement après l'exécution d'une action. Par exemple, supposons qu'il y ait deux obstacles, l'un dans la position 2 et l'autre dans la position 5, dans la situation initiale. L'extension est donc $obstacle = \{2, 5\}$. Supposons l'exécution de l'action 'enlever l'obstacle de la position 5', la nouvelle extension est donc $obstacle = \{2\}$ et ainsi de suite. Puisque le dernier argument, celui de la situation, saisit la notion du changement, pour l'exemple précédent, on a : $obstacle = \{(2, S_0), (5, S_0), (2, do(enlever\_obs, S_0)), \ldots\}$ définissant l'extension du fluent *obstacle* qui reste inchangé.

8. Un ensemble de symboles de fonctions d'arité $n+1$ et de type $(action \cup objet)^n \times situation \mapsto (action \times objet)$. Ces fonctions sont appelées *fonctions fluents*. Celles-ci dépendent des situations. Comme dans le cas des fluents ces fonctions "changent" leur valeur éventuellement après l'exécution d'une action.

9. Des symboles de variables de prédicat de différentes arités pour considérer les formules du second ordre.

2.3.2 Axiomatique du calcul des situations

Caractérisation des situations

Une situation est une séquence finie d'actions. Les quatre axiomes fondamentaux $\mathcal{S}$ caractérisant les situations sont [Rei93] :

$$do(a_1, s_1) = do(a_2, s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2 \tag{2.12}$$

$$\forall P\ [P(S_0) \wedge \forall a \forall s\,(P(s) \rightarrow P(do(a, s))) \rightarrow \forall s\, P(s)] \tag{2.13}$$

$$\neg(s \prec S_0) \tag{2.14}$$

$$s_1 \prec do(a, s_2) \leftrightarrow s_1 \prec s_2 \vee s_1 = s_2 \tag{2.15}$$

Le premier axiome caractérise l'unicité de noms des situations. 2.13 est l'axiome d'induction de second ordre (cf. formule 2.4), il caractérise le plus petit ensemble tel que : S_0 est dans l'ensemble et si s est dans l'ensemble et a appartient à l'ensemble d'actions alors $do(a, s)$ est dans l'ensemble. Autrement dit, toute situation est obtenue à partir de la situation initiale en appliquant la fonction do un nombre fini de fois. Il y a une analogie entre le calcul des situations et la théorie des nombres. Ces deux premiers axiomes ont une analogie avec les axiomes qui caractérisent la fonction successeur $succ$ des nombres naturels, $succ(x) = succ(y) \rightarrow x = y$ et $\forall P\ [P(0) \wedge \forall x\,(P(x) \rightarrow P(succ(x))) \rightarrow \forall x\, P(x)]$.

Deux situations différentes peuvent "assigner" à tous les fluents la même valeur de vérité, autrement dit, les exécutions de deux situations différentes peuvent donner comme résultat le même état du monde. Par exemple, les situations $s_1 = do(reculer, do(avancer, S_0))$ et $s_2 = do(avancer, do(reculer, S_0))$ donnent le même état du monde. Intuitivement, cela veut dire que le robot est dans la même position x, si d'abord il recule et après il avance ou si d'abord il avance et après il recule (bon, peut être une exception se produira si le robot est juste en face d'un précipice !). On peut remarquer que les états peuvent se répéter alors que ce n'est pas le cas pour les situations. Dans l'exemple précédant l'état initial et l'état après l'exécution des actions constituant s_1 (ou s_2) sont égaux, tandis que les situations "responsables" de ces états ne sont pas les mêmes $\neg(S_0 = s_1)$ (ou $\neg(S_0 = s_2)$).

Les deux derniers axiomes définissent une relation d'ordre sur les situations. Intuitivement, $s_1 \prec s_2$ signifie que s_2 est obtenu en rajoutant une ou plusieurs actions à s_1. Ces deux axiomes ont aussi leurs analogues caractérisant la relation d'ordre 'plus petit que' dans la théorie des nombres naturels, $\neg(n < 0)$ et $n < succ(m) \leftrightarrow n < m \vee n = m$.

Les axiomes fondamentaux sont indépendants du domaine.

La sémantique pour les axiomes du premier ordre est celle de la logique classique du premier ordre.

La sémantique est semblable à celle de la logique classique du premier ordre.

2.3.3 Caractérisation des situations exécutables

On vient de caractériser les situations. En principe on n'impose pas de conditions particulières pour construire des suites d'actions constituant les situations. Ainsi la construction d'une situation s peut éventuellement ne pas satisfaire les préconditions des actions qui la composent. Dans certaines applications on ne s'intéresse qu'aux situations *exécutables*, celles-ci sont des situations qui satisfont :

$$exécutable(s) \stackrel{\text{def}}{=} \forall a \forall s' \; [do(a, s') \prec s \vee do(a, s') = s \rightarrow Poss(a, s')].$$

La signification intuitive de $exécutable(s)$ est 'toutes les actions qui apparaissent dans s peuvent être exécutées de façon ordonnée'. Par exemple si $s = do(a_3, do(a_2, do(a_1, S_0)))$ et si a_1 peut être exécutée dans S_0, a_2 peut être exécutée dans $do(a_1, S_0)$ et a_3 peut être exécutée dans $do(a_2, do(a_1, S_0))$ alors $exécutable(s)$ est satisfait.

2.4 Composants des théories de l'action

On considère des théories constituées par :

- Les quatre axiomes fondamentaux pour les situations $\mathcal{S}$, 2.12 à 2.15. Ces sont des axiomes indépendants du domaine pour le calcul des situations. Ils incluent l'axiome d'unicité de noms pour les situations et l'axiome d'induction.

- Un ensemble d'axiomes des préconditions pour les actions $\mathcal{D}_{pa}$. Pour chaque action a, on a un axiome de la forme :

  $$Poss(a, s) \leftrightarrow \Pi_a(s),$$

  où $\Pi_a(s)$ est une formule uniforme[9] pour la situation s.

- Un ensemble d'axiomes de changement d'état $\mathcal{D}_{ce}$. Pour chaque fluent F on a un axiome de la forme[10] :

  $$F(do(a, s)) \leftrightarrow \Upsilon_F^+(a, s) \vee [F(s) \wedge \neg \Upsilon_F^-(a, s)],$$

  où $\Upsilon_F^+(a, s) \vee [F(s) \wedge \neg \Upsilon_F^-(a, s)]$ est une formule uniforme pour la situation s.

[9] Dans [Rei99], l'auteur définit une formule uniforme pour une situation s comme une formule qui ne contient pas les prédicats *Poss* et $\prec$, qui ne quantifie pas universellement sur les variables de type *situation*, qui ne contient pas le prédicat $=$ appliqué aux termes de type *situation* et dont le seul terme de type *situation* qui apparaît est s.

[10] Pour chaque fonction fluent il y a aussi un axiome similaire (cf. [Rei99]).

- Un ensemble d'axiomes d'unicité des noms pour les actions $\mathcal{D}_{una}$.

- Un ensemble d'axiomes décrivant l'état initial $\mathcal{D}_{S_0}$. Cet ensemble est constitué par des formules fermées représentant les faits qui sont vrais initialement, avant l'exécution de la premier action. Si les formules possèdent un argument de type *situation*, ce sera la situation S_0 et aucune autre. Cet ensemble est appelé la *théorie initiale* et représente le monde avant toute exécution des actions.

Ainsi, une théorie de l'action (dépendante du domaine) est un ensemble d'axiomes $\mathcal{D}$,

$$\mathcal{D} = \mathcal{S} \cup \mathcal{D}_{pa} \cup \mathcal{D}_{ce} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}.$$

2.5 Opérateur de régression

La régression est un mécanisme permettant d'automatiser des procédures de planification et la démonstration des théorèmes basés sur le calcul des situations. Le principe est le suivant : soit W une formule fermée et $\mathcal{D}$ une théorie d'action, pour démontrer que W est conséquence de $\mathcal{D}$, un processus itératif est déclenché. Dans un premier temps, le processus permet d'obtenir une formule W' équivalent à W dont la longueur des termes de type *situation* a diminué d'une unité par rapport aux termes de type *situation* apparaissant dans W. Le processus continue jusqu'à l'obtention d'une formule W^* équivalente à W où tous les termes de type *situation* qui apparaissent sont de la forme S_0. Dans ce processus on se sert des axiomes de changement d'état. Ainsi l'appartenance à la théorie $\mathcal{D}$ de W est déduite en démontrant que W^* est conséquence de la théorie initiale $\mathcal{D}_{S_0}$ (cf. Section 3.3.3).

2.6 GOLOG : un langage de programmation du calcul des situations

Dans les sections précédentes on a utilisé le calcul des situations pour représenter et raisonner sur des actions "primitives" (simples). Dans un environnement dynamique, les actions primitives sont en général insuffisantes pour représenter les changements d'état. D'ailleurs la plupart des spécifications qui n'utilisent que des actions primitives deviennent difficiles à manipuler. La définition des actions "complexes" et des procédures permet donc de représenter l'environnement d'une façon concise et complète. Leur interprétation est mieux comprise intuitivement. Par exemple, on peut définir deux actions complexes et une procédure pour contrôler le déplacement d'un robot.

- $avancer\_prudence \stackrel{\text{def}}{=}$ **if** $\neg obstacle(x{+}1)$ **then** *avancer* **else** *enlever_obs* **endif**.

- **while** $\neg position(13)$ **do** *avancer_prudence* **endwhile**.

- **proc** $début(n)$ { **if** $n = 0$ **then** *stop* **else** *reculer*;$début(n{-}1)$} **endif endproc**.

L'action conditionnelle *avancer_prudence* signifie intuitivement que le robot avance s'il n'y a pas d'obstacle, sinon il enlève l'obstacle. La deuxième action complexe, introduit la notion d'itération, celle-ci est construite à l'aide du constructeur **while** et signifie intuitivement que le robot avance avec prudence (action *avancer_prudence*) jusqu'à la position 13. La procédure *début* amène le robot au début de la voie quelque soit sa position.

Pour représenter ce type d'actions et introduire des procédures, dans [LRL+97], les auteurs proposent le langage de programmation GOLOG (alGOl in LOGic) où les actions complexes sont définies en utilisant des structures de contrôle qui n'appartiennent pas au langage de la logiques (**if**, **while**, etc.), qui sont des abréviations des expressions du calcul des situations. Les expressions qui incluent des structures de contrôle peuvent être vues comme des macros qui seront étendues et qui donneront une expression *authentique* (sans aucune structure de contrôle) du calcul des situations.

Dans GOLOG l'abréviation $Do(\delta, s, s')$ est définie en utilisant les six constructeurs ci-dessous[11], où δ est une expression qui représente une action complexe, et s' est une situation résultante après l'exécution de δ à partir de s. δ est l'un des six cas : une action primitive a, une action de test $\phi?$, une séquence d'actions $[\delta_1; \delta_2]$, un choix non déterministe d'une des deux actions $(\delta_1|\delta_2)$, un choix non déterministe d'un argument d'action $(\pi x)\,\delta(x)$ ou une interaction non déterministe δ^*.

(Do1) $Do(a, s, s') \stackrel{\text{def}}{=} Poss(a[s], s) \wedge s' = do(a[s], s)$

(Do2) $Do(\phi?, s, s') \stackrel{\text{def}}{=} \phi[s] \wedge s' = s$

(Do3) $Do([\delta_1; \delta_2], s, s') \stackrel{\text{def}}{=} (\exists s^*)(Do(\delta_1, s, s^*) \wedge Do(\delta_2, s^*, s'))$

(Do4) $Do((\delta_1|\delta_2), s, s') \stackrel{\text{def}}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$

(Do5) $Do((\pi x)\delta(x), s, s') \stackrel{\text{def}}{=} (\exists x) Do(\delta(x), s, s')$

(Do6) $Do(\delta^*, s, s') \stackrel{\text{def}}{=} \forall P\,[\,(\forall s_1)P(s_1, s_1) \wedge (\forall s_1, s_2, s_3)(P(s_1, s_2) \wedge Do(\delta, s_2, s_3)$
$\rightarrow P(s_1, s_3))\,] \rightarrow P(s, s')$.

δ n'est pas une formule du calcul des situations. Toutes les expressions apparaissant dans δ n'incluent pas des arguments de type *situation*. $\delta[s]$ dénote la formule δ telle que les arguments de type *situation* ont été restaurés. Par exemple, si $\delta = position(x) \vee obstacle(y)$ alors $\delta[s_n] = position(x, s_n) \vee obstacle(y, s_n)$.

$Do(\delta, s, s')$ peut être étendu de façon récursive jusqu'à former une expression du calcul des situations, où les conditions d'arrêt sont définies par les deux premiers constructeurs.

Par exemple, l'expansion de[12]

$$Do(r; [w?; d]|[\neg w?; l], S_0, s')$$

[11] Il ne faut pas confondre cette abréviation avec la fonction *do* vue dans la section précédente qui représente un terme de type *situation*.

[12] $r = se\_réveiller$, $w = week-end$, $d = dormir$ et $l = se\_lever$.

donne la formule 2.16.
Expansion :

(a) $\exists s_1\, Do(r, S_0, s_1) \wedge Do([w?; d]|[\neg w?; l], s_1, s')$

(b) $\exists s_1\, Poss(r, S_0) \wedge s_1 = do(r, S_0) \wedge Do([w?; d]|[\neg w?; l], s_1, s')$

(c) $\exists s_1\, Poss(r, S_0) \wedge s_1 = do(r, S_0) \wedge (Do([w?; d], s_1, s') \vee Do([\neg w?; l], s_1, s'))$

(d) $\exists s_1\, Poss(r, S_0) \wedge s_1 = do(r, S_0) \wedge ((\exists s_2\, Do(w?, s_1, s_2) \wedge Do(d, s_2, s')) \vee Do([\neg w?; l], s_1, s'))$

(e) $\exists s_1\, Poss(r, S_0) \wedge s_1 = do(r, S_0) \wedge ((\exists s_2\, w(s_1) \wedge s_2 = s_1 \wedge Poss(d, s_2) \wedge s' = do(d, s_2)) \vee Do([\neg w?; l], s_1, s'))$

(f) $\exists s_1\, Poss(r, S_0) \wedge s_1 = do(r, S_0) \wedge ((\exists s_2\, w(s_1) \wedge s_2 = s_1 \wedge Poss(d, s_2) \wedge s' = do(d, s_2)) \vee (\exists s_3\, Do(\neg w?, s_1, s_3) \wedge Do(l, s_3, s')))$

(g) $\exists s_1\, Poss(r, S_0) \wedge s_1 = do(r, S_0) \wedge ((\exists s_2\, w(s_1) \wedge s_2 = s_1 \wedge Poss(d, s_2) \wedge s' = do(d, s_2)) \vee (\exists s_3\, \neg w(s_1) \wedge s_3 = s_1 \wedge Poss(l, s_3) \wedge s' = do(l, s_3)))$

en utilisant les propriétés de l'égalité, on obtient la formule du calcul des situations suivante :

$$\begin{array}{c} Poss(r, S_0) \wedge \\ (\ [w(do(r, S_0)) \wedge Poss(d, do(r, S_0)) \wedge s' = do(d, do(r, S_0))] \vee \\ [\neg w(do(r, S_0)) \wedge Poss(l, do(r, S_0)) \wedge s' = do(l, do(r, S_0))]\ ) \end{array} \qquad (2.16)$$

Puisque les actions peuvent être non déterministes, la valeur de la situation finale s' n'est pas unique. Dans l'exemple précédent, la situation finale peut prendre l'une des deux valeurs possibles $s' = do(d, do(r, S_0))$ ou $s' = do(l, do(r, S_0))$, qui veulent dire intuitivement 'soit on se réveille et on s'endort tout de suite, soit on se réveille et on se lève'.

La valeur de vérité de $Do(\delta, s, s')$ est calculée en utilisant les axiomes de changement d'état pour les fluents et les axiomes de préconditions pour les actions.

Des actions complexes telles que conditionnelle et boucle **while** sont définies á partir des constructeurs. Par exemple :

- **if** ϕ **then** δ_1 **else** δ_2 endif $\stackrel{\text{def}}{=}$ $[\phi?; \delta_1]|[\neg\phi?; \delta_2]$.

- **while** ϕ **do** δ **endwhile** $\stackrel{\text{def}}{=}$ $[[\phi?; \delta]^*; \neg\phi?]$.

En général une action complexe est définie comme une expression qui contient des actions primitives et des appels de procédures, et qui est construite en utilisant les six constructeurs mentionnés.

Les actions complexes sont utilisées pour définir un programme de la façon suivante :

proc $P_1(\bar{x_1})\delta_1$ **endproc**; ...; **proc** $P_n(\bar{x_n})\delta_n$ **endproc**; δ_0.

Un programme GOLOG comporte une séquence de déclarations de procédures et un corps principal δ_0.

L'expansion d'une procédure P d'arité $n+2$ est définie comme suit :

$$Do(P(t_1, \ldots, t_n), s, s') \stackrel{\text{def}}{=} P(t_1[s], \ldots, t_n[s], s, s').$$

ce qui veut dire intuitivement que l'exécution de la procédure P avec les paramètres $t_1, \ldots, t_n$ donne comme conséquence la transition de la situation s vers s'.

Il existe des propriétés qui peuvent être vérifiées formellement en utilisant le langage GOLOG. Par exemple, étant donnée une expression (programme GOLOG) δ, et l'ensemble d'axiomes modélisant une application Σ, trouver une situation finale s dont l'expansion (exécution) est possible revient à démontrer le théorème (prouver que le programme termine) suivant :

$$\Sigma \models \exists s\, Do(\delta, S_0, s).$$

Si le programme δ termine, alors la propriété A est satisfaite, ce qui revient à prouver :

$$\Sigma \models \forall s\, (Do(\delta, S_0, s) \rightarrow A(s)).$$

Puisque le fluent *Poss* fait partie de l'expansion du premier constructeur quand il s'agit de l'expansion d'une action primitive (cf. Do1), on peut prouver que tout programme δ qui termine, atteint une situation finale s exécutable.

$$\mathcal{S} \models \forall s\, (Do(\delta, S_0, s) \rightarrow \mathit{exécutable}(s)).$$

où $\mathcal{S}$ est l'ensemble des axiomes fondamentaux du calcul des situations (cf. Section 2.3.2).

Un interpréteur du langage GOLOG a été implémenté en Prolog (cf. Section 5.1.6).

2.7 Applications

Depuis son apparition, le calcul des situations a été utilisé pour modéliser des applications de planification. Donner un sens logique aux problèmes de planification est très important, au moins dans la spécification de la planification, pour définir ce qui doit être respecté dans le système.

Le problème de la planification (cf. Section 1.2.4) peut être interprété comme suit : trouver une séquence d'actions telle que s'il est possible d'exécuter cette séquence dans le "monde réel" avec une axiomatisation de la situation initiale, alors une telle séquence amène à une situation dans laquelle une formule fermée, appelée but, est satisfaite (*vraie*).

Prenons l'exemple du robot qui avance et qui recule, ajoutons l'action d'enlever un obstacle (*enlever_obs*). Supposons qu'au début le robot est dans la position 5, qu'il y a un obstacle dans la position 7 et que le but à atteindre est le robot dans la position 8. Dans le calcul des situations on a donc les axiomes suivants :

- Les axiomes des préconditions des actions :

$$Poss(avancer, s) \leftrightarrow$$
$$\neg\exists x \exists y\ (position(x, s) \wedge obstacle(y, s) \wedge y = x + 1) \quad (2.17)$$

$$Poss(reculer, s) \leftrightarrow$$
$$\neg\exists x \exists y\ (position(x, s) \wedge obstacle(y, s) \wedge y = x - 1) \quad (2.18)$$

$$Poss(enlever\_obs, s) \leftrightarrow$$
$$\exists x \exists y\ (position(x, s) \wedge obstacle(y, s) \wedge (y = x + 1 \vee y = x - 1)) \quad (2.19)$$

Les significations intuitives de chaque axiome sont les suivantes : 's'il n'y pas d'obstacle le robot peut avancer d'une unité', 's'il n'y pas d'obstacle le robot peut reculer d'une unité' et 'le robot peut enlever l'obstacle qui est placé juste devant ou derrière'.

- Les axiomes de changement d'état sont :

$$position(x, do(a, s)) \leftrightarrow$$
$$[a = avancer \wedge position(x-1, s) \vee a = reculer \wedge position(x + 1, s)]$$
$$\vee position(x, s) \wedge \neg[(a = avancer \vee a = reculer) \wedge position(x, s)] \quad (2.20)$$

$$obstacle(x, do(a, s)) \leftrightarrow$$
$$obstacle(x, s) \wedge \neg(a = enlever\_obs) \quad (2.21)$$

- Les axiomes de la situation initiale sont :

$$position(5, S_0) \quad (2.22)$$
$$obstacle(7, S_0) \quad (2.23)$$

On peut déduire $\exists s\ position(8, s)$ comme suit :

1. $position(6, do(avancer, S_0))$ de 2.20 et 2.22
2. $\neg obstacle(7, do(enlever\_obs, do(avancer, S_0)))$ de 2.21 et 1
3. $position(7, do(avancer, do(enlever\_obs, do(avancer, S_0))))$ de 2.20 et 2
4. $position(8, do(avancer, do(avancer, do(enlever\_obs, do(avancer, S_0)))))$ de 2.20 et 3

Dans la preuve précédente, on a utilisé implicitement les axiomes d'unicité de noms des actions :

$$\neg(avancer = reculer),\ \ \neg(avancer = enlever\_obs),\ \ \neg(reculer = enlever\_obs).$$

Si $s = do(avancer, do(avancer, do(enlever\_obs, do(avancer, S_0))))$, alors on a $\exists s\ position(8, s)$. On peut considérer s comme un plan qui permet au robot d'atteindre la position 8. La signification intuitive est 'le robot avance jusqu'à la position 6, puis enlève l'obstacle se trouvant à la position 7 et finalement avance deux fois pour arriver à la position 8'.

Le plan est obtenu comme une conséquence adjacente des preuves des théorèmes. Ainsi une solution au problème de la planification peut être énoncée comme suit : prouver l'existence d'une situation s qui satisfasse la formule G.

$$\Sigma \vdash \exists x\ G(s) \tag{2.24}$$

où Σ est l'ensemble d'axiomes de préconditions des actions, de changement d'état, de la situation initiale et d'unicité de noms des actions.

Maintenant, supposons que l'on trouve le plan suivant : $s' = do(avancer, do($ $avancer, do(avancer, S_0)))$, c'est-à-dire 'le robot avance trois fois pour arriver à la position 8'. Or on peut démontrer $position(6, do(avancer, S_0))$ à partir de 2.20 et 2.22, et $obstacle(7, do(avancer, S_0))$ à partir de 2.21 et 2.23, alors on a :

$$\exists x \exists y\ (position(x, do(avancer, S_0)) \wedge obstacle(y, do(avancer, S_0)) \wedge y = x + 1)$$

en conséquence on a $\neg Poss(avancer, do(avancer, S_0))$, c'est-à-dire que le plan ne peut pas être exécuté à partir de la deuxième action.

Le premier plan s est exécutable tandis que le deuxième s' ne l'est pas. Un plan exécutable vérifie toutes les préconditions des actions qui le composent. Ainsi on dit que s est exécutable car on peut démontrer :

$$\begin{aligned} &Poss(avancer, S_0)\ \wedge\ Poss(enlever\_obs, do(avancer, S_0))\ \wedge \\ &Poss(avancer, do(enlever\_obs, do(avancer, S_0)))\ \wedge \\ &Poss(avancer, do(avancer, do(enlever\_obs, do(avancer, S_0)))) \end{aligned}$$

Autrement dit, l'action d'*avancer* est possible dans la situation initiale, l'action d'*enlever_obs* est possible dans la situation résultant d'*avancer*, l'action d'*avancer* après les deux premières actions est encore possible, et finalement il est toujours possible d'*avancer* après les trois premières actions. Par contre, on dit que s' n'est pas exécutable car on ne peut pas démontrer :

$$\begin{aligned} &Poss(avancer, S_0)\ \wedge\ Poss(avancer, do(avancer, S_0))\ \wedge \\ &Poss(avancer, do(avancer, do(avancer, S_0))) \end{aligned}$$

Alors la formule (2.24) permet de trouver des plans qui ne satisfont pas les *contraintes* du monde réel spécifiées par les axiomes de préconditions des actions. L'exécution de ce type de plan risque de ne pas aboutir car l'exécution de toutes les actions n'est pas assurée. Trouver un plan qui ait la possibilité d'aboutir peut être obtenu comme conséquence adjacente de la preuve d'existence d'une situation s qui satisfasse la formule G (le but à atteindre) et qui peut être exécutée (cf. Section 2.3.3) :

$$\Sigma \vdash \exists x\ (ex\acute{e}cutable(s) \wedge G(s)) \tag{2.25}$$

où Σ est l'ensemble d'axiomes de préconditions des actions, de changement d'état, de la situation initiale et d'unicité de noms des actions.

Cette formule permet de trouver des plans qui ont la possibilité d'aboutir, pourtant elle ne garantit pas l'achèvement de tels plans. Rien n'indique que le plan a été exécuté.

Un autre intérêt du calcul des situations est son pouvoir d'expression pour caractériser des théories de l'action.

Chapitre 3

Calcul des situations étendu aux croyances

Les phénomènes sont les choses
telles que nous les connaissons,
les noumènes sont les choses en soi,
telles qu'elles sont indépendamment
de la connaissance que nous avons.
(Kant)

Pour ne pas rentrer dans des problèmes philosophiques, supposons que le calcul des situations tel qu'il est présenté dans le chapitre précédent permet de modéliser les faits dans un monde changeant, ce que Kant appelle noumènes. On peut critiquer cette affirmation en disant que du fait que l'on a utilisé l'intellect pour les modéliser, ils abandonnent leur catégorie de noumène pour devenir phénomènes selon Kant. Mais il ne faut pas confondre les *connaissances du modélisateur* avec les *connaissances des agents modélisés*. Dans le premier cas, les connaissances se placent hors du système, celui-ci les accepte comme des vérités absolues. Dans le deuxième cas, les connaissances se placent dans "la tête" de chaque agent composant le système, et celui-ci accepte l'existence des connaissances erronées, des contradictions entre les connaissances des agents, des connaissances incomplètes, etc.

Dans ce chapitre nous nous intéressons au deuxième genre de connaissance. On les appelle croyances puisqu'elles sont des vérités relatives, c'est-à-dire des vérités particulières à chaque agent. On montrera deux manières d'étendre le calcul des situations pour considérer aussi les croyances des agents.

Les interactions des agents font évoluer éventuellement leurs croyances. Comment modéliser ces changements produits dans la "tête" de chaque agent? On propose deux solutions. La première s'appuie sur une solution proposée par les chercheurs du Cognitive Robotic Group de Toronto pour un seul agent [SL93, LL98, SPLL00], dans laquelle les croyances sont exprimées en termes d'une relation d'accessibilité (RA). Nous proposons une extension afin de résoudre les problèmes posés dans le cas de plusieurs agents. La seconde solution introduit des opérateurs modaux (OM). Les deux solutions proposées sont des extensions du calcul des situations.

3.1 Calcul des situations étendu aux croyances d'un agent en termes d'une relation d'accessibilité

Dans [Moo80, Moo85], on trouve pour la première fois l'introduction d'un fluent pour la représentation des croyances dans le calcul des situations, cependant cette approche ne propose pas de solution au frame problem.

Ce sera dans [SL93] que les auteurs étendent le calcul des situations afin de considérer les changements des croyances d'un agent.

D'un point de vue formel cette extension repose sur l'idée d'exprimer en logique classique du premier ordre la définition de la modalité de croyance. Pour cela le langage est étendu avec un fluent binaire $B(s', s)$ dont les arguments sont du type *situation*[1]. Ce fluent joue le même rôle qu'une relation d'accessibilité dans la sémantique des mondes possibles. La différence est qu'il relie des situations au lieu de mondes possibles, et qu'il est introduit dans l'axiomatique et non dans la sémantique.

Un agent *imagine* un ensemble de mondes qui représentent tous les mondes dans lesquels il peut être. Si ψ est *vrai* dans tous les mondes imaginés, alors on dit que l'agent "croit" ψ.

Le fait que dans une situation s un agent croit que ϕ est *vraie* est dénoté par $Bel(\phi, s)$. Ici ϕ est une formule obtenue à partir d'une formule du calcul des situations dans laquelle tous les arguments des fluents de type *situation* ont été supprimés. On suppose qu'ils concernent tous la situation s. On note $\phi[s]$ la formule obtenue à partir de ϕ en rajoutant l'argument s à chaque "fluent"[2] pour récupérer ainsi son format initial.

Par exemple, à partir de la formule $cassé(X1, s) \wedge \neg cassé(X2, s)$ on obtient $\phi_1 = cassé(X1) \wedge \neg cassé(X2)$, et $Bel(\phi_1, s)$ dénote 'dans s l'agent croit que $cassé(X1) \wedge \neg cassé(X2)$ est *vraie*'. La formule initiale avec la substitution de la situation s par s_1 est dénotée[3] par $\phi_1[s_1]$, et $\phi_1[s_1] = cassé(X1, s_1) \wedge \neg cassé(X2, s_1)$.

La signification attribuée à $Bel(cassé(X1) \wedge \neg cassé(X2), s)$ est que dans la situation s l'agent croit que dans toutes les situations s' compatibles avec ses croyances (c'est-à-dire telles que $B(s', s)$), on a $cassé(X1, s') \wedge \neg cassé(X2, s')$ qui est *vraie*. C'est pour cette raison que l'on impose que les croyances portent sur des formules dans lesquelles les arguments de type *situation* se réfèrent tous à la situation s qui apparaît dans $Bel(\phi, s)$.

La définition de Bel est la suivante :

$$Bel(\phi, s) \stackrel{\text{def}}{=} \forall s'(B(s', s) \rightarrow \phi[s'])$$

d'où

$$Bel(cassé(X1) \wedge \neg cassé(X2), s) \stackrel{\text{def}}{=} \forall s'(B(s', s) \rightarrow cassé(X1, s') \wedge \neg cassé(X2, s')).$$

[1] L'usage dans une relation d'accessibilité $R(w, w')$ veut que le monde w' représente une alternative au monde w par rapport à la croyance [Che88]. Nous avons respecté ici la notation utilisée dans [Rei99], où l'ordre des arguments est inversé afin d'attribuer la dernière position à la situation correspondante du monde réel.

[2] En réalité, ils ne sont plus de fluents puisque on a supprimé les arguments de type *situation*.

[3] Il ne faut pas confondre cette notation avec celle qui est présentée dans l'Annexe A.

Les formules telles que ϕ peuvent elles-mêmes contenir des modalités. On peut avoir, par exemple, $Bel(\neg Bel(cassé(X2)), S_0)$ qui exprime qu'en S_0 l'agent croit qu'il ne croit pas que $X2$ est cassé. D'après la définition précédente on a :

$$\begin{aligned} Bel(\neg Bel(cassé(X2)), S_0) &\stackrel{\text{def}}{=} \forall s'(B(s', s) \rightarrow \neg Bel(cassé(X2), s')) \\ &= \forall s'(B(s', s) \rightarrow \neg(\forall s''(B(s'', s') \rightarrow cassé(X2, s'')))) \\ &= \forall s'(B(s', s) \rightarrow \exists s''(B(s'', s') \wedge \neg cassé(X2, s''))). \end{aligned}$$

Décomposition de l'évolution des croyances

Jusqu'à présent on a fait l'hypothèse implicite que les actions transforment seulement l'état du monde réel (ou externe). A la différence de cette sorte d'actions, les actions d'acquisition d'information ne modifient pas le monde extérieur, mais plutôt les état mentaux (ou internes) des agents. C'est pour cette raison que dans [SL93], les auteurs les appellent *actions qui produisent des connaissances.*

Pour définir la façon dont les croyances évoluent quand des actions sont réalisées on distingue deux aspects.

Le premier concerne l'évolution de la relation d'accessibilité $B(s', s)$. Il faut définir les situations s'' qui sont accessibles après avoir réalisé l'action a à partir de s, c'est-à-dire telles que $B(s'', do(a, s))$. Pour cela on distingue deux types d'actions :

- celles qui permettent d'acquérir de l'information sur le monde. On les appelle "actions de perception" ("sensing action" en anglais). Chaque action de perception permet à l'agent de savoir si une formule ϕ associée à cette action est vraie, ou fausse, dans la situation où il se trouve. Ces actions ont pour effet d'éliminer des situations considérées comme possibles mais qui deviennent inconsistantes après l'ajout de l'information acquise par l'observation. On suppose que ces actions peuvent modifier les croyances mais qu'elles ne modifient pas le monde (révision).

- celles qui modifient le monde. Pour chaque situation possible elles projettent leurs effets sur la situation suivante, et éventuellement elles modifient les croyances en conséquence (mise-à-jour).

Le deuxième aspect concerne la détermination des valeurs de vérité des fluents dans les situations s'' accessibles depuis $do(a, s)$ par B. Pour cela on utilise les axiomes de changement d'état tels que (2.5).

Nous allons illustrer ces deux aspects avec l'exemple suivant. Supposons que l'on a un verre, que dans toute situation, si l'on réalise l'action *tomber*, il en résulte que le verre est cassé, et que si l'on réalise l'action *réparer*, il en résulte que le verre n'est plus cassé. Le fait que le verre est cassé dans la situation s est représenté par : $cassé(s)$. On a alors l'axiome de changement d'état de *cassé* suivant :

$$\forall s \forall a(cassé(do(a, s)) \leftrightarrow a = tomber \vee cassé(s) \wedge \neg(a = réparer)). \qquad (3.1)$$

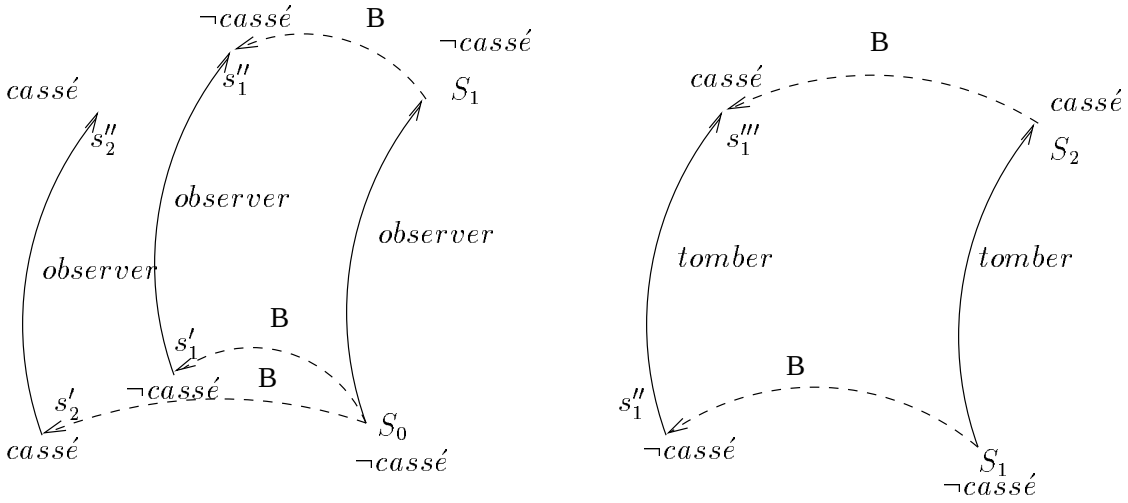

Figure 3.1: Evolution des croyances d'un agent. Action de perception et action sur le monde.

Maintenant, on considère l'action *observer* qui permet de savoir dans une situation donnée si le verre est cassé, ou non. Supposons, par exemple, qu'en S_0 l'agent ignore si le verre est cassé, et que le verre ne soit pas cassé. On a alors : $\neg Bel(cassé, S_0) \wedge \neg Bel(\neg cassé, S_0) \wedge \neg cassé(S_0)$.

Il y a donc une situation s_1' et une situation s_2' accessibles depuis S_0 où on a respectivement $\neg cassé(s_1')$ et $cassé(s_2')$. Le fait qu'après avoir réalisé *observer* l'agent sait que le verre n'est pas cassé, c'est-à-dire que l'on a $Bel(\neg cassé, do(observer, S_0))$, est exprimé formellement par le fait que les seules situations s' accessibles depuis S_0 qui ont un successeur $s'' = do(observer, s')$ accessible depuis $do(observer, S_0)$ sont celles pour lesquelles la valeur de vérité de *cassé* est la même en s' qu'en S_0. Dans l'exemple de la Figure 3.1 c'est le cas des situations telles que s_1'. Plus généralement on a :

$$\forall s \forall s'' \forall a (a = observer \rightarrow$$
$$(B(s'', do(a, s)) \leftrightarrow \exists s' (B(s', s) \wedge s'' = do(a, s') \wedge (cassé(s') \leftrightarrow cassé(s))))).$$

Comme on suppose que l'action *observer* ne change pas la valeur de vérité des fluents, dans toutes les situations telles que $s_1'' = do(observer, s_1')$ on a $\neg cassé$ qui est *vraie*. On a donc $Bel(\neg cassé, do(observer, S_0))$.

Soit S_1 la situation $do(observer, S_0)$. En S_1, après avoir réalisé une action qui n'est pas une action de perception, par exemple *tomber*, les nouvelles croyances de l'agent sont déterminées par les situations accessibles depuis $S_2 = do(tomber, S_1)$. Si, par exemple s_1'' est accessible depuis S_1, alors $s_1''' = do(tomber, s_1'')$ est accessible depuis S_2. On a donc $\forall s'''(B(s''', S_2) \leftrightarrow \exists s''(B(s'', S_1) \wedge s''' = do(tomber, s'')))$. Plus généralement on a :

$$\forall s \forall s'' \forall a (\neg(a = observer) \rightarrow$$
$$(B(s'', do(a, s)) \leftrightarrow \exists s' (B(s', s) \wedge s'' = do(a, s')))).$$

D'autre part la valeur de vérité de $cassé(do(tomber, s_1''))$ est déterminée par l'axiome de changement d'état 3.1. D'après celui-ci on a $\forall s(cassé(do(tomber, s)))$. Comme toutes les situations s''' accessibles depuis S_2 sont telles que $s''' = do(tomber, s'')$, on a $cassé(s''')$ qui est *vraie* dans toutes ces situations. On a donc $Bel(cassé, S_2)$.

La forme générale de l'axiome définissant l'évolution de la relation d'accessibilité B est la suivante :

$$\begin{aligned}\forall s \forall s'' \forall a(B(s'', do(a, s)) \quad \leftrightarrow \quad & \exists s'(B(s', s) \wedge s'' = do(a, s') \wedge (\\ & (\neg(a = \alpha_1) \wedge \ldots \wedge \neg(a = \alpha_n)) \\ & \vee\ a = \alpha_1 \wedge (\phi_1(s) \leftrightarrow \phi_1(s')) \\ & \vdots \\ & \vee\ a = \alpha_n \wedge (\phi_n(s) \leftrightarrow \phi_n(s')))) \qquad (3.2)\end{aligned}$$

dans lequel chaque α_i dénote une action de perception qui permet de connaître la valeur de vérité de la formule ϕ_i.

Hypothèses implicites

On peut remarquer que dans cette formalisation de l'évolution des croyances on fait implicitement deux hypothèses.

La première est que chaque fois qu'une action a été réalisée l'agent sait qu'elle a été réalisée. En effet, les croyances d'un agent évoluent chaque fois qu'une action a été réalisée. Dans le cas où on ne considère qu'un seul agent, et qu'il n'y a pas d'action exogène (réalisée par l'environnement ou par un autre agent), l'hypothèse est acceptable, mais dans le cas où on considère plusieurs agents, elle ne l'est plus.

La deuxième hypothèse est que l'agent sait quels sont les effets des actions. En effet, l'évolution de ses croyances est déterminée, entre autres, par les axiomes de changement d'état qui sont les mêmes en s et en s' telles que $B(s', s)$. Là aussi, dans le cas de plusieurs agents, il se peut que certains agents croient que les choses évoluent d'une certaine manière, et que d'autres croient qu'elles évoluent d'une manière différente. Donc, dans de nombreux cas cette hypothèse n'est pas acceptable.

C'est pour ne pas être contraint par ces deux hypothèses que nous proposons deux formalisations. Dans la section suivante nous montrons une variante de cette approche dans laquelle l'extension du calcul des situations est donnée en termes de relations d'accessibilité. L'autre approche est donnée plus loin, elle étend le calcul des situations en termes d'opérateurs modaux.

3.2 Calcul des situations étendu aux croyances de plusieurs agents en termes de relations d'accessibilité

Pour exprimer formellement le fait que les croyances de chaque agent peuvent évoluer de façons différentes on suppose qu'il y a une relation d'accessibi-

lité B_i pour chaque agent i. Ceci permet de définir une modalité $Bel_i(\phi, s)$ qui représente ce que croit l'agent i.

La définition de Bel_i est la suivante :

$$Bel_i(\phi, s) \stackrel{\text{def}}{=} \forall s'(B_i(s', s) \rightarrow \phi[s'])$$

Pour ne pas être contraint par les hypothèses implicites que nous avons mentionnées dans la section précédente on considère que :

- l'ensemble des actions dont la réalisation est connue par un agent est donné explicitement, et peut être différent d'un agent à l'autre,
- les axiomes de changement d'état dépendent des types de situations auxquelles on les applique. Selon qu'une situation représente une situation réelle, ou une situation correspondant aux croyances de tel ou tel agent, on peut avoir des axiomes de changement d'état différents.

Nous allons voir, d'abord sur un exemple, comment ces deux principes peuvent être exprimés formellement.

On considère maintenant deux agents : Nathalie et son petit enfant. Les actions que peut réaliser Nathalie sont : $tomber_N$, $réparer_N$ et $observer_N$, et celles que peut réaliser l'enfant sont : $tomber_e$ et $observer_e$. On a les relations d'accessibilité B_N et B_e pour représenter respectivement les croyances de Nathalie et celles de l'enfant.

On suppose que les actions dont la réalisation est connue de Nathalie sont $tomber_N$, $réparer_N$, $observer_N$ et $tomber_e$, celles connues de l'enfant sont : $tomber_e$, $observer_e$ et $réparer_N$.

Dans [LLR99], Lespérance, Levesque et Reiter proposent une solution dans laquelle un agent est toujours (sauf dans les situations qui suivent immédiatement une observation) dans l'ignorance des faits qui ont pu être modifiés par des actions dont il n'a pas connaissance.

Selon cette proposition, dans notre exemple, comme l'enfant ignore si la mère a réalisé l'action $tomber_N$, même s'il vient d'observer que le verre n'est pas cassé, il n'exclut pas, dans la situation suivante, que Nathalie l'ait laissé tombé et qu'il soit cassé. Et donc il sera dans l'ignorance du fait qu'il est cassé, ou non. Cette attitude nous paraît excessivement prudente, ou sceptique.

Dans la solution que nous présentons on considère, à l'inverse, qu'un agent raisonne comme si les actions dont il n'a pas connaissance n'avaient pas eu lieu. Ceci revient pour lui à ignorer ces actions tant qu'il n'a pas observé leurs effets. C'est une attitude plus confiante, ou crédule.

3.2.1 Axiomes de changement d'état des relations d'accessibilité

Dans notre proposition, le fait que la réalisation de l'action $tomber_N$ n'est pas connue de l'enfant est exprimé formellement par le fait que les situations accessibles depuis $do(tomber_N, s)$ par la relation B_e sont les mêmes que celles accessibles depuis s par B_e. Cela signifie intuitivement que les situations imaginées par l'enfant sont les mêmes en s et en $do(tomber_N, s)$, et donc que ses

croyances sont inchangées après la réalisation de $tomber_N$. Il en est de même pour l'action $observer_N$ qui n'est pas connue de l'enfant. Formellement on a:

$$\forall s \forall s'' \forall a((a = tomber_N \vee a = observer_N) \rightarrow (B_e(s'', do(a, s)) \leftrightarrow B_e(s'', s))).$$

Pour les actions dont la réalisation est connue de l'enfant l'évolution des croyances est définie de la façon suivante:

$$\begin{aligned}\forall s \forall s'' \forall a((\neg(a = tomber_N) \wedge \neg(a = observer_N)) \rightarrow \\ (B_e(s'', do(a, s)) \leftrightarrow \exists s'(B_e(s', s) \wedge s'' = do(a, s') \wedge \\ (a = tomber_e \vee a = réparer_N \vee \\ a = observer_e \wedge (cassé(s) \leftrightarrow cassé(s'))))))\end{aligned}$$

Pour représenter l'évolution des croyances quel que soit le type d'action on a alors:

$$\begin{aligned}\forall s \forall s'' \forall a(B_e(s'', do(a, s)) \quad \leftrightarrow \quad & (B_e(s'', s) \wedge \neg(a = tomber_e) \wedge \\ & \neg(a = observer_e) \wedge \neg(a = réparer_N)) \vee \\ & \exists s'(B_e(s', s) \wedge s'' = do(a, s') \wedge \\ & (a = tomber_e \vee a = réparer_N \vee \\ & a = observer_e \wedge (cassé(s) \leftrightarrow cassé(s')))))\end{aligned}$$

On a de la même manière l'axiome suivant pour représenter l'évolution des croyances de Nathalie:

$$\begin{aligned}\forall s \forall s'' \forall a(B_N(s'', do(a, s)) \quad \leftrightarrow \quad & (B_N(s'', s) \wedge \neg(a = tomber_N) \wedge \neg(a = réparer_N) \wedge \\ & \neg(a = observer_N) \wedge \neg(a = tomber_e)) \vee \\ & \exists s'(B_N(s', s) \wedge s'' = do(a, s') \wedge \\ & (a = tomber_N \vee a = réparer_N \vee a = tomber_e \vee \\ & a = observer_N \wedge (cassé(s) \leftrightarrow cassé(s')))))\end{aligned}$$

La forme générale de l'axiome définissant l'évolution d'une relation B_i est:

$$\begin{aligned}\forall s \forall s'' \forall a(B_i(s'', do(a, s)) \quad \leftrightarrow \quad & (B_i(s'', s) \wedge \neg(a = \alpha_1) \wedge \ldots \wedge \neg(a = \alpha_n) \wedge \\ & \neg(a = \beta_1) \wedge \ldots \wedge \neg(a = \beta_m)) \vee \\ & \exists s'(B_i(s', s) \wedge s'' = do(a, s') \wedge \\ & (a = \beta_1 \vee \ldots \vee a = \beta_m \\ & \vee\ a = \alpha_1 \wedge (\phi_1(s) \leftrightarrow \phi_1(s')) \\ & \vdots \\ & \vee\ a = \alpha_n \wedge (\phi_n(s) \leftrightarrow \phi_n(s'))))) \qquad (3.3)\end{aligned}$$

où les alphas sont les actions de perception réalisées par l'agent i, et les bétas sont les autres actions dont la réalisation est connue de l'agent i. La forme de cet axiome pourrait être généralisée en définissant l'ensemble des actions connues par l'agent i dans la situation s à l'aide d'une formule $\psi_i(a, s)$, au lieu de donner explicitement ces actions. La formule $\psi_i(a, s)$ pourrait exprimer, par exemple, que a est connue de i en s si l'action a est réalisée par un agent qui est dans la même pièce que i ou qui est dans un voisinage proche.

3.2.2 Axiomes de changement d'état particularisés

Pour déterminer l'évolution des croyances il faut aussi donner les axiomes de changement d'état qui s'appliquent aux situations accessibles par B_N ou B_e.

Supposons, par exemple, qu'en S_0 le verre ne soit pas cassé, et que Nathalie et l'enfant sachent que le verre n'est pas cassé. Soit $\neg casse(S_0) \wedge Bel_N(\neg casse, S_0) \wedge Bel_e(\neg casse, S_0)$. Dans les situations s'_1 et s'_2 respectivement accessibles par B_e et B_N on a donc $\neg casse(s'_1)$ et $\neg casse(s'_2)$ (voir Figure 3.2).

Si l'enfant croit que le verre ne se casse pas si l'on le laisse tomber, et qu'en S_0 il fasse tomber le verre (action $tomber_e$), alors après avoir fait tomber le verre (i.e. dans la situation $do(tomber_e, S_0)$) il croit que le verre n'est pas cassé. Donc dans les situations telles que $do(tomber_e, s'_1)$, qui sont accessibles depuis $do(tomber_e, S_0)$ par B_e, on doit avoir $\neg casse(do(tomber_e, s'_1))$. Par contre, si Nathalie croit que le verre est cassé après l'avoir laissé tomber, alors en $do(tomber_e, S_0)$ elle croit que le verre est cassé. Donc, dans les situations telles que $do(tomber_e, s'_2)$, accessibles depuis $do(tomber_e, S_0)$ par B_N, on doit avoir $casse(do(tomber_e, s'_2))$.

Il faut donc distinguer l'axiome de changement d'état qui s'applique en s'_1 et celui qui s'applique en s'_2. Pour cela on introduit les prédicats $croit_N(s)$ et $croit_e(s)$ qui expriment intuitivement qu'en s les axiomes de changement d'état qui s'appliquent sont ceux qui correspondent respectivement aux croyances de Nathalie et de l'enfant. Ces axiomes peuvent être, par exemple :

$$\forall s(croit_N(s) \rightarrow (casse(do(a,s)) \leftrightarrow a = tomber_e \vee a = tomber_N \vee casse(s) \wedge \neg(a = reparer_N))) \qquad (S_{casse,N})$$

$$\forall s(croit_e(s) \rightarrow (casse(do(a,s)) \leftrightarrow casse(s) \wedge \neg(a = reparer_N))) \qquad (S_{casse,e})$$

Pour définir les situations où s'appliquent les axiomes de changement d'état correspondant à la réalité on introduit le prédicat $reel(s)$, qui signifie intuitivement que s représente une situation réelle. On a, par exemple :

$$\forall s(reel(s) \rightarrow (casse(do(a,s)) \leftrightarrow a = tomber_e \vee a = tomber_N \vee casse(s) \wedge \neg(a = reparer_N))) \qquad (S_{reel})$$

Dans le cas général, pour chaque fluent p et pour chaque agent i, ou pour chaque fluent p et pour les situations réelles, on doit avoir un axiome de changement d'état de la forme:

$$\forall s \forall a(croit_i(s) \rightarrow ((p(do(a,s)) \leftrightarrow \Gamma^+_{p,i}(a,s) \vee p(s) \wedge \neg\Gamma^-_{p,i}(a,s)) \qquad (S_{p,i})$$

$$\forall s \forall a(reel(s) \rightarrow ((p(do(a,s)) \leftrightarrow \Gamma^+_{p,0}(a,s) \vee p(s) \wedge \neg\Gamma^-_{p,0}(a,s)) \qquad (S_{reel})$$

3.2.3 Type des situations

Supposons maintenant qu'en S_0 l'enfant croit que Nathalie croit que le verre n'est pas cassé, et que Nathalie croit que l'enfant croit lui aussi que le verre n'est pas cassé. Soit $Bel_e(Bel_N(\neg casse), S_0)$ et $Bel_N(Bel_e(\neg casse), S_0)$. On a donc $Bel_N(\neg casse, s'_1)$ et $Bel_e(\neg casse, s'_2)$.

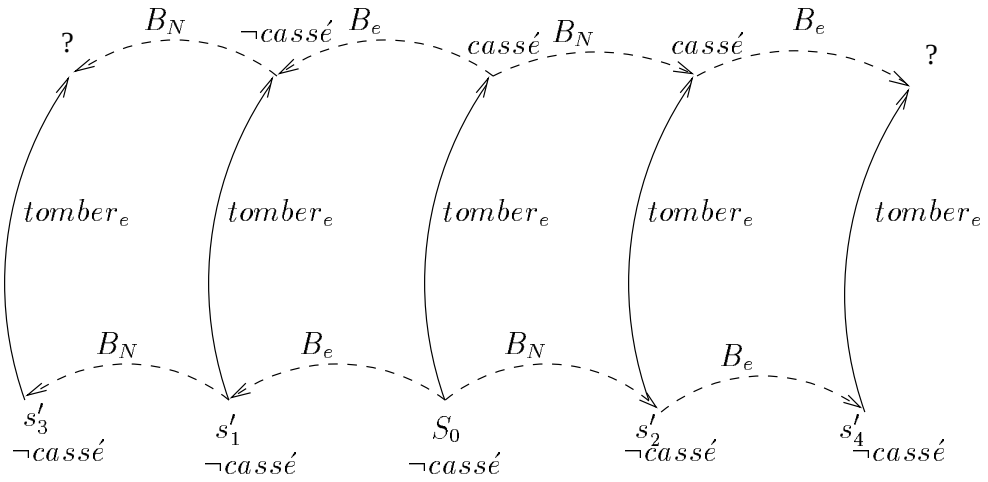

Figure 3.2: Evolution des croyances respectives des agents.

Soient s'_3 et s'_4 des situations respectivement accessibles depuis s'_1 par B_N et depuis s'_2 par B_e (voir Figure 3.2), la question qui se pose est alors : faut-il considérer s'_3 comme une situation qui représente des croyances de Nathalie, ou comme une situation qui représente des croyances de l'enfant sur les croyances de Nathalie ?

Si l'on considère que s'_3 représente une croyance de Nathalie, soit $croit_N(s'_3)$, alors l'axiome de changement d'état qui s'applique en s'_3 est $(S_{cassé,N})$. Mais ceci revient à supposer que l'enfant croit que sa mère croit que si l'on laisse tomber un verre il se casse, bien que lui croit le contraire. Cela pourrait être le cas, mais c'est peu vraisemblable. Il faut donc pouvoir éventuellement représenter que ce n'est pas le cas, c'est-à-dire qu'en s'_3 c'est $(S_{cassé,e})$ qui s'applique, et que l'on a $croit_e(s'_3)$.

Si on considère maintenant que s'_4 représente les croyances de la mère sur les croyances de l'enfant, on peut accepter que Nathalie sache que l'enfant croit que le verre ne se casse pas s'il tombe, et dans ce cas on acceptera d'appliquer en s'_4 l'axiome $(S_{cassé,e})$.

D'un point de vue pratique, comment peut-on exprimer la nature des situations ? On ne peut pas, pour chaque situation s, donner l'information $croit_i(s)$, car il y a une infinité de situations. Il paraît naturel cependant de considérer que les agents croient que les effets des actions sont toujours les mêmes, c'est-à-dire que les situations qui succèdent à une situation s sont de même nature que celle-ci. Ceci s'exprime formellement par l'axiome :

$$\forall s(croit_i(s) \leftrightarrow croit_i(do(a,s))) \qquad (CR_1)$$

Mais même en se limitant aux situations accessibles depuis la situation initiale, il en reste une infinité. On propose alors d'attribuer aux agents deux attitudes possibles qui peuvent être décrites par un nombre très limité de faits de la forme $croit_i(s)$.

La première suppose que chaque agent i sait comment évoluent les croyances d'un autre agent j. Par exemple, le cas de Nathalie vis-à-vis de l'enfant, elle croit que son enfant croit qu'un verre ne se casse pas si l'on le laisse tomber. Cette attitude s'exprime formellement par le fait que la nature d'une situation est déterminée uniquement par la relation d'accessibilité qui accède à cette situation. Par exemple, on a $croit_N(s'_2)$ parce que s'_2 est accessible par B_N, et $croit_e(s'_4)$ parce que s'_4 est accessible par B_e. Dans le cas général on a alors le schéma d'axiome :

$$\forall s \forall s' (B_i(s', s) \rightarrow croit_i(s')) \qquad (CR_2)$$

On notera que l'axiome (CR_2), même s'il ne résout pas de manière générale les problèmes de connaissances mutuelles mentionnés par Fagin et al. dans [FHMV95], il permet de déduire des formules de la forme :

$$Bel_{i_1}(\ldots(Bel_{i_p}(croit_{i_p}))\ldots, s).$$

En effet, de $\forall s \forall s'_p (B_{i_p}(s'_p, s) \rightarrow croit_{i_p}(s'_p))$ on peut déduire :

$$\forall s \forall s'_1 \ldots \forall s'_p (B_{i_1}(s'_1, s'_2) \wedge \ldots \wedge B_{i_{p-1}}(s'_{p-1}, s'_p) \wedge B_{i_p}(s'_p, s) \rightarrow croit_{i_p}(s'_p)).$$

La deuxième approche suppose que chaque agent i croit que les croyances des autres agents évoluent comme ses propres croyances. Par exemple, c'est le cas de l'enfant qui croit que sa mère croit, comme lui, qu'un verre ne se casse pas si l'on le laisse tomber. Dans ce cas la nature d'une situation sera déterminée par l'"agent réel" qui imagine les croyances des autres agents. Par "agent réel" on veut dénoter celui qui est dans une situation réelle et non dans une situation imaginée par un autre agent. Dans ce cas on a, par exemple, $croit_e(s'_1)$ parce que s'_1 est accessible par B_e depuis S_0 et que l'on a $réel(S_0)$, et on a $croit_e(s'_3)$, bien que la relation qui accède à s'_3 soit B_N, parce que s'_3 est accessible depuis s'_1 et que l'on a $croit_e(s'_1)$. Dans le cas général on a alors les schémas d'axiomes :

$$\forall s \forall s' (réel(s) \wedge B_i(s', s) \rightarrow croit_i(s')) \qquad (CR_3)$$
$$\forall s \forall s' (croit_i(s) \wedge B_j(s', s) \rightarrow croit_i(s')) \qquad (CR_4)$$

Dans le cas où le concepteur qui doit formaliser une application connaît l'évolution des croyances des agents, mais ne connaît pas les croyances qu'ont les agents sur l'évolution des croyances des autres agents, il peut faire l'hypothèse, par défaut, que les agents ont la deuxième attitude ci-dessus. Il se peut aussi que les agents eux-mêmes adoptent cette attitude, par défaut, quand ils ne connaissent pas l'évolution des croyances des autres agents. Par exemple, si l'enfant ignore comment évoluent les croyances de sa mère.

Que l'on choisisse l'une ou l'autre de ces attitudes, il ne doit y avoir qu'un seul axiome de changement d'état par fluent qui s'applique à une situation donnée. On doit donc avoir les schémas d'axiomes suivants pour i différent de j:

$$\neg\exists s (croit_i(s) \wedge croit_j(s)) \qquad (CR_5)$$
$$\neg\exists s (réel(s) \wedge croit_j(s)) \qquad (CR_6)$$

3.2.4 Révision des croyances

La formalisation proposée dans [SL93] et [LL98] pose un problème pour la révision des croyances. Considérons l'exemple du verre cassé. Si en S_0 l'agent croit que le verre n'est pas cassé $Bel(\neg \mathit{cassé}, S_0)$, alors on a $\forall s'(B(s', s) \rightarrow \neg \mathit{cassé}(s'))$. Or comme on a $\mathit{cassé}(S_0)$, on a donc $\neg\exists s'(B(s', s) \wedge (\mathit{cassé}(S_0) \leftrightarrow \mathit{cassé}(s'))$, et d'après (3.2) il n'y a aucune situation accessible depuis $do(\mathit{observer}, S_0)$ par B. La conséquence est que pour n'importe quelle formule ϕ on a $Bel(\phi, do(\mathit{observer}, S_0))$ et $Bel(\neg\phi, do(\mathit{observer}, S_0))$. Ce qui évidemment n'est pas acceptable.

Dans [SPLL00] Shapiro, Pagnucco, Lespérance et Levesque ont proposé une nouvelle version de cette formalisation dans laquelle on n'a plus ce problème. L'idée est d'assigner à chaque situation s un degré de plausibilité défini par la fonction $pl(s)$. On définit d'abord une relation d'accessibilité B_{max} qui accède aux situations les plus plausibles (celles pour lesquelles $pl(s')$ est minimal) $B_{max}(s', s) \stackrel{\text{def}}{=} B(s', s) \wedge \forall s''(B(s'', s) \rightarrow pl(s') \leq pl(s''))$. On peut alors définir une nouvelle modalité de croyance $Bel(\phi, s)$ avec B_{max} :

$$Bel(\phi, s) \stackrel{\text{def}}{=} \forall s'(B_{max}(s', s) \rightarrow \phi[s'])$$

De plus on suppose que toutes les situations qui succèdent à une situation donnée ont la même plausibilité, soit : $\forall s \forall a(pl(do(a, s)) = pl(s))$.

Si l'on considère l'exemple décrit sur la Figure 3.3, on a $Bel(\neg \mathit{cassé}, S_0)$ car dans toutes les situations accessibles de S_0 où $pl(s')$ est minimal on a $\neg \mathit{cassé}(s')$. Après avoir réalisé l'action $\mathit{observer}$, comme on a $\mathit{cassé}(S_0)$, les seules situations s' qui ont un successeur sont celles pour lesquelles on a $\mathit{cassé}(s')$. Dans l'exemple de la Figure 3.3, pour ces situations s' la valeur minimale de la plausibilité est 2, et pour les situations qui leur succèdent elle reste égale à 2. Donc, pour toutes les situations accessibles par B_{max} depuis $do(\mathit{observer}, S_0)$ on a $\mathit{cassé}(do(\mathit{observer}, s'))$, et on a $Bel(\mathit{cassé}, do(\mathit{observer}, S_0))$, comme souhaité.

Par contre, si dans toutes les situations s' telles que $B(s', S_0)$ on a $\neg \mathit{cassé}(s')$, alors il n'y a pas de situation accessible depuis $do(\mathit{observer}, S_0)$, et l'agent a des croyances inconsistantes. Pour que ce cas ne se produise pas il faut rajouter des conditions supplémentaires (voir Théorème 3 dans [SPLL00]).

L'extension du formalisme que nous venons de voir peut être appliquée directement à la formalisation présentée. Il suffit d'introduire autant de fonctions $pl_i(s)$ qu'il y a d'agents i, et de définir B_{max_i} par :

$$B_{max_i}(s', s) \stackrel{\text{def}}{=} B_i(s', s) \wedge \forall s''(B_i(s'', s) \rightarrow pl_i(s') \leq pl_i(s''))$$

La modalité $Bel_i(\phi, s)$ est ici définie par :

$$Bel_i(\phi, s) \stackrel{\text{def}}{=} \forall s'(B_{max_i}(s', s) \rightarrow \phi[s']).$$

3.2.5 Dépendance des croyances

Il faut noter aussi que dans la formalisation précédente on peut résoudre le problème de la révision, mais le frame problem n'a pas de solution évidente.

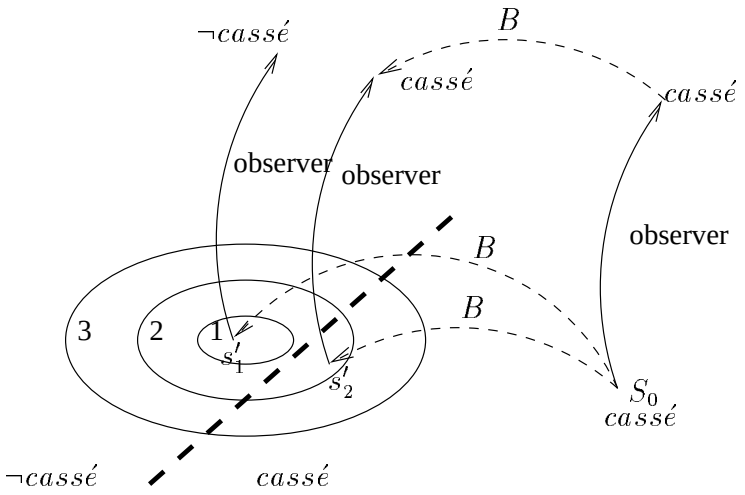

Figure 3.3: Evolution de croyances ayant différents degrés de plausibilité.

Supposons, par exemple, qu'en S_0 l'agent croit que le verre n'est pas cassé et qu'il est froid, soit $Bel(\neg cassé, S_0)$ et $Bel(froid, S_0)$, et que parmi les situations s' les plus plausibles ($pl(s')$ minimal) où on a $cassé(s')$ on ait $froid(s'_2)$ pour l'une d'entre elles, et $\neg froid(s'_3)$ pour une autre. Dans ce cas on a $froid(do(observer, s'_2))$ et $\neg froid(do(observer, s'_3))$, et les situations $do(observer, s'_2)$ et $do(observer, s'_3)$ font partie des situations les plus plausibles accessibles depuis $do(observer, S_0)$. Donc on a $\neg Bel(froid, do(observer, S_0))$ et $\neg Bel(\neg froid, do(observer, S_0))$. Ce qui montre que la croyance $Bel(froid, S_0)$ n'est pas persistante après avoir réalisé l'action *observer*, alors qu'elle devrait l'être, car cette action, intuitivement, ne modifie pas la croyance de l'agent sur le fait que le verre est froid.

Remarque: Le problème de la mise-à-jour des croyances (cf. Section 1.2.5) est résolu implicitement par l'axiome de changement d'état de la relation d'accessibilité. Dans l'approche présentée dans la section suivante ce problème est résolu explicitement par les axiomes de changement d'état des opérateurs modaux.

Le problème de la révision des croyances dans [SL93, LL98] est plutôt une expansion de croyances qui est un cas particulier de révision. La révision permet à un agent de croire ψ, d'acquérir des informations qui font changer ses croyances en $\neg\psi$ (sans que le monde ait changé) et même plus tard de croire à nouveau ψ. Ces changements ne sont pas envisagés dans les approches présentées dans [SL93, LL98].

3.3 Calcul des situations étendu aux croyances de plusieurs agents en termes d'opérateurs modaux

Dans [DP00] on propose une extension du calcul des situations pour considérer l'évolution des croyances des agents. Cette extension est exprimée en

termes d'opérateurs modaux.

Dans cette approche on considère que l'évolution du monde physique ou réel n'est pas parallèle à celle des croyances d'un agent (hypothèses implicite dans les approches précédentes). Par exemple, dans le monde physique, le prédicat *obstacle* peut être *vrai*, cependant le robot peut croire qu'il est $faux$, i.e. $\neg B_r(obstacle)$. Pour raisonner sur l'évolution du monde des croyances, on étend le langage avec des opérateurs épistémiques modaux. Ainsi on introduit des opérateurs, tels que B_i, où $B_i(\phi)$ signifie intuitivement que l'agent i croit ϕ dans la situation actuelle.

3.3.1 Axiomes de changement d'état des croyances

En général, un agent peut avoir quatre attitudes face à ses croyances concernant une proposition, par exemple : $B_i(obstacle)$: i croit qu'il existe un obstacle, $B_i(\neg obstacle)$: i croit qu'il n'existe pas d'obstacle, $\neg B_i(obstacle)$: i ne croit pas qu'il existe un obstacle, $\neg B_i(\neg obstacle)$: i ne croit pas qu'il n'existe pas d'obstacle.

Pour chaque fluent et chaque agent, on a donc quatre axiomes exprimant les quatre attitudes possibles de l'agent par rapport au fluent. Par exemple, pour le robot r et le fluent $position(x, s)$, on a : la croyance positive des effets positifs $B_r(position(x, s))$, la croyance négative des effets positifs $\neg B_r(position(x, s))$, la croyance positive des effets négatifs $B_r\neg(position(x, s))$ et la croyance négative des effets négatifs $\neg B_r(\neg position(x, s))$.

L'axiome (3.4) qui exprime la croyance positive des effets positifs, signifie intuitivement que si le robot exécute *avancer* (respectivement *reculer*) dans la situation s, et s'il croit que la position dans la situation s est $x - 1$ (respectivement $x + 1$), alors son attitude est de croire que la position est x dans la situation $do(a, s)$.

$$[a = avancer \wedge B_r(position(x-1, s)) \vee a = reculer \wedge B_r(position(x+1, s))] \rightarrow B_r(position(x, do(a, s))) \quad (3.4)$$

L'axiome (3.5) de la croyance négative des effets positifs, exprime que si le robot croit qu'il est dans la position x dans la situation s, et s'il exécute *avancer* ou *reculer*, alors il ne croit pas qu'il est dans la position x dans la situation $do(a, s)$.

$$(a = avancer \vee a = reculer) \wedge B_r(position(x, s)) \rightarrow \neg B_r(position(x, do(a, s))) \quad (3.5)$$

L'axiome (3.6) de la croyance positive des effets négatifs, exprime que si le robot croit qu'il est dans la position x dans la situation s, et s'il exécute *avancer* ou *reculer*, alors il croit qu'il n'est pas dans la position x dans la situation $do(a, s)$.

$$(a = avancer \vee a = reculer) \wedge B_r(position(x, s)) \rightarrow B_r(\neg position(x, do(a, s))) \quad (3.6)$$

L'axiome (3.7) de la croyance négative des effets négatifs, signifie intuitivement que si le robot exécute *avancer* (respectivement *reculer*) dans la situation s, et s'il croit que la position dans la situation s est $x-1$ (respectivement $x+1$), alors il ne croit pas que la position n'est pas x dans la situation $do(a,s)$.

$$[a = avancer \wedge B_r(position(x-1,s)) \vee a = reculer \wedge B_r(position(x+1,s))] \rightarrow \neg B_r(\neg position(x, do(a,s))) \quad (3.7)$$

On peut appliquer la méthode de Reiter pour déduire l'axiome de changement d'état des croyances des effets positifs (cf. format général (3.12)). Ainsi à partir de (3.4), (3.5) et de l'hypothèse de la complétude des causes on obtient :

$$\begin{aligned} &B_r(position(x, do(a,s))) \leftrightarrow \\ &\quad [a = avancer \wedge B_r(position(x-1,s)) \vee a = reculer \wedge B_r(position(x+1,s))] \\ &\quad \vee B_r(position(x,s)) \wedge \neg[(a = avancer \vee a = reculer) \wedge B_r(position(x,s))] \end{aligned} \quad (3.8)$$

De façon similaire, on peut déduire l'axiome de changement d'état des croyances des effets négatifs (cf. format général en (3.13)). A partir de (3.6), (3.7) et de l'hypothèse de la complétude des causes on obtient donc :

$$\begin{aligned} &B_r(\neg position(x, do(a,s))) \leftrightarrow \\ &\quad [(a = avancer \vee a = reculer) \wedge B_r(position(x,s))] \vee \\ &\quad B_r(position(x,s)) \wedge \neg[a = avancer \wedge B_r(position(x-1,s)) \vee \\ &\quad a = reculer \wedge B_r(position(x+1,s))] \end{aligned} \quad (3.9)$$

Avec les formules (3.8) et (3.9) on peut raisonner sur l'évolution du monde des croyances du robot par rapport à sa position. On appelle cette sorte de formules *axiomes de changement d'état des croyances* (ACEC).

Des questions telles que : Est-ce qu'après avoir exécuté l'action *observer_obs*, le pilote sait s'il existe ou non un obstacle? La position du robot est-elle la même que celle que croit le pilote? S'il existe un obstacle, le pilote le sait-il? Le robot ignore-t-il l'existence des obstacles? Etc., peuvent être résolues en utilisant les ACEC.

Des concepts tels que l'ignorance du robot concernant la position sont représentés en utilisant les opérateurs modaux comme suit : $\neg B_r(position(x,s)) \wedge \neg B_r(\neg position(x,s))$, qui signifie intuitivement 'le robot ne sait pas si la position est x, ni si la position n'est pas x'. Si l'on enlève la négation aux opérateurs modaux, la formule qui en résulte $B_r(position(x,s)) \wedge B_r(\neg position(x,s))$ exprime une contradiction dans les croyances. Si l'on veut assurer la consistance de ce que pense l'agent, les formules représentant des contradictions doivent être écartées de la théorie (cf. Section 3.3.2).

3.3.2 La logique du calcul des situations étendu aux croyances

Langage du calcul des situations étendu au croyances

Le langage du calcul des situations étendu aux croyances de plusieurs agents $\mathcal{L}_{\mathcal{CS}_B}$, dont l'extension est exprimée en termes d'opérateurs modaux est composé du langage du calcul des situations $\mathcal{L}_{\mathcal{CS}}$ (cf. Section 2.3.1) plus :

11. Un ensemble d'opérateurs modaux dénotés par B_i, où $B_i p$ signifie que l'agent i croit *toujours* p, p est un littéral. Ce qu'exprime ces opérateurs est indépendant des situations. On peut les imaginer comme une sorte de prédicats qui maintiennent leur extension "constante", qui ne changent pas après l'exécution des actions.

12. Un ensemble d'opérateurs modaux dénotés aussi par B_i, où $B_i p(s)$ signifie que dans la situation s l'agent i crois que p est *vraie* dans la situation s (cf. Annexe A), $p(s)$ est un fluent ou la négation d'un fluent[4]. Ces formules dépendent des situations. On peut imaginer les opérateurs comme une sorte de prédicats qui changent leur extension éventuellement après l'exécution d'une action.

Composants des théories de l'action et des croyances

Pour simplifier, on considère que les fonctions sont représentées sous forme de prédicats, par exemple, $y = g(\vec{x})$ est représenté par $G(\vec{x}, y)$; où $\vec{x} = x_1, x_2, \ldots, x_n$. Aussi on ne spécifie que les arguments représentant des actions et des situations, par exemple, l'action $a(x)$ et le fluent $F(x, y, s)$ sont représentés respectivement comme a et $F(s)$.

Les théories considérées $\mathcal{T}$ contiennent les axiomes suivants :

- *Les axiomes fondamentaux des situations* $\mathcal{S}$ (cf. Axiomes 2.12 à 2.15). Ceux-ci permettent de déduire que tout terme de la forme $do(a, s)$ doit être une situation générée à partir de S_0. Les axiomes fondamentaux permettent aussi de déduire l'unicité des noms pour les situations.

- *Les axiomes des préconditions des actions* $\mathcal{D}_{pa}$. Pour chaque action a, dans $\mathcal{T}$ il existe un axiome de la forme :

$$Poss(a, s) \leftrightarrow \pi_a(s) \tag{3.10}$$

  où $\pi_a(s)$ est une formule dans $\mathcal{LCS}_{\mathcal{B}}$ qui est uniforme pour la situations s (cf. Section 2.4).

- *Les axiomes de changement d'état des fluents* $\mathcal{D}_{ce}$. Pour chaque fluent F, dans $\mathcal{T}$ il existe un axiome de la forme :

$$F(do(a, s)) \leftrightarrow \Upsilon_F^+(a, s) \vee F(s) \wedge \neg\Upsilon_F^-(a, s) \tag{3.11}$$

  où $\Upsilon_F^+(a, s) \vee F(s) \wedge \neg\Upsilon_F^-(a, s)$ est une formule dans $\mathcal{LCS}_{\mathcal{B}}$ qui est uniforme pour la situation s.

- *Les axiomes de changement d'état des croyances* $\mathcal{D}_{cec}$. Pour chaque fluent F et chaque agent i, dans $\mathcal{T}$ il existe deux axiomes de la forme[5] :

$$B_i(F(do(a, s))) \leftrightarrow \Upsilon_{B_i F}^+(a, s) \vee B_i(F(s)) \wedge \neg\Upsilon_{B_i F}^-(a, s) \tag{3.12}$$

[4] Pour simplifier on omet toutes les variables sauf celle qui représente la situation.

[5] Pour chaque fonction fluent il y a aussi un axiome similaire (cf. [Rei99]).

où $\Upsilon^{+}_{B_iF}(a,s) \vee B_i(F(s)) \wedge \neg\Upsilon^{-}_{B_iF}(a,s)$ est une formule dans $\mathcal{L}_{\mathcal{CS}_B}$ qui est uniforme pour la situation s, et

$$B_i(\neg F(do(a,s))) \leftrightarrow \Upsilon^{+}_{B_i\neg F}(a,s) \vee B_i(\neg F(s)) \wedge \neg\Upsilon^{-}_{B_i\neg F}(a,s) \qquad (3.13)$$

où $\Upsilon^{+}_{B_i\neg F}(a,s) \vee B_i(\neg F(s)) \wedge \neg\Upsilon^{-}_{B_i\neg F}(a,s)$ est une formule dans $\mathcal{L}_{\mathcal{CS}_B}$ qui est uniforme pour la situation s.

- *Les axiomes d'unicité des noms pour les actions* $\mathcal{D}_{una}$.
- *Les axiomes décrivant la situation initiale* $\mathcal{D}_{S_0}$.

Ainsi, une théorie de l'action et des croyances (dépendante du domaine) est un ensemble d'axiomes $\mathcal{T}$,

$$\mathcal{T} = \mathcal{S} \cup \mathcal{D}_{pa} \cup \mathcal{D}_{ce} \cup \mathcal{D}_{cec} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}.$$

On suppose que l'opérateur B_i satisfait la logique KD (cf. Annexe B.4).

On a privilégié la définition de l'axiomatique par rapport à la sémantique car notre but est de faire de la déduction automatique.

En ce qui concerne la sémantique on peut dire, de façon très intuitive, que la notion de situation correspond à celle de monde possible.

Consistance

La définition de l'axiome de changement d'état d'un fluent F impose que les conditions $\Upsilon^{+}_{F}(a,s)$ et $\Upsilon^{-}_{F}(a,s)$ soient disjointes pour éviter des contradictions (cf. Section 2.2.3), i.e. pour éviter d'avoir F et $\neg F$ *vraies* simultanément. Dans le cas des ACEC, les contradictions ont lieu au niveau de la modalité. Ainsi, par exemple des formules telles que $B_r(position(x,s)) \wedge \neg B_r(position(x,s))$ représente une contradiction des croyances. Intuitivement cela veut dire que le robot *croit* que la position est x et en même temps il *ne croit pas* que la position est x. Il faut donc imposer des conditions pour éviter cette sorte de contradiction.

Il y a une autre sorte de contradiction qui peut se présenter dans les croyances. Un exemple est la formule $B_r(position(x,s)) \wedge B_r(\neg position(x,s))$, qui veut dire intuitivement que le robot croit que la *position est* x et simultanément il croit que la *position n'est pas* x.

Pour garantir la consistance (l'absence des contradictions) de $\mathcal{T}$, les six conditions suivantes doivent être satisfaites pour chaque fluent F et chaque agent i (on suppose toutes les variables libres quantifiées universellement)

$$\neg\left(\Upsilon^{+}_{F}(a,s) \wedge \Upsilon^{-}_{F}(a,s)\right) \qquad (3.14)$$

$$\neg\left(\Upsilon^{+}_{B_iF}(a,s) \wedge \Upsilon^{-}_{B_iF}(a,s)\right) \qquad (3.15)$$

$$\neg\left(\Upsilon^{+}_{B_i\neg F}(a,s) \wedge \Upsilon^{-}_{B_i\neg F}(a,s)\right) \qquad (3.16)$$

$$\neg\left(\Upsilon^{+}_{B_iF}(a,s) \wedge \Upsilon^{+}_{B_i\neg F}(a,s)\right) \qquad (3.17)$$

$$B_iF(s) \wedge \Upsilon^{+}_{B_i\neg F}(a,s) \rightarrow \Upsilon^{-}_{B_iF}(a,s) \qquad (3.18)$$

$$B_i\neg F(s) \wedge \Upsilon^{+}_{B_iF}(a,s) \rightarrow \Upsilon^{-}_{B_i\neg F}(a,s) \qquad (3.19)$$

On peut démontrer que si $\Upsilon^{+}_{F}(a,s)$ et $\Upsilon^{-}_{F}(a,s)$ sont inconsistantes (3.14) alors l'axiome (3.11) est équivalent aux quatre propriétés suivantes (cf. Section 2.2.3) :

$$\Upsilon^{+}_{F}(a,s) \rightarrow F(do(a,s))$$
$$\Upsilon^{-}_{F}(a,s) \rightarrow \neg F(do(a,s))$$
$$\neg\Upsilon^{-}_{F}(a,s) \rightarrow [F(s) \rightarrow F(do(a,s))]$$
$$\neg\Upsilon^{+}_{F}(a,s) \rightarrow [\neg F(s) \rightarrow \neg F(do(a,s))]$$

De la même façon on peut démontrer que si $\Upsilon^{+}_{B_iF}(a,s)$ et $\Upsilon^{-}_{B_iF}(a,s)$ sont inconsistantes (3.15) alors l'axiome (3.12) est équivalent aux quatre propriétés suivantes :

$$\Upsilon^{+}_{B_iF}(a,s) \rightarrow B_i(F(do(a,s)))$$
$$\Upsilon^{-}_{B_iF}(a,s) \rightarrow \neg B_i(F(do(a,s)))$$
$$\neg\Upsilon^{-}_{B_iF}(a,s) \rightarrow [B_i(F(s)) \rightarrow B_i(F(do(a,s)))]$$
$$\neg\Upsilon^{+}_{B_iF}(a,s) \rightarrow [\neg B_i(F(s)) \rightarrow \neg B_i(F(do(a,s)))].$$

et si $\Upsilon^{+}_{B_i\neg F}(a,s)$ et $\Upsilon^{-}_{B_i\neg F}(a,s)$ sont inconsistantes (3.16) alors l'axiome (3.13) est équivalent aux quatre propriétés suivantes :

$$\Upsilon^{+}_{B_i\neg F}(a,s) \rightarrow B_i(\neg F(do(a,s)))$$
$$\Upsilon^{-}_{B_i\neg F}(a,s) \rightarrow \neg B_i(\neg F(do(a,s)))$$
$$\neg\Upsilon^{-}_{B_i\neg F}(a,s) \rightarrow [B_i(\neg F(s)) \rightarrow B_i(\neg F(do(a,s)))]$$
$$\neg\Upsilon^{+}_{B_i\neg F}(a,s) \rightarrow [\neg B_i(\neg F(s)) \rightarrow \neg B_i(\neg F(do(a,s)))].$$

La condition (3.17) garantie l'absence des formules du type $B_i(F(s)) \wedge B_i(\neg F(s))$.

On peut démontrer à partir de (3.17), (3.18) et (3.19), par récurrence, que si les croyances de l'agent sont consistantes à l'état initial alors elles sont consistantes dans toutes les états successeurs.

Les conditions (3.18) et (3.19) assurent la satisfaction du schéma D de la logique KD pour B_i. La signification intuitive de (3.18) est que si l'agent croit que F est *vrai* dans s et si les conditions pour croire que $\neg F$ est *vrai* sont satisfaites alors les conditions pour ne pas croire F doivent être satisfaites. La signification intuitive de (3.19) est que si l'agent croit que $\neg F$ est *vrai* dans s et si les conditions pour croire que F est *vrai* sont satisfaites alors les conditions pour ne pas croire $\neg F$ doivent être satisfaites.

On n'impose pas de conditions particulières pour satisfaire le schéma K de la logique KD puisque dans la logique que l'on a définie les opérateurs modaux ne portent que sur des littéraux et le schéma K inclut des expressions composées, par exemple $B_i(F_1 \rightarrow F_2) \wedge B_i(F_1) \rightarrow B_i(F_2)$.

Définition des préconditions

Etant donné que pour les théories de l'action on fait l'hypothèse que tous les changements sont provoqués par des actions, les ACEC ont donc toujours des

termes contenant des actions. Et les actions sont exécutées si et seulement si certaines préconditions sont satisfaites.

On utilise le fluent *Poss* pour représenter les préconditions. Intuitivement $Poss(a, s)$ signifie que dans la situation s il est possible d'exécuter l'action a. Par exemple, le robot peut avancer dans la situation s si et seulement s'il n'existe pas d'obstacle qui l'empêche d'avancer :

$$Poss(avancer, s) \leftrightarrow \neg\exists x\,(position(x, s) \wedge obstacle(x + 1, s))$$

Maintenant on peut considérer aussi des états mentaux, par exemple le robot peut avancer si et seulement s'il croit qu'il n'existe pas d'obstacle devant lui, et si *vraiment* il n'existe pas d'obstacle qui l'empêche d'avancer :

$$\begin{aligned} &Poss(avancer, s) \leftrightarrow \\ &\quad \neg\exists x\,(B_r(position(x, s)) \wedge B_r(obstacle(x + 1, s))) \wedge \\ &\quad \neg\exists x\,(position(x, s) \wedge obstacle(x + 1, s)). \end{aligned}$$

Dans cet exemple, le rôle des croyances concernant l'exécution de l'action n'est pas clair car on peut voir les croyances non comme des préconditions pour avancer, mais plutôt comme le *choix* que fait l'agent d'avancer seulement quand il croit qu'il n'y a pas d'obstacle. Les croyances n'empêchent pas *vraiment* d'avancer, c'est l'agent qui décide de ne pas avancer s'il croit qu'il y a un obstacle. Un autre exemple, plus convaincant de la nécessité d'inclure des état mentaux dans les conditions à satisfaire pour la réalisation d'une action est celui de l'appel téléphonique (pris de [Rei99]) :

$$\begin{aligned} &Poss(appel(x, y), s) \leftrightarrow \\ &\quad \neg\exists t \exists n\,(B_x(numéro\_tél(y, n, s)) \wedge téléphone(t) \wedge proche(x, t)). \end{aligned}$$

où $appel(x, y)$ signifie intuitivement que la personne x appelle la personne y, $numéro\_tél(y, n, s)$ signifie que la personne y a comme numéro téléphonique n dans la situation s, $téléphone(t)$ signifie que t est un téléphone et $proche(x, t)$ signifie que x est à côté de t.

L'absence de conditions est permise. Par exemple, le pilote peut toujours regarder la position du robot, sans condition explicite.

$$Poss(observer\_pos) \leftrightarrow true.$$

3.3.3 Opérateur de régression étendu aux croyances

Définition 1 *(Formule de régression).* Soit W une formule qui satisfait les conditions suivantes :

- Tout terme de type *situation* apparaissant dans W est de la forme

$$do(a_1, do(a_2 \ldots, do(a_n, S_0) \ldots)).$$

- Si $Poss(\alpha, \sigma)$ apparaît dans W, alors α est un terme de la forme $A(t_1, \ldots, t_n)$.

- Dans W il n'y a pas de quantification sur les variables de type *situation*.
- Dans W l'égalité ne porte pas sur des situations.

Alors W est une *formule de régression.*

Reiter a défini, dans [Rei99], un opérateur de régression R sur lequel se base un des résultats les plus importants de sa théorie. On a aussi défini un opérateur de régression R_T en ajoutant à R la régression dans le cas des littéraux modaux. La définition de l'opérateur de régression étendu aux croyances est la suivante :

Définition 2 *(Opérateur de régression).* On définit l'*opérateur de régression* R_T des formules de régression de $\mathcal{L}_{\mathcal{CS}_B}$ vers des formules de régression de $\mathcal{L}_{\mathcal{CS}_B}$ comme suit :

(1) Si W est un atome qui n'est pas un fluent (y compris les atomes de égalité) ou si W est un atome qui est un fluent et qui a comme argument de type *situation* la constante $S0$, alors l'opérateur de régression R_T est défini comme suit :

$$R_T[W] = W.$$

(2) Si W est une formule constituée de l'atome $Poss$ et si W a la forme $Poss(A, \sigma)$ telle que l'axiome de préconditions pour A dans $\mathcal{T}$ est : $Poss(A, s) \leftrightarrow \pi_A(s)$ (cf. Axiome 3.10), alors l'opérateur de régression R_T est défini comme suit :

$$R_T[W] = R_T[\pi_a(\sigma)].$$

$\pi_A(\sigma)$ représente une instance de la formule $\pi_A(s)$ dans laquelle on a substitué toutes les occurrences de s par σ. S'il y a d'autres arguments on suppose aussi la substitution de ces arguments par leurs termes respectifs.

(3) Si W est un atome constitué du fluent F et si W a la forme[6] $F(do(\alpha, \sigma))$, telle que l'axiome de changement d'état dans $\mathcal{T}$ est : $F(do(a, s)) \leftrightarrow \Phi_F(a, s)$ (cf. Axiome 3.11), alors l'opérateur de régression R_T est défini comme suit :

$$R_T[W] = R_T[\Phi_F(\alpha, \sigma)].$$

$\Phi_F(\alpha, \sigma)$ représente une instance de la formule $\Phi_F(a, s)$ dans laquelle on a substitué toutes les occurrences de a par α et toutes les occurrences de s par σ. S'il y d'autres arguments on suppose aussi la substitution de ces arguments par leurs termes respectifs.

(4) Si W est un atome de la forme $B_i F(do(\alpha, \sigma))$ ou $B_i \neg F(do(\alpha, \sigma))$, et si les axiomes de changement d'état des croyances dans $\mathcal{T}$ sont :

$$B_i F(do(a, s)) \leftrightarrow \Phi_{B_i F}(a, s) \text{ et}$$

$$B_i \neg F(do(a, s)) \leftrightarrow \Phi_{B_i \neg F}(a, s),$$

[6] Pour simplifier la notation, on omet tous les arguments sauf celui qui représente la situation.

alors l'opérateur de régression R_T est défini comme suit :

$$R_T[W] = R_T[\Phi_{B_iF}(\alpha, \sigma)] \text{ et}$$

$$R_T[W] = R_T[\Phi_{B_i\neg F}(\alpha, \sigma)].$$

$\Phi_{B_iF}(\alpha, \sigma)$ et $\Phi_{B_i\neg F}(\alpha, \sigma)$ représentent les instances des formules $\Phi_F(a, s)$ et $\Phi_{B_i\neg F}(a, s)$ respectivement, dans lesquelles on a substitué toutes les occurrences de a par α et toutes les occurrences de s par σ. S'il y d'autres arguments on suppose aussi la substitution de ces arguments par leurs termes respectifs.

(5) Si W est une formule de $\mathcal{L}_{\mathcal{CS}_\mathcal{B}}$, alors l'opérateur de régression R_T pour la négation $\neg$ et le quantificateur universel $\forall$ est défini comme suit :

$$R_T[\neg W] = \neg R_T[W] \text{ et}$$

$$R_T[\forall x W] = \forall x R_T[W].$$

(6) Si W_1 and W_2 sont deux formules de $\mathcal{L}_{\mathcal{CS}_\mathcal{B}}$, alors l'opérateur de régression R_T pour la disjonction $\vee$ est défini comme suit :

$$R_T[W_1 \vee W_2] = R_T[W_1] \vee R_T[W_2].$$

La définition de R_T pour le quantificateur existentiel $\exists$, la conjonction $\wedge$, l'implication $\rightarrow$ et l'équivalence $\leftrightarrow$ est obtenue à partir de la définition du quantificateur et des connecteurs logiques (cf. Annexe B.2) en fonction de $\forall$, $\neg$ et $\vee$.

Intuitivement, les fluents, l'atome *Poss* ou les croyances qui contiennent le terme de type *situation* $do(\alpha, \sigma)$ sont remplacés par des formules qui contiennent le terme de type *situation* σ. Le processus est répété jusqu'à obtention des formules telles que le seul terme de type *situation* qu'elles contiennent soit S_0.

Théorème 1 Soient W une formule de régression qui ne contient pas de variable libre, $\mathcal{T}$ une théorie comme celles que l'on a définie dans la Section 3.3.2, $\mathcal{D}_{S_0}$ l'ensemble des axiomes décrivant la situation initiale, $\mathcal{D}_{una}$ l'ensemble d'axiomes d'unicité des noms pour les actions et $R_T^*[W]$ la formule qui résulte de l'application itérative de R_T jusqu'à ce que le résultat soit inchangé. Si $\mathcal{T}' = \mathcal{D}_{S_0} \cup \mathcal{D}_{una}$, alors

$$\models \mathcal{T} \rightarrow W \iff \models \mathcal{T}' \rightarrow R_T^*[W]$$

La démonstration du Théorème 1 utilise la même méthode que celle qui a été utilisée par Scherl et Levesque dans la démonstration du Théorème 5 en [SL93]. La preuve se fait par induction sur la profondeur des termes qui représentent les situations. Ici la preuve est plus simple car on n'explicite pas de relations d'accessibilité pour représenter les opérateurs modaux.

On exige que W soit une formule de régression afin d'assurer que l'application itérative de l'opérateur R_T produira une formule sans termes de type *action* et où le seul terme de type *situation* apparaissant soit S_0.

Au niveau de l'implémentation, ce résultat est intéressant car il permet de réduire l'évaluation des formules de régression sans variables libres à des démonstrations de théorèmes du premier ordre dans la théorie initiale $\mathcal{D}_{S_0}$, évitant ainsi l'emploi des axiomes de changement d'état, des fluents et des croyances, et les axiomes des préconditions des actions.

Le Théorème 1 est utilisé pour vérifier qu'une propriété est satisfaite après l'exécution d'une séquence d'actions. Une autre application permet de vérifier qu'une séquence d'actions est exécutable, c'est-à-dire vérifier, pour chaque action, qu'après l'exécution d'une telle action, les préconditions pour réaliser l'action suivante sont satisfaites. Ces deux applications sont nécessaires pour le problème de la génération de plans.

Par exemple, si l'on applique l'opérateur R_T à $B_r(position(6, do(reculer, do($ $avancer, S_0))))$, d'après les ACEC de on obtient :

$$
\begin{aligned}
&R_T[B_r(position(6, do(reculer, do(avancer, S_0))))] = \\
&R_T[reculer = avancer \wedge B_r(position(5, do(avancer, S_0))) \vee \\
&reculer = reculer \wedge B_r(position(7, do(avancer, S_0))) \vee \\
&\neg(reculer = avancer \vee reculer = reculer) \wedge B_r(position(6, do(avancer, S_0)))] = \\
&reculer = avancer \wedge R_T[B_r(position(5, do(avancer, S_0)))] \vee \\
&reculer = reculer \wedge R_T[B_r position(7, do(avancer, S_0))] \vee \\
&\neg(reculer = avancer \vee reculer = reculer) \wedge R_T[B_r(position(6, do(avancer, S_0)))].
\end{aligned}
$$

De la même façon, on obtient :

$$
\begin{aligned}
&R_T[B_r(position(5, do(avancer, S_0)))] = \\
&avancer = avancer \wedge R_T[B_r(position(4, S_0))] \vee \\
&avancer = reculer \wedge R_T[B_r(position(6, S_0))] \vee \\
&\neg(avancer = avancer \vee avancer = reculer) \wedge R_T[B_r(position(5, S_0))],
\end{aligned}
$$

$$
\begin{aligned}
&R_T[B_r(position(7, do(avancer, S_0)))] = \\
&avancer = avancer \wedge R_T[B_r(position(6, S_0))] \vee \\
&avancer = reculer \wedge R_T[B_r position(8, S_0)] \vee \\
&\neg(avancer = avancer \vee avancer = reculer) \wedge R_T[B_r(position(7, S_0))]
\end{aligned}
$$

et

$$
\begin{aligned}
&R_T[B_r(position(6, do(avancer, S_0)))] = \\
&avancer = avancer \wedge R_T[B_r(position(5, S_0))] \vee \\
&avancer = reculer \wedge R_T[B_r(position(7, S_0))] \vee \\
&\neg(avancer = avancer \vee avancer = reculer) \wedge R_T[B_r(position(6, S_0))].
\end{aligned}
$$

En simplifiant, par exemple, l'atome $reculer = avancer$ est $faux$ d'après les axiomes d'unicité de noms, on obtient donc :

$$R_T[B_r position(6, do(reculer, do(avancer, S_0)))] = B_r position(6, S_0).$$

Ce qui intuitivement veut dire 'le robot croit qu'il se trouve dans la position 6, après avoir avancé et reculé, s'il croit qu'il était dans la position 6 à l'état initial'.

3.4 Avantages et inconvénients des approches

L'approche qui utilise la relation d'accessibilité ne résout pas de façon satisfaisante le problème de la révision des croyances. S'il n'existe pas une situation accessible par la relation à partir d'une situation qui résulte d'une action de perception, alors l'agent aura des croyances contradictoires (cf. Section 3.2.4).

Un solution est d'étendre cette approche en utilisant l'idée proposée dans [SPLL00], i.e. en introduisant autant de fonctions $pl_i(s)$ qu'il y a d'agents. Mais cette extension ne propose pas de solution au problème de la dépendance des croyances.

La dépendance des croyances créée par la relation d'accessibilité n'est pas réaliste. Par exemple, en principe le fait 'verre cassé' n'a aucune relation avec le fait 'verre froid'. Cependant dans cette approche la dépendance est créée implicitement (cf. Section 3.2.5).

Un des inconvénients de l'approche qui utilise la relation d'accessibilité est donc qu'elle rend la modélisation complexe. L'introduction des possibilités et la dépendance entre les croyances font perdre la simplicité de la solution au frame problem que Reiter a proposé au début dans [Rei91].

Ces problèmes ne se présentent pas dans l'approche qui utilise des opérateurs modaux. La simplicité de la solution proposée par Reiter est retrouvée dans cette solution. Le frame problem a une solution évidente, en définissant explicitement l'évolution de chacune des quatre attitudes de l'agent pour créer les axiomes appropriés. Cette façon de construire les axiomes de changement d'état des croyances permet d'avoir une indépendance entre les faits. Ainsi par exemple, les conditions qui définissent la croyance 'verre cassé' sont indépendantes des conditions qui définissent la croyance 'verre froid'.

Par contre, l'approche qui utilise des opérateurs modaux a une restriction forte, qui est similaire à la celle que Reiter impose aux axiomes de changement d'état des fluents. Le pouvoir d'expression est limité. On accepte seulement des littéraux dans le champ des opérateurs modaux. Par exemple, supposons qu'un agent entend le bruit que fait un verre qui se casse et que l'agent sait qu'il existe deux verres $X1$ et $X2$, mais qu'il ne sait pas lequel a été cassé. Cela peut être décrit par $B_i(cassé(X1, s) \vee cassé(X1, s))$, mais dans cette approche on ne peut pas le représenter. Par contre, le pouvoir d'expression de l'approche qui utilise la relation d'accessibilité admet cette représentation.

Le choix de l'approche doit permettre de trouver un compromis entre le pouvoir d'expression et la solution proposée au problème de la révision et au frame problem.

Nous avons implétementé la deuxième approche pour le cas des théories complètes (cf. Section 5.1). A notre connaissance la première approche n'a pas été implémentée.

Partie II

Démonstration automatique

Introduction

Aristote dit de la logique, que "son sujet, c'est la démonstration", saint Thomas d'Aquin souligne que c'est "l'art qui dirige l'acte même de la raison, art par lequel nous procédons par ordre, facilement et sans erreur dans cet acte même de la raison". Et Church dit qu'elle "s'occupe de l'analyse des phrases ou des propositions et de celle des preuves, l'attention portant sur la forme par abstraction du contenu". L'idée commune est que la logique traite de la démonstration ou de la preuve du raisonnement.

Dans les chapitres précédents nous avons considéré la résolution des problèmes par le biais de la démonstration de théorèmes. L'automatisation des preuves est un sujet de recherche actuel. Nous envisageons la création de démonstrateurs *automatiques* de théorèmes. Pour cela, dans le reste des chapitres on envisage des procédures donnant des preuves dans des théories en général, et en particulier on montre des automatisations de preuves dans des théories proposées dans les chapitres précédents.

En particulier on décrit deux façons d'implémenter la logique du CS étendu aux croyances en termes d'opérateurs modaux. Leur différence est au niveau des axiomes décrivant la situation initiale. Dans la première implémentation, on fait l'hypothèse que la théorie initiale est complète. La deuxième considère aussi des théories initiales incomplètes.

Afin de considérer des théories incomplètes, on a construit un démonstrateur général dans des théories avec égalité (SOLE-résolution), qui est présenté dans le chapitre suivant.

Comme dans le langage de la logique du CS étendu aux croyances nous n'autorisons pas d'imbrication des croyances, on implémente l'opérateur de croyance à l'aide d'un métaprédicat en Prolog.

Chapitre 4

De Prolog vers un E-démonstrateur général

D'abord on souligne les avantages et les inconvénients que possède Prolog en ce qui concerne la démonstration automatique de théorèmes. Parmi les inconvénients, l'emploi de la négation par échec est éliminé dans un démonstrateur général tel que la SOL-résolution[1]. Avec ce dernier, la démonstration de théorèmes ne se limite pas au cas des théories complètes.

La SOL-résolution a l'avantage de générer des conséquences et pas seulement la clause vide comme le font d'autres démonstrateurs généraux.

Un autre problème qui se pose dans la démonstration automatique est celui du traitement de l'égalité, pour cela on propose un E-démonstrateur[2] général appelé SOLE-résolution.

Le reste du chapitre est divisé comme suit : dans la section suivante on fait un résumé de l'histoire de la démonstration automatique, ensuite on donne un aperçu de Prolog, puis on fait un rappel de la SOL-résolution, après on présente quelques travaux concernant la déduction automatique dans les théories avec égalité, puis on montre une façon de prendre en compte l'égalité dans la SOL-résolution, et finalement on présente la SOLE-résolution.

4.1 Antécédents historiques

La création d'une procédure générale déterminant si une formule est valide à été étudiée depuis Leibniz (1646-1716). L'idée sera reprise par Peano autour de la fin du $19^{ème}$ siècle et par l'école d'Hilbert dans les années 1920's. Ce ne seront que Church et Turing qui prouveront séparément qu'il n'existe pas une procédure générale déterminant la validité ou inconsistance des formules

[1] La connotation de SOL-résolution dans [Ino92a] est celle d'une méthode (procédure) de déduction et non d'une règle d'inférence. Ce qui peut créer de confusion si l'on veut faire une analogie avec la règle d'inférence de résolution.

[2] On ajoute le symbole "E-" au démonstrateur pour indiquer qu'il s'agit d'un démonstrateur de théories qui incluent l'*égalité*. On a choisi cette notation afin de s'ajuster aux notations de plusieurs auteurs qui rajoutent ce symbole aux notions déjà établies pour les théories du premier ordre afin de désigner les mêmes notions pour les théories du premier ordre avec l'égalité. Par exemple, E-interprétation, E-sémantique, E-arbre sémantique, règle d'E-inférence, etc.

de la logique des prédicats du premier ordre malgré l'existence des procédures permettant de prouver qu'une formule est valide si effectivement elle l'est. En ce qui concerne les formules non valides, les procédures, en général, ne finissent pas. C'est pour cette raison qu'il faut se contenter d'une procédure donnant des preuves des formules valides plutôt que d'une procédure générale déterminant si une formule est valide ou non.

Une approche importante à considérer à été donnée par Herbrand en 1930 [Her30]. Il développe un algorithme qui permet de trouver une interprétation qui falsifie une formule, mais si la formule est valide (formule vraie pour n'importe quelle interprétation, cf. Annexe C.1), alors il n'existe pas une telle interprétation et cet algorithme se terminera après un nombre fini d'essais. Plusieurs procédures automatiques donnant des preuves sont basées sur la méthode d'Herbrand.

Le principe de résolution, introduit par Robinson en 1965 [Rob65], a été la plus importante percée dans cette discipline. Les procédures donnant des preuves par résolution sont beaucoup plus efficaces que leurs précédentes. Depuis son introduction, le principe de résolution a bénéficié de raffinements qui offrent de meilleures performances. Parmi ceux-ci on peut citer, la résolution sémantique, "lock résolution" et la résolution linéaire.

4.2 Déduction automatique en Prolog

Prolog (Programmation en logique) est basé sur l'idée d'utiliser la logique comme langage de programmation. Fondé sur la logique des prédicats du premier ordre, Prolog[3] fait des manipulations symboliques basées sur le principe de résolution. Pour montrer qu'un théorème peut être déduit en utilisant Prolog il faut analyser le fonctionnement de ce langage. Il faut aussi considérer les termes employés et leur correspondance avec ceux de la démonstration classique de théorèmes. C'est le but de la section suivante.

4.2.1 Terminologie

Définition 3 *(Atome).* Un *atome* est une formule de la forme $P(t_1, \ldots, t_n)$, où P est un symbole de prédicat et t_i $(1 \leq i \leq n)$ est un terme.

Définition 4 *(Littéral).* Un *littéral* est un atome ou la négation d'un atome.

Définition 5 *(Clause).* Une *clause* est une formule de la forme $L_1 \vee \ldots \vee L_n$ oú L_i $(1 \leq i \leq n)$ est un littéral.

Définition 6 *(Clause de Horn).* Une clause qui a au plus un littéral positif est appelée une *clause de Horn.*

[3]Nous utilisons le terme 'Prolog' pour désigner le langage de programmation Prolog, un programme Prolog ou bien un interpréteur Prolog. Le contexte permettra d'identifier quelle des trois interprétations doit être utilisée.

Définition 7 *(Programme Prolog).* Un *programme Prolog* est une séquence finie de clauses[4]. Chaque clause dans Prolog a une des deux formes, soit une *règle*[5] $L_1 \wedge \ldots \wedge L_n \rightarrow A$ (représentée dans Prolog comme `A:-L1, ..., Ln.`) où A est un atome (excepté l'égalité) et L_i $(1 \leq i \leq n)$ est un littéral, soit un *fait* A (représentée dans Prolog comme `A.`) où A est un atome[6].

Définition 8 *(Une requête).* Une *requête* est une conjonction de littéraux $L_1 \wedge \ldots \wedge L_n$ (représentée dans Prolog comme `L1, ..., Ln.`).

Revenons à notre objectif principal, la démonstration automatique de théorèmes des théories, qui se traduit intuitivement par une déduction (démonstration), à l'aide d'une machine (automatique), des propriétés (théorèmes) à partir des affirmations (axiomes de la théorie). Dans la terminologie de Prolog on peut l'énoncer comme suit: répondre (démonstration) aux requêtes (théorèmes) à partir de faits et de règles (axiomes de la théorie) en utilisant un interpréteur (automatique).

Pour résoudre le problème de la démonstration en Prolog on considère dans une première étape la transformation d'un ensemble de formules en un programme Prolog. Dans une deuxième étape on considère l'analyse du fonctionnement des interpréteurs, lesquels utilisent comme moteur d'inférence l'unification (cf. Section C.4) et la résolution pour prouver une propriété[7] à partir d'un programme. Et finalement on considère la garantie de la validité des implémentations en Prolog de certains types des théories (les théories complètes).

4.2.2 Programmes représentant des théories

La première étape consiste à transformer un ensemble de formules (les axiomes de la théorie et les théorèmes à démontrer) en un ensemble de clauses (programme) tout en gardant les propriétés de satisfaisabilité des formules. Pour cela chaque formule est représentée en forme standard de Skolem.

La forme standard de Skolem[8] d'une formule a été introduite par Davis et Putnam [DP60]. Cette forme est une conjonction de clauses[9] qui préserve la propriété d'inconsistance de la formule.

Pour trouver la forme standard d'une formule les transformations ci-dessous sont effectuées.

[4] Une clause de la forme $\neg L_1 \vee \ldots \vee \neg L_n \vee A$ peut être réécrite par $L_1 \wedge \ldots \wedge L_n \rightarrow A$. Donc nous nous permettons parfois d'utiliser les termes 'antécédent' et 'conséquent' d'une clause pour désigner $L_1 \wedge \ldots \wedge L_n$ et A respectivement.

[5] Au sens strict cela n'est plus une clause parce que la négation des littéraux en Prolog n'utilise pas l'opérateur de négations classique.

[6] Si l'atome contient des variables, elles sont considérées comme quantifiées universellement, et au lieu d'un fait on a une propriété.

[7] Dans la terminologie Prolog le terme adéquat est 'répondre aux requêtes' et non 'prouver (démontrer) une propriété (théorème)' car ce dernier appartient à la terminologie de la logique. Nous faisons volontairement cet abus de terminologie afin de ne pas oublier notre objectif 'la démonstration de théorèmes'.

[8] Dans la littérature on l'appelle aussi forme standard, forme normale de Skolem ou forme clausale.

[9] Représentée aussi comme un ensemble des clauses.

- La formule de la logique du premier ordre est transformée[10] en une formule en forme normale prénexe, c'est-à-dire une formule de la forme $(Q_1x_1)\cdots(Q_nx_n)M$, où M est une formule sans quantificateur appelée matrice et $(Q_1x_1)\cdots(Q_nx_n)$ une séquence de quantificateurs appelé préfix.

- La matrice est transformée jusqu'à l'obtention d'une formule en forme normale conjonctive (conjonction des clauses).

- Le préfixe est éliminé. Les quantificateurs sont enlevés, et les variables liées aux quantificateurs existentiels sont remplacées par des fonctions de Skolem [CL73]. La formule résultante est en forme Standard de Skolem. Cette forme est une conjonction des clauses dont chaque variable est quantifiée universellement de façon implicite.

Propriété 1 Pour montrer qu'une formule A est conséquence logique d'un ensemble de formules S où S est satisfaisable (consistante), il suffit de montrer que $S' = S \cup \neg A$ est insatisfaisable (inconsistante), ou encore il suffit de montrer que $T(S')$ est insatisfaisable, où $T(S')$ est le résultat de la transformation de chaque formule dans S' en un ensemble de clauses.

En général dans les systèmes usuels de programmation logique pour améliorer l'efficacité des systèmes de démonstration on n'admet pas les clauses disjonctives (clauses avec plus d'un littéral positif). En particulier dans Prolog on se limite aux clauses de Horn. Prolog est donc plus restrictif au niveau syntaxique que la logique du premier ordre.

Enrichir le pouvoir d'expression du langage a pour conséquence l'affaiblissement (parfois inacceptable) de la performance du système de démonstration. Dans Prolog pour enrichir le langage des clauses de Horn on autorise l'utilisation d'un opérateur de négation par échec (`not`) dans les antécédents des clauses composant un programme Prolog. Cet opérateur donne une "approximation" de la négation logique ($\neg$). Par contre, les négations ne sont pas admises dans les conséquents des clauses composant un programme. Néanmoins les faits négatifs sont déduits implicitement par l'hypothèse du monde fermé, i.e. ils sont déduits par l'absence de preuve de leurs correspondants positifs.

Bref on peut exprimer le problème de la démonstration de théories dans le langage des clauses de Horn enrichi avec l'opérateur `not`. Dans une deuxième étape on considère la façon d'interpréter ce langage. Dans la section suivante on présente l'interprétation de l'opérateur `not`. Ensuite le mécanisme d'inférence utilisé par Prolog, la résolution.

4.2.3 Négation par échec

L'opérateur de négation dans Prolog est exprimé par le prédicat prédéfini `not`, lequel est interprété à l'aide du principe de négation par échec. Soit G un prédicat, le prédicat `not` G réussit si la preuve de G échoue.

[10] Nous utilisons ici 'transformer' dans le sens de réécrire.

Il faut remarquer que le prédicat `not` repose sur un algorithme de Prolog et non sur une définition logique. A la différence de l'hypothèse du monde fermé, qui est basée sur une notion de démonstration logique pure.

L'emploi de la négation par échec est donc extralogique, il existe une claire différence entre cette négation et la négation classique (négation de la logique de premier ordre, cf. Annexe B.2).

Supposons l'information suivante : "toute personne qui va en Amérique et qui ne prend pas un avion, prend un bateau", représentée par la formule suivante :

$$\forall x(aller\_Amerique(x) \wedge \neg A(x) \rightarrow B(x)). \tag{4.1}$$

Supposons aussi que l'on veut exprimer que pour aller en Amérique le fait 'prendre un avion' est privilégié par rapport à 'prendre un bateau'. Or si l'unique information que l'on possède, est 'elle va en Amérique', on ne peut pas conclure 'elle prend un avion'. En effet, la formule (4.1) est équivalente a :

$$\forall x(aller\_Amerique(x) \rightarrow (A(x) \vee B(x))).$$

Cette formulation ne correspond pas à ce que l'on veut exprimer, étant donné le caractère symétrique de la disjonction. Ce qu'il faudrait, c'est un mécanisme qui permettrait de distinguer $A \vee B$ de $\neg A \rightarrow B$, où la signification de cette dernière formule serait : B sauf si A.

En réponse à ce souci, dans [Cla78] Clark introduit la négation par échec (négation as failure) pour les bases de données logiques. Il propose un algorithme d'évaluation des requêtes basé sur la résolution linéaire ordonnée pour les clauses de Horn (SLD), augmentée de la règle d'inférence de négation par échec : (SLDNF). Ainsi B peut être inféré si toute preuve possible de A échoue.

Prolog utilise cet algorithme pour interpréter l'opérateur de négation ($\neg$). L'inconvénient de SLDNF est qu'il a des problèmes de bouclage.

4.2.4 Résolution

L'interpréteur Prolog est un système de démonstration qui utilise comme unique règle d'inférence la règle de résolution [Rob65]. Il opère par réfutation.

Définition 9 *(Règle d'inférence principe de résolution).* Soient C_1 et C_2 deux clauses. Si C_1 contient un littéral L_1, et C_2 contient un littéral L_2 et L_1 et $\neg L_2$ sont unifiables avec σ comme mgu, alors on peut inférer par *principe de résolution*[11] la clause[12] $C_1\sigma - L_1\sigma \vee C_2\sigma - L2\sigma$. La clause inférée est appelée *résolvant* de C_1 et C_2.

Théorème 2 Soient C_1 et C_2 deux clauses. Si C est le résolvant de C_1 et C_2, alors C est conséquence logique de C_1 et C_2 (cf. Théorème 5.1 dans [CL73]).

Définition 10 *(Déduction par résolution).* Soit S un ensemble de clauses, une *déduction par résolution de C à partir de S* est une séquence finie de clauses

[11] Parfois appelé simplement *résolution.*

[12] $C - L$ dénote la clause C auquel on a enlevé le littéral L.

$C_1, \ldots C_n$ telle que C_i est une clause dans S ou bien un résolvant de deux clauses précédentes, et $C_n = C$. Une déduction par résolution de la clause vide ($\square$) est appelée *réfutation par résolution.*

On dit que C peut être dérivée (déduite, inférée ou générée) de S s'il existe une déduction par résolution de C à partir de S.

Comment Prolog utilise le principe de résolution pour démontrer des théorèmes? L'idée est d'utiliser la Propriété 1 et une réfutation par résolution de la façon suivante : le principe de résolution est appliqué à un ensemble de clauses S afin de tester son insatisfaisabilité. Si la clause vide est générée à partir de S, alors S est insatisfaisable.

Une procédure de démonstration est la suivante :

(1) Utiliser la règle de résolution pour générer des clauses nouvelles à partir de S.

(2) Ajouter ces clauses nouvelles à S pour obtenir S'.

(3) Si $\square$ est dans S' arrêter le processus, sinon recommencer le processus à partir de (1) avec le nouvel ensemble S'.

Cette méthode appelée saturation par niveau parcourt l'ensemble de tous les choix de résolution en *largeur d'abord.* Son avantage est qu'il trouve toujours une solution, s'il y en a. Son inconvénient est que de nombreuses clauses irrelevantes sont générées.

Une autre méthode réalise une recherche en *profondeur d'abord*, de gauche à droite. L'avantage est que la gestion de mémoire est plus efficace, mais la recherche dans une branche peut éventuellement ne pas se terminer. La stratégie est la suivante. Une demande de résolution est faite (correspondant à une requête). La résolvante, une suite de littéraux restant à résoudre, est construite avec les littéraux de la requête.

(1) L'interpréteur Prolog tente de résoudre le premier littéral de la résolvante. i.e. examine séquentiellement les clauses du programme jusqu'à trouver une clause dont le conséquent de la clause (la tête) est unifiable avec la négation du littéral à résoudre.

(2) S'il n'existe pas une telle clause, la résolution échoue et un retour en arrière est effectué sur le littéral précédemment résolu.

(3) Si cette clause est trouvée les littéraux de l'antécédent de la clause (la queue) sont ajoutés au début de la résolvante

Chaque méthode a un inconvénient donc il faut faire un compromis entre le nombre exponentiel de clauses généré à chaque niveau et l'incertitude de la terminaison. En général les interpréteurs Prolog utilisent la méthode en profondeur d'abord dû à la limitation de mémoire et au temps de calcul.

Quelques stratégies pour améliorer l'efficacité consistent à raffiner la méthode pour éviter la redondance et pour éliminer des clauses inutiles à la déduction telles que les tautologies et les clauses subsumées.

On trouve aussi des stratégies pour améliorer l'efficacité de Prolog telles qu'un contrôle dans la stratégie de résolution, par exemple, ordonner les littéraux des clauses pour faciliter l'évaluation.

4.2.5 Validité de Prolog

Prolog est basé sur un système logique qui est valide et complet mais l'algorithme d'implémentation n'est pas complet.

Les seules théories qui peuvent être implémentées en Prolog doivent contenir les définitions des prédicats, i.e la théorie doit contenir pour chaque prédicat n-aire P (excepté l'égalité) un axiome de la forme:

$$\forall x_1, \ldots, \forall x_n P(x_1, \ldots, x_n) \leftrightarrow \psi \tag{4.2}$$

où ψ est une formule dont les variables libres sont $x_1, \ldots, x_n$. Ce type de théorie est appelé *complète*. Des formules telles que $\exists x\, \mathit{président}(x)$, $\mathit{président}(Clinton) \vee \mathit{président}(Bush)$ ne peuvent pas être exprimées dans ce type de théorie. D'ailleurs des informations telles que: $\neg \mathit{président}(Toto)$ où la signification intuitive est 'le président n'est pas Toto' ne peuvent pas être exprimées. L'hypothèse du monde fermé est utilisée pour chaque définition incomplète du prédicat. Cette hypothèse rendre toujours complète la théorie.

Un interpréteur Prolog standard (dont l'interprétation correspond à la théorie) est celui qui évalue un littéral négatif (`not A`) avec la négation par échec. Cette évaluation s'effectue seulement quand toutes les variables apparaissant dans `A` sont instanciées, i.e quand `A` est clos (cf. Annexe C.2). Ce type d'interpréteur ne cherche pas d'échec sur des atomes non clos.

Clark introduit le résultat qui justifie l'implémentation en Prolog des théories complètes. De façon générale celui-ci peut être énoncé comme suit: si un programme Prolog réussit sur une formule, alors cette formule est une conséquence logique de la théorie et s'il échoue (dans le sens de *donner une réponse négative* et non dans le sens de *ne pas répondre* à cause d'une boucle infinie par exemple) sur une formule alors la négation de cette formule est une conséquence logique de la théorie [Cla78, Llo87]. Donc la garantie de la validité d'une implémentation Prolog d'une théorie complète est justifiée. La complétude d'une implémentation Prolog n'est pas garantie.

Dans [Cla78, Llo87] on trouve aussi la justification de la complétion, c'est-à-dire l'écriture d'une clause Prolog représentant un prédicat contenant seulement la "moitié de la définition" (if-half en anglais). Par exemple la définition de P (formule 4.2) est représenté dans Prolog par $\mathtt{P(x_1, \ldots, x_n)} : -\psi$.

Les transformations de Lloyd-Topor permettent de réécrire une formule quelque soit en format Prolog tout en gardant le résultat de la validité de Clark.

Le résultat de Clark impose certaines conditions aux théories, parmi lesquelles celle de contenir les axiomes d'unicité de noms. C'est-à-dire la théorie considère deux symboles de fonction (y compris les constantes) comme deux éléments différents du domaine (cf. Section 5.1.1).

4.3 Déduction automatique avec égalité

La déduction automatique dans le contexte des théories avec égalité, tel est le cas du calcul des situations, provoque de nombreux problèmes d'efficacité. Par exemple, si l'on applique la règle de la paramodulation (cf. Définition 14) sans restrictions, on génère un grand nombre de clauses inutiles à la déduction.

4.3.1 Rappels théoriques

Dans cette section on présente un ensemble minimum de concepts à propos des théories avec égalité qui ont servi de base à la plupart des E-démontrateurs.

Dans [CL73] on trouve les définitions et le théorème suivants :

Définition 11 *(E-interprétation).* Une *E-interprétation* I d'un ensemble de clauses Σ est une interprétation de Σ qui satisfait les quatre conditions ci-dessous. Soient α, β et γ des termes dans l'univers d'Herbrand de Σ (cf. Annexe C.2), et soit L un littéral dans I, alors

1. $(\alpha = \alpha) \in I$;
2. si $(\alpha = \beta) \in I$, alors $(\beta = \alpha) \in I$;
3. si $(\alpha = \beta) \in I$ et $(\beta = \gamma) \in I$, alors $(\alpha = \gamma) \in I$;
4. si $(\alpha = \beta) \in I$ et L' est le littéral résultant du remplacement d'une occurrence de α par β dans L, alors $L' \in I$.

Définition 12 *(E-satisfaisabilité).* Un ensemble de clauses Σ est appelé *E-satisfaisable* si et seulement si il existe une E-interprétation qui satisfait toutes les clauses de Σ. Dans le cas contraire, Σ est appelé *E-insatisfaisable.*

Définition 13 *(Ensemble des schémas d'axiomes de l'égalité)*[13]. Soit Σ un ensemble des clauses, alors *l'ensemble des schémas d'axiomes de l'égalité* pour Σ $K(\Sigma)$ est l'ensemble constitué des clauses suivantes :

(a) $x = x$

(b) $\neg(x = y) \vee y = x$

(c) $\neg(x = y) \vee \neg(y = z) \vee x = z$

(d) $\neg(x_i = x_0) \vee \neg P(x_1, \ldots, x_i, \ldots, x_n) \vee P(x_1, \ldots, x_0, \ldots, x_n)$, P est un prédicat de Σ

(e) $\neg(x_i = x_0) \vee f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_0, \ldots, x_n)$, f est une fonction de Σ.

Théorème 3 Soit Σ un ensemble de clauses et $K(\Sigma)$ son ensemble de schémas d'axiomes de l'égalité, alors

[13] Pour simplifier la lecture, parfois on appelle cet ensemble simplement les axiomes de l'égalité, ou l'ensemble d'axiomes de l'égalité.

Σ est E-insatisfaisable ssi $\Sigma \cup K(\Sigma)$ est insatisfaisable.

Ce résultat est intéressant au niveau automatisation car il permet de reprendre les résultats déjà obtenus pour les théories sans l'égalité et de les appliquer aux théories avec égalité. Intuitivement cela veut dire qu'en rajoutant les axiomes de l'égalité à la théorie, celle-ci peut être traitée comme une théorie sans l'égalité.

Un autre résultat sur lequel sont basés plusieurs travaux et qui n'interprète pas l'égalité à partir des axiomes, est celui qui introduit une règle d'inférence appelée paramodulation[RW69]. Par exemple, il existe une méthode de déduction qui combine résolution et paramodulation.

4.3.2 Résolution + Paramodulation

Désormais si une expression E (une clause, un littéral ou un terme) contient le terme t et s'il n'y a pas d'ambiguïté on dénotera E comme $E[t]$. Par exemple $P[f(g(a))]$ dénote le littéral $P(x, f(g(a)))$ qui contient le terme $f(g(a))$, et $P[a]$ dénote le même littéral $P(x, f(g(a)))$ mais en faisant référence au terme a. En cas d'ambiguïté, on ajoutera un index pour indiquer l'occurrence en question. Ainsi par exemple $Q[a]_3$, dénote le littéral $Q(f(g(a, l(a))), x, d(a), s, a)$ en faisant référence au terme a qui apparaît dans $d(a)$ et qui est la troisième occurrence (cf. Annexe A).

Définition 14 *(Règle de la paramodulation).* Soient C_1 et C_2 deux clauses, telles que C_1 contient un littéral d'égalité et dans un argument de l'égalité le terme r, c'est-à-dire C_1 a la forme $r = s \vee C_1'$, et C_2 contient un littéral L qui contient le terme t, c'est-à-dire C_2 a la forme $L[t] \vee C_2'$. Où C_1' et C_2' sont deux clauses quelconques. Si t et r ont comme unifieur le plus général σ (cf. Annexe C.4), alors on peut inférer $L\sigma[s\sigma] \vee C_1'\sigma \vee C_2'\sigma$, où $L\sigma[s\sigma]$ dénote le résultat du remplacement de l'occurrence $t\sigma$ par $s\sigma$ dans $L\sigma$. La règle peut être définie graphiquement comme suit[14] :

$$\frac{r = s \vee C_1' \qquad L[t] \vee C_2'}{L\sigma[s\sigma] \vee C_1'\sigma \vee C_2'\sigma} \; t\sigma \equiv r\sigma.$$

La paramodulation linéaire [CL73] combine la résolution et la paramodulation. On reproche à cette méthode la manque d'efficacité car la génération de clauses inutiles à la déduction est considérable. C'est une méthode orientée par le but mais seulement pour résolution et elle n'impose pas de restrictions à l'application de la règle de paramodulation.

4.3.3 Exemples

Dans cette section on s'appuie sur deux exemples pour montrer comment peut être utilisée la paramodulation dans un domaine spécifique. On se place dans le domaine de la robotique, on suppose un contexte composé d'un robot

[14] On utilise la notation $t\sigma \equiv r\sigma$ pour indiquer que les termes $t\sigma$ et $r\sigma$ sont syntaxiquement identiques.

qui avance et recule. On suppose aussi l'existence éventuelle d'obstacles sur le trajet du robot. On utilise les prédicats $R(x,s)$ et $O(x,s)$ pour dénoter 'la position du robot est x dans la situation s' et 'il existe un obstacle dans la position x dans la situation s'.

Supposons que l'on a les axiomes suivants :

(a) $R(4,S_0)$: Le robot est dans la position 4 en S_0.

(b) $\forall x \forall y(R(x,s) \wedge R(y,s) \rightarrow x = y)$: La position du robot est unique.

(c) $\exists x \exists y(R(x,S_0) \wedge O(y,S_0) \wedge y = x+1)$ Il y un obstacle juste une unité après le robot.

La forme clausale de ces axiomes est :

(1) $R(4,S_0)$

(2) $\neg R(x,s) \vee \neg R(y,s) \vee x = y$

(3) $R(\alpha,S_0)$

(4) $O(\beta,S_0)$

(5) $\beta = \alpha + 1$

A partir de l'ensemble des clauses on veut dériver la position de l'obstacle, i.e. on veut inférer $O(5,S_0)$.

Une dérivation en utilisant résolution et paramodulation est :

(6) $\neg R(y,S_0) \vee \alpha = y$ résolution 2 et 3

(7) $\alpha = 4$ résolution 1 et 6

(8) $\beta = 4+1$ paramodulation 5 et 7

(9) $\beta = 5$ opération arithmétique 8

(10) $O(5,S_0)$ paramodulation 9 et 4

On a utilisé paramodulation dans le pas (8) et (10) . Dans le pas (8) paramodulation est utilisée pour calculer la valeur de la constante de Skolem, et dans le pas (10), c'est utilisé pour remplacer une constante de Skolem par sa valeur dans le littéral.

Dans le pas (9) on utilise implicitement les propriétés arithmétiques. On suppose qu'une procédure externe peut être appelée pour faire cette opération. La caractérisation syntaxique de ces expressions est un problème qui se pose si l'on souhaite l'automatisation de la dérivation de ce type de formules.

On a implicitement $\neg(4 = 5)$. En général deux symboles différents représentent deux éléments différents du domaine. Ainsi on peut dériver $\neg R(4,s) \vee \neg R(5,s)$ à partir de $\neg R(4,s) \vee \neg R(5,s) \vee 4 = 5$. Cette hypothèse n'est pas étendue aux constantes de Skolem, ainsi on n'a pas implicitement $\neg(\alpha = 5)$.

Maintenant, si on veut obtenir une dérivation linéaire qui permette de dériver la clause vide à partir de la clause initiale $\neg O(5, S_0)$, alors un problème se pose, on ne peut pas résoudre le littéral $\neg O(5, S_0)$ avec le littéral $O(\beta, S_0)$ parce que β ne peut pas être unifié avec 5. Cependant, on peut essayer de prouver que la valeur de β est 5 ($\beta = 5$). Une déduction linéaire peut être :

(6) $\neg O(5, S_0)$ clause initiale

(7) $\neg O(\beta, S_0) \vee \neg(\beta = 5)$ de 6

(8) $\neg(\beta = 5)$ résolution 4 et 7

(9) $\neg(\alpha + 1 = 5)$ paramodulation 5 et 8

(10) $\neg(\alpha = 4)$ opération arithmétique 9

(11) $\neg R(\alpha, S_0) \vee \neg R(4, S_0)$ résolution 2 et 10

(12) $\neg R(4, S_0)$ résolution 3 et 11

(13) $\square$ résolution 1 et 12

Ici la paramodulation est utilisée uniquement dans le pas 9.
Un autre exemple suppose les hypothèses suivantes :

(d) $\exists x \exists y(R(x, S_0) \wedge O(y, S_0) \wedge x = y - 1)$: Il y un obstacle juste une unité après le robot.

(e) $\forall x(O(x, S_0) \rightarrow x = 5)$: Il y a un unique obstacle, il est dans la position 5.

La forme clausale des hypothèses est :

(1) $R(\alpha, S_0)$

(2) $O(\beta, S_0)$

(3) $\alpha = \beta - 1$

(4) $\neg O(x, S_0) \vee x = 5$

A partir de ces hypothèses on veut dériver la position du robot. C'est-à-dire on veut inférer $R(4, S_0)$.
Une dérivation est :

(5) $\beta = 5$ résolution 2 et 4

(6) $\alpha = 5 - 1$ paramodulation 3 et 5

(7) $\alpha = 4$ opération arithmétique 6

(8) $R(4, S_0)$ paramodulation 1 et 7

La paramodulation est utilisée dans les pas 6 et 8.

Une dérivation linéaire de la clause vide à partir de la clause initiale $\neg R(4, S_0)$ est :

(5)	$\neg R(4, S_0)$	clause initiale
(6)	$\neg R(\alpha, S_0) \vee \neg(\alpha = 4)$	de 5
(7)	$\neg(\alpha = 4)$	résolution 1 et 6
(8)	$\neg(\beta - 1 = 4)$	paramodulation 3 et 7
(9)	$\neg(\beta = 5)$	opération arithmétique 8
(10)	$\neg O(\beta, S_0)$	résolution 4 et 9
(11)	$\square$	résolution 2 et 10

On peut noter que la règle de justification du pas 6 (également celle du pas 7 de la dérivation linéaire précédente) n'a pas été introduite. Dans Brand [Bra75] on trouve une règle similaire qu'il appelle E-modification. Dans la procédure que nous proposons (cf. Section 4.6), une règle, qui a une fonction similaire, est introduite, on l'appelle E-lit.

A partir de ces deux exemples on peut soupçonner des solutions possibles aux problèmes posés par l'égalité et par les expressions arithmétiques dans ce type de théorie.

Ces solutions sont basées sur les idées suivantes :

1. L'implémentation d'axiomes d'unicité de noms des constantes (à l'exception des constantes Skolem) a et b. En utilisant résolution à partir de $C \vee a = b$ on peut inférer C.

2. Restreindre l'utilisation de la paramodulation. A partir de $C_1 \vee R(\alpha)$ et $\neg R(a) \vee C_2$, inférer $C_1 \vee C_2 \vee \neg(\alpha = a)$.

3. L'appel aux procédures externes pour le traitement des expressions arithmétiques.

Ces conjectures nous donnent une direction de recherche. Par exemple, concernant la procédure d'évaluation arithmétique, il est souhaitable de générer des clauses formées uniquement avec des opérations arithmétiques qui seront évaluées par cette procédure externe.

Les caractéristiques spécifiques du contexte montrent l'utilité de l'égalité. Elle permet de spécifier l'unicité (formule (b)), la complétion d'un prédicat (formule (e)) et des expressions arithmétiques. Dans le calcul des situations elle est utilisée aussi pour exprimer la solution du frame problem.

4.4 SOL-résolution

Dans [Ino91, Ino92a, Ino92b] Inoue a définit la SOL-résolution (Skip Ordered Linear résolution) qui est un type de résolution linéaire ordonnée. C'est une stratégie pour la génération de conséquences qui satisfont une certaine propriété.

La terminologie utilisée dans cette approche est la suivante :

Définition 15 *(Champ de production).* Un *champ de production* $\mathcal{P}$ (cf. Definition 2.1 dans [Ino92a]) est une paire ordonnée $< L, Cond >$, où L est un ensemble de littéraux fermé pour l'instanciation et $Cond$ est une condition. Une clause C appartient au champ de production $\mathcal{P}$ si chaque littéral apparaissant dans C est dans L et C satisfasse $Cond$.

Définition 16 *(Clause structurée).* Une *clause structurée* est une paire $< P, \vec{Q} >$, telle que P est une clause et $\vec{Q}$ est une clause ordonnée (une séquence de littéraux différents).

Définition 17 *(SOL-déduction).* Soient Σ un ensemble de clauses, C une clause et $\mathcal{P}$ un champ de production. Une *SOL-déduction* de la clause S à partir de[15] $\Sigma + C$ et $\mathcal{P}$ est une séquence de clauses structurées $D_0, D_1, \ldots, D_n$ qui satisfont certaines conditions. Afin d'éviter les redondances nous n'explicitons pas ici ces conditions car elles sont similaires à celles de la définition d'une SOLE-déduction (cf. Définition 21). Seulement dans une SOL-déduction la règle E-inférence ne s'applique pas.

Définition 18 *(SOL-résolution).* Le procédure pour trouver une SOL-déduction est appelée *SOL-résolution.*

4.5 SOL-résolution pour les théories avec égalité

Dans cette section nous montrons le résultat sur lequel s'appuie la démonstration de la complétude de la SOLE-résolution. L'idée est d'étendre une théorie E-satisfaisable (respectivement E-insatisfaisable) à une théorie satisfaisable (respectivement insatisfaisable) en rajoutant deux schémas d'axiomes à la théorie initiale. L'ajout des schémas préserve la complétude de la SOL-résolution pour la génération de conséquences dans des théories avec égalité.

On présente la définition d'un ensemble des schémas d'axiomes généraux d'égalité, suivi d'une preuve d'équivalence entre cet ensemble de schémas et celui défini précédemment (cf. Définition 13), finalement on prouve la complétude de la SOL-résolution pour des théories avec égalité qui ont été complétées par les axiomes généraux d'égalité.

[15] Si E est un ensemble de clauses, alors $E + C$ dénote l'ensemble de clauses $E \cup C$, qui considère C comme clause initiale.

Définition 19 *(L'ensemble des schémas d'axiomes généraux d'égalité).* Soit Σ un ensemble de clauses, alors *l'ensemble des schémas d'axiomes généraux d'égalité* pour[16] Σ, $K^*(\Sigma)$ est l'ensemble formé des clauses suivantes :

(a*) $x = x$

(b*) $\neg(x_i = x_0) \vee \neg P[x_i] \vee P[x_0]$, P est un prédicat de Σ (cf. Annexe A).

Intuitivement, la généralisation se fait sur le schéma (d) de la définition 13. x_i dans (d) dénote un argument de P, et dans le schéma (b*) x_i dénote soit un argument de P soit un terme contenu dans un argument de P.

Le théorème suivant montre l'équivalence existant entre les deux ensembles de schémas $K(\Sigma)$ et $K^*(\Sigma)$.

Théorème 4 Soient Σ un ensemble des clauses, $K(\Sigma)$ l'ensemble de schémas d'axiomes d'égalité de Σ, $K^*(\Sigma)$ l'ensemble des schémas d'axiomes généraux d'égalité de Σ. On a :

$$\vdash K^*(\Sigma) \leftrightarrow K(\Sigma).$$

Preuve :

($\Rightarrow$) On va prouver $K^*(\Sigma) \vdash K(\Sigma)$.
On a :
(a) $K^*(\Sigma) \vdash x = x$ (de a*);
(b) $K^*(\Sigma) \vdash \neg(x = y) \vee y = x$.
En effet, on a :
1. $x = x$ (de a*)
2. $\neg(x = y) \vee \neg(x = x_2) \vee (y = x_2)$ (b* pour le prédicat $=$)
3. $\neg(x = y) \vee y = x$ (MP 1,2 et x/x_2);
(c) $K^*(\Sigma) \vdash \neg(x = y) \vee \neg(y = z) \vee x = z$.
En effet, on a :
1. $\neg(y = z) \vee \neg(x_1 = y) \vee (x_1 = z)$ (b* pour le symbole de prédicat $=$)
2. $\neg(x = y) \vee \neg(y = z) \vee x = z$ (de 1, commutativité de $\vee$ et x/x_1);
(d) $K^*(\Sigma) \vdash \neg(x_i = x_0) \vee \neg P(x_1, \ldots, x_i, \ldots, x_n) \vee P(x_1, \ldots, x_0, \ldots, x_n)$ (de b*);
(e) $K^*(\Sigma) \vdash \neg(x_i = x_0) \vee f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_0, \ldots, x_n)$.
En effet, on a :
1. $f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_i, \ldots, x_n)$ (de a*)
2. $\neg(x_i = x_0) \vee \neg(f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_i, \ldots, x_n)) \vee$
$f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_0, \ldots, x_n)$ (b* pour le prédicat $=$)
3. $\neg(x_i = x_0) \vee f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, x_0, \ldots, x_n)$ (MP 1,2).

On a donc démontré que a, b, c, d et e sont des conséquences de $K^*(\Sigma)$, i.e. on a $K^*(\Sigma) \vdash K(\Sigma)$. On peut donc conclure que : $\vdash K^*(\Sigma) \rightarrow K(\Sigma)$.

($\Leftarrow$) On va prouver que $K(\Sigma) \vdash K^*(\Sigma)$.
On a :
(a*) $K(\Sigma) \vdash x = x$ (de a)

[16] S'il n'y a pas d'ambiguïté dans l'interprétation, on utilise la notation $K^*(C)$ au lieu de $K^*(\{C\})$ quand C est une clause ou la négation d'une clause.

(b*) $K(\Sigma) \vdash \neg(x_i = x_0) \vee \neg P[x_i] \vee P[x_0]$, est démontré par induction sur la profondeur des termes x_i et x_0. On adopte la notation[17] $P(t_m(x_i))$ (respectivement $P(t_m(x_0))$ pour dénoter le fait suivant $t_m(x_i)$ (respectivement $t_m(x_0)$) est l'argument de P dans lequel apparaît x_i (respectivement x_0), où la profondeur de t_m est m. L'hypothèses d'induction (h_m) est :
$K(\Sigma) \vdash \neg(x'_i = x'_0) \vee \neg P(t_m(x'_i)) \vee P(t_m(x'_0))$.
Les deux conditions d'une induction sont satisfaites :

1. (h_0) (de d).
2. Supposons (h_m). Considérons l'instance de (h_m) obtenue en remplaçant x'_i par $f(x_i)$, et x'_0 par $f(x_0)$, où f est un symbole de fonction. On a donc :
   (h'_m) $K(\Sigma) \vdash \neg(f(x_i) = f(x_0)) \vee \neg P(t_m(f(x_i))) \vee P(t_m(f(x_0)))$, de (h'_m) et e par MP on a :
   $K(\Sigma) \vdash \neg(x_i = x_0) \vee \neg P(t_m(f(x_i))) \vee P(t_m(f(x_0)))$, qui représente (h_{m+1}), où $t_m(f(x))$ dénote un terme de profondeur $m+1$.

On a donc montré que a* et b* sont des conséquences de $K(\Sigma)$, i.e. on a $K(\Sigma) \vdash K^*(\Sigma)$. Par conséquent on a $\vdash K(\Sigma) \rightarrow K^*(\Sigma)$. ■

4.5.1 SOL-résolution avec les axiomes généraux d'égalité

Les lemmes suivants montrent que l'E-insatisfaisabilité d'une théorie Σ est équivalente à l'insatisfaisabilité de l'extension d'une telle théorie avec les axiomes généraux de l'égalité $K^*(\Sigma)$.

Lemme 5 Soit Σ un ensemble de clauses et $K^*(\Sigma)$ son ensemble d'axiomes généraux, alors on a :

Σ est E-insatisfaisable ssi $\Sigma \cup K^*(\Sigma)$ est insatisfaisable.

Preuve : Conséquence du Théorème 3 et du Théorème 4. ■

Lemme 6 Soient Σ et Σ_1 deux ensembles de clauses. On a :

$\Sigma \cup K^*(\Sigma)$ est satisfaisable ssi $\Sigma \cup K^*(\Sigma) \cup K^*(\Sigma_1)$ est satisfaisable.

Preuve :
($\Leftarrow$) Trivial.
($\Rightarrow$) On démontre que l'interprétation d'Herbrand qui satisfait s $\Sigma \cup K^*(\Sigma)$ peut être étendue à une interprétation d'Herbrand qui satisfait $\Sigma \cup K^*(\Sigma) \cup K^*(\Sigma_1)$. ■

Les Lemmes 7 et 8 sont une transposition des Lemmes 5 et 6 respectivement, en termes de conséquence logique.

On utilise les notations suivantes :

- $\Sigma \vdash T$ dénote le fait que T peut être déduit de Σ. $Th(\Sigma)$ dénote l'ensemble de clauses T tel que $\Sigma \vdash T$.

[17] Afin d'éviter des notations complexes, dans P on a omis tous les arguments sauf celui qui nous intéresse et de manière semblable dans t_m. En général P et t_m peuvent avoir un nombre fini d'arguments.

- $\Sigma \vdash_E T$ dénote le fait que T peut être déduit de $\Sigma \cup K(\Sigma) \cup K(\neg T)$, i.e. $\Sigma, K(\Sigma), K(\neg T) \vdash T$. $Th\_E(\Sigma)$ dénote l'ensemble de clauses T tel que $\Sigma \vdash_E T$.

- $Th_P(\Sigma)$ dénote l'ensemble de clauses dans $Th(\Sigma)$ qui appartiennent au champ de production $\mathcal{P}$ (cf. Définition 15). $Th\_E_P(\Sigma)$ dénote l'ensemble de clauses dans $Th\_E(\Sigma)$ qui appartiennent au champ de production $\mathcal{P}$.

Lemme 7 Soit Σ un ensemble de clauses et T une clause. On a :

$$T \in Th\_E(\Sigma) \quad \text{ssi} \quad T \in Th(\Sigma \cup K^*(\Sigma) \cup K^*(\neg T)).$$

Preuve : Conséquence du Théorème 3 et du Lemme 5. ■

Lemme 8 Soient Σ et Σ_1 deux ensembles de clauses et T une clause. On a :

$$T \in Th(\Sigma \cup K^*(\Sigma) \cup K^*(\neg T)) \quad \text{ssi} \quad T \in Th(\Sigma \cup K^*(\Sigma) \cup K^*(\neg T) \cup K^*(\Sigma_1)).$$

Preuve : Conséquence du Lemme 6. ■

Théorème 9 *(Complétude de la SOL-résolution pour des théories avec égalité).* Soient Σ un ensemble de clauses, C une clause et $\mathcal{P}$ un champ de production. Si T est une clause telle que $T \notin Th\_E_P(\Sigma)$ et $T \in Th\_E_P(\Sigma \cup \{C\})$, alors il existe une SOL-déduction de la clause S de $\Sigma \cup K^*(\Sigma) \cup K^*(\neg T) \cup K^*(C) + C$ et $\mathcal{P}$ telle que S subsume T.

Preuve : La démonstration est basée sur le Lemme 7 et le Théorème 4.7 de [Ino92a]. ■

Ce théorème montre que la SOL-résolution peut être utilisée pour la déduction des théories avec égalité étendues avec leur axiomes généraux d'égalité correspondants. Avec cette extension le prédicat d'égalité peut être considéré comme un prédicat "normal".

Il faut remarquer que la SOL-résolution est utilisée pour la dérivation de conséquences (et non seulement pour générer la clause vide), et que l'ensemble d'axiomes $K^*(\neg T)$ ne peut pas être connu si T n'est pas connu. Cependant si on cherche des conséquences S construites à partir des prédicats qui apparaissent dans Σ, alors $K^*(\neg T)$ est inclus dans $K^*(\Sigma)$, et $K^*(\neg T)$ peut être ignoré.

4.6 SOLE-résolution (SOL-résolution + E-inférence)

On introduit la SOLE-résolution qui est une nouvelle méthode de déduction efficace, complète et valide pour la génération des conséquences (qui appartiennent à un champ de production) pour des théories avec égalité. Cette méthode est basée sur la SOL-résolution, à laquelle on a rajouté deux règles d'inférence, appelées E-equ et E-lit pour la manipulation de l'égalité. La SOLE-résolution a été implémentée en Prolog.

La méthode est basée sur les trois résultats suivants :

1. Il est bien connu que la clause vide $\square$ peut être déduite d'un ensemble insatisfaisable de clauses par résolution [Rob65]. C'est-à-dire la résolution est une méthode de déduction complète pour la réfutation des théories du premier ordre.

2. Dans [RW69] les auteurs montrent que $\square$ peut être générée à partir d'un ensemble E-insatisfaisable de clauses (voir Définition 12) par résolution et paramodulation. C'est-à-dire, la résolution + paramodulation est une méthode de déduction complète pour la réfutation des théories du premier ordre avec égalité. Un résultat analogue, celui de la paramodulation linéaire, a été présenté dans [CL73].

3. Dans [Ino92a], Inoue propose la SOL-résolution, une méthode de déduction efficace qui est complète pour la génération des conséquences. En particulier la SOL-résolution est complète pour la génération de la clause vide $\square$.

Une stratégie telle que la SOL-résolution est plus efficace que celle de la résolution à cause de son caractère linéaire ordonné (mis à part l'avantage de générer non seulement la clause vide mais aussi des conséquences). D'une façon semblable aux auteurs qui ont étendu la résolution pour considérer des théories avec égalité en rajoutant à celle-ci la paramodulation (cf. Section 4.3.2), nous avons rajouté deux règles d'inférence à la SOL-résolution afin d'obtenir une méthode plus efficace que celle de la résolution + paramodulation pour la génération des conséquences des théories avec égalité. Le problème se pose donc dans la caractérisation des règles afin d'obtenir une fonction similaire à celle de la paramodulation ainsi que d'étendre la SOL-résolution sans qu'elle perde son efficacité, sa complétude et sa validité.

Dans ces conditions, notre contribution (voir rectangles en pointillés dans la Figure 4.1) consiste en la définition des règles d'inférence E-lit et E-equ et aussi en la construction de la SOLE-résolution.

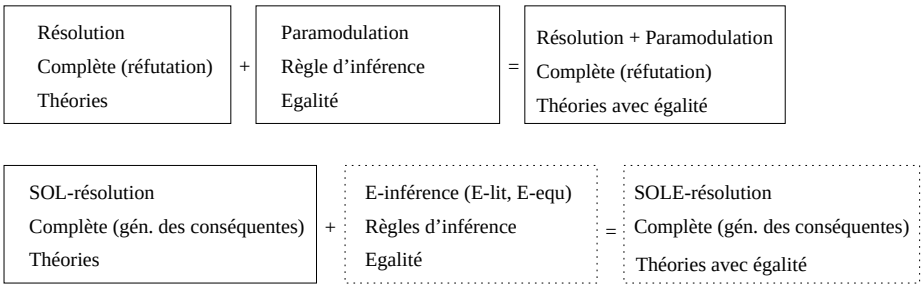

Figure 4.1: *La SOLE-résolution (SOL-résolution+E-inférence).*

4.6.1 Règles d'inférence E-lit et E-equ

Dans cette section on introduit deux règles d'inférence permettant le traitement de l'égalité dans le contexte des SOL-déductions. Ces règles jouent le même rôle que les axiomes K^* (à l'exception de $x = x$).

Dans le reste du chapitre on suppose que les clauses sont ordonnées.

Les règles d'inférence E-lit et E-equ sont utilisées en fonction du premier littéral qui apparaît dans $\vec{Q}$. On utilise la règle "adéquate" selon que le littéral est ou non constitué par le prédicat d'égalité.

Définition 20 *(Règles d'inférence pour l'égalité).* Les règles E-lit et E-equ sont définies comme suit :

E-lit. Soit $L[t] \vee C$ une clause telle que t est un terme qui apparaît dans L (cf. Annexe A), soit x une variable qui n'apparaît pas dans $L[t] \vee C$. On peut inférer la clause[18] $L[x] \vee \neg(t = x) \vee \boxed{L[t]} \vee C$ à partir de $L[t] \vee C$, où $L[x]$ dénote le résultat du remplacement d'une seule occurrence de t par x dans L.

Cette règle peut être représentée graphiquement comme suit :

$$\frac{L[t] \vee C}{L[x] \vee \neg(t = x) \vee \boxed{\text{L[t]}} \vee C} \qquad \text{(E-lit)}$$

E-equ. Soit $r = s \vee C$ une clause telle que le premier littéral est constitué par le prédicat d'égalité. On peut inférer la clause[19] $\neg L[r] \vee L[s] \vee \boxed{r = s} \vee C$ à partir de $r = s \vee C$.

Cette règle peut être représentée graphiquement comme suit :

$$\frac{r = s \vee C}{\neg L[r] \vee L[s] \vee \boxed{\text{r=s}} \vee C} \qquad \text{(E-equ)}$$

Maintenant on étend la définition de la SOL-déduction [Ino92a] afin de prendre en compte les propriétés de l'égalité. La différence entre la définition originale de Inoue et celle-ci est que la première ne comporte pas la règle de l'E-inférence.

Définition 21 *(SOLE-déduction).* Soient Σ un ensemble de clauses tel que $x = x$ est dans Σ, C une clause et $\mathcal{P}$ un champ de production. Une SOLE-déduction de la clause S à partir de $\Sigma + C$ et $\mathcal{P}$ est une séquence de clauses structurées $D_0, D_1, \ldots, D_n$ telle que :

1. $D_0 =< \Box, \vec{C} >$.
2. $D_n =< S, \Box >$.
3. Pour chaque $D_i =< P_i, \vec{Q}_i >$, $P_i \cup Q_i$ n'est pas une tautologie.
4. Pour chaque $D_i =< P_i, \vec{Q}_i >$, Q_i n'est pas subsumée par une clause Q_j avec la substitution vide, où $D_j =< P_j, \vec{Q}_j >$ est une clause structurée antérieure (i.e. $j < i$).

[18] Les littéraux encadrés sont utilisés pour garder la trace de leur élimination.

[19] Si L est de la forme $\neg P(t_1, \ldots, t_n)$, où P est un prédicat, alors $\neg L$ dénote $P(t_1, \ldots, t_n)$.

5. Pour chaque $D_i =< P_i, \vec{Q}_i >$, P_i appartient au champ de production $\mathcal{P}$.

6. $D_{i+1} =< P_{i+1}, \vec{Q}_{i+1} >$ est généré à partir de $D_i =< P_i, \vec{Q}_i >$ comme suit :

   (a) Soit l le littéral le plus à gauche dans $\vec{Q}_i$. P_{i+1} et $\vec{R}_{i+1}$ sont obtenues en appliquant l'une des règles suivantes :

      i. **Skip:** Si $P_i \cup \{l\}$ appartient à $\mathcal{P}$, alors $P_{i+1} = P_i \cup \{l\}$ et $\vec{R}_{i+1}$ est la clause ordonnée qui est obtenue en effaçant l de $\vec{Q}_i$.

      ii. **Resolve:** S'il existe une clause B_i dans $\Sigma \cup \{C\}$ telle que $\neg k \in B_i$ et l et k sont unifiables avec σ comme mgu (cf. Annexe C.4), alors $P_{i+1} = P_i\sigma$ et $\vec{R}_{i+1}$ est une clause ordonnée qui est obtenue par concaténation de $\vec{B}_i\sigma$ et $\vec{Q}_i\sigma$, suivi d'un encadrement de $l\sigma$, et un effacement de $\neg k\sigma$ (*extension).*

      iii. **E-inférence:**

         A. Si t est un terme dans l et si la variable x n'apparaît pas dans D_i, alors $P_{i+1} = P_i$ et $\vec{R}_{i+1}$ est la clause qui est obtenue par **E-lit** par concaténation de $l[x] \vee \neg(t = x)$ et $\vec{Q}_i$, et en encadrant $l[t]$.

         B. Si l est de la forme $r = s$, où r et s sont des termes, et si k est un littéral constitué d'un symbole de prédicat, alors $P_{i+1} = P_i$ et $\vec{R}_{i+1}$ est la clause ordonnée qui est obtenue par **E-equ** par concaténation de $\neg k[r] \vee k[s]$ et $\vec{Q}_i$, en encadrant $r = s$.

      iv. **Reduce:** Si l'un de deux cas :

         A. P_i ou $\vec{Q}_i$ contient un littéral non encadré k qui est différent de l (*factoring*) ou qui est une autre occurrence de l (*merge*), ou

         B. $\vec{Q}_i$ contient un littéral encadré $\boxed{\neg k}$ (*ancestry*),

         se produit, et l et k sont unifiables avec σ comme mgu, alors $P_{i+1} = P_i\sigma$ et $\vec{R}_{i+1}$ est obtenu de $\vec{Q}_i\sigma$ en effaçant $l\sigma$.

   (b) $\vec{Q}_{i+1}$ est obtenu de $\vec{R}_{i+1}$ en effaçant les premiers littéraux encadrés qui ne sont pas précédés par un littéral non encadré (*truncation*).

Définition 22 *(SOLE-résolution).* La procédure qui permet de trouver des SOLE-déductions est appelée *SOLE-résolution.*

Théorème 10 *(Validité et Complétude de la SOLE-résolution).* Soient Σ un ensemble de clauses contenant $x = x$, C une clause et $\mathcal{P}$ un champ de production.

(1) Validité de la SOLE-résolution : Si une clause S est dérivée en utilisant une SOLE-déduction de $\Sigma + C$ et $\mathcal{P}$, alors S appartient à $Th\_E_P(\Sigma \cup \{C\})$.

(2) Complétude de la SOLE-résolution : Si l'on a $T \notin Th\_E_P(\Sigma)$ et $T \in Th\_E_P(\Sigma \cup \{C\})$, alors il existe une SOLE-déduction d'une clause S de $\Sigma + C$ et $\mathcal{P}$ telle que S subsume T.

Preuve :
(Complétude).
La preuve est basée sur la complétude de la SOL-résolution pour les théories avec égalité (Théorème 9). On montre qu'une SOL-déduction peut être transformée en une SOLE-déduction. $T \notin Th\_E_P(\Sigma)$ et $T \in Th\_E_P(\Sigma \cup \{C\})$ $\Longleftrightarrow$ il existe une SOL-déduction de la clause S de $\Sigma \cup K^*(\Sigma) \cup K^*(\neg T) \cup K^*(C) + C$ et $\mathcal{P}$ telle que S subsume T, (Théorème 9). Notons $D_0, D_1, \ldots, D_n$ une telle SOL-déduction.
Si D_{k_i} est une clauses ordonée obtenue par la règle **resolve** à partir de D_{k_i-1} et B_{k_i-1} $(1 \leq i \leq m)$, où B_{k_i-1} est une clause appartenant à $K^*(\Sigma) \cup K^*(C) \cup K^*(\neg T)$. Alors la clause D_{k_i} peut être obtenue en appliquant la règle **E-inférence**. En fait, il y a seulement deux cas :

1. Si D_{k_i-1} est de la forme $< P_{k_i-1}, L[t] \vee C_i >$, B_{k_i-1} est nécessairement de la forme $\neg(x = y) \vee \neg L[x] \vee L[y]$. Si cela est le cas D_{k_i} est obtenu par l'application de E-lit et D_{k_i} est $< P_{k_i}, \neg(t = y) \vee L[y] \vee \boxed{L[t]} \vee C_i >$.

2. Si D_{k_i-1} est de la forme $< P_{k_i-1}, r = s \vee C_i >$, B_{k_i-1} est nécessairement de la forme $\neg(x = y) \vee \neg p[x] \vee p[y]$. Si cela est le cas D_{k_i} est obtenu par l'application de E-equ et D_{k_i} est $< P_{k_i}, \neg p[r] \vee p[s] \vee \boxed{(r = s)} \vee C_i >$.

Dans tous les cas la clause D_{k_i} peut être inférée de D_{k_i-1} par l'application de la **E-inférence**.
Donc, il existe une SOLE-déduction $D_0, D_1, \ldots, D_n$, de $\Sigma \cup \{C\}$ et $\mathcal{P}$ avec C comme clause initiale, telle que $D_0 =< \Box, \vec{C} >$, $D_n =< S, \Box >$ et S subsumé T.
(Validité).
Si dans la SOLE-déduction de S les deux règles de E-inférence ne sont pas utilisées, la déduction de S est une SOL-déduction, et à partir du Théorème 4.7 dans [Ino92a], S appartient à $Th_P(\Sigma \cup \{C\})$, et donc S appartient à $Th\_E_P(\Sigma \cup \{C\})$.
Si dans la SOLE-déduction de S les règles E-lit et E-equ sont utilisées, le résolvant est une conséquence logique de la clause antécédent et un axiome correspondent de $K^*(\Sigma) \cup K^*(C) \cup K^*(\neg T)$. En effet, dans le cas de E-lit, $\neg(t = x) \vee L[x] \vee C_i$ est une conséquence logique de $L[t] \vee C_i$ et $\neg(x' = x) \vee \neg L[x'] \vee L[x]$, laquelle est dans $K^*(\Sigma) \cup K^*(C) \cup K^*(\neg T)$. Dans le cas de E-equ, $\neg L[r] \vee L[s] \vee C_i$ est une conséquence logique de $r = s \vee C_i$ et $\neg(x' = x) \vee \neg L[x'] \vee L[x]$. A partir du Lemme 7, les conséquences générées à partir de $K^*(\Sigma) \cup K^*(C) \cup K^*(\neg T)$ sont les mêmes que celles générées à partir de $K(\Sigma) \cup K^*(C)$. Donc, Toutes les clauses dérivées avec E-lit et E-equ sont dans $Th\_E_P(\Sigma \cup \{C\})$. ■

4.6.2 Exemples

Le premier exemple permet de montrer comment la SOLE-résolution peut être utilisée pour prouver des propriétés des théories avec égalité qui ne sont pas triviales à démontrer "à la main". Ici on veut seulement générer la clause vide.

Dans cet exemple, un enseignant pose le problème suivant aux élèves 'exprimer en logique du premier ordre l'affirmation *il existe une et seulement une personne qui aime Paul*, à l'aide prédicat $A_P(x)$ qui signifie intuitivement x *aime Paul*'.

La plupart des élèves ont exprimé l'affirmation avec la formule suivante :

$$\Phi = \exists x A_P(x) \wedge \forall x \forall y (A_P(x) \wedge A_P(y) \rightarrow x = y),$$

mais un élève a proposé la formule suivante :

$$\Psi = \exists x \forall y (A_P(y) \leftrightarrow x = y).$$

L'enseignant souhaite vérifier s'il existe une équivalence logique entre Ψ et Φ.

Nous avons implémenté la SOLE-résolution. Nous avons prouvé de façon automatique qu'en effet il y a une équivalence. Ici on donne la preuve obtenue de Ψ implique Φ, i.e. la clause vide est inférée à partir $\Psi \cup \neg\Phi$.

La forme clausale de Ψ est : $\{y = \gamma \vee \neg A_P(y),\ \neg(y = \gamma) \vee A_P(y)\}$, et celle de $\neg\Phi$ est $\{\neg A_P(x) \vee A_P(\alpha),\ \neg A_P(x) \vee A_P(\beta),\ \neg A_P(x) \vee \neg(\alpha = \beta)\}$.

Alors, on a :

$\Sigma = \{(\mathbf{1})\ y = \gamma \vee \neg A_P(y),\ (\mathbf{2})\ \neg(y = \gamma) \vee A_P(y),\ (\mathbf{3})\ \neg A_P(x) \vee A_P(\alpha),$
$(\mathbf{4})\ \neg A_P(x) \vee A_P(\beta),\ (\mathbf{5})\ x = x\}$

Nous utilisons comme clause initiale : $\neg A_P(x) \vee \neg(\alpha = \beta) \in \neg\Phi$. Le champ de production $\mathcal{P}$ est vide. Nous avons obtenu la SOLE-déduction ci-dessous de la clause vide.

(D_0) $< \square, \neg A_P(x) \vee \neg(\alpha = \beta) >$ clause initiale

(D_1) $< \square, \neg(x = \gamma) \vee \boxed{\neg A_P(x)} \vee \neg(\alpha = \beta) >$ rés D_0 et 2

(D_2) $< \square, \neg(\alpha = \beta) >$ rés D_1 et 5

(D_3) $< \square, \neg(\alpha = x) \vee \neg(\beta = x) \vee \boxed{\neg(\alpha = \beta)} >$ E-lit

(D_4) $< \square, \neg A_P(\alpha) \vee \boxed{\neg(\alpha = \gamma)} \vee \neg(\beta = \gamma) \vee \boxed{\neg(\alpha = \beta)} >$ rés D_3 et 1

(D_5) $< \square, \neg A_P(x) \vee \boxed{\neg A_P(\alpha)} \vee \boxed{\neg(\alpha = \gamma)} \vee \neg(\beta = \gamma) \vee \boxed{\neg(\alpha = \beta)} >$ rés D_4 et 3

(D_6) $< \square, \neg(x = \gamma) \vee \boxed{\neg A_P(x)} \vee \boxed{\neg A_P(\alpha)} \vee \boxed{\neg(\alpha = \gamma)} \vee \neg(\beta = \gamma) \vee \boxed{\neg(\alpha = \beta)} >$ rés D_5 et 2

(D_7) $< \square, \neg(\beta = \gamma) \vee \boxed{\neg(\alpha = \beta)} >$ rés D_6 et 5

(D_8) $< \square, \neg A_P(\beta) \vee \boxed{\neg(\beta = \gamma)} \vee \boxed{\neg(\alpha = \beta)} >$ rés D_7 et 1

(D_9) $< \square, \neg A_P(x) \vee \boxed{\neg A_P(\beta)} \vee \boxed{\neg(\beta = \gamma)} \vee \boxed{\neg(\alpha = \beta)} >$ rés D_8 et 4

(D_{10}) $< \square, \neg(x = \gamma) \vee \boxed{\neg A_P(x)} \vee \boxed{\neg A_P(\beta)} \vee \boxed{\neg(\beta = \gamma)} \vee \boxed{\neg(\alpha = \beta)} >$ rés D_9 et 2

(D_{11}) $< \Box, \Box >$ rés D_{10} et 5

On considère maintenant un autre exemple dans le contexte de la robotique. Celui-ci permet de montrer comment peut être utilisée la SOLE-résolution pour inférer des conséquences qui satisfont une certaine propriété. La propriété à satisfaire est donnée par le champ de production.

Les prédicats $R(x)$ et $O(x)$ signifient intuitivement 'le robot est dans la position x' et 'il y a un obstacle dans la position x' respectivement. On suppose qu'il y a un obstacle juste une unité en face du robot (formule (a)), et que s'il y a un obstacle alors l'obstacle occupe nécessairement la position 3 ou 7 (formule (b)).

(a) $\exists x \exists y \exists z (R(x) \wedge O(y) \wedge Plus(x, 1, z) \wedge z = y)$.

(b) $\forall x (O(y) \rightarrow y = 3 \vee y = 7)$.

On souhaite connaître la position du robot.

Afin d'éviter le traitement des opérations arithmétiques on introduit des prédicats représentant les opérateurs arithmétiques, ainsi on peut utiliser la SOLE-résolution pour déduire les contraintes arithmétiques et différer leur traitement. Ici on a introduit le prédicat $Plus(x, y, z)$ qui signifie intuitivement $z = x + y$. Ainsi au lieu d'utiliser la formule $\exists x \exists y \exists z (R(x) \wedge O(y) \wedge y = x + 1)$ pour représenter la première hypothèse (cf. les exemples de la Section 4.3.3) on utilise la formule (a), et on se propose de déduire des formules fournies avec le prédicat $Plus$. Le champ de production est donc $< L, Cond >$, où L est l'ensemble des instances du littéral $Plus(x, y, z)$ et $\neg Plus(x, y, z)$, et la condition $Cond$ est *vraie* (i.e. il y n'a pas de condition).

Ainsi, on a :

$\Sigma \cup C = \{(\mathbf{1})\ R(\alpha),\ (\mathbf{2})\ O(\beta),\ (\mathbf{3})\ Plus(\alpha, 1, \gamma),\ (\mathbf{4})\ \gamma = \beta,$
$(\mathbf{5})\ \neg O(y) \vee y = 3 \vee 7 = y,\ (\mathbf{6})\ x = x\}.$

Dans (5) nous avons permuté les arguments de l'égalité $y = 7$ afin de montrer comment est traite la symétrie de l'égalité dans la SOLE-déduction.

La clause initiale choisie est (5). Nous avons obtenu la SOLE-déduction suivante :

D_0. $< \Box, \neg O(y) \vee y = 3 \vee 7 = y >$ clause initiale

D_1. $< \Box, \neg O(x) \vee \neg(y = x) \vee \boxed{\neg O(y)} \vee y = 3 \vee 7 = y >$ E-lit

D_2. $< \Box, \neg(y = \beta) \vee \boxed{\neg O(y)} \vee y = 3 \vee 7 = y >$ rés D_1 et 2

D_3. $< \Box, \gamma = 3 \vee 7 = \gamma >$ rés D_2 et 4

D_4. $< \Box, \neg Plus(x, y, \gamma) \vee Plus(x, y, 3) \vee \boxed{\gamma = 3} \vee 7 = \gamma >$ E-equ

D_5. $< \Box, Plus(\alpha, 1, 3) \vee \boxed{\gamma = 3} \vee 7 = \gamma >$ rés D_4 et 3

D_6. $< Plus(\alpha, 1, 3), 7 = \gamma >$ skip

D_7. $< Plus(\alpha, 1, 3), Plus(x, y, 7) \vee \neg Plus(x, y, \gamma) \vee \boxed{7 = \gamma} >$ E-equ

D_8. $< Plus(\alpha, 1, 3) \vee Plus(x, y, 7), \neg Plus(x, y, \gamma) \vee \boxed{7 = \gamma} >$ skip

D_9. $< Plus(\alpha, 1, 3) \vee Plus(\alpha, 1, 7), \Box >$ rés D_8 et 3

La signification intuitive de la clause dérivée est $\alpha + 1 = 3 \vee \alpha + 1 = 7$. Un programme spécifique pour le traitement des opérateurs arithmétiques (contraintes) peut être appelé en prenant comme argument la clause obtenue de la SOLE-résolution et en donnant comment résultat la position α du robot, i.e. $\alpha = 2 \vee \alpha = 6$.

Chapitre 5

Implémentation de la déduction automatique dans le calcul des situations

5.1 Théories complètes

Dans cette section on présente une méthode d'implémentation des théories de l'action (cf. Section 2.4 et Section 3.3.2). Cette méthode utilise le langage Prolog. Dans le chapitre précédent on a mentionné les résultats qui justifient cette implémentation (cf. Section 4.2.5). L'implémentation est restreinte à un certain type de théories, celles qui sont compatibles avec Prolog : les théories complètes.

Dans cette approche il y a deux fortes limitations qui sont les hypothèses implicites faites pour toute théorie complète. Celle d'unicité des noms et celle du monde fermé.

5.1.1 Hypothèse d'unicité des noms

L'hypothèse d'unicité des noms affirme intuitivement que différents termes dénotent différents éléments du domaine. Elle offre une solution pour la démonstration de l'inégalité, car cette dernière n'est pas explicitement démontrable dans Prolog.

Par exemple, si l'on a l'expression 'tout déchet qui n'est pas organique n'est pas biodégradable', exprimée par :

$$\forall x\,(O(x) \wedge \neg(x = organique) \rightarrow \neg B(x)),$$

sachant que 'plastique est un déchet', on ne peut pas déduire 'plastique n'est pas biodégradable'. Car rien ne permet de déduire $\neg(plastique = organique)$.

L'ajout aux axiomes de toutes les formules exprimant que deux noms différents correspondent à des individus différents ($\neg(plastique = organique)$, $\neg(verre = organique)$, etc.) est appelé *hypothèse d'unicité de noms*. Cette méthode est souvent utilisée implicitement dans les langages de programmation et les bases de données.

5.1.2 Hypothèse du monde fermé

Comme on l'a vu dans le chapitre précédent, les théories qui peuvent être représentées par un programme Prolog doivent contenir seulement des axiomes correspondant à des définitions d'extensions des prédicats. Donc on ne peut pas exprimer des informations incomplètes telles que des faits existentiels ou des disjonctions de faits. Or si un formalisme est basé sur une théorie représentant des informations complètes du monde, il pose le problème de l'explicitation de toutes les informations du monde. Par exemple, si tout ce qui n'est pas vrai constitue une grande quantité d'information, on se pose le problème de décrire explicitement une grande quantité de faits négatifs. Une solution pour exprimer ce type d'informations (faits négatifs), proposé par Reiter dans [Rei78], consiste à considérer cette information comme implicite grâce à l'hypothèse du monde fermé. Cette hypothèse étend la théorie en ajoutant des instances closes de la forme $\neg A$ si leur instance close A correspondante ne peut pas être déduite de la théorie. Une façon simple de décrire cette hypothèse est :

Si A n'est pas déductible, alors inférer $\neg A$, où A est un atome.

Ce qui veut dire intuitivement 'tout ce qui est inconnu et qui ne peut pas être déduit est considéré "non-vrai" '. Par exemple, si la seule information que l'on connaît sur la porte est qu'elle est bleue, on en déduira que la porte n'est pas fermée. L'hypothèse du monde fermé est un raisonnement non-monotone car l'ajout de nouvelles informations peut invalider des conclusions précédentes sans introduire des incohérences. D'ailleurs, l'ajout ou la suppression d'information ne demandera aucune modification de la représentation globale.

En revanche, cette hypothèse produit des contradictions si l'information dépasse le cadre des clauses de Horn. Les disjonctions peuvent produire des incohérences. Par exemple, si la seule information que l'on possède est $A \vee B$, on peut déduire $\neg(A \vee B)$ grâce à l'hypothèse du monde fermé.
Preuve :

(1) $\neg A$	hyp monde fermé
(2) $\neg B$	hyp monde fermé
(3) $\neg A \wedge \neg B$	1 et 2
(4) $\neg(A \vee B)$	loi de Morgan 3

L'hypothèse du monde fermé est utilisée aussi dans modélisation des bases de données, dont l'information qui n'est pas donnée explicitement est considérée comme étant fausse.

5.1.3 Transformations de Lloyd-Topor

Supposons la définition des deux prédicats suivants :

$\forall x\,(homme(x) \leftrightarrow x = Adam \vee x = Romeo \vee x = Cesar)$ et
$\forall x \forall y\,(marie(x,y) \leftrightarrow$
$(x = Adam \wedge y = Eva) \vee (x = Romeo \wedge y = Juliette) \vee (x = Cesar \wedge y = Cleopatre))$

Les parties des équivalences utilisées par Prolog sont :

$$\forall x\,(x = Adam \vee x = Romeo \vee x = Cesar \rightarrow homme(x)) \tag{5.1}$$

$$\forall x \forall y\,((x = Adam \wedge y = Eva) \vee (x = Romeo \wedge y = Juliette) \vee (x = Cesar \wedge y = Cleopatre) \rightarrow marie(x, y)) \tag{5.2}$$

Prenons l'implication (5.1), elle peut être réécrite comme suit :

$$\forall x\,(x = Adam \rightarrow homme(x) \wedge x = Romeo \rightarrow homme(x) \wedge x = Cesar \rightarrow homme(x))$$

d'où on infère :

$$\forall x\,(x = Adam \rightarrow homme(x))$$
$$\forall x\,(x = Romeo \rightarrow homme(x))$$
$$\forall x\,(x = Cesar \rightarrow homme(x))$$

Maintenant on peut prouver que $\forall x\,(x = Adam \rightarrow homme(x))$ est équivalent à $homme(Adam)$.
Preuve :
($\Rightarrow$)

(1) $\forall x\,(x = Adam \rightarrow homme(x))$	hypothèse
(2) $Adam = Adam$	identité
(3) $homme(Adam)$	MP 1 et 2

($\Leftarrow$)
Par réfutation de $homme(Adam)$ et $\neg\forall x\,(x = Adam \rightarrow homme(x))$. Après la transformation des formules en forme clausale, on a les hypothèses suivantes : $\{homme(Adam), \alpha = Adam, \neg homme(\alpha)\}$, d'où on infère la clause vide.

(1) $homme(Adam)$	hypothèse
(2) $\alpha = Adam$	hypothèse
(3) $homme(\alpha)$	paramodulation 1 et 2
(4) $\neg homme(\alpha)$	hypothèse
(5) $\square$	résolution 3 et 4

De façon similaire on peut démontrer les deux équivalences :

$$\forall x\,((x = Romeo \rightarrow homme(x)) \leftrightarrow homme(Romeo)) \text{ et}$$
$$\forall x\,((x = Cesar \rightarrow homme(x)) \leftrightarrow homme(Cesar)).$$

De façon similaire on peut aussi démontrer que $\forall x \forall y\,((x = Adam \wedge y = Eva) \vee (x = Romeo \wedge y = Juliette) \vee (x = Cesar \wedge y = Cleopatre) \rightarrow marie(x, y))$ peut être écrit en forme clausale comme suit :

$$\{marie(Adam, Eva),\ marie(Romeo, Juliette),\ marie(Cesar, Cleopatre)\}.$$

Donc le programme Prolog résultant est :

```
homme(adam).
homme(romeo).
homme(cesar).

marie(adam,eva).
marie(romeo,juliette).
marie(cesar,cleopatre).
```

On peut généraliser ce résultat pour les prédicat n-aires.

On peut remarquer que l'implication (5.1) n'a pas le format d'une clause Prolog (disjonction des littéraux). Cependant on a effectué une transformation qui remplace cette formule par un ensemble de formules équivalentes satisfaisant le format Prolog. On a appliqué l'une des transformations proposées par Lloyd et Topor [Llo87] qui permettent de transcrire une formule quantifiée universellement (définissant un prédicat) en une disjonction de littéraux. En général les transformations sont des équivalences logiques, sauf pour des formules telles que :

$$\forall x \forall y (\forall e \, (\mathit{élément}(e, x) \rightarrow \mathit{élément}(e, y)) \rightarrow \mathit{sousensemble}(x, y)).$$

Dans ce cas l'introduction d'un nouveau symbole de prédicat est nécessaire pour la transformation. Par exemple, la définition du prédicat *sousensemble* après les transformations est :

$$\neg \mathit{auxiliaire}(x, y) \rightarrow \mathit{sousensemble}(x, y),$$
$$\mathit{élément}(e, x) \wedge \mathit{élément}(e, y) \rightarrow \mathit{auxiliaire}(x, y).$$

Les transformations de Lloyd-Topor peuvent aussi être appliquées aux requêtes qui ne satisfont pas la syntaxe Prolog (conjonction des littéraux). Le résultat de Clark est toujours valide pour des théories qui ont suivi des transformations Lloyd-Topor.

Dans l'implémentation que nous présentons, afin d'éviter des formules difficiles à lire et à comprendre intuitivement, parfois nous utilisons des formules qui n'ont pas le format qui satisfait la syntaxe Prolog. Nous utilisons les outils qu'offre Quintus pour représenter une formule quantifiée universellement. En particulier nous utilisons le prédicat prédéfini (`;`) qui représente la disjonction ($\vee$). D'ailleurs dans les exemples que nous présentons on peut facilement faire la correspondance entre un programme et sa représentation logique.

5.1.4 Une pseudonégation par échec

Le prédicat représentant la négation par échec dans Quintus (`not`) est très restrictif, il arrête l'exécution du programme dès qu'il est appelé en ayant comme argument une formule non close. Cependant ce langage offre un autre prédicat de négation (`\+`), qui est plus "tolérant". S'il est appelé en ayant comme argument une formule non close, Quintus suspend son évaluation et

travaille avec d'autres littéraux jusqu'à ce que la formule devienne close. Malheureusement il y a un inconvénient. Le test de la négation par échec est effectué aussi sur des formules non closes.

Si l'interpréteur Prolog essaye une négation par échec sur des formules non closes l'erreur décrite ci-dessous peut se produire. Supposons la théorie suivante :

$$\forall x\,(femme(x) \leftrightarrow x = Eva)$$
$$\forall x\,(homme(x) \leftrightarrow x = Adam)$$

qui signifie intuitivement que sur la terre la seule femme qui existe est Eva et le seul homme est Adam. Le programme Prolog obtenu est :

```
femme(eva).
homme(adam).
```

Si l'on pose la question : est-ce qu'il y a quelqu'un qui ne soit pas femme et qui soit homme sur la terre? représentée en Prolog par[1]:

```
|?- not femme(X), homme(X).
```

En principe, le programme doit réussir en donnant la réponse `X=adam` car on peut inférer $\neg femme(adam) \wedge homme(adam)$ de la théorie. Mais la réponse est `no`, i.e. il n'y a pas d'homme sur la terre qui ne soit pas femme. L'erreur consiste en évaluer `not femme(X)` (évaluation du but de gauche à droite) qui n'est pas clos. L'interpréteur appelle `femme(X)` qui réussit avec `X=eva` donc sa négation échoue, et le but échoue aussi. Par contre, la réponse à la question `|?- homme(X), not femme(X).` donne le résultat attendu.

Etant donné que la plupart des interpréteurs Prolog, parmi lesquels Quintus pour le prédicat (`\+`), font ce mauvais traitement de la négation, la garantie de la correction de l'implémentation est de la responsabilité du programmeur. Il doit s'assurer que les clauses constituant les programmes permettent que les littéraux négatifs soient évalués à la fin pour obtenir auparavant des instaciations des variables. D'autre part la responsabilité de l'utilisateur consiste à poser la "bonne question" (placer les littéraux négatifs à la fin de la requête) s'il souhaite obtenir une réponse concordant avec la théorie.

Dans notre cas, il faut faire un compromis entre l'usage de (`\+`) qui a comme conséquence l'acceptation des négations de formules contenant des variables non instanciées ou l'usage de (`not`) qui a comme conséquence l'acceptation des négations de formules closes seulement. Il faut souligner que Quintus n'attend pas l'instaciation des variables par d'autres littéraux dans l'évaluation du prédicat (`not`). Au moment de trouver un prédicat (`not`), il vérifie si son argument est clos, si ce n'est pas le cas il arrête l'exécution donnant comme réponse un message d'erreur. C'est une limitation très forte. Par exemple, supposons le programme suivant :

[1] On utilise le préfixe `|?-` pour indiquer que la formule qui le suit est une question en Quintus.

```
poss(avancer,S):-
                  not ( (
                        position(X,S),
                        obstacle(Y,S),
                        Y is X+1
                      ) ) .

position(3,s0).

obstacle(5,s0).
```

où la définition de `poss` veut dire intuitivement que le robot peut avancer si[2] il n'y a pas d'obstacle juste une unité devant lui. Les faits veulent dire que le robot est dans la position 3 à l'état initial et qu'il y a un obstacle dans la position 5 à l'état initial. La question posée est 'est-ce que le robot peut avancer à l'état initial ?', représentée par `|?- poss(avancer,s0).` La réponse donnée par Quintus est `Instantiation error in argument 1 of (not)/1` (erreur d'intanciation dans le premier argument de not). Par contre, si l'on utilise (`\+`) pour représenter la négation, la réponse obtenue est `yes` qui correspond à ce que l'on attend.

Pour simuler `not p` on doit être sûr de la complétude de l'information sur `p`, i.e. le programme doit répondre affirmativement dans un temps fini pour toute instance close de `p` vraie, et doit répondre négativement dans un temps fini pour toute instance close de `p` fausse. Considérons l'exemple suivant :

```
homme(paul)
homme(serge)

femme(nathalie)
femme(marilyne)

marie(serge,marilyne)

marie(X):- marie(X,_).
marie(X):- marie(_,X).

humain(X):-homme(X).
humain(Y):-femme(Y).
```

La question 'qui n'est pas marié?' représentée par `| ?- \+ marie(Qui).` est résolue négativement car il y des instances de `marie(Qui)` qui sont démontrables (`serge` et `marilyne`). Si l'on veut obtenir les instances qui ne satisfont pas une certaine propriété, il faut numéroter les candidats et faire la vérification pour

[2] Grâce à la completion de Clark on peut dire 'si et seulement si'.

chacun. Ainsi une "bonne question" qui donne le résultat attendu est `| ?- humain(Qui), \+ marie(Qui).` qui veut dire intuitivement 'quel humain n'est pas marié?'.

Par contre, la question `| ?- \+ divorce(Qui).` est résolue avec une solution positive[3] car il n'y pas d'instance de `divorce(Qui)` qui soit démontrable.

Remarquons que $\neg divorce(paul)$ n'est pas une conséquence logique de la théorie correspondant au programme. Mais la théorie étendue avec l'hypothèse du monde fermé permet de déduire $\neg divorce(paul)$, ce qui correspond à une réponse affirmative de Prolog à la question `| ?- \+ divorce(paul).`

5.1.5 Prolog pour des actions atomiques

Les schémas d'axiomes des préconditions ($Poss(a,s) \leftrightarrow \pi_a(s)$) et ceux des axiomes de changement d'état ($F(do(a,s)) \leftrightarrow \Upsilon_F^+(a,s) \vee F(s) \wedge \neg\Upsilon_F^-(a,s)$) sont exprimés sous forme de définitions (cf. formule 4.2), mais une fois instanciées les formules ne sont plus des définitions des extensions de prédicats. Par exemple, une instance de $Poss(a,s) \leftrightarrow \pi_a(s)$ est $\forall s(Poss(avancer,s) \leftrightarrow \neg\exists x\exists y\,(position(x,s) \wedge obstacle(y,s) \wedge y = x+1))$ qui contient un argument qui n'est pas une variable, celle d'*avancer*.

Dans [Rei99] Reiter affirme que l'ensemble des axiomes des préconditions peut être exprimé sous forme d'une définition (formule 5.3). Si $A_1, \ldots A_n$ l'ensemble des actions du langage du calcul des situations $\mathcal{L}_{\mathcal{CS}}$, et si les formules suivantes :

$$\begin{aligned} Poss(A_1,s) &\leftrightarrow \Pi_{A_1}(s), \\ &\vdots \\ Poss(A_n,s) &\leftrightarrow \Pi_{A_n}(s). \end{aligned}$$

constituent l'ensemble d'axiomes des préconditions, alors la définition de $Poss$ est donnée par :

$$\forall a\forall s\,(Poss(a,s) \leftrightarrow a = A_1 \wedge \Pi_{A_1}(s) \vee \ldots \vee a = A_n \wedge \Pi_{A_n}(s)) \qquad (5.3)$$

La partie d'équivalence requise $\forall a\forall s\,(a = A_1 \wedge \Pi_{A_1}(s) \vee \ldots \vee a = A_n \wedge \Pi_{A_n}(s) \rightarrow Poss(a,s))$ est équivalente à l'ensemble suivant :

$$\begin{aligned} \Pi_{A_1}(s) &\rightarrow Poss(A_1,s) \\ &\vdots \\ \Pi_{A_n}(s) &\rightarrow Poss(A_n,s) \end{aligned}$$

donc l'affirmation de Reiter garantie une implémentation valide pour les axiomes des préconditions.

En ce qui concerne les fluents, il sont caractérisés par les axiomes de changement d'état et les axiomes décrivant la situation initiale. Reiter propose la

[3] Supposons que le prédicat `divorce` avec un argument est défini.

définition d'un fluent F suivante. Si l'axiome qui décrit le fluent F dans la situation initiale est $F(S_0) \leftrightarrow \phi_F(S_0)$, alors la définition de F est donnée par :

$$\forall s\,(F(do(a,s)) \leftrightarrow s = S_0 \wedge \phi_F(S_0) \vee \exists a \exists s'\,(s = do(a,s') \wedge \Upsilon_F^+(a,s') \vee F(s') \wedge \neg\Upsilon_F^-(a,s')))$$

d'où la partie qui nous intéresse est équivalente à :

$$\phi_F(S_0) \rightarrow F(S_0),$$
$$\Upsilon_F^+(a,s) \vee F(s) \wedge \neg\Upsilon_F^-(a,s) \rightarrow F(do(a,s)).$$

Donc les conditions pour garantir une implémentation valide des axiomes de changement d'état, et celles des axiomes décrivant la situation initiale ainsi que celles des axiomes des préconditions, sont satisfaites.

La complétude imposée aux axiomes oblige à donner des informations complètes de la situation initiale, ainsi par exemple, il n'est pas possible de représenter des expressions telles que $F(A,S_0) \vee F(B,S_0)$ ou $\exists x\, F(x,S_0)$.

Reiter démontre le théorème (cf. Théorème 5.3.1 dans [Rei99]) sur lequel est basé un des résultats les plus importants de son approche : la validité d'implémentation de démonstrations des formules régressables dans une théorie de l'action. Le résultat peut être énoncé comme suit.

Propriété 2 Soit $\mathcal{D}$ une théorie de l'action qui satisfait les conditions suivantes : le langage $\mathcal{L}_{\mathcal{CS}}$ de la théorie ne contient pas des fonctions fluent, le nombre de fluents est fini, le nombre d'actions est fini et la théorie initiale $\mathcal{D}_{S_0}$ est complète. Si $\mathcal{P}$ est le programme Prolog constitué des clauses obtenues par des transformations en leurs formes clausales des formules suivantes :

1. Pour chaque définition de prédicat de $\mathcal{D}_{S_0}$ de la forme $P(\vec{(}x)) \leftrightarrow \phi_P(\vec{(}x))$, on considère la formule suivante :
$$\phi_P(\vec{(}x)) \rightarrow P(\vec{(}x)).$$

2. Pour chaque définition de $\mathcal{D}_{S_0}$ de la forme[4] $F(S_0) \leftrightarrow \phi_F(S_0)$, on considère la formule suivante :
$$\phi_F(S_0) \rightarrow F(S_0).$$

3. Pour chaque axiome de préconditions d'action de $\mathcal{D}_{pa}$ de la forme $(Poss(A,s) \leftrightarrow \pi_A(s))$, on considère la formule suivante :
$$\pi_A(s) \rightarrow Poss(A,s).$$

4. Pour chaque axiome de changement d'état du fluent F de $\mathcal{D}_{ce}$ de la forme $(F(do(a,s)) \leftrightarrow \Upsilon_F^+(a,s) \vee F(s) \wedge \neg\Upsilon_F^-(a,s))$, on considère la formule suivante :
$$\Upsilon_F^+(a,s) \vee F(s) \wedge \neg\Upsilon_F^-(a,s) \rightarrow F(do(a,s)).$$

Alors $\mathcal{P}$ est une implémentation valide de la théorie $\mathcal{D}$ pour la démonstration des formules régressables. i.e. si un interpréteur standard Prolog répond affirmativement à une requête représentant une formule régressable alors cette formule régressable est conséquence logique de $\mathcal{D}$, et s'il répond négativement, la négation de la formule régressable est conséquence logique de $\mathcal{D}$.

[4] On omet les arguments autres que ceux des situations.

Exemples

Considérons un scénario dans le contexte de la robotique. Supposons un robot qui peut se déplacer sur une piste droite, les actions qu'il peut effectuer sont: *avancer* qui lui permet d'avancer d'une unité, *reculer* qui lui permet de reculer d'une unité. L'exécution d'*avancer* ou *reculer* change sa position d'une unité. L'existence des obstacles (tels que branches d'arbres arrachées par le vent ou des petits lapins qui se promènent) placés au long de la piste est possible. La formalisation dans le calcul des situations est la suivante:

- Les axiomes des préconditions des actions sont:

  $Poss(avancer, s) \leftrightarrow \neg\exists x \exists y\,(position(x, s) \wedge obstacle(y, s) \wedge y = x + 1)$

  $Poss(reculer, s) \leftrightarrow \neg\exists x \exists y\,(position(x, s) \wedge obstacle(y, s) \wedge y = x - 1)$

- Les axiomes de changement d'état sont:

$position(x, do(a, s)) \leftrightarrow$

$[a = avancer \wedge position(x-1, s) \vee a = reculer \wedge position(x+1, s)]$

$\vee position(x, s) \wedge \neg[(a = avancer \vee a = reculer) \wedge position(x, s)]$

$obstacle(x, do(a, s)) \leftrightarrow obstacle(x, s).$

Supposons que le robot est dans la position 3 dans la situation initiale et qu'il y a un obstacle dans la position 5 dans la situation initiale. Le programme Prolog représentant ce scénario est le suivant:

```
/* Primitive control actions */

primitive_action(avancer).        /* Move the robot forward. */
primitive_action(reculer).        /* Move the robot backward. */


/*Preconditions for Primitive Actions */

poss(avancer,S):-
                  \+ ( (
                        position(X,S),
                        obstacle(Y,S),
                        Y is X+1
                      ) ) .

poss(reculer,S):-
                  \+ ( (
                        position(X,S),
                        obstacle(Y,S),
                        Y is X-1
                      ) ) .
```

```
/* Successor state axioms for primitive fluents. */

position(X,do(A,S)):-  position(Y,S),
                       (
                         (A=avancer,X is Y + 1) ;
                         (A=reculer,X is Y - 1)
                       );
                       (
                         position(X,S),
                         \+ A=avancer,
                         \+ A=reculer
                       ) .

obstacle(X,do(A,S)):-
                      obstacle(X,S).

/* Restore suppressed situation arguments.   */

restoreSitArg(position(N),S,position(N,S)).
restoreSitArg(obstacle(N),S,obstacle(N,S)).


/* Initial situation */

position(3,s0).

obstacle(5,s0).
```

Maintenant on pose la question 'quelle sera la position du robot s'il avance deux fois?':

```
| ?- position(X,do(avancer,do(avancer,s0))).
```

la réponse est `X = 5`. Pour répondre à cette question deux appels à l'axiome de changement d'état de *position* sont effectués mais les préconditions à satisfaire pour pouvoir effectuer l'action *avancer* ne sont pas vérifiées. Des appels à *Poss* sont nécessaires pour cette vérification. La première action *avancer* satisfait les préconditions, ainsi la question `| ?- poss(avancer,s0)` est résolue par Quintus donnant la réponse `yes`, mais le deuxième *avancer* ne satisfait pas les préconditions, la réponse à `| ?- poss(avancer,do(avancer,s0))` est donc `no`.

Considérons maintenant la question 'jusqu'à quelle position le robot peut avancer?', on peut utiliser une méthode naïve telle que questionner à l'interpréteur sur la posibilité d'avancer de façon récursive, `| ?- poss(avancer,s0)`, `| ?- poss(avancer,do(avancer,s0))`, `| ?- poss(avancer,do(avancer,do(avancer,s0)))`, etc.

Une façon compacte d'exprimer ce type de questions utilise le langage de programmation GOLOG que nous allons voir maintenant.

5.1.6 Golog pour des actions complexes

Dans la Section 2.6 on a présenté le langage de programmation GOLOG qui est basé sur le calcul des situations. On présente ici un interpréteur de ce langage implémenté en Quintus Prolog. On met aussi en valeur l'avantage du pouvoir d'expression de ce langage en ce qui concerne les actions. L'interpréteur GOLOG que nous utilisons est le suivant[5] :

```
:- dynamic proc/2.            /* Compiler directives. Be sure   */
:- multifile restoreSitArg/3.
:- multifile proc/2.

:- op(800, xfy, [&]).   /* Conjunction */
:- op(850, xfy, [v]).   /* Disjunction */
:- op(870, xfy, [=>]).  /* Implication */
:- op(880,xfy, [<=>]).  /* Equivalence */
:- op(950, xfy, [:]).   /* Action sequence */
:- op(960, xfy, [#]).   /* Nondeterministic action choice */

do(E1 : E2,S,S1) :- do(E1,S,S2), do(E2,S2,S1).
do(?(P),S,S) :- holds(P,S).
do(E1 # E2,S,S1) :- do(E1,S,S1) ; do(E2,S,S1).
do(if(P,E1,E2),S,S1) :- do(?(P) : E1 # ?(-P) : E2,S,S1).
do(star(E),S,S1) :- S1 = S ; do(E : star(E),S,S1).
do(while(P,E),S,S1):- do(star(?(P) : E) : ?(-P),S,S1).
do(pi(V,E),S,S1) :- sub(V,_,E,E1), do(E1,S,S1).      /* Generate a new
             Prolog variable, and substitute it into the action E.  */
do(E,S,S1) :- proc(E,E1), do(E1,S,S1).
do(E,S,do(E,S)) :- primitive_action(E), poss(E,S).

/* sub(Name,New,Term1,Term2): Term2 is Term1 with Name replaced by New. */

sub(X1,X2,T1,T2) :- var(T1), T2 = T1.
sub(X1,X2,T1,T2) :- \+ var(T1), T1 = X1, T2 = X2.
sub(X1,X2,T1,T2) :- \+ T1 = X1, T1 =..[F|L1], sub_list(X1,X2,L1,L2),
                    T2 =..[F|L2].
sub_list(X1,X2,[],[]).
sub_list(X1,X2,[T1|L1],[T2|L2]) :- sub(X1,X2,T1,T2),
                                   sub_list(X1,X2,L1,L2).

/* The holds predicate implements the revised Lloyd-Topor
   transformations on test conditions.  */

holds(P & Q,S) :- holds(P,S), holds(Q,S).
```

[5] Cet interpréteur a été pris de [Rei99], il a été adapté pour fonctioner dans l'environnement de Quintus.

```
holds(P v Q,S) :- holds(P,S); holds(Q,S).
holds(P => Q,S) :- holds(-P v Q,S).
holds(P <=> Q,S) :- holds((P => Q) & (Q => P),S).
holds(-(-P),S) :- holds(P,S).
holds(-(P & Q),S) :- holds(-P v -Q,S).
holds(-(P v Q),S) :- holds(-P & -Q,S).
holds(-(P => Q),S) :- holds(-(P v Q),S).
holds(-(P <=> Q),S) :- holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- holds(some(V,-P),S).
holds(-some(V,P),S) :- \+ holds(some(V,P),S).  /* Negation */
holds(-P,S) :- isAtom(P), \+ holds(P,S).     /* by failure */
holds(all(V,P),S) :- holds(-some(V,-P),S).
holds(some(V,P),S) :- sub(V,_,P,P1), holds(P1,S).

/* The following clause treats the holds predicate for all atoms,
   including Prolog system predicates. For this to work properly,
   the GOLOG programmer must provide, for all atoms tasking a
   situation argument, a clause giving the result of restoring
   its suppressed situation arguments, for example:
         restoreSitArg(ontable(X),S,ontable(X,S)).            */

holds(A,S) :- restoreSitArg(A,S,F), F ;
              \+ restoreSitArg(A,S,F), isAtom(A), A.

isAtom(A) :- \+ (A = -W ; A = (W1 & W2) ; A = (W1 => W2) ;
    A = (W1 <=> W2) ; A = (W1 v W2) ; A = some(X,W) ; A = all(X,W)).

restoreSitArg(poss(A),S,poss(A,S)).
```

Le prédicat, le plus important est[6] `do`, lequel possède trois arguments : une expression d'action du langage GOLOG et deux termes indiquant la situation initiale et la situation finale. Une question s'écrit par exemple `do(e,s0,S)`. Son évaluation produit une situation finale possible `S`. L'action complexe `e` est construite avec les huit expressions d'action GOLOG : la séquence $\mathtt{e}_1 : \mathtt{e}_2$, le test d'un fluent `?(p)`, le choix non déterministe entre deux actions $\mathtt{e}_1 \# \mathtt{e}_2$, la conditionnelle **if**$(\mathtt{p}, \mathtt{e}_1, \mathtt{e}_2)$, la répétition non déterministe **star**`(e)`, l'itération **while**`(p,e)`, l'assignation non déterministe d'arguments d'action `pi(v,e(v))` et les actions primitives déclarées ou les appels de procédure déclarées `a`.

On peut constater que la condition `p`, dans une action de test `?(p)`, peut être soit une formule atomique soit une expression de la forme p_1 & p_2, p_1 `v` p_2, $-p$, p_1 `=>` p_2, p_1 `<=>` p_2, $all(v,p)$, $some(v,p)$, où v est une constante Prolog, et p une condition qui utilise v. Pour évaluer cette condition l'interpréteur utilise la négation par échec pour évaluer la négation logique (-), ainsi que la base de données initiale et les axiomes de changement d'état pour déterminer la valeur de vérité des fluents.

[6] Ne pas confondre avec l'abréviation *Do*, ni avec la fonction *do*, vues précédemment.

Dans la section suivante on analyse une autre façon d'implémenter le test d'une condition qui n'est pas basée sur Prolog mais plutôt sur un démonstrateur général qui ne suppose pas l'hypothèse du monde fermé.

Exemples

Considérons le scénario de la section précédente et reprenons la question 'jusqu'à quelle position le robot peut avancer?' représentée par :

```
| ?- do(while(poss(avancer), avancer),s0,S),position(X,S).
```

l'action complexe "while" signifie intuitivement 'avancer si c'est possible'. La réponse obtenue est :

```
S = do(avancer,s0),
X = 4
```

qui veut dire 'le robot peut avancer jusqu'à la position 4 et pour attendre cette position il lui faut réaliser l'action *avancer*'.

Si nous changeons la théorie initiale et nous considérons maintenant que le robot est dans la position 1 dans la situation initiale. La réponse à la même question est :

```
S = do(avancer,do(avancer,do(avancer,s0))),
X = 4
```

qui veut dire 'le robot peut avancer jusqu'à la position 4 et pour attendre cette position il lui faut réaliser l'action *avancer* trois fois'.

Une question telle que 'jusqu'à quelle position le robot peut reculer?' représentée par :

```
| ?- do(while(poss(reculer), reculer),s0,S),position(X,S).
```

n'a pas de solution affirmative ni d'ailleurs négative puisqu'en principe il n'y a rien qui gêne le robot pour reculer itérativement et donc une boucle "infinie" se produit.

5.1.7 Un interprète pour des théories épistémiques complètes

En ce qui concerne les croyances des agents, elles sont caractérisées par les axiomes de changement d'état des croyances et les axiomes décrivant la situation initiale. Basée sur les résultats de Reiter nous proposons la définition suivante des croyances positives concernant le fluent F. Si l'axiome qui décrit les croyances positives du fluent F dans la situation initiale est $B_i(F(S_0)) \leftrightarrow \phi_{B_iF}(S_0)$, alors la définition de B_iF est donnée par :

$$\forall s\,(B_i(F(do(a,s))) \leftrightarrow$$
$$s = S_0 \wedge \phi_{B_iF}(S_0) \vee \exists a \exists s'\,(s = do(a,s') \wedge \Upsilon^{+}_{B_iF}(a,s') \vee B_i(F(s')) \wedge \neg\Upsilon^{-}_{B_iF}(a,s')))$$

dont la partie qui nous intéresse est équivalente à :

$$\phi_F(S_0) \rightarrow B_i(F(S_0)),$$
$$\Upsilon^{+}_{B_iF}(a,s) \vee B_i(F(s)) \wedge \neg\Upsilon^{-}_{B_iF}(a,s) \rightarrow B_i(F(do(a,s))).$$

De façon similaire nous proposons la définition suivante des croyances négatives concernant le fluent F. Si l'axiome qui décrit les croyances négatives du fluent F dans la situation initiale est $B_i(\neg F(S_0)) \leftrightarrow \phi_{B_i\neg F}(S_0)$, alors la définition de $B_i\neg F$ est donnée par :

$$\begin{aligned}\forall s\,(B_i(\neg F(do(a,s))) \leftrightarrow \\ s = S_0 \wedge \phi_{B_i\neg F}(S_0) \vee \exists a \exists s'\,(s = do(a,s') \wedge \Upsilon^{+}_{B_i\neg F}(a,s') \\ \vee B_i(\neg F(s')) \wedge \neg\Upsilon^{-}_{B_i\neg F}(a,s')))\end{aligned}$$

dont la partie qui nous intéresse est équivalente à :

$$\begin{aligned}&\phi_{\neg}F(S_0) \rightarrow B_i(\neg F(S_0)),\\ &\Upsilon^{+}_{B_i\neg F}(a,s) \vee B_i(\neg F(s)) \wedge \neg\Upsilon^{-}_{B_i\neg F}(a,s) \rightarrow B_i(\neg F(do(a,s))).\end{aligned}$$

Donc les conditions pour garantir une implémentation correcte des axiomes de changement d'état des croyances sont satisfaites.

La complétude imposée aux axiomes oblige à donner des informations des croyances complètes dans la situation initiale.

On peut utiliser le théorème de régression (cf. Théorème 5.3.1 dans [Rei99]) pour étendre le résultat de la validité de l'implémentation des démonstrations des formules régressables au cas des théories de l'action et des croyances. Le résultat peut être énoncé comme suit.

Propriété 3 Soit $\mathcal{T}$ une théorie de l'action qui satisfait les conditions suivantes : le langage $\mathcal{L}_{\mathcal{CS}}$ de la théorie ne contient pas des fonctions fluent, le nombre de fluents est fini, le nombre d'agents est fini, le nombre d'actions est fini et la théorie initiale $\mathcal{D}_{S_0}$ est complète. Si $\mathcal{P}$ est le programme Prolog constitué des clauses obtenues par des transformations en leurs formes clausales des formules énoncées dans la Propriété 2, plus

5. Pour chaque paire d'axiomes de changement d'état des croyances de l'agent i du fluent F de $\mathcal{D}_{cec}$ de la forme :

$$\begin{aligned}&B_i(F(do(a,s))) \leftrightarrow \Upsilon^{+}_{B_iF}(a,s) \vee B_i(F(s)) \wedge \neg\Upsilon^{-}_{B_iF}(a,s)\\ &B_i(\neg F(do(a,s))) \leftrightarrow \Upsilon^{+}_{B_i\neg F}(a,s) \vee B_i(\neg F(s)) \wedge \neg\Upsilon^{-}_{B_i\neg F}(a,s)\end{aligned}$$

on considère les formules suivantes :

$$\begin{aligned}&\Upsilon^{+}_{B_iF}(a,s) \vee B_i(F(s)) \wedge \neg\Upsilon^{-}_{B_iF}(a,s) \rightarrow B_i(F(do(a,s)))\\ &\Upsilon^{+}_{B_i\neg F}(a,s) \vee B_i(\neg F(s)) \wedge \neg\Upsilon^{-}_{B_i\neg F}(a,s) \rightarrow B_i(\neg F(do(a,s)))\end{aligned}$$

Alors $\mathcal{P}$ est une implémentation valide de la théorie $\mathcal{T}$ pour la démonstration des formules régressables. i.e. si un interpréteur standard Prolog répond affirmativement à une requête représentant une formule régressable alors cette formule régressable est conséquence logique de $\mathcal{T}$, et s'il répond négativement, la négation de la formule régressable est conséquence logique de $\mathcal{T}$.

La démonstration est faite en deux étapes d'abord le renommage des modalités par de nouveaux symboles de prédicats. Ainsi $B_i(F(s))$ se réécrit $BiF(s)$, où BiF est un nouveau symbole de prédicat. La deuxième étape consiste à appliquer la Propriété 2 à la théorie résultant du renommage qui ne contient plus de modalité.

Cette démonstration donne la clef pour l'implémentation en Prolog des théories de l'action étendues aux croyances. i.e utiliser un métaprédicat pour représenter les modalités de croyance.

Exemples

Considérons toujours le même scénario. Ajoutons maintenant l'existence d'un deuxième agent, le pilote du robot. Considérons que le déplacement du robot s'effectue pendant la nuit. Le pilote peut reconnaître la présence d'un obstacle s'il allume une lampe (action $allumer\_observer$) et si la distance de l'obstacle ne dépasse pas la limite de visibilité d. La lampe n'est pas toujours allumée car elle consomme des batteries qui sont limitées. Le robot possède un panneau de contrôle qui montre sa position actualisée par le robot chaque fois qu'il en change. Le pilote peut connaître la position du robot en regardant le panneau de contrôle (action $observer\_pos$). Le pilote peut informer le robot de l'existence d'un obstacle dans la position x (action $informer\_obs(x)$). Le robot s'arrête s'il croit qu'il y a un obstacle à une distance rapprochée rd.

Certaines hypothèses ne sont pas réalistes mais notre intérêt principal est de montrer comment un tel scénario peut s'implémenter.

Cette extension du scénario évoque l'évolution du monde physique et celle des croyances du robot et du pilote. L'évolution du monde physique a été considérée dans les sections précédantes. Nous considérons ici les axiomes de changement d'état des croyances. Ceux-ci sont exprimés formellement comme suit :

$$B_r(position(x, do(a, s))) \leftrightarrow$$
$$[a = avancer \wedge B_r(position(x-1, s)) \vee a = reculer \wedge B_r(position(x+1, s))]$$
$$\vee B_r(position(x, s)) \wedge \neg[(a = avancer \vee a = reculer) \wedge B_r(position(x, s))]$$

$$B_r(\neg position(x, do(a, s))) \leftrightarrow$$
$$[(a = avancer \vee a = reculer) \wedge B_r(position(x, s))] \vee$$
$$B_r(position(x, s)) \wedge \neg[a = avancer \wedge B_r(position(x-1, s)) \vee$$
$$a = reculer \wedge B_r(position(x+1, s))]$$

$$B_r(obstacle(x, do(a, s))) \leftrightarrow a = informer\_obs(x) \vee B_r(obstacle(x, s))$$

$$B_r(\neg obstacle(x, do(a, s))) \leftrightarrow \neg a = informer_obs(x) \wedge B_r(\neg obstacle(x, s))$$

$$B_p(position(x, do(a, s))) \leftrightarrow$$

$$a = observer\_pos \wedge position(x,s) \vee \neg a = observer\_pos \wedge B_p(position(x,s))$$

$$B_p(\neg position(x, do(a,s))) \leftrightarrow$$
$$a = observer\_pos \wedge position(Y,s) \wedge \neg x = Y \vee$$
$$\neg a = observer\_pos \wedge B_p(\neg position(x,s))$$

$$B_p(obstacle(x, do(a,s))) \leftrightarrow$$
$$a = allumer\_observer \wedge obstacle(x,s) \wedge position(y,s) \wedge y \leq x \leq y+d \vee$$
$$B_p(obstacle(x,s)) \wedge \neg(a = allumer\_observer \wedge \neg obstacle(x,s) \wedge$$
$$position(y,s) \wedge y \leq x \leq y+d)$$

$$B_p(\neg obstacle(x, do(a,s))) \leftrightarrow$$
$$a = allumer\_observer \wedge \neg obstacle(x,s) \wedge position(y,s) \wedge y \leq x \leq y+d \vee$$
$$B_p(\neg obstacle(x,s)) \wedge \neg(a = allumer\_observer \wedge obstacle(x,s) \wedge$$
$$position(y,s) \wedge y \leq x \leq y+d)$$

Afin de simplifier les axiomes de changement d'état des croyances du pilote par rapport aux obstacles (les deux derniers axiomes), on considère que le robot ne fait qu'*avancer* et que l'éclairage de la lampe se fait toujours avant.

Supposons que le robot est dans la position 1 dans la situation initiale, qu'il y a un obstacle dans la position 6 dans la situation initiale et que le robot croit que sa position est 1 dans la situation initiale.

Comme nous avons vu dans la démonstration de la Propriété 3, les modalités de croyance peuvent être remplacées par un métaprédicat dans l'implémentation. Ici nous utilisons une particularité offerte par les interpréteurs Prolog qui permettent d'utiliser des prédicats comme arguments d'autres prédicats. Ainsi le programme correspondant à la théories est :

```
/*Preconditions for Primitive Actions */

poss(avancer,S):- \+ ( (
                        br(position(X,S)),
                        br(obstacle(Y,S)),
                        Y-X=<2          % rd =2
                      ) ),
                  \+ ( (
                        position(X,S),
                        obstacle(Y,S),
                        Y is X+1
                      ) ) .
poss(reculer,S):-
                  \+ ( (
                        br(position(X,S)),
                        br(obstacle(Y,S)),
```

```
                    X-Y=<2          % rd=2
                ) ),
              \+ ( (
                    position(X,S),
                    obstacle(Y,S),
                    Y is X -1
                ) ) .

poss(allumer_obsever,S).
poss(observer_pos,S).
poss(informer_obs(X),S):-bp(obstacle(X,S)).

/* Primitive control actions */

primitive_action(avancer).
primitive_action(reculer).
primitive_action(allumer_obsever).
primitive_action(observer_pos).
primitive_action(informer_obs(X)).

/* Successor state axioms for primitive fluents. */

position(X,do(A,S)):-  position(Y,S),
                       (
                         (A=avancer,X is Y + 1) ;
                         (A=reculer,X is Y - 1)
                       );
                       (
                         position(X,S),
                         \+A=avancer,
                         \+A=reculer
                       ) .

obstacle(X,do(A,S)):-  obstacle(X,S).

br(position(X,do(A,S))):-  br(position(Y,S)),
                           (
                             (A=avancer,X is Y+1);
                             (A=reculer,X is Y-1)
                           );
                           (
                              br(position(X,S)),
                              \+A=avancer,
                              \+A=reculer
                           ) .

br(-position(X,do(A,S))):-  br(position(X,S)),
```

```
                              (A=avancer;A=reculer);
                              (
                                 br(-position(X,S)),
                                 \+((
                                       br(position(Y,S)),
                                       (A=avancer,X is Y+1);
                                       (A=reculer,X is Y-1)
                                   ))
                              ).

br(obstacle(X,do(A,S))):- A=informer_obs(X);
                           br(obstacle(X,S)).

br(-obstacle(X,do(A,S))):- \+A=informer_obs(X),
                            br(-obstacle(X,S)).

bp(position(X,do(A,S))):- A=observer_pos, position(X,S) ;
                         \+A=observer_pos, bp(position(X,S)).
bp(-position(X,do(A,S))):-A=observer_pos, position(Y,S), \+X=Y;
                         \+A=observer_pos, bp(-position(X,S)).

bp(obstacle(X,do(A,S))):-
                              A=allumer_obsever,
                              obstacle(X,S) ,
                              position(Y,S),
                              Y=<X , X=<Y+3      % d=3
                             ;
                              bp(obstacle(X,S)),
                              \+(( A=allumer_obsever,
                                   \+ obstacle(X,S) ,
                                   position(Y,S),
                                   Y=<X , X=<Y+3        % d=3
                                ) ).
bp(-obstacle(X,do(A,S))):-
                               A=allumer_obsever,
                               \+ obstacle(X,S),
                               position(Y,S),
                               Y=<X , X=<Y+3
                              ;
                               bp(-obstacle(X,S)),
                                  \+(( A=allumer_obsever,
                                       obstacle(X,S),
                                       position(Y,S),
                                       Y=<X , X=<Y+3
                                    )).

/* Restore suppressed situation arguments.   */
```

```
restoreSitArg(position(N),S,position(N,S)).
restoreSitArg(obstacle(N),S,obstacle(N,S)).

restoreSitArg(br(position(N)),S,br(position(N,S))).
restoreSitArg(br(-position(N)),S,br(-position(N,S))).
restoreSitArg(br(obstacle(N)),S,br(obstacle(N,S))).
restoreSitArg(br(-obstacle(N)),S,br(-obstacle(N,S))).

restoreSitArg(bp(position(N)),S,bp(position(N,S))).
restoreSitArg(bp(-position(N)),S,bp(-position(N,S))).
restoreSitArg(bp(obstacle(N)),S,bp(obstacle(N,S))).
restoreSitArg(bp(-obstacle(N)),S,bp(-obstacle(N,S))).

/*   Initial Situation  */

position(1,s0).

obstacle(6,s0).

br(position(1,s0)).

br(-position(X,s0)) :-  \+ br(position(X,s0)).
```

Maintenant on pose la question 'est-ce que le robot croit qu'il y a des obstacles à l'état initial?' :

```
| ?- br(obstacle(X,do(avancer,s0))).
```

La réponse est `no`, car la seule croyance initiale est celle du robot concernant sa position, donc le robot ignore tout ce qui concerne les obstacles.

Maintenant on pose la question 'est-ce que le pilote croit qu'il y a des obstacles après avoir fait une observation à l'état initial?':

```
| ?- bp(obstacle(X,do(allumer_obsever,s0))).
```

La réponse est `no` car la limite de visibilité est dépassée. Une réponse négative va aussi se produire si l'on pose la question 'est-ce que le pilote croit qu'il y a des obstacles après avoir fait une observation après que le robot ait avancé d'une unité depuis l'état initial?'. En revanche, si l'on pose la question 'est-ce que le pilote croit qu'il y a des obstacles après avoir fait une observation après que le robot ait avancé de deux unités depuis l'état initial?' :

```
| ?- bp(obstacle(X,do(allumer_obsever,do(avancer,do(avancer,s0))))).
```

La réponse est `X = 6`, qui veut dire intuitivement que le pilote croit qu'il y a un obstacle dans la position 6 après la séquence d'actions mentionnée.

Par contre, si l'on pose une question similaire concernant les croyances du robot 'est-ce que le robot croit qu'il y a des obstacles après une observation après qu'il ait avancé de deux unités depuis l'état initial?' :

```
br(obstacle(X,do(allumer_obsever,do(avancer,do(avancer,s0))))).
```

La réponse est `no` car le robot ne perçoit pas d'obstacle. C'est le pilote qui l'informe de la présence d'un obstacle. Ainsi la question 'est-ce que le robot croit qu'il y a des obstacles après que le pilote l'ait informé de la présence d'un obstacle dans la position 6, après une observation, après qu'il ait avancé de deux unités depuis l'état initial?' :

```
| ?- br(obstacle(X,do(informer_obs(6),do(allumer_obsever,
do(avancer,do(avancer,s0)))))).
```

est résolue avec la réponse `X = 6`.

Nous avons montré donc comment on peut enrichir le pouvoir d'expression du langage GOLOG avec des notions épistémiques. L'interpréteur GOLOG présenté dans la section précédante peut être utilisé pour poser des questions incluant des actions épistémiques complexes.

5.2 Théories incomplètes

Les implémentations Prolog n'offrent pas de mécanisme pour représenter des informations incomplètes qui expriment un état d'ignorance concernant les faits vrais du monde ou les croyances vraies des agents. Pour cette raison leurs axiomatisations correspondantes sont souvent formalisées en considérant l'hypothèse du monde fermé.

Il y a des nombreuses applications pour lesquelles une représentation sous cette forme est suffisante. Cependant le nombre d'applications qui requièrent une axiomatisation du monde ouvert est relativement grand. Parmi celles-ci on peut mentionner la robotique. Il n'est pas réaliste de penser qu'un robot possède toute l'information concernant l'état initial du monde. Si c'était le cas alors pourquoi doter les robots de systèmes de capteurs d'informations?

Reprenons l'exemple de la section précédante. Considérons maintenant que la seule information connue concernant l'état mental du robot dans la situation initiale est qu'il croit qu'il n'y a pas d'obstacle dans la position 6.

Ce scénario peut être implémenté en Prolog. En effet, on peut considérer comme faux tout ce que nous ignorons concernant les croyances du robot et représenter cette information par l'axiome suivant : $\forall x\,(B_r(obstacle(x, S_0)) \leftrightarrow \neg x = 6)$, et ainsi écrire la clause correspondante dans un programme Prolog.

Considérons la question 'est-ce que le pilote croit qu'il y a un obstacle dans la position 8?'. La réponse donnée par cette implémentation est 'non' qui signifie intuitivement 'le robot ne croit pas qu'il y a un obstacle dans la position 8'.

Supposons qu'à partir de cette réponse une séquence d'actions soit déterminée. Par exemple, que le robot continue son déplacement. Le robot pourrait arriver à cogner un obstacle que, selon "l'implémentation" par l'hypothèse du monde fermé, le robot croyait inexistant. Dans ces conditions, il est préférable de donner une réponse "sincère de la part de l'implémentation" correspondant à la réalité telle qu'on l'ignore ('on ne sait pas si le robot croit qu'il l'existe un obstacle dans la position 8') pour pouvoir ainsi déterminer la séquence d'actions d'une manière plus prudente. Par exemple, avant de continuer le déplacement

il faudrait que le pilote (ou même plus sophistiqué, un capteur du robot) vérifie s'il y a un obstacle dans la position 8 et qu'il informe le robot de la présence de cet obstacle.

Les réponses sur l'ignorance ne sont pas envisageables sur une implémentation Prolog car elle ne laisse pas la possibilité de l'incertitude. Tout est résolu par l'un des deux cas : oui (qui veut dire que la propriété peut être prouvée) ou non (qui veut dire que la négation de la propriété peut être prouvée). Donc un démonstrateur plus général est requis. Intuitivement le démonstrateur doit fournir les réponses : oui (si la propriété peut être prouvée) ou non (si la propriété ne peut pas être prouvée).

Dans la Section 4.6 nous avons proposé la SOLE-résolution qui est un démonstrateur de ce type. Nous avons implémenté ce démonstrateur en Prolog (cf. Annexe D). On peut utiliser la SOLE-résolution pour implémenter une méthode qui réponde aux questions plus générales telles que 'est-ce que l'on a des informations sur la propriété ψ (dans le sens de savoir si elle est vraie ou fausse)'. La méthode peut consister à poser la question représentant ψ, si la réponse est oui, arrêter et répondre par oui. Si la réponse est non, alors poser la question représentant $\neg\psi$, et si la réponse est oui, arrêter et répondre par oui. Si la réponse est non, arrêter et répondre par non. Cette méthode peut être utilisée aussi afin de connaître les faits ignorés par les agents.

Comme nous l'avons déjà mentionné les caractéristiques de la SOLE-résolution (sa stratégie linéaire efficace et son pouvoir de générer des conséquences de façon restreinte) permettent de justifier le choix de ce démonstrateur comme un candidat adéquat pour l'implémentation des théories incomplètes.

Dans le cas particulier du calcul des situations, l'efficacité et la possibilité de traiter des clauses séparément ne sont pas les seuls problèmes rencontrés. En effet, nous avons vu que dans la démonstration d'une formule régressable, les seuls axiomes requis sont ceux des préconditions, ceux de changement d'état et ceux de l'unicité des noms. Or les axiomes de changement d'état ont une forme récursive, donc on doit assurer la terminaison de la procédure. En plus, dans le cas des actions complexes, l'utilisation de GOLOG est requise et une façon d'étendre ce langage doit être étudiée.

Les trois sections suivantes présentent les idées que nous avons développées pour une implémentation des théories incomplètes. Chaque section essaie de répondre aux questions suivantes : Comment évaluer les contraintes arithmétiques ? Comment contrôler la gestion de la récursivité des axiomes de changement d'état ? Comment étendre GOLOG pour traiter des informations incomplètes ?

5.2.1 Evaluation des littéraux arithmétiques

De façon similaire que pour le prédicat désignant la négation par échec, en Prolog, les opérateurs désignant les opérations arithmétiques telles que l'addition, la soustraction, la multiplication, etc., doivent prendre des arguments qui tous sauf un doivent être clos. A la différence de la négation par échec, en Prolog, les expressions arithmétiques peuvent avoir une variable, ainsi la variable est instanciée avec le résultat de l'opération indiquée. Donc on retrouve le même problème mentionné pour la négation par échec. Les opérateurs ne sont

pas "tolérants". Parfois pour démontrer une formule, il est nécessaire de suspendre l'évaluation de ces opérateurs et d'attendre l'instantiation des variables (comme le fait le prédicat \+) pour pouvoir effectuer son évaluation.

La solution proposée ici est de regrouper les contraintes arithmétiques à satisfaire et de différer leur évaluation à la fin de la déduction. Comme nous avons vu dans la Section 4.6.2, la SOLE-résolution permet de déduire des contraintes arithmétiques.

Par exemple, l'implémentation donnée dans l'Annexe D produit la déduction suivante :

```
***************sol au niveau-87
[[],[non(o(_7263)),_7263=7,_7263=3]]
[[],[non(_8075=b),non(o(b)),frame(non(o(_8075))),_8075=7,_8075=3]]
[[],[non(o(b)),frame(non(o(g))),g=7,g=3]]
[[],[g=7,g=3]]
[[],[non(_10115=a),non(_10122=1),non(plus(a,1,g)),plus(_10115,_10122,7),
frame(g=7),g=3]]
[[],[non(_10245=1),non(plus(a,1,g)),plus(a,_10245,7),frame(g=7),g=3]]
[[],[non(plus(a,1,g)),plus(a,1,7),frame(g=7),g=3]]
[[],[plus(a,1,7),frame(g=7),g=3]]
[[plus(a,1,7)],[g=3]]
[[plus(a,1,7)],[non(_12255=a),non(_12262=1),non(plus(a,1,g)),
plus(_12255,_12262,3),frame(g=3)]]
[[plus(a,1,7)],[non(_12378=1),non(plus(a,1,g)),plus(a,_12378,3),frame(g=3)]]
[[plus(a,1,7)],[non(plus(a,1,g)),plus(a,1,3),frame(g=3)]]
[[plus(a,1,7)],[plus(a,1,3),frame(g=3)]]
[[plus(a,1,3),plus(a,1,7)],[]]

X = g
```

qui correspond à la déduction du second exemple de la Section 4.6.2. La théorie initiale donnée est :

```
/*****************************************/
/*          Initial Situation            */
/*****************************************/

hyp([r(a)]).
hyp([o(b)]).
hyp([plus(a,1,g)]).
hyp([g=b]).


champs([plus(_,_,_)]).
```

L'étape suivante serait l'implémentation d'une procédure d'évaluation des contraintes arithmétiques déduites. La procédure d'évaluation peut avoir plusieurs niveau de difficulté selon l'acceptation des contraintes à résoudre. Nous avons réfléchi à une procédure possible qui fonctionnerait de la même manière que Prolog, i.e. accepter d'évaluer seulement des contraintes qui ont au plus une variable (ou contante de Skolem).

5.2.2 Gestion de la récursivité des axiomes de changement d'état

Les formules caractérisant les axiomes de changement d'état ont une forme récursive. Le démonstrateur risque donc de tomber dans une boucle de recherche infinie. La solution proposée est de trouver une heuristique qui assure la régression d'une formule. L'heuristique que nous utilisons, peut-être un peu primitive, est d'imposer que le nombre d'imbrications de la fonction *do* dans une formule régressable ne grandira pas. De cette façon on assure la terminaison de la procédure.

La démonstration de la complétude de cette procédure est basée sur l'opérateur de régression. La recherche de la démonstration des formules est restreinte. A chaque résolution les formules doivent diminuer la profondeur de leurs termes de type situation. Par induction on peut démontrer que parmi les résolutions trouvées à chaque étape, il existe une résolution qui correspond à l'application de l'opérateur de régression, et par la complétude de cet opérateur on assure la complétude de la méthode.

5.2.3 Le test basé sur SOLE-résolution

Comme nous l'avons vu dans la section précédente une implémentation de GOLOG permet de représenter des actions complexes, et de définir des procédures. Donc augmenter le pouvoir de déduction de GOLOG au cas des théories incomplètes offre la possibilité d'implémentation d'un grand nombre d'applications.

Pour attendre ce but, on propose la modification de l'évaluation de l'action complexe 'test d'un fluent' `?(p)`. Cette action utilise explicitement les axiomes de la situation initiale ainsi que ceux de changement d'état. Elle utilise implicitement l'hypothèse du monde fermé au moment de l'évaluation de la négation par échec.

La solution proposée est d'utiliser un appel à la SOLE-déduction pour l'évaluation d'un fluent. Ainsi dans le code de GOLOG `do(?(P),S,S):- holds(P,S)` peut être remplacé par `do(?(P),S,S):- sole-holds(P,S)`.

Quelques détails techniques doivent être considérés tels que la transformation de `P` en un fluent dépendant de la situation `S` ainsi que la transformation de `P` dans un format accepté par la SOLE-résolution.

5.2.4 Un interprète pour des théories épistémiques incomplètes

En résumé, l'implémentation de la déduction des théorèmes dans le calcul des situations pour les théories épistémiques incomplètes est envisageable. Nous proposons des solutions aux principaux problèmes posés par l'incomplétude de la théorie initiale.

La terminaison de la procédure est assurée pour les formules démontrables grâce à la contrainte de régression imposée aux termes de type situation.

Par contre, comme les autres démonstrateurs, la terminaison n'est pas assurée pour les formules qui ne sont pas démontrables.

Les modalités des croyances peuvent aussi s'implémenter avec un métaprédicat. La justification est similaire à celle utilisée dans la section précédante. Etant donné que le langage n'accepte pas d'imbrications de modalités, celles-ci peuvent se réécrire avec des nouveaux symboles des prédicats et être traitées comme des prédicats.

L'hypothèse du monde fermé n'est plus requise, par contre les axiomes d'unicité des noms sont toujours nécessaires.

Conclusions et perspectives

Conclusion de la formalisation en logique

Modéliser des systèmes dynamiques pose des problèmes tels que le frame problem et celui de la qualification, plusieurs solutions ont été proposées. On a présenté une solution basée sur le calcul des situations. Cette solution a l'avantage de posséder une implémentation simple en Prolog pour des théories initiales complètes.

Si le système est capable de tester son environnement, i.e. si les agents composant le système peuvent réaliser des actions telles que acquérir des informations sur le monde dans lequel ils se trouvent (tel est le cas d'un robot équipé de capteurs), alors de nouveaux problèmes se posent, par exemple ceux de la révision et la mise-à-jour des croyances. Dans ces conditions la nécessité d'étendre le formalisme, afin de considérer les actions d'acquisition ainsi que leurs effets sur les états mentaux des agents, est justifiée.

Dans [SL93], les auteurs ont proposé une solution au frame problem au niveau des croyances. Leur approche définit une extension du calcul des situations.

Cette solution est intéressante d'un point de vue du pouvoir d'expression parce qu'elle permet de définir l'évolution des croyances qui portent sur des formules quelconques, y compris des formules contenant des modalités. Cependant elle suppose implicitement qu'un agent connaisse toutes les actions qui sont réalisées et qu'il connaisse les conséquences de l'exécution de ces actions. Ces hypothèses sont acceptables si l'on considère un seul agent, mais dans le cas de systèmes composés de plusieurs agents, en général les agents ne connaissent pas les actions réalisées par les autres agents. Cette solution n'est donc pas acceptable dans le contexte multiagents.

Dans [LL98], il est aussi proposé une extension du calcul des situations pour tenir compte des croyances. C'est un travail théorique très précis qui présente la sémantique et une axiomatique complète par rapport à la sémantique. Mais la solution proposée au frame problem est basée sur les mêmes hypothèses que dans [SL93].

Afin d'affaiblir ces hypothèses sur les croyances des agents, i.e. afin de permettre aux agents de croire des choses différentes, d'ignorer si l'exécution d'une action a eu lieu, et d'ignorer (ou se tromper sur) les effets produits par les actions, on a introduit deux formalismes basés sur le calcul des situations qui résolvent les problèmes de la révision et de la mise-à-jour des croyances dans un contexte multiagents.

Dans le formalisme définissant les axiomes de changement d'état des croyances, l'introduction d'une action ou d'une propriété nouvelles ne modifie pas les axiomes déjà construits. S'il s'agit d'une action, les axiomes de changement sont éventuellement étendus, s'il s'agit d'une propriété, de nouveaux axiomes de changement sont introduits (un axiome définissant le changement du fluent, et deux axiomes pour chaque agent définissant le changement des croyances de l'agent par rapport à ce fluent). Par contre dans l'approche utilisant une relation d'accessibilité, cette introduction oblige à reformuler les axiomes de changement d'état de la relation ainsi que la fonction de plausibilité pour inclure les dépendances créées par la modification.

L'extension proposée en termes d'opérateurs modaux définit explicitement l'évolution des croyances à partir des croyances précédentes, tandis que les autres extensions définissent implicitement l'évolution des croyances à partir de l'évolution de l'ensemble des situations accessibles, qui est défini par la relation accessibilité.

Si à l'état initial (ou après l'exécution des actions d'observation) les croyances de l'agent correspondent à la réalité, ces croyances doivent maintenir cette correspondance avec la réalite car on suppose que les effets des actions sont connus par l'agent. Ce qui implicitement veut dire que toute information, que l'agent croit après une action d'observation, est vraie dans le monde réel. Mais c'est une hypothèse trop forte de supposer que l'agent n'a que des croyances vraies.

Il faut signaler qu'actuellement à notre connaissance les solutions proposées dans [SL93] et [LL98] n'ont pas été implémentées. Tandis que la seconde approche que nous proposons a été implémentée. Une première version est déjà réalisée même si elle se limite à accepter seulement des théories complètes. Une autre version plus générale est en phase de réalisation.

Les axiomes de changement d'état des croyances que nous avons proposés pour résoudre le frame problem, ont une limitation similaire à celle des axiomes de changement d'état dans [Rei91]. Ils calculent l'évolution seulement pour des formules atomiques ou leurs négations. Malgré cette contrainte, l'utilisation de ce formalisme semble approprié dans le cas où les applications ne considèrent pas des agents "philosophes" (qui se posent des questions par rapport aux croyances d'autres agents et d'eux-mêmes).

Il y a des résultats intéressants issus de l'introduction des croyances des agents dans la modélisation de systèmes dynamiques. Par exemple, Levesque s'en sert pour la spécification des plans quand l'agent qui planifie peut acquérir des informations sur son environnement [Lev96].

Conclusion de la démonstration automatique

Nous avons présenté une extension de la SOL-résolution appelée SOLE-résolution. Cette extension est complète pour les théories avec égalité. Nous avons démontré qu'elle est valide et complète par rapport à résolution et paramodulation. Il faut remarquer que la SOLE-résolution a été implémentée et que très peu de stratégies orientées par le but sur l'égalité l'ont été (cf. [SL91]).

Dans [Bra75] Brand a présenté une méthode qui est basée sur la même idée

intuitive de la règle d'inférence E-lit. Une transformation de l'ensemble des clauses initiales Σ est faite afin d'obtenir un ensemble de clauses Σ' tel que, par exemple, la clause $\neg(t = x) \vee \neg(f(x) = y) \vee C(y)$ est dans Σ' ssi la clause $C(f(t))$ est dans Σ, où $f(t)$ est le seul terme qui apparaît dans C. Donc, l'ensemble de clauses généré par E-lit dans une SOLE-déduction est un sous-ensemble de Σ'. D'ailleurs, l'ensemble de clauses généré par E-lit est plus petit que Σ' car les clauses générées par E-lit sont en rapport avec la clause initiale tandis que Σ' est généré indépendamment de la clause initiale. Ce sont ces raisons qui nous permettent d'affirmer que la SOLE-résolution est plus efficace.

Une autre différence significative entre la méthode de Brand et celle de la SOLE-résolution, est que la complétude de cette dernière a été prouvée pour la génération des conséquences alors que Brand considère seulement la génération de la clause vide.

La SOLE-résolution conserve l'avantage proposé par Inoue, celui de restreindre la génération des conséquences aux clauses satisfaisant un champ de production. La complétude, pour la génération de conséquences restreintes par le champ de production, est donc préservée.

Donc, même si E-lit est basée sur la même idée intuitive que la méthode de Brand, la SOLE-résolution est plus efficace et offre d'autres fonctionnalités.

Nous avons présenté une méthode pour implémenter les théories de l'action basée sur le langage de programmation logique Prolog. La relation qui existe entre une théorie logique Σ (constituée d'axiomes) et un programme Prolog est justifiée par Clark. On utilise le résultat de Clark qui prouve la complétude de Prolog, énoncée de la manière suivante. Si un programme Prolog réussit à prouver une formule fermée ψ, alors ψ est une conséquence logique de Σ, et si le programme Prolog échoue dans la preuve de ψ, alors $\neg\psi$ est une conséquence logique de Σ. Ce sur quoi Reiter s'appuie pour démontrer la validité d'une implémentation valide pour des théories complètes de l'action. Nous étendons ce résultat aux théories complètes de l'action et des croyances. Nous donnons aussi les outils pour la construction d'une méthode qui considère les théories incomplètes.

Perspectives

Modélisation du passé et du futur

L'une des approches que nous avons présenté modélise les croyances des agents avec des opérateurs modaux. Cette approche considère les croyances seulement dans la situation présente. En effet, la signification de $B_i p(s)$ est que dans la situation s l'agent i croit que p est *vraie* dans la situation s. Remarquons que 'dans la situation s' apparaît deux fois dans cette signification, la première dénote la situation dans laquelle est évaluée la croyance de l'agent et la deuxième dénote la situation dans laquelle est évalué le fluent p. Une autre notation pourrait être $B_i(p(s), s)$ qui permet de considérer indépendamment la situation de la croyance et la situation du fluent de la croyance. Ainsi avec cette notation on peut envisager l'introduction de croyances concernant le passé ($B_i(p(s_1), s_2)$ telle que $s_1 \prec s_2$) ou le futur ($B_i(p(s_1), s_2)$ telle que $s_2 \prec s_1$).

Des applications sont nombreuses notamment dans la génération de plans. Si l'agent possède une image d'un hypothétique monde futur, la prise de décision peut être dirigée par cette représentation et non determinée uniquement par l'analyse de la situation présente.

Une extension du formalisme dans ce sens doit résoudre des problèmes très délicats du point de vue logique, tels que la nouvelle représentation des axiomes de changement d'état pour le passé et pour le futur. Par exemple, est-ce qu'un axiome de la forme ci-dessous convient pour modéliser les changements des croyances du passé?

$$s_1 \prec s_2 \rightarrow (B_i(do(a, s_1), s_2) \leftrightarrow \Upsilon^{+}_{B_i F}(a, s_1, s_2) \vee B_i(F(s_1), s_2) \wedge \neg \Upsilon^{-}_{B_i F}(a, s_1, s_2)).$$

Calcul des situations déontique

Nous avons montré une façon d'étendre le calcul des situations aux croyances. La méthode utilisée est basée sur l'idée intuitive d'introduire la modalité de la croyance qui satisfait les schémas d'axiome de la logique modale KD, et de définir l'évolution des croyances à l'aide des axiomes de changement d'état. Or si nous considérons des notions déontiques, on peut remarquer que la notion d'obligation est modélisée aussi par la logique KD (logique déontique standard), et donc une extension du calcul des situations au niveau déontique est envisageable.

Une transcription de la solution proposée pour les croyances au niveau déontique ne peut pas se faire de façon automatique. Pour considérer une telle approche, il faudrait analyser des problèmes posés par les modalités déontiques tels que la "vérification de la véracité" des propositions modales. Par exemple, dans le cas des croyances, les agents possédant des capteurs peuvent réviser leurs croyances en s'appuyant sur l'observation du monde physique. Dans le cas déontique ce type de révision n'est pas envisageable.

E-lit restreinte

D'après la Définition 20, dans une SOLE-résolution la règle d'inférence E-lit peut être appliquée aux variables apparaissant dans un littéral. Par exemple, $\neg(y = x) \vee L[y] \vee C$ peut être générée à partir de $L[x] \vee C$. Brand interdit l'application de la transformation à cette sorte de clauses et prouve qu'une telle restriction préserve la complétude de son formalisme. Nous envisageons la démonstration d'un résultat similaire, i.e. démontrer que l'interdiction de l'application de E-lit aux variables, préserve la complétude de la SOLE-résolution. Au niveau de l'implémentation nous envisageons l'introduction d'une stratégie qui effectue cette restriction et qui serait plus efficace.

Annexe A

Notations

- ssi, si et seulement si.
- $E[t]$, l'expression E (formule, littéral ou fonction) contient le terme ou le sous-bterme t. On peut faire l'analogie avec la notation $f(x)$ désignant une fonction dépendant de x, sauf que dans $E[t]$, t fait référence à *une* seule occurrence de t, alors que dans $f(x)$, x fait référence à *toutes* les occurrences de x. Pour être plus précis, on doit avoir une notation indiquant le numéro d'occurrence référencée $E[t]_n$, où n dénote le numéro d'occurrence. Ainsi $P[a]_3$ dénote l'occurrence de a dans $P(f(a), a, g(a))$, celle qui apparaît comme argument de la fonction g. Pour ne pas alourdir la notation nous noterons $E[t]$ par $E[t]_n$ quand il n'y aura pas d'ambiguïté (plus de deux occurrences de t). Si deux expressiones de la forme $E[t]$ et $E[s]$ apparaissent dans le même contexte, on suppose que la deuxième dénote le résultat du remplacement de t par s dans E.
- $B_i p(s)$, est une abréviation de $B_i(p(s), s)$ signifiant intuitivement que dans la situation s l'agent i croit que p est $vraie$ dans la situation s. Puisque il s'agit de la même situation celle de l'évaluation de la croyance de l'agent et celle de l'évaluation du fluent, on a décidé de simplifier la notation. Cette croyance se rapporte au présent. Quand la croyance se rapporte au passé ou au futur on est obligé d'utiliser la notation $B_i(p(s), s)$.

Annexe B

Rappels de logique

Il y a trois aspects à considérer dans une logique. Les *formules bien formées* de la logique, ce sont les expressions qui peuvent être construites et qui sont définies comme un langage formel. La *théorie de la preuve* qui contient les axiomes et les règles d'inférence avec lesquels on peut déduire un ensemble de formules bien formées appelées théorèmes. La *théorie des modèles*, qui donne une signification formelle aux formules bien formées. Le premier aspect est aussi appelé syntaxe[1], la théorie de la preuve est aussi appelée théorie de la démonstration, axiomatique, système d'inférence ou système de déduction, et la théorie des modèles est connue comme la sémantique de la logique. L'objectif de la sémantique est de mettre en relation des formules avec une représentation simplifiée de la réalité à laquelle on s'intéresse.

Il faut relever que la formalisation est, dans l'état actuel de la logique, la garantie de rigueur maximum et que, d'autre part, elle offre souvent la possibilité de plusieurs interprétations différentes, ce qui constitue une grande économie de pensée et donne une unité au savoir.

B.1 Logique des propositions

La logique des propositions est la plus simple, elle est fréquemment utilisée pour la représentation des informations basées sur des faits (dans les SMA, elle est bien adaptée pour décrire l'environnement). Dans son langage, les formules sont construites à partir de propositions atomiques (phrases déclaratives qui sont, soit vraies, soit fausses, et qui représentent intuitivement des faits atomiques sur le monde) et des connecteurs $\neg$, $\wedge$, $\vee$, $\rightarrow$ et $\leftrightarrow$ dénotant respectivement 'non', 'et', 'ou', 'si ... alors' et 'si et seulement si'.

Par exemple, les faits 'il pleut' et 'il y a des nuages' peuvent être représentés respectivement par les propositions atomiques *pleut* et *nuages*. L'implication 's'il pleut alors il y a des nuages' peut être représentée par la formule propositionnelle $pleut \rightarrow nuages$.

Etant donné l'ensemble de propositions atomiques Φ, le langage de la logique des propositions $\mathcal{L}_{\mathcal{P}}$ est défini récursivement par les règles suivantes :

[1] Il y a des auteurs qui considèrent la syntaxe comme l'union du premier et du deuxième aspect.

SYN-1. $p \in \Phi$ implique $p \in \mathcal{L}_\mathcal{P}$

SYN-2. ψ, $\phi \in \mathcal{L}_\mathcal{P}$ implique $\neg\psi$, $\psi \vee \phi \in \mathcal{L}_\mathcal{P}$

En ce qui concerne la sémantique, si $M_P \stackrel{\text{def}}{=} < L >$ est une interprétation de $\mathcal{L}_\mathcal{P}$, où L définit l'ensemble des propositions atomiques qui sont vraies dans $\mathcal{L}_\mathcal{P}$, alors savoir si une formule ψ est vraie pour cette interprétation, noté $M_P \models \psi$ consiste à vérifier la véracité de ses différents composants à partir des règles suivantes :

SEM-1. $M_P \models p$ ssi $p \in L$

SEM-2. $M_P \models \neg\psi$ ssi $M_P \not\models \psi$

SEM-3. $M_P \models \psi \vee \phi$ ssi $M_P \models \psi$ ou $M_P \models \phi$

La définition des autres opérateurs est la suivante : $\psi \wedge \phi \stackrel{\text{def}}{=} \neg(\neg\psi \vee \neg\phi)$, $\psi \rightarrow \phi \stackrel{\text{def}}{=} \neg\psi \vee \phi$ et $\psi \leftrightarrow \phi \stackrel{\text{def}}{=} (\psi \rightarrow \phi) \wedge (\phi \rightarrow \psi)$.

Parmi d'autres théories axiomatiques existent celle de Gentzen. Ici nous définisons une axiomatique de Hilbert. Soient A, B et C des formules bien formées, les schémas d'axiome sont les suivants :

AXI-1. $\vdash A \rightarrow (B \rightarrow A)$

AXI-2. $\vdash (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

AXI-3. $\vdash (\neg A \rightarrow B) \rightarrow ((\neg A \rightarrow \neg B) \rightarrow A)$

La règle d'inférence est la suivante :

AXI-RI-1. Si $\vdash A \rightarrow B$ et $\vdash A$ alors $\vdash B$ *Modus Ponens*

B.2 Logique des prédicats du premier ordre

La logique des prédicats du premier ordre inclut la logique des propositions. Le vocabulaire s'étend avec des symboles de constantes (a, b, c, ...), des symboles de variables (x, y, z, ...) et des symboles de fonctions (f, g, h, ...)[2], et avec les symboles $\forall$ (quantificateur universel) et $\exists$ (quantificateur existentiel). Dans ce langage les formules sont construites à partir de prédicats atomiques, des mêmes connecteurs que ceux de la logique des propositions et des quantificateurs liés aux symboles de variables. Chaque prédicat atomique $p(t_1, t_2, \ldots, t_n)$ a n arguments (quand $n = 0$ la formule est une proposition) qui sont des termes, i.e. soit des constantes, soit des variables, soit des fonctions. Les fonctions ont éventuellement des termes comme arguments.

Par exemple, les affirmations 'x est un homme' et 'x est mortel' peuvent être représentées par les prédicats $homme(x)$ et $mortel(x)$ respectivement. L'implication 'si x est un homme alors x est mortel' peut être représentée par la formule $\forall x\,(homme(x) \rightarrow mortel(x))$.

[2]Sans oublier que ce sont des symboles, nous les appellerons parfois simplement constantes, variables et fonctions, respectivement.

Etant donné l'ensemble de prédicats atomiques Φ, le langage de la logique des prédicats du premier ordre $\mathcal{L}_{\mathcal{P}'}$ est donné par :

SYN-3. Les règles de $\mathcal{L}_{\mathcal{P}}$ appliquées à $\mathcal{L}_{\mathcal{P}'}$

SYN-4. Soit x une variable et $\psi \in \mathcal{L}_{\mathcal{P}'}$ alors $\forall x\ \psi$, $\exists x\ \psi \in \mathcal{L}_{\mathcal{P}'}$

Dans la logique propositionnelle, une interprétation est donnée par l'ensemble des propositions atomiques qui sont vraies. Dans la logique des prédicats du premier ordre, la notion d'interprétation est étendue pour définir les interprétations des nouveaux symboles. En particulier, une fonction v est définie afin de donner une signification aux symboles de variable.

Soit $M_{P'} \stackrel{\text{def}}{=} < \mathbb{D}, p_1^{M_{P'}}, p_2^{M_{P'}}, \ldots, p_n^{M_{P'}}, f_1^{M_{P'}}, f_2^{M_{P'}}, \ldots, f_m^{M_{P'}} >$ une structure de $\mathcal{L}_{\mathcal{P}'}$, où $\mathbb{D}$ est un ensemble non vide appelé univers ou domaine de $M_{P'}$. Chaque $p_i^{M_{P'}} \subseteq \mathbb{D}^{n_i}$ définit l'ensemble de n_i-uplets qui sont associés au symbole de prédicat p_i. $p_i^{M_{P'}}$ est appelé l'extension de p_i, cette extension du prédicat, qui définit une relation, ne doit pas être confondue avec le symbole du prédicat, ni avec une instance de prédicat qui ne sont que des symboles. Chaque $f_i^{M_{P'}} : \mathbb{D}^{n_i} \mapsto \mathbb{D}$ est une fonction n_i-aire sur $\mathbb{D}$. Les constantes sont des cas particuliers des fonctions où $n_i = 0$.

Soit $v : V \mapsto \mathbb{D}$ une fonction qui assigne à chaque variable du langage un élément du domaine $\mathbb{D}$.

Soit $\bar{v} : T \mapsto \mathbb{D}$ une extension de la fonction v qui assigne à chaque terme du langage un élément du domaine $\mathbb{D}$. Si x est variable $\bar{v}(x) = v(x)$ et si $t_1, t_2, \ldots, t_n$ sont des termes et f un symbole de fonction n-aire alors $\bar{v}(f(t_1, t_2, \ldots, t_n)) = f^{M_{P'}}(\bar{v}(t_1), \bar{v}(t_2), \ldots, \bar{v}(t_n))$.

Une interprétation est constituée d'une structure $M_{P'}$ et d'une fonction d'assignation de variables v. La satisfaction d'une formule par l'interprétation $(M_{P'}, v)$ est définie récursivement comme suit :

SEM-4. $M_{P'}, v \models p(t_1, t_2, \ldots, t_n)$ ssi $< \bar{v}(t_1), \bar{v}(t_2), \ldots, \bar{v}(t_n) > \in p^{M_{P'}}$

SEM-5. Les règles SEM-2 et SEM-3 avec $(M_{P'}, v)$ comme interprétation

SEM-6. $M_{P'}, v \models \forall x\ \psi$ ssi pour toute $d \in \mathbb{D}$ $M_{P'}, v_{(x|d)} \models \psi$, où $v_{(x|d)}$ est une fonction exactement égale à v sauf que la variable x prend la valeur d

La définition du quantificateur ($\exists$) est la suivante : $\exists x\ \psi \stackrel{\text{def}}{=} \neg\forall x\ (\neg\psi)$.

Parmi d'autres axiomatiques existent celle de Gentzen. Nous étendons ici l'axiomatique de la logique des propositions afin d'avoir une axiomatique associée à la logique des prédicats du premier ordre.

Les schémas d'axiome sont les suivants, soient A, B et C des formules :

AXI-4. Les schémas d'axiome de $\mathcal{L}_{\mathcal{P}}$

AXI-5. $(\forall x\ (A \to B)) \to (A \to \forall x\ B)$
où A ne contient aucune occurrence libre de x

AXI-6. $(\forall x\ A) \to A\{t/x\}$
où x est libre pour t dans A (cf. Annexe C.3)

Les règles d'inférence sont :

AXI-RI-1. La règle de $\mathcal{L}_{\mathcal{P}}$ *Modus Ponens*

AXI-RI-2. Si $\vdash A$ alors $\vdash \forall x\ A$ *Généralisation*

B.3 Logique des prédicats du premier ordre avec égalité

La logique des prédicats du premier ordre avec égalité est une logique des prédicats du premier ordre qui inclut dans son vocabulaire le symbole de prédicat $=$. Ce symbole reçoit une sémantique spécifique[3] : $t_1 = t_2$ signifie 'le terme t_1 est égal au terme t_2'.

Par exemple, l'affirmation 'la personne x a l'âge y' peut être représentée par le prédicat $\hat{a}ge(x, y)$. L'affirmation 'il n'existe qu'un seul homme qui a 125 ans' peut être représentée par la formule $\exists x\,(\hat{a}ge(x, 125)) \wedge \forall y \forall z\,(\hat{a}ge(y, 125) \wedge \hat{a}ge(z, 125) \rightarrow y = z)$.

Le langage de cette logique $\mathcal{L}_{\mathcal{P}^{\mathcal{E}}}$ est le même que celui de la logique des prédicats du premier ordre $\mathcal{L}_{\mathcal{P}'}$.

En ce qui concerne la sémantique, une interprétation (M_{PE}, v) est constituée d'une structure M_{PE} et d'une fonction d'assignation de variables v. Cette interprétation est semblable à celle de la logique des prédicats du premier ordre sauf que l'extension du prédicat $=$, $=^{M_{PE}} \subseteq \mathbb{D}^2$ contient (seulement) les couples de la forme (d, d) pour tout d élément du domaine. Autrement dit, l'extension du prédicat $=$ définit la relation d'identité sur le domaine. Ainsi une formule de la forme $t_1 = t_2$ est *vraie* quand l'évaluation de t_1, $\bar{v}(t_1)$, donne le même élément du domaine que l'évaluation de t_2, $\bar{v}(t_2)$.

SEM-7. Les règles de $\mathcal{L}_{\mathcal{P}'}$ avec (M_{PE}, v) comme interprétation

SEM-8. $M_{PE}, v \models t_1 = t_2$ ssi $\bar{v}(t_1)$ est identique à $\bar{v}(t_2)$.

Nous étendons ici l'axiomatique de la logique des prédicats du premier ordre afin d'avoir une axiomatique associée à la logique des prédicats du premier ordre avec l'égalité.

Les schémas d'axiome sont les suivants, soient x une variable, f une fonction n-aire et P un prédicat atomique n-aire :

AXI-7. Les axiomes de $\mathcal{L}_{\mathcal{P}'}$

AXI-8. $x = x$ *réflexivité*

AXI-9. $x_i = y \rightarrow f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n)$
pour $1 \leq i \leq n$ *substitutivité des fonctions*

AXI-10. $x_i = y \rightarrow (P(x_1, \ldots, x_i, \ldots, x_n) \rightarrow P(x_1, \ldots, y, \ldots, x_n))$
pour $1 \leq i \leq n$ *substitutivité des prédicats*

[3] Par convention on utilise la notation $t_1 = t_2$ et non la notation habituelle des prédicats binaires $= (t_1, t_2)$.

Les règles d'inférence sont les mêmes que pour $\mathcal{L}_{\mathcal{P}'}$, c'est-à-dire *Modus Ponens* et *Généralisation*.

En utilisant la règle d'inférence de généralisation on peut obtenir les axiomes suivants : $\forall x\ (x = x)$, $\forall x_1 \ldots \forall x_n\ \forall y\ (x_i = y \rightarrow f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n))$ et $\forall x_1 \ldots \forall x_n\ \forall y\ (x_i = y \rightarrow (P(x_1, \ldots, x_i, \ldots, x_n) \rightarrow P(x_1, \ldots, y, \ldots, x_n)))$.

En utilisant la notation $P[t]$ (cf. Annexe A) les schémas d'axiome AXI-9 et AXI-10 peuvent être remplacés par le schéma d'axiome suivant : $x = y \rightarrow (P[x] \rightarrow P[y])$ qui dénote une substitution plus générale que celle de AXI-10. Puisque dans $x = y \rightarrow (P[x] \rightarrow P[y])$, x et y peuvent être des sous-termes, le schéma AXI-9 est redondant.

Malgré les résultats obtenus, qui permettent de prouver qu'un ensemble de formules Σ est satisfaisable dans la logique des prédicats du premier ordre avec égalité si et seulement si $\Sigma \cup \mathcal{E}$ est satisfaisable dans la logique des prédicats de premier ordre, où $\mathcal{E}$ représente les axiomes associés à l'égalité (réflexivité, substitutivité pour les fonctions et substitutivité pour les prédicats), il n'existe pas d'équivalence entre les deux logiques. La logique avec égalité a un pouvoir d'expression plus grand. Par exemple, dans cette logique la formule $(x = a \vee x = b \wedge \neg(a = b)$ exprime le fait que 'le modèle a un nombre d'éléments au moins égal à deux', fait que l'on ne peut exprimer dans la logique des prédicats du premier ordre sans l'égalité.

B.4 Logique Modale

Les logiques précédentes assignent à une formule la valeur $vraie$ ou $fausse$. La logique modale, largement utilisée dans l'Intelligence Artificielle, donne d'autres significations aux formules. Les philosophes l'ont utilisée pour s'informer des différents modes de vérité, tels que *possiblement vrai* et *nécessairement vrai*.

Par exemple, si l'on veut représenter formellement 'l'existence des extraterrestres est possible' on peut utiliser $\Diamond \exists x\ extraterrestre(x)$, où $extraterrestre(x)$ veut dire 'x est un extraterrestre'. De façon similaire 'nécessairement toute personne a une mère' peut être représenté par $\Box(\forall y\ personne(y) \rightarrow \exists x\ mère(x, y))$, où $personne(x)$ veut dire 'x est une personne' et $mère(x, y)$ veut dire 'x est la mère de y'.

Le langage de la logique modale est celui de la logique du premier ordre $\mathcal{L}_{\mathcal{P}'}$ étendu avec deux opérateurs modaux $\Box$ (pour nécessaire) et $\Diamond$ (pour possible). Le langage de cette logique $\mathcal{L}_{\mathcal{PM}}$ est défini récursivement comme suit :

SYN-5. Les règles de $\mathcal{L}_{\mathcal{P}'}$ appliquées à $\mathcal{L}_{\mathcal{PM}}$

SYN-6. Si $\psi \in \mathcal{L}_{\mathcal{PM}}$ alors $\Box\psi$, $\Diamond\psi \in \mathcal{L}_{\mathcal{PM}}$

En ce qui concerne la sémantique, la notion de *monde possible* ou *état possible* est introduite. Ainsi par exemple, la satisfaisabilité de 'nécessairement ψ' est définie, selon Leibniz, comme 'nécessairement ψ' est $vrai$ si ψ est $vrai$ dans tous les mondes possibles. Plus tard, le rapport existant entre les mondes

possibles est aussi considéré. Pour cela une relation d'accessibilité est introduite. Cette relation lie des paires de mondes possibles. Soit R la relation d'accessibilité, $w_1 R w_2$ signifie que w_2 est accessible à partir de w_1.

Une interprétation (M_{PM}, v) est constituée d'une structure M_{PM} et d'une fonction d'assignation de variables v.

Cette interprétation est semblable à celle de la logique des prédicats du premier ordre sauf qu'ici on ajoute à la structure M_{PM} un ensemble des mondes possibles $W = \{w_1, \ldots, w_o\}$ et une relation d'accessibilité R . Donc $M_{PM} \stackrel{\text{def}}{=} < W, R, \mathbb{D}, p_1^{(M_{PM},w_1)}, \ldots, p_1^{(M_{PM},w_o)}, p_2^{(M_{PM},w_1)}, \ldots, p_2^{(M_{PM},w_o)}, \ldots, p_n^{(M_{PM},w_1)}, \ldots, p_n^{(M_{PM},w_o)}, f_1^{(M_{PM},w_1)}, \ldots, f_1^{(M_{PM},w_o)}, f_2^{(M_{PM},w_1)}, \ldots, f_2^{(M_{PM},w_o)}, \ldots, f_m^{(M_{PM},w_1)}, \ldots, f_m^{(M_{PM},w_o)} >$, où $\mathbb{D}$ est un ensemble non vide appelé univers ou domaine[4] de M_{PM}. Chaque $p_i^{(M_{PM},w_j)} \subseteq \mathbb{D}^{n_i}$ définit l'ensemble de n_i-uples qui sont vrais associés au symbole de prédicat p_i dans le monde w_j. $p_i^{(M_{PM},w_j)}$ est appelé l'extension de p_i dans le monde w_j. Chaque $f_i^{(M_{PM},w_j)} : \mathbb{D}^{n_i} \mapsto \mathbb{D}$ est une fonction n_i-aire sur $\mathbb{D}$ dans le monde w_j. Les constantes sont des cas particuliers des fonctions où $n_i = 0$.

La fonction d'assignation des variables est définie de la même façon que celle de la logique des prédicats du premier ordre, i.e. $v : V \mapsto \mathbb{D}$.

La fonction d'assignation des termes est définie de la façon suivante[5]. Si x est variable $\bar{v}(x, w) = v(x)$ et si $t_1, t_2, \ldots, t_n$ sont des termes et f un symbole de fonction n-aire alors $\bar{v}(f(t_1, t_2, \ldots, t_n), w) = f^{M_{PM}}(\bar{v}(t_1, w), \bar{v}(t_2, w), \ldots, \bar{v}(t_n, w), w)$.

La satisfaction d'une formule dans un monde w par une interprétation (M_{PM}, v) est définie récursivement comme suit :

SEM-9. $M_{PM}, v, w \models p(t_1, t_2, \ldots, t_n)$ ssi $< \bar{v}(t_1, w), \bar{v}(t_2, w), \ldots, \bar{v}(t_n, w) > \in p^{(M_{PM},w)}$

SEM-10. Les règles SEM-5 et SEM-6 avec (M_{PM}, v) comme interprétation et w comme monde

SEM-11. $M_{PM}, v, w \models \Box\psi$ ssi pour toute $w' \in W$ si wRw' alors $M_{P'}, v, w \models \psi$

La définition de l'opérateur ($\Diamond$) est la suivante : $\Diamond\psi \stackrel{\text{def}}{=} \neg\Box(\neg\psi)$.

Plusieurs logiques modales sont caractérisées sémantiquement par les propriétés satisfaites par leurs relations d'accessibilité. Ainsi par exemple, la relation d'accessibilité de la logique **KT** a la propriété que chaque monde est accessible par lui-même, la relation est reflexive.

Quelques propriétés utilisées pour la caractérisation des logiques sont :

Sérielle, R est sérielle ssi pour tout $w \in W$ il existe un $w' \in W$ tel que wRw'.

Réflexive, R est réflexive ssi pour tout $w \in R$ on a wRw.

[4]Cette structure est définie à domaine constant. Dans la sémantique on peut aussi définir une structure à domaine variable, laquelle associe un domaine à chaque monde [FM98].

[5]Dans cette définition le terme est appelé non rigide. La terminologie *terme rigide* $\bar{v}$: $T \times W \mapsto \mathbb{D}$, où v assigne à chaque terme du langage un élément fixé du domaine $\mathbb{D}$ dans un monde spécifique w, qui a été introduite par Kripke, signifie intuitivement que le terme désigne toujours le même objet quelque soit le monde dans lequel on le mentionne [Kri80, FM98].

Symétrique, R est symétrique ssi pour tout $w, w' \in W$ si wRw' alors $w'Rw$.

Transitive, R est transitive ssi pour tout $w, w', w'' \in W$ si wRw' et $w'Rw''$ alors wRw''.

Le tableau ci-dessous montre quelques logiques standards et les propriétés satisfaites par leur relation d'accessibilité.

logique	*propriété(s)*
K	aucune
KD	sérielle
KT	réflexive
K4	transitive
S4	réflexive et transitive
S5	réflexive, symétrique et transitive

La caractérisation axiomatique de ces logiques est basée sur celle de la logique **K** qui se compose des schémas d'axiomes suivants, soient A et B deux formules :

AXI-11. Les schémas d'axiomes de $\mathcal{L}_{\mathcal{P}'}$

AXI-12. $\Box(A \to B) \to (\Box A \to \Box B)$ *schéma K*

et les règles d'inférence sont les mêmes que celles de $\mathcal{L}_{\mathcal{P}'}$, c'est-à-dire *Modus Ponens* et *Généralisation* plus la règle de *Nécessitation*.

AXI-RI-3. Les règles AXI-RI-1 et AXI-RI-2 de $\mathcal{L}_{\mathcal{P}'}$

AXI-RI-4. Si $\vdash A$ alors $\vdash \Box A$ *Nécessitation*

L'axiomatique des logiques **KD**, **KT**, etc. considèrent les schémas suivants[6] :

D. $\Box A \to \Diamond A$

T. $\Box A \to A$

4. $\Box A \to \Box\Box A$

5. $\Diamond A \to \Box\Diamond A$

Ainsi par exemple, la logique **KD** est une extension de la logique **K** avec le schéma D. Le tableau ci-dessous montre les logiques et les schémas correspondants ajoutes à la logique **K**.

logique	*schéma ajouté à la logique* **K**
KD	D
KT	T
K4	4
S4	T, 4
S5	T, 5

[6] En fait, la caractérisation axiomatique des logiques a été donnée avant la caractérisation sémantique. Le succès le plus important des sémantiques des mondes possibles est d'avoir permis de caractériser ces logiques en imposant des conditions aux relations d'accessibilité.

Les formules de Barcan (plutôt schémas d'axiome) caractérisent des permutations entre les opérateurs modaux et les quantificateurs, elles ont la forme suivante :

$$\begin{aligned} \forall x \Box\psi &\rightarrow \Box\forall x\, \psi \\ \Box\forall x\, \psi &\rightarrow \forall x \Box\psi. \end{aligned}$$

Ces schémas doivent être ajoutés aux axiomatiques des logiques à domaine constant pour avoir une axiomatique complète, pour le cas des logiques à domaine variable ils ne sont pas nécessaires [FM98].

Annexe C

Notions fondamentales en Logique

C.1 Satisfaisabilité

Satisfaisabilité. Une formule ψ est satisfaite par une interprétation M (ou ψ est vraie dans M) ssi $M \models \psi$. Une formule est satisfaisable ssi il existe une interprétation qui la satisfasse.

Modèle. Une interprétation M est un modèle d'une formule ψ ssi M satisfait ψ. M est un modèle d'un ensemble de formules, éventuellement infini ssi M est un modèle pour chaque formule de l'ensemble.

Validité. Une formule ψ est valide, dénoté par $\models \psi$, ssi chaque interprétation du langage est un modèle de la formule ψ.

Insatisfaisabilité. Une formule est insatisfaisable ssi elle n'a aucun modèle.

Conséquence logique. Si Γ est un ensemble de formules, éventuellement infini, et ψ une formule, ψ est conséquence logique de Γ, dénoté par $\Gamma \models \psi$, ssi tout modèle de Γ est un modèle de ψ. Un résultat intéressant pour la déduction automatique est le suivant : $\Gamma \models \psi$ ssi il existe un sous-ensemble fini Γ' de Γ tel que $\Gamma' \models \psi$.

Validité et complétude de la logique. La déduction des formules satisfaisables peut être formalisée en donnant un ensemble convenable de formules appelées axiomes (d'habitude un ensemble fini de schémas d'axiomes) et les règles d'inférence qui permettent de dériver des formules à partir des formules précédentes. Cette axiomatisation permet de construire des preuves de formules à partir des ensembles de formules appelées prémisses ou hypothèses. Dans une axiomatisation donnée de la logique, l'expression $\Gamma \vdash \psi$ signifie qu'il existe une preuve de ψ à partir des hypothèses Γ. Une axiomatisation est *complète* ssi chaque fois que l'on a $\Gamma \models \psi$ alors on a $\Gamma \vdash \psi$. Une axiomatisation est *valide* ssi chaque fois que l'on a $\Gamma \vdash \psi$ alors on a $\Gamma \models \psi$.

C.2 Univers de Herbrand et termes clos

Par définition un ensemble de clauses Σ est insatisfaisable si et seulement si il est *faux* pour toute interprétation. On remarque qu'il n'est pas possible de considérer toutes les interprétations, néanmoins il existe un domaine spécial $\mathbb{H}(\Sigma)$, appelé l'univers d'Herbrand de Σ, tel que Σ est insatisfaisable si et seulement si Σ est insastisfaisable sous toute interprétation qui a pour le domaine $\mathbb{H}(\Sigma)$.

Définition 23 *(Univers de Herbrand).* Soit Σ un ensemble de clauses, l'*univers de Herbrand* de Σ $\mathbb{H}(\Sigma)$ est défini récursivement par :

1. Tout symbole de constante apparaissant dans une clause de Σ est un élément de $\mathbb{H}(\Sigma)$. S'il n'y a pas des symboles de constantes apparaissant dans les clauses, alors $\mathbb{H}(\Sigma)$ contient le symbole de constante a.

2. Si f^n es un symbole de fonction n-aire apparaissant dans Σ et si $t_1, t_2, \ldots t_n$ sont des éléments de $\mathbb{H}(\Sigma)$ alors $f(t_1, t_2, \ldots t_n)$ est un élément de $\mathbb{H}(\Sigma)$.

Définition 24 *(Terme clos).* Un élément de $\mathbb{H}(\Sigma)$ est appelé *terme de Herbrand* ou *terme clos*[1] de Σ. Une *clause close* de Σ est une clause de Σ telle que toutes ses variables ont été remplacées par des termes clos de Σ. En général, une expression (terme, littéral ou clause) qui ne contienne pas de variable est appelé une *expression close*.

C.3 Occurrence de variable liée et libre

Définition 25 *(Occurrence de variable liée, occurrence de variable libre)*[2]. Dans une formule de la forme $\forall x\ \psi$, chaque occurrence de la variable x dans ψ est appelée *occurrence de variable liée*. Les occurrences de variable qui ne sont pas liées sont appelées *occurrence de variable libre*.

Définition 26 *(Formule fermée).* Une *formule fermée* est une formule sans variable libre.

Définition 27 *(Terme libre pour une variable dans une formule).* Soient x une variable, t un terme et A une formule. La définition de *t est libre pour x dans A* est donnée de façon récursive comme suit :

(a) Si A est une formule atomique alors t est libre pour x dans A.

(b) Si A a une des cinq formes, $\neg B$, $B \vee C$, $B \wedge C$, $B \rightarrow C$ ou $B \leftrightarrow C$, et si t est libre pour x dans B et dans C alors t est libre pour x dans A.

(c) Si A a une des deux formes, $\forall v\ B$ ou $\exists v\ B$, et si $x = v$ alors t est libre pour x dans A.

(d) Si A a une des deux formes, $\forall v\ B$ ou $\exists v\ B$, si $x \neq v$, si v n'a aucune occurrence dans t et si t est libre pour x dans B, alors t est libre pour x dans A.

[1] En anglais *ground term*.

[2] On utilise parfois les noms : variable liée et variable libre pour designer ces occurrences.

C.4 Substitution et unification

Définition 28 *(Substitution).* Une *substitution* $\{t_1/x_1, t_2/x_2, \ldots, t_n/x_n\}$ est un ensemble fini d'éléments, où chaque t_i est un terme, chaque x_i est une variable, et aucune d'éléments $t_i/x_i, t_j/x_j$ n'ont la même variable après le symbole '/'.

Définition 29 *(Instance).* Soit E une expression (terme ou formule) du langage et $\sigma = \{t_1/x_1, t_2/x_2, \ldots, t_n/x_n\}$ une substitution. Une *instance* de E, $E\sigma$ est une expression qui résulte du remplacement de chaque occurrence libre de x_i par t_i dans E pour $1 \leq i \leq n$.

Définition 30 *(Unificateur).* Soit $\{E_1, E_2, \ldots, E_n\}$ un ensemble fini (non vide) d'expressions. Un *unificateur* θ pour $\{E_1, E_2, \ldots, E_n\}$ est une substitution telle que $E_1\theta = E_2\theta = \ldots = E_n\theta$. L'ensemble est dénommé *unifiable* s'il possède un unificateur.

Définition 31 *(Unificateur le plus général (mgu)).* Un unificateur σ pour un ensemble d'expressions est un appelé *unificateur le plus général* ssi pour chaque unificateur θ de l'ensemble il existe une substitution λ telle que $\theta = \sigma\lambda$, où $\sigma\lambda$ est une composition de substitutions [CL73].

Définition 32 *(Unification).* Soit $\{E_1, E_2, \ldots, E_n\}$ un ensemble unifiable d'expressions. L'*unification* de l'ensemble consiste en trouver l'unificateur le plus général.

Annexe D

Code de la SOLE-résolution avec des heuristiques

Le code ci-dessous correspond à la procédure de la SOLE-résolution avec des heuristiques dans l'application de la règle E-inférence.

```
/*****************************************/
/* Compilation des librairies necessaires */
/*****************************************/


:- [library(antiunify)].    % anti_unify/5
:- [library(lists)].
:- [library(occurs)].
:- [library(same_functor)].
:- [library(term_depth)].
:- [library(unify)].


/*************************************/
/*                                   */
/*  Version 5 de la SOL-Deduction    */
/*                                   */
/*************************************/

% C is the list with the top clause and [P,Q] the list with the resultat
% 100 fois, il essaie

sol(C):- retractall(c(X)),
 assert(c(C)),
 appelle_sol([[],C],S,100),
 write_list(S).
% retract(c(C)).
```

```
appelle_sol([P,[]],[P,[]],N):- !,
            nl, write('***************sol au niveau'-N), nl.
appelle_sol(D,D,0):- !, fail.
appelle_sol(Di,[Di1|LDf],N):-
        copy_term(Di,Di1),
sol_d(Di,Df),
N1 is N - 1,
nl, write(niveau-N), nl, write(Df), nl,
appelle_sol(Df,LDf,N1).

write_list([P,[]]):-!, write([P,[]]), nl.
write_list([T|R]):- write(T), nl, write_list(R).

sol_d(Di, [P,Q]):-
\+(tautologie(Di)),
        \+subsu(Di),
app_champ(Di),
left_most(Di,[P,R]),
truncation(R,Q).

left_most(Di,Df):-
( reduce(Di, Df)  , X=red     |

                  resolve(Di, Df) , Y=res      |

                    skip2(Di, Df)   , Z=ski    )  |

                    \+((X==red|Y==res|Z==ski)) , resolve_sub(Di, Df).


/*********************************/
/*              SKIP             */
/*********************************/

/* skip2(D1,D2) vrai si D2 est la clause structuree obtenue par */
/* application du skip sur D1 */

skip2([P1,[L|R]], [P2,R]) :-
 reunion(L, P1, P2),
                 app_champ([P2,_]).


         % reunion pour ne pas avoir repete L dans P

reunion(L,P1,[L|P2]) :-
                enlever(L,P1,P2).
```

```
/*********************************/
/*           RESOLVE             */
/*********************************/

   /* resolve(D1,D2) vrai si D2 est la clause structuree obtenue par
      application du resolve sur D1 */

resolve([P,[L|R]],[P,Q]) :-
                negation(L,NL),
choisi_hyp(NL,R,B),
enlever(NL,B,B1),
append(B1,[frame(L)|R],Q).

choisi_hyp(NL,R,B) :-
   copy_term(NL,NL1),
   (c(B) | hyp(B)),
   unifier(NL,B),
                  profundeur(B,NL1).

profundeur(H,NLP):- findall(ST,sub_term(ST,H),S),
     depth_do(S,D),
     findall(STP,sub_term(STP,NLP),SP),
     depth_do(SP,DP),
     D=<DP,!.

depth_do([],0):-!.
depth_do([T|R],D):-(nonvar(T),T=do(X,Y),term_depth(do(X,Y),D1),
  depth_do(R,D2),
  (D1<D2,D=D2|D=D1))| depth_do(R,D),!.

/*************************************************/
/*          E-inf\'erence avec des heuristiques     */
/*************************************************/

    /* resolve(D1,D2) vrai si D2 est la clause structuree obtenue par
       application du resolve sur D1 avec  substitution */

resolve_sub([P,[non(X=X)|R]],[P,R]).

resolve_sub([P,[L|R]],[P,Q]) :-
                copy_term(L,LP),
 ( c(B) | hyp(B) ),
                ags_lateral(L,B,B1),
                profundeur(B1,LP),
         append(B1,[frame(L)|R],Q).
```

```
resolve_sub([P,[non(L)|R]],[P,Q]) :-
                copy_term(L,LP),
                lit_egal(L,X=Y),
                \+simple(X),
                ags(X=Y,L_egal,NL), % ags_central
                ( c(B) | hyp(B) ),
                unifier_list(L_egal,B),
                append(L_egal,[NL],B1),  %pour verifier aussi NL
                is_posible_egal(B1,B2),
                profundeur(B2,non(L)),
append(B2,[frame(non(L))|R],Q).

unifier_list([],_):-!,fail.

unifier_list([L|R],B):-
    negation(L,L1),
    unifier(L1,B).

unifier_list([_|R],B):-unifier_list(R,B).

ags_lateral(_,[],_):-!, fail.

ags_lateral(X=Y,[L2|_],H):-    % lateral  p(X)    central X=Y
   % \+lit_egal(L2,L2),   % posible que soit non(L2) \`a la place de L2
    negation(L2,NL2),
    ags(NL2,L_egal,NNL2),
    unifier(non(X=Y),L_egal),
    enlever(non(X=Y),L_egal,L_egal_2),
    is_posible_egal(L_egal_2,L_egal_3),
    append(L_egal_3,[NL2,NNL2],H).

ags_lateral(non(L),[L2|_],H):-  % lateral X1=Y1  central non (X=Y)
    lit_egal(L,X=Y),
    unifier(X=Z,[L2]),
    is_posible_egal([non(Y=Z)],L_egal),
    append(L_egal,[non(X=Z)],H).

ags_lateral(L,[L2|R],H):-    % central et lateral <> =
    \+lit_egal(L,L),
    ags(L2,L_egal,NL2),
    unify(L,NL2),   % predicat diferent d'egal
    negation(L2,NL2_2),
    is_posible_egal(L_egal,L_egal_2),
    append(L_egal_2,[NL2_2],H).

ags_lateral(L,[_|R],H):-ags_lateral(L,R,H).
```

```
is_posible_egal([],[]):-!.

is_posible_egal([non(X=Y)|R],R1):- X==Y,is_posible_egal(R,R1),!.

is_posible_egal([non(X=Y)|R],[non(X=Y)|R1]):-
   (c(B)|hyp(B)),
           copy_term(X,X1),
           copy_term(Y,Y1),
           ( unifier(X1=Z,B) | unifier(Y1=Z,B)),
           profundeur(B,non(X=Y)),
           is_posible_egal(R,R1),!.

is_posible_egal([non(X=Y)|R],[non(X=Y)|R1]):-!,
           X=Y,
           is_posible_egal(R,R1),!.

is_posible_egal([L|R],[L|R1]):-  is_posible_egal(R,R1),!.  %Un autre predicat

ags(L,L_Egal,non(L1)):-
    \+L=..[non|_],
    L=..[F|L_Para],
    param(L_Para , L1_Para , L_Egal),
    L1=..[F|L1_Para].

ags(non(L),L_Egal,L1):-
    L=..[F|L_Para],
    param(L_Para , L1_Para , L_Egal),
    L1=..[F|L1_Para].

param([],[],[]):-!.

param([A|R],[A1|L2_Para],L_Egal):-
    \+simple(A),
    A=..[F|L_Para],
    param(L_Para,L1_Para,L1_Egal),
    param(R,L2_Para,L2_Egal),
    A1=..[F|L1_Para],
    append(L1_Egal,L2_Egal,L_Egal).

param([A|R],[X|L_Para],[non(X=A)| L_Egal]):-
     param(R,L_Para,L_Egal).


/*********************************/
/*           REDUCE              */
/*********************************/
```

```
/* reduce(D1,D2) vrai si D2 est la clause structuree obtenue par */
/* application du reduce sur D1 */

reduce([P,[L|R]],[P,R]) :-
                ( factoring_merge(P,R,L) |
                  ancestry(R,L) ).


factoring_merge(P,Q,K):-
  append(P,Q,PQ),
  unifier(K,PQ).  % pour merge, appartient

   /* ancestry vrai si le litteral deja resolu non(L) est unifiable
      avec un litteral de Q */


ancestry(Q,L) :-
negation(L,NL),
unifier(frame(NL),Q).



/*********************************/
/*          TAUTOLOGIE           */
/*********************************/

   /* tautologie(C) vrai si la clause structuree C est une tautologie */

tautologie([P,Q]) :-
append(P,Q,C),
construire_clause(C,C2),
tautologie2(C2).

tautologie2([X=Y|_]):-X==Y,!.
%tautologie2([non(X=Y)|_]) :- atomic(X), atomic(Y), \+X==Y,!. apres por NUniques
tautologie2([_|[]]) :- !,fail.

tautologie2([L|LX]) :-
negation(L,NL),
appartient(NL,LX) ,!.
tautologie2([_|R]) :- tautologie2(R).


/*********************************/
/*          SUBSUME              */
/*********************************/
```

```
subsu([_,Q]):- subsu2(Q).

subsu2([]):-!,fail.

subsu2([L|R]):-
      appartient(frame(L),R), !.

subsu2([L|R]):-
      litteral(L,L1),
      sub_anti_unifier(frame(L1),R,I),
      sub_arguments(I),!.

subsu2([_|R]):-subsu2(R).


sub_arguments([]):-!.
sub_arguments([non(X=Y)|R]):-(var(Y) | X==Y),sub_arguments(R).

sub_anti_unifier(_,[],_) :- fail.
sub_anti_unifier(L1,[L2|_],I) :-
                \+L1==L2,  % verifie avec appartient
anti_unify(L1,L2,R1,R2,U),
                sub_ver_unificateur(U),
                ver_arguments(R1,R2,I).

sub_anti_unifier(L,[_|LX],B) :-
sub_anti_unifier(L,LX,B).



sub_ver_unificateur(frame(non(U))):-!,nonvar(U).
sub_ver_unificateur(frame(U)):-!,nonvar(U).
%sub_ver_unificateur(non(U)):-!,nonvar(U).
%sub_ver_unificateur(U):-nonvar(U).

ver_arguments([],[],[]):-!.

%ver_arguments([V=T1|R1],[V=T2|R2],[non(T1=T2)|I]):-ver_arguments(R1,R2,I).

ver_arguments([V=T1|R1],[V=T2|R2],[non(T2=T1)|I]):-ver_arguments(R1,R2,I).



/*********************************/
/*   QUELQUES PREDICATS UTILES   */
/*********************************/
```

```
appartient(_,[]) :- !,fail.
appartient(frame(L),[L2|_]) :- litteral(L,L1), frame(L1)==L2,!.
appartient(X,[Y|_]) :- litteral(X,X1), X1 == Y,!.
appartient(X,[_|R]) :- appartient(X,R).


    /* app_champ_de_production(L,C) vrai si le litteral L
       et la clause C appartiennent au champ de production */

app_champ([[],_]):-!.
app_champ([[L|C],_]):-
champs(C2),
app_langage(L,C2),
%satisfait_lit_cond(L),
satisfait_cond([L|C]).

app_langage(_,[]) :- !,fail.
app_langage(non(L),[C|_]) :-
                same_functor(L,C),!.
app_langage(non(L),[_|LC]) :-!,
                app_langage(L,LC).
app_langage(L,[C|_]) :-
                same_functor(L,C),!.
app_langage(L,[_|LC]) :-
                app_langage(L,LC).



    /* construire_clause(C1,C2) vrai si C2 est la clause C1
       sans ses litteraux encadres */

%construire_clause([],[]) :- !.
%construire_clause([frame(_)|Q],C) :- construire_clause(Q,C),!.
%construire_clause([L|Q],[L|C]) :- construire_clause(Q,C).

construire_clause([frame(_)|[]],[]) :- !.
construire_clause([frame_SI(_)|[]],[]) :- !.
construire_clause([L|[]],[L]) :- !.
construire_clause([frame(_)|Q],C) :-
construire_clause(Q,C),!.
construire_clause([frame_SI(_)|Q],C) :-
construire_clause(Q,C),!.
construire_clause([L|Q],C) :-
construire_clause(Q,TEMP),
append([L],TEMP,C).
```

```
    /* enlever(X,L,LR) enleve la premiere occurence de X dans la liste L
       pour donner la liste LR */

enlever(_,[],[]) :- !.
enlever(X,[Y|L],L) :-
                litteral(X,X1),
X1 == Y,!.
enlever(X,[Y|L],[Y|R]) :-
enlever(X,L,R).


     /* litteral(L,L1) vrai si L1 est L ou L simetrique */

litteral(L,L).
litteral(X=Y,Y=X).
litteral(non(X=Y),non(Y=X)).


    /* lit_egal(L,L1) vrai si L1 est l'egalite, inclues sa simetrique */

lit_egal(X=Y,X=Y).
lit_egal(non(X=Y),non(X=Y)).
lit_egal(X=Y,Y=X).
lit_egal(non(X=Y),non(Y=X)).


     /* negation(L,NL) vrai si NL est la negation du litteral L */

negation(non(L),L) :- !.
negation(L,non(L)).


     /* Ces deux predicats peuvent etre redefinis dans la base de donnees.
Le premier permet de poser une condition de longueur de
la clause solution
Le deuxieme permet de poser une condition de profondeur des termes */

satisfait_cond(_).

%satisfait_lit_cond(_).


     /* truncation(D1,D2) vrai si D2 est la clause structuree obtenue par
application du truncation sur D1 */

truncation([],[]) :- !.
truncation([frame(_)|R],Q) :- truncation(R,Q),!.
```

```
truncation(Q,Q).


     /* unifier(L,C) vrai si le litteral L s'unifie avec un litteral de C */

unifier(_,[]) :- fail.

unifier(frame(L),[L2|_]) :-
                litteral(L,L1),
                unify(frame(L1),L2).

unifier(L,[L2|_]) :-
                litteral(L,L1),
unify(L1,L2).

unifier(L,[_|R]) :-
unifier(L,R).
```

Bibliographie

[AGM85] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2):510–530, 1985.

[And58] A.R. Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, 67:100–103, 1958.

[BC96] M.A. Brown and J. Carmo (eds). *Deontic Logic, Agency and Normative Systems*. Springer, 1996.

[BGLS95] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[Bra75] D. Brand. Proving Theorems with the Modification Method. *SIAM Journal of Computing*, 4(4):412–430, 1975.

[Bra84] M. Bratman. Two faces of intention. *Philos. Rev.*, 93:375–405, 1984.

[Bra87] M. Bratman. Intentions, Plans, and Practical Reason. 1987.

[Bro87] F. M. Brown, editor. *The Frame Problem in Artificial Intelligence*. Morgan Kaufmann Publishers, 1987.

[Bro88] M. Brown. On the logic of ability. *Journal of Philosophical Logic*, 17:1–26, 1988.

[Cas95] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pages 41–48, 1995.

[Che88] B.F. Chellas. *Modal Logic: An introduction*. Cambridge University Press, 1988.

[CL73] C.L. Chang and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CL90a] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[CL90b] P.R. Cohen and H.J. Levesque. Rational interaction as the basis for communication. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communications*. The MIT Press, 1990.

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 292–322, New York, 1978. Plenum Press.

[CMM90] P.R. Cohen, J. Morgan, and M.Pollack. *Intentions in Communication*. The MIT Press, 1990.

[Colss] M. Colombetti. A modal logic of intentional communication. *Mathematical Social Science*, In press.

[Dem99] R. Demolombe. To trust information sources: a proposal for a modal logical framework. In C. Castelfranchi and Y-H. Tan, editor, *Trust and Deception in Virtual Societies*. Kluwer, 1999.

[Dem00] R. Demolombe. Action et Causalité: essais de formalisation en logique. In H. Prade and R. Jeansoulin and C. Garbay, editor, *Le Temps, l'Espace et l'Evolutif en sciences du traitement de l'information*. CEPADUES-EDITIONS, 2000.

[Dem01] R. Demolombe. Formalisation en logique des interactions entre agents: quels concepts formaliser? Technical report, ONERA-CERT, DTIM, 2001.

[DF91] R. Demolombe and L. Fariñas del Cerro. An Inference Rule for Hypothesis Generation. In *Proc. of International Joint Conference on Artificial Intelligence*, Sydney, 1991.

[DH00] R. Demolombe and R. Hilpinen. *Proc. of 5th Int. Workshop on Deontic Logic and Computer Science*. ONERA, 2000.

[DMWK96] F. Dignum, J-J. Meyer, R. J. Wieringa, and R. Kuiper. A modal approach to intentions, commitments and obligations: intention plus commitment yields obligation. In M. Brown and J. Carmo, editors, *Deontic Logic, Agency and Normative Systems*, pages 194–215. Springer, 1996.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *ACM*, 7(3):201–215, 1960.

[DP00] R. Demolombe and P. Pozos Parra. A simple and tractable extension of situation calculus to epistemic logic. In Z. W. Ras and S. Ohsuga, editors, *Proc. of 12th International Symposium ISMIS 2000*. Springer. LNAI 1932, 2000.

[DP01a] R. Demolombe and P. Pozos Parra. An extension of sol-resolution to theories with equality. In *3er Encuentro Internacional de Computación*, Aguascalientes, Mexico, 2001.

[DP01b] R. Demolombe and P. Pozos Parra. Formalisation de l'évolution des croyances dans le calcul des situations. In *1ères journées francophones des Modèles formels de l'interaction (MFI)*, Toulouse, France, 2001.

[DP01c] R. Demolombe and P. Pozos Parra. Relación de accesibilidad u operador modal: un dilema para formalizar la evolución de las creencias en el cálculo de situaciones. In *3er Encuentro Internacional de Computación*, Aguascalientes, Mexico, 2001.

[DvL96] F. Dignum and B. van Linder. Modeling Social Agents: Communication as Action. In M. Wooldridge and N. Jennings, editors, *Proc. of 3rd International Workshop on Agents, Theories and Languages ATAL-96*, 1996.

[Far82] L. Fariñas del Cerro. A simple deduction method for modal logic. *Information Processing Letters*, 14(2), 1982.

[Far85] L. Fariñas del Cerro. MOLOG : A system that extends PROLOG with modal logic. Technical report, Université Paul Sabatier, Toulouse, France, 1985.

[FH88] R. Fagin and J. Halpern. Belief, awareness and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.

[FHMV95] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[FM98] M. Fitting and R. L. Mendelsohn. *First-Order Modal Logic*. Kluwer, 1998.

[FN71] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–205, 1971.

[Gär88] P. Gärdenfors. *Knowledge in flux: modeling the dynamics of epistemic states*. The MIT Press, 1988.

[GK96] B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.

[GN87] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufman, 1987.

[Göd81] K. Gödel. *Obras completa*. Alianza Editorial, 1981.

[Har84] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 2. Reidel, 1984.

[HB95] J. F. Horty and N. Belnap. The deliberative STIT: a study of action, omission, ability, and obligation. *Journal of Philosophical Logic*, 24:583–644, 1995.

[HC72] G. E. Hughes and M. J. Cresswell. *An introduction to modal logic.* Methuen London and New York, 1972.

[Her30] J. Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et des Lettres de Varsovie. Classe III sciences mathématiques et physiques*, (No. 33):128 pages, 1930.

[Hil97] R. Hilpinen. On Action and Agency. In E. Ejerhed and S. Lindstrom, editors, *Logic, Action and Cognition: Essays in Philosophical Logic.* Kluwer, 1997.

[Hin63] J. Y. Hintikka. *Knowledge and belief.* Cornell University Press, Ithaca, New-York, 1963.

[HL00] A. Herzig and D. Longin. Belief dynamics in cooperative dialogues. *J. of Semantics*, 17(2), 2000. vol. published in 2001.

[Ino91] K. Inoue. Consequence-Finding Based on Oredered Linear Resolution. In *Proc. of International Joint Conference on Artificial Intelligence*, Sydney, 1991.

[Ino92a] K. Inoue. Linear Resolution for Consequence Finding. *Artificial intelligence, an International Journal*, 56, 1992.

[Ino92b] K. Inoue. *Studies on Abductive and Nonmonotonic Reasoning.* PhD thesis, Kyoto University, 1992.

[Jon90] A.J.I. Jones. Toward a Formal Theory of Communication and Speech Acts. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communications.* The MIT Press, 1990.

[Kle67] S.C. Kleene. *Mathematical Logic.* John Wiley and Sons, 1967.

[KM91] H. Katsuno and A.O. Mendelzon. On the difference between updating a Knowledge Base and Revising it. In *Proc. of KR'91*, 1991.

[Kon86] K. Konolidge. *A Deduction Model of Belief.* Pitman Publishing, 1986.

[Kri80] S. Kripke. *Naming and Necessity.* Harvard University Press, Cambridge, 1980.

[LCN90] H.J. Levesque, P.R. Cohen, and J.T. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*, pages 94–99, 1990.

[Lev84] H. J. Levesque. A logic of implicit and explicit belief. In *Proc. of the 4th National Conference on Artificial Intelligence*, 1984.

[Lev96] H. J. Levesque. What is planning in the presence of sensing? In *Proceedings of the National Conference on Artificial Intelligence*, pages 1139–1146, 1996.

[Lew18] C. I. Lewis. *A Survey of Symbolic Logic.* PhD thesis, University of California, 1918.

[LL98] G. Lakemeyer and H. Levesque. AOL: a logic of acting, sensing, knowing and only knowing. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 316–327, 1998.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, 1987.

[LLR99] Y. Lespérance, H. Levesque, and R. Reiter. A situation calculus approach to modeling and programming agents. In A. Rao and M. Wooldbridge, editors, *Foundations and Theories of Rational Agents.* Kluwer, 1999.

[Lon99] D. Longin. *Interaction rationnelle et évolution des croyances dans le dialogue : une logique basée sur la notion de topique.* PhD thesis, Université Paul Sabatier, Toulouse France, 1999.

[Lov78] D. Loveland. *Automated Theorem Proving: a logical basis.* North-Holland, 1978.

[LRL+97] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 31:59–84, 1997.

[McC68] J. McCarthy. Programs with Common Sense. In M. Minski, editor, *Semantic Information Processing.* The MIT press, 1968.

[McD82] D. McDermott. Non-monotonic logic ii : Non-monotonic modal theories. *ACM*, 29(1):34–57, 1982.

[MD80] D. McDermott and J. Doyle. Non-monotonic logic i. *Artificial Intelligence*, 13:41–72, 1980.

[Me99] P. McNamara and H. Prakken (eds). *Norms, Logics and Information Systems.* IOS Press, 1999.

[Men64] E. Mendelsson. *Introduction to Mathematical Logic.* D. Van Nostrand Company, 1964.

[MH69] J. C. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intellifence. In D. Michie B. Meltzer, editor, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[Moo80] R. C. Moore. *Reasonning about Knowledge and Action.* PhD thesis, Massachussets Institute of Technology, 1980.

[Moo84] R. C. Moore. Possible-world semantics for auto-epistemic logic. *AAAI-Workshop on Non-Monotonic Reasoning*, pages 344–354, 1984.

[Moo85] R. C. Moore. A formal theory of knowledge and action. In Jerry B. Hobbs and Robert C. Moore, editors, *Formal Theories of Commonsense World*, pages 319–358. Ablex Publishing Corp., Norwood, New Jersey, 1985.

[MS86] J. P. Martins and S. C. Shapiro. Theoretical foundations for belief revision. In *Reasoning about knowledge*, 1986.

[MW93] J.-J. Ch. Meyer and R. J. Wieringa. Deontic Logic: a Concise Overview. In J.-J. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*. John Wiley and Sons, 1993.

[NR99] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 1999.

[Per90] C.R. Perrault. An Application of Default Logic to Speech Act Theory. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions in Communications*. The MIT Press, 1990.

[Por77] I. Porn. Action Theory and Social Science. Some Formal Models. *Synthese Library*, 120, 1977.

[Poz99] P. Pozos Parra. Modelando agentes: sus acciones y conocimientos. In *2do Encuentro Nacional de Computación*, Pachuca, Mexico, 1999.

[Poz00] P. Pozos Parra. Modélisation des croyances des agents dans le calcul des situations. In *5ème Recontres Nationales des Jeunes Chercheurs en IA*, Lyon, France, 2000.

[Pri67] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[Rei78] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.

[Rei80] R. Reiter. A logic for default reasonning. *Artificial Intelligence*, 13:81–132, 1980.

[Rei87] R. Reiter. Nonmonotonic reasoning. In *Annual Reviews of Computer Science, 2*, 1987.

[Rei91] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Rei93] R. Reiter. Proving Properties of States in the Situation Calculus. *Artificial Intelligence*, 64:337–351, 1993.

[Rei99] R. Reiter. Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. Technical report, University of Toronto, 1999.

[Rei01] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[RG91] A.N. Rao and M.P. Georgeff. Modeling Rational Agents within a BDI Architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann, 1991.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[RW69] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In R. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, Edinburgh, 1969. Edinburgh University Press.

[Sad92] D. Sadek. A study in the logic of intention. In *Proc. of the 3rd Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, 1992.

[SC96] F. Santos and J. Carmo. Indirect action, influence and responsibility. In M. Brown and J. Carmo, editors, *Deontic Logic, Agency and Normative Systems*, pages 194–215. Springer, 1996.

[Sch90] L. K. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H. E. Kyberg, R. P Loui, and G. N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, 1990.

[Seg89] K. Segerberg. Bringing it about. *Journal of Philosophical Logic*, 18:327–347, 1989.

[Seg92] K. Segerberg. Getting started: beginnings in the logic of action. *Studia Logica*, 51:347–378, 1992.

[Sho67] J. R. Shoenfiels. *Mathematical Logic.* Addison-Wesley, 1967.

[Sho88] Y. Shoham. *Reasoning about change.* The MIT Press, 1988.

[Sho97] Y. Shoham. Agent-oriented programming. In Michel N. Huhns and Munindar P. Singh, editors, *Reading in Agents*, pages 329–349. Morgan Kaufmann, 1997.

[Sin94] M. P. Singh. *Multiagent Systems. A Theoretical Framework for Intentions, Know-How, and Communications.* Springer LNAI 799, 1994.

[SL91] W. Snyder and C. Lynch. Goal directed strategies for paramodulation. In *Proc. 14th Int. Conf. on Rewriting Techniques and Applications (LNCS 488)*. Springer-Verlag, 1991.

[SL93] R. Scherl and H. Levesque. The Frame Problem and Knowledge Producing Actions. In *Proc. of the National Conference of Artificial Intelligence*. AAAI Press, 1993.

[SPLL00] S. Shapiro, M. Pagnuco, Y. Lespérance, and H. Levesque. Iterated belief change in the situation calculus. In *Proc. of the 7th Conference on Principles on Knowledge Representation and Reasoning (KR2000)*. Morgan Kaufman Publishers, 2000.

[SRG98] M. P. Singh, A. Rao, and M. Georgeff. Formal Method in DAI: Logic Based Representation and Reasoning. In G. Weis, editor, *Introduction to Distributed Artificial Intelligence*, New York, 1998. MIT Press.

[T+89] A. Thayse et al. *Approche logique de l'intelligence artificielle*, volume 2. Dunod, 1989.

[T+90] A. Thayse et al. *Approche logique de l'intelligence artificielle*, volume 1. Dunod, 1990.

[Tho84] R. H. Thomason. Combination of Tense and Modality. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosphical Logic Vol. II*. Reidel, 1984.

[Tur84] R. Turner. *Logics for Artificial Intelligence*. Ellis Horwood, Chichster, 1984.

[vL96] B. van Linder. *Modal Logics for Rational Agents*. PhD thesis, University of Utrecht, 1996.

[vvM98] B. van Linder, W. van der Hoek, and J-J. Ch. Meyer. Formalising Abilities and Opportunities of Agents. *Fundamenta Informaticae*, 34, 1998.

[vW51] G.H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.

[vW63] G. H. von Wright. *Norm and Action*. Routledge and Kegan, 1963.

[vW68] G. H. von Wright. An Essay in Deontic Logic and the General Theory of Action. *Acta Philosophica Fennica*, 21, 1968.

[War74] D. H. Warren. Warplan: A system for generating plans. Memo 76, Departement of Computational Logic, University of Edinburgh, June 1974.

[WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Enginnering Review*, 10(2), 1995.

[WJ97] M. Wooldridge and N. Jennings. Formalizing the cooperative problem solving process. In Michel N. Huhns and Munindar P. Singh, editors, *Reading in Agents*. Morgan Kaufmann, 1997.

[Woo92] M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, University of Manchester, 1992.