N° d'ordre : 22

Année 1988

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR

de

L'ECOLE NATIONALE SUPERIEURE
DE L'AERONAUTIQUE ET DE L'ESPACE

SPECIALITE : INFORMATIQUE

par

Frédéric CUPPENS

COMMENT FOURNIR DES REPONSES COOPERATIVES AUX REQUETES A UNE BASE DE DONNEES

Soutenue le 3 novembre 1988 devant la Commission d'Examen :

MM. H. FARRENY — **Président**

F. BANCILHON
A. COLMERAUER
R. DEMOLOMBE — **Examinateurs**

Résumé

Cette thèse présente une méthode, et sa formalisation, pour fournir des informations supplémentaires intéressantes en réponse à des requêtes posées à une base de données relationnelle. Ces informations intéressantes sont définies en utilisant une base de connaissances contenant des règles représentant le savoir-faire d'un expert habitué à fournir des réponses à des utilisateurs occasionnels.

Cette base de connaissances contient également une description de la base de données. Cette description de haut niveau utilise les notions d'entité, d'attributs d'entité, de relation et de *"thème"*. Les *thèmes* sont associés aux attributs et aux relations et permettent de regrouper les informations de la base de données qui appartiennent à un même champ sémantique.

Les bases de données et de conaissances sont toutes deux formalisées en logique du premier ordre. Il y a toutefois deux niveaux différents de formalisme : un niveau objet pour représenter la base de données elle-même, et un méta-niveau pour représenter la base de connaissances utilisée pour transformer les requêtes. Ces requêtes transformées définissent les informations supplémentaires à fournir à l'utilisateur. Elles sont obtenues en utilisant un mécanisme de déduction classique. Pour réaliser cette transformation, il est important de tenir compte des caractéristiques de chaque utilisateur.

Dans cette thèse, seul l'aspect sémantique de l'information a été considéré, et l'on ne s'est pas intéressé aux aspects linguistiques, tel que la représentation en langue naturelle.

Il faut également signaler qu'un premier prototype implanté en PROLOG fonctionne à l'heure actuelle.

Mots Clés : Base de données
Base de connaissances
Logique
Métalangage
Réponse coopérative
Information voisine
Modèle de l'utilisateur

Abstract

This thesis presents a method, and its formalization, to provide interesting additional information to users asking queries to a Relational Data Base. The interesting information is defined using a Knowledge Base containing rules which represent the expertise of a person having a long experience in providing information to casual users.

This Knowledge Base also contains a high level description of the Data Base content. This description uses the concepts of entity, attribute of entity, relationship and topic. The topics are related to attributes and relationships in order to be able to characterize all the information in the Data Base which belongs to a given semantic field.

The Data Base and Knowledge Base are both formalized in First Order Logic. There are two different formalization levels: object level to represent the Data Base itself, and meta level to represent the Knowledge Base which is used to transform queries. The transformed queries define the additional information which is provided to users. They can be obtained with a standard deduction mechanism. To perform this transformation, it is important to take into account significant user's characteristics.

In this thesis, only the semantic aspect of the information is considered and we are not interested on linguistic aspects, such as natural language representation.

We have also to notice that a first prototype implemented in Prolog is running.

Keywords : Database
Knowledge base
Logic
Metalanguage
Cooperative answering
Neighborhood information
User model

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réalisation de cette thèse :

- *Monsieur Robert Demolombe, ingénieur de recherches au Département d'Etudes et de Recherches en Informatique du C.E.R.T, responsable scientifique de ce travail, pour son soutien patient, sa disponibilité et ses conseils éclairés qui m'ont permis de mener à bien cette étude. Je lui suis très reconnaissant de la confiance qu'il m'a toujours témoignée.*
- *Monsieur Henri Farreny, professeur à l'Ecole Nationale Supérieur d'Electrotechnique, d'Electronique, d'Informatique et d'Hydraulique de Toulouse, qui m'a fait l'honneur de présider le jury de cette thèse,*
- *Messieurs Alain Colmerauer, professeur à la faculté des Sciences de Luminy, et Daniel Kayser, professeur à l'université de Paris Nord, qui ont bien voulu s'intéresser à ce travail et accepter d'en être les rapporteurs,*
- *Messieurs François Bancilhon, directeur du G.I.P Altair, et Jean-Marie Nicolas, senior researcher au European Computer Research Center de Munich, pour avoir accepté de participer au jury de cette thèse,*
- *Messieurs François Régis Valette, chef du Département d'Etudes et de Recherches en Informatique du C.E.R.T, et René Jacquart, son successeur, qui m'ont permis d'effectuer ce travail dans les meilleures conditions,*
- *Mademoiselle Laurence Cholvy et monsieur Kioumars Yazdanian, tous deux ingénieurs de recherches au Département d'Etudes et de Recherches en Informatique du C.E.R.T, pour leurs conseils et leurs critiques constructifs,*
- *Mademoiselle Elsa Pascual et monsieur Jacques Virbel pour m'avoir aidé dans mes recherches bibliographiques,*
- *Madame Leslie Lamport, ingénieur à la Digital Equipment Corporation et réalisatrice de LaTeX, qui m'a permis d'effectuer une présentation convenable de cette thèse,*
- *Le personnel d'édition du C.E.R.T qui en a assuré la réalisation matérielle.*

Table des Matières

Introduction

La coopération est un aspect essentiel de la conversation entre les individus. En effet, pour qu'un dialogue puisse avoir lieu, il est nécessaire que les interlocuteurs respectent un niveau minimum de coopération. Pour coopérer, les êtres humains observent un grand nombre de règles, de conventions et de postulats qui permettent de communiquer les intentions et les croyances en plus de la signification littérale de leur propos.

En première approche, on peut considérer qu'un individu a un comportement coopératif si, face à une question, il ne fournit pas seulement la réponse exacte du point de vue logique, mais y ajoute des informations supplémentaires qui ne sont pas explicitement demandées, mais qui lui semblent liées implicitement à la question posée. Ce sont les usages de la conversation qui permettent de déduire les informations pertinentes à fournir en plus dans la réponse.

Par exemple, si un individu A demande à un autre individu B :

A : Pouvez-vous me donner l'heure, s'il vous plait ?

et si B a une montre qui marche, alors la réponse *"oui"* est correcte d'un point de vue logique.

Toutefois B n'a pas fourni une réponse qui respectait les règles de coopération. En effet B doit comprendre que le but de A est de connaître l'heure et non pas de savoir si B connait l'heure. B doit ensuite fournir une réponse qui respecte les conventions de la conversation. Dans l'exemple ci-dessus, ces conventions conduisent B à essayer de satisfaire les intentions de A, c'est-à-dire connaître l'heure. Pour cela, il va fournir, par exemple la réponse suivante :

B : Il est trois heures dix.

Dans certains cas, d'autres règles de coopération peuvent conduire B à fournir une autre réponse telle que :

B : Ma montre indique trois heures et quart, mais je crois qu'elle avance un peu.

Ici, un comportement coopératif a conduit B à préciser qu'il n'était pas sûr de l'exactitude de sa réponse.

Ce petit exemple montre que l'on peut distinguer plusieurs formes de coopération qui utilisent chacune leurs propres règles et conventions. Ce sont la question et son contexte qui conduisent l'interlocuteur à choisir telles règles de coopération plutôt que telles autres.

Ainsi, une première forme de coopération se situe dans un contexte où la requête échoue, c'est-à-dire retourne l'ensemble vide comme réponse. Il s'agit alors de fournir, lorsque c'est possible, une réponse indirecte. Par exemple, considérons le dialogue suivant qui pourrait avoir lieu avec une base de données "classique" :

Q1 : Quels sont les étudiants qui ont obtenu la licence de langue au printemps 1979 ?

R : NIL. (ensemble vide)

Q2 : Est-ce que quelqu'un a échoué à la licence de langue au printemps 1979 ?

R : Non.

Q3 : Combien de personnes ont passé la licence de langue au printemps 1979 ?

R : Zéro.

Q4 : Est-ce que le cours de licence de langue était assuré en 1979 ?

R : Non.

Dans cet exemple, le SGBD n'a pas été coopératif. En effet, un système coopératif doit être capable de reconnaître que la requête initiale suppose incorrectement qu'un cours de licence en matière linguistique était dispensé en 1979, et dans ce cas, fournir une réponse qui prenne en compte cette hypothèse erronée. Une conversation similaire avec un interlocuteur humain se serait déroulée de la façon suivante :

Q1 : Quels sont les étudiants qui ont obtenu la licence de langue au printemps 1979 ?

R : Le cours de licence de langue n'était pas assuré en 1979.

La réponse fournie par R est appelée **réponse indirecte.** Nous verrons par la suite que cet aspect de la coopération dans le dialogue a déjà fait l'objet d'études.

Une deuxième forme de coopération dans le dialogue consiste à fournir une réponse suggestive. Il s'agit de suggérer, en plus de la réponse exacte à la question de l'utilisateur, une alternative pertinente.

Il existe plusieurs contextes pouvant conduire une personne à fournir une réponse suggestive.

Ainsi, considérons la question suivante :

Q : Puis-je avoir des places à l'orchestre pour la représentation de la Traviata de dimanche prochain ?

S'il n'y a pas de représentation de la *Traviata* ce dimanche, un interlocuteur coopératif remarquera que la requête suppose incorrectement qu'une représentation de la *Traviata* a lieu dimanche. Il fournira donc, comme nous venons de le voir, une réponse indirecte qui pourra avoir la forme suivante :

R1 : Il n'y a pas de représentation de la Traviata dimanche.

Par contre, si la représentation a bien lieu ce jour là, mais s'il n'y a plus de place à l'orchestre alors la requête échoue sans que celle-ci présente une présupposition érronée. Un interlocuteur peut donc fournir la réponse directe suivante :

R2 : Non, toutes les places à l'orchestre sont déjà réservées.

Toutefois, s'il reste des places au balcon, un interlocuteur coopératif en fera probablement la suggestion à l'utilisateur, et fournira donc la réponse suivante :

R3 : Désolé, il n'y a plus de place à l'orchestre, mais il en reste au balcon. Est-ce que ça vous intéresse ?

Ici, l'interlocuteur ajoute à la réponse directe (*Désolé, il n'y a plus de place à l'orchestre*) une alternative qu'il juge intéressante (*mais il en reste au balcon. Est-ce que ça vous intéresse ?*). C'est ce que nous appellerons par la suite une **réponse suggestive.**

Dans l'exemple ci-dessus, c'est l'échec de la question qui a conduit l'interlocuteur à fournir une telle réponse. Toutefois ce n'est pas la seule situation qui peut inciter un interlocuteur à avoir ce type de comportement coopératif, comme l'on peut s'en rendre compte au travers du dialogue suivant :

A : Quel bus faut-il prendre pour se rendre à la place du Capitole ?

B : Vous pouvez prendre le 22, mais vous aurez plus vite fait d'y aller à pied, c'est à 5 minutes d'ici.

Dans cet exemple, bien que la requête n'échoue pas, B fournit également une alternative qu'il juge pertinente. Pour cela, B reconnait que le but de A est probablement de se rendre à la place du Capitole, et considère que la meilleure solution est de s'y rendre à pied. Il en fait donc la suggestion à A. Il faut également remarquer que B fournit des informations supplémentaires (*"c'est à 5 minutes d'ici"*) destinées à convaincre A de la pertinence de cette suggestion.

Une troisième forme de coopération consiste à fournir dans certains cas, ce que l'on appelle une **réponse complétive**. Il s'agit de fournir à l'utilisateur une réponse qui apporte des renseignements complémentaires nécessaires à l'utilisation de la réponse, comme c'est le cas pour la question suivante :

Q : Quels sont les prix des actions des entreprises du secteur pétrolier ?

Une réponse courante d'une base de données :

Prix
395
470
...

Ces réponses ne sont d'aucune utilité puisque qu'on ne connait pas les actions auxquelles sont associés les prix. Une réponse complétive pourrait être dans ce cas :

Prix	Action
395	Elf-Aquitaine
470	Total
...	

Une quatrième forme de coopération consiste à fournir des **informations supplémentaires pertinentes** concernant la réponse à une question. Ainsi, examinons l'exemple de dialogue suivant où A, transportant un bidon vide, vient vers B et lui demande :

A : Où est la station service la plus proche ?

et si B répond :

B : Au prochain carrefour.

en sachant pertinemment que cette station est fermée, alors B n'a pas été coopératif car il devait fournir en plus ce fait à l'utilisateur. En effet, la situation doit conduire B à reconnaître que le but de A est de remplir son bidon d'essence, ce qu'il ne pourra pas réaliser si la station est fermée. B, ayant détecté cet obstacle, doit, s'il est coopératif, l'indiquer à A.

Nous verrons par la suite qu'il y a plusieurs situations pouvant conduire B à avoir ce genre de comportement coopératif.

Une dernière forme de coopération consiste à répondre de façon pertinente à des questions *oui-non*, comme c'est le cas pour la question que nous avons déjà présentée ci-dessus :

Q : Pouvez-vous me donner l'heure, s'il vous plait ?

Cette liste des différentes formes de coopération que l'on rencontre lorsque l'on étudie des dialogues ne cherche pas à être exhaustive. Son seul but est d'essayer de présenter de façon intuitive les différents problèmes sur lesquels nous allons revenir dans la suite de ce travail. Les idées conductrices de cette thèse seront :

1. Etudier les règles permettant à un individu d'avoir un comportement coopératif, comme dans les exemples présentés ci-dessus.
2. Montrer comment ces règles peuvent s'exprimer dans un formalisme logique.
3. Décrire les inférences effectuées par un individu qui essaye d'avoir un tel comportement.

Dans le chapitre 1, nous présentons, à l'aide d'exemples, les divers aspects du problème que nous envisageons de traiter. Dans le chapitre 2, nous faisons une étude bibliographique des travaux portant sur ce problème. Cette partie se termine par un bilan comparatif entre ces travaux et l'approche que nous explorons. Le chapitre 3, qui termine la première partie, essaye de récapituler les connaissances que nous manipulons dans cette étude. La deuxième partie présente le formalisme logique que nous utilisons pour représenter ces connaissances. Enfin la troisième partie étudie en détail divers aspects de la coopération dans un dialogue et montre comment ces problèmes peuvent être résolus. Dans la conclusion, après avoir récapitulé les divers problèmes étudiés dans cette thèse, nous essayons de dégager les limites de notre approche et suggérons des prolongements naturels à ce travail.

Partie I

Présentation du problème

Chapitre 1

Présentation du problème sous forme d'exemples

Nous essayons, dans cette partie, de montrer comment une personne, ayant une compétence dans un domaine particulier, peut fournir des informations pertinentes supplémentaires à une personne posant une requête. Dans ce qui suit, la personne qui pose la question sera appelée "l'utilisateur", et la personne qui fournit les réponses sera appelée "le serveur". Notre but est de simuler le raisonnement du serveur, avec un logiciel appelé "système de réponses coopératives".

Notre approche, dans cette partie, est de mettre en évidence, au travers d'exemples, les différents concepts permettant la représentation des connaissances utilisées par le serveur pour déterminer les informations pertinentes. Par la suite, nous montrerons comment ces concepts peuvent être formalisés en logique. Le sous-ensemble de la base de données est présenté de façon semi-formelle à l'aide de concepts similaires à ceux du modèle Entité-Relation.

Pour montrer la généralité des notions que nous introduisons, nous présentons des exemples empruntés à deux domaines différents :

- Le domaine financier lié à la gestion de portefeuilles.
- Le domaine d'une agence de voyage.

Nous commençons par présenter les données manipulées dans ces deux exemples :

Exemple financier :

Type d'entité : Action
Attributs : Dernier-cours
Cours-précédent
Dernier-coupon :

Date
Montant-global
Variation
Rendement
Secteur-d'activité
Nationalité

Exemple "agence de voyage" :

Type d'entité : Vol
Attributs : Ville-de-départ
Ville-d'arrivée
Heure-de-départ
Heure-d'arrivée
Prix
Période
Opère

Requêtes :

Considérons maintenant les requêtes suivantes, exprimées également dans un langage semi-formel :

QS1 : Pour les Actions
Telles que :
Secteur-d'activité = Exploitation-Pétrolière
Rendement > 5%
Variation > 0%
Fournir les valeurs des attributs : Rendement, Variation.

La requête QS1 est adressée à une base de données financière. La réponse fournie par un SGBD relationnel standard pourrait être, par exemple :

AS1 :

Action	Rendement	Variation
Elf-Aquitaine	+7%	+65%
Total	+8%	+50%
Exxon	+5.3%	+12%
Royal-Dutch	+5.5%	+12%

La requête suivante est adressée à la base "agence de voyage".

QT1 : Pour les Vols
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = New-York

Heure-de-départ appartient à [8 00 , 12 00]
Fournir la valeur de l'attribut : Heure-de-départ.

La réponse, fournie par une base de données relationnelle standard pourrait être :

AT1 :

Vol	Heure-de-départ
AF001	10h00
AF001	11h00
AF015	10h30

1.1 Transformation des conditions

Dans le cas de la requête QS1, un serveur coopératif pourrait proposer d'autres actions qui ne sont pas des solutions exactes, mais qui sont dans un certain sens "voisines". Par exemple, comme la requête fait référence à des actions du secteur de l'Exploitation-Pétrolière, on peut en déduire que l'utilisateur est intéressé par les actions dont l'activité est liée au Pétrole, et peut donc être également intéressé par des actions appartenant à d'autres secteurs d'activité du domaine pétrolier tels que : Raffinage, Service-Pétrolier ou Stockage-Pétrolier.

Il pourrait également proposer des actions ayant des valeurs "similaires" pour les attributs Rendement et Variation.

Ainsi, des informations pertinentes supplémentaires pourraient être obtenues avec la requête :

QS2 : Pour les Actions
Telles que:
Secteur-d'activité = Exploitation-Pétrolière **ou**
Raffinage **ou**
Service-Pétrolier **ou**
Stockage-Pétrolier
Rendement > 4%
Variation > -5%
Fournir les valeurs des attributs : Rendement, Variation.

La réponse AS2 pourrait être, par exemple :

AS2 :

Action	Rendement	Variation
Elf-Aquitaine	+7%	+65%
Total	+8%	+50%
Exxon	+5.3%	+12%
Royal-Dutch	+5.5%	+12%
Esso-SAF	+7%	−2.5%
Petrole-BP	+4.5%	+75%
Raffinage-Total	+12%	+80%
C.I.M.	+8%	+1%

Toutefois, comme le serveur a modifié la requête initiale, il est nécessaire de rendre explicite la nouvelle réponse pour l'utilisateur. Ceci peut être partiellement réalisé en montrant à l'utilisateur la requête modifiée, mais c'est insuffisant. Dans la requête initiale, seules les actions du secteur Exploitation-Pétrolière étaient demandées, aussi est-il souhaitable d'indiquer à l'utilisateur le secteur d'activité des actions. Pour cela on ajoute Secteur-d'activité à la liste des attributs à fournir et on obtient ainsi la requête QS3 :

QS3 : Pour les Actions
Telles que :
Secteur-d'activité = Exploitation-Pétrolière **ou**
Raffinage **ou**
Service-Pétrolier **ou**
Stockage-Pétrolier
Rendement > 4%
Variation > -5%
Fournir les valeurs des attributs :
Rendement, Variation et Secteur-d'activité.

et les réponses à QS3 pourraient être :

AS3 :

Action	Rendement	Variation	Secteur-d'activité
Elf-Aquitaine	+7%	+65%	Exploitation
Total	+8%	+50%	Exploitation
Exxon	+5.3%	+12%	Exploitation
Royal-Dutch	+5.5%	+12%	Exploitation
Esso-SAF	+7%	−2.5%	Exploitation
Petrole-BP	+4.5%	+75%	Exploitation
Raffinage-Total	+12%	+80%	Raffinage
C.I.M.	+8%	+1%	Stockage

Naturellement, la notion de pertinence des informations supplémentaires fournies à l'utilisateur dépend de nombreux critères.

En particulier, cette notion dépend des caractéristiques de l'usager et de sa capacité de formuler la "bonne" question (c'est-à-dire la question qui lui fournira les réponses dont il a besion).

Dans de nombreux cas, cette notion de pertinence, dépend également de la valeur d'un certain critère choisi pour comparer les entités. Ces critères dépendent fortement du domaine d'application, du contexte de la question ainsi que caractéristiques de l'utilisateur.

Par exemple, pour certains types d'usager, un critère d'optimisation pourra être le risque du placement (que l'on cherchera à minimiser) pour d'autres ce sera la plus-value possible (que l'on cherchera à maximiser).

Nous verrons par la suite comment nous avons essayé de prendre en compte ces différentes notions de pertinence.

Pour la requête QT1, de la même façon que pour la requête QS1, le serveur pourrait proposer des vols ayant une ville de départ proche de Paris, par exemple Bruxelles ou Londres, ou ayant une ville d'arrivée proche de New-York, par exemple Washington ou Boston, et dont les conditions sur l'heure de départ diffèrent de moins de deux heures de la requête initiale. Cela conduit à la nouvelle requête modifiée QT2 :

QT2 : Pour les Vols
Tels que :
Ville-de-départ = Paris **ou** Bruxelles **ou** Londres
Ville-d'arrivée = New-York **ou** Washington **ou** Boston
Heure-de-départ appartient à [6 00 , 14 00]
Fournir les valeurs des attributs : Heure-de-départ.

La réponse pourrait être, par exemple :

AT2 :

Vol	Heure-de-départ
AF001	10h00
AF001	11h00
AF015	10h30
AF077	13h00
AF020	10h30
BA017	10h00
BA142	12h15

Comme pour la requête QS2, il est évident que l'utilisateur doit connaître également la ville de départ et la ville d'arrivée pour chaque vol solution. La requête modifiée QT3 doit donc fournir la valeur de ces attributs.

QT3 : Pour les Vols
Tels que :
Ville-de-départ = Paris **ou** Bruxelles **ou** Londres

Ville-d'arrivée = New-York **ou** Washington **ou** Boston
Heure-de-départ appartient à [6 00 , 14 00]
Fournir les valeurs des attributs :
Heure-de-départ, Ville-de-départ, Ville-d'arrivée.

Les réponses correspondantes pourraient être, par exemple :

AT3 :

Vol	Heure-de-départ	Ville-de-départ	Ville-d'arrivée
AF001	10h00	Paris	New-York
AF001	11h00	Paris	New-York
AF015	10h30	Paris	New-York
AF077	13h00	Paris	New-York
AF020	10h30	Paris	Washington
BA017	10h00	Londres	New-York
BA142	12h15	Londres	Boston

Dans ces deux exemples (requêtes QS1 et QT1) les informations demandées par l'utilisateur sont définies par une requête ayant la forme générale suivante :

"Pour les entités ayant un **type** donné
satisfaisant certaines **conditions**,
fournir les valeurs de tels **attributs**."

Les informations pertinentes supplémentaires fournies par le serveur sont définies par une requête ayant la forme générale suivante :

"Pour les entités ayant un **type** donné
satisfaisant certaines **conditions étendues**,
fournir les valeurs de tels **attributs**
et de tels **attributs supplémentaires**."

La caractéristique principale de l'information supplémentaire est qu'elle concerne un ensemble d'entités plus large que l'ensemble explicitement demandé dans la requête de l'utilisateur, le type des entités recherchées restant le même. Cet ensemble étendu est défini par une nouvelle requête dans laquelle les **conditions** sont modifiées en assouplissant certaines contraintes. Une conséquence de ce changement est qu'il est nécessaire de fournir à l'utilisateur la valeur des attributs dont la condition a été modifiée.

De ces exemples, on peut conclure qu'une première façon de définir l'information pertinente supplémentaire est de modifier la partie **condition** de la requête initiale, en relâchant certaines contraintes, et en ajoutant à la liste des **attributs** à fournir, ceux dont les conditions ont été modifiées. Ce type de transformation sera appelé dans la suite : "transformation des conditions".

Pour réaliser ces transformations, il semble raisonnable de supposer que le serveur possède une certaine expérience concernant les informations par lesquelles l'utilisateur

peut être intéressé, et qu'il connait des règles indiquant comment la partie condition d'une requête peut être modifiée.

Par exemple, dans le contexte de la base de données financière, une de ces règles exprimée de façon informelle, pourrait être :

RS1 : **Si** la requête concerne les actions,
Et il y a une partie condition qui impose un secteur d'activité donné,
Alors ce secteur d'activité peut être remplacé par une disjonction d'autres secteurs d'activité appartenant au même secteur d'activité générique.

Dans l'exemple QS1, l'utilisateur a précisé Exploitation-Pétrolière comme secteur d'activité. Dans ce cas, le secteur d'activité générique est Pétrole, et la disjonction est : Exploitation-Pétrolière **ou** Raffinage **ou** Service-Pétrolier **ou** Stockage-Pétrolier. Cet exemple montre que le serveur doit connaître non seulement des règles pour réaliser la transformation mais également des faits sur les informations contenues dans la base de données. Ces faits peuvent être représentés par la relation suivante :

Secteur-d'activité	Secteur-d'activité-générique
Exploitation-Pétrolière	Pétrole
Raffinage	Pétrole
Service-Pétrolier	Pétrole
Stockage-Pétrolier	Pétrole

D'autres règles concernant le Rendement et la Variation pourraient être :

RS2 : **Si** une requête concerne les actions,
Et il y a une partie condition concernant un Rendement donné
Alors ce rendement peut être remplacé par ce même rendement plus ou moins 1%

RS3 : **Si** une requête concerne les actions,
Et il y a une partie condition concernant une Variation donnée
Alors cette Variation peut être remplacée par cette même Variation plus ou moins 5%

On remarquera que les règles RS1, RS2 et RS3 sont liées au domaine d'application (ici la gestion de portefeuille). Par ailleurs, on aura des règles indiquant quelles sont les informations supplémentaires à fournir lorqu'on applique l'une des règles RS1, RS2 ou RS3 ci-dessus pour transformer une requête. Ces règles, indépendantes du domaine d'application seront présentées par la suite.

Dans le cas de la base de données "Agence de Voyage" les règles pourraient être :

RT1 : **Si** une requête concerne les Vols,
Et il y a une partie condition concernant une Heure-de-départ donnée
Alors l'Heure-de-départ peut être remplacée par cette même heure plus ou moins une heure.

Pour les conditions sur la Ville-de-départ et sur la Ville-d'arrivée, il est évident que les deux conditions ne peuvent être modifiées indépendamment. Par exemple, pour un vol de Paris à Bruxelles, la Ville-de-départ ne peut être remplacée par Bruxelles. On pourrait donc avoir une règle telle que :

RT2 : **Si** une requête concerne les Vols,
Et il y a une partie condition sur une Ville-de-départ donnée et sur une Ville-d'arrivée donnée
Alors ces villes peuvent être remplacées par d'autres villes dont la distance par rapport aux originales n'excède pas 10% de la distance de la Ville-de-départ à la Ville-d'arrivée.

A partir des exemples précédents, on peut exhiber la forme générale de ces règles :

Si une requête concerne un type d'entité donnée,
Et il y a une partie condition concernant la valeur d'un attribut donné,
Alors la valeur donnée peut être remplacée par un ensemble de valeurs voisines.

Comme on peut s'en rendre compte au travers des exemples, ces valeurs voisines peuvent être définies de différentes manières qui dépendent des domaines de ces valeurs.

1.2 Transformation des informations demandées

Considérons maintenant un autre type d'information additionnelle pertinente. L'idée, dans ce cas, n'est pas de fournir des entités supplémentaires, mais de fournir davantage d'informations pour les mêmes entités.

Par exemple, dans le cas de la requête QS1, il pourrait être intéressant de fournir en plus la valeur des attributs : Dernier-Cours, Montant-global et la Nationalité pour les Actions dont la nationalité n'est pas celle de l'utilisateur; disons la France, par exemple. Ceci pourrait être réalisé par la requête QS4 :

QS4 : Pour les Actions
Telles que :
Secteur-d'activité = Exploitation-Pétrolière
Rendement > 5%
Variation > 0%
Fournir les valeurs des attributs : Rendement, Variation,
Dernier-Cours, Montant-global, Nationalité.

La réponse AS4 pourrait être, par exemple :

AS4 :

Action	Rendement	Variation	Dernier Cours	Montant Global	Nationalité
Elf-Aquitaine	+7%	+65%	340	20.2	
Total	+8%	+50%	430	30	
Exxon	+5.3%	+12%	450	26.7	U.S.A
Royal-Dutch	+5.5%	+12%	595	26.8	Pays-Bas

L'une des raisons qui conduisent l'utilisateur à poser la question QS1 est probablement qu'il désire acheter (ou vendre) les actions solutions de la requête QS1. Lorsque l'utilisateur voudra réaliser son intention d'acheter ou de vendre, le cours de l'action sera une information qu'il sera utile de connaître. C'est la raison pour laquelle le système fournit cette information supplémentaire à l'utilisateur.

D'après sa question, l'utilisateur semble intéressé par le rendement des actions. On lui fournit donc l'information concernant le montant global du dernier dividende versé, car cette information est rattachée au même *thème*.

Enfin, la nationalité de l'action est une information qu'il est intéressant de fournir puisque l'utilisateur peut avoir des avantages divers lorsqu'il investit dans des valeurs de son propre pays.

On peut remarquer que l'on n'a fourni la valeur de cet attribut que pour certaines solutions. Ceci pose un problème car un langage de requête relationnel standard ne permet pas de fournir la valeur d'un attribut seulement pour des tuples satisfaisant certaines conditions.

Les règles utilisées pour dériver la nouvelle requête QS4 à partir de QS1 pourraient être par exemple les règles RS4 et RS5 :

RS4 : **Si** une requête concerne les Actions,
Et il y a des attributs à fournir concernant le *thème* Plus-Value,
Alors les autres attributs concernant la Plus-Value doivent être insérés dans la liste des Attributs à fournir.

Dans cette règle la notion de *thème* est plus abstraite et moins précise que la notion d'attributs. Cette notion permet de regrouper plusieurs attributs qui font référence à un sujet similaire. La relation entre attributs et *thèmes* doit être connue par le serveur; dans cet exemple on pourrait avoir :

Attribut	Thème
Rendement	Plus-Value
Variation	Plus-Value
Montant-Global	Plus-Value

En général, un attribut donné peut faire référence à plusieurs *thèmes*. De plus, les *thèmes* eux-mêmes peuvent être stucturés hiérarchiquement.

RS5 : **Si** une requête concerne les Actions,
Et la nationalité de l'utilisateur est différente de la nationalité d'une action donnée,
Alors la valeur de l'attribut Nationalité de cette Action doit être fournie.

Cette règle montre que le serveur doit connaître certaines informations concernant l'utilisateur; dans le cas présent, la nationalité de l'utilisateur.

Considérons maintenant un exemple similaire avec la base "agence de voyage". Pour la requête QT1, l'information supplémentaire à fournir pourrait être la valeur des attributs : Heure-d'arrivée, Période et Opère. Plus précisément, la valeur de l'attribut période n'est intéressante que pour les vols qui ne fonctionnent pas toute l'année et celle de l'attribut Opère n'est intéressante que pour les vols qui n'opèrent pas tous les jours.

Ceci peut être fourni par la requête QT4 :

QT4 : Pour les Vols
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = New-York
Heure-de-départ appartient à [8 00 , 12 00]
Fournir les valeurs des attributs : Heure-de-départ,
Heure-d'arrivée, Période, Opère.

La réponse pourrait être :

AT4 :

Vol	Heure-de-départ	Heure-d'arrivée	Période	Opère
AF001	10h00	8h45	1/01 27/9	
AF001	11h00	9h45	28/9 31/12	
AF015	10h30	14h55	1/01 27/9	1,3,6
AF015	10h30	14h55	28/9 31/12	1,3,6,7

Il est clair, dans cet exemple, que la personne qui est intéressée par l'Heure-de-départ peut également être intéressée par l'Heure-d'arrivée. La période de validité est également intéressante puisqu'elle permet de comprendre l'étrange réponse AT1, dans laquelle un vol donné a deux heures de départ différentes. De plus, certains vols n'opèrent pas tous les jours, et ces jours dépendent de la période de validité.

Dans cet exemple, comme pour la requête QS4, il y a deux attributs, Période et Opère, dont les valeurs ne doivent être fournies que pour certains tuples : respectivement, les vols qui ne fonctionnent pas toutes l'année, et les vols qui n'opèrent pas tous les jours.

Les règles utilisées par le serveur pour transformer la requête QT1 en QT4 pourraient être :

RT3 : **Si** une requête concerne les vols,
Et les attributs à fournir concernent le *thème* Horaire,
Alors les autres attributs concernant l'Horaire doivent être insérés dans la liste des attributs à fournir.

Cette règle conduit le serveur à fournir l'heure d'arrivée comme information supplémentaire.

RT4 : **Si** une requête concerne les vols
Et la période de validité du vol n'est pas du 1 Janvier au 31 Décembre
Alors la valeur de l'attribut Période doit être fournie.

RT5 : **Si** une requête concerne les vols
Et le vol n'opére pas tous les jours de la semaine
Alors la liste des jours de la semaine pour lesquels ce vol opère doit être fournie.

Le serveur pourrait fournir la valeur d'autres attributs en fonction de ses connaissances sur les préoccupations de l'utilisateur. Par exemple, suivant que l'utilisateur voyage pour affaires ou pour des raisons touristiques, il peut en déduire que l'utilisateur est intéressé par les prix, et plus généralement par toute information concernant le *thème* "Argent", ou bien qu'il n'est pas intéressé par cette information. Ainsi, le serveur pourrait appliquer la règle :

RT6 : **Si** la requête concerne les Vols,
Et l'utilisateur veut faire un voyage touristique,
Alors les attributs concernant le *thème* Argent doivent être insérés dans la liste des attributs à fournir.

ce qui conduit à transformer la requête QT1 en QT5 :

QT5 : Pour les Vols
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = New-York
Heure-de-départ appartient à [8 00 , 12 00]
Fournir les valeurs des attributs : Heure-de-départ, Prix.

conduisant à la réponse AT5 :

AT5 :

Vol	Heure-de-départ	Prix
AF001	10h00	27.715 FF
AF001	11h00	27.715 FF
AF015	10h30	8.825 FF

Le prix du vol AF001 est approximativement trois fois celui du vol AF015. Il s'agit en effet d'un vol Concorde. Même si l'utilisateur voyage pour affaires, il est peu fréquent

de prendre le Concorde, et l'utilisateur serait probablement intéressé par le prix d'un vol si ce prix est très élevé. Le serveur pourrait donc appliquer la règle suivante pour n'importe quel utilisateur :

RT7 : **Si** une requête concerne les Vols,
Et le prix du vol est très élevé,
Alors la valeur de l'attribut Prix doit être fournie.

la réponse étant alors :

AT6 :

Vol	Heure-de-départ	Prix
AF001	10h00	27.715 FF
AF001	11h00	27.715 FF
AF015	10h30	

En fait, on peut considérer que les règles RT4, RT5 et RT7 sont des cas particuliers de règles générales :

Si une requête concerne un type d'entité donné,
Et la valeur d'un attribut de cette entité diffère de la valeur la plus fréquente de cet attribut,
Alors la valeur de cet attribut doit être fournie

D'après les exemples précédents, on peut conclure que, dans certains cas, la valeur d'un attribut ne doit être fournie que pour certains tuples satisfaisant des conditions particulières. Ces attributs seront appelés par la suite **"Attributs-Conditionnels"**.

De plus, les informations supplémentaires fournies dans ces exemples concernent le même ensemble d'entités que la requête initiale, et pour une requête ayant la forme suivante :

"Pour les entités d'un **type** donné,
satisfaisant certaines **conditions**,
fournir certains **attributs**."

la requête transformée a la forme :

"Pour les entités d'un **type** donné,
satisfaisant certaines **conditions**,
fournir certains **attributs**
et certains **attributs supplémentaires**
et certains **attributs conditionnels**."

On peut maintenant conclure qu'une seconde façon de définir des informations pertinentes supplémentaires est de transformer la requête initiale en une autre requête qui caractérise le même ensemble d'entités et qui fournit la valeur d'autres attributs que ceux explicitement demandés par l'utilisateur. Ce type de transformation est appelé par la suite **"transformation des informations demandées"**.

1.3 Transformation du sujet

Nous allons maintenant étudier une troisième façon de transformer les requêtes dans laquelle des entités d'un autre type que celui apparaissant dans la requête peuvent être fournies. Les autres types peuvent être définis en utilisant une structure prédéfinie pour les types d'entités. Cette structure hiérarchique pourrait être connue d'un serveur ayant une certaine expérience de la coopération avec les usagers. La façon dont est définie la hiérarchie conditionne fortement les autres types d'entités qui seront fournies à l'utilisateur.

Supposons que nous ayons la structure suivante pour les bases de données :

Exemple financier :

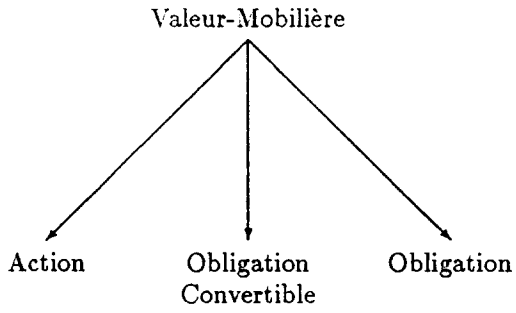
Type d'entité : Valeur-mobilière
Attributs : Dernier-cours
Cours
Dernier-coupon :
Date
Montant-global
Variation
Rendement
Nationalité

Type d'entité : Action
Attribut : Secteur-d'activité
Opération-en-cours

Type d'entité : Obligation convertible
Attribut : Taux
Date-de-conversion
Taux-de-conversion
Date-d'échéance
Secteur-d'activité

Type d'entité : Obligation
Attribut : Taux
Date-d'échéance
Emetteur

La structure hiérarchique sur les types d'entités est représentée par le graphe :



On suppose que les types Action, Obligation-Convertible et Obligation héritent des attributs définis pour le type Valeur-mobilière.

Exemple "Agence de voyage" :

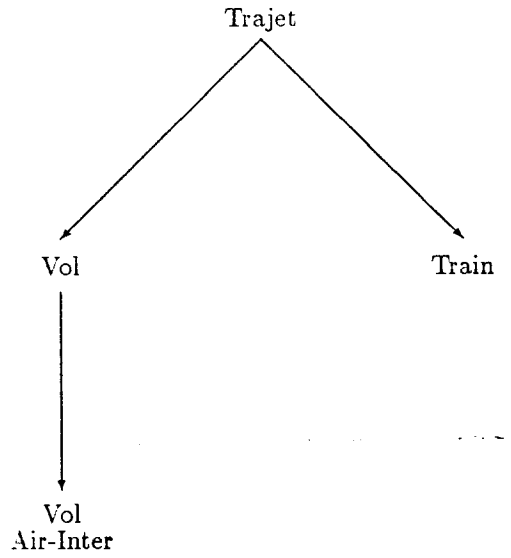
Type d'entité : Trajet
Attributs : Ville-de-départ
Ville-d'arrivée
Heure-de-départ
Heure-d'arrivée
Prix

Type d'entité : Vol
Attribut : Période
Opère

Type d'entité : Vol-Air-Inter
Attribut : Tarification

Type d'entité : Train
Attribut : Via

La structure hiérarchique sur les entités est représentée par le graphe suivant:



Les entités ayant le même père dans la hiérarchie sont appelées entités voisines.

Le notion d'entités voisines pourrait permettre au serveur, dans le cas de la requête QS1, de proposer des obligations convertibles ou des obligations en plus des actions. En général, on ne peut pas toujours remplacer dans une requête un type d'entité donné par un autre type. Cela dépend de l'expérience du serveur et peut être représenté par des règles. De plus, des problèmes non triviaux peuvent surgir lorsqu'un attribut apparaissant dans une requête n'est pas défini pour l'entité voisine. Par exemple, dans la partie condition de QS1 apparait l'attribut Secteur-d'activité, et cet attribut est défini pour les obligations convertibles, mais pas pour les obligations.

Ainsi on pourrait avoir la règle :

RS6 : **Si** une requête concerne les Actions,
Et les attributs apparaissant dans la requête sont définis pour les Obligations Convertibles,
Alors les entités de type Action peuvent être remplacées par celles de type Obligation Convertible.

En réalité, cette règle est probablement trop générale et certaines conditions concernant les intentions de l'utilisateur devraient être ajoutées à cette règle. Ce type de condition est utilisé dans la règle suivante :

RS7 : **Si** une requête concerne les Obligations Convertibles,
Et l'utilisateur est intéressé par les réductions fiscales,
Alors les entités de type Obligation Convertible peuvent être remplacées par des Actions.

Dans l'exemple de QT1, la règle suivante serait visiblement trop générale :

RT8 : **Si** la requête concerne les Vols,
Et les attributs apparaissant dans la requête sont définis pour les Trains,
Alors les entités de type Vol peuvent être remplacées par celles de type Train.

La requête transformée pourrait être stupide puisqu'il n'y a pas de train entre Paris et New-York. Toutefois, si l'utilisateur pose une requête concernant les Vols sur Paris-Bruxelles alors il est pertinent de proposer un déplacement en train plutôt qu'en avion. On pourrait donc avoir la règle suivante :

RT9 : **Si** une requête concerne les Vols,
Et il y a une condition définissant des vols entre deux villes dont la distance n'excède pas 400 km,
Alors les entités de type Vol peuvent être remplacées par celles de type Train.

En appliquant cette règle à la requête QT5 :

QT5 : Pour les Vols
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = Bruxelles
Heure-de-départ appartient à [8 00, 12 00]
Fournir la valeur des attributs : Heure-de-départ.

on obtient la requête transformée QT6 :

QT6 : Pour les Vols ou les Trains
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = Bruxelles
Heure-de-départ appartient à [8 00, 12 00]
Fournir la valeur des attributs : Heure-de-départ.

En fait, la requête QT6 pourrait être transformée de nouveau pour insérer dans la liste des attributs à fournir l'Heure-d'arrivée des trains. Ceci pourrait être réalisé avec une règle similaire à celles utilisées pour transformer les attributs à fournir.

Cette structure hiérarchique des types d'entités permet également de définir des règles pour un type donné puis de les appliquer pour un sous-type. Par exemple, plutôt que la règle RT6, on aurait pu définir la règle RT10 :

RT10 : **Si** une requête concerne les Trajets,
Et l'utilisateur veut faire un voyage touristique,
Alors les attributs concernant le concept d'Argent peuvent être insérés à la liste des attributs à fournir.

Cette règle peut alors être appliquée à la requête QT1, en utilisant le mécanisme d'héritage.

Réciproquement, si une règle concerne un type d'entité donné, et s'il existe une régle concernant un sous-type, alors la règle peut être appliquée, mais le processus de transformation doit insérer dans la requête une condition qui restreint les conséquences de la transformation aux entités appartenant au sous-type. Ceci peut être réalisé avec les attributs conditionnels.

On peut conclure qu'il existe une troisième façon de transformer les requêtes, dans laquelle le type des entités à fournir peut être modifié, et qui sera appelée par la suite **"transformation du sujet"**. Cette appellation est justifiée par le fait que, souvent, la définition du type des entités dans la requête correspond au sujet d'une phrase en langue naturelle.

Arrivé à ce point de cette présentation informelle, il faut remarquer qu'il pourrait être intéressant de définir, en plus de la relation de généralisation présentée ci-dessus, d'autres abstractions entre les types d'entités telles que l'agrégation et l'association [6,7].

La relation d'agrégation permet de définir un type d'entité portefeuille, ayant trois composants : < Ensemble-d'Actions, Ensemble-d'Obligations, Or>, où une instance du type d'entité : Ensemble-d'Actions, est un ensemble d'instances d'Action, et une instance du type d'entité : Ensemble-d'Obligations, est un ensemble d'instances d'Obligation.

La relation d'association permet de représenter un lien entre les types d'entités Ensemble-d'Actions et Action, ou entre Ensemble-d'Obligations et Obligation.

Chapitre 2

Etude bibliographique

Dans la partie précédente, nous avons présenté de façon très intuitive et très informelle le problème des "réponses coopératives" tel que nous envisageons de l'étudier dans la suite de ce travail. Pour résumer, nous avons distingué trois manières de transformer une requête pour fournir des informations supplémentaires pertinentes, à savoir :

- Transformation des conditions
- Transformation des informations demandées
- Transformation du sujet

Dans cette partie, nous allons montrer comment ce problème de la coopération a été abordé dans la littérature, en essayant de dégager les analogies et les différences avec notre approche.

En fait, ce problème a surtout été étudié dans le cas d'une base de données que l'on peut interroger en langage naturel. Dans ce cadre, le problème consiste à fournir à l'utilisateur une réponse qui ressemble à celle que donnerait un être humain ayant connaissance de la base de données.

Plusieurs systèmes ont ainsi été réalisés dont le but commun est de rendre plus convivial un système de gestion de base de données. Derrière ce but commun, on peut distinguer au moins quatre mécanismes différents qui peuvent chacun être appliqués pour établir une certaine forme de coopération entre la base de données et l'utilisateur. Nous essayons de les présenter et de montrer comment ils ont été traités dans la littérature.

2.1 Réponses indirectes

On présente ici le problème qui consiste à fournir une réponse pertinente lorsqu'une requête à une base de données échoue (c'est-à-dire retourne comme réponse l'ensemble vide). Plusieurs chercheurs ont déjà abordé ce sujet tels Kaplan [42,43], Janas [39,40].

Dans toutes ces études, le point de départ est que, dans certains cas, on ne peut pas donner une réponse pertinente directe à une requête telle que :

"Trouver tous les x ayant la propriété y"

lorsqu'il n'y a pas de x dans la base de données qui satisfont la condition y.

Ainsi, bien que la réponse directe "Il n'y a pas de tel x" soit une réponse correcte dans cette situation, celle-ci n'est généralement pas coopérative pour l'utilisateur comme on peut s'en rendre compte dans l'exemple suivant [42] :

Q1 : Quels sont les étudiants qui ont obtenu la licence de langue au printemps 1979 ?

R : NIL. (ensemble vide)

Q2 : Est-ce que quelqu'un a échoué à la licence de langue au printemps 1979 ?

R : Non.

Q3 : Combien de personnes ont passé la licence de langue au printemps 1979 ?

R : Zéro.

Q4 : Est-ce que le cours de licence de langue était assuré en 1979 ?

R : Non.

Ce manque de coopération peut conduire l'utilisateur à une mauvaise interprétention de la réponse. Ainsi, dans l'exemple ci-dessus, si l'on en reste à la question Q1 et à sa réponse, l'utilisateur va peut-être conclure que l'examen était particulièrement difficile cette année là.

En fait, le système aurait dû fournir à Q1 la réponse suivante :

R : Le cours de licence de langue n'était pas assuré en 1979.

L'étude d'une interface coopérative passe dans ce cas par l'examen des erreurs commises dans l'expression des questions. Ainsi, la question suivante :

(1) *Qui est le 39 ième président des Etats-Unis ?*

autorise le serveur à tirer la conclusion suivante sur la croyance de l'utilisateur :

(2) *Il y a un 39 ième président des Etat-Unis.*

(2) doit être vrai pour n'importe quelle réponse directe. S'il n'y avait pas de 39 ième président, alors aucun nom ne pourrait être correct. Puisque (2) est une précondition pour que toute réponse à la question ait un sens, elle est appelée présupposition.

Dans COOP [42], Kaplan étudie le problème consistant à répondre correctement à une requête, même si celle-ci fait une présupposition erronée.

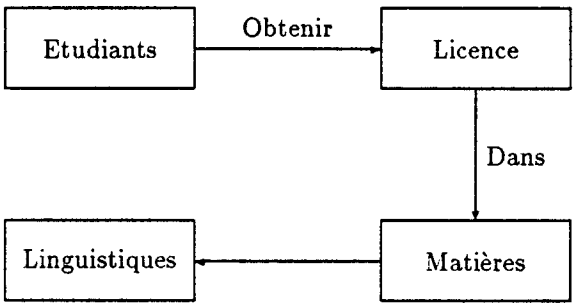
COOP est une interface qui accepte une question exprimée en langage naturel, la comprend, puis la transforme en une structure MQL (Meta Query Language).

Le MQL est une structure de graphe, où les nœuds représentent les ensembles présentés par l'utilisateur dans la question et les arcs des relations binaires définies sur ces ensembles.

Ainsi la question suivante :

Q : Quels sont les étudiants qui ont obtenu la licence dans les matières linguistiques ?

est représentée par le graphe suivant :



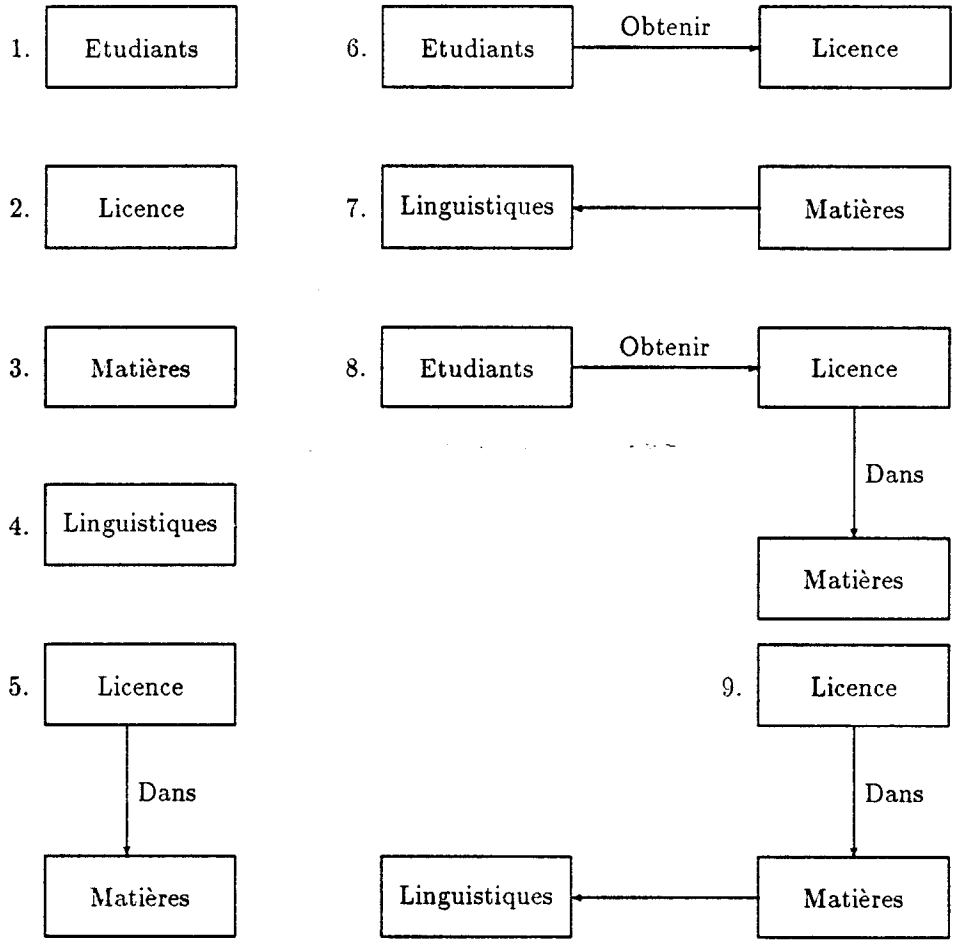
On distingue quatre ensembles :

- Etudiants
- Licence
- Matières
- Linguistique

Chaque sous-graphe de la représentation MQL représente une sous-question bien définie. Pour que la question ci-dessus ait une réponse non vide il faut que chaque sous-question ait une réponse non vide.

Par suite, si la requête initiale retourne une réponse vide, le générateur de réponses indirectes est appelé. Celui-ci parcourt le graphe de la question pour établir la liste des présuppositions afin de tester celles qui échouent.

Par exemple pour la requête ci-dessus, les sous-graphes et leurs réponses indirectes correspondantes sont les suivants :



Supposons qu'il n'y ait pas de matière linguistique, le sous-graphe 7 implique une réponse vide. On fournit alors à l'utilisateur la réponse indirecte suivante :

"Je ne connais pas de matière linguistique"

Les présuppositions peuvent être partiellement ordonnées suivant les relations d'implication : l'échec de certaines présuppositions peut entrainer l'échec d'autres présuppositions. Ainsi l'échec de 7 entraine l'échec de 9. Dans ce cas le système fournit comme réponse l'ensemble des plus petites réponses indirectes ayant échoué.

Par exemple si l'on suppose qu'il n'y a ni matière linguistique ni licence, la réponse pourrait être :

"Je ne connais pas de matière linguistique et je ne connais pas la licence"

et par suite il n'y a pas de licence dans les matières linguistiques (échec de 8) et aucun élève n'a obtenu la licence (échec de 9).

La génération des réponses indirectes est un problème intéressant mais qui a déjà été abondamment étudié. Ainsi, depuis les premiers travaux assez informels de Kaplan, Janas dans [39,40] a proposé une étude plus théorique utilisant le formalisme du calcul des prédicats. Depuis, des améliorations à ces méthodes ont été proposées dans [37] et [28].

2.2 Réponses suggestives

Dans COOP, lorsqu'il n'y a pas de réponse à la requête initiale alors qu'aucune présupposition n'échoue, le système essaye de générer une "réponse suggestive indirecte".

En l'occurrence, une nouvelle question est rédigée à partir de la question initiale. Celle-ci élargit le but de la question. Tout le problème consiste alors à déterminer correctement le but de la question.

Kaplan propose deux heuristiques pour y parvenir.

Ainsi considérons l'exemple suivant :

Q1 : Quelle est la longueur des sous-marins américains dans l'Atlantique ?

R1 : NIL

Q2 : Quelle est la longueur des sous-marins américains en Arctique ?

R2 : Je ne connais pas la longueur des sous-marins américains en Atlantique ou en Arctique, mais vous êtes peut-être intéressé par la longueur des sous-marins américains partout ailleurs ?

COOP commence par appliquer une première heuristique basée sur la syntaxe : on choisit dans le graphe de représentation de la requête, l'ensemble le plus éloigné de la racine. Cet ensemble, une fois sélectionné, COOP se contente de l'éliminer. Cette heuristique appliquée à Q1 permet de générer deux requêtes :

$Q1_a$: Quelle est la longueur des sous-marins dans l'Atlantique ?

$Q1_b$: Quelle est la longueur des sous-marins américains partout ailleurs ?

Comme cette première heuristique ne permet pas de générer une seule requête, COOP considère que le but de la question Q1 n'est pas déterminé, et ne fournit donc pas de réponse suggestive à la question Q1.

Lorsque l'utilisateur pose la question Q2, COOP tente alors d'appliquer une deuxième heuristique : on sélectionne les sous-ensembles de noeuds dans Q2 n'appartenant pas à Q1. Ceci permet de remarquer que la longueur des sous-marins américains constitue le centre d'intérêt dans l'exemple ci-dessus. On obtient ainsi la requête $Q1_b$ (mais plus $Q1_a$) et COOP propose de répondre à cette question.

Dans la démarche suivie par Kaplan pour générer des réponses suggestives, il est intéressant de remarquer l'utilisation de la gestion du dialogue. Ainsi, lorsque l'utilisateur pose la question Q2, COOP tient compte de la question Q1 déjà posée pour déduire que la longueur des sous-marins américains constitue le centre d'intérêt de l'utilisateur lorsqu'il pose ces deux questions.

On peut hélas déplorer que COOP se contente alors d'éliminer certaines parties de la requête posée par l'utilisateur. En fait, l'idéal serait de fournir des réponses suggestives intelligentes comme dans l'exemple suivant :

Q3 : Combien de trains Paris-Toulouse y a-t-il ce soir ?

R3 : Aucun, mais il y en a trois demain matin.

C'est cette démarche qu'essaye de formaliser A. Motro dans [47] et [48]. Celui-ci considère que certaines requêtes soumises à une base de données sont en fait des buts caractérisés par une cible et toute donnée "proche" de cette cible peut être une réponse intéressante pour l'utilisateur. Par exemple, considérons la requête suivante :

Q : Quels sont les restaurants français bon marché dans le sud de la ville ?

S'il n'y a pas de réponse à cette requête, un restaurant européen à prix modérés à proximité du sud de la ville peut faire l'affaire.

Pour prendre en compte ces notions de proximité, Motro introduit un concept de distance entre les valeurs d'un domaine, puis définit trois types de buts : les buts **voisins**, les buts **optimaux** et les buts **prioritaires**.

Dans son formalisme, Motro se place dans le cadre du modèle relationnel et ne considère que des relations ayant une clé unique. On peut en fait considérer que cette clé représente le nom d'une entité et les autres attributs fournissent une description de l'entité.

Par suite le schéma d'une relation est une séquence de noms d'attributs, l'un d'entre eux étant la clé. Le schéma de la base de données est alors un ensemble de schémas de relations. Chaque attribut est défini à l'aide des paramètres suivants :

Mesure : le nom de la mesure qui permet de calculer la distance entre les valeurs de cet attribut. Il s'agit soit du nom d'une autre relation soit du nom d'une fonction distance.

Facteur d'échelle : un nombre permettant de corriger les distances entre les valeurs de cet attribut après calcul par la mesure.

Poids relatif : un nombre indiquant l'importance relative de l'attribut lors du calcul de la distance entre les descriptions de deux entités.

Rayon de voisinage : un nombre utilisé pour établir le domaine de voisinage de la distance autour de la valeur de cet attribut.

La définition formelle de la distance entre x et y avec la mesure M, notée $d_M(x,y)$ est la suivante :

1. Si M est une fonction distance alors celle-ci est calculée. Par exemple on peut définir la fonction $M = Number$ de la façon suivante :

$$d_M(x,y) = d_{Number}(x,y) = \mid x - y \mid .$$

2. Si M n'est pas une fonction distance, c'est une relation et la distance est définie de la façon suivante :

   - Soit M une relation ayant pour attributs $A_0, A_1, ..., A_n$ et soit A_0 la clé. Pour $i = 1, .., n$ soit M_i, s_i et w_i respectivement la mesure, le facteur d'échelle et le poids relatif de l'attribut A_i et soit $w = w_1 + w_2 + ... + w_n$. Soient x et y les valeurs de l'attribut A_0 et $x_1, ..., x_n$ et $y_1, ..., y_n$ les valeurs respectives des autres attributs. On a alors :

$$d_M(x,y) = \sum_{i=1}^{n} d_{M_i}(x_i, y_i) \times (1/s_i) \times (w_i/w)$$

3. Si x et y ne sont pas des valeurs des domaines alors on définit :

$$d_M(x,y) = \infty$$

Une fois la notion de distance définie, on peut étendre le calcul relationnel en introduisant un nouvel opérateur de comparaison, appelé similaire-à, noté $\sim$, défini de la façon suivante :

$$x \sim y \text{ si } d_M(x,y) \times (1/s) \leq r$$

où s est le facteur d'échelle et r le rayon de voisinage.

Un but **voisin** est une requête incorporant des opérateurs similaire-à. Une requête "précise" peut alors être transformée en remplaçant l'opérateur = par similaire-à. Comme la réponse à un but voisin contient toujours la réponse à la requête précise dont elle est dérivée, le but voisin est une requête plus générale que le requête précise.

Dans le but de fournir une réponse qui soit bien adaptée au besoin de l'utilisateur, Motro définit deux nouvelles notions : les buts optimaux et les buts prioritaires.

Soit Q une requête relationnelle et soit $(x_i \sim y_i), i = 1, ..., k$ les formules atomiques de Q dans lesquelles un opérateur similaire-à apparait. Supposons que x_i et y_i aient une mesure commune M_i, un facteur d'échelle commun s_i et un rayon de voisinage r_i. Soit T la réponse à cette requête.

Le but **optimal** Q_{op} est l'ensemble des tuples qui minimisent la somme :

$$\sum_{i=1}^{n} d_{M_i}(x_i, y_i) \times (1/s_i)$$

Le but **prioritaire** est obtenu en appliquant le processus suivant. Soit T_0 l'ensemble des réponses à la requête Q. Dans T_0, on choisit les solutions qui minimisent $d_{M_1}(x_1, y_1)$, soit T_1 l'ensemble ainsi obtenu. On construit ensuite T_2 en choisissant les solutions de T_1 qui minimise $d_{M_2}(x_2, y_2)$ et on itère le processus jusqu'à l'obtention de T_n. Si à une étape, T_i est vide, on redéfinit T_{i-1}. Dans les valeurs de T_{i-2} qui avaient été écartées à l'étape précédente, on choisit celles qui minimisent $d_{M_{i-1}}(x_{i-1}, y_{i-1})$, on les ajoutent à T_{i-1} et on reprend le processus.

Les notions de buts optimaux et prioritaires permettent d'orienter la recherche pour fournir de "bonnes" réponses à l'utilisateur.

On peut trouver plusieurs limites à cette représentation de la notion de voisinage à l'aide d'une distance.

D'abord, on peut estimer que la définition de ces distances est, en grande partie, arbitraire, en particulier la notion de poids relatif d'un attribut. En effet, l'importance accordée à un attribut lors du calcul des solutions voisines dépend de chaque utilisateur, et ceci ne peut pas être pris en compte en associant un poids statique, défini une fois pour toute, à chaque attribut.

Ensuite, cette notion de distance ne permet pas de prendre en compte les négations. Ainsi, considérons la requête suivante :

$$Restaurant(x) \wedge Type(x, y) \wedge \neg(y = Chinois)$$

c'est-à-dire :

Q : Quels sont les restaurants qui ne sont pas Chinois ?

Si maintenant on transforme cette requête en remplaçant l'opérateur $=$ par $\sim$, alors on obtient la requête suivante :

$$Restaurant(x) \wedge Type(x, y) \wedge \neg(y \sim Chinois)$$

Si l'on considère qu'un restaurant Japonnais est voisin d'un restaurant Chinois, alors la réponse à cette nouvelle requête ne contient pas la réponse à la requête initiale, puisque tous les restaurants qui sont voisins d'un restaurant Chinois ne sont plus solutions.

Enfin, dans son étude, Motro se limite au cas où l'on fournit une réponse suggestive lorsqu'une requête échoue. Or, ce seul cas semble très limitatif. Par exemple, dans [60], un problème similaire est étudié mais la motivation est différente : on ne fournit pas des réponses "voisines" à l'utilisateur dans le seul cas où la requête échoue, mais également lorsque le serveur peut indiquer une alternative intéressante par rapport à la valeur d'un critère donné choisi pour comparer les entités. Ainsi, dans l'exemple présenté dans cet article, un employé d'une agence de voyage propose à l'utilisateur de modifier la date de son départ lorsqu'il peut proposer des vols moins chers comme alternative aux réponses exactes. Toutefois, l'auteur ne propose ni formalisation ni méthode générale pouvant être utilisés pour d'autres domaines d'application. Dans le chapitre 8, nous

montrons comment nous traitons le même genre de problèmes. En particulier, nous verrons qu'une représentation logique des connaissances que nous utilisons permet de changer facilement de domaine d'application, car il y a une séparation nette entre les règles qui sont dépendantes du domaine d'application de celles qui ne le sont pas.

2.3 Réponses complétives

En plus des réponses indirectes et suggestives, Kaplan propose une troisième forme de réponse coopérative : la réponse indirecte complétive. Il s'agit ici de fournir à l'utilisateur une réponse qui apporte plus de renseignements que ceux explicitement demandés, mais ceci dans des cas bien particuliers tels que le suivant :

Question : Quels sont les numéros de téléphone des stagiaires de thèse du DERI ?

Réponses courantes d'une base de données :

Téléphone
61 55 74 40
61 55 70 48
...

Ces réponses ne sont d'aucune utilité puisque nous ne connaissons pas les personnes à qui appartiennent les numéros de téléphone. Il est, en effet utile de connaître les noms des personnes si l'on désire les contacter.

Un exemple de réponse complétive pourrait être dans ce cas :

Téléphone	Nom	Prénom
61 55 74 40	Ostermann	Pascal
61 55 70 48	Thibault	Xavier
...		

Pour obtenir de telles réponses, Kaplan suppose que si l'utilisateur mentionne explicitement certains aspects du domaine dans sa question, alors cet aspect peut apparaître dans la réponse. Or à chaque aspect mentionné correspond un ensemble dans MQL. Ainsi la question ci-dessus met en évidence trois ensembles : Numéro de téléphone, Stagiaire de thèse et DERI. La stratégie est alors de fournir des informations sur chaque ensemble qui ne soit pas un singleton (DERI est un singleton).

Dans le lexique, sont associés aux concepts les attributs de la BD susceptibles d'être communiqués si le concept est mentionné dans la question : au concept "stagiaire de thèse", on associe donc les attributs nom et prénom.

Un problème similaire existe pour la question suivante :

Question : Quels sont les vols qui partent de Paris et qui arrivent à New-York ou *Washington ?*

Réponses courantes d'une base de données :

Vol
AF001
AF015
AF020
...

Cette réponse n'est pas très coopérative dans la mesure où l'utilisateur aimerait très probablement distinguer parmi les solutions les vols qui arrivent à New-York de ceux qui arrivent à Washington. Bien sûr, l'utilisateur peut reformuler sa question en indiquant qu'il veut également connaître dans la réponse la ville d'arrivée des vols solutions. Toutefois, il semble que la façon naturelle pour l'utilisateur de formuler sa question soit bien celle ci-dessus. C'est le système qui doit être suffisamment convivial pour fournir une réponse complétive qui pourrait être :

Vol	Ville-d'arrivée
AF001	New-York
AF015	New-York
AF020	Washington
...	

Parmi les travaux qui se rapprochent de ce genre de problèmes on peut citer Wahlster et *al.* dans [62] dont le problème est de fournir des informations complémentaires qui anticipent des questions futures. Par exemple, à la question :

Question : Combien de camions se sont arrêtés au carrefour depuis une heure ?

on pourra répondre :

Réponse : Deux, Un Ford et un Renault.

Toutefois, parmi tous les travaux existant sur le problème des réponses complétives, nous n'en connaissons aucun qui tente une approche formelle.

Dans notre étude, nous commençons par préciser cette notion de réponse complétive, avant de traiter ce problème comme un cas particulier de transformation des informations demandées. Ce traitement est présenté en détail dans la partie 7.3.

2.4 Interventions pertinentes

Nous présentons maintenant un modèle de comportement coopératif très sensiblement différent des trois premiers types de coopération présentés précédemment. Ce modèle de comportement coopératif repose essentiellement sur les notions d'actions et

de plans. Ainsi, on suppose que lorsqu'un acteur A pose une question à un autre acteur B, ce dernier essaye de reconnaître le plan et le but de A, puis d'utiliser ce plan et ce but pour formuler des réponses à la question de A. En particulier, on montre comment ce modèle peut être utilisé pour fournir plus d'informations que celles explicitement demandées par l'utilisateur et donner des réponses correctes à des phrases incomplètes et des actes de langage indirects. C'est la démarche suivie par Allen, Frish, Litman et Perrault dans [2,3,1], ainsi que par De dans [25].

Cette démarche s'inspire de la théorie philosophique de l'intention dévellopés par H. Grice [34,35]. Grice remarque que l'auditoire est supposé tirer un certain nombre de conclusions sur les intentions du locuteur. Ainsi, il considère que, si un locuteur S communique une intention à un auditoire A, alors les intentions suivantes doivent être présentes de la part de S :

(1) : A fournit une réponse, une réaction ou un effet r.

(2) : A reconnait (1).

(3) : (2) est une raison pour que A fournisse r.

(4) : A reconnait (3).

A "comprend" l'intention de S si les intentions (2) et (4) sont satisfaites.

En philosophie, on distingue en général dans le fait d'ennoncer une phrase plusieurs aspects :

Acte locutoire : le fait de produire le son.

Acte illocutoire : l'action réalisée par ce qu'on vient de dire.

Acte perlocutoire : l'effet produit sur l'auditoire par le fait de réaliser cet acte. En général, le locuteur veut que l'auditoire entende le message et reconnaisse quel le locuteur réalise (acte illocutoire).

Par exemple, en disant *"Le taureau va charger !"*, l'acte illocutoire consiste à prévenir l'auditoire et l'acte perlocutoire est la volonté de voir l'auditoire se sauver.

J. Searle [57] présente une formulation de la structure des actes illocutoires en suggérant un nombre de conditions nécessaires et suffisantes pour qu'ils se réalisent correctement. Le formalisme présenté ici s'inspire fortement de ces travaux.

2.4.1 Introduction

Un bon système de questions-réponses a souvent besoin de fournir des réponses contenant plus d'informations que celles explicitement demandées dans la question. Toutefois, il ne faut pas fournir trop d'informations ni fournir des informations qui ne sont pas utiles pour l'utilisateur. Par exemple, considérons ce petit dialogue ayant lieu dans une gare :

(1.1) A : *A quelle heure est le départ du train pour Montréal ?*

(1.2) B : *3h15 quai numéro 7.*

Bien que le lieu du départ ne soit pas explicitement demandé, *B* le fournit en réponse.

Dans cette approche, on fait les hypothèses suivantes :

- Les gens ont des comportements rationnels et sont capables de construire des plans pour atteindre leur but.
- Ils sont capables de déduire un plan en voyant un autre acteur réalisant une action.
- Ils sont capables de détecter les obstacles présents dans les plans d'un autre acteur.

Les obstacles sont les buts d'un plan qui ne peuvent être atteints (facilement) sans aide.

Une forme de comportement coopératif survient lorsqu'un acteur observe un obstacle dans le plan d'un autre acteur. Le plan pour atteindre ce but fera intervenir des actions coopératives.

Le langage peut être vu comme un cas particulier de comportement orienté par les buts. Les phrases sont produites par les actions (actes de langage) et sont énoncées pour avoir un effet sur l'auditeur. L'effet peut en particulier modifier les croyances et les buts de l'auditeur. Un acte de langage, comme toute action, peut être observé par l'auditeur et permet à l'auditeur de déduire quel est le plan du demandeur. Souvent un acte de langage peut présenter explicitement un but qui est un obstacle. Par exemple, la phrase (1.1) montre à l'auditeur que le demandeur a besoin de savoir l'heure de départ du train. Mais, il peut y avoir dans le plan d'autres obstacles qui n'apparaissent pas explicitement. Une réponse coopérative essaiera de résoudre ces obstacles implicites aussi bien que les obstacles explicites.

Le modèle présenté offre la possibilité de prendre en compte certains aspects intéressants du langage, tels que :

- La génération de réponses qui fournissent plus d'informations que demandé (comme dans l'exemple ci-dessus).
- La génération de réponses à des phrases incomplètes.
- L'analyse des actes de langage indirect.

Etudions chacun de ces trois points :

- Il est intuitivement assez simple de voir comment le modèle permet de fournir plus d'informations qu'il n'est explicitement demandé. Dans le domaine des trains, le serveur suppose que l'utilisateur a certains buts tels que prendre ou attendre l'arrivée d'un train. La requête concernant l'heure de départ du train (1.1) montre

que le but de l'utilisateur est de prendre un train. De plus, en supposant que le serveur croit que l'utilisateur ne connait pas le lieu de départ du train, il croit donc que la non connaissance de ce lieu est un obstacle dans le plan. Il fournit une réponse qui résoud cet obstacle (1.2).

- Un deuxième problème survient lorsque les personnes communiquent en utilisant des phrases incomplètes. Par exemple :

  (2.1) A : *Le train de 3h15 pour Windsor ?*
  (2.2) B : *Quai numéro 10.*

  Avec le modèle ci-dessus, l'information contenue dans le fragment de phrase (2.1) est suffisante pour déduire quel est le plan de l'utilisateur. Ainsi, il peut fournir une réponse raisonnable fondée sur les éventuels obstacles du plan. Ainsi un obstacle dans le plan est la non connaissance du lieu de départ, d'où la réponse (2.2).

- Le troisième problème apparait dans l'exemple suivant :

  (3.1) A : *Connaissez-vous l'heure de départ du train pour Windsor ?*

  Syntaxiquement, il s'agit d'une question oui-non. Toutefois, en faisant quelques petites modifications dans le modèle de détection d'obstacles ci-dessus, une réponse coopérative correcte peut également être formulée.

Dans ce qui suit, nous commençons, dans la section 2.4.2, par présenter une description assez informelle du système élaboré par Allen et *al.* et présenté dans [3].

Dans la section 2.4.3, nous présentons un exemple de réponses coopératives emprunté au domaine des trains.

2.4.2 Vue d'ensemble du modèle

Commençons par une description intuitive de ce qui se passe lorsqu'un acteur A pose une question à un acteur B qui ensuite lui répond. A a pour but d'acquérir certaines informations : il construit donc un plan pour poser à B une question dont la réponse fournira cette information. B recevant cette question essaiera d'inférer le plan de A. Dans ce plan il peut y avoir des obstacles que A ne peut résoudre sans assistance. B peut alors accepter ces obstacles comme ses propres buts et construire un plan qui les résoud. La réponse de B est générée lorsqu'il exécute le plan.

Dans cette section, nous commençons par présenter rapidement les diverses logiques modales développées pour représenter la croyance, la connaissance et la volonté d'un acteur. Nous présentons ensuite les règles qui permettent :

1. A un acteur X, de construire un plan.
2. A un acteur S, de déduire le plan qu'un autre acteur A veut réaliser.

Enfin, une fois que S a déduit le plan de l'utilisateur, nous étudions comment S peut avoir un comportement coopératif.

Croyance, connaissance, et vouloir

Le traitement de la croyance est en fait identique à celle de Hintikka [38]. On peut également regarder la représentation de Cohen [16] et celle de Moore [46].

La croyance d'un acteur A est représentée en utilisant un opérateur modal *Croit* :

$$Croit_A(P) = A \text{ croit que } P \text{ est vrai.}$$

On définit comme abréviations :

$$Connait_A P = P \wedge Croit_A(P)$$

ainsi que :

$$Connait\text{-}si_A P = (Connait_A P) \vee (Connait_A(\neg P))$$
(A connait si une proposition P est vraie).

et également :

$$Connait\text{-}ref_A D(x) = \exists!(y)((D(x) = y) \wedge Croit_A(D(x) = y))$$
(par exemple l'acteur A connait le coût de x si x a un coût unique y, et si l'acteur A croit que y est le coût unique de x).

Les actions et les plans d'un acteur sont représentés en utilisant l'opérateur modal *Veut* :

$$Veut_A(P) = A \text{ a l'intention d'atteindre } P.$$

Dans la suite, on utilisera également CROIRE (A,P), SAVOIR (A,P) et VOULOIR (A,P) comme notation équivalente à $Croit_A(P)$, $Connait_A(P)$ et $Veut_A(P)$.

On dira qu'une action est intentionnelle si chaque fois que l'action est réalisée, l'acteur désirait sa réalisation à ce moment. Ainsi si ACT est une action intentionnelle, A un acteur, t le temps, alors :

- $ACT(A) < t > \longrightarrow Veut_A(ACT(A) < t >)$
  où $ACT(A) < t >$ est un prédicat qui n'est vrai que si l'action ACT est exécutée par A à l'instant t.

Il y a donc une précondition sur chaque action intentionnelle : l'acteur désire la réalisation de cette action.

Construction de plans

Plusieurs systèmes de résolutions de problèmes fournissent une formulation des actions et des plans [30,63]. Dans ces systèmes, le monde est représenté par un ensemble de propositions. Ce monde est modifié par des actions, une action étant décrite par des préconditions (des conditions qui doivent être vérifiées avant l'exécution de l'action) et par des effets (la modification que l'action produit sur le monde). Etant donné un état initial W et un état final G, un plan est une séquence d'actions qui transforme W en G.

Austin [4] suggère que toute phrase est le résultat de plusieurs actions ou actes de langage. On s'intéresse surtout aux actes de langage tels que Demander, Avertir, Affirmer ou Promettre. Cohen [16] montre que des actes de langage tels que Demander et Informer peuvent être modélisés comme des actions dans un système de planification.

Nous ne revenons pas en détail sur les méthodes conçues pour construire un plan permettant d'atteindre un but donné. On peut trouver dans [31] une présentation détaillée de ce problème.

Rappelons seulement qu'un plan est constitué par une succession d'actions permettant d'atteindre le but désiré. Une action est composée d'un nom et d'un ensemble de formules appartenant à l'une des classes suivantes : Préconditions, Effets, Corps. Rappelons également que les méthodes classiques pour construire un plan sont :

- Le chaînage avant
- La planification avec plusieurs niveaux d'abstraction.

Les règles de planification utilisées dans [3] sont de la forme :

Si l'acteur A veut atteindre X,
Alors il peut vouloir atteindre Y,

et écrites : $Veut_A(X) \longrightarrow Veut_A(Y)$.

Exemples :

(C 1) : **Si** un acteur veut atteindre un but E,
Et ACT est une action qui a E pour effet,
Alors l'acteur peut vouloir exécuter ACT.

est écrite :

$$Veut_X(E) \longrightarrow Veut_X(ACT)$$

où E est un effet de ACT.

(C 2) : **Si** un acteur veut atteindre P,
Et ne sait pas si P est vrai,
Alors l'acteur peut vouloir atteindre le but : "l'acteur connait si P est vrai".

est écrite :

$$Veut_X(P) \longrightarrow Veut_X(Connait\text{-}si_X\, P)$$

Déduction d'un plan

La déduction d'un plan consiste à reconstruire le plan d'un agent à partir des actions qu'il effectue. Il s'agit, en fait, du processus inverse du processus de construction de plans décrit ci-dessus.

Comme dans le processus de construction, le processus de déduction de plans est représenté avec un ensemble de règles d'inférence et une stratégie de contrôle. Les règles sont de la forme :

Si l'acteur S croit que l'acteur A a X pour but,
Alors l'acteur S peut déduire que l'acteur A a Y pour but.

et s'écrivent : $Croit_S(Veut_A(X)) \longrightarrow Croit_S(Veut_A(Y))$

Par exemple les règles qui correspondent aux règles de planification (C 1) et (C 2) sont les suivantes :

(D 1) : **Si** S croit que A a l'intention d'exécuter l'action ACT,
Et ACT a pour effet E,
Alors S peut croire que A a l'intention d'atteindre E.

est écrite :

$$Croit_S(Veut_A(ACT)) \longrightarrow Croit_S(Veut_A(E))$$

où E est un effet de l'action ACT.

(D 2) : **Si** S croit que A a l'intention de connaître si la proposition P est vraie,
Alors S peut croire que A a l'intention d'atteindre P.

est écrite :

$$Croit_S(Veut_A(Connait\text{-}si_A(P))) \longrightarrow Croit_S(Veut_A(P))$$

Bien sûr, étant donnée la condition de (D 2), S peut déduire que A a l'intention d'atteindre $\neg P$. Ceci est traité par la règle suivante :

$$Croit_S(Veut_A(Connait\text{-}si_A(P))) \longrightarrow Croit_S(Veut_A(\neg P))$$

Détection des obstacles

La plupart des réponses coopératives surviennent quand le serveur détecte des obstacles dans le plan de l'utilisateur, c'est-à-dire des plans que l'utilisateur ne peut atteindre (facilement) sans aide. Les obstacles les plus évidents sont ceux que l'utilisateur

annonce dans ces phrases. Ces obstacles explicites sont des buts essentiels dans la chaîne d'inférences que le serveur réalise quand il déduit le plan de l'utilisateur. Toutefois, le serveur ne peut pas construire sa réponse sur les seuls obstacles explicites. Par exemple, si A, transportant un bidon vide, vient vers S et lui demande :

(5.1) *Où est la station service la plus proche ?*

et si S répond :

(5.2) *Au prochain carrefour.*

en sachant pertinemment que cette station est fermée, alors S n'a pas été coopératif, bien qu'il ait répondu correctement à la question. S doit indiquer à A s'il y a d'autres obstacles, et plus particulièrement ceux que A ne connait pas.

Les obstacles sont sélectionnés en utilisant les règles de préférence suivantes :

1. Les buts pour lesquels S et A sont en désaccord sur la valeur de vérité.
2. Les buts qui sont explicitement indiqués comme obstacles dans les phrases.
3. Les buts qui n'interviennent pas dans la réalisation des actions liées aux buts de la classe (2), mais qui ne sont pas atteints dans le plan.
4. Les buts qui sont précédés par d'autres buts dans l'ordre partiel des obstacles, et qui ne sont pas atteints dans le plan.

2.4.3 Exemples de réponses coopératives

Etudions le dialogue suivant ayant lieu dans une gare :

(1) Utilisateur : *Le huit heures cinq pour Milan ?*

(2) Serveur : *Le huit heures cinq pour Milan. Quai numéro 7.*

(3) Utilisateur : *Pourriez-vous me dire où cela se trouve ?*

(4) Serveur : *En descendant à gauche. Deuxième à gauche. Pas besoin de vous presser. Le train est en retard.*

Pour comprendre des phrases incomplètes telles que (1), le système a besoin de connaître le contexte de la phrase. Par exemple, si le serveur sait qu'une personne cherchant des informations essaye de prendre un train, d'attendre un train ou de réserver une place, (1) peut être compris en reconnaissant que l'utilisateur veut prendre un train, et pour cela il a besoin de savoir à quel quai ce train arrive.

La connaissance des plans et des buts de l'utilisateur est aussi utile pour comprendre les actes du langage indirects et fournir plus d'informations que demandé. Ainsi,

bien que (3) soit une question oui-non, le serveur répond comme si l'utilisateur voulait savoir où se trouvait le quai 7. De plus, la réponse du serveur inclut des informations supplémentaires. En effet, le serveur a reconnu le contexte du plan "prendre un train" dans la phrase (1), et il inclut donc des informations supplémentaires utiles pour l'utilisateur dans ce contexte.

Dans la suite de cette section, nous étudions en détail, l'ensemble des connaissances utilisées par le système pour simuler ce dialogue.

Le domaine des trains

On commence par donner une représentation simplifiée des actions, des plans et des buts. Elle est inadaptée pour représenter un monde réel, mais suffit ici. Le monde est décrit par un ensemble de propositions du calcul des prédicats du premier ordre. De plus, on a un ensemble d'actions définies par des conditions tombant dans l'une des trois classes suivantes :

Préconditions : ensemble de formules logiques qui doivent être vraies avant exécution de l'action.

Effets : ensemble de formules logiques qui doivent être vraies après exécution de l'action.

Corps : ensemble d'actions décrivant la décomposition de l'action en sous-actions.

Intuitivement, si le corps d'une action est exécuté dans une situation où les préconditions sont satisfaites, alors les effets seront satisfaits. Un plan sera représenté par un arbre dont les noeuds représentent les actions auxquelles sont associés les préconditions et les effets.

Considérons un ensemble de propositions, fonctions et actions utilisées pour représenter une sitation donnée survenant dans le contexte d'une gare. On pourrait avoir des prédicats tels que :

- POSSEDE (Acteur,Objet)
- A-BORD (Acteur,Train)
- PRES-DE (Acteur,Objet)
- DANS (Acteur,Ville)

et des fonctions telles que :

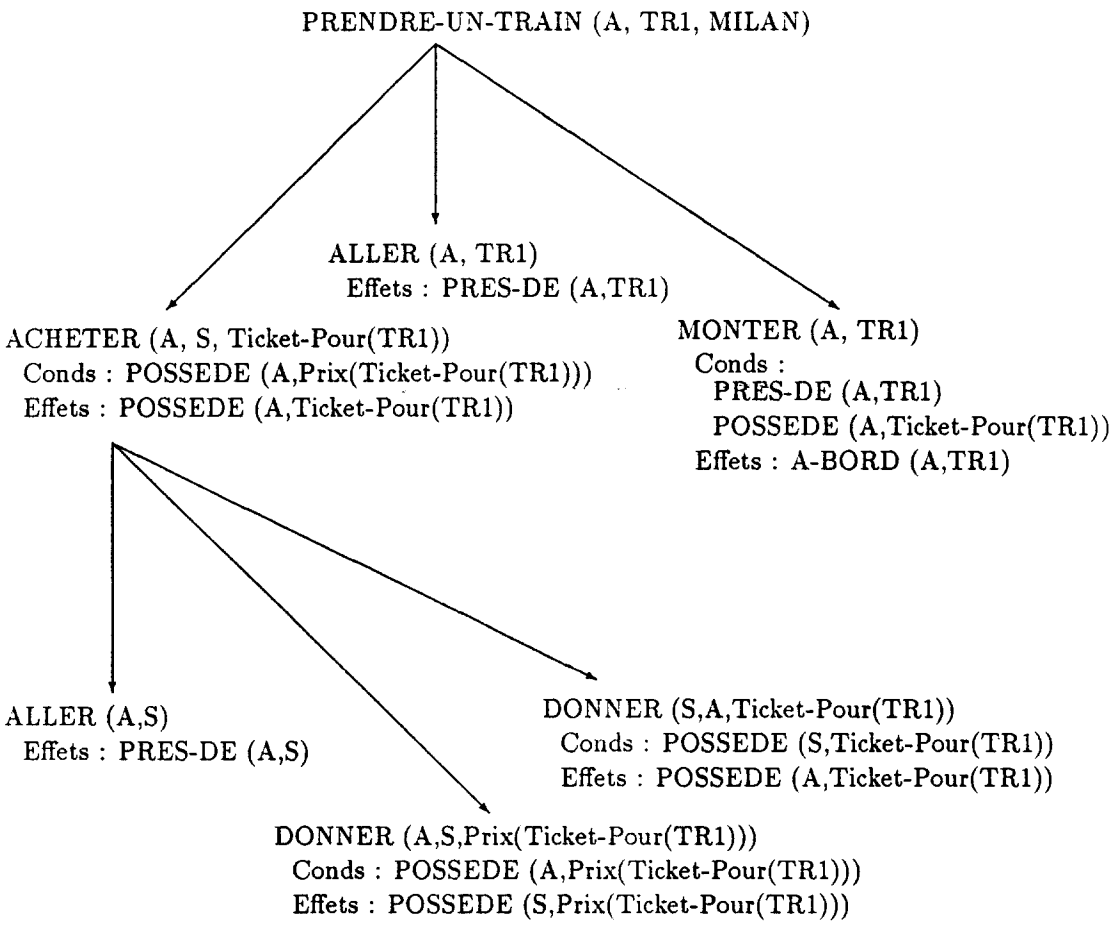
- Prix (Ticket)
- Loc (Objet)
- Ticket-Pour (Destination)

ainsi qu'un ensemble de type d'actions, tel que :

```
ACHETER (Acteur,Vendeur,Objet)
Préconditions : POSSEDE (Acteur,Prix(Objet))
Corps : ALLER (Acteur,Vendeur)
        DONNER (Acteur,Vendeur,Prix(Objet))
        DONNER (Vendeur,Acteur,Objet)
Effet : POSSEDE (Acteur,Objet)

PRENDRE-UN-TRAIN (Acteur,Train,Destination)
Préconditions : DESTINATION (Train,Destination)
Corps : ACHETER (Acteur,Serveur,Ticket-Pour(Train))
        ALLER (Acteur,Train)
        MONTER (Acteur,Train)
Effet : DANS (Acteur,Destination)

MONTER (Acteur,Train)
Préconditions : PRES-DE (Acteur,Train)
                POSSEDE (Acteur,Ticket-Pour(Train))
Effet : A-BORD (Acteur,Train)

DONNER (Acteur,Receveur,Objet)
Préconditions : POSSEDE (Acteur,Objet)
Effet : POSSEDE (Receveur,Objet)
```

Etant données des propositions représentant l'état initial, un but final ainsi qu'un ensemble d'actions possibles, on peut construire un plan (une séquence d'actions qui conduisent à l'état final, par exemple DANS (A,MILAN)). Pour construire un plan exécutable, le système utilise deux modes de raisonnements.

On commence d'abord par décomposer chaque action en ses sous-actions jusqu'aux actions indécomposables. On obtient ainsi le plan complet suivant :



Ensuite, on raisonne en chaînage arrière pour s'assurer que les préconditions de chaque action dans le plan sont satisfaites. Si une précondition n'est pas initialement satisfaite, et n'est pas atteinte par une action déjà dans le plan, alors une action ayant cette précondition comme effet est ajoutée au plan. Par exemple, si dans la génération du plan précédent, l'acteur n'a pas l'argent pour acheter le ticket, une action sera insérée (Aller à la banque par exemple).

Utilisation des plans

Si un acteur construit et exécute un plan, pour qu'il puisse comprendre les actions d'un autre acteur, il faut pouvoir reconnaître les plans qui motivent ces actions. La reconnaissance d'un plan peut être vue comme le processus inverse de celui décrit ci-dessus. Au lieu d'exécuter un plan pour atteindre un but, on observe l'exécution d'une action et on utilise notre connaissance des autres actions pour déterminer le but final à atteindre.

Tout ceci peut être mieux perçu en étudiant l'exemple suivant. Soit la phrase : "Paul achète un ticket pour MILAN, et se précipite vers le train". Indépendamment du contexte, on peut interpréter cette phrase comme suit :

- ACHETER(PAUL,s1,Ticket-Pour(Milan))
- DESTINATION(train1,MILAN)

pour la première partie de la phrase et

- COURIR(PAUL,train2)

pour la seconde, s1, train1 et train2 étant des constantes de Skolem.

En utilisant nos connaissances sur les actions, on peut donner une meilleure interprétation de cette phrase. Pour cela, on essaye de construire le plan de l'acteur PAUL dans cette histoire. En partant du fait que l'action ACHETER(PAUL,s1,Ticket-Pour(Milan)) est exécutée, on recherche les actions dans lesquelles l'action ACHETER intervient. Avec l'ensemble d'actions ci-dessus, on a une seule possibilité à considérer : PAUL a réalisé l'action ACHETER pour effectuer l'action

PRENDRE-UN-TRAIN(PAUL,train1,MILAN).

On examine alors les préconditions de PRENDRE-UN-TRAIN, c'est-à-dire DESTINATION(train1,MILAN), qui doivent être satisfaites si l'action PRENDRE-UN-TRAIN est exécutée.

Une fois ceci fait, on peut examiner la deuxième partie de l'interprétation à savoir COURIR(PAUL,train2). On essaye de l'interpréter comme une partie du plan déjà en construction. Si l'action COURIR est définie comme un sous-type de l'action ALLER, alors un peut unifier l'action COURIR avec l'une des actions ALLER qui font parties de PRENDRE-UN-TRAIN. Il y a ici une hypothèse implicite dans le langage : l'action ACHETER précède l'action COURIR. Ainsi, on ne peut unifier COURIR qu'avec l'action ALLER qui suit l'action ACHETER. L'unification permet de conclure que les constantes train1 et train2 sont égales. Ainsi, en supposant que les deux phrases sont des parties de l'action PRENDRE-UN-TRAIN, on peut identifier correctement chacun des référents.

De la même façon on peut interpréter correctement le dialogue suivant:

A : *Je voudrais acheter un ticket pour Milan. Voici la monnaie.*

B : *OK.* (B donne le ticket à A)

A : *Où dois-je me rendre ?*

B : *Quai numéro 7. Mieux vaut vous dépêcher, le train va bientôt partir.*

B peut utiliser ses connaissances sur PRENDRE-UN-TRAIN pour inclure dans la dernière réponse non seulement l'information explicitement demandée, mais également des informations utiles pour A. Ainsi, le comportement coopératif consistant à dire que le train va bientôt partir peut être modélisé à l'intérieur d'un plan de la façon suivante :

1. B reconnait le plan de A : PRENDRE-UN-TRAIN (A, train1, MILAN)
2. B utilise ce plan pour comprendre et répondre aux questions de A
3. B examine les obstacles présents dans ce plan et construit des plans pour les supprimer.

Définition des actes de langage

Rappelons que la décomposition de l'action ACHETER requiert trois étapes :

1. ALLER (A,vendeur)
2. DONNER (A,vendeur,Prix(Ticket-Pour(TR1))) où TR1 va à MILAN;
3. DONNER (vendeur,A,Ticket-Pour(TR1))

Il y a deux problèmes majeurs qui peuvent survenir lorsque A essaye d'exécuter ce plan. Le premier : A peut ne pas connaître le prix d'un ticket pour MILAN et donc ne peut pas exécuter (2). Le second : il n'y a pas de raison de supposer que le vendeur connaisse ce plan : le vendeur ne saura probablement pas qu'il faut exécuter (3). Intuitivement pour résoudre ces problèmes il faut :

- Pour le premier, demander au serveur le prix du ticket
- Pour le second, demander au serveur de donner le ticket à A.

Pour formaliser cela, il faut définir les actions correspondant aux actions linguistiques "informer" et "demander". Ces actions sont appelés des actes de langage par les philosophes [57] qui les ont étudiés.

De façon simplifiée, une formulation de l'action DEMANDER pourrait être :

DEMANDER (A,B,Action)
Préconditions : Aucune
Effets : VOULOIR (B,Action)

On peut définir l'action de l'information sincère de la façon suivante:

INFORMER (A,B,fonction)
Effets : SAVOIR (B,fonction)
Préconditions : SAVOIR (A,fonction)

Intuitivement, dans l'action INFORMER, A fournit à B la valeur d'une fonction (Prix(Ticket) par exemple).

En fait, pour être complet, il faut également ajouter des préconditions implicites à chaque action, à savoir : pour exécuter une action ayant pour paramètres fonctionnels $P_1, ..., P_n$, l'acteur doit connaître la valeur de chaque paramètre.

Considérons de nouveau l'acteur A essayant d'acheter un ticket pour MILAN. Pour exécuter l'étape (2) ci-dessus, A a besoin d'atteindre la précondition implicite :

- SAVOIR (A,Prix(Ticket-Pour(TR1)))

En cherchant une action réalisant ce but, on trouve que INFORMER résoud le problème, et donc A désire réaliser l'action :

- INFORMER (Acteur,A,Prix(Ticket-Pour(TR1)))

Si d'autre part, il y a dans la base de connaissances le fait que le vendeur connait ce prix, c'est-à-dire :

- SAVOIR (Vendeur,Prix(Ticket-Pour(TR1)))

alors en unifiant Acteur=Vendeur, on satisfait la précondition. Mais, INFORMER est une action intentionnelle. Par suite, pour que le vendeur réalise l'action :

- INFORMER (Vendeur,A,Prix(Ticket-Pour(TR1)))

il faut qu'il désire la réalisation de cette action, c'est-à-dire :

- VOULOIR (Vendeur, INFORMER (Vendeur,A,Prix(Ticket-Pour(TR1))))

Ceci peut être réalisé si A demande au vendeur de réaliser l'action. Ainsi, A peut accomplir l'étape (2) en exécutant le sous-plan :

(2.1) DEMANDER(A,Vendeur,INFORMER(Vendeur,A,Prix(Ticket-Pour(TR1))))
qui permet de réaliser

VOULOIR(Vendeur,INFORMER (Vendeur,A,Prix(Ticket-Pour(TR1))))

(2.2) INFORMER (Vendeur,A,Prix(Ticket-Pour(TR1)))
qui permet de réaliser

SAVOIR (A,Prix(Ticket-Pour(TR1)))

(2.3) DONNER (A,Vendeur,Prix(Ticket-Pour(TR1)))

De même, on peut réaliser l'étape (3) en exécutant l'action :

- DEMANDER (A,Vendeur,DONNER(Vendeur,A,Ticket-Pour(TR1)))

Un exemple de dialogue représentant ces actes de langage pourrait être:

A : *Combien coûte un ticket pour Milan ?* (Demande 2.1)

B : *13.50* (Information 2.2)

A : *Puis-je avoir un ticket, s'il vous plait ?* (Demande 3)

Comment fournir plus d'informations que demandées ?

Il est utile de reconnaître le plan qu'un acteur essaye de réaliser à l'aide d'actes de langage pour pouvoir générer des réponses pertinentes. Par exemple, si le vendeur observe l'acte de langage :

- DEMANDER(A,Vendeur,DONNER(Vendeur,A,Ticket-Pour(TR1)))
  avec DESTINATION (TR1,MILAN),

alors on peut déduire d'après l'effet de DEMANDER, que le plan de A est :

- PRENDRE-UN-TRAIN(A,TR1,MILAN).

Si l'on veut être coopératif, on peut examiner le plan pour voir si l'on peut aider A. Par exemple, on peut croire que A va ensuite exécuter l'action ALLER(A,TR1), et donc A doit réaliser l'action SAVOIR(A,Loc(TR1)) qui est une précondition implicite de ALLER(A,TR1). On peut donc réaliser l'action DONNER(Vendeur,A,Ticket-Pour(TR1)) pour satisfaire la requête, et en plus INFORMER (Vendeur,A,Loc(TR1)) pour satisfaire la précondition de l'action ALLER. Bien sûr, si le vendeur croyait que A savait déjà où prendre le train, il n'aurait pas généré cette action supplémentaire.

2.4.4 Conclusion

L'approche développée par Allen et *al.* est très intéressante et très complète. On retiendra, en particulier, l'utilisation des logiques de la connaissance et de la croyance pour gérer un dialogue, ainsi que l'utilisation des plans et des actions pour comprendre des questions.

Un nombre important de chercheurs ont étudié ce genre de problèmes parmi lesquels on peut citer S. Carberry [9], B. Grosz [36] et C. Sidner [59].

En ce qui nous concerne, nous retiendrons plus particulièrement l'utilisation de ce formalisme pour générer des réponses coopératives, mais il faut rappeler que ce modèle

a été également utilisé pour comprendre des actes de langage indirects, pour interpréter des requêtes mal formées et pour comprendre des ellipses entre plusieurs phrases.

Toutefois, nous partageons le point de vue de S. Carberry présenté dans [10] et considérons que l'approche suivie dans ces travaux est un peu limitative. En effet, dans tous les cas, quatre hypothèses ont été faites :

1. Les connaissances de l'utilisateur sur le domaine peuvent être incomplètes mais pas erronées.
2. L'utilisateur n'interroge jamais le système sur des aspects de l'application qui sont en dehors des connaissances du système.
3. Les informations fournies par le système sont correctes et sans ambiguité.
4. Le plan déduit par le système après analyse d'une nouvelle question est une version partiellement instantiée du plan réel considéré par l'utilisateur.

Or S. Carberry considère que si l'on veut des systèmes capables de comprendre et de répondre correctement à des dialogues naturels, les systèmes doivent être capables de prendre en compte des situations où ces hypothèses ne sont plus vraies.

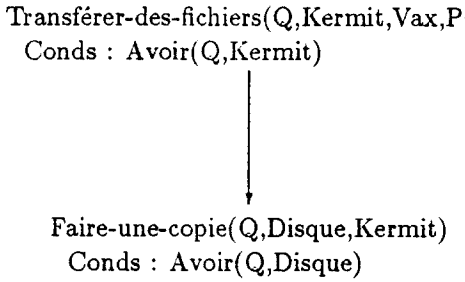
Elle étudie donc quatre cas (correspondant chacun à une situation où une des hypothèses ci-dessus n'est pas satisfaite) pour lesquels le modèle construit par le système ne correspond pas au plan réel de l'utilisateur :

1. Modèle erroné : le modèle déduit par le système est inconsistant avec le modèle de l'utilisateur.
2. Modèle trop spécialisé : le modèle déduit par le système est plus restrictif que celui de l'utilisateur.
3. Modèle trop généralisé : le modèle est moins spécifique que celui de l'utilisateur.
4. Modèle limité en connaisssance : le système ne parvient pas à déduire le plan que construit l'utilisateur à cause d'une connaisssance limitée de l'application.

Ainsi, considérons le dialogue suivant, qui illustre le cas où le système a une connaissance limitée de l'application (cas 4) :

1. Q : J'aimerais transférer des fichiers entre un VAX et mon PC.
2. R : Kermit vous permet de faire cela.
3. Q : Comment puis-je obtenir Kermit ?
4. R : Le centre de calcul vous fournira une copie si vous apportez un disque.
5. Q : Jusqu'à quelle heure la boutique universitaire est-elle ouverte ?

Après la phrase [4], le modèle du plan de l'utilisateur construit par le système est le suivant :



Si l'on suppose que R ne sait pas comment acheter un disque alors la requête [5] de Q ne pourra être rattachée au plan déduit par R.

La maxime de relation de Grice [33] suggère pourtant que Q croit que cette requête est pertinente avec le reste du dialogue. Par suite, R doit essayer de déterminer comment la phrase de Q peut se rattacher au plan en construction et considérer ici la possibilité que les connaissances de Q dépassent celles de R.

Pour cela, R pourrait faire le raisonnement suivant :

> La boutique universitaire est un endroit où l'on achète des livres scolaires qui sont une sous-classe des fournitures scolaires de même que les disques. Par suite, R peut faire l'hypothèse que Q pense que la boutique universitaire vend des disques, puisqu'elle vend d'autres fournitures scolaires.

R peut alors répondre de la façon suivante :

6. R : La boutique universitaire est ouverte jusqu'à 4 heures 30, mais je ne sais pas si l'on y vend des disques. Toutefois, on y trouve beaucoup de fournitures scolaires, et vous avez donc des chances d'y trouver ce que vous cherchez.

Ensuite, R va insérer ces nouveaux composants dans le plan global représentant les intentions de Q, en indiquant toutefois que les propres connaissances de R n'offrent qu'une aide limitée pour les prendre en compte.

Pour récapituler, une étude des dialogues réels montre que les êtres humains utilisent, pour détecter et prendre en compte des structures de plan inadaptées, une approche se décomposant en quatre phases :

1. Détection du désaccord entre les modèles de l'utilisateur et du système.
2. Construction des hypothèses sur la cause du désaccord.
3. Génération d'une réponse adaptée, incluant souvent une phase de négociation pour confirmer la cause du désaccord.

4. Modification des modèles, pour mettre à jour les structures des plans de chaque interlocuteur.

La simulation de chaque phase par le système qui sert d'interlocuteur à l'utilisateur est un problème difficile. En particulier, la phase [3] de génération des réponses dépend de la nature du désaccord entre l'utilisateur et le système.

Ainsi, dans le cas de modèle trop généraux (cas [3]), R doit entrer dans une phase de clarification du dialogue pour préciser ce que désire Q.

Dans les cas de modèles trop spécialisés ou erronés (cas [1] et [2]), R doit entrer dans un processus de négociation ayant un but double :

1. R peut déterminer les parties du plan suspectes et essayer de convaincre Q que son plan doit être modifié.
2. R peut poser des questions pour déterminer les désaccords et trouver ainsi pourquoi son modèle est erroné.

2.5 Bilan de l'étude bibliographique

En conclusion, nous essayons de faire un bilan comparatif de l'étude bibliographique et des problèmes que nous étudions dans la suite de ce travail.

Pour ce qui est de la forme, remarquons tout d'abord que nous ne travaillons pas en langue naturelle. Pour exprimer les questions, nous utilisons un formalisme logique, langage qui est souvent utilisé pour interroger une base de données.

Si l'on voulait traduire les questions exprimées dans ce formalisme en langue naturelle, on obtiendrait des requêtes ayant un format syntaxique restreint de la forme suivante :

Pour les entités x
Satisfaisant les conditions y
Quelles sont les valeurs des attributs z_1, ..., z_n

C'est plus ce format syntaxique qui constitue une restriction que le fait de travailler dans un formalisme logique plutôt qu'en langue naturelle. En effet, une méthode classique utilisée pour interpréter une requête exprimée en langue naturelle, consiste à la tranformer dans une représentation interne, qui peut utiliser un formalisme logique.

En revanche, nous ne traitons pas les questions de la forme "Pourquoi ?" ni "Comment ?". La restriction que nous imposons fournit cependant un problème suffisamment difficile et intéressant.

Sur le fond, nous ne nous sommes pas attachés à étudier le problème consistant à fournir une réponse indirecte à une requête lorsque celle-ci échoue (Cf. problème présenté dans la partie 2.1). Nous considérons en effet que ce problème a déjà été étudié dans les travaux de Kaplan et de Janas en particulier.

Nous étudions le problème consistant à fournir une réponse suggestive (Cf 2.2). Toutefois, nous ne limitons pas la génération de telles réponses au cas où la requête initiale échoue. En effet, s'il est tout à fait pertinent d'essayer de fournir des alternatives dans le cas d'un échec de la requête, ce seul cas semble très limitatif.

Ainsi, si l'on se place dans une situation où l'on suppose que l'utilisateur n'a pas une idée très précise de ses intentions, alors l'un des rôles d'un serveur coopératif est de faire des suggestions qu'il juge pertinentes.

Pour s'en convaincre, il suffit de reprendre l'exemple de dialogue présenté dans l'introduction :

A : Quel bus faut-il prendre pour se rendre à la place du Capitole ?

B : Vous pouvez prendre le 22, mais vous aurez plus vite fait d'y aller à pied, c'est à 5 minutes d'ici.

Dans cet exemple, B bien qu'ayant fourni la réponse exacte à la requête de A, fournit également une alternative qu'il juge pertinente.

Nous avons essayé de faire une étude exhaustive du problème des réponses complétives (Cf. 2.3). Il est intéressant de noter que ce problème est traité comme un cas particulier de transformation des informations demandées.

Nous étudions également en détail le problème consistant à effectuer une intervention pertinente. Toutefois, notre approche est sensiblement différente de celle présentée dans la section 2.4.

En effet, dans l'approche suivie par Allen par exemple, on considère que lorsqu'un utilisateur A pose une question à un interlocuteur B, il cherche en fait à atteindre un but. La réponse à la question va fournir à A des informations utiles à la réalisation de son but.

Par suite si B est coopératif, il va fournir à A des informations supplémentaires dont la non connaissance peut constituer un obstacle à la réalisation du but de A. Pour cela, B doit d'abord reconnaître le but de A, puis concevoir un plan permettant la réalisation de ce but.

Comme nous l'avons indiqué dans la partie précédente, nous considérons que cette façon de voir les choses, bien que très séduisante, est un peu limitative.

Nous étudierons en particulier le cas où B ne connait pas avec précision le but que A cherche à atteindre, et pourtant fournit des informations supplémentaires pertinentes. Si l'on revient aux problèmes considérés par S. Carberry, celui-ci correspond au cas où le système ne parvient pas à déduire le plan que construit l'utilisateur à cause d'une connaissance limitée de l'application.

Ensuite, nous considérons que le cadre dans lequel se place Allen ne constitue pas la seule raison pouvant conduire B à effectuer des interventions pertinentes. Nous étudions ce problème plus en détail par la suite.

Chapitre 3

Différents types de connaissances et concepts

Au travers des exemples présentés dans le chapitre 1, nous avons essayé de mettre en évidence plusieurs concepts et types de connaissances qui semblent utiles pour définir la notion d'information supplémentaire pertinente. Dans cette partie, nous essayons de récapituler et de présenter plus en détail ces différentes notions.

3.1 Modèle Entité/Relation

Pour représenter les informations contenues dans la base de données, nous utilisons les concepts d'entité et d'attribut d'entité. Bien que cela n'apparaisse pas clairement dans les exemples presentés, le concept de relation entre entités nous semble également intéressant. Donc pour représenter le schéma de la base de données, nous utilisons les concepts :

- Type d'entité
- Relation entre entités
- Attribut d'entité ou de relation

Nous faisons ici un bref rappel sur ces différentes notions empruntées au modèle entité-relation [13].

Intuitivement, une entité est un "objet" qui peut être identifié. Une *personne* ou une *compagnie* sont des exemples d'entités. Une relation indique une association entre entités. Par exemple, *père-fils* est une relation entre deux entités *personne*.

Les entités sont classées en "ensembles d'entités" tels que *Employé, Projet ou Département*. Il y a un prédicat associé à chaque ensemble d'entités qui teste l'appartenance

d'une entité à l'ensemble. Par la suite, on utilisera aussi les mots "type" et "classe" comme synonyme d'"ensemble d'entités".

La notion d'ensemble d'entités est semblable à celle de classification introduite par Brodie dans [6]. Pour lui, un objet est une instance d'une classe d'objet s'il a les propriétés définies dans la classe. Plutôt que d'associer à chaque ensemble d'entités un prédicat pour tester l'appartenance de l'entité à l'ensemble, Brodie préfère introduire la relation *instance-de* entre un objet particulier dans la base et une classe d'entités.

Une relation R_i est une relation mathématique entre n entités e_i appartenant chacune à un type donné E_i. Les types E_i peuvent ne pas être distincts. Par exemple, un *mariage* est une relation entre deux entités de types *Personne*.

Le "rôle" d'une entité dans une relation est la fonction que l'entité joue dans la relation. *Mari* et *Epouse* sont des rôles de la relation *Mariage*.

Les informations concernant une entité s'expriment à l'aide d'un ensemble de couples attribut-valeur. Les valeurs sont classées en différents "ensembles valeur". Il y a un prédicat associé à chaque "ensemble valeur" pour tester si une valeur appartient à l'ensemble.

Un attribut peut être défini comme une fonction d'un ensemble d'entités ou un ensemble relation dans un ensemble valeur ou un produit cartésien d'ensemble valeur. Toutefois, dans la suite, nous représenterons plutôt un attribut par un prédicat. Ainsi, l'attribut *Age* d'une *Personne* sera représentée en utilisant le prédicat $Age(x, y)$ plutôt que la fonction $y = Age(x)$.

Il faut remarquer que les relations peuvent également avoir des attributs. Par exemple, on peut définir une relation *Réservation* entre les entités de type *Personne* et *Vol* pour indiquer qu'une personne a réservé une place sur un vol. On peut ensuite associer, à cette relation, l'attribut *Numéro-de-place* pour indiquer la place reservée par cette personne.

Toutes les entités ont un nom qui permet de les identifier de façon unique. Ici, Chen [13] parle de la clé primaire de l'entité.

3.2 Structuration des entités

Il nous a également semblé utile de définir une structure entre les types d'entité pour représenter différents types d'abstraction et en particulier représenter la relation d'inclusion (appelée souvent relation *Is-a*).

Ainsi, si l'on reprend la base de données "agence de voyage", on peut définir les types d'entités *Vol* et *Trajet*. La relation *Is-a* $(Vol, Trajet)$ permet alors d'exprimer que *Vol* est un sous-type de *Trajet*.

Dans le cadre de la génération de réponse coopérative, il est intéressant d'avoir la relation d'inclusion lorsqu'une même règle de transformation s'applique à plusieurs types d'entités; il est alors possible d'avoir une seule règle, concernant le type d'entité le plus général, et d'appliquer cette règle a chaque type d'entité particulier.

Par exemple, si la même règle s'applique aux *Vols* et aux *Trains*, elle peut être exprimée en utilisant le type d'entité *Trajet*.

3.3 Regroupement des attributs en *thèmes*

Nous avons également introduit la notion de "*Thème*".

Les *thèmes* sont associés aux attributs, et permettent de représenter certains liens sémantiques entre les attributs.

Ainsi, dans la base de données "agence de voyage", on peut introduire le *thème Horaire* et considérer que les attributs *Heure-de-départ* et *Heure-d'arrivée* sont tous deux associés à ce *thème*.

Les *thèmes* permettent une expression plus générale des règles fournissant des informations sur un ensemble d'attributs concernant le même *thème*.

Comme les types d'entités, ils peuvent être structurés en utilisant la relation de généralisation.

On peut, par exemple, introduire un deuxième *thème Date* et la relation *Is-a* (*Horaire, Date*) permet d'exprimer que *Horaire* est un sous-cas de *Date*.

3.4 Les objets structurés

Il nous a également semblé intéressant d'introduire la notion d'objet structuré.

Ceci permet par exemple de représenter et de manipuler des objets constitués par l'agrégation de plusieurs sous-objets tel que l'objet *Triangle* emprunté à [14].

Ainsi, un triangle est défini comme étant constitué par trois segments $Segment_1$, $Segment_2$ et $Segment_3$.

Un segment est lui-même défini comme étant constitué par deux points *Org* (l'origine) et *Ext* (l'extrémité).

Enfin, un point est constitué par deux réels *Abs* (l'abscisse) et *Ord* (l'ordonnée), les réels étant des objets de base indécomposables.

Il est également intéressant de pouvoir représenter des objets constitués par un ensemble (non-ordonné) de sous-objets ou par une liste (ordonnée) de sous-objets.

Ceci permet par exemple de définir l'objet structuré *Portefeuille* constitué des trois sous-objets :

- *Ensemble d'Actions*
- *Ensemble d'Obligations*
- *Or*

où *Ensemble d'Actions* (resp. *Ensemble d'Obligations*) est un objet constitué par un ensemble d'objets de type *Actions* (resp. *Obligations*).

On peut également définir l'objet *Voyage* constitué par les attributs suivants :

- *Ville-de-Départ*
- *Ville-d'Arrivée*
- *Heure-de-départ*
- *Heure-d'Arrivée*
- *Prix*
- *Liste de Trajets*

où *Liste de Trajets* est un objet constitué par une liste ordonnée d'objets de type *Trajet*.

3.5 Expression d'une requête

Pour définir la transformation d'une requête, il est utile de pouvoir distinguer trois parties dans une requête :

1. Le type des entités à fournir en réponse ; cette partie sera appelée par la suite partie "Sujet".
2. Les conditions que les attributs de ces entités doivent satisfaire ; cette partie sera appelée partie "Condition".
3. Les attributs des entités à fournir dans la réponse ; cette partie sera appelée partie "Informations Demandées".

La forme générale des requête sera donc :

$$Sujet \wedge Condition \wedge Informations\ Demandées$$

Ce format particulier que nous imposons aux requêtes est seulement introduit comme une convention syntaxique intéressante pour la suite de ce travail. On peut remarquer qu'il s'apparente au format syntaxique de la plupart des langages d'interrogation d'une base de données tels que QUEL (*Range of ... Retrieve ... Where ...*) ou SEQUEL (*Select ... From ... Where ...*).

3.6 Représentation des informations concernant l'utilisateur

Nous avons déjà signalé dans les parties précédentes que le système avait besoin, pour avoir un comportement coopératif, de connaître un certain nombre de faits concernant l'utilisateur.

Si l'on considère une conversation entre deux personnes, chacun obtient des informations sur son interlocuteur tout au long du dialogue. Dans le cas d'une conversation avec une machine, le même processus peut être simulé par un système de gestion du dialogue dont le rôle est d'obtenir et de mémoriser ces informations. On construit ainsi un "modèle de l'utilisateur". Les approches envisagées pour construire de tels modèles sont nombreuses comme on peut s'en rendre compte en lisant la synthèse présentée dans [12].

Une approche simple consiste à reconnaître et à construire un modèle canonique de l'utilisateur et à développer un système adapté à cet utilisateur "standard". Pour la majorité des personnes, le système ainsi obtenu sera mieux adapté que s'il avait été construit sans tenir compte de ces études. Toutefois, on peut faire mieux et une approche plus ambitieuse consiste à construire un système capable de s'adapter aux caractéristiques de chaque utilisateur.

Ce problème important commence à attirer l'intérêt à la fois des psychologues et des informaticiens. Un cas plus particulièrement abordé par les réalisateurs d'interface homme-machine est celui de la classe des utilisateurs "occasionnels" (Cf., par exemple, [18]). Malheureusement, peu de systèmes sont utilisés par des personnes appartenant à une seule classe. Or, il apparait que les modifications destinés à faciliter la tâche de certaines classes d'utilisateurs, se transforment en difficultés supplémentaires pour d'autres classes. Par exemple, une étude sur les performances d'utilisateurs d'éditeur de texte expérimentés suggère que le nombre d'actions à effectuer pour réaliser une action devrait être minimisé [11]. Une autre étude concernant les personnes apprenant à utiliser un éditeur [45] suggère que des commandes avec des mots complets devraient être utilisées. Ces contraintes conflictuelles montrent clairement le besoin de systèmes s'adaptant à chaque utilisateur.

Dans [55], E. Rich distingue trois dimensions possibles lorsqu'on construit un modèle de l'utilisateur :

1. Un modèle d'un utilisateur canonique contre une collection de modèles d'utilisateurs particuliers.
2. Des modèles spécifiés par le concepteur du système ou par les utilisateurs eux-mêmes contre des modèles déduits par le système en se fondant sur le comportement des utilisateurs.
3. Des modèles de caractéristiques de l'utilisateur stables pour une durée assez longue (comme par exemple le niveau d'expertise ou les domaines d'intérêt) contre des modèles de caractéristiques de l'utilisateur valables pour une durée plus courte (comme par exemple le problème que l'utilisateur est en train de résoudre).

Nous avons indiqué que l'utilisation d'un modèle canonique pour s'adapter à des utilisateurs ayant des caractéristiques différentes est d'une efficacité limitée. Des modèles particuliers pour chaque utilisateur permettraient souvent de fournir une interface plus adaptée. Toutefois, si le système veut utiliser un grand nombre de modèles correspondant à chaque utilisateur, on doit alors se poser la question de savoir comment ces modèles peuvent être construits.

Il y a en fait deux approches pour adapter un système à des utilisateurs différents.

La première consiste à permettre aux utilisateurs de modifier le système de la façon qui leur convient le mieux. Malheureusement, une telle approche demande en général une connaissance approfondie du système et s'avère donc inadaptée pour les utilisateurs occasionnels ou débutants.

Dans la seconde approche, il s'agit de fournir au système les informations sur les utilisateurs. Ceci peut se faire trivialement ou de façon plus intelligente.

Un exemple trivial consiste à demander à l'utilisateur son niveau d'expérience du système. Le programme contient alors des modèles associant un ensemble de connaissances à chaque niveau d'expérience. Cette approche a été critiquée dans [32] : on ne peut en effet considérer que les connaissances d'un débutant peuvent être perçues comme un simple sous-ensemble des connaissances de l'expert et que le passage du débutant à l'expert n'est constitué que par une simple augmentation de connaissances. L'apprentissage implique en fait une restructuration complète des connaissances.

D'autres arguments peuvent être avancés contre la construction explicite du modèle par l'utilisateur. D'abord les utilisateurs ne peuvent pas toujours indiquer au système ce que ce dernier a besoin de savoir. Par exemple, comme le montre les travaux sur les systèmes d'enseignement intelligents, l'étudiant ne sait pas ce qu'il ne sait pas, et encore moins ce qu'il connait de façon incorrecte. Les étudiants ne sont pas les seuls pour lesquels se pose ce probème. De nombreux travaux en psychologie viennent confirmer que les personnes ne sont pas en général des sources d'informations fiables lorsqu'ils doivent se décrire. De plus, en général, les utilisateurs n'aiment pas répondre. Ceci est particulièrement vrai pour les utilisateurs occasionnels.

Pour résoudre ce problème, le système doit "se débrouiller" pour construire un modèle initial et laisser à l'utilisateur l'accès immédiat du sytème. Ce modèle initial s'inspirera d'un modèle canonique de l'utilisateur "standard" auquel on ajoutera un ensemble de faits caractéristiques d'un nouvel utilisateur ainsi que les informations "système" dont on dispose pour tous les utilisateurs (par exemple, le nom et le prénom). L'utilisateur dialogue ensuite avec le système et fournit ainsi des informations supplémentaires le concernant. Le système effectue des mises à jour et construit peu à peu un modèle particulier de l'utilisateur distinct du modèle standard. Dans cette approche, on voit donc que le travail doit surtout se porter sur la construction d'un "bon" modèle canonique des utilisateurs fréquents. D'autre part, il faut remarquer que les informations supplémentaires contenues dans le modèle particulier construit par le système sont des conjectures, des paris sur l'utilisateur. Par suite, dans [55], E. Rich suggère que le système doit avoir la possibilité de représenter la vraissemblance des faits, de résoudre les conflits et d'effectuer des mises à jour lorsque de nouvelles informations sont connues.

Celle-ci considère ensuite qu'il est commode de distinguer, parmi les informations concernant l'utilisateur, certains faits stables pour une durée assez longue et d'autres faits valables pour une durée plus courte; des techniques différentes peuvent, en effet, être adéquates pour résoudre ces deux problèmes.

La modélisation d'informations valables pour une durée courte est importante pour comprendre un dialogue en langue naturelle. Ainsi, le plan construit par le système pour comprendre les intentions de l'utilisateur (Cf. [3] par exemple ainsi que le chapitre précédent) est valable tant que le but de l'utilisateur ne change pas.

Mais, le système peut également utiliser des informations plus stables concernant l'utilisateur, comme par exemple, le niveau d'expérience de l'utilisateur :

- en informatique,
- pour le système particulier utilisé,
- pour le domaine d'application de ce système.

Le modèle devra également contenir des informations spécifiques utiles pour le système, comme par exemple, dans le cas de notre application type "agence de voyage" :

- L'utilisateur voyage pour raison touristique
- L'utilisateur est un homme d'affaires
- L'utilisateur est pressé
- L'utilisateur n'a pas de probème d'argent
- L'utilisateur n'a jamais pris l'avion

Etant donné la diversité des informations à représenter, on peut utiliser plusieurs techniques différentes pour construire un modèle de l'utilisateur, en particulier :

1. Identification du vocabulaire et des concepts employés par l'utilisateur.
2. Etude du comportement de l'utilisateur face aux réponses (en particulier est-il satisfait ou non ?).
3. Construction de stéréotypes pour représenter des catégories d'utilisateurs.
4. Prise en compte des connaissances et des croyances de l'utilisateur.
5. Reconnaissances des intentions de l'utilisateur.

Dans la suite de cette section, nous présentons rapidement ces diverses techniques.

3.6.1 Identification du vocabulaire

Une technique simple pour déduire de l'information concernant un utilisateur est de considérer sa façon d'utiliser le système [55]. Une personne qui utilise immédiatement des commandes élaborées est probablement un expert. Une autre personne dont les premières tentatives sont rejetées par le système est probablement un débutant.

Partant de là, on peut construire un dictionnaire des commandes et de leurs options en associant les informations que leur utilisation fournit concernant l'utilisateur. Ces informations peuvent avoir plusieurs dimensions telles que l'expérience du système ou du domaine d'application. Elles sont utilisées pour déterminer le vocabulaire utilisé par le système dans les réponses et les explications.

Ainsi, si l'on considère l'application type "gestion de portefeuille" présentée au chapitre 1, il est certainement très important de fournir des réponses dans un vocabulaire adapté à chaque utilisateur. En effet, ce domaine d'application utilise des notions complexes qui ne sont connues que par les personnes ayant déjà de l'expérience dans le domaine financier. Ces notions peuvent paraître ésotériques pour le néophyte. Par suite, le système doit tenir compte des caractéristiques de l'utilisateur s'il veut se faire comprendre.

3.6.2 Etude du comportement

Une autre façon d'obtenir de l'information sur l'utilisateur consiste à observer son comportement face aux réponses obtenues. S'il est satisfait, il va continuer avec une autre requête ou bien quitter le système. En revanche, s'il n'a pas obtenu ce qu'il voulait, il va essayer de restructurer sa question pour obtenir ce dont il a besoin. Cette tentative doit indiquer au système qu'il n'a pas satisfait les besoins de l'utilisateur avec sa première réponse et lui suggérer que son modèle de l'utilisateur est peut-être inadapté.

L'étude du comportement et l'observation du vocabulaire sont deux techniques utilisées dans le système Scribe [55] pour fournir une aide aux utilisateurs d'un système de traitement de texte.

3.6.3 Construction de stéréotypes

La technique utilisée maintenant pour représenter les caractéristiques de l'utilisateur part de l'hypothèse que les caractères humains ne sont pas distribués aléatoirement parmi la population. Souvent, on peut réaliser des regroupement (par exemple, une personne aisée financièrement est supposée avoir voyagé davantage qu'une autre personne très pauvre). Par suite, dans de nombreuses situations, il est possible de déduire, de l'observation d'un fait ou d'un ensemble de faits, un ensemble de faits supplémentaires caractérisant l'utilisateur.

Ces connaissances peuvent être représentées en utilisant un ensemble de stéréotypes, chaque stéréotype permettant de regrouper une collection de caractéristiques. En fait, il existe une analogie très forte entre la structure Stéréotype-Caractéristique et la structure

Entité-Attribut que nous avons déjà introduite. Ainsi, un stéréotype peut être perçu comme une collection de couple Attribut-Valeur. Souvent, on peut également introduire une structure hiérarchique sur les stéréotypes, le stéréotype le plus général représentant un modèle (canonique) de l'utilisateur standard.

Certaines caractéristiques (*démons*) sont facilement observables et sont utilisées pour "activer" un stéréotype. Comme la présence d'une caractéristique suggère seulement un stéréotype sans qu'on ait de certitude le concernant, on associe souvent (Cf. [55] par exemple) à chaque démon une mesure indiquant si le stéréotype est adéquat lorsque le démon est observé. En introduisant cette notion de mesure, un problème important se pose : comment détecter et résoudre les conflits entre les inférences ? Pour le résoudre, on utilise en général des méthodes *ad-hoc* : Par exemple, si un stéréotype indique une valeur pour une caractéristique particulière, et si le modèle de l'utilisateur construit par le système contient une valeur différente pour cette caractéristique, le conflit est résolu si la valeur du modèle provient d'un stéréotype plus général que celui qui vient d'être sélectionné.

Cette technique de représentation des utilisateurs a été utilisée pour construire Grundy [54], un système qui recommande des romans à ses utilisateurs.

Pour cela, Grundy utilise une description de chaque livre et des stéréotypes contenant des caractéristiques représentant les goûts des gens pour les livres.

Grundy commence par demander à l'utilisateur de décrire ses goûts en quelques mots et commence à construire un premier modèle. Puis, Grundy propose des livres adaptés au modèle construit.

- Si l'utilisateur a déjà lu le livre, Grundy sait qu'il est sur la bonne voie; il renforce ses croyances pour le modèle en construction.
- Si l'utilisateur n'aime pas le livre, Grundy cherche pourquoi en posant des questions jusqu'à ce que le problème soit localisé; le modèle de l'utilisateur ainsi que la base de données de stéréotypes sont mis à jour.
- Si l'utilisateur n'a pas lu le livre, Grundy essaye de le décrire en utilisant le modèle de l'utilisateur pour choisir les caractéristiques qu'il est intéressant de mentionner. Si l'utilisateur apprécie, Grundy est satisfait. Sinon, Grundy utilise la procédure ci-dessus pour localiser le problème et mettre à jour son modèle de l'utilisateur.

Il est intéressant de remarquer que le stéréotype initial de Grundy représentant un lecteur masculin standard, indiquait que les hommes aimaient lire des livres avec une intrigue rapide et beaucoup de suspense. Mais, dans la population universitaire considérée pour le test, les goûts se sont plutôt portés sur les livres intellectuels et philosophiques. Grundy a alors modifié le prototype *Homme* pour prendre en compte les goûts de la population considérée. Par suite, il n'est pas important pour Grundy, que les informations décrivant les utilisateurs soient correctes au départ. Si le même système est utilisé dans plusieurs communautés, ses stéréotypes vont évoluer différemment.

Les informations contenues dans les stéréotypes peuvent être utilisées de plusieurs façons par le module de génération de réponses coopératives. Par exemple, le niveau

d'expérience de l'utilisateur par rapport au domaine d'application permet de déterminer un niveau de coopération adapté pour cet utilisateur. Ce niveau de coopération peut être utilisé pour moduler le volume d'informations supplémentaires à fournir pour chaque utilisateur. Les stéréotypes peuvent être également uitlisés pour déterminer les thèmes d'intérêt de chaque utilisateur.

Dans ce travail, nous n'avons pas étudié en détail comment un stéréotype de l'utilisateur peut être construit, et nous supposons qu'une telle représentation de l'utilisateur a déjà été déterminée par un autre module (par exemple le module de gestion de dialogue). Il est intéressant de remarquer que le module de génération de réponses coopératives n'est pas le seul module a utiliser les caractéristiques de l'utilisateur. Ainsi, dans le domaine de la gestion de portefeuille, le module de résolution de problèmes a également besoin des caractéristiques de l'utilisateur (par exemple, son attitude face au risque).

3.6.4 Prise en compte des connaissances et croyances

Un autre aspect important dans la construction d'un modèle de l'utilisateur est constitué par la prise en compte des connaissances et croyances de l'utilisateur.

Cette approche est surtout étudiée dans les systèmes d'enseignement intelligents qui essayent de contrôler l'apprentissage de l'élève en utilisant de tels modèles.

J. Self dans [58] considère que le modèle de l'élève peut être vu comme un quadruplet (P, C, T, H) dans lequel :

- P décrit le niveau de connaissances procédurales de l'élève.
- C décrit les connaissances conceptuelles de l'élève.
- T décrit les traits particuliers de l'élève.
- H est l'historique (contraction des sessions antérieures).

L'ensemble $k = P \cup C$ décrit ce que l'élève sait, K étant l'ensemble de connaissances que l'élève doit atteindre.

J. Self a analysé le rôle et les usages effectifs des modèles de l'élève dans les tuteurs existants. Il a ainsi pu identifier 20 fonctions qu'il a regroupées en six catégories : correctives, élaboratives, stratégiques, diagnostiques, prédictives et évaluatives.

Fonctions correctives : il s'agit déliminer des connaissances repérées comme erronées dans k. Les tâches à réaliser sont alors du type identification de la faute, mise en œuvre d'une correction directe ou indirecte, émission d'un contre-exemple, trace pas à pas pour révéler l'erreur, *etc...*

Fonction élaboratives : elles concernent le cas où les connaissances de l'élève sont correctes mais incomplètes (k inclus dans K). Le choix de la prochaine action à entrependre peut être dirigé par le profil de l'élève, une comparaison entre les connaissances de l'élève et de l'expert, ou bien encore être laissé à l'étudiant en offrant un menu.

Fonctions stratégiques : les fonctions précédentes ont un aspect local, à un instant donné de la session. Le modèle d'élève peut être employé pour prendre des décisions à un niveau stratégique plus global, quand on utilise une représentation des plans de session.

Fonctions diagnostiques : on peut faire du diagnostique sur le modèle de l'élève pour contribuer à l'établir, en demandant par exemple à l'élève ce qu'il pense, quelles sont ses croyances.

Fonctions prédictives : le modèle de l'élève peut être utilisé non seulement pour analyser, expliquer des actions passées, mais aussi pour prévoir des attitudes futures.

Fonctions évaluatives : le modèle de l'élève peut être utilisé pour faire une évaluation de l'élève; la composante historique H peut être utilisée pour fournir un résumé des performances de l'élève. On peut aussi utiliser k pour décrire ce que l'élève est supposé savoir.

Dans un TI (pour tuteur intelligent), les connaissances de l'élève sont comparées à celles du système pour déterminer celles qui sont manquantes, celles qui sont erronées et pour choisir les sujets de travail.

Dans [49], J. F. Nicaud considère, qu'actuellement deux constructions théoriques sont utilisées :

1. Les modèles de type expertise-partielle [32] : les connaissances de l'élève sont considérées comme formant un sous-ensemble de celles du système.
2. Les modèles différentiels [61] : les faiblesses de l'élève sont détectées par un raisonnement obtenu en utilisant les connaissances du système dans le contexte où se trouvait l'élève.

Actuellement, l'essentiel du travail se focalise sur l'étude des erreurs de l'utilisateur. Les erreurs des élèves peuvent se situer à tous les niveaux de connaissances : faits faux, règles de base déviées, mauvaises heuristiques, mécanisme de raisonnement erroné.

Ces études ont été faites, principalement dans des domaines mathématiques (par exemple DEBUGGY [8] ou LMS [61]).

En ce qui concerne l'utilisation des connaissances et des croyances pour générer des réponses coopératives, la finalité est souvent différente de celle suivie dans les TI. Nous avons commencé par décomposer ces informations en plusieurs sous-ensembles.

Le premier sous-ensemble est constitué par les croyances et les connaissances que le système obtient en interrogeant l'utilisateur, ou en observant ses réactions. Par exemple :

- L'utilisateur croit que voyager en avion est dangereux
- L'utilisateur sait que pour prendre l'avion, il faut se rendre à la porte d'embarquement une demi-heure avant le départ.

- L'utilisateur sait que le Concorde traverse l'Atlantique en deux heures et demi et que c'est un vol cher.
- Tous les utilisateurs savent que les vols ont un prix

Ces informations peuvent être associées à des stéréotypes. Par exemple, on pourra associer, avec un bon degré de vraisemblance, au stéréotype représentant les personnes qui n'ont jamais pris l'avion, l'information *"l'utilisateur croit que voyager en avion est dangereux"*.

Ces informations peuvent être utilisées, par le système de génération de réponses coopératives, pour fournir à l'utilisateur des informations supplémentaires. Ainsi, supposons que l'utilisateur sait que le Concorde est un vol cher et rapide. Dans ces conditions, si le Concorde est une solution d'une requête, il suffit de l'indiquer à l'utilisateur. Il est ensuite inutile de lui indiquer qu'il s'agit d'un vol cher, car il peut déduire lui-même cette information.

Un autre sous-ensemble est constitué par les informations que le système fournit à l'utilisateur en répondant à ses questions. Le système doit également tenir compte des déductions que l'utilisateur peut réaliser en prenant connaissance des réponses qui lui sont fournies. Par exemple, si l'utilisateur formule la requête suivante :

Q : Pour les vols
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = New-York
Heure-de-départ appartient à [8 00, 12 00]
Fournir la valeur de l'attribut : Heure-de-départ

et si le système indique que AF001 est une entité solution partant à 10 heures, alors l'utilisateur sait que AF001 est un Vol qui satisfait les conditions de la requête. Il peut donc déduire que la ville de départ de ce vol est Paris et que sa ville d'arrivée est New-York.

Ici encore, ces connaissances peuvent être utilisées par le système de génération de réponses coopératives. En particulier, elles permettent de ne pas fournir des informations supplémentaires redondantes, c'est-à-dire des informations qui seraient connues de l'utilisateur. Ainsi dans l'exemple ci-dessus, lorsqu'on répond à la question Q, il est inutile de fournir comme information supplémentaire la ville de départ du vol AF001, puisque cette information est imposée imposée par l'utilisateur (et donc connue de lui).

Enfin, le système peut également supposer que la plupart des utilisateurs, lorsqu'ils interrogent un système d'information, possèdent un ensemble de règles de "bon sens". Par exemple, si aucun fait ne les contredit, un utilisateur fera les hypothèses suivantes :

- Un vol fonctionne du 1 Janvier au 31 Décembre.
- Un vol fonctionne tous les jours de la semaine.
- Un vol est sans supplément.

Ces règles sont supposées "par défaut", mais l'utilisateur sait qu'elles peuvent avoir des exceptions. Les faits qui expriment ces exceptions sont des faits intéressants, et il faut les fournir à l'utilisateur.

Les croyances par défaut ci-dessus sont supposées par beaucoup d'utilisateurs, et peuvent être rattachées au stéréotype de l'utilisateur "standard".

Remarquons tout de suite que les informations concernant la croyance et la connaissance ne peuvent pas, en général, s'exprimer directement en logique du premier ordre. Une façon classique de représenter ces informations consiste à introduire une logique modale (logique épistémique [38]). Si l'on veut représenter ces logiques en utilisant un méta-langage (*Savoir* et *Croire* ne sont plus des modalités mais des méta-prédicats), alors, comme D. Perlis le montre dans [51], pour éviter les problèmes d'inconsistance, on est obligé de se limiter à une logique simple sans "nested belief". On ne pourra donc pas représenter une croyance complexe comme par exemple :

L'utilisateur sait que le système croit qu'une personne agée croit que prendre l'avion est dangereux.

D'autre part, il est important de remarquer que la plupart des logiques épistémiques développées font l'hypothèse que l'utilisateur est omniscient (c'est-à-dire, l'utilisateur peut déduire de sa base de connaissances, tous les théorèmes possibles). Une telle hypothèse ne permet pas de représenter la notion d'oubli ni celle de raisonnement "limité". Dans le cadre de la génération de réponses coopératives l'hypothèse de l'omniscience est donc probablement trop forte, et il serait intéressant d'étudier de nouvelles logiques pour lesquelles cette hypothèse n'est plus faite (Cf. [29] par exemple).

Dans la suite de ce travail, nous n'avons pas essayé de représenter ni d'utiliser les connaissances et les croyances de l'utilisateur. Toutefois, nous proposons un formalisme particulier permettant de représenter les règles de "bon sens" et nous montrons comment ces informations sont utilisées pour fournir des réponses coopératives.

3.6.5 Reconnaissance des intentions

Un dernier ensemble d'information faisant partie du modèle de l'utilisateur est constitué par les informations représentant les intentions de l'utilisateur, c'est-à-dire le but qui l'amène à utiliser le système. Par exemple :

- L'utilisateur veut faire un voyage touristique en Grèce au mois d'août.
- L'utilisateur veut faire un voyage d'affaires à New-York pendant la première semaine de septembre.

Ces informations peuvent être fournies explicitement par l'utilisateur ou bien plutôt, comme on le montre dans [3], déduite par le système d'après les questions que pose l'utilisateur. Dans la partie précédente, nous avons montré, en décrivant l'approche suivie dans [3], comment les intentions de l'utilisateur sont utilisées pour fournir des

réponses coopératives. Ici encore, on peut remarquer que ces informations sur l'utilisateur ne s'expriment pas non plus, en général, en logique du premier ordre "classique". Dans [17], P. Cohen et H. Levesque présentent une logique modale permettant de représenter cette notion d'intention.

En fait, pour le moment, les informations concernant l'utilisateur utilisées dans le système pour fournir des informations supplémentaires sont celles qui peuvent être formalisées en logique du premier ordre (les stéréotypes en particulier), ainsi que les règles de "bon sens" que possèdent tout utilisateur. Nous n'avons pas étudié le problème consistant à formaliser et à utiliser les informations sur la connaissance et les croyances d'un utilisateur particulier, ainsi que sur ses intentions.

3.7 Représentation du savoir-faire d'un expert en coopération

Les informations que nous manipulons dans le système peuvent être divisées en deux parties :

1. Les informations contenues dans la base de données.
2. Les connaissances expertes représentant le savoir-faire d'une personne ayant l'habitude de fournir des réponses coopératives.

Ces connaissances expertes requises pour transformer les requêtes peuvent également être décomposées en deux parties contenant des faits ou des règles.

3.7.1 Les faits

Les faits concernent :

1. Les informations indiquant les types, les attributs et les relations de la base de données.
2. La structure des types d'entités
3. L'association entre Attribut et "*Thème*"
4. La structure sur les *Thèmes*
5. La représentation de l'utilisateur

Les informations contenues dans 1-4 constituent la représentation de haut niveau de la base de données.

3.7.2 Les règles

Les règles définissent dans quel contexte certaines parties d'une requête peuvent être transformées, et comment effectuer cette transformation. Le lien entre la partie initiale et la partie transformée de la requête exprime que la partie transformée peut fournir des informations supplémentaires pertinentes. En utilisant cette notion de pertinence, la forme générale des règles est :

Si Contexte
Alors Pertinent (Partie-initiale, Partie-transformée)

Ces règles seront appelées "règles de pertinence". Dans la suite, on sera amené à distinguer les règles de pertinences qui permettent de transformer le sujet, les conditions, ou les informations demandées de la requête. Toutefois, pour ces trois types de transformations différentes, les règles utilisées ressembleront toujours au schéma général ci-dessus.

Un contexte peut être défini par :

1. Des informations concernant la requête, c'est-à-dire :
   - Un type d'entité qui apparait dans la partie "Sujet" de la requête, ou
   - Un attribut qui apparait dans les "Conditions" ou les "Informations Demandées".
2. Les informations concernant l'utilisateur.

Le contenu de la base de connaissances, et plus particulièrement les règles de pertinence, dépendent fortement du domaine d'application. Par exemple :

RT7 : **Si** une requête concerne les Vols,
Et le prix du vol est très élevé,
Alors la valeur de l'attribut Prix doit être fournie.

n'est valable que dans le domaine d'une agence de voyage.

Nous avons toutefois, dégagé des lois générales de coopération qui permettent de définir des règles indépendantes du domaine. Par exemple :

Si une question concerne des entités de type E,
Et elle contient un attribut A donné,
Et cet attribut est associé au *thème* T,
Alors ce *thème* T est intéressant pour cette question

D'autre part, il existe également des règles plus générales, ne dépendant pas du domaine, nécessaires pour définir comment une requête peut être transformée en une requête fournissant des informations supplémentaires. Ces règles générales sont appelées par la suite "règles de transformation". "L'exécution" des règles de transformation et des règles de pertinence par un processus de déduction fournit les requêtes transformées.

3.8 Organisation d'ensemble du système

Pour résumer, la démarche générale suivie par un serveur pour générer des réponses coopératives est la suivante : il commence par considérer la requête, et utilise la représentation de l'utilisateur et des règles exprimant le savoir-faire en matière de coopération pour déterminer les informations intéressant l'utilisateur; il utilise ensuite une représentation de haut niveau du contenu de la base de données pour transformer la requête et pour générer une (ou plusieurs) requêtes qui fournissent des informations supplémentaires pertinentes. Cette démarche peut être schématisée par la figure 3.1.

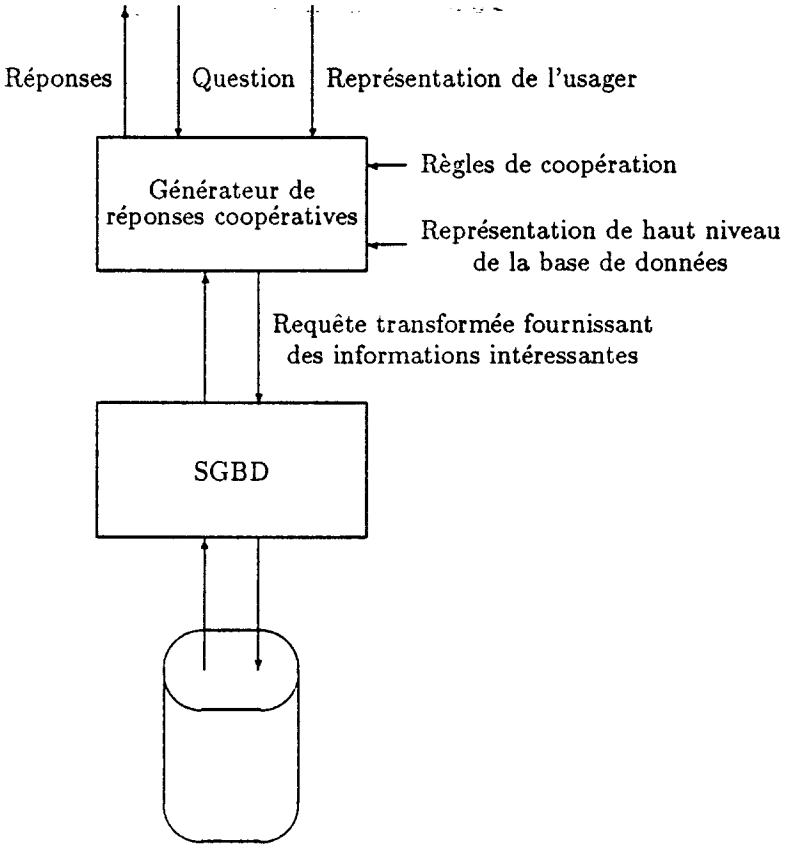


Figure 3.1: Organisation d'ensemble du système

Partie II

Représentation des connaissances de la base de données

Chapitre 4

Représentation des connaissances : niveau objet / niveau méta

Dans la partie précédente, nous avons présenté informellement un langage qui nous semble commode pour représenter le contenu de la base de connaissances ainsi que l'expression des requêtes. Nous nous intéressons maintenant à la définition précise de la sémantique de ce langage. Nous essayons de définir cette sémantique dans le cadre de la logique formelle.

Toutefois, même dans un cadre logique, il reste plusieurs manières de formaliser la connaissance. L'idée de base dans le formalisme proposé ici, comme cela a été suggéré par R. Kowalski dans [44], est de n'utiliser que la logique du premier ordre standard (FOL en abrégé pour First Order Logic). Toutefois, les notions introduites dans FOL ont une sémantique pauvre. Par exemple, les notions de type d'entité, d'attributs d'entités, de relation entre entités ainsi que d'attributs de relation sont toutes représentées par des prédicats dans FOL ; et en représentant ces notions par des prédicats, on perd une partie importante de leur sémantique. La sémantique attachée à chaque notion apparait, dans la partie précédente, lorsque nous présentons les différentes façons de considérer, par exemple, un type d'entité ou un attribut, dans la représentation de la requête, dans les règles de pertinence ou dans les règles de transformation.

Notre approche consiste à prendre en compte cette sémantique dans une méta-théorie, représentée également dans FOL. Dans cette méta-théorie sont représentés les faits et les régles définissant la sémantique des prédicats. Ainsi, nous distinguons deux niveaux de langage :

- Le niveau objet, dans lequel sont représentées la base de données et les requêtes; cette base de données peut contenir également des règles, si nous sommes dans le contexte d'une base de données déductive, mais ces règles ne doivent jamais être confondues avec les règles de la base de connaissances définies dans le chapitre 1.

A ce niveau, tous les prédicats ont un statut identique.

- Le niveau méta, dans lequel est représentée la sémantique qui ne peut s'exprimer au niveau objet; en particulier, la base de connaissances et les règles de transformation de la requête sont représentées à ce niveau.

Cette séparation de la connaissance en plusieurs niveaux est la démarche suivie par A. Bowen et R. Kowalski dans [5]. Nous rappelons dans l'annexe A les problèmes théoriques que soulève cette approche.

Dans cette partie, nous présentons formellement le langage objet et le métalangage que nous utilisons et montrons comment les connaissances que nous manipulons sont représentées dans ces langages.

4.1 Le langage objet

Commençons par définir le langage objet. Comme d'habitude, le langage objet L est défini par une paire (A, W) où A est un alphabet de symboles et W est un ensemble de formules bien formées construites en utilisant les symboles de A. L'alphabet A contient les ensembles de symboles suivants :

- Des variables : elles seront représentées par des identificateurs minuscules (*x, y, etc.*).
- Des constantes : elles seront représentées par des identificateurs majuscules (*A, B, Paris, etc.*). Parmi ces constantes, il y a le symbole *NIL*.
- Des prédicats. Parmi ces prédicats, il y a un prédicat d'arité 2, l'égalité notée $=$.
- Des symboles de ponctuation : (,).
- Des symboles logiques : $\longrightarrow$ (implique), $\wedge$ (et), $\vee$ (ou), $\neg$ (négation).
- Des quantificateurs : $\forall$ (Quantificateur universel) et $\exists$ (Quantificateur existentiel).
- Un symbole de fonction binaire : *List*.

Remarquons, qu'une seule fonction (*List*) est incluse dans l'alphabet. En effet, il n'est pas utile de définir d'autres fonctions pour représenter les connaissances de la base.

Intuitivement, la fonction *List* est introduite pour représenter les objets structurés.

Avec un tel alphabet, on peut construire un ensemble de formules bien formées, de la façon suivante :

- Termes :
  - Une variable de A est un terme

- Une constante de A est un terme
- Si x est un terme et y est un terme, alors $List(x, y)$ est un terme. Pour simplifier, $List(x, y)$ sera noté $[x.y]$ et $List(x, NIL)$, $[x]$.

- Formules atomiques :
  - Si P est un prédicat n-aire de A et $t_1, t_2, .., t_n$ sont des termes, alors $P(t_1, .., t_n)$ est une formule atomique.

- Formules bien formées (fbf) :
  - C'est le plus petit ensemble tel que :
    1. Une formule atomique est une formule bien formée.
    2. Si W_1 et W_2 sonf des fbfs, alors $(W_1 \wedge W_2)$, $(W_1 \vee W_2)$, $(W_1 \longrightarrow W_2)$ et $(\neg W_1)$ sont des fbfs.
    3. Si x est une variable et W une fbf, alors $\forall(x)W$ et $\exists(x)W$ sont des fbfs.

On utilisera $\forall(x_1, x_2, ..., x_n)W$ (resp. $\exists(x_1, x_2, ..., x_n)W$) comme abréviation pour $\forall(x_1)(\forall(x_2)...(\forall(x_n)W))$ (resp. $\exists(x_1)(\exists(x_2)...(\exists(x_n)W))$)

4.1.1 Axiomatique du langage

Les axiomes de la théorie objet sont les axiomes du calcul des prédicats avec égalité :

A1 : $P \longrightarrow (Q \longrightarrow P)$

A2 : $(P \longrightarrow (Q \longrightarrow R)) \longrightarrow ((P \longrightarrow Q) \longrightarrow (P \longrightarrow R))$

A3 : $(\neg P \longrightarrow \neg Q) \longrightarrow ((\neg P \longrightarrow Q) \longrightarrow P)$

A4 : $(\forall(x)P(x)) \longrightarrow P(t)$ si t est libre pour x dans P

A5 : $(\forall(x)\ (P \longrightarrow Q)) \longrightarrow (P \longrightarrow (\forall(x)Q))$
si x n'est pas libre dans P

E1 : Réflexivité de l'égalité :

$$\forall(x)(x = x)$$

E2 : Commutativité de l'égalité :

$$\forall(x, y)((x = y) \longrightarrow (y = x))$$

E3 : Transitivité de l'égalité :

$$\forall(x, y, z)((x = y) \wedge (y = z) \longrightarrow (x = z))$$

E4 : Principe de substitution des équivalents :

Pour tout symbole de prédicat n-aire,
$\forall\,(x_1,...,x_n,y_1,...,y_n)$
$P(x_1,...,x_n) \wedge (x_1 = y_1) \wedge ... \wedge (x_n = y_n) \longrightarrow P(y_1,...,y_n)$

E5 : Axiome d'égalité pour le prédicat *List* :

$$\forall(x_1,x_2,y_1,y_2)(List(x_1,y_1) = List(x_2,y_2)) \longrightarrow (x_1 = x_2) \wedge (y_1 = y_2)$$

Les règles d'inférence sont les règles classiques suivantes :

Modus ponens :
De $\vdash P \longrightarrow Q$ et de $\vdash P$ on déduit $\vdash Q$

Généralisation :
De $\vdash P$ on déduit $\vdash \forall(x)P$

4.1.2 Sémantique du langage

La sémantique du langage est définie en terme de modèle. Une fonction d'interprétation I du langage du premier ordre $F = (A, W)$ est un couple (D, f).

1. D est appelé le domaine de I, et est un ensemble non vide d'individus.
2. f est une fonction d'interprétation :
   - Pour tout symbole de constante a : $f(a) = a'$, ou a' est un individu de D.
   - Pour la fonction *List*, f assigne une application L de $(D \times D)$ dans D.
   - Pour tout symbole de prédicat n-aire P :
     $f(P)$ est un ensemble de n-uplets d'éléments de D.

Une assignation AS est une fonction de l'ensemble de symboles de variables vers D.

La valuation des formules sous l'interprétation I et l'assignation AS est définie par :

- $V_{I,AS}(P\,(t_1,...,t_n)) = 1 \longleftrightarrow < V_{I,AS}(t_1),...,V_{I,AS}(t_n) >$ est élément de $f(P)$
  avec $V_{I,AS}(t_i) = f(t_i)$ si t_i est un symbole de constante
  $V_{I,AS}(t_i) = AS(t_i)$ si t_i est un symbole de variable
  $V_{I,AS}(List(t_1,t_2) = L(V_{I,AS}(t_1), V_{I,AS}(t_2))$
- $V_{I,AS}(F_1 \wedge F_2) = 1 \longleftrightarrow V_{I,AS}(F_1) = 1$ et $V_{I,AS}(F_2) = 1$
- $V_{I,AS}(F_1 \vee F_2) = 1 \longleftrightarrow V_{I,AS}(F_1) = 1$ ou $V_{I,AS}(F_2) = 1$
- $V_{I,AS}(\neg F) = 1 \longleftrightarrow V_{I,AS}(F) = 0$
- $V_{I,AS}(\forall x F) = 1 \longleftrightarrow$ pour tout élément d de D, $V_{I,AS[x/d]}(F) = 1$
- $V_{I,AS}(\exists x F) = 1 \longleftrightarrow$ il existe un élément d de D tel que $V_{I,AS[x/d]}(F) = 1$

avec $AS[x/d]$ l'assignation qui coïncide partout avec AS, sauf en x où elle assigne l'individu d à x.

Une interprétation $I =< D, f >$ est un modèle d'une formule F si et seulement si $V_{I,AS}(F) = 1$ pour toute assignation.

Une formule de F est valide si et seulement si toute interprétation en est un modèle.

4.2 Le métalangage

Définissons maintenant le métalangage. Le métalangage M est défini par une paire (A', W') avec des notations similaires à celles du langage objet. L'alphabet A' contient les ensembles de symboles suivants :

- Des (méta) variables : comme les variables du langage objet, elles seront représentées par des identificateurs minuscules (x, y, etc).
- Des constantes. Parmi elles :
  - les codes de constantes du langage objet. Ceux-ci seront représentées par des identificateurs majuscules entre guillemets ("A", "B", etc).
  - les codes de variables du langage objet. Ceux-ci seront représentées par des identificateurs minuscules entre guillemets ("x", "y", etc).
  - le code de la fonction *List*.
  - les codes de prédicats que l'on notera "P", "Q", etc.
  - les codes de symboles logiques et de ponctuation que l'on notera "$\wedge$", "$\vee$", "$\exists$", "(", etc...
  - on a également les constantes *Vrai, Faux, Nil.*
- Un symbole de fonction binaire : *List.*
- Des prédicats.
- Les mêmes symboles logiques et de ponctuation ainsi que les mêmes quantificateurs que dans le langage objet.

Les règles de construction des termes, des formules atomiques et des formules bien formées du métalangage sont les memes que celles définies pour le langage objet.

Ce langage permet de coder toutes les expressions du langage objet défini dans la section précédente. Ainsi, on peut coder l'expression $\neg P(A, B) \wedge Q(C)$ par le terme fonctionnel :

$$[\text{“}\neg P(A, B)\text{”}, \text{“}\wedge\text{”}, \text{“}Q(C)\text{”}]$$

avec "$\neg P(A, B)$" représentant en fait le terme : ["$\neg$", "$P(A, B)$"]

et "$P(A,B)$" : ["P", "A", "B"]

Par la suite, le terme fonctionnel représentant le code de la formule $\neg P(A,B) \wedge Q(C)$ sera denoté par cette formule entre guillemets : "$\neg P(A,B) \wedge Q(C)$".

Il est important de remarquer que de tels termes peuvent contenir des méta variables pour représenter une formule objet qui n'est pas entièrement spécifiée. Dans ce cas, nous ajoutons les marques < et > autour de l'expression qui représente une méta variable. Par exemple, dans l'expression :

$$"P(< x >, B)"$$

x est une méta variable. Ceci permet d'éviter la confusion avec "$P(x,b)$", où x est une variable objet.

4.2.1 Axiomatique du métalangage

Dans la méta théorie, on retrouve les mêmes schémas d'axiome que ceux définis pour la théorie objet (axiomes A1–A5 et E1–E4) et les mêmes règles d'inférence (Modus Ponens et Généralisation).

Formules bien formées

Pour définir, dans la méta théorie, la structure syntaxique d'une formule bien formée du langage objet, on introduit dans la méta théorie les axiomes suivants (Cf. [31]) :

- Un terme est soit une constante objet, soit une variable, soit une terme fonctionnel :

$$\forall(x) Terme(x) \longleftrightarrow Constante(x) \vee Variable(x) \vee Funexpr(x)$$

- Une *termlist* est une liste ordonnées de termes :

$$\begin{aligned} \forall(l) Termlist(l) \quad \longleftrightarrow \quad & (l = Nil) \vee \\ & \exists(x,l')(l = list(x,l') \wedge Terme(x) \wedge Termlist(l')) \end{aligned}$$

- Un terme fonctionnel est une expression construite à l'aide d'une fonction et d'une liste de termes (dans cette définition, on ne tient pas compte de l'arité de la fonction) :

$$\forall(f,l) Funexpr([f.l]) \longrightarrow (Fonction(f) \wedge Termlist(l))$$

- Une formule atomique est une expression construite à l'aide d'un prédicat et d'une liste de termes (on ignore toujours l'arité) :

$$\forall(p,l) Atome([p.l]) \longrightarrow (Prédicat(p) \wedge Termlist(l))$$

- Enfin, la règle définissant une formule bien formée du langage objet est la suivante :

$$\begin{aligned}\forall(x)fbf(x) \longleftrightarrow\ & Atome(x) \vee \\ & \exists(z_1,z_2)(x = \text{“} < z_1 > \wedge < z_2 > \text{”} \wedge fbf(z_1) \wedge fbf(z_2)) \vee \\ & \exists(z_1,z_2)(x = \text{“} < z_1 > \vee < z_2 > \text{”} \wedge fbf(z_1) \wedge fbf(z_2)) \vee \\ & \exists(z_1,z_2)(x = \text{“} < z_1 > \longrightarrow < z_2 > \text{”} \wedge fbf(z_1) \wedge fbf(z_2)) \vee \\ & \exists(z)(x = \text{“}\neg < z > \text{”} \wedge fbf(z)) \vee \\ & \exists(v,z)(x = \text{“}\forall(< v >) < z > \text{”} \wedge Variable(v) \wedge fbf(z)) \vee \\ & \exists(v,z)(x = \text{“}\exists(< v >) < z > \text{”} \wedge Variable(v) \wedge fbf(z))\end{aligned}$$

Cette règle exprime seulement que les formules bien formées constituent le plus petit ensemble tel que (Cf. page 75) :

1. Une formule atomique est une formule bien formée.
2. Si W_1 et W_2 sonf des fbfs, alors $(W_1 \wedge W_2)$, $(W_1 \vee W_2)$, $(W_1 \longrightarrow W_2)$ et $(\neg W_1)$ sont des fbfs.
3. Si x est une variable et W une fbf, alors $\forall(x)W$ et $\exists(x)W$ sont des fbfs.

Principe de résolution

Nous pouvons maintenant montrer comment construire une méta théorie qui formalise le Principe de Résolution.

En fait, la méthode est la même que celle définie dans [5] où l'on définit un prédicat *Demo* vérifiant la propriété suivante :

> Pour tout ensemble fini de formules A de L et
> Pour toute formule B de L,
> $A \vdash_L B$ ssi $Pr \vdash_M Demo(\text{“}A\text{”}, \text{“}B\text{”})$

où L est le langage objet, M est le métalangage et Pr la méta théorie.

Pour cela, il suffit que les axiomes associés au prédicat *Demo* définissent une méthode de résolution complète pour l'ensemble des clauses A de la théorie objet.

Par exemple, dans [5], les axiomes de Pr sont les règles D1–D2 (Cf. page 180) qui définissent la *"lush resolution"*, qui est une méthode de résolution complète dans le cas où l'on se limite à un ensemble de clauses de Horn. On trouvera dans l'annexe A, un rappel des travaux presentés dans cet article.

Dans [31], les axiomes de Pr associés au prédicat *Demo* définissent le principe de résolution de Robinson [56] :

$$\forall(d,r)Demo(d,r) \longleftrightarrow Member(r,d) \vee \exists(p,q)(Demo(d,p) \wedge Demo(d,q) \wedge Resolvent(p,q,r))$$

où r est une formule mise sous forme clausale et d est un ensemble de formules sous forme clausale.

On a également :

$$\begin{aligned}\forall(c,d,r)Resolvent(c,d,r) \longleftrightarrow\ & \exists(x,y,c_1,d_1,r_1) \\ & Member(x,c) \wedge Member(\text{“}\neg < y > \text{”},d) \wedge \\ & Mgu(x,y,s) \wedge \\ & Delete(x,c,c_1) \wedge Delete(\text{“}\neg < y > \text{”},d,d_1) \wedge \\ & Append(c_1,d_1,r_1) \wedge Subst(r_1,s,r)\end{aligned}$$

Cette règle indique que si un littéral x est une partie d'une clause et y est une partie d'une autre clause et s'il y a un unificateur le plus général (*Mgu* pour "Most general unifier") pour x et y, alors la résolvante des deux clauses est construite en détruisant les littéraux complémentaires, en liant les littéraux restants et un appliquant l'unificateur.

En fait, le principe de résolution, appliqué à un ensemble de formules mises sous forme clausale, n'est pas complet en génération mais seulement en réfutation :

$$\begin{aligned}\forall(d,p)Provable(d,p) \longleftrightarrow\ & Clauses(\text{“}\neg < p > \text{”},p') \wedge \\ & Append(p',d,d') \wedge Demo(d',[])\end{aligned}$$

Cette règle indique qu'une formule est démontrable en utilisant le principe de résolution si et seulement si la clause vide peut être dérivée des clauses de la théorie et de celles qui résultent de la mise sous forme clausale de la négation de cette formule.

4.2.2 Sémantique du métalangage

De la même façon que pour le langage objet L, la sémantique du métalangage M est définie en terme de modèle. En gardant les mêmes notations que pour le langage objet L, on définit une interprétation I' du langage $M = (A', W')$ par la donnée d'un couple (D', f') où D' est le domaine de I' et f' est une fonction d'interprétation. En particulier, on donnera l'interprétation de tous les méta prédicats de M.

Les notions de valuation et de modèle d'une formule seront également définis de la même façon que pour le langage objet.

Soit maintenant un langage objet L. On associe L et M en se donnant :

- Une fonction de codage *code* qui associe à chaque expression de L un terme sans variable libre de M.

  La fonction *code* doit satisfaire les conditions suivantes :

  - Soit A une expression de L. Si $A' = code(A)$ alors :
    * A constante de $L \longleftrightarrow A' \in f'(Constante)$
    * A variable de $L \longleftrightarrow A' \in f'(Variable)$
    * A fonction de $L \longleftrightarrow A' \in f'(Fonction)$
    * A prédicat de $L \longleftrightarrow A' \in f'(Prédicat)$
    * A terme de $L \longleftrightarrow A' \in f'(Terme)$
    * A fbf de $L \longleftrightarrow A' \in f'(fbf)$

- Une représentation de $\vdash_L$ par un symbole de prédicat *Demo* dans le contexte d'un ensemble d'axiomes propres Pr de M qui doit satisfaire la condition suivante :

  - $A \vdash_L B$ ssi $Pr \vdash_M Demo(\text{“}A\text{”}, \text{“}B\text{”})$

4.3 Représentation du modèle Entité / Relation

L'objectif de cette partie est de montrer comment nous représentons les connaissances associées à un modèle entité-relation dans le langage présenté ci-dessus. L'idée générale est de représenter la base de données comme une théorie objet (ou un modèle de cette théorie), alors que la sémantique associée à ces données est représentée dans la méta-théorie (Cf. [21]).

4.3.1 Représentation des entités et des classes d'entités

Niveau objet

Toutes les entités ont un nom qui permet de les identifier de façon unique. Ce nom sera en fait une constante du langage objet.

Les entités sont classées en “ensemble d'entités”. Pour tester l'appartenance d'une entité particulière à l'ensemble, on associe un prédicat unaire à chaque ensemble d'entités.

Ainsi, l'information “v_1 est le nom d'une entité de type *Vol*” est représentée dans la théorie objet par le fait suivant :

- $Vol\,(v_1)$

Niveau méta

Au niveau méta, pour indiquer qu'un prédicat du langage objet représente un type d'entité, on utilise le prédicat :

- *Type-entité* (x)

Par exemple, pour indiquer que le prédicat objet *Vol* représente un type d'entité, il suffit d'introduire dans la méta-théorie le fait suivant :

- *Type-entité* (Vol)

Il faut bien remarquer que dans cet axiome *Vol* représente en fait le codage du prédicat objet *Vol.*

4.3.2 Types

Dans notre modèle, tous les éléments ont un type et il y a deux sortes de types : le type *Entité* et le type *Valeur d'attribut.*

Le type *Entité* a déjà été introduit dans la partie précédente et permet de classer les entités en "ensembles d'entités".

Le type *Valeur d'attribut* est introduit pour typer les attributs des entités. Ceci permet de classer les valeurs des attributs en "ensembles valeurs" (Cf. [13]).

Niveau objet

Au niveau objet, chaque type est représenté par un prédicat unaire. Par exemple :

- *Vol* (x), *Heure* (x), *Ville* (x)

A ce niveau, rien ne permet de distinguer un type *Entité* d'un type *Valeur d'attribut.*

Niveau méta

Au niveau méta, on distingue les types *Entités* des types *Valeurs d'attribut* en utilisant les méta-prédicats :

- *Type-entité* (x)
- *Type-attribut* (x)

Exemple :

- *Type-entité* (Vol)
- *Type-attribut* $(Heure)$
- *Type-attribut* $(Ville)$

Encore une fois, remarquons que dans ces axiomes, *Vol, Heure* et *Ville* représentent en fait le codage des prédicats *Vol, Heure* et *Ville.*

4.3.3 Types des arguments des prédicats

Niveau objet

Au niveau objet, on a des axiomes pour définir le type des arguments de chaque prédicat.

Ainsi, si P est un prédicat n-aire ayant pour type $T_1, ..., T_n$, on introduira dans la théorie objet l'axiome suivant :

- $\forall(x_1, .., x_n) P(x_1, ..., x_n) \longrightarrow T_1(x_1) \wedge ... \wedge T_n(x_n)$

Exemple :

- $\forall(x, y)$ *Heure-de-départ* $(x, y) \longrightarrow$ *Vol* $(x) \wedge$ *Heure* (y)
- $\forall(x, y)$ *Réservation* $(x, y) \longrightarrow$ *Vol* $(x) \wedge$ *Personne* (y)
- $\forall(x, y, z)$ *Numéro-de-place* $(x, y, z) \longrightarrow$ *Vol* $(x) \wedge$ *Personne* $(y) \wedge$ *Place* (z)

Niveau méta

Au niveau méta, ces informations sont représentées en utilisant le méta prédicat :

- *Type* (x, y)

où x représente le code d'un prédicat, et y est une liste de codes représentant des types, chaque type dans la liste correspondant à l'argument du prédicat ayant le même rang.

Exemple :

- *Type* (*Heure-de-départ*,$[Vol, Heure]$)
- *Type* (*Réservation*,$[Vol, Personne]$)
- *Type* (*Numéro-de-place*,$[Vol, Personne, Place]$)

où $[Vol, Heure]$ est une notation pour représenter la liste : *Vol, Heure.*

4.3.4 Représentation des attributs

Niveau objet

Les informations concernant une entité s'expriment à l'aide d'un ensemble d'attributs. Au niveau objet, à chaque attribut est associé un prédicat.

Ainsi, dans l'exemple du chapitre 2, à un *Vol* étaient associés les attributs *Ville-de-départ, Ville-d'arrivée, Heure-de-départ, Heure-d'arrivée, Prix, Tarification et Opère.* Pour représenter une entité de type *Vol*, on introduira donc dans le langage objet les prédicats suivants :

- *Ville-de-départ* (x, y)
- *Ville-d'arrivée* (x, y)
- *Heure-de-départ* (x, y)
- *Heure-d'arrivée* (x, y)
- *Prix* (x, y)
- *Tarification* (x, y, z)
- *Opère* (x, y)

Pour chaque type d'entité E et pour les attributs $A_1, ..., A_n$ définies pour ce type d'entité, on a un axiome au niveau objet exprimant que chaque entité de ce type a une valeur pour les attributs $A_1, ..., A_n$. Cet axiome a la forme suivante :

- $\forall(x)E(x) \longrightarrow \exists(y_1, .., y_n)A_1(x, y_1) \wedge ... \wedge A_n(x, y_n)$

où chaque y_i représente un t-uplet de variables dépendant de l'arité du prédicat A_i.

Si l'on considère que les attributs $A_1, ..., A_n$ caractérisent les entités de type E, alors on pourra également introduire l'axiome suivant :

- $\forall(x, y_1, ..., y_n)A_1(x, y_1) \wedge .. \wedge A_n(x, y_n) \longrightarrow E(x)$

Niveau méta

Dans la méta-théorie, le fait qu'à une entité E est associée un ensemble d'attributs $A_1, ..., A_n$ est représenté en utilisant le méta-prédicat :

- *Attribut* (x, y)

et en introduisant dans la méta théorie les faits suivants :

- *Attribut*(A_1, E)
- ...
- *Attribut*(A_n, E)

Ainsi, dans l'exemple ci-dessus, on aurait dans la méta-théorie les faits suivants :

- *Attribut* (*Ville-de-départ*,Vol)
- *Attribut* (*Ville-d'arrivée*,Vol)
- *etc.*

4.3.5 Représentation des relations

Niveau objet

Une relation R entre plusieurs entités e_i de types E_i est représentée dans le langage objet par le prédicat n-aire $R(x_1, ..., x_n)$. Comme nous l'avons déjà indiqué, on indiquera le type des arguments du prédicat R en introduisant dans la théorie objet l'axiome suivant :

- $\forall(e_1, ..., e_n) R(e_1, ..., e_n) \longrightarrow E_1(e_1) \wedge ... \wedge E_n(e_n)$

On peut ainsi définir les relations *Réservation* entre un *Vol* et une *Personne* et *Mariage* entre deux *Personnes*, en introduisant dans le langage objet les prédicats :

- *Réservation* (x, y), *Mariage* (x, y)

et, dans la théorie objet, les axiomes :

- $\forall(x, y)$ *Réservation* $(x, y) \longrightarrow$ *Vol* $(x) \wedge$ *Personne* (y)
- $\forall(x, y)$ *Mariage* $(x, y) \longrightarrow$ *Personne* $(x) \wedge$ *Personne* (y)

On peut associer un attribut à une relation en introduisant dans le langage objet un nouveau prédicat.

Par exemple, on peut définir pour la relation *Réservation*, l'attribut *Numéro de place*, en introduisant dans le langage objet le prédicat :

- *Numéro-de-place* (x, y, z)

ainsi que l'axiome suivant dans la théorie objet :

- $\forall(x, y, z)$ *Numéro-de-place* $(x, y, z) \longrightarrow$ *Vol* $(x) \wedge$ *Personne* $(y) \wedge$ *Place* (z)

Enfin, l'association d'un "rôle" aux arguments d'une relation pour indiquer la fonction que l'entité joue dans la relation, revient, en fait, à définir de nouveaux types d'entités.

Ainsi, pour pouvoir considérer que dans la relation *Mariage* (x, y), x est le *Mari* et y l'*Epouse*, il suffit d'introduire dans le langage objet, les nouveaux types :

- *Mari* (x), *Epouse* (x)

et dans la théorie objet, l'axiome :

- $\forall(x, y)$ *Mariage* $(x, y) \longrightarrow$ *Mari* $(x) \wedge$ *Epouse* (y)

Niveau méta

Au niveau méta, pour indiquer qu'un prédicat objet n-aire R représente une relation entre n entités de types $E_1, ..., E_n$, on utilise le méta-prédicat :

- *Relation* (x)

et on introduit dans la méta-théorie les faits suivants :

- *Relation* (R)
- *Type* $(R, [E_1, ..., En])$

On peut également associer un attribut à une relation en utilisant le méta prédicat

- *Attribut* (x, y)

de la même façon que pour les attributs des entités.

Enfin, on exprimera que $r_1, ..., r_n$ sont les rôles des entités $E_1, ..., E_n$ dans la relation R, en introduisant dans la méta théorie :

- *Type-Entité* (r_1), ..., *Type-Entité* (r_n)
- *Type* $(R, [r_1, ..., r_n])$

ainsi que les faits indiquant que $r_1, ..., r_n$ sont respectivement des sous-types de $E_1, ..., E_n$ (Cf. section suivante).

Par exemple, pour les relations *Réservation* et *Mariage*, on introduit, dans la méta-théorie, les faits suivants :

- *Relation* (*Réservation*)
- *Type* (*Réservation*,[Vol, *Personne*])
- *Attribut* (*Numéro-de-place*,*Réservation*)
- *Relation* ($Mariage$)
- *Type* ($Mariage$, [$Mari$, *Epouse*])

ainsi que les faits indiquant que $Mari$ et *Epouse* sont des sous-types de *Personne*.

4.3.6 Structuration des entités : hiérarchie et héritage

Niveau objet

Définir une hiérarchie entre ensembles entités (on parle aussi de relation de généralisation dans la littérature) consiste en fait à introduire la relation d'inclusion entre ensembles d'entités.

Dans la théorie objet, le fait que toute entité de type E_1 est aussi une entité de type E_2 s'exprime simplement par l'axiome :

- $\forall(x)E_1(x) \longrightarrow E_2(x)$

Ainsi, les axiomes suivants indiquent que les *Vols* et les *Trains* sont des *Trajets*, et que les *Maris* et *Epouses* sont des *Personnes*.

- $\forall(x)Vol(x) \longrightarrow Trajet(x)$
- $\forall(x)Train(x) \longrightarrow Trajet(x)$
- $\forall(x)Mari(x) \longrightarrow Personne(x)$
- $\forall(x)Epouse(x) \longrightarrow Personne(x)$

Une fois introduit la relation de généralisation, on introduit souvent la notion d'héritage entre classe d'entités. Ainsi, si toute entité de type E_1 est aussi une entité de type E_2, alors toutes les propriétés définies pour les entités de type E_2 (attribut, relation, *etc.*) sont aussi valables pour les entités de type E_1.

Dans la théorie objet, cette notion d'héritage est entièrement contenue dans le processus de déduction. Par exemple l'héritage des attributs s'exprime au niveau objet par la dérivation suivante :

- Soit A_1 l'axiome :
  - $\forall(x)E_1(x) \longrightarrow E_2(x)$

qui exprime dans la théorie objet, que toute entité de type E_1 est aussi une entité de type E_2.

- Soit A_2 l'axiome :
  - $\forall(x)E_2(x) \longrightarrow \exists(y)A(x,y)$

qui exprime dans la théorie objet, que A est un attribut des entités de la classe E_2.

Alors de $\{A_1, A_2\}$ on peut déduire le théorème :

- $\forall(x)E_1(x) \longrightarrow \exists(y)A(x,y)$

qui exprime dans la théorie objet, que A est aussi un attribut de la classe d'entités E_1 (héritage des attributs).

Nous aimerions également pouvoir exprimer que deux classes d'entités E_1 et E_2 sont disjointes (c'est-à-dire n'ont pas d'élément commun). Ceci s'exprime simplement dans la théorie objet par l'axiome :

- $\neg\exists(x)(E_1(x) \wedge E_2(x))$

Ainsi, les axiomes suivants indiquent que les *Trains* et les *Vols* sont disjoints, de même que les *Trajets* et les *Personnes* ainsi que les *Maris* et les *Epouses*.

- $\neg\exists(x)(Vol(x) \wedge Train(x))$
- $\neg\exists(x)(Trajet(x) \wedge Personne(x))$
- $\neg\exists(x)(Mari(x) \wedge Epouse(x))$

Niveau méta

Au niveau méta, pour représenter que deux types d'entités sont inclus ou disjoints, nous introduisons dans la méta-théorie les prédicats :

- $ISA(x, y)$
- $DIS(x, y)$

Exemple :

- $ISA(Vol, Trajet)$
- $ISA(Train, Trajet)$
- $ISA(Mari, Personne)$
- $ISA(Epouse, Personne)$
- $DIS(Vol, Train)$
- $DIS(Trajet, Personne)$
- $DIS(Mari, Epouse)$

Pour traduire la sémantique des prédicats *ISA* et *DIS*, on doit introduire dans la méta théorie les axiomes suivants, qui expriment la réflexivité et la transitivité du prédicat *ISA*, la symétrie du prédicat *DIS*, et le lien existant entre ces deux prédicats :

R1 : $\forall(e)Type\text{-}Entité(e) \longrightarrow ISA(e, e)$

R2 : $\forall(e,e',e'')ISA(e,e') \wedge ISA(e',e'') \longrightarrow ISA(e,e'')$

R3 : $\forall(e,e')DIS(e,e') \longrightarrow DIS(e',e)$

R4 : $\forall(e,e',e'')DIS(e,e'') \wedge ISA(e',e'') \longrightarrow DIS(e,e')$

Il n'est pas très difficile de démontrer que l'ensemble d'axiomes $S = \{R1, R2, R3, R4\}$ de la méta théorie traduit effectivement la signification des prédicats *ISA* et *DIS*. Ainsi, ces axiomes sont valides et complets dans le sens suivant des termes :

Complétude : Soit un ensemble d'axiomes *Obj-ax* au niveau objet ayant la forme : $(T(x) \longrightarrow T'(x))$ ou bien $\neg(\exists(x)T(x) \wedge T'(x))$ et l'ensemble d'axiomes correspondant *Méta-ax* au niveau méta ayant la forme : $ISA(T,T')$ ou bien $DIS(T,T')$. Alors pour tout théorème au niveau objet dérivable de l'ensemble *Obj-ax*, et ayant la même forme, il existe un théorème correspondant au niveau méta dérivable de *Méta-ax* et de S.

Validité : Pour chaque théorème de la forme $ISA(T,T')$ ou $DIS(T,T')$ dérivable de *Méta-ax* et S au niveau méta, il existe un théorème au niveau objet ayant la forme : $(T(x) \longrightarrow T'(x))$ ou bien $\neg(\exists(x)T(x) \wedge T'(x))$ et dérivable de *Obj-ax*.

De plus, si l'on ajoute, au niveau objet, pour chaque type d'entité, des axiomes de la forme $\exists(x)T(x)$ pour exprimer que chaque type d'entité est non vide, alors on doit ajouter au niveau méta l'axiome :

R5 : $\neg\exists(e,e')(ISA(e,e') \wedge DIS(e,e'))$

Avec l'axiome R5, les résultats ci-dessus de complétude et de validité peuvent être étendus aux théorèmes de la forme : $\neg ISA(T,T')$ et $\neg DIS(T,T')$.

La démonstration de ces résultats est donnée dans l'annexe B.

Pour exprimer l'héritage d'un attribut d'un type d'entité à un sous-type, on a également l'axiome suivant :

R6 : $\forall(a,t,t')Att(a,t) \wedge ISA(t',t) \longrightarrow Att(a,t')$

4.3.7 Représentation des objets structurés

La notion d'objet structuré (on parle aussi d'attribut complexe) permet de définir une nouvelle entité dont les constituants sont eux-mêmes des entités. Cette notion n'existe pas dans la version initiale du modèle entité-relation présentée dans [13], mais elle a été depuis introduite dans certaines réalisations ([50] par exemple) et permet d'enrichir considérablement le pouvoir d'expression du modèle entité-relation.

En ce qui nous concerne, nous n'entrons pas dans les détails et nous limitons notre étude des objets structurés à la présentation d'un exemple : l'objet *segment*.

Niveau objet

Au niveau objet, pour représenter dans notre formalisme logique, un objet structuré tel que l'objet *Segment*, il suffit d'introduire dans le langage objet les prédicats suivants :

- $Segment(x)$
- $Org(x, y)$
- $Ext(x, y)$

où Org et Ext représentent respectivement l'origine et l'extrémité du segment.

Pour représenter un *Point* on introduira les prédicats suivants :

- $Point(x)$
- $Abs(x, y)$
- $Ord(x, y)$

où Abs et Ord représentent respectivement l'abscisse et l'ordonnée du point.

et l'on aura, dans la théorie objet, les axiomes suivants :

- $\forall(x) Segment(x) \longrightarrow \exists(y_1, y_2) Org(x, y_1) \wedge Ext(x, y_2)$

avec éventuellement la réciproque si l'on considère que les attributs Ord et Ext définissent un *Segment*.

- $\forall(x, y) Org(x, y) \longrightarrow Segment(x) \wedge Point(y)$

et le même genre d'axiome pour *Ext*.

- $\forall(x) Point(x) \longrightarrow \exists(y_1, y_2) Abs(x, y_1) \wedge Ord(x, y_2)$

avec éventuellement la réciproque si l'on considère que les attributs Abs et Ord définissent un *Point*.

- $\forall(x, y) Abs(x, y) \longrightarrow Point(x) \wedge Réel(y)$

et le même genre d'axiome pour *Ord*.

Pour définir une instance de l'objet *Segment*, il suffit d'introduire, dans la théorie objet, les faits suivants :

- $Segment(S_1)$, $Org(S_1, P_1)$, $Ext(S_1, P_2)$

et

- $Point(P_1)$, $Abs(P_1, 2.5)$, $Ord(P_1, 3)$

ainsi que

- $Point(P_2)$, $Abs(P_2, 3)$, $Ord(P_2, 1.2)$

Niveau méta

Dans la méta-théorie, on aura les faits suivants :

- $Type\text{-}Entité(Segment)$
- $Type\text{-}Entité(Point)$
- $Attribut(Org, Segment)$
- $Attribut(Ext, Segment)$
- $Attribut(Abs, Point)$
- $Attribut(Ord, Point)$
- $Type(Org, [Segment, Point])$
- $Type(Ext, [Segment, Point])$
- $Type(Abs, [Point, Réel])$
- $Type(Ord, [Point, Réel])$

4.3.8 Représentation des ensembles et des listes

Nous étudions ici des cas particuliers d'objets structurés constitués par un ensemble (non ordonné) ou par une liste (ordonnée) de sous-objets. Ici encore, nous limitons notre étude à la présentation d'un exemple.

Pour définir un objet *Voyage* constitué d'une liste ordonnée d'entités *Trajet*, on introduit dans la théorie objet les axiomes suivants :

- $\forall\ (x, v_1, v_2)$
  $Trajet(x) \wedge$
  $Ville\text{-}de\text{-}Départ(x, v_1) \wedge$
  $Ville\text{-}d'Arrivée(x, v_2)$
  $\longrightarrow Liaison(v_1, v_2, [x])$

- $\forall\ (x_1, x_2, x_3, v_1, v, v_2, h_1, h_2)$
  $Trajet(x_1) \wedge$
  $Ville\text{-}de\text{-}Départ(x_1, v_1) \wedge$
  $Ville\text{-}d'Arrivée(x_1, v) \wedge$
  $Liaison(v, v_2, [x_2.x_3]) \wedge$
  $Heure\text{-}d'Arrivée(x_1, h_1) \wedge$
  $Heure\text{-}de\text{-}Départ(x_2, h_2) \wedge$
  $h_1 < h_2$
  $\longrightarrow Liaison(v_1, v_2, [x_1, x_2.x_3])$

La première règle indique qu'il y a une liaison entre les villes v_1 et v_2 empruntant le chemin $[x]$ s'il existe un trajet x de v_1 à v_2.

La seconde règle indique qu'il y a une liaison entre les villes v_1 et v_2 empruntant le chemin $[x_1, x_2.x_3]$ s'il existe un trajet entre v_1 et une troisième ville v, et une liaison entre v et v_2 et si de plus l'heure d'arrivée du trajet arrivant dans la ville v précède l'heure de départ de celui partant de v.

Une fois défini le prédicat *Liaison*, on peut introduire un objet structuré *Voyage* ayant pour attributs :

- $Ville\text{-}de\text{-}Départ(x, y)$
- $Ville\text{-}d'Arrivée(x, y)$
- $Heure\text{-}de\text{-}Départ(x, y)$
- $Heure\text{-}d'Arrivée(x, y)$
- $Prix(x, y)$
- $Etape(x, n, y)$

L'attribut $Etape(x, n, y)$ indique que la *n-ième* étape du voyage x emprunte le trajet y.

Pour cela, on introduit dans la théorie objet les axiomes suivants :

- $\forall(v_1, v_2, x) Liaison(v_1, v_2, x) \longrightarrow Voyage(x)$
- $\forall(x, x_1, v)$
  $Voyage(x) \wedge Premier(x, x_1) \wedge Ville\text{-}de\text{-}Départ(x_1, v)$
  $\longrightarrow Ville\text{-}de\text{-}Départ(x, v)$

ainsi que les axiomes similaires définissant *Ville-d'Arrivée, Heure-de-Départ* et *Heure-d'arrivée.*

- $\forall(x, p)$ *Voyage (x)* $\wedge$ *Somme-Prix (x, p)* $\longrightarrow$ *Prix (x, p)*
- $\forall(x, i, x_i)\ Voyage(x) \wedge Position(x_i, x, i) \longrightarrow Etape(x, i, x_i)$

avec

- *Somme-Prix*$(nil, 0)$
- $\forall\ (v_1, v_2, p_1, p_2, p)$
  Prix $(v_1, p_1) \wedge$ *Somme-Prix* $(v_2, p_2) \wedge p = p_1 + p_2$
  $\longrightarrow$ *Somme-Prix* $([v_1.v_2], p)$

Somme-Prix(l, p) calcule pour la liste de Trajet l, la somme des prix de chaque *Trajet.*

et

- $\forall(e, l)$*Position*$(e, [e.l], 1)$
- $\forall(e_1, e_2, l, n_1, n_2)$*Position*$(e_1, l, n_2) \wedge n_1 = n_2 + 1 \longrightarrow$ *Position*$(e_1, [e_2.l], n_1)$

Pour la liste ordonnée l, *Position*(x, l, i) sera démontrable si x est le *i-ième* élement de cette liste.

Nous ne présentons pas les axiomes de la théorie objet et de la méta-théorie indiquant que Voyage est une entité possédant les attributs *Ville-de-Départ, Ville-d'Arrivée, etc.*

4.4 Regroupement des attributs en *thèmes*

Les *thèmes* sont associés aux attributs, et permettent de représenter certains liens sémantiques entre les attributs. Cette notion n'est pas représentée au niveau objet, mais seulement au niveau méta. Pour représenter cette notion nous introduisons donc dans le métalangage les prédicats :

- *Topic*(x)
- *Att-Top*(x, y)

Dans *Att-Top* (x, y), x est un attribut ou une relation, et y est un *thème.*

Ainsi, en reprenant l'exemple type "agence de voyage" présenté dans le chapitre 1, on pourrait introduire les *thèmes* suivants :

- *Localisation*
- *Horaire*
- *Coût*
- *Condition-de-Validité*
- *Temps*

Ces *thèmes* sont définis comme des constantes du métalangage.

Pour exprimer que ces constantes sont des *thèmes*, on introduit dans la méta-théorie les faits suivants :

- *Topic(Coût)*
- *Topic(Temps)*
- *etc.*

On peut associer un *thème* aux attributs définis pour l'entité Vol en introduisant dans la méta-théorie les faits suivants :

- *Att-Top(Ville-de-Départ, Localisation)*
- *Att-Top(Ville-d'Arrivée, Localisation)*
- *Att-Top(Heure-de-Départ, Horaire)*
- *Att-Top(Heure-d'Arrivée, Horaire)*
- *Att-Top(Prix, Coût)*
- *Att-Top(Tarification, Coût)*
- *Att-Top(Opère, Condition-de-Validité)*
- *Att-Top(Période, Condition-de-Validité)*

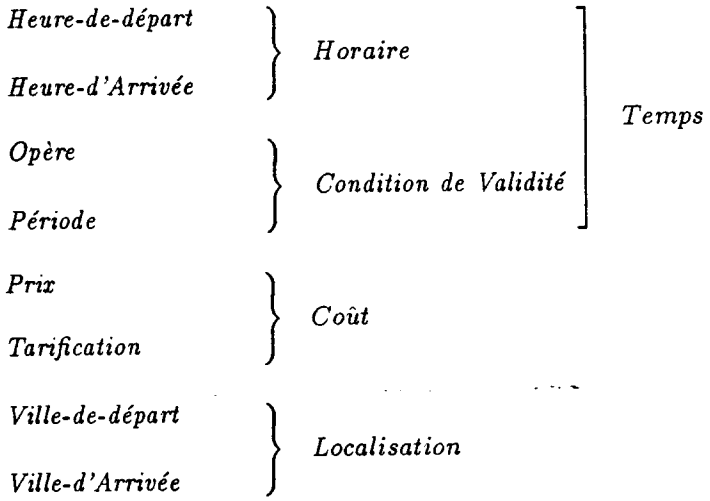
où *Ville-de-Départ, Ville-d'Arrivée, etc* représente en fait le codage dans la méta-théorie des prédicats objets *Ville-de-Départ, Ville-d'Arrivée, etc.*

Naturellement les *Thèmes* peuvent être structurés de la même façon que les types d'entités. On pourrait par exemple introduire dans la méta-théorie les faits suivants :

- *ISA(Horaire, Temps)*
- *ISA(Condition-de-Validité, Temps)*

où *ISA* est toujours une relation réflexive et transitive (Cf. partie précédente).

De façon schématique la structuration des attributs en *thèmes* définie ci-dessus peut être représentée par l'arbre suivant :


Heure-de-départ
Heure-d'Arrivée
Horaire
Opère
Période
Condition de Validité
Temps
Prix
Tarification
Coût
Ville-de-départ
Ville-d'Arrivée
Localisation

Chapitre 5

Le langage de requêtes

5.1 Format syntaxique

Comme il est signalé dans la partie précédente, le format des requêtes est fortement restreint. La raison principale de cette restriction est toujours de pouvoir exprimer de façon plus précise la "sémantique" d'une question (Cf. [22]).

La forme générale est donc la suivante :

$$Sujet \wedge Condition \wedge Informations\ demandées$$

où

"Sujet" est une formule construite en utilisant seulement les opérateurs logiques $\wedge$ (conjonction) ou $\vee$ (disjonction) et où les seuls prédicats sont des prédicats de type *"Classe"*.
Soit $A = Var(Sujet)$ (ensemble des variables libres apparaissant dans la formule *"Sujet"*)

"Condition" est une formule quelconque du langage objet.
Soit $B = Var(Condition)$.

"Informations Demandées" est une formule construite en utilisant seulement l'opérateur logique $\wedge$ et où les seuls prédicats sont des prédicats correspondant à des attributs.
Soit $C = Var(Informationsdemandées)$.

On introduit les conditions supplémentaires suivantes :

- $B \subset (A \cup C)$ (Les variables libres de *"Condition"* doivent donc être des variables libres de *"Sujet"* ou de *"Informations Demandées"*)

- Les requêtes doivent être des formules indépendantes du domaine, au sens défini dans [26].

Pour représenter et pour manipuler une requête, nous introduisons, dans le méta-langage, les prédicats suivants :

Requête(x) : x est une requête.

Sujet-i(x, y) : y est le code de la formule représentant le sujet (initial) de la requête x.

Condition-i(x, y) : y est le code de la formule représentant les conditons (initiales) de la requête x.

Info-i(x, y) : y est le code de la formule représentant les informations (initiales) demandées de la requête x.

Par exemple, si l'utilisateur pose la requête $qt1$ du chapitre 1 :

qt1 : Pour les Vols
Tels que :
Ville-de-départ = Paris
Ville-d'arrivée = New-York
Heure-de-départ appartient à [8 00 , 12 00]
Fournir la valeur de l'attribut : Heure-de-départ.

on introduit dans la méta-théorie les faits suivants :

Requête$(qt1)$
avec $qt1 =$ "$Vol(x) \wedge$
Ville-de-Départ$(x, Paris) \wedge$ *Ville-d'Arrivée*$(x, New\text{-}York) \wedge$
Heure-de-Départ$(x, y) \wedge (8 < y) \wedge (y < 12) \wedge$
Heure-de-Départ(x, y)"

(qt1 est le code de la conjonction des formules définissant les parties Sujet-i, Condition-i et Info-i de la requête)

Sujet-i$(qt1,$ "$Vol(x)$")

Condition-i$(qt1,$ "*Ville-de-Départ*$(x, Paris) \wedge$
Ville-d'Arrivée$(x, New\text{-}York) \wedge$
Heure-de-Départ$(x, y) \wedge (8 < y) \wedge (y < 12)$")

Info-i$(qt1,$ "*Heure-de-Départ*(x, y)")

Dans l'annexe C, nous étudions quelles sont les formules du langage des prédicats qui peuvent s'exprimer dans le format syntaxique défini ci-dessus pour une requête.

En fait, nous montrons, dans cette partie, que notre langage de requête à une puissance d'expression comparable à celle du langage Alpha présenté dans [15].

5.2 Question hétérogène

Nous présentons maintenant la raison pour laquelle nous introduisons un nouvel opérateur logique dans le langage de requête.

Considérons par exemple la requête élémentaire suivante :

$qt1$:
Sujet : $Vol(x)$

et supposons qu'en appliquant la règle RT7 de la partie 1.2, nous ayons généré une nouvelle requête qui fournit le prix des vols pour les vols "chers". Supposons que $Vol\text{-}Cher(x)$ soit une formule indiquant que le prix d'un vol est cher. Le résultat de la transformation pourrait être représenté par les deux requêtes $qt2$ et $qt3$:

$qt2$:
Sujet : $Vol(x)$
Condition : $\neg Vol\text{-}Cher(x)$

$qt3$:
Sujet : $Vol(x)$
Condition : $Vol\text{-}Cher(x)$
Informations-Demandées : $Prix(x, y)$

ce qui correspond dans la notation de la logique du premier ordre aux deux formules suivantes :

$qt2 : Vol(x) \wedge \neg Vol\text{-}Cher(x)$

$qt3 : Vol(x) \wedge Vol\text{-}Cher(x) \wedge Prix(x, y)$

On peut remarquer que cet ensemble de deux requêtes ne peut être représenté par la seule requête suivante :

$$(Vol(x) \wedge \neg Vol\text{-}Cher(x)) \vee (Vol(x) \wedge Vol\text{-}Cher(x) \wedge Prix(x, y))$$

qui est équivalente à :

$qt4 : (Vol(x) \wedge (Vol\text{-}Cher(x) \longrightarrow Prix(x, y))$

En effet la réponse à cette requête n'est pas l'union des réponses aux requêtes $qt2$ et $qt3$, car les deux opérandes de la disjonction n'ont pas les mêmes variables libres. De plus, la formule ci-dessus n'a pas de signification intuitive car ce n'est pas une formule indépendante du domaine [26].

Il serait pourtant intéressant de représenter $qt2$ et $qt3$ sous forme d'une seule requête et de fournir une seule réponse ayant la forme de la réponse AT6 du chapitre 1; naturellement, cette réponse est hétérogène dans le sens où tous les tuples de la réponse n'ont pas le même nombre de composants. Ici certains tuples sont des singletons $< x >$ alors que d'autres sont des paires $< x, y >$.

L'opérateur $\Longrightarrow$ est donc introduit dans le langage de requête pour pouvoir représenter un ensemble de questions hétérogènes. En utilisant cet opérateur, les deux requêtes ci-dessus sont représentées par une seule requête :

$$Vol(x) \wedge (Vol\text{-}Cher(x) \Longrightarrow Prix(x, y))$$

La signification intuitive de cette formule est la suivante : pour une valeur de x donnée, si $Cher(x)$ est faux, alors la formule a une seule variable libre x, et si $Cher(x)$ est vrai, la formule a deux variables libres, x et y. Par suite la valeur de y est fournie pour les seuls vols chers.

La définition précise du pseudo-implique est la suivante :

Soit $F = G(X) \wedge (f(X, Y) \Longrightarrow g(X, Y, Z))$
où X, Y, Z sont des n-uplets de variables.
Alors, par définition, cette formule est considérée comme une notation pour représenter les deux formules :

- $F_1(X) = G(X) \wedge \neg\exists(Y) f(X, Y)$
- $F_2(X, Y, Z) = G(X) \wedge f(X, Y) \wedge g(X, Y, Z)$

Pour que F soit une formule indépendante du domaine, il faut que F_1 et F_2 soient deux formules indépendantes du domaine.

D'un point de vue opérationnel, les réponses à la requête $F(X, Y, Z)$ peuvent être obtenues de la façon suivante :

Pour $< X_0 >$ satisfaisant $G(X)$,
  Pour $< X_0, Y_0 >$ satisfaisant $f(X_0, Y)$,
    la réponse est l'ensemble des $< X_0, Y_0, Z_0 >$ satisfaisant $g(X_0, Y_0, Z)$;
  S'il n'y a pas de $< X_0, Y_0 >$ satisfaisant $f(X_0, Y)$
    alors la réponse est $< X_0 >$;

Par la suite, les requêtes contenant l'opérateur $\Longrightarrow$ seront appelées "requêtes hétérogènes".

5.3 Réponses aux requêtes

Nous indiquons dans cette partie comment nous définissons la réponse à une requête.

En général, la réponse à une requête est définie de la façon suivante (Cf. [53] par exemple) :

- Soit $L = (A, W)$ un langage du premier ordre (où A est l'alphabet et W l'ensemble des fbfs du langage).
- Soit $F(X)$ une fbf de L . Les seules variables libres de F sont $X =< x_1, .., x_n >$.
- Soit T une théorie de L.

La réponse à la requête $F(X)$ dans la théorie T est l'ensemble des n-uplets de constantes $C =< c_1, .., c_n >$ tels que :

$$T \vdash_L F(\sigma(C/X))$$

où $\sigma(C/X)$ représente la substitution de X par C.

En ce qui concerne notre langage de requête, une telle définition de la réponse à une question ne fournit pas suffisamment d'informations à l'utilisateur comme on peut s'en rendre compte dans l'exemple suivant :

Considérons une base de données où sont représentées des entités de type *Train*.

Nous considérons deux sous-classes de *Train* :

Train-TEE : L'ensemble des trains *Trans Europ Express.*

Train-Couchette : L'ensemble des trains ayant des couchettes.

L'attribut suivant est associé à la classe d'entités *Train-TEE* :

Supplément-TEE(x, y) *:* Pour le *Trans Europ Express* x, il y a un supplément y.

L'attribut suivant est associé à la classe d'entités *Train-Couchette* :

Supplément-Couchette(x, y) *:* Pour le train couchette x, il y a un supplément y.

Considérons maintenant le requête hétérogène :

Entité : Train(x)

Informations demandées :
$(\textit{Train-TEE}(x) \Longrightarrow \textit{Supplément-TEE}(x, y)) \wedge$
$(\textit{Train-Couchette}(x) \Longrightarrow \textit{Supplément-Couchette}(x, z))$

qui correspond en langue naturelle à la question suivante :

"Pour les trains, fournir le Supplément TEE pour les Trains TEE, et le Supplément Couchette pour les Trains Couchette"

Dans une structure de n-uplets, la réponse à cette question est constituée des ensembles suivants :

- Un ensemble S_1 de singletons $< x >$ correspondant aux Trains qui ne sont ni des Trains TEE ni des Trains Couchettes.
- Un ensemble S_2 de paires $< x, y >$ correspondant aux Trains TEE.
- Un ensemble S_3 de paires $< x, z >$ correspondant aux Trains Couchettes.
- Un ensemble S_4 de triplets $< x, y, z >$ correspondant aux Trains TEE ayant des wagons-lits.

En présentant une réponse sous forme de tuples, l'utilisateur n'a donc aucun moyen de savoir si le couple $< a, b >$ appartient à S_2 ou S_3. C'est l'une des raisons pour laquelle nous disons que fournir des réponses sous forme de n-uplet à des requêtes hétérogènes ne fournit pas suffisamment d'informations à l'utilisateur.

Dans notre système, la réponse correspondant aux Trains TEE serait représentée par la formule instantiée suivante :

$$\textit{Train-TEE}(a) \wedge \textit{Supplément-TEE}(a, b)$$

alors que les réponses correspondant aux Trains Couchette serait représentée par la formule instantiée suivante :

$$\textit{Train-Couchette}(a) \wedge \textit{Supplément-Couchette}(a, c)$$

L'utilisateur peut ainsi distinguer les différentes solutions.

La deuxième raison correspond au cas où l'on fournit des réponses complétives (Cf. section 7.3).

Ainsi, prenons la requête suivante :

$$\textit{Personne}(x) \wedge (\textit{Travaille}(x, \textit{Toulouse}) \vee \textit{Habite}\ (x, \textit{Toulouse}))$$

c'est-à-dire *"Quelles sont les personnes qui travaillent à Toulouse ou qui habitent à Toulouse"*.

Nous avons déjà indiqué qu'il était intéressant de fournir pour ce genre de questions des réponses qui permettent de distinguer parmi les solutions les personnes qui travaillent à Toulouse de celles qui habitent à Toulouse.

Pour que l'utilisateur puisse faire cette distinction, nous fournissons des réponses ayant la forme suivante :

1. $\textit{Personne}(a_1) \wedge \textit{Travaille}(a_1, \textit{Toulouse})$
   pour les personnes qui travaillent à Toulouse
2. $\textit{Personne}(a_2) \wedge \textit{Habite}(a_2, \textit{Toulouse})$
   pour les personnes qui habitent à Toulouse

3. $Personne(a_3) \wedge Travaille(a_3, Toulouse) \wedge Habite(a_3, Toulouse)$
   pour les personnes qui travaillent et habitent à Toulouse.

Nous revenons sur les problèmes liés aux réponses complétives et sur la façon de générer la réponse intéressante dans la partie suivante.

Dans l'annexe D, après avoir redéfini les conditions de satisfiabilité des formules, nous construisons une algèbre permettant de fournir des formules instanciées commes réponse à une requête.

5.4 Principe de transformation des requêtes

Maintenant que nous avons défini notre langage de requête, nous allons montrer comment nous avons formalisé les trois transformations présentées de façon intuitive dans la deuxième partie, à savoir :

1. Transformation des informations demandées
2. Transformation des conditions
3. Transformation du sujet

Le principe général de la première transformation est le suivant :

Partant d'une requête initiale Q, il s'agit d'obtenir une seule requête transformée Q'. C'est la réponse à cette dernière requête que l'on fournit à l'utilisateur.

Nous n'avons pas appliqué le même principe aux deux autres transformations. Dans les deux cas, on part toujours d'une requête initiale Q_0, mais, en général, la transformation nous permet d'obtenir plusieurs requêtes $Q_1, ..., Q_n$. En répondant à Q_0, on fournit les solutions exactes. En répondant aux requêtes $Q_1, \ldots, Q_n$, on fournit les réponses voisines. Dans la définition de ces deux transformations, on peut faire en sorte que les solutions exactes n'apparaissent pas dans les réponses voisines.

Partie III

Représentation des connaissances expertes

Chapitre 6

Représentation de l'utilisateur

Nous avons déjà remarqué dans les parties précédentes que, pour avoir un comportement coopératif, le système devait connaître des informations concernant l'utilisateur.

Dans ce chapitre, nous récapitulons les informations que nous prenons en compte, et nous montrons comment celles-ci sont formalisées.

La façon la plus simple de représenter un utilisateur consiste à avoir un ensemble de types d'utilisateurs, et d'indiquer, au système, le type d'un utilisateur donné. Ceci correspond à introduire dans le système un ensemble de stéréotypes. Par exemple, si l'on désire savoir si l'utilisateur voyage pour raison touristique ou pour affaires, on peut alors introduire les deux types d'utilisateurs correspondants.

Ces types sont représentés au niveau méta, en utilisant le prédicat :

- *Type-Utilisateur*(*Touriste*)
- *Type-Utilisateur*(*Homme-d'affaires*)

Au niveau objet, il suffit d'introduire autant de prédicats unaires qu'il y a de stéréotypes. On aura donc, par exemple, les prédicats : *Touriste*(x) et *Homme-d'affaires*(x).

On peut aussi avoir besoin d'informations particulières concernant l'utilisateur comme par exemple :

- L'utilisateur est Français.
- L'utilisateur a 50 ans.

Ce type d'information peut être facilement formalisé en logique du premier ordre en définissant un type d'entité *Utilisateur* ayant *Nationalité* et *Age* pour attributs. Pour cela, il suffit d'introduire dans le langage les prédicats suivants :

- *Utilisateur* (x) : x est l'utilisateur.

- *Nationalité* (x,y) : la nationalité de x est y.
- *Age* (x,y) : l'age de x est y.

Les prédicats *Nationalité* et *Age* peuvent être considérés comme des caractéristiques de l'utilisateur et regrouper dans un stéréotype. De la même façon que dans le chapitre 4, on a introduit, au niveau méta, le prédicat binaire $Attribut(x,y)$ pour indiquer que l'attribut x est défini pour l'entité y, on introduit le méta prédicat $Caractéristique(x,y)$ pour indiquer que la caractéristique x est définie pour le stéréotype y.

Nous verrons par la suite (Cf. règle RD4 de la page 115) comment la notion de stéréotype peut être utilisée pour déterminer les thèmes d'intérêt de l'utilisateur.

Nous avons également indiqué qu'il était intéressant de représenter des croyances "de bon sens" qui sont supposées par tous les utilisateurs, jusqu'à preuve du contraire, comme par exemple :

- Un vol fonctionne du 1 Janvier au 31 Décembre.
- Un vol fonctionne tous les jours de la semaine.
- Un vol est sans supplément.

Au niveau objet, ces croyances s'expriment par les règles :

D1: $\forall(x)Vol(x) \longrightarrow Période(x, 01/01, 31/12)$

D2: $\forall(x,y)Vol(x) \wedge Jour(y) \longrightarrow Opère(x,y)$

D3: $\forall(x)Vol(x) \longrightarrow \neg Supplément(x)$

Comme ces règles peuvent avoir des exceptions, il faut fournir à l'utilisateur les faits qui expriment ces exceptions et qui sont en relation avec la question; sinon, ce dernier va peut-être inférer "par défaut" des informations erronées.

Un tel comportement coopératif a été étudié dans [41] où l'on commence par proposer une modification de la maxime de qualité des réponses de Grice [33] :

Maxime de Grice : Ne dites pas ce que vous croyez faux ou pour lequel il vous manque une preuve correcte.

Modification proposée par Joshi : Si vous estimez dire, dans votre réponse, une information qui peut conduire votre interlocuteur à déduire quelque chose que vous croyez faux, alors fournissez une information supplémentaire pour bloquer cette déduction.

Or, un raisonnement par défaut (Cf. [52]) effectuer par votre interlocuteur peut l'amener à des conclusions érronées de la façon suivante :

Soit Q la personne qui pose la question et R la personne qui répond. Si Q pose la question :

$$\text{"}Quels\ sont\ les\ x\ tels\ que\ B(x)\text{"}$$

si R répond "$B(a)$" et si Q croit que "la plupart des B sont F", ce que l'on notera :

$$Plupart(x)(B(x) \longrightarrow F(x))$$

alors Q va effectuer la déduction suivante :

$$\frac{Informer(R,Q,B(a)) \wedge (Plupart(x)B(x) \longrightarrow F(x)) \wedge \neg Informer(R,Q,\neg B(a)) : M(F(a))}{F(a)}$$

Comme dans [52], $M(P)$ signifie qu'il est consistent de supposer P.

Par exemple, si l'on considère la règle D1, qui, en fait, va sécrire :

D1: $Plupart(x)(Vol(x) \longrightarrow Période(x, 01/01, 31/12))$

alors $B(x)$ correspond à la formule $Vol(x)$, $F(x)$ correspond à $Période(x, 01/01, 31/12)$ et a à un vol particulier. En utilisant le schéma de déduction ci-dessus, Q va déduire que $Période(a, 01/01, 31/12)$. Si cette information est fausse, alors R doit bloquer l'inférence en réalisant l'action :

$$Informer(R, Q, \neg Période(a, 01/01, 31/12))$$

Pour représenter ce mécanisme de déduction, nous introduisons, dans le métalangage, le prédicat :

Croyance-Normale("$E(x)$", "a", "$C(x)$") : Pour une entité x de type E, tout utilisateur suppose a priori que la propriété $C(x)$ est verifiée; si ce n'est pas le cas, la valeur de l'attribut a doit être fournie pour l'entité x.

Ceci peut aussi s'écrire :

- Par défaut :

$$Plupart(x)(E(x) \longrightarrow C(x))$$

- Si on a pour un x_0 donné, $E(x_0) \wedge \neg C(x_0)$, alors il faut fournir la valeur de a pour x_0; soit :

$$\neg C(x) \Longrightarrow a(x, y)$$

où $\Longrightarrow$ est l'opérateur pseudo-implique.

Par exemple, les règles suivantes :

RD1: *Croyance-Normale*("*Vol*(x)", "*Période*", "*Période*(x,01/01, 31/12)")

RD2: *Croyance-Normale*("*Vol*(x)", "*Opère*",
"$\forall(y)\,Jour(y) \longrightarrow Opère(x,y)$")

RD3: *Croyance-Normale*("*Vol*(x)", "*Prix*", "$\neg$*Supplément*(x)")

sont respectivement la traduction au niveau méta des règles D1, D2 et D3.

Par la suite, nous n'utilisons pas d'autres informations concernant l'utilisateur que celles présentées dans ce chapitre.

Chapitre 7

Transformation des informations demandées

7.1 Principe général

Nous présentons dans cette partie, le formalisme choisi pour réaliser la transformation qui a été appelée dans les parties précédentes "transformation des informations demandées".

L'idée générale de la transformation consiste, pour une requête initiale Q, à transformer la partie "Information-Demandée" de cette question pour obtenir une nouvelle requête Q'. On fournit alors la réponse de la requête Q' à l'utilisateur.

En fait, cette transformation peut prendre deux aspects :

1. Effectuer une intervention pertinente (Cf. [24]).
2. Fournir une réponse complétive (Cf. section 2.3 pour une présentation intuitive de cette notion)

Toutefois, les formalismes choisis pour représenter ces deux transformations étant assez proches l'un de l'autre, nous avons préféré les présenter tous les deux dans cette partie, pour pouvoir mettre en évidence leurs aspects communs.

Nous commençons par rappeler intuitivement quel est le but de ces transformations.

Ainsi, supposons que l'utilisateur pose la requête suivante :

$Requête(qt1)$

$Sujet\text{-}i(qt1, "Vol(x)")$

$Condition\text{-}i(qt1, "Ville\text{-}de\text{-}Départ(x, Paris) \wedge$

$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)$")

Info-i($qt1$, "*Heure-de-Départ*(x, y)")

Une traduction en langue naturelle de cette requête est la suivante :

Quelle est l'heure de départ des vols qui partent de Paris entre 8 et 12 heures et qui arrivent à New-York ou Boston ?

Un expert dans une agence de voyage va probablement fournir des informations supplémentaires telles que l'heure d'arrivée. En effet, cette information est intéressante pour une personne qui organise un voyage, par exemple si elle doit prendre une correspondance.

Un serveur coopératif peut aussi fournir à l'utilisateur, les jours de la semaine pour lesquels le vol opère. Cette information est particulièrement intéressante si le vol n'opère pas tous les jours de la semaine. En effet, connaissant cette information, l'utilisateur va peut-être modifier la date de son départ.

Pour la même raison, la période de validité est également une information intéressante à fournir lorsque le vol n'opère pas toute l'année.

Enfin, dans le cas d'un vol très onéreux, il est intéressant de l'indiquer à l'utilisateur même s'il s'agit d'un homme d'affaires qui n'est pas préoccupé par les problèmes pécuniaires.

D'autre part, il est également intéressant de fournir à l'utilisateur la ville d'arrivée des vols solutions de sa requête. En effet, l'utilisateur, bien qu'il ne l'ait pas explicitement demandé, souhaite distinguer parmi les solutions les vols qui arrivent à New-York de ceux qui arrivent à Boston.

Tout ceci conduit à générer une nouvelle requête $qt2$:

Requête($qt2$)

Sujet-i($qt2$, "$Vol(x)$")

Condition-i($qt2$, "$Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)$")

Info-i($qt2$, "$Heure\text{-}de\text{-}Départ(x, y) \wedge$
$Heure\text{-}d'Arrivée(x, y_1) \wedge$
$(\exists(y_2) Jour(y_2) \wedge \neg Opère(x, y_2)) \Longrightarrow Opère(x, y_3) \wedge$
$\neg Validité(x, 01/01, 31/12) \Longrightarrow Validité(x, y_4, z) \wedge$
$Supplément(x) \Longrightarrow Prix(x, y_5) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \Longrightarrow Ville\ d'Arrivée(x, New\text{-}York) \wedge$
$Ville\text{-}d'Arrivée(x, Boston) \Longrightarrow Ville\ d'Arrivée(x, Boston)$")

Lorsque l'expert fournit à l'utilisateur la valeur des attributs *Heure de départ, Validité, Opère* et *Prix*, nous disons qu'il effectue une intervention pertinente. Lorsqu'il précise la valeur de l'attribut *Ville d'arrivée*, il réalise une réponse complétive.

Dans ce chapitre nous commençons par présenter les différentes informations utilisées pour réaliser des interventions pertinentes. Nous étudions ensuite des méthodes permettant de fournir des réponses complétives. Dans l'avant dernière section, nous présentons les règles générales permettant de transformer la requête initiale en une nouvelle requête qui fournit des informations supplémentaires intéressantes. Enfin, nous montrons comment un mécanisme d'inférence standard utilisant ces connaissances peut dériver la requête transformée.

7.2 Interventions pertinentes

Nous montrons dans cette partie comment les règles permettant de fournir, en fonction du contexte de la question de l'utilisateur, la valeur d'attributs supplémentaires, peuvent s'exprimer dans notre formalisme.

Nous commençons par étudier trois situations différentes pour lesquelles il est utile de fournir des informations supplémentaires à l'utilisateur :

1. Réaliser un plan
2. Etablir ou modifier un plan
3. Convaincre l'utilisateur de la pertinence des solutions

Nous présentons ensuite les règles générales qui permettent de fournir ces informations supplémentaires.

7.2.1 Informations utiles pour réaliser un plan

Nous étudions le cas où l'on fournit des informations utiles à la réalisation d'un plan.

Remarquons tout d'abord que notre approche diffère sensiblement de celle suivie par Allen [3] et présentée dans la partie 2.4.

Ainsi reprenons l'exemple de la section 2.4.1 :

(1.1) A : A quelle heure est le départ du train pour Montréal ?

(1.2) B : 3:15 quai numéro 7.

Pour fournir la réponse (1.2), le serveur a d'abord supposé que l'utilisateur a certains buts à atteindre. La requête concernant l'heure de départ du train, (1.1) montre que le but de l'utilisateur est probablement de prendre un train. En plus, en supposant que le serveur croit que l'utilisateur ne connait pas le lieu de départ du train, il croit donc que

la non-connaissance de ce lieu est un obstacle dans le plan. Il fournit donc une réponse qui résout cet obstacle (1.2).

Nous ne revenons pas sur la pertinence de cette approche mais nous considérons que celle-ci est un peu limitative.

D'abord, B peut ne pas connaître avec précision le but que A cherche à atteindre, et pourtant fournir des informations supplémentaires pertinentes. Ainsi, considérons le dialogue suivant :

(2.1) : Quelle est l'heure de départ des Vols Toulouse/Paris partant entre 8 heures et 10 heures ?

(2.2) : Vous avez le vol AI315 qui part à 8 heures 30 et qui arrive à Orly à 10 heures 30 ainsi que le vol AI316 qui part à 9 heures 30 et qui arrive à Roissy à 11 heures 20.

La question (2.1) montre que le but de l'utilisateur est probablement de se rendre à Paris en avion. Toutefois, le serveur ne peut pas déduire d'après cette question quelle sera l'action que réalisera l'utilisateur une fois arrivé à Paris. Il peut par exemple se rendre à une conférence dans le sud de la ville, ou bien déposer ses affaires dans un hotel, ou bien encore prendre un autre avion à destination de New-York, *etc...* C'est en ce sens que nous disons que le serveur ne connait pas avec précision le but que veut atteindre l'utilisateur lorsqu'il pose la question (2.1). Néanmoins, cela n'empêche pas le serveur de fournir des informations supplémentaires, à savoir l'heure d'arrivée du vol ainsi que des informations sur l'aéroport d'arrivée.

Ces informations sont fournies à l'utilisateur car un expert sait par expérience qu'elles peuvent être utiles dans la réalisation de l'action qui amène l'utilisateur à Paris.

C'est en particulier pour prendre en compte ce genre de situation que nous introduisons la notion de *"thème"* ainsi que la notion *"d'intérêt"*.

La notion de *thème* qui a déjà été présentée dans la section 3.3 nous permet de regrouper certains attributs en fonction de leur contenu sémantique.

Nous avons également introduit la notion d'intérêt qui nous permet de déduire, d'après la question posée, les informations pouvant intéresser l'utilisateur.

Ainsi, si l'on reprend la question (2.1), on peut déduire que le but de l'utilisateur lorsqu'il pose cette question est probablement de se rendre à Paris en avion, mais on peut également conclure que le thème *horaire* l'intéresse. C'est la raison pour laquelle un serveur coopératif fournit dans la réponse (2.2) la valeur d'autres attributs regroupés sous le même thème. De même, l'information concernant l'aéroport d'arrivée (Orly ou Roissy) est intéressante car elle est associée au même thème (*localisation*) que l'attribut *Ville d'arrivée* qui apparait dans la requête (2.1).

En revanche, si l'on considère la question suivante :

(3.1) Quels sont les vols Paris/New-York à moins de 8 000 francs ?

on peut déduire que le but de l'utilisateur est probablement de se rendre à New-York, mais on peut également conclure que le thème *coût* l'intéresse.

En tenant compte de cette notion d'intérêt, une réponse coopérative à la question (3.1) pourrait être :

(3.2) Vous avez le vol bleu AF025 à 7.800 francs ou bien le vol charter AF028 à 7.200 francs.

Ici un serveur coopératif fournit dans sa réponse, la valeur de l'attribut *Prix* ainsi que la tarification en vol bleu pour le vol AF025 car ces deux attributs concernent le thème *coût*. Pour le vol AF028, le serveur a indiqué qu'il s'agissait d'un vol charter car ce fait modifie généralement le prix du vol. Cette information est donc utile pour quelqu'un qui s'intéresse au thème *coût*.

Pour représenter cette notion d'intérêt, nous introduisons dans le méta-langage les prédicats suivants :

$Apparait(x, y)$: l'attribut y apparait dans la requête x.

$Int\text{-}Top(x, y)$: le thème y est intéressant pour la requête x.

Les axiomes associés à ces prédicats sont les suivants :

S1 : $\forall\ (q, a, top)$
$Apparait(q, a) \wedge Att\text{-}Top(a, top) \longrightarrow Int\text{-}top(q, top)$

Cette règle indique que les thèmes associés aux attributs apparaissant dans la requête sont des thèmes intéressants.

En utilisant cette règle, on peut déduire que l'utilisateur s'intéresse au thème *Horaire*, du fait qu'il a posé une question concernant *l'heure de départ* et que cet attribut concerne les *horaires*.

S2 : $\forall\ (q, top, top')$
$Int\text{-}Top(q, top) \wedge ISA(top', top) \longrightarrow Int\text{-}Top(q, top')$

Cette règle indique que dans la structure hiérarchique des thèmes, la notion d'intérêt est héritée.

A l'aide de cette règle, on peut déduire que l'utilisateur s'intéresse au thème *Horaire* du fait qu'il s'intéresse au thème plus général qu'est le *Temps*.

La règle RD4 qui suit est valide uniquement pour le domaine d'application d'une agence de voyage. Elle indique que le thème *Coût* est intéressant pour toute requête posée par des utilisateurs voyageant pour raison touristique :

RD4: $\forall\ (q)$
$Requête(q) \wedge Type\text{-}d'utilisateur(Touriste) \longrightarrow Int\text{-}Top(q, Coût)$

Cette notion d'intérêt est ensuite utilisée pour déduire les attributs supplémentaires qu'il est pertinent de fournir à l'utilisateur. Ainsi, la règle S3 indique que, pour les entités v de type e, il est pertinent de fournir les valeurs des attributs a qui concernent un thème *top* qui a été reconnu comme thème intéressant. Cette valeur sera fournie par l'instantiation de la variable objet v_1, qui est codée par une constante au niveau méta.

$$
\begin{aligned}
&S3\text{:}\ \forall\ (q, e, v, top, a)\\
&\quad Sujet(q, \text{“}e(v)\text{”})\wedge\\
&\quad Int\text{-}Top(q, top)\wedge\\
&\quad Att(a, e)\wedge\\
&\quad Att\text{-}Top(a, top)\\
&\qquad \longrightarrow Att\text{-}Rel(q, \text{“}e(v)\text{”}, \text{“}a(v, v_1)\text{”})
\end{aligned}
$$

où $Att\text{-}Rel(q, \text{“}e(v)\text{”}, \text{“}a(v, v_1)\text{”})$ signifie qu'il est pertinent de fournir la valeur de l'attribut a pour l'entité v de type e qui apparait dans la partie *Entité* de la requête q.

Remarque : La notion de thème est également utile lorsqu'on interroge une base de données contenant un très grand nombre de prédicats. Exemple : une base de données où il y aurait un prédicat par nationalité. On aurait ainsi les prédicats : $Français(x)$, $Anglais(x)$, $Allemand(x)$, *etc...* En utilisant la notion de thème, on peut associer à chacun de ces prédicats le thème ***nationalité***. On peut alors demander quelles sont les personnes ayant une double nationalité en posant la question suivante :

$$
\begin{aligned}
&Q : Personne(x)\wedge\\
&\quad Att\text{-}Top(a_1,\ Nationalité)\wedge\\
&\quad Att\text{-}Top(a_2,\ Nationalité)\wedge\\
&\quad \neg(a_1 = a_2)\wedge\\
&\quad a_1(x) \wedge a_2(x)
\end{aligned}
$$

Une telle requête mélange les niveaux objet et méta. Une méthode simple pour l'évaluer consiste à la tranformer en une (ou plusieurs) requête entièrement évaluable au niveau objet. On obtient ainsi :

Q1 :$Personne(x) \wedge Français(x) \wedge Anglais(x)$

Q2 :$Personne(x) \wedge Français(x) \wedge Allemand(x)$

etc..

D'autre part, on peut écrire une règle de coopération qui indique, pour les requêtes recherchant des personnes ayant une nationalité donnée, les personnes solutions ayant une double nationalité. Cette règle est la suivante :

$$
\begin{aligned}
&\forall\ (q, x, a_1, a_2)\\
&\quad Sujet(q, \text{“}Personne(x)\text{”})\wedge\\
&\quad Int\text{-}Top(q, Nationalité)\wedge\\
&\quad Att\text{-}Top(a_1, Nationalité)\wedge
\end{aligned}
$$

$$Att\text{-}Top(a_2, Nationalité) \wedge$$
$$\neg(a_1 = a_2)$$
$$\longrightarrow Att\text{-}Rel(q, \text{“}Personne(x)\text{”},$$
$$\text{“}a_1(x) \wedge a_2(x) \Longrightarrow a_1(x) \wedge a_2(x)\text{”})$$

Il est intéressant de noter que les requêtes générées en utilisant cette règle ne mélangent plus les deux niveaux de langage et sont entièrement évaluables au niveau objet.

7.2.2 Informations utiles pour établir ou modifier un plan

Etudions maintenant le cas où l'on fournit des informations supplémentaires qui seront utilisées par l'utilisateur pour établir ou modifier un plan.

Nous présentons dans l'introduction un exemple, inspiré par Allen, illustrant ce type de situation. Ainsi, supposant que A, transportant un bidon vide, vienne vers B et lui demande :

(3.1) A : Où est la station service la plus proche ?

et si B répond :

(3.2) B : Au prochain carrefour, mais la station est fermée.

alors l'information supplémentaire fournie par B ("la station est fermée") va conduire A à modifier son plan initial qui était de se rendre à la station service la plus proche pour remplir son bidon vide.

Pour représenter ce type de comportement coopératif, Allen utilise toujours la notion de plan. Ainsi B va reconnaître, dans la question (3.1), que le but de A est de remplir son bidon d'essence, ce qu'il ne pourra pas réaliser si la station est fermée. B, ayant détecté cet obstacle, il va, s'il est coopératif, l'indiquer à A.

Dans notre formalisme, pour simuler ce type de comportement coopératif, nous utilisons la notion de *Croyance normale* qui a été introduite dans le chapitre précédent.

Ainsi, pour représenter la situation présentée par Allen, on peut utiliser les prédicats :

$Station\text{-}Service(x)$: x est une entité de type *Station Service*.

$Etat(x, y)$: l'entité x est dans l'état y (ouvert ou fermé).

Une croyance "normale" pour l'utilisateur sera de supposer que la station service est ouverte. Au niveau objet, cette croyance s'exprime par la règle :

$$\forall(x) Station\text{-}Service(x) \longrightarrow Etat(x, Ouvert)$$

Au niveau méta, pour représenter cette information, nous introduisons le fait suivant :

$$Croyance\text{-}Normale(\text{“}Station\text{-}Service(x)\text{”}, \text{“}Etat\text{”}, \text{“}Etat(x, Ouvert)\text{”})$$

Ce fait indique que, pour une entité x de type *Station-Service*, tout utilisateur suppose que la condition $Etat(x, Ouvert)$ est vérifiée; si ce n'est pas la cas, la valeur de l'attribut *Etat* doit être fournie pour l'entité x.

Pour effectivement fournir la valeur de cet attribut, nous utilisons la règle suivante :

$$\begin{aligned} \text{S4} : \forall & (q, E, x_1, x_2, a, C, C') \\ & Sujet(q, \text{“}E(x_1)\text{”}) \wedge \\ & Croyance\text{-}Normale(\text{“}E(x_2)\text{”}, a, C) \wedge \\ & Substituer(C, x_2, x_1, C') \\ & \longrightarrow Att\text{-}Rel(q, \text{“}E(x_1)\text{”}, \text{“}(\neg C') \Longrightarrow a(x_1, v_1)\text{”}) \end{aligned}$$

La règle S4 exprime que s'il y a une requête concernant les entités x_1 de type E et il y a un fait, de la méta théorie, représentant une croyance normale C pour ce type d'entité, comme par exemple RD1, RD2 ou RD3 (cf. chapitre précédent), alors la valeur de l'attribut a doit être fournie pour les entités qui ne satisfont pas la condition C (c'est à dire pour les entités qui sont des exceptions). L'opérateur *pseudo-implique*, dans *Att-Rel*, permet de représenter ces situations exceptionnelles.

Le prédicat *Substituer* change le nom de la variable x de $C(x)$ dans *Croyance-Normale*, par le nom de la variable utilisé dans la requête. Intuitivement, nous devons exprimer que la condition C s'applique aux entités solutions de la requête.

Si maintenant un utilisateur pose la question suivante :

(4.1) Quels est l'heure de départ des vols Paris/New-York partant entre 10 heures et midi ?

alors un serveur utilisant la règle RD3 fournira la réponse suivante :

(4.2) Vous avez le vol AF015 qui part à 10 heures 30 ainsi que le vol AF001 qui part à 10 heures mais il vous en coûtera 27.715 francs.

Etant donné le prix du vol AF001, beaucoup d'utilisateurs, en prenant connaissance de ce prix, auront tendance à écarter ce vol lors de l'élaboration du plan permettant de réaliser le but qu'ils souhaitent atteindre (Aller à New-York).

On remarquera qu'il n'est pas facile de simuler le même comportement en utilisant des plans.

En effet, pour que l'information concernant le prix des vols chers soit fournie à l'utilisateur, il faut que celle-ci constitue un obstacle à la réalisation du plan “prendre un vol”.

Si l'on reprend le formalisme choisi par Allen, nous sommes dans cette situation si l'action suivante :

ACHETER (A, S, Ticket-Pour (V_1))
Conds : Possede (A, Prix (Ticket-Pour (V_1)))
Effets : Possede (A, Ticket-Pour (V_1))

ne peut être réalisée par l'utilisateur. C'est bien le cas si l'utilisateur ne possède par l'argent pour acheter le ticket du vol.

Toutefois, il se peut que l'utilisateur, bien que possédant l'argent lui permettant de prendre le Concorde, ne soit pas d'accord pour dépenser l'argent que nécessite un tel voyage.

Ce genre de situation ne s'exprime pas simplement dans le formalisme présenté par Allen.

7.2.3 Informations destinées à convaincre l'utilisateur

Une troisième situation pour laquelle on est amenée à fournir des informations supplémentaires se présente lorsqu'on essaye de convaincre l'utilisateur de la pertinence des solutions trouvées.

Ce type de coopération apparaît dans l'exemple de dialogue suivant :

(5.1) A : Pourriez-vous me conseiller un placement en actions françaises ?

(5.2) B : Je vous suggère des actions Peugeot.

(5.3) A : Mais Peugeot n'est-elle pas une entreprise en déficit ?

(5.4) B : C'est exact pour cette année, mais on prévoit un retour au bénéfice pour l'année prochaine.

Pour fournir la réponse (5.4), B doit reconnaître que l'intervention (5.3) de A est en fait une objection. Cette intervention indique en effet que A estime qu'une entreprise déficitaire n'est pas un bon placement. Cette objection étant exacte, B doit alors trouver un argument destiné à convaincre A de la pertinence de la solution proposée. Pour cela, B doit examiner les informations qui l'on conduit à proposer un placement en actions Peugeot. Ces informations pourraient être par exemple :

1. Peugeot a gagné le championnat du monde des rallyes en 1985 et 1986.
2. Peugeot a augmenté ses parts de marché en France et en Europe.
3. On prévoit un retour au bénéfice pour l'année prochaine.

Etant donnée la nature de l'objection (5.3), l'information 3 est l'argument qui pourra le mieux convaincre l'utilisateur. Ceci conduit B à fournir cette information supplémentaire dans la réponse (5.4).

Si ensuite l'utilisateur pose la question :

(5.5) A : Les ventes de Peugeot n'ont elles pas baissé en Afrique ?

alors B utilisera plutôt l'information 2, et fournira la réponse coopérative suivante :

(5.6) B : C'est exact, mais Peugeot a augmenté ses parts de marchés en France et en Europe.

Nous n'avons pas encore étudié en détail ce type particulier de coopération.

En revanche, nous présentons dans les deux chapitres suivants, d'autres situations dans lesquelles on fournit également des informations supplémentaires pour convaincre l'utilisateur de l'intérêt des réponses qui lui sont proposées. En particulier, dans le chapitre suivant, lorsqu'on propose des réponses satisfaisant des conditions voisines, on accompagne ces réponses d'informations comparatives (cf. page 148); ces informations sont destinées à convaincre l'utilisateur que les réponses voisines proposées présentent certains avantages par rapport aux solutions de la requête initiale.

7.2.4 Règles générales pour fournir les attributs pertinents

On a un premier ensemble de règles S5, S6 et S7 dont le rôle est de déduire un ensemble de formules atomiques définissant les types d'entités qui apparaissent dans le partie *Sujet* de la requête :

$$\text{S5} : \forall\,(q,e)\quad \textit{Sujet-i}(q,e) \longrightarrow \textit{Sujet-atom}(q,e)$$

$$\text{S6} : \forall\,(q,e_1,e_2)\quad \textit{Sujet-atom}(q, \text{“}e_1 \wedge e_2\text{”}) \longrightarrow \textit{Sujet-atom}(q,e_1) \wedge \textit{Sujet-atom}(q,e_2)$$

$$\text{S7} : \forall\,(q,e_1,e_2)\quad \textit{Sujet-atom}(q, \text{“}e_1 \vee e_2\text{”}) \longrightarrow \textit{Sujet-atom}(q,e_1) \wedge \textit{Sujet-atom}(q,e_2)$$

où le méta prédicat $\textit{Sujet-atom}(x,y)$ signifie que, dans la partie *Sujet* de la requête x, apparait le formule atomique y.

La règle S8 indique qu'un type d'entité qui n'est pas disjoint d'un type d'entité apparaissant dans la requête, peut, par extension, être considéré comme un type d'entité sujet de la requête.

$$\text{S8} : \forall\,(q,e,e',x)\quad \textit{Type-Entité}(e') \wedge \textit{Sujet-atom}(q, \text{“}e(x)\text{”}) \wedge \neg \textit{DIS}(e,e') \longrightarrow \textit{Sujet}(q, \text{“}e'(x)\text{”})$$

où le méta prédicat $\textit{Sujet}(x,y)$ signifie que le type d'entité y n'est pas disjoint d'au moins un type d'entité dans la partie *Sujet* de la requête x.

Nous avons introduit le prédicat *Sujet* car, il peut arriver que, dans la requête, une formule atomique apparaisse dans la partie *Sujet*, par exemple $Homme(x)$, et qu'il soit alors possible de déduire que pour un autre type d'entité, par exemple $Professeur(x)$, qui n'est pas disjoint de $Homme(x)$ et n'apparait pas dans la requête, il soit pertinent de fournir la valeur d'un certain attribut a alors que ce n'est pas le cas pour les hommes en général. Dans ce cas, $Professeur(x) \Longrightarrow a(x, y)$ est ajouté à la liste des attributs à fournir, bien que $Professeur(x)$ n'apparaisse pas dans la requête.

Les formules DEF1 et DEF2 définissent deux méta prédicats utilisés dans les règles S9 et S10.

Le méta prédicat $Att\text{-}Rel\text{-}pour\text{-}tous(q, f)$ signifie qu'il est possible de déduire, pour tous les types d'entités de la partie *Sujet* de la requête q, que l'attribut, représenté par la formule atomique f, est pertinent pour ces types d'entités.

Le méta prédicat $Att\text{-}Rel\text{-}pour\text{-}certains(q, \text{“}e(v)\text{”}, f)$ signifie que l'attribut représenté par f est pertinent pour au moins un type d'entité e, qui n'est pas disjoint d'au moins un type d'entité apparaissant dans la requête q.

$$\begin{aligned}&\text{DEF1} : \forall\,(q, f)\\ &\quad Att\text{-}Rel\text{-}pour\text{-}tous(q, f) \longrightarrow\\ &\qquad \forall(e, v)\exists(e')(Sujet\text{-}atom(q, \text{“}e(v)\text{”})\\ &\qquad\qquad \longrightarrow Att\text{-}Rel(q, \text{“}e'(v)\text{”}, f) \wedge ISA(e, e'))\end{aligned}$$

$$\begin{aligned}&\text{DEF2} : \forall\,(q, e, v, f)\\ &\quad Att\text{-}Rel\text{-}pour\text{-}certains(q, \text{“}e(v)\text{”}, f) \longleftrightarrow\\ &\qquad Sujet(q, \text{“}e(v)\text{”}) \wedge Att\text{-}Rel(q, \text{“}e(v)\text{”}, f)\end{aligned}$$

La règle S9 indique que, si l'attribut f est pertinent pour chaque type d'entité dans la requête, alors il faut l'ajouter à la liste des attributs à fournir.

$$\begin{aligned}&\text{S9} : \forall\,(q, f)\\ &\quad Att\text{-}Rel\text{-}pour\text{-}tous(q, f) \longrightarrow Info\text{-}Rel(q, f)\end{aligned}$$

Le méta prédicat $Info\text{-}Rel(q, f)$ exprime que la formule f fournit des informations pertinentes pour la requête q.

La règle S10 indique que, si l'attribut f est pertinent pour certains types d'entité mais pas pour tous, alors la valeur de l'attribut de f doit être fournie mais seulement pour les entités de type e. C'est la raison pour laquelle, la formule ajoutée à la liste des attributs à fournir contient un pseudo-implique.

$$\begin{aligned}&\text{S10} : \forall\,(q, e, v, f)\\ &\quad Att\text{-}Rel\text{-}pour\text{-}certains(q, \text{“}e(v)\text{”}, f) \wedge \neg Att\text{-}Rel\text{-}pour\text{-}tous(q, f)\\ &\qquad \longrightarrow Info\text{-}Rel(q, \text{“}e(v) \Longrightarrow f\text{”})\end{aligned}$$

7.3 Réponses complétives

7.3.1 Idée intuitive

Avant d'introduire la notion de réponses complétives, commençons par rappeler la définition "standard" de la réponse à une requête :

> Soit $F(X)$ une *fbf* d'un langage L. Les seules variables libres de F sont $X = x_1, .., x_n$.
> La réponse à la requête $F(X)$ dans une théorie T de L est l'ensemble des n-uplets de constantes $C = \{c_1, .., c_n\}$ tels que :
>
> $$T \vdash_L F(\sigma(C/X))$$
>
> où $\sigma(C/X)$ représente la substitution de X par C.

Nous avons déjà signalé, dans la section 5.3, qu'une telle définition ne fournit pas suffisamment d'informations à l'utilisateur (c'est la raison pour laquelle nous fournissons des conjonctions de formules atomiques comme réponses).

En effet, lorsque l'utilisateur pose comme requête une formule conjonctive sans quantificateur, comme par exemple :

$$\begin{aligned} \text{Q1} : \; & Vol(x) \wedge \\ & Ville\text{-}de\text{-}Départ(x, Paris) \wedge Ville\text{-}d'Arrivée(x, New\text{-}York) \\ & Heure\text{-}de\text{-}Départ(x, y) \end{aligned}$$

et si le système répond que le couple $< x = AF001, y = 10h00 >$ est une solution de cette requête, alors comme :

$$(f_1 \wedge f_2) \longrightarrow f_1$$

est une tautologie du calcul des prédicats, l'utilisateur peut déduire, pour cette réponse, la valeur de vérité de toutes les sous-formules de la requête Q. En particulier, il peut déduire que :

$$\begin{aligned} & T \vdash Vol(AF001) \\ & T \vdash Ville\text{-}de\text{-}Départ(AF001, Paris) \\ & T \vdash Ville\text{-}d'Arrivée(AF001, New\text{-}York) \\ & T \vdash Heure\text{-}de\text{-}Départ(AF001, 10h00) \end{aligned}$$

En revanche, si l'utilisateur pose la requête :

$$\begin{aligned} \text{Q2} : \; & Vol(x) \wedge \\ & Ville\text{-}de\text{-}Départ(x, Paris) \wedge \\ & (Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)) \wedge \\ & Heure\text{-}de\text{-}Départ(x, y) \end{aligned}$$

si le système répond que le couple $< x = AF001, y = 10h00 >$ est solution, comme :

$$(f_1 \vee f_2) \longrightarrow f_1$$

n'est pas une tautologie, l'utilisateur ne peut plus déduire si les formules $Ville\text{-}d'Arrivée(x, New\text{-}York)$ ou $Ville\text{-}d'Arrivée(x, Boston)$ sont des théorèmes de T (il se peut d'ailleurs, dans le cas d'une base de données incomplète, qu'aucune de ces deux formules ne soit théorème).

Pourtant, l'utilisateur aimerait très probablement distinguer parmi les solutions, les vols qui arrivent à New-York de ceux qui arrivent à Boston.

Dans le cas ci-dessus, l'utilisateur peut reformuler sa question pour obtenir l'heure d'arrivée dans la réponse :

Q3 : $Vol(x) \wedge$
$Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge Ville\text{-}d'Arrivée(x, z)$

Toutefois, cette solution n'est pas toujours applicable, comme on peut s'en rendre compte pour la requête suivante :

Q4 : $Vol(x)$
$(Ville\text{-}de\text{-}Départ(x, Paris) \vee Ville\text{-}d'Arrivée(x, New\text{-}York))$

Dans cette section, nous présentons une méthode qui complète automatiquement la réponse fournie à l'utilisateur. (que nous appelons réponse complétive). Ainsi pour la requête Q4, nous fournissons des réponses ayant la forme suivante :

1. $Vol(a_1) \wedge Ville\text{-}de\text{-}Départ(a_1, Paris) \wedge Ville\text{-}d'Arrivée(a_1, New\text{-}York)$
   pour les vols qui partent de Paris et qui arrivent à New-York.
2. $Vol(a_2) \wedge Ville\text{-}de\text{-}Départ(x, Paris)$
   pour les vols qui partent de Paris.
3. $Vol(a_3) \wedge Ville\text{-}d'Arrivée(x, New\text{-}York)$
   pour les vols qui arrivent à New-York.
4. $Vol(a_4)$
   pour les vols qui partent de Paris ou qui arrivent à New-York (cas d'une base de données incomplète dans laquelle on peut déduire $Ville\text{-}de\text{-}départ(x, Paris) \vee Ville\text{-}d'arrivée(x, New\text{-}York)$ sans pouvoir déduire ni $Ville\text{-}de\text{-}départ(x, Paris)$ ni $Ville\text{-}d'arrivée(x, New\text{-}York)$).

Dans le même ordre d'idée, si l'on considère le modèle Entité-Relation étendu en introduisant une structure hiérarchique sur les types d'entité, il est intéressant pour l'utilisateur de connaître le type des entités solutions. Ainsi, considérons la requête suivante :

Q5 : $Trajet(x) \wedge$
$Ville\text{-}de\text{-}Départ(x, Paris) \wedge Ville\text{-}d'Arrivée(x, Bruxelles)$

et si le système répond que $< x = T123 >$ est solution, cette information n'est pas utilisable par l'utilisateur; il est important de préciser si l'entité $T123$ est un vol ou un train.

Dans, notre système, nous fournissons des réponses ayant la forme suivante :

1. $Trajet(T123) \wedge Vol(T123)$
   si $T123$ est un Vol.
2. $Trajet(T123) \wedge Train(T123)$
   si $T123$ est un $Train$.
3. $Trajet(T123)$
   si $T123$ est un $Trajet$ (dans le cas d'une base de données où l'on ne sait pas si $T123$ est une entié de type Vol ou de type $Train$).

Nous considérons que ce type de réponses constitue également un cas particulier de réponse complétive.

Intuitivement, une réponse complétive doit être fournie à l'utilisateur chaque fois que la réponse "standard" ne constitue pas une information utilisable par l'utilisateur.

Dans cette section, nous abordons plusieurs situations dans lesquelles il est intéressant de fournir une réponse complétive. Nous commençons par étudier comment générer une réponse en utilisant la structure hiérarchique sur les types d'entités. Nous présentons ensuite une méthode permettant de fournir des réponses complétives respectivement dans le cas d'une formule où apparait des disjonctions, des quantificateurs ainsi que des négations.

7.3.2 Utilisation de la structure hiérarchique

Pour préciser , dans la réponse fournie à l'utilisateur, le type des entités solutions, nous utilisons la règle suivante :

S11 : $\forall\ (q, e, e', v)$
$Sujet\text{-}atom(q, \text{“}e(v)\text{”}) \wedge$
$ISA(e', e)$
$\longrightarrow Info\text{-}Rel(q, \text{“}e'(v) \Longrightarrow e'(v)\text{”})$

Cette règle indique que si une formule atomique $e(v)$ apparait dans la partie sujet d'une requête q, et si e' est un sous-type de e. alors pour les entités solutions ayant pour type e', il est pertinent de fournir cette information dans la réponse.

Par exemple, si le sujet de la requête est $Trajet(x)$, alors, en utilisant les faits $ISA(Trajet, Trajet)$, $ISA(Vol, Trajet)$ et $ISA(Train, Trajet)$, on va pouvoir déduire :

- $\vdash Info\text{-}Rel(q, \text{“}Trajet(x) \Longrightarrow Trajet(x)\text{”})$
- $\vdash Info\text{-}Rel(q, \text{“}Vol(x) \Longrightarrow Vol(x)\text{”})$
- $\vdash Info\text{-}Rel(q, \text{“}Train(x) \Longrightarrow Train(x)\text{”})$

Pour un x_0 solution, on va donc préciser dans la réponse s'il s'agit d'un *Trajet*, d'un *Vol* ou d'un *Train*.

7.3.3 Cas des conditions disjonctives

Nous nous intéressons maintenant au cas d'une réponse complétive lorsque la partie condition de la requête fait apparaître une formule disjonctive.

Ainsi, reprenons l'exemple de la partie 2.3 :

Quels sont les vols qui partent de Paris et qui arrivent à New-York **ou** *Washington ?*

Nous aimerions dans ce cas fournir à l'utilisateur une réponse qui précise, pour les solutions, la ville d'arrivée.

Dans notre formalisme, la question $qt1$ s'exprimerait de la façon suivante :

$Requête(qt1)$
$Sujet\text{-}i(qt1, \text{“}Vol(x)\text{”})$
$Condition\text{-}i(qt1, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee$
$Ville\text{-}d'Arrivée(x, Washington))\text{”})$
$Info\text{-}i(qt1, Vrai)$

Pour fournir dans la réponse, la ville d'arrivée des vols solutions, il suffit de transformer la partie Information Demandée de la question $qt1$ pour obtenir une nouvelle requête $qt2$ ayant la forme suivante :

$Requête(qt2)$
$Sujet\text{-}i(qt2, \text{“}Vol(x)\text{”})$
$Condition\text{-}i(qt2, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee$
$Ville\text{-}d'Arrivée(x, Washington))\text{”})$
$Info\text{-}i(qt2, \text{“}(Ville\text{-}d'Arrivée(x, New\text{-}York)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, New\text{-}York)) \wedge$
$(Ville\text{-}d'Arrivée(x, Washington)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, Washington))\text{”})$

Une traduction en langue naturelle de cette requête pourrait être :

Pour les vols qui partent de Paris et qui arrivent à New-York ou Washington, fournir la ville d'arrivée.

Il est intéressant de remarquer que l'on aurait pu également transformer la partie Information Demandée de $qt1$ pour obtenir :

$$Info\text{-}i(qt2,\ \text{“}Ville\text{-}d'Arrivée(x,y)\text{”})$$

On fournit ainsi la ville d'arrivée des solutions, ce que nous recherchions. Toutefois, cette transformation ne convient pas dans le cas général, par exemple lorsque l'on essaye de transformer une formule disjonctive telle que

$$\text{“}Ville\text{-}de\text{-}Départ(x, Paris) \vee Ville\text{-}d'Arrivée(x, Paris)\text{”}$$

où l'on veut fournir, suivant les cas, l'information ville de départ ou ville d'arrivée. Dans ce cas, nous allons rajouter à la partie Information Demandée de la requête initiale la formule suivante :

$$\begin{aligned}&\text{“}(Ville\text{-}de\text{-}Départ(x, Paris)\\ &\quad \Longrightarrow Ville\text{-}de\text{-}Départ(x, Paris))\wedge\\ &(Ville\text{-}d'Arrivée(x, Paris)\\ &\quad \Longrightarrow Ville\text{-}d'Arrivée(x, Paris))\text{”}\end{aligned}$$

On fournit ainsi la ville de départ s'il s'agit de Paris et la ville d'arrivée s'il s'agit de Paris.

Les axiomes permettant de réaliser cette transformation sont les suivants :

$$\begin{aligned}&\text{S12} : \forall\ (q, c, res)\\ &\qquad Condition\text{-}i(q, c)\wedge\\ &\qquad Sous\text{-}formule(\text{“}f_1 \vee f_2\text{”}, c)\wedge\\ &\qquad Construire(\text{“}f_1 \vee f_2\text{”}, res)\\ &\qquad\quad \longrightarrow Info\text{-}Rel(q, res)\end{aligned}$$

Cette règle exprime que si dans la partie condition d'une requête apparait une sous-formule disjonctive “$f_1 \vee f_2$” alors l'information res obtenue avec $Construire(\text{“}f_1 \vee f_2\text{”}, res)$ est une information pertinente à fournir à l'utilisateur.

L' axiome définissant le prédicat $Construire$ est le suivant :

$$\begin{aligned}&\text{S13} : \forall\ (f_1, f, res)\\ &\qquad Select(f_1, f) \wedge Info\text{-}Int(f_1, res)\\ &\qquad\quad \longrightarrow Construire(f, \text{“}f_1 \Longrightarrow res\text{”})\end{aligned}$$

Soit f une formule quelconque du calcul des prédicats et soit

$$f' = f_1 \vee f_2 \vee ... \vee f_n$$

sa forme normale disjonctive.

Alors $Select(x, f)$ va "sélectionner" chacune des sous-formules $f_1, f_2, ..., f_n$. On aura donc :

- $\vdash Select(f_1, f)$
- $\vdash Select(f_2, f)$
- ...
- $\vdash Select(f_n, f)$

Les axiomes définissant le prédicat *Select* sont les suivants :

S14 : $\forall\ (f_1, f_2, g_1, g_2)$
$Select(f_1, g_1) \wedge Select(f_2, g_2) \longrightarrow Select(\text{“}f_1 \wedge f_2\text{”}, \text{“}g_1 \wedge g_2\text{”})$

S15 : $\forall\ (f, g_1, g_2)$
$Formule(g_2) \wedge Select(f, g_1) \longrightarrow Select(f, \text{“}g_1 \vee g_2\text{”})$

S16 : $\forall\ (f, g_1, g_2)$
$Formule(g_1) \wedge Select(f, g_2) \longrightarrow Select(f, \text{“}g_1 \vee g_2\text{”})$

S17 : $\forall\ (f) Atome(f) \longrightarrow Select(f, f)$

Le prédicat $Info\text{-}Int(x, y)$ choisit dans la formule x les informations y qu'il est intéressant de fournir à l'utilisateur. Ce prédicat utilise seulement des critères syntaxiques pour reconnaître ces informations (En fait, y est construit avec les prédicats de type *attribut* qui apparaissent dans x).

7.3.4 Cas des quantificateurs

Nous nous intéressons maintenant au cas de réponse complétive lorsque la partie condition de la requête fait apparaître une formule quantifiée.

Quantificateur existentiel

Commençons par étudier le cas d'une formule positive (sans négation) quantifiée existentiellement. Le cas où dans la formule quantifiée apparait une négation est étudié dans la partie suivante.

Considérons par exemple la requête $qt3$:

Quels sont les vols qui partent de Paris et qui arrivent à New-York et pour lesquels il existe un jour où la tarification est Bleu ?

Nous aimerions dans ce cas fournir à l'utilisateur une réponse qui précise pour les solutions les jours où la tarification est Bleu.

Dans notre formalisme, la question $qt3$ s'exprimerait de la façon suivante :

$Requête(qt3)$
$Sujet\text{-}i(qt3, "Vol(x)")$
$Condition\text{-}i(qt3, "Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$\exists(y) Jour(y) \wedge Tarification(x, y, Bleu)")$
$Info\text{-}i(qt3, Vrai)$

Pour fournir, dans la réponse, les jours pour lesquels la tarification d'un vol solution est Bleu, il suffit de transformer la partie Information demandée de la question $qt3$ pour obtenir une nouvelle requête $qt4$ ayant la forme suivante :

$Requête(qt4)$
$Sujet\text{-}i(qt4, "Vol(x)")$
$Condition\text{-}i(qt4, "Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$\exists(y) Jour(y) \wedge Tarification(x, y, Bleu)")$
$Info\text{-}i(qt4, "(Jour(y_1) \wedge Tarification(x, y_1, Bleu))$
$\Longrightarrow Tarification(x, y_1, Bleu)")$

Une traduction en langue naturelle de cette requête pourrait être :

Pour les vols qui partent de Paris et qui arrivent à New-York et pour lesquels il existe un jour où la tarification est Bleu, fournir la liste des jours pour lesquels la tarification est Bleu.

Ici encore, il est intéressant de remarquer qu'il faut générer une formule hétérogène. Ainsi, on pouvait envisager de transformer les parties Information Demandée et Condition de $qt3$ pour obtenir :

$Requête(qt4)$
$Sujet\text{-}i(qt4, "Vol(x)")$
$Condition\text{-}i(qt4, "Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$Jour(y) \wedge Tarification(x, y, Bleu)")$
$Info\text{-}i(qt4, "Tarification(x, y, Bleu)")$

On fournit ainsi les jours pour lesquels la Tarification est Bleu. Dans ce cas, c'est ce que nous recherchions. Toutefois, cette transformation ne convient pas dans le cas général. Ainsi, considérons la requête :

Requête(*qt*5)
Sujet-i(*qt*5, "$Vol(x)$")
Condition-i(*qt*5, "*Ville-de-Départ*$(x, Paris) \wedge$
Ville-d'Arrivée$(x, New\text{-}York) \wedge$
$(Opère(x, Lundi) \vee \exists(y) Jour(y) \wedge Tarification(x, y, Bleu))$")
Info-i(*qt*5, *Vrai*)

Une traduction en langue naturelle de cette requête pourrait être :

Quels sont les vols Paris/New-York qui opèrent le lundi ou pour lesquels il existe un jour où la tarification est Bleu ?

Dans ce cas, on veut fournir une réponse qui précise si un vol solution opère le Lundi ou bien la liste des jours pour lesquels la Tarification est Bleu (s'il en existe au moins un).

On va donc générer la requête transformée suivante :

Requête(*qt*6)
Sujet-i(*qt*6, "$Vol(x)$")
Condition-i(*qt*6, "*Ville-de-Départ*$(x, Paris) \wedge$
Ville-d'Arrivée$(x, New\text{-}York) \wedge$
$(Opère(x, Lundi) \vee \exists(y) Jour(y) \wedge Tarification(x, y, Bleu))$")
Info-i(*qt*6, "$(Opère(x, Lundi) \Longrightarrow Opère(x, Lundi)) \wedge$
$(Jour(y_1) \wedge Tarification(x, y_1, Bleu))$
$\Longrightarrow Tarification(x, y_1, Bleu)$")

Les axiomes permettant de réaliser cette transformation sont les suivants :

S18 : $\forall\ (q, c, l, f, f_1, res)$
Condition-i$(q, c) \wedge$
Sous-formule$(c,$ "$\exists l\ f$"$) \wedge$
Rename$(f, f_1) \wedge$
Construire(f_1, res)
$\longrightarrow$ *Info-Rel*(q, res)

Cette règle exprime que si dans la partie condition d'une requête apparait une sous-formule quantifiée existentiellement "$\exists l\ f$" alors l'information *res* obtenue avec *Construire*(f_1, res) (où f_1 est obtenue à partir de f en renommant les variables de la liste l) est une information pertinente à fournir à l'utilisateur.

Pour prendre en compte les formules quantifiées existentiellement, il faut aussi ajouter l'axiome suivant à la définition du prédicat *Select* :

S19 : $\forall\ (l, g_1, g_2, f)$
Rename$(l, g_1, g_2) \wedge$ *Select*(f, g_2)
$\longrightarrow$ *Select*$(f,$ "$\exists l\ g_1$"$)$

Quantificateur universel

Considérons maintenant la requête $qt7$ où apparait une formule positive (sans négation) quantifiée universellement :

$$\begin{array}{l}\textit{Requête}(qt7)\\ \textit{Sujet-i}(qt7, \text{“}Vol(x)\text{”})\\ \textit{Condition-i}(qt7, \text{“}\textit{Ville-de-Départ}(x, Paris) \wedge\\ \qquad\qquad \textit{Ville-d'Arrivée}(x, New\text{-}York) \wedge\\ \qquad\qquad \forall(y) Jour(y) \longrightarrow Opère(x, y)\text{”})\\ \textit{Info-i}(qt7, \text{“}\textit{Heure-de-Départ}(x, z)\text{”})\end{array}$$

Une traduction en langue naturelle de cette requête pourrait être :

Pour les vols Paris/New-York qui opèrent tous les jours de la semaine, fournir la valeur de l'attribut Heure de départ

Si le système indique que la formule :

$$Vol(AF001) \wedge \textit{Heure-de-Départ}(AF001, 10h00)$$

est une réponse à cette requête, alors l'utilisateur peut déduire :

$$T \vdash \forall(y) Jour(y) \longrightarrow Opère(AF001, y)$$

c'est-à-dire :

$$T \vdash \forall(y) \neg Jour(y) \vee Opère(AF001, y)$$

(où T est la théorie représentant la base de données)

En revanche, l'utilisateur ne peut pas déduire, pour une constante c donnée, si les formules $\neg Jour(c)$ ou $Opère(AF001, c)$ sont théorèmes. On pourrait donc fournir à l'utilisateur l'ensemble des constantes qui ne sont pas des jours. Toutefois, la formule $\neg Jour(y)$ est une formule qui n'est pas indépendante du domaine (cf. [26]), et nous considérons qu'une telle formule ne fournit pas des informations intéressantes pour l'utilisateur. On pourrait également fournir la liste des jours pour lesquels le vol opère. Ici encore nous considérons que ces informations ne sont pas intéressantes pour l'utilisateur, car ce dernier sait qu'un vol solution de la requête $qt7$ est un vol qui opère tous les jours.

Ce sont les raisons pour lesquels nous ne fournissons pas de réponse complétive à l'utilisateur dans le cas de la requête $qt7$.

En revanche, considérons la requete suivante :

$Requête(qt8)$
$Sujet\text{-}i(qt8, \text{“}Vol(x)\text{”})$
$Condition\text{-}i(qt8, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$\forall(y) Jour(y) \longrightarrow (Tarification(x, y, Bleu) \vee$
$Tarification(x, y, Blanc))\text{”})$
$Info\text{-}i(qt8, \text{“}Heure\text{-}de\text{-}Départ(x, z)\text{”})$

Une traduction en langue naturelle de cette requête pourrait être :

Pour les vols Paris/New-York pour lesquels la Tarification est quotidiennement soit Bleu, soit Blanc, fournir l'heure de départ

Dans ce cas, nous aimerions distinguer dans la réponse, les jours pour lesquels la Tarification est Bleu, de ceux pour lesquels la Tarification est Blanc.

Pour cela, il suffit de transformer la partie Information Demandée de la question $qt8$ pour obtenir une nouvelle requête $qt9$ ayant la forme suivante :

$Requête(qt9)$
$Sujet\text{-}i(qt9, \text{“}Vol(x)\text{”})$
$Condition\text{-}i(qt9, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$\forall(y) Jour(y) \longrightarrow (Tarification(x, y, Bleu) \vee$
$Tarification(x, y, Blanc))\text{”})$
$Info\text{-}i(qt9, \text{“}Heure\text{-}de\text{-}Départ(x, z) \wedge$
$Tarification(x, y, Bleu) \Longrightarrow Tarification(x, y, Bleu) \wedge$
$Tarification(x, y, Blanc) \Longrightarrow Tarification(x, y, Blanc)\text{”})$

Pour réaliser cette transformation, on utilise l'axiome suivant :

S20 : $\forall (q, c, l, f, f', f'', f_1, f_2, res)$
$Condition\text{-}i(q, c) \wedge$
$Sous\text{-}formule(c, \text{“}\forall l\ f\text{”}) \wedge$
$Sous\text{-}formule\text{-}dom\text{-}ind(f, f') \wedge$
$Sous\text{-}formule(f', \text{“}f_1 \vee f_2\text{”}) \wedge$
$Rename(f', f'') \wedge$
$Construire(f'', res)$
$\longrightarrow Info\text{-}Rel(q, res)$

Soit f une formule quelconque du calcul des prédicats et soit :

$$f' = f_1 \vee f_2 \vee \ldots \vee f_n$$

sa forme normale disjonctive.

Alors, *Sous-formule-dom-ind*(f, g) est vrai si g est obtenu en supprimant dans f' toutes les sous formules f_i qui ne sont pas des formules indépendantes du domaine.

Par exemple, si :

$$f = Jour(y) \longrightarrow (Tarification(x, y, Bleu) \vee Tarification(x, y, Blanc))$$

alors :

$$f' = \neg Jour(y) \vee Tarification(x, y, Bleu) \vee Tarification(x, y, Blanc)$$

et :

$$g = Tarification(x, y, Bleu) \vee Tarification(x, y, Blanc)$$

La règle S20 indique donc que si dans la partie condition d'une requête apparait une sous-formule quantifiée universellement "$\forall l\ f$", si f' est une formule satisfaisant la condition *Sous-formule-dom-ind*(f, f') et si, dans f' apparait une sous formule disjonctive, alors la formule *res* satisfaisant la condition *Construire*(f'', res) (où f'' est obtenue à partir de f' en renommant les variables de la liste l) est une information pertinente à fournir à l'utilisateur.

Pour prendre en compte les formules quantifiées existentiellement, il faut aussi ajouter l'axiome suivant à la définition du prédicat *Select* :

S21 : $\forall\ (f, g, g', g'', g_1, g_2)$
Sous-formule-dom-ind$(g, g') \wedge$
Sous-formule$(g', "g_1 \vee g_2") \wedge$
Rename$(g', g'') \wedge$
Select(f, g'')
$\longrightarrow$ *Select*$(f, "\exists l\ g")$

7.3.5 Cas de la négation

Enfin, nous présentons le cas de réponse complétive lorsque la partie condition de la requête fait apparaitre une formule négative.

Considérons tout d'abord la requete $qt10$:

Requête$(qt10)$
Sujet-i$(qt10, "Trajet(x)")$
Condition-i$(qt10,$ "*Ville-de-Départ*$(x, Paris) \wedge$
Ville-d'Arrivée$(x, Bruxelles) \wedge$
$\neg Train(x) \wedge \neg Tarification(x, Lundi, Rouge)")$
Info-i$(qt10, "Heure\text{-}de\text{-}Départ(x, z)")$

Une traduction de cette requête en langue naturelle pourrait être :

Pour les Trajets Paris/Bruxelles qui ne sont pas des trains et pour lesquels la Tarification le Lundi n'est pas Rouge, fournir l'heure de départ

Pour cette requête, nous ne fournisssons pas de réponse complétive à l'utilisateur. En effet si le système indique que la formule :

$$Trajet(AF001) \wedge Vol(AF001) \wedge Heure\text{-}de\text{-}Départ(AF001, 10h00)$$

est une réponse à cette requête, alors l'utilisateur peut déduire :

$$T \vdash \neg Train(AF001)$$

et :

$$T \vdash \neg Tarification(AF001, Lundi, Rouge)$$

En revanche, il ne peut pas déduire la Tarification du vol $AF001$ le Lundi, ni la liste des jours pour lesquels la Tarification est Rouge. On peut considérer que ces informations sont intéressantes mais dans la transformation que nous proposons, ces informations ne sont pas fournies à l'utilisateur.

Considérons maintenant une autre requête :

$Requête(qt11)$
$Sujet\text{-}i(qt11, \text{“}Trajet(x)\text{”})$
$Condition\text{-}i(qt11, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, Bruxelles) \wedge$
$\neg(Vol(x) \wedge \forall(y) Jour(y) \longrightarrow Tarification(x, y, Rouge))\text{”})$
$Info\text{-}i(qt11, \text{“}Heure\text{-}de\text{-}Départ(x, z)\text{”})$

Une traduction en langue naturelle de cette règle pourrait être :

Pour les Trajets Paris/Bruxelles qui ne sont pas des vols pour lesquels la Tarification est quotidiennement Rouge, fournir l'heure de départ

En appliquant la loi de Morgan, on a :

$$\neg(Vol(x) \wedge \forall(y) Jour(y) \longrightarrow Tarification(x, y, Rouge))$$
$$\longleftrightarrow (\neg Vol(x) \vee \exists(y)(Jour(y) \wedge \neg Tarification(x, y, Rouge)))$$

Par suite, on aimerait fournir une réponse précisant si un Trajet solution n'est pas un Vol ou bien la liste des jours lesquels la Tarification n'est pas Rouge (s'il en existe au moins un).

Pour cela, il suffit de générer la requête transformée $qt12$:

$Requête(qt12)$
$Sujet\text{-}i(qt12, \text{“}Trajet(x)\text{”})$
$Condition\text{-}i(qt12, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, Bruxelles) \wedge$
$\neg(Vol(x) \wedge \forall(y) Jour(y) \longrightarrow Tarification(x, y, Rouge))$
$Info\text{-}i(qt12, \text{“}Heure\text{-}de\text{-}Départ(x, z) \wedge$
$\neg Vol(x) \Longrightarrow \neg Vol(x) \wedge$
$(Jour(y) \wedge \neg Tarification(x, y, Rouge))$
$\Longrightarrow \neg Tarification(x, y, Rouge)\text{”})$

. L'axiome permettant de réaliser cette transformation est la suivant :

S22 : $\forall\ (q, c, f, res)$
$Condition\text{-}i(q, c) \wedge$
$Sous\text{-}formule(c, \text{“}\neg f\text{”}) \wedge$
$Construire(\text{“}\neg f\text{”}, res)$
$\longrightarrow Info\text{-}Rel(q, res)$

Il faut également ajouter à la définition du prédicat *Select*, l'axiome suivant :

S23 : $\forall\ (f, g)$
$Select2(f, g) \longrightarrow Select(\text{“}\neg f\text{”}, \text{“}\neg g\text{”})$

avec

S24 : $\forall(f_1, f_2, g_1, g_2)$
$Select2(f_1, g_1) \wedge Select2(f_2, g_2)$
$\longrightarrow Select2(\text{“}f_1 \vee f_2\text{”}, \text{“}g_1 \vee g_2\text{”})$

S25 : $\forall(f, g_1, g_2)$
$Formule(g_2) \wedge Select2(f, g_1)$
$\longrightarrow Select2(f, \text{“}g_1 \wedge g_2\text{”})$

S26 : $\forall(f, g_1, g_2)$
$Formule(g_1) \wedge Select2(f, g_2) \longrightarrow Select2(f, \text{“}g_1 \wedge g_2\text{”})$

S27 : $\forall(f, g)$
$Select(f, g) \longrightarrow Select2(\text{“}\neg f\text{”}, \text{“}\neg g\text{”})$

S28 : $\forall(l, g, g', g'', g_1, g_2, f)$
$Sous\text{-}formule\text{-}dom\text{-}ind(\text{“}\neg g\text{”}, \text{“}g'\text{”}) \wedge$
$Sous\text{-}formule(g', \text{“}g_1 \vee g_2\text{”}) \wedge$
$Rename(l, g', g'') \wedge Select2(f, \text{“}\neg g''\text{”})$
$\longrightarrow Select2(f, \text{“}\exists l\ g\text{”})$

7.4 Règles générales pour transformer la requête

Dans cette partie, nous définissons une transformation de la requête posée par l'utilisateur utilisant les informations dérivées sur le prédicat *Info-Rel*.

Pour construire la formule f qui est la conjonction de toute les formules f_i satisfaisant le prédicat $Info\text{-}Rel(q, e, f_i)$, on introduit le méta-prédicat :

$$List\text{-}Info\text{-}Rel(q, f)$$

Les axiomes définissant le prédicat *List-Info-Rel* sont les suivants :

S29 : $\forall\ (q, f)$
$List\text{-}Info\text{-}Rel'(q, Vrai, f) \longrightarrow List\text{-}Info\text{-}Rel(q, f)$

S30 : $\forall\ (q, f, f_1, f_2)$
$Info\text{-}Rel(q, f_1) \wedge$
$\neg Sous\text{-}formule(f_2, f_1) \wedge$
$Rename(f_1, f_1') / \wedge$
$List\text{-}Info\text{-}Rel'(q, \text{“}f_1' \wedge f_2\text{”}, f)$
$\longrightarrow List\text{-}Info\text{-}Rel'(q, f_2, f)$

S31 : $\forall\ (q, f)$
$Requête(q) \wedge Formule(f) \longrightarrow List\text{-}Info\text{-}Rel'(q, f, f)$

Soit $f_1, ..., f_n$ l'ensemble des informations pertinentes à fournir. Alors *List-Info-Rel* construit une formule f qui est la conjonction des formules $f_1, ..., f_n$.

En fait, *List-Info-Rel* réalise une sorte de fonction *"Group-By"* sur l'ensemble des formules satisfaisant *Info-Rel (q, f)*.

Le prédicat *Rename* renomme éventuellement les variables libres de la formule f_1 pour éviter d'introduire des liens artificiels entre les variables. En effet, s'il existe plusieurs attributs, disons a_1 et a_2, pertinents pour un même type d'entité e, les formules atomiques $a_1(v, v_1)$ et $a_2(v, v_1)$ doivent être ajoutées à la partie *Informations Demandées* de la requête. Ceci conduirait à introduire artificiellement un lien par l'intermédiaire de la variable v_1. Pour éviter cette situation, il est nécéssaire de renommer les variables lorsque la partie *Informations Demandées* de la requête transformée est générée. Par exemple, dans le cas ci-dessus, $a_2(v, v_2)$ est généré à la place de $a_2(v, v_1)$.

La règle S32 indique qu'une réponse coopérative à la requête q ayant pour Sujet s, Condition c et Informations Demandées f_1, est obtenue en recherchant la formule f_2 des informations supplémentaires pertinentes à fournir et en calculant les solutions d'une nouvelle requête q' identique à q sauf pour la partie Informations Demandées.

Le prédicat $Solution(q, res)$ exprime que res est une solution à la requête q.

S32 : $\forall\ (q, q', res, s, c, f_1, f_2)$

$Sujet\text{-}i(q, s) \wedge$
$Condition\text{-}i(q, c) \wedge$
$Info\text{-}i(q, f_1) \wedge$
$List\text{-}Info\text{-}Rel(q, f_2) \wedge$
$q' = "s \wedge c \wedge f_1 \wedge f_2" \wedge$
$Solution(q', res)$
$\longrightarrow Solution\text{-}Coop(q, res)$

7.5 Exemple de transformation

La simulation à la main du processus de dérivation appliqué à la requête $qt1$ nous permet d'obtenir une requête transformée de la façon suivante :

De : $Sujet\text{-}i(qt1, "Vol(x)")$
on déduit en utilisant S5 : $Sujet\text{-}atom(qt1, "Vol(x)")$
De R1, on déduit : $ISA(Vol, Vol)$
donc, en utilisant R5, on obtient : $\neg DIS(Vol, Vol)$
et par suite en utilisant S8 : $Sujet(qt1, "Vol(x)")$

On a : $Apparait(qt1,$ *Heure-de-Départ*$)$
$Att\text{-}Top($*Heure-de-Départ, Horaire*$)$
par suite en utilisant S1, on obtient : $Int\text{-}Top(qt1, Horaire)$

On a : $Attribut($*Heure-d'Arrivée, Trajet*$)$
$ISA(Vol, Trajet)$
donc, en utilisant R6, on obtient : $Attribut($*Heure-d'Arrivée, Vol*$)$
On a : Att-Top(Heure-d'Arrivée, Horaire)
par suite en utilisant S3, on a :
$Att\text{-}Rel(qt1, "Vol(x)",$ *"Heure-d'Arrivée*$(x, v_1)")$

De RD1, RD2, RD3 et de S4, on déduit :
$Att\text{-}Rel(qt1, "\neg Validité(x, 01/01, 31/12) \Longrightarrow Validité(x, v_1, v_1')")$
$Att\text{-}Rel(qt1, "\neg(\forall(y) Jour(y) \longrightarrow Opère(x, y))$
$\Longrightarrow Opère(x, v_1)")$
$Att\text{-}Rel(qt1, "\neg\neg Cher(x) \Longrightarrow Prix(x, v_1)")$

De DEF1 et S9, et après simplification des formules, on obtient :
$Info\text{-}Rel(qt1, "Heure\text{-}d'Arrivée(x, v_1)")$
$Info\text{-}Rel(qt1, "Vol(x)", "Cher(x) \Longrightarrow Prix(x, v_1)")$
$Info\text{-}Rel(qt1, "\exists(y)(Jour(y) \wedge \neg Opère(x, y))$
$\Longrightarrow Opère(x, v_1)")$
$Info\text{-}Rel(qt1, "\neg Validité(x, 01/01, 31/12) \Longrightarrow Validité(x, v_1, v_1')")$

En appliquant les règles S14-S17, on déduit :
$Select("Ville\text{-}d'Arrivée(x, New\text{-}York)",$

"$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston))$"

$Select($"$Ville\text{-}d'Arrivée(x, Boston)$",

"$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston))$"

et comme :

$Info\text{-}Int($"$Ville\text{-}d'Arrivée(x, New\text{-}York)$", "$Ville\text{-}d'Arrivée(x, New\text{-}York)$"$)$

$Info\text{-}Int($"$Ville\text{-}d'Arrivée(x, Boston)$", "$Ville\text{-}d'Arrivée(x, Boston)$"$)$

on obtient, en utilisant S13 :

$Construire($"$Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)$",
"$Ville\text{-}d'Arrivée(x, New\text{-}York)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, New\text{-}York)$"$)$

$Construire($"$Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)$",
"$Ville\text{-}d'Arrivée(x, Boston)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, Boston)$"$)$

et par suite, de S12, on déduit :

$Info\text{-}Rel(qt1,$ "$Ville\text{-}d'Arrivée(x, New\text{-}York)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, New\text{-}York)$"$)$

$Info\text{-}Rel(qt1,$ "$Ville\text{-}d'Arrivée(x, Boston)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, Boston)$"$)$

En utilisant S29, S30 et S31, et après renommage des variables, on obtient :

$List\text{-}Info\text{-}Rel(qt1,$ "$Heure\text{-}d'Arrivée(x, y_1) \wedge$
$(\exists(y_2) Jour(y_2) \wedge \neg Opère(x, y_2)) \Longrightarrow Opère(x, y_3) \wedge$
$\neg Validité(x, 01/01, 31/12) \Longrightarrow Validité(x, y_4, z) \wedge$
$Cher(x) \Longrightarrow Prix(x, y_5) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$Ville\text{-}d'Arrivée(x, Boston)$
$\Longrightarrow Ville\text{-}d'Arrivée(x, Boston)$"$)$

Après renommage de la variable y dans $Prix(x, y)$, on obtient en utilisant S32, une nouvelle requête $qt2$:

$Requête(qt2)$

$Sujet\text{-}i(qt1,$ "$Vol(x)$"$)$

$Condition\text{-}i(qt2,$ "$Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$(Ville\text{-}d'Arrivée(x, New\text{-}York) \vee Ville\text{-}d'Arrivée(x, Boston)) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)$"$)$

$Info\text{-}i(qt2,$ "$Heure\text{-}de\text{-}Départ(x, y) \wedge$
$Heure\text{-}d'Arrivée(x, y_1) \wedge$
$(\exists(y_2) Jour(y_2) \wedge \neg Opère(x, y_2)) \Longrightarrow Opère(x, y_3) \wedge$
$\neg Validité(x, 01/01, 31/12) \Longrightarrow Validité(x, y_4, z) \wedge$
$Cher(x) \Longrightarrow Prix(x, y_5) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \Longrightarrow Ville\ d'Arrivée(x, New\text{-}York) \wedge$
$Ville\text{-}d'Arrivée(x, Boston) \Longrightarrow Ville\ d'Arrivée(x, Boston)$"$)$

Chapitre 8

Transformation des conditions

8.1 Principe général

Nous présentons dans cette partie, le formalisme choisi pour réaliser la transformation qui a été appelée dans les parties précédentes "transformation des conditions". Ce problème est également présenté dans [23].

Le problème consiste à définir une transformation de la requête initiale Q en une (ou plusieurs) requêtes Q' qui fournissent des réponses "proches" ou "voisines" des solutions de la requête initiale.

D'un côté, on fournit à l'utilisateur les solutions exactes en répondant à la requête Q, et de l'autre, on propose à l'utilisateur des réponses voisines en répondant à l'ensemble des requêtes transformées. Pour que l'utilisateur puisse distinguer aisément les solutions exactes des réponses voisines, dans la transformation que nous proposons, l'ensemble des solutions exactes et celui des réponses voisines constituent des ensembles disjoints.

En fait, cette transformation peut se décomposer en deux parties :

1. Dans une première phase, on transforme la partie condition C de la requête initiale pour obtenir une nouvelle requête Q' ayant pour condition C'. Les parties "sujet" et "informations demandées" de Q' sont identiques à Q.

2. Dans une deuxième phase, on transforme la partie "informations demandées" de Q' pour fournir la valeur des conditions modifiées lors de la première phase (sinon, l'utilisateur ne va pas savoir comment ces conditions ont été changées).

Ainsi, si l'on reprend l'exemple de la partie 2.1, pour la requête $qt1$ suivante :

Requête($qt1$)

Sujet-i($qt1$, "$Vol(x)$")

$$\begin{aligned}&\textit{Condition-i}(qt1,\ \text{“}\textit{Ville-de-Départ}(x, Paris) \wedge\\ &\qquad\qquad \textit{Ville-d'Arrivée}(x,\ \textit{New-York}) \wedge\\ &\qquad\qquad \textit{Heure-de-Départ}(x, y) \wedge (8 < y) \wedge (y < 12)\text{”})\end{aligned}$$

$$\textit{Info-i}(qt1,\ \text{“}\textit{Heure-de-Départ}(x, y)\text{”})$$

le serveur pourrait proposer des vols ayant une ville de départ proche de Paris et une ville d'arrivée proche de New-York. On peut par exemple considérer que Paris et New-York peuvent être remplacées par deux autres villes à condition que la distance entre ces deux villes ne soit pas supérieur de 10% à la distance Paris/New-York.

Comme les villes de départ et d'arrivée sont modifiées lors de cette transformation, il faut, dans un deuxième temps, transformer la partie "informations demandées" de la requête pour fournir la valeur de ces attributs.

Ceci conduit à la nouvelle requête modifiée $qt2$:

$$\textit{Requête}(qt2)$$

$$\textit{Sujet-i}(qt2, \text{“}\textit{Vol}(x)\text{”})$$

$$\begin{aligned}&\textit{Condition-i}(qt2, \text{“}\textit{Ville-de-Départ}(x, v_1) \wedge\\ &\qquad\qquad \textit{Ville-d'Arrivée}(x, v_2) \wedge\\ &\qquad\qquad \textit{Trajet-Voisin}(v_1, v_2, Paris, New - York) \wedge\\ &\qquad\qquad \textit{Heure-de-Départ}(x, y) \wedge (8 < y) \wedge (y < 12)\text{”})\end{aligned}$$

$$\begin{aligned}&\textit{Info-i}(qt2,\ \text{“}\textit{Heure-de-Départ}(x, y) \wedge\\ &\qquad\qquad \textit{Ville-de-Départ}(x, v_1) \wedge\\ &\qquad\qquad \textit{Ville-d'Arrivée}(x, v_2)\text{”})\end{aligned}$$

où *Trajet-Voisin* est un prédicat du langage objet définissant les villes pour lesquelles deux trajets sont voisins.

Dans la suite de ce chapitre, nous présentons les différentes informations utilisées pour réaliser cette transformation. La première section introduit la notion de voisinage entre deux conditions ainsi que les règles définissant le contexte dans lequel cette notion peut être appliquée. Nous montrons également comment définir et utiliser des critères d'optimisation qui permettent de restreindre "intelligemment" l'ensemble des entités voisines. Nous présentons ensuite les règles générales permettant de transformer la requête initiale en un ensemble de requêtes qui fournissent des réponses voisines. Enfin, dans la dernière section, nous montrons comment un mécanisme d'inférence utilisant ces connaissances peut dériver les requêtes fournissant les réponses voisines.

8.2 Caractérisation des conditions voisines

Pour représenter la notion de voisinage entre conditions, nous introduisons le méta prédicat :

$$Cond\text{-}Vois(q, E(x), C_1, C_2)$$

pour exprimer que, pour une requête q dont le sujet concerne les entités de type E, la formule C_1 peut être remplacée par la formule C_2.

En utilisant ce prédicat, on peut représenter la connaissance suivante :

$$\begin{aligned}
&\text{RT1} : \forall\ (q, x, v_1, v_2)\\
&\quad Sujet(q, \text{“}Trajet(x)\text{”}) \wedge\\
&\quad Condition(q,\ \text{“}Ville\text{-}de\text{-}Départ(x, v_1) \wedge Ville\text{-}d'Arrivée(x, v_2)\text{”})\\
&\qquad \longrightarrow Cond\text{-}Vois(q,\ \text{“}Trajet(x)\text{”},\\
&\qquad\qquad \text{“}Ville\text{-}de\text{-}Départ(x, v_1) \wedge\\
&\qquad\qquad Ville\text{-}d'Arrivée(x, v_2)\text{”},\\
&\qquad\qquad \text{“}Ville\text{-}de\text{-}Départ(x, v_1') \wedge\\
&\qquad\qquad Ville\text{-}d'Arrivée(x, v_2') \wedge\\
&\qquad\qquad Trajet\text{-}Voisin(v_1', v_2', v_1, v_2)\text{”})
\end{aligned}$$

Le prédicat $Condition(q, f)$ est utilisé pour tester si la formule f est une sous-formule de la partie condition de la requête q.

On remarquera que dans cette règle x, v_1, v_2 sont des méta-variables alors que v_1' et v_2' sont des constantes codant des variables objets.

En langue naturelle la traduction de cette règle pourrait être :

RT1 : **Si** une requête concerne les Trajets,
Et il y a une partie condition sur une Ville-de-départ donnée et sur une Ville-d'arrivée donnée
Alors ces villes peuvent être remplacées par d'autres villes définies par le prédicat du langage objet *Trajet-Voisin*.

Nous donnons un autre exemple de règle qui cette fois-ci transforme les conditions sur l'heure de départ :

$$\begin{aligned}
&\text{RT2} : \forall\ (q, x, h)\\
&\quad Sujet(q, \text{“}Trajet(x)\text{”}) \wedge\\
&\quad Condition(q,\ \text{“}Heure\text{-}de\text{-}Départ(x, h)\text{”})\\
&\qquad \longrightarrow Cond\text{-}Vois(q,\ \text{“}Trajet(x)\text{”},\\
&\qquad\qquad \text{“}Heure\text{-}de\text{-}Départ(x, h)\text{”},\\
&\qquad\qquad \text{“}Heure\text{-}de\text{-}Départ(x, h') \wedge |h' - h| < 15mn\text{”})
\end{aligned}$$

Une traduction de cette règle en langue naturelle pourrait être :

RT2 : **Si** une requête concerne les Vols,
Et il y a une partie condition sur une Heure-de-départ donnée
Alors cette heure peut être remplacée par une autre heure ne différant pas de plus d'un quart d'heure de l'heure initiale.

Dans les règles RT1 et RT2, les conditions de voisinage sont définies en remplaçant des constantes (les villes et l'heure de départ) par d'autres constantes. Ces constantes voisines sont définies par un ensemble explicite (dans le cas des villes) ou par un ensemble implicite (dans le cas de l'heure de départ).

On peut également utiliser la notion de voisinage pour exprimer que deux prédicats différents représentent des conditions voisines. Par exemple, si l'on introduit, dans le langage objet, les prédicats *Première-Classe*(x, y) et *Classe-Affaire*(x, y) pour représenter le prix du ticket en première classe ou en classe affaire, alors la règle suivante indique que ces prédicats imposent des conditions voisines :

RT3 : $\forall$ (q,x,y)
$Sujet(q, \text{"}Vol(x)\text{"}) \wedge$
$Condition(q, \text{"}Première\text{-}Classe(x,y)\text{"})$
$\longrightarrow Cond\text{-}Vois(q, \text{"}Première\text{-}Classe(x,y)\text{"},$
$\text{"}Classe\text{-}Affaire(x,y)\text{"})$

Cette règle peut être utilisée pour la requête :

Quelle est la compagnie aérienne assurant des vols de Paris à New-York tels que le prix du ticket en première classe soit inférieur à 10.000 francs ?

de façon à remplacer la condition sur le prix en première classe par une condition sur le prix en classe affaire. On peut remarquer que ce type de voisinage ne peut être définie par la méthode présentée dans [48].

Les règles RT1, RT2 et RT3 sont spécifiques à un domaine d'application. La règle suivante C1 est générale, et exprime que la propriété de voisinage est héritée par les types d'entités plus spécifiques.

C1 : $\forall$ (q, c_1, c_2, e, e', x)
$Cond\text{-}Vois(q, \text{"}e(x)\text{"}, c_1, c_2) \wedge ISA(e', e)$
$\longrightarrow Cond\text{-}Vois(q, \text{"}e'(x)\text{"}, c_1, c_2)$

Toutes les règles présentées dans cette section permettent de dériver, pour une application et une requête données, les entités voisines de celles explicitement demandées par l'utilisateur.

Dans notre approche, la notion de voisinage est définie à l'aide de connaissances sémantiques contrairement à d'autres approches fondées sur des règles de calcul prédéfinies, comme les ensembles flous [27].

Ainsi, dans l'axiomatique du prédicat *Cond-Vois* que nous donnons, on peut remarquer, qu'en dehors de la règle C1, on ne définit pas de propriété générale pour la relation de voisinage. En particulier, on ne fait pas l'hypothèse que cette relation est :

- **Réflexive**, c'est-à-dire :

$$\forall(q,e,f)Sujet(q,e) \wedge Condition(q,f) \longrightarrow Cond\text{-}Vois(q,e,f,f)$$

Nous n'avons pas retenu cette règle car son seul intérêt est de permettre de retrouver les solutions exactes de la requête initiale dans les réponses voisines. Or la transformation que nous proposons essaye justement de rendre disjoints ces deux ensembles de réponses.

- **Symétrique**, c'est-à-dire :

$$\forall(q,e,c,c')Cond\text{-}Vois(q,e,c,c') \longleftrightarrow Cond\text{-}Vois(q,e,c',c)$$

En effet, nous raisonnons en terme de transformation de formules, le prédicat $Cond\text{-}Vois(q,e,c,c')$ exprimant que pour la requête q la formule c peut être remplacée par c'. Ceci ne suffit pas en général pour déduire que pour la requête q, la formule c' peut être remplacée par la formule c. En particulier, ceci n'aurait pas de sens dans le cas des règles RT1 et RT2.

En revanche, si l'on considère que pour la règle RT3, le rôle joué par les prédicats *Première-Classe*(x,y) et *Classe-Affaire*(x,y) est symétrique, alors il faut introduire une nouvelle règle :

RT4 : $\forall$ *(q,x,y)*
Sujet$(q,$ "*Vol*(x)"$) \wedge$
Condition$(q,$ "*Classe-Affaire*(x,y)"$)$
$\longrightarrow$ *Cond-Vois*$(q,$ "*Classe-Affaire*(x,y),
"*Première-Classe*(x,y)"$)$

qui permet de remplacer une condition sur le prix en première classe par une condition sur le prix en classe affaire.

- **Transitive.** Il serait en effet absurde de supposer que des requêtes voisines des voisins d'une requête initiale q sont encore des requêtes voisines de la requête q.

Toutefois, la transitivité est une propriété intéressante sur laquelle nous revenons dans la section 8.4.

8.3 Critère d'optimisation

Dans de nombreux cas, la notion d'intérêt des entités voisines dépend de la valeur d'un certain critère choisi pour comparer les entitiés. Ces critères dépendent fortement du domaine d'application, et de chaque contexte. En particulier, ils peuvent dépendre des caractéristiques de l'utilisateur.

Par exemple, pour certains types d'utilisateur, un critère d'optimisation peut être le prix du voyage, alors que pour d'autres il peut s'agir de sa durée.

L'idée consiste donc à éviter de fournir des entités voisines dont la valeur du critère est plus grande que celle de la plus petite valeur des réponses à la requête initiale (dans le cas où l'on cherche à minimiser ce critère).

Dans cette section, nous présentons les prédicats utilisés pour représenter cette notion de critère d'optimisation ainsi que les règles associées.

Pour introduire un nouveau critère, on utilise le méta prédicat :

$$Opt\text{-}Criterium(q, E(x), C)$$

pour exprimer que, pour la requête q concernant les entités de type E, C est une formule caractérisant un critère d'optimisation.

Par exemple, la règle Cr1 indique que pour les entités de type *Trajet*, le *Prix* peut être un critère d'optimisation. Ainsi, si la partie entité de la requête initiale q concerne les entités de type E, des entités voisines de type *Trajet* seront fournies si leur prix est inférieur à celui des solutions initiales (caractérisé par la formule $E(x') \wedge C'$).

La formule C' est une variante de la partie condition C dans laquelle les variables libres $X = x_1, \ldots, x_n$ de C ont été renommées en $X' = x'_1, \ldots, x'_n$.

$$\begin{aligned}
&\text{Cr1} : \forall\ (q, x, E, C, C', X') \\
&\quad Sujet(q, \text{“}E(x)\text{”}) \wedge Condition\text{-}i(q, C) \wedge Variant(C, C', X') \\
&\qquad \longrightarrow Opt\text{-}Criterium(q, \text{“}Trajet(x)\text{”}, \\
&\qquad\qquad \text{“}Prix(x, y) \wedge \\
&\qquad\qquad \forall\ (x', y', X') \\
&\qquad\qquad\quad E(x') \wedge Prix(x', y') \wedge C' \longrightarrow y < y'\text{”})
\end{aligned}$$

Dans la règle Cr2, nous avons considéré que la durée du vol est un critère d'optimisation particulièrement intéressant pour les *hommes d'affaires.*

$$\begin{aligned}
&\text{Cr2} : \forall\ (q, x, E, C, C', X') \\
&\quad Sujet(q, \text{“}E(x)\text{”}) \wedge Condition\text{-}i(q, C) \wedge Variant(C, C', X') \wedge \\
&\quad Type\text{-}Utilisateur(Homme\text{-}d'Affaires) \\
&\qquad \longrightarrow Opt\text{-}Criterium(q, \text{“}Vol(x)\text{”}, \\
&\qquad\qquad \text{“}Durée(x, y) \wedge \\
&\qquad\qquad \forall\ (x', y', X') \\
&\qquad\qquad\quad E(x') \wedge Durée(x', y') \wedge C' \longrightarrow y < y'\text{”})
\end{aligned}$$

Il faut également ajouter la règle Cr3. qui est indépendante d'un domaine d'application particulier, pour exprimer que le critère d'optimisation est hérité par les types d'entités plus spécifiques.

$$\begin{aligned}
&\text{Cr3} : \forall\ (q, e_1, e'_1, c, x) \\
&\quad Opt\text{-}Criterium(q, \text{“}e_1(x)\text{”}, c) \wedge ISA(e'_1, e_1) \\
&\qquad \longrightarrow Opt\text{-}Criterium(q, \text{“}e'_1(x)\text{”}, c)
\end{aligned}$$

En plus des règles Cr1, Cr2 et Cr3 qui définissent la notion de critère d'optimisation, il existe des règles générales qui indiquent comment utiliser ce critère lorsque l'on transforme une requête pour fournir des réponses voisines. Ces règles sont présentées dans la section suivante.

8.4 Transformation de la requête

Dans cette partie, nous définissons une transformation de la requête posée par l'utilisateur utilisant les informations dérivées sur le prédicat *Cond-Vois*.

Intuitivement, dans la section 8.2, nous avons montré comment déterminer les informations voisines intéressantes. Dans cette partie, nous définissons comment ces informations peuvent être effectivement fournies à l'utilisateur.

Pour réaliser cette transformation, on introduit dans le méta-langage le prédicat suivant :

$$Cond\text{-}Rel(q, c_1, c_2, c)$$

où, partant de la requête initiale q, c est la nouvelle partie condition obtenue lorsque la formule c_1 peut être remplacée par la formule c_2.

Les faits dérivés sur le prédicat *Cond-Vois* sont utilisés dans la règle C2 pour déduire cette nouvelle condition :

$$\begin{aligned}
&\text{C2} : \forall\ (q, e, c, c', c_1, c_2) \\
&\quad Condition\text{-}i(q, c) \wedge \\
&\quad Cond\text{-}Vois(q, e, c_1, c_2) \wedge \\
&\quad Rel\text{-}pour\text{-}Tous(q, e) \wedge \\
&\quad Substitute(c_1, c_2, c, c') \\
&\qquad \longrightarrow Cond\text{-}Rel(q, c_1, c_2, c')
\end{aligned}$$

Le prédicat *Substitute* est utilisé pour remplacer dans c, la sousformule c_1 par c_2 afin d'obtenir c'.

Le prédicat *Rel-pour-Tous* est défini par :

$$\begin{aligned}
&\text{DEF} : \forall\ (q, e, x) \\
&\quad Rel\text{-}pour\text{-}Tous(q, \text{“}e(x)\text{”}) \\
&\qquad \longleftrightarrow (\forall(e') Sujet\text{-}Atomique(q, \text{“}e'(x)\text{”}) \longrightarrow ISA(e', e))
\end{aligned}$$

Ce prédicat exprime que toutes les formules atomiques apparaissant dans la partie *sujet* de q sont des sous-types du type d'entité e. Avec ce prédicat, la transformation définie par la règle C2 est restreinte aux cas où le voisinage entre c_1 et c_2 est valide pour tous les types d'entités apparaissant dans la requête q.

Par exemple, si c_1 est voisin de c_2 pour les seules entités de type *Vol*, et si la partie *sujet* de la requête q est : $Vol(x) \vee Train(x)$, la transformation définie par la règle C2

ne peut être appliquée. En revanche, si la partie sujet est : $Vol\text{-}Air\text{-}France(x) \vee Vol\text{-}TWA(x)$, comme ces deux types d'entité sont des sous-types de *Vol*, alors on peut appliquer la transformation.

La règle C3 définit la transformation dans le cas où *Rel-pour-Tous* ne s'applique pas. Dans ce cas, la transformation des conditions ne s'applique qu'aux entités satisfaisant le prédicat *Cond-Vois*. Cette remarque nous conduit à substituer "$e \wedge c_2$" à c_1, à la place de c_2.

$$
\begin{aligned}
\text{C3} : \forall\ & (q, e, c, c', c_1, c_2) \\
& Condition\text{-}i(q, c) \wedge \\
& Cond\text{-}Vois(q, e, c_1, c_2) \wedge \\
& Sujet(q, e) \wedge \\
& \neg Rel\text{-}pour\text{-}Tous(q, e) \wedge \\
& Substitute(c_1, \text{“}e \wedge c_2\text{”}, c, c') \\
& \quad \longrightarrow Cond\text{-}Rel(q, c_1, c_2, c, c')
\end{aligned}
$$

La formule obtenue en appliquant la transformation définie par le prédicat *Cond-Rel* ne correspond pas tout-à-fait à la partie condition de la requête transformée que l'on désire obtenir. En effet, on peut parfois utiliser un critère d'optimisation pour restreindre l'ensemble des entités voisines que l'on fournit en réponse (cf. partie précédente).

Pour obtenir la partie condition de la requête transformée en tenant compte de ce critère d'optimisation, on utilise le prédicat :

$$Transf\text{-}Cond(q, c_1, c_2, c)$$

où, partant de la requête initiale q, c est la partie condition de la requête transformée (obtenue en remplaçant la formule c_1 par la formule c_2).

La règle C4 indique simplement que la formule C représentant le critère d'optimisation doit être ajoutée à la formule obtenue en appliquant la transformation définie par le prédicat *Cond-Vois*.

$$
\begin{aligned}
\text{C4} : \forall\ & (q, e, c_1, c_2, c, c', C, C') \\
& Cond\text{-}Vois(q, e, c_1, c_2) \wedge \\
& Cond\text{-}Rel(q, c_1, c_2, c) \wedge \\
& Rename(c, c') \wedge \\
& Opt\text{-}Criterium(q, e, C) \wedge \\
& Rename(C, C') \\
& \quad \longrightarrow Transf\text{-}Cond(q, c_1, c_2, \text{“}c' \wedge C'\text{”})
\end{aligned}
$$

Le prédicat *Rename* renomme éventuellement les variables libres des formules c et C pour éviter d'introduire des liens artificiels entre les variables. Par exemple, lorsqu'on applique la règle Cr1, on ajoute un critère d'optimisation où apparait la variable y (dans $Prix(x, y)$). Comme cette variable est déjà utilisée dans la requête $qt1$ (dans *Heure-de-départ*(x, y)), il faut renommer la variable y dans $Prix(x, y)$ et insérer la formule $Prix(x, y_1)$.

Dans le cas où il n'y a pas de critère d'optimisation applicable, il est quand même intéressant de restreindre l'ensemble des réponses voisines à celles qui ne sont pas déjà solutions de la requête initiale. Il suffit, pour cela, d'indiquer que les réponses voisines ne doivent pas satisfaire les conditions de la requête initiale. C'est ce qu'exprime la règle C5 :

$$
\begin{aligned}
\text{C5} : \forall\ & (q, e, c_1, c_2, c, c', C, C_1, C_2) \\
& \textit{Cond-Vois}(q, e, c_1, c_2) \wedge \\
& \textit{Cond-Rel}(q, c_1, c_2, c) \wedge \\
& \textit{Rename}(c, c') \wedge \\
& \neg \textit{Opt-Criterium}(q, e, C) \wedge \\
& \textit{Condition-i}(q, C_1) \wedge \\
& \textit{Variant}(C_1, C_2, X') \\
& \quad \longrightarrow \textit{Transf-Cond}(q, c_1, c_2, \text{“}c' \wedge \neg\exists(X')C_2\text{”})
\end{aligned}
$$

La formule C_2 est une variante de la formule C_1 dans laquelle les variables libres ont été renommées en $X' = x'_1, \ldots, x'_n$.

Lorsque des entités satisfaisant des conditions voisines sont fournies, on doit également fournir la valeur des conditions modifiées, car l'utilisateur ne sait pas comment elles ont été changées.

Par exemple, on doit fournir la valeur de la ville de départ et de la ville d'arrivée lorsque la règle RT1 est utilisée, et celle de l'heure de départ pour la règle RT2.

Pour fournir ces informations, on utilise le méta prédicat :

$$\textit{Att-Int}(q, c_1, c_2, I)$$

où I est une information intéressante à fournir avec les réponses voisines lorsque la condition c_1 est modifiée en c_2.

La règle C6 définit la valeur des attributs à fournir lorsqu'une condition est modifiée. Le méta prédicat *Info-Int* utilise seulement des critères syntaxiques pour reconnaître les attributs intéressants (en fait, I est construit avec les prédicats de type ***attribut*** qui apparaissent dans c_2).

$$
\begin{aligned}
\text{C6} : \forall\ & (q, e, c_1, c_2, I) \\
& \textit{Cond-Vois}(q, e, c_1, c_2) \wedge \\
& \textit{Info-Int}(c_2, I) \\
& \quad \longrightarrow \textit{Att-Int}(q, c_1, c_2, I)
\end{aligned}
$$

Lorsqu'un critère d'optimisation est inséré dans la requête transformée, il est utile de fournir à l'utilisateur la valeur de l'attribut optimisé; ainsi, l'utilisateur aura la possibilité de comparer les avantages des entités voisines par rapport au critère d'optimisation.

Nous utilisons la règle C7, indépendante d'un domaine d'application particulier, pour exprimer déterminer les informations intéressantes à fournir dans ce cas.

$$\begin{aligned}
\text{C7} : \forall\ & (q, e, c_1, c_2, C, I) \\
& \mathit{Cond\text{-}Vois}(q, e, c_1, c_2) \wedge \\
& \mathit{Opt\text{-}Criterium}(q, e, C) \wedge \\
& \mathit{Info\text{-}Int}(C, I) \\
& \quad \longrightarrow \mathit{Att\text{-}Int}(q, c_1, c_2, I)
\end{aligned}$$

Il faut aussi remarquer que les informations comparatives doivent être fournies pour les réponses aux requêtes transformées, mais également pour les réponses à la requête initiale. Ceci est exprimé par la règle S33 :

$$\begin{aligned}
\text{S33} : \forall\ & (q, e, c_1, c_2, C, I) \\
& \mathit{Cond\text{-}Vois}(q, e, c_1, c_2) \wedge \\
& \mathit{Opt\text{-}Criterium}(q, e, C) \wedge \\
& \mathit{Info\text{-}Int}(C, I) \wedge \\
& \quad \longrightarrow \mathit{Att\text{-}Rel}(q, e, I)
\end{aligned}$$

Cette règle indique que pour la requête initiale q, si la formule c_1 peut être remplacée par la formule c_2 et si le critère d'opitmisation C peut être utilisé dans cette transformation, alors les informations comparatives I sont des attributs supplémentaires pertinents à fournir pour les entités de type e (Cf. la partie précédente pour une définition du prédicat *Att-Rel*).

Pour construire la formule f qui est la conjonction de toutes les formules f_i satisfaisant le prédicat $\mathit{Att\text{-}Int}(q, e, f_i)$, on introduit le méta prédicat :

$$\mathit{List\text{-}Att\text{-}Int}(q, c_1, c_2, f)$$

Les axiomes définissant le prédicat *List-Att-Int* sont les mêmes que ceux définissant le prédicat *List-Info-Rel* (Cf. partie précédente).

Enfin, la règle C8 indique que des réponses voisines de la requête initiale q, sont obtenues en recherchant une condition c_2 proche de c_1, en appliquant la transformation définie par le prédicat *Transf-Cond* et en recherchant la liste des valeurs d'attribut qu'il est alors intéressant de fournir à l'utilisateur. On construit ainsi une nouvelle requête q' et il n'y a plus alors qu'à rechercher les solutions de cette requête.

$$\begin{aligned}
\text{C8} : \forall\ & (q, q', s, c_1, c_2, f, res) \\
& \mathit{Sujet\text{-}i}(q, s) \wedge \\
& \mathit{Transf\text{-}Cond}(q, c_1, c_2, c) \wedge \\
& \mathit{Info\text{-}i}(q, f) \wedge \\
& \mathit{List\text{-}Att\text{-}Int}(q, c_1, c_2, f') \wedge \\
& q' = \text{“}s \wedge c \wedge f \wedge f'\text{”} \wedge \\
& \mathit{Solution}(q', res) \\
& \quad \longrightarrow \mathit{Solution\text{-}Vois}(q, res)
\end{aligned}$$

En fait, on peut appliquer à la requête q' ainsi obtenue la transformation sur les informations demandées définie dans la partie précédente pour fournir à l'utilisateur

la valeur d'attributs pertinents et non demandées. Pour cela, il suffit de remplacer dans les prémisses de la règle C8, le prédicat $Solution(q', res)$ par le prédicat $Solution\text{-}Coop(q', res)$ dont la définition a été donnée dans le chapitre précédent.

La transformation que nous venons de définir, nous conduit à faire les remarques suivantes :

1. Supposons que l'on puisse démontrer les deux théorèmes :

$$Cond\text{-}Vois(q, s, c_1, c'_1) \text{ et } Cond\text{-}Vois(q, s, c_2, c'_2)$$

Supposons également que le sujet de la requête q soit s et que la partie condition soit $C = c_1 \wedge c_2$.

Alors d'après l'axiome C2, on va pouvoir déduire :

$$Cond\text{-}Rel(q, c_1, c'_1, c'_1 \wedge c_2) \text{ et } Cond\text{-}Rel(q, c_2, c'_2, c_1 \wedge c'_2)$$

Pour simplifier notre propos, nous faisons l'hypothèse qu'aucun critère d'optimisation n'est applicable pour la requête q et nous allons également supposer que nous n'utilisons pas la règle C6. Par suite, en utilisant C8, on va directement générer deux nouvelles requêtes q_1 et q_2 ayant respectivement pour partie condition : $C_1 = c'_1 \wedge c_2$ et $C_2 = c_1 \wedge c'_2$.

En conclusion, dans la transformation des conditions que nous venons de définir, nous ne modifions qu'une seule condition à la fois.

En fait, nous aurions pu décider de générer une seule requête q' ayant pour partie condition : $C' = c'_1 \wedge c'2$.

Nous n'avons pas choisi cette solution, car nous considérons que les formules $c'_1 \wedge c_2$ et $c_1 \wedge c'_2$ sont plus proches de $c_1 \wedge c_2$ que la formule $c'_1 \wedge c'_2$.

Bien sur, nous ne supposons pas que la relation de voisinage est transitive, mais on peut remarquer que les réponses a la requête q' peuvent être obtenues en appliquant le processus de transformation des conditions, que nous venons de définir, aux requêtes q_1 et q_2. En effet, si l'on peut déduire les informations suivantes :

$$Cond\text{-}Vois(q, s, c'_1, c''_1) \text{ et } Cond\text{-}Vois(q, s, c'_2, c''_2)$$

alors d'après C2 et C8, on va pouvoir générer trois nouvelles requêtes q'_1, q'_2 et q'_3 ayant respectivement pour parties conditions : $c''_1 \wedge c_2$, $c_1 \wedge c''_2$ et $c'_1 \wedge c'_2$.

Ces requêtes sont des requêtes voisines des voisins de la requête initiale q.

On peut ainsi introduire la notion de degré de voisinage qui permet d'obtenir de plus en plus d'informations en propageant la notion de voisinage. Partant de la requête initiale q, les requêtes q_1 et q_2 sont des requêtes voisines au degré 1 alors que les requêtes q'_1, q'_2 et q'_3 sont des requêtes voisines au degré 2.

La propagation par transitivité de la notion de voisinage doit être contrôlée par l'utilisateur qui peut arrêter le processus lorsqu'il a obtenu les informations qui lui conviennent.

Remarque : en fait, notre langage de requête nous permet de représenter les requêtes q_1 et q_2 sous forme d'une seule requête Q ayant pour partie condition $(c'_1 \wedge c_2) \vee (c_1 \wedge c'_2)$. Toutefois, d'après les axiomes C6 et C7, les informations intéressantes à fournir ne sont pas les mêmes pour q_1 et q_2. Par suite, la formule Q est assez compliquée (on est obligé d'utiliser l'opérateur pseudo-implique). C'est la raison pour laquelle nous avons préféré générer plusieurs requêtes.

2. Supposons toujours que l'on puisse démontrer les deux théorèmes :

$$Cond\text{-}Vois(q, s, c_1, c'_1) \text{ et } Cond\text{-}Vois(q, s, c_2, c'_2)$$

et supposons maintenant que le sujet de la requête q soit s et que la partie condition soit $C = c_1 \vee c_2$.

Dans ce cas, nous allons générer deux requêtes q_1 et q_2 ayant respectivement pour partie condition c'_1 et c'_2 (ici encore, on aurait pu générer une seule requête Q ayant pour partie condition $c'_1 \vee c'_2$).

3. Supposons enfin que l'on puisse démontrer le théorème :

$$Cond\text{-}Vois(q, s, c, c')$$

et supposons que le sujet de la requête q soit s et que la partie condition soit $C = \neg c$.

Dans ce cas, nous n'allons pas générer une nouvelle requête q' ayant pour partie condition $\neg c'$. Cette nouvelle requête n'a en général pas d'intérêt. Ainsi, si l'on utilise les règles RT1 ou RT2, les réponses aux requêtes ainsi obtenues constituent des ensembles plus petits que l'ensemble des solutions de la requête initiale.

En fait, pour que l'on puisse transformer la formule $\neg c$, il faut que l'on puisse déduire explicitement :

$$Cond\text{-}Vois(q, s, \neg c, \neg c')$$

8.5 Exemple de transformation de requête

Dans cette partie, nous montrons comment un processus de dérivation appliqué à la requête $qt1$ nous permet d'obtenir une requête transformée fournissant des informations voisines intéressantes.

Pour simplifier les expressions, nous commençons par poser :

$$\begin{aligned}
c_1 &= Ville\text{-}de\text{-}Départ(x, Paris) \wedge Ville\text{-}d'Arrivée(x, New\text{-}York) \\
c_2 &= Ville\text{-}de\text{-}Départ(x, v'_1) \wedge Ville\text{-}d'Arrivée(x, v'_2) \wedge \\
&\quad Trajet\text{-}Voisin(v'_1, v'_2, Paris, New\text{-}York) \\
C' &= Ville\text{-}de\text{-}Départ(x', Paris) \wedge Ville\text{-}d'Arrivée(x', New\text{-}York) \wedge \\
&\quad Heure\text{-}de\text{-}Départ(x', y'') \wedge (8 < y'') \wedge (y'' < 12)
\end{aligned}$$

De : $Sujet\text{-}i(qt1, \text{“}Vol(x)\text{”})$
on déduit en utilisant R8 : $Sujet(qt1, \text{“}Trajet(x)\text{”})$.

De : $Condition\text{-}i(qt1, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)\text{”})$
on déduit : $Condition(qt1, c_1)$

Par suite, en utilisant RT1, on a :
$Cond\text{-}Vois(qt1, \text{“}Trajet(x)\text{”}, c_1, c_2)$

On a : $Rel\text{-}pour\text{-}Tous(qt1, \text{“}Trajet(x)\text{”})$
Par suite, en utilisant C2, on obtient :
$Cond\text{-}Rel(qt1, c_1, c_2,$
$\text{“}Ville\text{-}de\text{-}Départ(x, v_1') \wedge$
$Ville\text{-}d'Arrivée(x, v_2') \wedge$
$Trajet\text{-}Voisin(v_1', v_2', Paris, New\text{-}York) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)\text{”})$

De Cr1, on déduit que :
$Opt\text{-}Criterium(qt1, \text{“}Trajet(x)\text{”},$
$\text{“}Prix(x, y) \wedge$
$\forall\ (x', y', y'')$
$Vol(x') \wedge Prix(x', y') \wedge C' \longrightarrow y < y'\text{”})$
Par suite, en utilisant C4, on obtient (après renommage de la variable y dans $Prix(x, y)$) :
$Transf\text{-}Cond(qt1, c_1, c_2,$
$\text{“}Ville\text{-}de\text{-}Départ(x, v_1') \wedge$
$Ville\text{-}d'Arrivée(x, v_2') \wedge$
$Trajet\text{-}Voisin(v_1', v_2', Paris, New\text{-}York) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12) \wedge$
$Prix(x, y_1) \wedge$
$\forall\ (x', y', y'')$
$Vol(x') \wedge Prix(x', y') \wedge C' \longrightarrow y_1 < y'\text{”})$

En utilisant C6 on déduit :
$Att\text{-}Int(q, c_1, c_2, \text{“}Ville\text{-}de\text{-}Départ(x, v_1')\text{”})$
$Att\text{-}Int(q, c_1, c_2, \text{“}Ville\text{-}de\text{-}d'Arrivée(x, v_2')\text{”})$
et en utilisant C7 :
$Att\text{-}Int(q, c_1, c_2, \text{“}Prix(x, y)\text{”})$
et par suite, après renommage de la variable y dans $Prix(x, y)$, on obtient :
$List\text{-}Att\text{-}Int(q, c_1, c_2,$
$\text{“}Ville\text{-}de\text{-}Départ(x, v_1') \wedge$
$Ville\text{-}de\text{-}d'Arrivée(x, v_2') \wedge$
$Prix(x, y_1)\text{”})$

ce qui nous permet, en utilisant la règle C8, de générer la nouvelle requête $qt2$:

$Requête(qt2)$

$Sujet\text{-}i(qt2, \text{“}Vol(x)\text{”})$

$$Condition\text{-}i(qt2, \text{“}Ville\text{-}de\text{-}Départ(x, v_1') \wedge Ville\text{-}d'Arrivée(x, v_2') \wedge Trajet\text{-}Voisin(v_1', v_2', Paris, New\text{-}York) \wedge Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12) \wedge Prix(x, y_1) \wedge \forall (x', y', y'') \; Vol(x') \wedge Prix(x', y') \wedge C' \longrightarrow y_1 < y'\text{”})$$

$$Info\text{-}i(qt2, \text{“}Heure\text{-}de\text{-}Départ(x, y) \wedge Ville\text{-}de\text{-}Départ(x, v_1') \wedge Ville\text{-}d'Arrivée(x, v_2') \wedge Prix(x, y_1)\text{”})$$

Si à la place d'utiliser la règle RT1 on applique la règle RT2, de façon similaire, on génère une autre requête voisine $qt3$:

$Requête(qt3)$

$Sujet\text{-}i(qt3, \text{“}Vol(x)\text{”})$

$$Condition\text{-}i(qt3, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge Ville\text{-}d'Arrivée(x, New\text{-}York) \wedge Heure\text{-}de\text{-}Départ(x, h') \wedge |h' - y| < 15mn \wedge (8 < y) \wedge (y < 12) \wedge Prix(x, y_1) \wedge \forall (x', y', y'') \; Vol(x') \wedge Prix(x', y') \wedge C' \longrightarrow y_1 < y'\text{”})$$

$$Info\text{-}i(qt3, \text{“}Heure\text{-}de\text{-}Départ(x, h') \wedge Prix(x, y_1)\text{”})$$

On peut remarquer, que dans cette requête, la formule :

$$Heure\text{-}de\text{-}Départ(x, h') \wedge |h' - y| < 15mn \wedge (8 < y) \wedge (y < 12)$$

ne peut pas être évaluée par une base de données standard (à cause de la variable y apparaissant dans les inégalités). Dans les exemples d'exécution de programmes présentés dans l'annexe E, avant d'évaluer cette requête, nous l'avons transformée en utilisant l'équivalence suivante :

$$|h' - y| < 15mn \wedge (8 < y) \wedge (y < 12) \longleftrightarrow (7h45 < h') \wedge (h' < 12h15)$$

8.6 Commentaires

Nous avons indiqué qu'il était important, lorsqu'on fournit des réponses satisfaisant des conditions voisines, de prendre en compte des informations concernant l'utilisateur. Nous devons également signaler qu'il est souvent intéressant de tenir compte du nombre des réponses. Par exemple, comme on l'indique dans [48], il est particulièrement intéressant de fournir des réponses voisines lorsque la requête échoue. En revanche, il est moins utile d'en fournir lorsque la requête fournit déjà un grand nombre de réponses.

Dans notre formalisme, il y a une distinction claire entre les réponses de la requête initiale et celles fournies par les requêtes transformées. Supposons que l'on introduise dans le métalangage le prédicat :

$$Nb\text{-}Réponses(q, nb)$$

où nb représente le nombre de réponses de la requête q. La sémantique de ce prédicat est semblable à celle de la fonction agrégat *Count* des SGBD relationnels.

On peut alors utiliser ce prédicat pour limiter l'application des règles de voisinage (telles que RT1 et RT2) en fonction du nombre de réponses de la requête initiale. Par exemple, ces règles de voisinage peuvent être appliquée systématiquement si $Nb\text{-}Réponse(q, 0)$ (cas où la requête échoue).

D'autre part, la transformation que nous présentons dans ce chapitre n'est utilisée que pour fournir des réponses supplémentaires. Dans certains cas, il est probablement intéressant de l'appliquer également pour restreindre l'ensemble des réponses de la requête initiale.

Par exemple, si un utilisateur pose la question :

Quels sont les vols Paris/New-York qui partent dans la matinée ?

et si les informations concernant l'utilisateur permettent de déduire que le vol Concorde AF001 n'est pas une réponse "intéressante", alors il peut être pertinent de ne pas fournir cette réponse à l'utilisateur bien qu'elle soit solution de sa requête.

On peut aussi utiliser la cardinalité pour restreindre les réponses, en particulier lorsque la requête initiale fournit déjà un grand nombre de réponses.

Nous n'avons pas étudié en détail la transformation des conditions sous cet aspect. Il s'agit là d'un problème intéressant mais qui doit être étudié avec précaution.

Chapitre 9

Transformation du sujet

9.1 Principe général et problèmes spécifiques

Dans cette partie, nous étudions la transformation qui a été appelée dans les parties précédentes "transformation du sujet" de la requête.

Avant de présenter les connaissances utilisées pour effectuer cette transformation, nous commençons par rappeler intuitivement le principe général et les problèmes spécifiques.

En fait, nous avons décomposé la transformation du sujet d'une question en plusieurs étapes succesives :

1. Dans une première phase, on utilise des règles permettant de déterminer si des entités d'un autre type que celui apparaissant dans la requête peuvent être fournies à l'utilisateur.

   Ainsi, reprenons l'exemple de la requête $qt5$ de la section 1.3 :

   $Requête(qt5)$

   $Sujet\text{-}i(qt5, "Vol(x)")$

   $Condition\text{-}i(qt5, "Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
   $Ville\text{-}d'Arrivée(x, Bruxelles) \wedge$
   $Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)")$

   Info-i (qt5, "Heure-de-Départ (x, y)")

   et considérons la règle RT9 :

RT9 : **Si** une requête concerne les Vols,
Et la partie condition concerne une Ville de départ donnée et une Ville d'arrivée donnée,
ET la distance séparant ces deux villes n'excède pas 400 kilomètres,
Alors il est pertinent de transformer la requête en remplaçant les entités de type Vol par celles de type Train.

(Nous verrons par la suite comment cette règle s'exprime dans notre formalisme)

Comme la requête $qt5$ satisfait les prémisses de la règle RT9, on va pouvoir déduire que, pour cette requête, les entités de type *Vol* peuvent être remplacées par celle de type *Train*.

2. Dans la première phase, nous avons déterminé, pour une requête q, les entités de type e_1 peuvant être remplacées par celles de type e_2; après substitution de e_1 par e_2 dans la partie sujet de la requête initiale, nous obtenons une nouvelle requête. Toutefois, la transformation ne s'arrête pas là.

   Par exemple, les attributs apparaissant dans les parties "conditions" et "informations demandées" de la requête q peuvent ne pas être définis pour les entités de type e_2. Dans ce cas, la deuxième phase consiste à transformer (lorsque c'est possible) les parties "conditions" et "informations demandées" de la requête de façon à obtenir des formules n'utilisant que des attributs définis pour les entités apparaissant dans la partie sujet transformée.

   Ainsi, prenons pour une fois un exemple financier. Soit la requête $qs1$ du chapitre 1 :

   $$\textit{Requête}(qs1)$$

   $$\textit{Sujet-i}(qs1, \text{“}\textit{Action}(x)\text{”})$$

   $$\textit{Condition-i}(qs1, \text{“}\textit{Secteur-d'Activité}(x, \textit{Exploitation-Pétrolière}) \wedge \textit{Rendement}(x, y) \wedge (y > 5\%) \wedge \textit{Variation}(x, z) \wedge (z > 0\%)\text{”})$$

   $$\textit{Info-i}(qs1, \text{“}\textit{Rendement}(x, y) \wedge \textit{Variation}(x, z)\text{”})$$

   Supposons qu'à la fin de la première phase, on ait déterminé que, pour la requête $qs1$, on pouvait remplacer les entités de types *Action* par celles de type *Obligation*. Comme l'attribut *Secteur-d'Activité* est défini pour les Actions mais pas pour les Obligations, il faut impérativement transformer la partie "condition" de $qs1$. Pour réaliser cette transformation, on utilise des règles *ad-hoc*, dépendant du domaine d'application. Dans l'exemple ci-dessus, une transformation possible consiste à supprimer l'attribut *Secteur-d'Activité* dans la condition transformée. On obtient ainsi une nouvelle requête transformée :

   $$\textit{Requête}(qs2)$$

$Sujet\text{-}i(qs2, \text{“}Obligation(x)\text{”})$

$Condition\text{-}i(qs2, \text{“}Rendement(x,y) \wedge (y > 5\%) \wedge$
$Variation(x,z) \wedge (z > 0\%)\text{”})$

$Info\text{-}i(qs2, \text{“}Rendement(x,y) \wedge Variation(x,z)\text{”})$

Le même problème nous conduit également, dans certains cas, à modifier la partie "informations demandées" de la requête initiale.

Pour résumer, le problème que nous développons dans ce chapitre consiste à définir une transformation de la requête Q en une (ou plusieurs) requêtes Q' fournissant des entités d'un type différent de celui des solutions de la requête initiale.

Cette transformation est définie à l'aide de trois sous-transformations associées respectivement aux trois parties de la requête.

9.2 Type d'entités voisines

Pour représenter la notion de voisinage entre entités, nous introduisons le méta prédicat :

$$Ent\text{-}Vois(q, E_1(x), E_2(x))$$

pour exprimer que pour la requête q, le type d'entité E_1 peut être remplacé par le type d'entité E_2.

Avec ce prédicat, il est possible de représenter les connaissances expertes relatives à un domaine d'application particulier en utilisant des règles telles que :

RT9 : $\forall\ (q, x, v_1, v_2)$
$Sujet(q, \text{“}Vol(x)\text{”}) \wedge$
$Condition(q, \text{“}Ville\text{-}de\text{-}Départ(x, v_1) \wedge\ Ville\text{-}d'Arrivée(x, v_2)\text{”}) \wedge$
$Distance(v_1, v_2) < 400km$
$\longrightarrow Ent\text{-}Vois(q, \text{“}Vol(x)\text{”}, \text{“}Train(x)\text{”})$

La traduction de cette règle en langue naturelle est la suivante :

RT9 :	**Si**	une requête concerne les Vols,
	Et	la partie condition concerne une Ville de départ donnée et une Ville d'arrivée donnée,
	ET	la distance séparant ces deux villes n'excède pas 400 kilomètres,
	Alors	il est pertinent de transformer la requête en remplaçant les entités de type Vol par celles de type Train.

Il est facile de constater dans cet exemple qu'une transformation du sujet utilisant uniquement la structure ISA des entités conduirait à des transformations non pertinentes et stupides; par exemple générer des requêtes concernant les Trains allant de Paris à New-York !

Les prémisses de ce type de règles permettent d'exprimer le contexte particulier dans lequel un type d'entité peut être considéré voisin d'un autre type d'entité.

Il faut remarquer que dans la partie antécédent de la règle RT9, certaines prémisses doivent être démontrées au niveau méta, alors que d'autres, comme : $Distance(v_1, v_2) < 400km$, doivent l'être au niveau objet (Cf. l'annexe A pour un rappel des problèmes théoriques survenant lors de l'amalgame du langage et du méta langage).

On utilise également la règle R2, indépendante d'un domaine d'application particulier, pour exprimer que la propriété de voisinage est héritée par les types d'entités plus spécifiques.

$$\begin{aligned}&\text{R2: } \forall\ (q, e_1, e_2, e'_1, x)\\ &\quad Ent\text{-}Vois(q, \text{“}e_1(x)\text{”}, \text{“}e_2(x)\text{”}) \wedge ISA(e'_1, e_1)\\ &\quad\quad \longrightarrow Ent\text{-}Vois(q, \text{“}e'_1(x)\text{”}, \text{“}e_2(x)\text{”})\end{aligned}$$

9.3 Critère d'optimisation

La notion de critère d'optimisation, que nous avons introduite dans le chapitre précédent, peut également être utilisée lors de la transformation de la partie sujet de la requête.

Pour représenter cette notion nous utilisons le même méta prédicat :

$$Opt\text{-}Criterium(q, E(x), C)$$

introduit dans la partie précédente.

Les règles Cr1, Cr2 et Cr3 (Cf. page 144), définies initialement pour transformer la partie condition de la requête, peuvent être également utilisées lors de la transformation de la partie sujet.

D'autres règles définissent un critère d'optimisation pour une transformation particulière de la partie sujet de la requête.

Par exemple, la règle Cr4 indique que si la requête initiale q concerne les entités de type *Vol*, alors des entités de type *Train* seront fournies s'ils sont deux fois moins cher que les vols solutions de la requête initiale.

$$\begin{aligned}&\text{Cr4 : } \forall\ (q, x, y, C, C', X')\\ &\quad Sujet(q, \text{“}Vol(x)\text{”}) \wedge Condition\text{-}i(q, C) \wedge Variant(C, C', X')\\ &\quad\quad \longrightarrow Opt\text{-}Criterium(q, \text{“}Train(x)\text{”},\\ &\quad\quad\quad\quad \text{“}Prix(x, y) \wedge\\ &\quad\quad\quad\quad \forall\ (x', y', X')\end{aligned}$$

$$E(x') \wedge Prix(x', y') \wedge C' \longrightarrow y < (y'/2)")$$

Dans la domaine financier, la règle Cr5 indique que pour les entités de type *Obligation-Convertible*, le *Rendement* peut être un critère d'optimisation. Ainsi, si la requête q concerne les entités de type *Action*, alors des entités de type *Obligation-Convertible* seront fournies si l'obligation convertible présente un rendement supérieur à l'action associée à la conversion.

Le prédicat $Titre\text{-}Support(x, x')$ définit une relation entre les entités de type *Obligation-Convertible* et *Action*. $Titre\text{-}Support(x, x')$ est vrai si l'obligation convertible x est convertible en l'action x'.

$$\begin{aligned}
&\text{Cr5} : \forall\ (q, x, y) \\
&\quad Sujet(q, \text{"}Action(x)\text{"}) \\
&\quad\quad \longrightarrow Opt\text{-}Criterium(q, \text{"}Obligation\text{-}Convertible(x)\text{"}, \\
&\quad\quad\quad\quad \text{"}Rendement(x, y) \wedge \\
&\quad\quad\quad\quad \exists\ (x', y') \\
&\quad\quad\quad\quad\quad Titre\text{-}Support(x, x') \wedge \\
&\quad\quad\quad\quad\quad Rendement(x', y') \wedge (y > y')
\end{aligned}$$

Les règles générales qui indiquent comment utiliser les règles Cr1–Cr5 lorqu'on transforme une requête pour fournir des entités voisines sont présentées dans la section suivante.

9.4 Transformation de la requête

Dans cette partie nous définissons une transformation de la requête posée par l'utilisateur utilisant les informations dérivées sur le prédicat *Ent-Vois*.

Les requêtes transformées sont définies à l'aide de trois méta prédicats représentant la transformation des trois parties de la requête.

Nous commençons donc par introduire dans le méta langage le prédicat :

$$Transf\text{-}Ent(q, e_1, e_2, e, e')$$

où e est la partie sujet de la requête q, e_1 est un type d'entité apparaissant dans e, e_2 est le type d'entité qui peut remplacer e_1 dans e, et e' est la nouvelle partie sujet obtenue après transformation.

Les faits dérivés sur le prédicat *Ent-Vois* sont utilisés dans la règle R3 pour déduire cette nouvelle partie sujet.

$$\begin{aligned}
&\text{R3} : \forall\ (q, e_1, e_2, e, e') \\
&\quad Sujet\text{-}i(q, e) \wedge \\
&\quad Ent\text{-}Vois(q, e_1, e_2) \wedge
\end{aligned}$$

$$\neg Sujet(q, e_2) \wedge$$
$$Substitute(e_1, e_2, e, e')$$
$$\longrightarrow Transf\text{-}Ent(q, e_1, e_2, e, e')$$

Une traduction en langue naturelle de cette règle est la suivante :

R3 : **Si** une requête q a pour partie sujet e,
Et pour cette requète le type d'entité e_1 est voisin de e_2,
Et e_2 n'apparait pas dans q,
Alors la partie sujet de la requête transformée est la formule e' obtenue en remplaçant dans e la sous-formule e_1 par e_2.

Lorsque la partie sujet d'une requête est transformée, les conditions doivent parfois être adaptées au nouveau type d'entité. Pour réaliser cette transformation, nous introduisons le méta prédicat :

$$Transf'\text{-}Cond(q, e_1, e_2, c, c')$$

où c' est la partie condition transformée lorsque le type d'entité e_1 est remplacée par e_2 dans la requête q.

Il y a plusieurs raisons justifiant cette transformation.

1. Tout d'abord les attributs qui apparaissent dans la formule c peuvent ne pas être tous définis pour le type d'entité e_2.

   Dans ce cas, une transformation possible peut consister à supprimer pûrement et simplement l'attribut dans la condition transformée. On peut donc avoir la règle suivante :

   R4 : $\forall\ (q, e_1, e_2, c, x, y)$
   $Condition(q, \text{“}c(x.y)\text{”}) \wedge Attribut(c, e_1) \wedge \neg Attribut(c, e_2)$
   $\longrightarrow Transf'\text{-}Cond(q, \text{“}e_1(x)\text{”}, \text{“}e_2(x)\text{”}, \text{“}c(x.y)\text{”}, VRAI)$

   Dans l'expression $c(x.y)$, la variable y peut s'unifier avec une liste de variables objets. Par suite, $c(x.y)$ peut s'unifier avec toute formule atomique d'arité supérieure à 1.

   En fait, pour supprimer la formule $c(x.y)$ dans la requête transformée, nous remplaçons cette formule par la formule tautologique *"VRAI"*.

   Dans certaines applications, on peut parfois établir une correspondance entre les attributs définis pour deux types d'entité différents. Par exemple, dans le domaine financier, pour le type d'entité *Compte-sur-livret*, on parlera *d'Intérêt* pour représenter les sommes perçues annuellement par le détenteur du compte; en revanche, pour les types d'entité *Action* et *Obligation*, on parlera plutôt de *Dividende*. Par suite, si pour chaque type d'entité, on introduit respectivement les prédicats $Interêt(x, y)$ et $Dividende(x, y)$, on peut alors utiliser la règle suivante :

$$\begin{aligned}
\text{RS1} : \forall\, (q, x, y) \\
&\textit{Condition}(q, \text{“}\textit{Intérêt}(x, y)\text{”}) \wedge \\
&\textit{Ent-Vois}(q, \text{“}\textit{Compte-sur-Livret}(x)\text{”}, \text{“}\textit{Obligation}(x)\text{”}) \\
&\quad \longrightarrow \textit{Transf'-Cond}(q, \text{“}\textit{Compte-sur-Livret}(x)\text{”}, \\
&\qquad\qquad\qquad\qquad \text{“}\textit{Obligation}(x)\text{”}, \\
&\qquad\qquad\qquad\qquad \text{“}\textit{Intérêt}(x, y)\text{”}, \\
&\qquad\qquad\qquad\qquad \text{“}\textit{Dividende}(x, y)\text{”})
\end{aligned}$$

Cette règle indique que si le type d'entité *Compte-sur-Livret* peut être remplacé par le type *Obligation*, alors une condition sur les *Intérêts* peut être remplacée par la même condition sur le *Dividende*.

2. Une deuxième raison de transformer les conditions apparait lorsque par la nature du type d'entité e_2 les conditions doivent être modifiées. Par exemple, si le type *Vol* peut être remplacé par *Train*, la condition sur *l'heure de départ* pourrait être remplacée par la même condition sur *l'heure d'arrivée*, car la durée du voyage est sensiblement plus longue en train. On peut donc utiliser la règle RT1 :

$$\begin{aligned}
\text{RT1:} \forall\, (q, x, y) \\
&\textit{Condition}(q, \text{“}\textit{Heure-de-Départ}(x, y)\text{”}) \wedge \\
&\textit{Ent-Vois}(q, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”}) \\
&\quad \longrightarrow \textit{Transf'-Cond}(q, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”}, \\
&\qquad\qquad\qquad\qquad \text{“}\textit{Heure-de-Départ}(x, y)\text{”}, \\
&\qquad\qquad\qquad\qquad \text{“}\textit{Heure-d'Arrivée}(x, y)\text{”})
\end{aligned}$$

3. Enfin, il faut transformer la partie condition de la requête lorsqu'on utilise un critère d'optimisation. Ceci est exprimé par la règle R5 :

$$\begin{aligned}
\text{R5} : \forall\, (q, e_1, e_2, c, C, C') \\
&\textit{Ent-Vois}(q, e_1, e_2) \wedge \\
&\textit{Condition-i}(q, c) \wedge \\
&\textit{Opt-Criterium}(q, e_2, C) \wedge \\
&\textit{Rename}(C, C') \\
&\quad \longrightarrow \textit{Transf'-Cond}(q, e_1, e_2, c, \text{“}c \wedge C\text{”})
\end{aligned}$$

Cette règle indique que le critère d'optimisation C doit être ajouté à la partie condition de la requête transformée.

Comme nous l'avons déjà remarqué dans la chapitre précédent, le prédicat *Rename* renomme les variables libres de la formule C pour éviter d'introduire des liens artificiels entre les variables.

Nous introduisons également le méta prédicat :

$$\textit{Appliquer-Transf'-Cond}(q, e_1, e_2, c, c')$$

dont le rôle est d'appliquer récursivement à la partie condition initiale de la requête q toutes les transformations définies par le prédicat *Transf'-Cond* (lorsque le type d'entité e_1 peut être remplacé par e_2); c' est la nouvelle partie condition obtenue après transformation.

Les règles définissant le prédicat *Appliquer-Transf'-Cond* sont les suivantes :

R6 : $\forall\ (q, f, e_1, e_2, c_1, c_2, c, c')$
$Transf'\text{-}Cond(q, e_1, e_2, c_1, c_2) \wedge$
$\neg Sous\text{-}formule(c_2, c) \wedge$
$Substitute(c_1, c_2, c, c') \wedge / \wedge$
$Appliquer\text{-}Transf'\text{-}Cond(q, e_1, e_2, c', f)$
$\longrightarrow Appliquer\text{-}Transf'\text{-}Cond(q, e_1, e_2, c, f)$

R7 : $\forall\ (q, e_1, e_2, f)$
$Requête(q) \wedge Formule(f) \longrightarrow Appliquer\text{-}Transf'\text{-}Cond'(q, e_1, e_2, f, f)$

On peut remarquer l'analogie entre ces deux règles et les règles définissant les prédicats *List-Att-Rel* et *List-Att-Int*.

D'autre part, comme la partie condition de la requête peut être modifiée lors de la transformation du sujet, il faut également modifier la partie informations demandées (pour que l'utilisateur sache comment les conditions ont été transformées).

Pour réaliser cette transformation, nous introduisons le méta prédicat :

$$Transf\text{-}Info(q, e_1, e_2, f, f')$$

où f' est la partie informations demandées transformée lorsque le type d'entité e_1 est remplacé par e_2 dans la requête q.

De la même façon que pour la transformation de la partie condition, on est amené à transformer la partie informations demandées de la requête lorsque un attribut apparaissant dans cette partie de la requête est défini pour le type d'entité e_1 mais pas pour e_2. Par suite, on peut utiliser les règles R8 et RS2 obtenues en remplaçant dans R4 et RS1 le prédicat *Transf'-Cond* par *Transf-Ent*.

On utilise également la règle indépendante du domaine :

R9 : $\forall\ (q, e, e_1, e_2, c_1, c_2)$
$Ent\text{-}Vois(q, e_1, e_2) \wedge$
$Transf'\text{-}Cond(q, e_1, e_2, c_1, c_2)$
$Info\text{-}Int(c_2, I)$
$\longrightarrow Transf\text{-}Info(q, e_1, e_2, True, I)$

La règle R9 définit la valeur des attributs à fournir lorsque la partie condition est modifiée.

En particulier, lorsqu'un critère d'optimisation est inséré dans la requête, cette règle permet de fournir la valeur de l'attribut optimisé (qui est une information comparative intéressante pour l'utilisateur).

Cette règle permet également de fournir la valeur de *l'heure d'arrivée* lorsque la règle RT1 est utilisée.

Nous introduisons également le méta prédicat :

$$Appliquer\text{-}Transf\text{-}Info(q, e_1, e_2, f, f')$$

dont le rôle est d'appliquer récursivement à la partie informations demandées de la requête q toutes les transformations définies par le prédicat $Transf\text{-}Info$ (lorsque le type d'entité e_1 peut être remplacé par e_2); f' est la nouvelle partie informations demandées obtenue après transformation.

Les règles définissant le prédicat *Appliquer-Transf-Info* sont celles obtenues en remplaçant respectivement, dans les règles R6 et R7, les prédicats $Transf'\text{-}Cond$ et $Appliquer\text{-}Transf'\text{-}Cond$ par $Transf\text{-}Info$ et $Appliquer\text{-}Transf\text{-}Info$.

Enfin, la règle R10 indique que des entités voisines autres que celles explicitement demandées dans la requête q, sont obtenues en appliquant respectivement aux parties sujet, condition et informations demandées de la requête les transformations définies par les prédicats $Transf\text{-}Ent$, $Appliquer\text{-}Transf'\text{-}Cond$ et $Appliquer\text{-}Transf\text{-}Info$. On obtient ainsi une (ou plusieurs) requêtes q' pour lesquelles on fournit une réponse coopérative à l'utilisateur.

$$\begin{aligned}
\text{R10} : \forall\ & (q, e, c, f, q', e', c', f', e_1, e_2) \\
& Sujet\text{-}i(q, e) \\
& Transf\text{-}Ent(q, e_1, e_2, e, e') \wedge \\
& Condition\text{-}i(q, c) \wedge \\
& Appliquer\text{-}Transf'\text{-}Cond(q, e_1, e_2, c, c') \wedge \\
& Info\text{-}i(q, f) \wedge \\
& Appliquer\text{-}Transf\text{-}Info(q, e_1, e_2, f, f') \wedge \\
& q' = \text{“}e' \wedge c' \wedge f'\text{”} \wedge \\
& Solution\text{-}Coop(q', res) \\
& \quad \longrightarrow Sujet\text{-}Vois(q, res)
\end{aligned}$$

9.5 Exemple de transformation de requête

Dans cette section, nous montrons comment un processus de dérivation appliqué à la requête $qt5$ nous permet d'obtenir une requête transformée fournissant des entités voisines.

Pour simplifier les expressions, nous commençons par poser :

$$c = Ville\text{-}de\text{-}Départ(x, Paris) \wedge Ville\text{-}d'Arrivée(x, Bruxelles)$$

$$C' = \begin{array}{l} \textit{Heure-de-Départ}(x,y) \wedge (8 < y) \wedge (y < 12) \\ \textit{Ville-de-Départ}(x', \textit{Paris}) \wedge \textit{Ville-d'Arrivée}(x', \textit{Bruxelles}) \wedge \\ \textit{Heure-de-Départ}(x', y'') \wedge (8 < y'') \wedge (y'' < 12) \end{array}$$

De : $\textit{Sujet-i}(qt5, \text{“}\textit{Vol}(x)\text{”})$
on déduit en utilisant S5 et S8 : $\textit{Sujet}(qt5, \text{“}\textit{Vol}(x)\text{”})$.

De : $\textit{Condition-i}(qt5, c)$
on déduit : $\textit{Condition}(qt5, \text{“}\textit{Ville-de-Départ}(x, \textit{Paris}) \wedge \textit{Ville-d'Arrivée}(x, \textit{Bruxelles})\text{”})$
et comme : $\textit{Distance}(\textit{Paris}, \textit{Bruxelles}) < 400km$
en utilisant RT9, on a :
$\textit{Ent-Vois}(qt5, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”})$

Comme : $\neg\textit{Sujet}(qt5, \text{“}\textit{Train}(x)\text{”})$
de R3, on déduit :
$\textit{Transf-Ent}(qt5, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”}, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”})$

Comme : $\textit{Condition}(qt5, \text{“}\textit{Heure-de-Départ}(x,y)\text{”})$
de RT1, on déduit :
$\textit{Transf'-Cond}(qt5, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”}, \text{“}\textit{Heure-de-Départ}(x,y)\text{”} \text{“}\textit{Heure-d'arrivée}(x,y)\text{”})$

De Cr4, on déduit :
$\textit{Opt-Criterium}(qt5, \text{“}\textit{Train}(x)\text{”}, \text{“}\textit{Prix}(x,y) \wedge \forall\,(x', y', y'') \; \textit{Vol}(x') \wedge \textit{Prix}(x', y') \wedge C' \longrightarrow y < (y'/2)\text{”})$
et en utilisant R5, (après renommage de la variable y dans $\textit{Prix}(x,y)$), on a :
$\textit{Transf'-Cond}(qt5, \text{“}\textit{Train}(x)\text{”}, \text{“}\textit{Vol}(x)\text{”}, c, \text{“}c \wedge \textit{Prix}(x, y_1) \wedge \forall\,(x', y', y'') \; \textit{Vol}(x') \wedge \textit{Prix}(x', y') \wedge C' \longrightarrow y_1 < (y'/2)\text{”})$

Par suite, en utilisant les règles R6 et R7, on déduit :
$\textit{Appliquer-Transf'-Cond}(qt5, \text{“}\textit{Train}(x)\text{”}, \text{“}\textit{Vol}(x)\text{”}, c, \text{“}\textit{Ville-de-Départ}(x, \textit{Paris}) \wedge \textit{Ville-d'Arrivée}(x, \textit{Bruxelles}) \wedge \textit{Heure-d'Arrivée}(x,y) \wedge (8 < y) \wedge (y < 12) \wedge \textit{Prix}(x, y_1) \wedge \forall\,(x', y', y'') \; \textit{Vol}(x') \wedge \textit{Prix}(x', y') \wedge C' \longrightarrow y_1 < (y'/2)\text{”})$

En utilisant R9, on a :
$\textit{Transf-Info}(qt5, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”}, \textit{True}, \text{“}\textit{Heure-d'Arrivée}(x,y)\text{”})$
$\textit{Transf-Info}(qt5, \text{“}\textit{Vol}(x)\text{”}, \text{“}\textit{Train}(x)\text{”}, \textit{True}, \text{“}\textit{Prix}(x, y_1)\text{”})$

et par suite, après renommage de la variable y dans $Heure\text{-}d'Arrivée(x, y)$, on obtient :

$Appliquer\text{-}Transf\text{-}Info(qt5, "Vol(x)", "Train(x)",$
$"Heure\text{-}de\text{-}Départ(x, y)"$
$"Heure\text{-}de\text{-}Départ(x, y) \wedge$
$Heure\text{-}d'Arrivée(x, y_2) \wedge$
$Prix(x, y_1)")$

ce qui permet, en utilisant la règle R10, de générer la nouvelle requête $qt6$:

$Requête(qt6)$

$Sujet\text{-}i(qt6, "Vol(x)")$

$Condition\text{-}i(qt6, "Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, Bruxelles) \wedge$
$Heure\text{-}d'Arrivée(x, y_2) \wedge (8 < y_2) \wedge (y_2 < 12) \wedge$
$Prix(x, y_1) \wedge$
$\forall (x', y', y'')$
$Vol(x') \wedge Prix(x', y') \wedge C' \longrightarrow y_1 < (y'/2)")$

$Info\text{-}i(qt6, "Heure\text{-}de\text{-}Départ(x, y) \wedge$
$Heure\text{-}d'Arrivée(x, y_2) \wedge$
$Prix(x, y_1)")$

Conclusion

Nous avons présenté une méthode permettant des accès intelligents à une base de données fondée sur la coopération entre le système et l'utilisateur. Dans cette méthode, nous avons distingué trois types d'informations supplémentaires pertinentes à fournir à l'utilisateur :

1. Pour les entités solutions de la requête, la valeur d'attributs intéressants.
2. Des entités du même type que les entités solutions et satisfaisant des conditions voisines.
3. Des entités de type voisin.

Pour déterminer ces informations pertinentes, nous proposons trois méthodes reposant sur des principes de base semblables : des règles générales sont utilisées pour représenter le savoir-faire d'une personne ayant l'habitude de fournir des renseignements à des personnes qui ne sont pas des familiers du domaine d'application. Pour déterminer les informations pertinentes, la méthode utilise également une description de haut niveau du contenu de la base de données qui est plus riche, d'un point de vue sémantique, que le schéma du modèle relationnel standard. Cette description utilise les concepts du modèle entité-relation, ainsi que la notion de *"thème"*.

La notion de *thème* est très utile pour définir les informations intéressantes. Elle constitue une approche différente et complémentaire de celle suivie par J. F. Allen [2,3] où les informations intéressantes sont définies en utilisant la notion de plan. Les plans permettent une définition plus précise des informations pertinentes que les *thèmes*, mais dans de nombreuses situations, l'utilisateur n'a pas de plan précis en tête, ou il n'y a pas de plan prédéfini dans le système correspondant à ce que veut réaliser l'utilisateur. Dans ce genre de situations, les plans ne peuvent être utilisés.

La méthode qui est présentée est formalisée en logique du premier ordre. Ce formalisme simple et bien connu possède une sémantique précise. Il a été légèrement étendu en introduisant un nouvel opérateur logique, appelé pseudo-implique, qui est utilisé pour représenter, à l'aide d'une seule formule, les requêtes "hétérogènes" transformées.

On peut remarquer que cette représentation logique nous permet de modifier facilement les règles lorsque le domaine d'application change, car il y a une distinction précise entre les règles qui dépendent du domaine d'application et celles qui n'en dépendent pas.

Ainsi, comparons notre formalisme avec celui présenté par A. Motro dans [48]. Cette approche a le même objectif que la notre dans ce que nous avons appelé "transformations des conditions". Dans son travail, les informations intéressantes sont définies en utilisant une mesure sur les valeurs d'attributs et des poids sur chaque attribut, un algorithme calculant ensuite les entités intéressantes. Cette approche ne permet pas de prendre en compte le savoir-faire de l'expert, si ce n'est dans la définition des poids associés aux attributs. De plus, on ne peut pas étendre facilement la méthode pour que le système tienne compte des informations concernant l'utilisateur et ait un comportement adapté à chacun d'eux.

Dans notre approche, nous avons montré comment il est possible d'utiliser une représentation de l'utilisateur pour fournir des réponses coopératives. Bien sûr, la représentation proposée est très simple et pourrait être beaucoup plus sophistiquée. D'autre part, la méthode développée s'applique à une requête particulière sans chercher à l'insérer dans un dialogue réel entre l'utilisateur et la machine.

Un prolongement naturel de notre travail consisterait donc à utiliser les techniques de gestion de dialogue pour tenir compte de l'évolution des croyances et des connaissances de l'utilisateur; nous avons montré dans la section 3.6, comment ces informations peuvent être utilisées pour fournir des réponses adaptées à chaque utilisateur. Un autre développement possible consisterait à incorporer une gestion de plans pour reconnaître les intentions et motivations qui ont conduit l'utilisateur à interroger le système.

Un système sachant gérer et utiliser intelligemment toutes ces informations pourrait avoir un comportement coopératif particulièrement efficace.

Pour terminer, on peut remarquer que la méthode présentée dans cette thèse est implantée dans un prototype écrit en Prolog. L'implantation en Prolog des axiomes logiques introduit quelques modifications de la sémantique, dûes en particulier à l'évaluation de la négation par l'échec. En effet, les théories que nous avons considérées ne sont pas complètes et contiennent des clauses qui ne sont pas de Horn. Toutefois, ces modifications ont peu de conséquences et n'introduisent pas d'inconsistance.

Ce prototype tourne actuellement sur un petit exemple relatif à une agence de voyage. Nous présentons dans l'annexe E, les résultats de l'exécution du programme sur plusieurs requêtes. Nous sommes en train d'étudier une application plus significative relative à la gestion de portefeuilles.

Bibliographie

[1] J. F. Allen, A. M. Frish, and D. J. Litman. ARGOT: The Rochester Dialogue System. In *Proc. National Conference on Artificial Intelligence*, Pittsburgh, 1982.

[2] J. F. Allen and D. J. Litman. *Plan, Goals and Natural Language.* Research Review, Computer Science and Engineering. University of Rochester, 1986.

[3] J. F. Allen and C. R. Perrault. Analysing intention in utterance. *Artificial Intelligence*, 15(3):143–178, 1980.

[4] J. L. Austin. *How to do things with words ?* Oxford University Press, New-York, 1962.

[5] K. Bowen and R. Kowalski. Amalgamating language and metalanguage in Logic Programming. In Clark, editor, *Logic Programming*, Tårnlund, 1980.

[6] M. Brodie. On modelling behavioural semantics of Data Bases. In *Proc. VLDB*, 1981.

[7] M. Brodie. On the development of data models. In *On conceptual modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer Verlag, 1983.

[8] R. Burton. Diagnosis bugs in a simple procedural skill. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 157–183, Academic Press, New York, 1982.

[9] S. Carbery. Tracking User Goals in an Information-Seeking Environment. In *AAAI*, Washington D. C., 1983.

[10] S. Carbery. User models: the problem of disparity. 1986.

[11] S. K. Card, T. P Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the Association for Computing Machinery*, 23:396–410, 1980.

[12] J. M. Carroll and J. McKendree. Interface Design Issues For Advice-Giving Expert Systems. *Communication of the Association for Computing Machinery*, 30(1), 1987.

[13] P. P. Chen. The Entity / Relationship Model: toward a unified view of data. *ACM TODS*, 1(1), March 1976.

[14] L. Cholvy. *Structuration et intégrité des informations dans les bases de donnés en C.A.O : définition d'un modèle de données et réalisation d'une maquette.* Thèse de Docteur Ingénieur, ENSAE, 1983.

[15] E. F. Codd. Relational Completeness of Data Base Sublanguages. In R. Rustin, editor, *Data Base Systems, Courant Computer Science Symposium 6*, pages 65–98, Prentice-Hall, 1972.

[16] P. R. Cohen. *On knowing what to say: Planning speech acts.* Technical Report 118, Departement of Computer Science, University of Toronto, 1978.

[17] P. R. Cohen and H. J. Levesque. Persistence, Intention, and Commitment. In A. L. Lansky M. P. Georgeff, editor, *Reasoning about actions and plans*, pages 297–340, Timberline, USA, 1986.

[18] R. N. Cuff. On casuel users. *International Journal of Man-Machine Studies*, 12:163–187, 1980.

[19] F. Cuppens. *Vérification du maintien de la cohérence des inter-liaisons entre objets-complexes en CAO.* Rapport de DEA, ENSEEIHT, 1985.

[20] F. Cuppens and R. Demolombe. A PROLOG-Relational DBMS interface using delayed evaluation. In *Third International Conference on Data and Knowledge Bases*, Jerusalem, Israel, 1988.

[21] F. Cuppens and R. Demolombe. Comment reconnaître les centres d'intérêt pour fournir des réponses coopératives. In *Quatrièmes Journées Bases de Données Avancées*, Benodet, France, 1988.

[22] F. Cuppens and R. Demolombe. Cooperative Answering: a methodology to provide intelligent access to Databases. In *Second International Conference on Expert Database Systems*, Tysons Corner, Virginia, 1988.

[23] F. Cuppens and R. Demolombe. *Extending answers to neighbour entities in a Cooperative Answering context.* Technical Report, ONERA-CERT, Deri, 1988.

[24] F. Cuppens and R. Demolombe. *How to recognize topics to provide Cooperative Answering.* Technical Report, ONERA-CERT, Deri, 1988.

[25] S. De. Providing Effective Decision Support: Modeling users and their requirements. *Decision Support Systems*, 2:309–319, 1986.

[26] R. Demolombe. *Syntactical Characterization of a Subset of Domain Independent Formulas.* Technical Report, ONERA-CERT, 1982.

[27] D. Dubois and H. Prade. *Théorie des Possibilités: Applications à la représentation des connaissances en informatique.* Masson, 1985.

[28] B. Di Eugenio. Cooperative Behavior in the FIDO System. *Information Systems*, 12(3):295–316, 1987.

[29] R. Fagin and J. Halpern. Belief, awareness and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.

[30] 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.

[31] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufman, 1987.

[32] I. P. Goldstein. The genetic graph: a representation for the evoluation of procedural knowledge. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 51–77, Academic Press, New York, 1982.

[33] H. P. Grice. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and Semantics*, Academic Press, New York, 1975.

[34] H. P. Grice. Meaning. *Philosophical Review*, 56:377–388, 1957.

[35] H. P. Grice. Utterer's Meaning and Intentions. *Philosophical Review*, 68:147–177, 1969.

[36] B. Grosz. The Representation and Use of Focus in a System for Understandings Dialogs. In *Proc. 5th Int. Joint Conf. on Artificial Intelligence*, 1977.

[37] M. Guyomard and J. Siroux. Suggestive and Corrective Answers: a Single Mechanism. In *Pre-acts Structure of Multimodal Dialogues Including Voice*, Venaco, France, 1986.

[38] J. Y. Hintikka. *Knowledge and belief.* Cornell University Press, Ithaca, New-York, 1963.

[39] J. M. Janas. How to not say NIL - Improving Answers to failing queries in Data Base Systems. In *Proc. 6th Int. Joint Conf. on Artificial Intelligence*, pages 429–434, Tokyo, 1979.

[40] J. M. Janas. On the feasability of informative answers. In *Journées d'études "Bases formelles pour les bases de données"*, CERT-DERI, 1979.

[41] A. K. Joshi, B. L. Webber, and R. M. Weishedel. Some aspects of default reasonning in interactive discourse. In R. G. Reilly, editor, *Communication Failure in Dialogue and Discourse*, North-Holland, 1987.

[42] S. J. Kaplan. *COOPerative Responses from a Portable Natural Language Data Query System.* PhD thesis, University of Pennsylvania, 1979.

[43] S. J. Kaplan. Cooperative Responses from a Portable Natural Language Query System. *Artificial Intelligence*, 19:165–187, 1982.

[44] R. Kowalski. The limitations of Logic. In J. Schmidt and C. Thanos, editors, *Proc. Workshop on Foundations of Knowledge Base Management*, Crete, 1986.

[45] H. Ledgarg, J. A. Whiteside, A. Singer, and W. Seymour. The natural language of interactive systems. *Communications of the Association for Computing Machinery*, 23:556–563, 1980.

[46] R. C. Moore. *Reasonning about Knowledge and Action.* PhD thesis, Massachussets Institute of Technology, 1980.

[47] A. Motro. Query generalization: a technique for handling query failure. In *Proc. First International Workshop on Expert Database Systems*, pages 314–325, Kiawah Island, South Carolina, 1984.

[48] A. Motro. Supporting goal queries in Relational Databases. In *Proc. First International Conf. on Expert Database Systems*, 1986.

[49] J. F. Nicaud. *APLUSIX : un expert en résolution pédagogique d'exercices d'algèbre.* Thèse d'état, LRI, Université de Paris XI, 1987.

[50] C. Parent and S. Spaccapietra. Un modèle et une algèbre pour les bases de données de type Entité-Relation. *Technique et science informatiques*, 6(5):435–455, 1987.

[51] D. Perlis. Languages with Self-References II: Knowledge, Belief, and Modality. *Artificial Intelligence*, 34:179–212, 1988.

[52] R. Reiter. A logic for default reasonning. *Artificial Intelligence*, 13:81–132, 1980.

[53] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer Verlag, 1983.

[54] E. A. Rich. User modeling via stereotypes. *Cognitive Science*, 3:329–354, 1979.

[55] E. A. Rich. Users are individuals: individualizing user models. *International Journal of Man-Machine Studies*, 18:199–214, 1983.

[56] J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12:23–41, 1965.

[57] J. R. Searle. *Speech Acts: An essay in the philosophy of language.* Cambridge University Press, New-York, 1969.

[58] J. Self. Student models: what use are they? In R. Lewis and P. Ercoli, editors, *Proc. IFIP_TC.3 Frascati*, North Holland, 1987.

[59] C. L. Sidner and D. J. Israel. Recognizing Intented Meaning and Speakers' Plans. In *Proc. 7th Int. Joint Conf. on Artificial Intelligence*, Vancouver, 1981.

[60] L. Siklossy. Impertinent question-answering: justifications and theory. In *Proc. ACM National Conf.*, pages 39–44, 1978.

[61] D. Sleeman. Assessing aspects of competence in basic algebra. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 185–199, Academic Press, New York, 1982.

[62] W. Wahlster, H. Marburger, A. Jameson, and S. Busemann. Over-answering yes-no questions: extended responses in a natural language interface to a vision system. In *Proc. of 8th Int. Joint. Conf. on Artificial Intelligence*, pages 643–646, Karlsruhe, Germany, 1983.

[63] D. H. Warren. *WARPLAN: A system for generating plans.* Memo 76, Departement of Computational Logic, University of Edinburgh, June 1974.

[64] R. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.

Partie IV

Annexes

Annexe A

Rappels théoriques : séparation de la connaissance en plusieurs niveaux

A.1 Préliminaires

L'approche suivie par A. Bowen et R. Kowalski dans [5] consiste à construire un système qui amalgame un niveau logique objet avec une portion d'un méta-langage utile pour formaliser le processus de déduction dans le langage objet. L'utilisation du méta-langage est semblable à celle de Weyhrauch [64], la différence principale étant que ce dernier ne prend pas en compte des systèmes qui amalgament complètement les niveaux objet et méta. Dans [19], nous étudions également le problème de la représentation et de la manipulation d'une preuve dans un méta langage.

Dans ce qui suit, A, B, C et D sont des méta-variables représentant des formules du langage objet, et s et t sont des méta-variables représentant les autres termes du langage objet. Les autres identificateurs minuscules (tels que x, y, *pred*, *etc...*) représentent les variables du langage objet, alors que les identificateurs majuscules représentent les constantes.

Etant donné un langage objet L et des formules A_1, .., A_n et B de L, on représentera le fait que B peut se déduire de A_1, .., A_n en utilisant une seule règle d'inférence en écrivant :

$$\frac{A_1, \ldots, A_n}{B}$$

Si T est une théorie de L et B peut être prouvé dans T alors on écrira :

$$T \vdash_L B$$

Dans le cas où la précision de L est inutile on écrira pour simplifier $T \vdash B$.

Un système d'inférence consiste en la donnée d'un langage du premier ordre avec les axiomes logiques et les règles d'inférence. Le système est complet si toute formule logiquement valide possède une preuve dans une théorie vide (celle qui n'a pas d'axiomes propres).

Un littéral est soit une formule atomique (on parlera de littéral positif) ou la négation d'une formule atomique (on parlera de littéral négatif).

Dans [5] on s'intéresse à l'amalgame d'un langage objet et d'un méta-langage qui sont deux logiques de Horn. Commençons par rappeler la notion de clause de Horn.

Une clause de Horn est une formule fermée universellement quantifiée de la forme :

$$\forall(x_1, x_2, ..., x_n)[B_0 \vee B_1 \vee ... \vee B_k]$$

où les B_i sont des littéraux parmi lesquels au plus un littéral est positif. Si B_0 est le seul littéral positif, on écrira la clause de Horn ci-dessus sous la forme suivante :

$$B_0 \longleftarrow B_1 \wedge B_2 \wedge ... \wedge B_k$$

Une clause de Horn n'ayant pas de littéral positif est un but et est écrit de la façon suivante :

$$\longleftarrow B_1 \wedge B_2 \wedge ... \wedge B_k$$

Une clause de Horn ayant un seul littéral positif et n'ayant pas de littéral négatif sera appelé un fait.

A.2 Le problème de la représentation

L'amalgame d'un langage objet et d'un méta-langage est un cas particulier de la représentation d'une relation par un ensemble de règles. La représentation de la relation *append* pour des listes par le symbole de prédicat App est donnée par l'ensemble de règles suivant :

- $A1 : App(Nil, y, y) \longleftarrow$
- $A2 : App(u.x, y, u.z) \longleftarrow App(x, y, z)$

est un exemple simple de cette notion générale. Ici, on suppose que Nil code la liste vide et $x.y$ code le résultat de l'application de l'objet codé x sur le tête de la liste codée y : alors $A1$-$A2$ représente la relation *append* dans le sens suivant :

Pour toutes listes x, y et z
z est le résultat de append de x sur y
si et seulement si
$A1\text{-}A2 \vdash App(x', y', z')$

où x' code x, y' code y et z' code z.

Un symbole de prédicat P dans le contexte d'un ensemble de règles S représente une relation R si et seulement si il y a un codage qui associe les individus i du domaine de R avec des termes i' tel que pour tout $i_1, i_2, ..., i_n$ du domaine de R, on ait :

$$(i_1, i_2, ..., i_n) \in R \text{ ssi } S \vdash P(i'_1, i'_2, i'_n)$$

La représentabilité n'impose pas que $\neg P$ dans S représente le complément de R c'est-à-dire :

Pour tout $i_1, i_2, ..., i_n$ du domaine de R,
$(i_1, i_2, ..., i_n)$ n'appartient pas à R
ssi
$S \vdash \neg P(i'_1, i'_2, ..., i'_n)$

En fait, il est clair que R est décidable (il existe une procédure de décision pour l'appartenance dans R) si et seulement si, pour un prédicat P et une théorie S, P dans S représente R et $\neg P$ dans S représente le complément de R.

Supposons maintenant que R soit la relation de déduction $\vdash_L$ du langage L. Pour représenter $\vdash_L$ dans un autre langage M il est nécessaire de coder les formules, les ensembles de formules et toute autre expression de L par un terme de M. Par exemple, on peut coder la formule atomique suivante :

$$P(x, Bill)$$

de L par le terme fonctionnel :

$$atom(pred(1), var(1).constante(212).Nil)$$

de M, où *atom, pred, var* et *constante* sont des symboles de fonction et 1, 212 et *Nil* sont des constantes. On suppose qu'un tel codage est utilisé et pour simplifier on écrit

$$\text{“}P(x, Bill)\text{”}$$

pour représenter un terme de M qui code $P(x, Bill)$. En général, si A est une expression ou un ensemble d'expressions de L, “A” représentera un terme de M qui code A.

Soit *Demo* un symbole de prédicat binaire de M. Alors dans le contexte d'un ensemble Pr de formule de M, *Demo* représente $\vdash_L$ si et seulement si :

Pour tout ensemble fini de formules de A et
Pour toute formule B de L,
$A \vdash_L B$ ssi $Pr \vdash_M Demo(A', B')$

Remarquons que cette représentation n'impose pas que $\neg Demo$ dans Pr représentente l'improuvabilité. En effet, l'indécidabilité de la logique du premier ordre implique qu'aucune représentation de la prouvabilité ne représente également l'improuvabilité.

A.3 Représentation du processus de déduction

Les clauses D1-D2 ci-dessous constituent le top-level du processus de déduction pour un ensemble de clauses de Horn :

$$\begin{aligned}
&\text{D1} : Demo(prog, goals) \longleftarrow Empty(goals)\\
&\text{D2} : Demo(prog, goals) \longleftarrow Select(goals, goal, rest) \wedge\\
&\qquad\qquad Member(proc, prog) \wedge\\
&\qquad\qquad Rename(proc, goals, variantproc) \wedge\\
&\qquad\qquad Parts(variantproc, concl, conds) \wedge\\
&\qquad\qquad Match(concl, goal, sub) \wedge\\
&\qquad\qquad Apply(conds + rest, sub, newgoals) \wedge\\
&\qquad\qquad Demo(prog, newgoals)
\end{aligned}$$

D1 indique qu'une démonstration est terminée lorsqu'il n'y a plus de but à démontrer.

D2 indique qu'on démontre un ensemble non-vide de buts si l'on démontre un ensemble de nouveaux buts obtenus en :

1. Choisissant un but,
2. Trouvant une règle R dans le programme,
3. Renommant les variables de R pour qu'elles soient distinctes des variables des buts,
4. Unifiant la conclusion de R avec le but choisi,
5. Appliquant la substitution après concaténation des conditions de R aux buts restants.
6. Résolvant les buts restants.

Remarquons que si le méta-langage M inclut la négation et Pr est une représentation de la relation de dérivation du langage L, alors on ne peut jamais avoir :

$$Pr \vdash \neg Demo(A', B')$$

En effet les clauses de Horn ne peuvent avoir de conclusions négatives.

Toutefois, les problèmes d'indécidabilité commencent lorsqu'on exprime les axiomes D1-D2 sous forme d'équivalence, c'est-à-dire :

$$\begin{aligned} &Demo(prog, goals) \longleftrightarrow \\ &\quad Empty(goals) \text{ ou} \\ &\quad (Select(goals, goal, rest) \wedge ... \wedge Demo(prog, new\text{-}goals)) \end{aligned}$$

Supposons que *Demo*, non seulement représente $\vdash_L$, mais aussi simule le comportement du prédicat preuve de L au moins dans le cas où $A \vdash_L B$ échoue en un temps fini ssi $Pr \vdash_M Demo(A', B')$ échoue en un temps fini. Dans ce cas si $Pr*$ consiste en la représentation de Pr sous forme de clauses de Horn exprimées sous forme d'équivalence avec les axiomes de l'égalité appropriés alors :

$Pr* \vdash_M \neg Demo(A', B')$ ssi
$A \vdash_L B$ se termine par un échec.

Toutefois, la preuve de $A \vdash_L B$ termine dans les seuls cas où ni $Pr* \vdash Demo(A', B')$ ni $Pr* \vdash \neg Demo(A', B')$ ne sont démontrables.

Pour amalgamer complètement le langage objet et le méta-langage il est nécessaire d'établir une communication entre les deux niveaux. Ceci est réalisé en ajoutant des règles de *liens* :

$$(1) \quad \frac{\text{Pr} \vdash_M \text{Demo (A',B')}}{\text{A} \vdash_L \text{B}}$$

$$(2) \quad \frac{\text{A} \vdash_L \text{B}}{\text{Pr} \vdash_M \text{Demo (A',B')}}$$

L'amalgame des langages L et M est donc constitué par la donnée des langages L et M ainsi que :

1. Une relation de codage qui associe chaque expression de L à au moins un terme sans variable de M (Une expression de L peut avoir plusieurs codes associés dans M, mais chaque expression codée dans M est associée à une seule expression de L)
2. Une représentation de $\vdash_L$ par un symbole de prédicat *Demo* dans le contexte d'une théorie Pr de M et,
3. Les règles (1) et (2).

L'amalgame de L et M est une extension conservative dans le sens qu'aucun théorème n'est démontrable dans l'amalgame s'il n'est pas déjà démontrable dans L ou M. En

fait, chaque preuve dans l'amalgame peut être transformée en une preuve normale dans L ou M en éliminant les règles de liens.

De plus, il n'est pas nécessaire de se restreindre à une logique de Horn pour M. Remarquons enfin, que l'amalgame autorise le cas $L = M$, où les deux langages sont identiques.

Annexe B

Preuve de complétude et de validité

B.1 Première preuve

Complétude : Considérons un ensemble d'axiomes *Obj-ax* au niveau objet de la forme : $(T(x) \longrightarrow T'(x))$ ou bien $\neg\exists(x)(T(x) \wedge T'(x))$, et un ensemble d'axiomes correspondant *Meta-ax*, au niveau méta, de la forme : $ISA(T, T')$ ou bien $DIS(T, T')$. De plus, soit F un ensemble de faits *Type-entité*$(T_1) \ldots$ *Type-entité*(T_n) où $T_1 \ldots T_n$ est l'ensemble des prédicats qui apparaissent dans *Obj-ax*. Alors, à chaque théorème, au niveau objet, ayant la même forme et dérivable de *Obj-ax*, correspond un théorème au niveau méta dérivable de *Meta-ax*, S et F.

Preuve : Supposons que *Obj-ax* $\vdash \forall(x)(T_1(x) \longrightarrow T_2(x))$.

Il y a alors deux possibilités.

1. La clause $\neg T_1(x) \vee T_2(x)$ est une tautologie. Ceci implique $T_1 = T_2$.
   Dans ce cas, il y a une dérivation correspondante dans $\{Meta\text{-}ax, S, F\}$.
   En effet, nous avons *Type-entité*(T_1) et de R1, on déduit que $ISA(T_1, T_1)$. □
2. La clause $\neg T_1(x) \vee T_2(x)$ est dérivable en utilisant le principe de résolution sur *Obj-ax*. Ceci n'est pas toujours vrai car le principe de résolution n'est pas complet en dérivation mais seulement en réfutation. Toutefois, dans le cas présent, ce résultat est vérifié à cause de la forme particulière des axiomes que nous considérons.
   Il y a deux cas où la règle de résolution peut être appliquée à deux clauses de *Obj-ax*.
   Dans le premier cas, on applique résolution à deux clauses de la forme :

$$\neg T(x) \vee T'(x) \text{ et } \neg T'(x) \vee T''(x)$$

Le résolvant est : $\neg T(x) \vee T''(x)$, qui a la même forme que les clauses de *Obj-ax*.

Dans le second cas, on applique résolution à deux clauses de la forme :

$$\neg T(x) \vee T'(x) \text{ et } \neg T'(x) \vee \neg T''(x)$$

Le résolvant est : $\neg T'(x) \vee \neg T''(x)$, qui a la même forme que les clauses de *Obj-ax*.

Pour chaque étape d'inférence dans cette théorie, il y a une étape correspondante dans $\{Meta\text{-}ax, S, F\}$.

En effet, dans le premier cas, on a :

$$ISA(T, T') \text{ et } ISA(T', T'')$$

et, de R2, on déduit : $ISA(T', T'')$.

Dans le second cas, on a :

$$ISA(T, T') \text{ et } DIS(T', T'')$$

et, de R4, on déduit : $DIS(T', T'')$.

Par suite, il y a une preuve de $ISA(T_1, T_2)$ dérivable de $\{Meta\text{-}ax, S, F\}$ qui correspond à la preuve de $\neg T_1(x) \vee T_2(x)$ dérivable de *Obj-ax*. □

Le même raisonnement s'applique aux théorèmes de la forme $\neg T_1(x) \vee \neg T_2(x)$. □

Validité : A chaque théorème de la forme $ISA(T, T')$ ou bien $DIS(T, T')$ dérivable de *Meta-ax*, S et F au niveau méta, correspond un théorème au niveau objet, de la forme : $(T(x) \longrightarrow T'(x))$ ou bien $\neg\exists(x)(T(x) \wedge T'(x))$, dérivable de *Obj-ax*.

Preuve : Supposons que $Meta\text{-}ax, S, F \vdash ISA(T_1, T_2)$.

Meta-ax et F sont deux ensembles de faits construits en utilisant les prédicats : *Type-entité*(x), $ISA(x, y)$ et $DIS(x, y)$.

Tous les faits relatifs à $ISA(x, y)$ dérivable de *Meta-ax*, S et F peuvent être obtenus en appliquant une stratégie standard en chaînage avant.

Il est alors facile de vérifier qu'à chaque application d'une règle de S correspond une étape de dérivation dans *Obj-ax*. □

Le même raisonnement s'applique aux théorèmes de la forme $DIS(T_1, T_2)$. □

B.2 Deuxième preuve

Complétude : Supposons maintenant que l'on ajoute à l'ensemble d'axiomes *Obj-ax* au niveau objet, l'ensemble des faits ayant la forme suivante : $\exists(x)T_1(x) \ldots \exists(x)T_n(x)$ où $T_1 \ldots T_n$ est l'ensemble des prédicats apparaissant dans *Obj-ax*. Alors à tout théorème, au niveau objet, de la forme : $\exists(x)T(x) \wedge \neg T'(x)$ ou bien $\exists(x)T(x) \wedge T'(x)$, dérivable de *Obj-ax*, correspond un méta-théorème de la forme $\neg ISA(T, T')$ ou bien $\neg DIS(T, T')$, dérivable de *Meta-ax*, $S' = \{R_1, \ldots, R_5\}$ et F.

Preuve : Supposons que $Obj\text{-}ax \vdash \exists(x)(T_1(x) \wedge \neg T_2(x))$.

On en déduit que les clauses $T_1(\alpha)$ et $\neg T_2(\alpha)$ (α est une constante de Skolem) sont dérivables de *Obj-ax* en utilisant le principe de résolution.

La clause $T_1(\alpha)$ appartient à l'ensemble *Obj-ax* par hypothèse.

En considérant la forme des axiomes apparaissant dans *Obj-ax*, il n'y a que deux façons de dériver, en appliquant résolution, une clause de la forme $\neg T(\alpha)$. En fait, d'après la première preuve, il existe deux autres façons d'appliquer résolution sur des clauses de *Obj-ax*, mais nous avons montré que dans les deux cas, le résolvant obtenu a la même forme que les clauses de *Obj-ax*.

La première façon de dériver $\neg T(\alpha)$ consiste à appliquer résolution à deux clauses ayant la forme suivante :

$$T(\alpha) \text{ et } \neg T(x) \vee \neg T'(x)$$

et le résolvant est $\neg T'(\alpha)$.

Dans le deuxième cas, on applique résolution à deux clauses ayant la forme suivante :

$$\neg T(\alpha) \text{ et } T(x) \vee \neg T'(x)$$

et le résolvant est encore $\neg T'(\alpha)$.

Dans ces deux cas, il y a une étape d'inférence correspondante dans $\{Meta\text{-}ax, S', F\}$.

En effet, dans le premier cas, nous avons au niveau méta :

$$DIS(T, T')$$

et en utilisant R5, on déduit $\neg ISA(T, T')$, qui est bien, au niveau méta, l'axiome correspondant aux clauses :

$$T(\alpha) \text{ et } \neg T'(\alpha).$$

Dans le deuxième cas, nous avons au niveau méta :

$$ISA(T, T')$$

et en utilisant R5, on déduit $\neg DIS(T, T')$ qui est, au niveau méta, l'axiome correspondant aux clauses :

$$\neg T(\alpha) \text{ et } \neg T'(\alpha).$$

Par suite, le premier cas montre qu'il y a une preuve de $\neg ISA(T_1, T_2)$ dérivable des axiomes $\{Meta\text{-}ax, S', F\}$ qui correspond à la preuve de $\exists(x)T_1(x) \wedge \neg T_2(x)$ dérivée en utilisant les axiomes de *Obj-ax*.

Le second cas montre qu'il y a une preuve de $\neg DIS(T_1, T_2)$ à partir des axiomes $\{Meta\text{-}ax, S', F\}$ qui correspond à la preuve de $\exists(x)T_1(x) \wedge T_2(x)$. □

Validité : A chaque méta théorème de la forme $\neg ISA(T, T')$ ou bien $\neg DIS(T, T')$ dérivable de *Meta-ax*, S' et F, correspond un théorème objet, de la forme : $\exists(x)T_1(x) \wedge \neg T_2(x)$ ou bien $\exists(x)T_1(x) \wedge T_2(x)$, dérivable de *Obj-ax*.

Preuve : De la même façon que pour la première preuve, il est facile de vérifier qu'à chaque application d'une règle dans S' correspond une étape de dérivation dans *Obj-ax*. □

Enfin, si l'on associe à un type d'entité E un attribut A et si l'on considère que chaque entité de type E a une valeur pour l'attibut A, ce qui se traduit au niveau objet par l'axiome :

$$\forall(x)E(x) \longrightarrow \exists(y)A(x, y)$$

alors, on peut établir, de la même façon que pour les preuves précédentes, des résultats de complétude et de validité en ajoutant à la méta-théorie l'axiome R6 :

$$\forall(a, t, t')Attribut(a, t) \wedge ISA(t', t) \longrightarrow Attribut(a, t')$$

Annexe C

Complétude du langage de requêtes

Dans cette partie, nous essayons de comparer la puissance d'expression de notre langage de requêtes (présenté au chapitre 6) que nous notons L avec celle du langage Alpha défini par E. F. Codd dans [15].

En fait, nous nous contentons de montrer comment transformer toute expression A du langage Alpha en un expression W de notre langage de requêtes et sémantiquement équivalente. Ceci permet de prouver que notre langage de requêtes a une puissance d'expression au moins égale à celle du langage Alpha.

Pour établir cette correspondance, nous reprenons la définition du langage Alpha, et montrons comment, à chaque notion de ce langage, on peut associer (de façon injective) une notion de notre langage.

C.1 Rappel sur le langage Alpha

C.1.1 Alphabet et formules

L'alphabet du langage Alpha est défini de la façon suivante :

- Des constantes : a_1, a_2, a_3, ...
- Des index : 1, 2, 3, 4, ...
- Des variables n-uplets : r_1, r_2, r_3, ...
- Des prédicats :
  - Unaires : P_1, P_2, P_3, ...
  - Binaires : $=$, $<$, $>$, $\leq$, $\geq$, $\neq$.

- Des symboles logiques : $\exists$, $\forall$, $\wedge$, $\vee$, $\neg$.
- Des délimiteurs.

Si T est une théorie du langage Alpha, et B peut être prouvé dans T, alors on écrira $T \vdash_{Alpha} B$ ou plus simplement $T \vdash B$.

Avec cette représentation, on peut établir une correspondance bijective entre les prédicats unaires $P_1, ...P_n$ et les relations $R_1, ..., R_n$ d'une base de données.

Ainsi, un symbole de prédicat P dans le contexte d'une théorie T représente une relation R si et seulement si il y a un codage qui associe les individus i du domaine de R à des termes i' tel que pour tout $i_1, ..., i_p$ du domaine de R, on ait :

$$(i_1, ..., i_p) \in R \text{ ssi } S \vdash P(i'_1, ..., i'_p)$$

La formule $P(r)$ (où r est une variable n-uplet) est applelée une formule domaine et s'interprète comme l'appartenance du n-uplet r à la relation R.

Un tuple indexé a la forme $r[N]$ où r est une variable tuple et N est un index. Son rôle est d'identifier la $n^{\underline{ième}}$ composante du tuple r.

Soit Θ un des prédicats $=$, $\neq$, $<$, $\leq$, $>$, $\geq$, soit λ, μ des tuples indexés et α une constante. Alors $\lambda\Theta\mu$ et $\lambda\Theta\alpha$ sont appelés des formules de jointure.

Les formules atomiques du calcul relationnel sont de deux types : les formules domaine et les formules de jointure.

Une fbf du calcul relationnel est définie de la façon suivante :

- Toute formule atomique est un fbf.
- Si f est une fbf alors $\neg f$ est une fbf.
- Si f_1 et f_2 sont des fbf, alors $(f_1 \wedge f_2)$ et $(f_1 \vee f_2)$ sont des fbf.
- Si f est une fbf et r une variable tuple, alors $\exists r(f)$ et $\forall r(f)$ sont des fbf.
- Rien d'autre n'est une fbf.

C.1.2 Séparation des domaines

Une fbf domaine est une fbf sans quantificateur dont toutes les sous-formules sont des formules domaine. Une fbf domaine sur r est une fbf domaine dont la seule variable libre est r.

Une fbf à domaine défini sur r est une fbf domaine sur r satisfaisant les deux contraintes suivantes :

- Toute négation suit immédiatement une conjonction.

- Si r apparait dans deux formules domaine (ou plus), le domaine des prédicats apparaissant dans ces formules doit être associé à des relations qui sont compatibles avec l'union.

On introduit également la notion de quantificateurs associés à un domaine. Ceux-ci sont définis par les équations :

$$\exists P(r)(f) = \exists r(P'(r) \wedge f)$$
$$\forall P(r)(f) = \forall r(\neg P'(r) \vee f)$$

Une fbf à domaine séparable est une conjonction W de la forme :

$$U_1 \wedge U_2 \wedge ... \wedge U_n \wedge V$$

avec :

1. $n \geq 1$.
2. U_1 ... U_n sont des fbf domaine sur n variables tuples distinctes.
3. Aucune formule domaine n'apparait dans V.
4. Toutes les variables libres de V appartiennent à l'ensemble dont les domaines sont spécifiés par $U_1, ..., U_n$.
5. Tous les quantificateurs apparaissant dans V sont associés à un domaine.

C.1.3 Expressions Alpha

Les expressions Alpha simples sont de la forme :

$$(t_1, t_2, ..., t_k) : W$$

où

1. W est une fbf à domaine séparable.
2. $t_1, ..., t_k$ sont des termes distincts; ce sont soit des variables tuples, soit des variables tuples indexés.
3. L'ensemble des variables tuples apparaissant dans $t_1, ..., t_k$ est précisément l'ensemble des variables libres de W

Une expression Alpha est alors définie récursivement de la façon suivante :

- Toute expression Alpha simple est une expression Alpha.
- Si $t : w_1$ et $t : w_2$ sont des expressions Alpha alors $t : (w_1 \vee w_2)$, $t : (w_1 \wedge \neg w_2)$ et $t : (w_1 \wedge w_2)$ sont aussi des expressions Alpha.
- Aucune autre expression n'est une expression Alpha.

C.2 Correspondance entre le langage Alpha et notre langage de requête L

C.2.1 Correspondance entre les alphabets et les formules

Pour établir la correspondance entre les langages, nous commençons par définir l'alphabet suivant :

- Des constantes : les constantes du langage Alpha.
- Des index : 1, 2, 3, 4, ...
- Des variables : $r'_1, r'_2, r'_3, ...$ et $x_{1,1}, x_{1,2}, ..., x_{2,1}, x_{2,2}, ..., ...$
- Des prédicats :
  - Unaires : $P'_1, P'_2, P'_3...$ (Il y a autant de prédicats P'_i dans ce langage que de prédicats unaires P_i dans la langage Alpha. Intuitivement ces prédicats correspondent aux types d'entité du modèle Entité-Relation).
  - Binaires : $=, <, >, \leq, \geq, \neq$, ainsi qu'un ensemble de prédicats de la forme $A_{i,j}$ (Si P_i est un prédicat associé à une relation n-aire R_i, alors on introduit n prédicats binaires $A_{i,1}, ..., A_{i,n}$. Intuitivement ces prédicats correspondent aux attributs des entités).
- Des symboles logiques et des délimiteurs : ceux du langage Alpha.

Soit R une relation n-aire, et P le prédicat associé dans le langage Alpha.

A cette relation n-aire $R(i_1, ..., i_n)$, on associe une relation $R'(i, i_1, ..., i_n)$ $n+1$-aire, le rôle de l'attribut i étant de numéroter chaque tuple de la relation R.

Par exemple, si l'interprétation de R est :

R (	A	B	C)
	a	1	1
	a	2	1
	b	1	2
	c	2	5

alors l'interprétation de R' sera :

R' (	I	A	B	C)
	1	a	1	1
	2	a	2	1
	3	b	1	2
	4	c	2	5

Pour une numérotation donnée, on a l'équivalence :

$$(i_1, ..., i_n) \in R \text{ ssi } (i, i_1, ..., i_n) \in R'$$

Avec cette représentation, on peut établir une correspondance entre les prédicats $P'_j, A_{j,1}, ..., A_{j,n}$ dans le contexte d'une théorie T de L et une relation $R'_j(i, i_1, ..., i_n)$ de la façon suivante :

$$(i, i_1, ..., i_n) \in R'_j \text{ ssi } T \vdash_L P'_j(i') \wedge A'_{j,1}(i', i'_1) \wedge ... \wedge A'_{j,n}(i', i'_n)$$

où $(i', i'_1, ..., i'_n)$ est un codage de $(i, i_1, ..., i_n)$.

Pour une numérotation donnée, on a alors l'equivalence :

$$S \vdash_{Alpha} P_j(i'_1, ..., i'_n) \text{ ssi } T \vdash_L P'_j(i') \wedge A'_{j,1}(i', i'_1) \wedge ... \wedge A'_{j,n}(i', i'_n)$$

La notion de fbf du langage L est définie de façon habituelle.

A une fbf à domaine séparable $W = U_1 \wedge ... \wedge U_n \wedge V$ on fait correspondre une fbf $W' = U'_1 \wedge ... \wedge U'_n \wedge V'$ de L où :

- Chaque U'_i est obtenu en remplaçant dans U_i chaque prédicat unaire P_k du langage Alpha par le prédicat P'_k de L correspondant et en remplaçant chaque variable r_l par r'_l.
- V' est obtenue à partir de V de la façon suivante :
  - Soit Θ l'un des prédicats $=, \neq, <, >, \leq, \geq$, soit $r_j[N_1]$ et $r_k[N_2]$ deux tuples indexés et α une constante.
    Soit $P_{j'}(r_j)$ et soit $P_{k'}(r_k)$ deux formules domaine apparaissant dans W (par définition d'une fbf à domaine séparable, ces formules existent).
    Alors, à chaque formule de jointure $r_j[N_1]\Theta r_k[N_2]$ apparaissant dans V', on associe la formule :
    $$A_{j',N_1}(r'_j, x_{j',N_1}) \wedge A_{k',N_2}(r'_k, x_{k',N_2}) \wedge x_{j',N_1}\Theta x_{k',N_2}$$
    et à chaque formule de jointure $r_j[N_1]\Theta\alpha$, on associe la formule :
    $$A_{j',N_1}(r'_j, x_{j',N_1}) \wedge x_{j',N_1}\Theta\alpha$$
  - Les quantificateurs associés au domaine apparaissant dans V sont transformées de la façon suivante :
    $$\exists P(r)(f) \longrightarrow \exists r'(P(r') \wedge f')$$
    $$\forall P(r)(f) \longrightarrow \forall r'(\neg P(r') \vee f')$$
    où f' est la formule obtenue en transformant f.

Soit T, la transformation qui fait correspondre W et W'.

Exemple

A la fbf du langage Alpha :

$$P_8(r_1) \wedge \neg P_6(r_1) \wedge \exists P_7(r_2)(r_1[3] = r_2[5])$$

on fait correspondre la fbf de L suivante :

$$P'_8(r'_1) \wedge \neg P'_6(r'_1) \wedge \exists r'_2(P'_7(r'_2) \wedge A_{8,3}(r'_1, x_{8,3}) \wedge A_{7,5}(r'_2, x_{7,5}) \wedge x_{8,3} = x_{7,5})$$

C.2.2 Correspondance entre les expressions Alpha et les requêtes du langage L

A une expression Alpha simple $A = (t_1, t_2, ..., t_k) : U_1 \wedge ... \wedge U_n \wedge V$, on fait correspondre une requête Q de L de la façon suivante :

- La partie sujet S de Q est la formule $U'_1 \wedge ... \wedge U'_n$ obtenue en appliquant la transformation T à la formule $U_1 \wedge ... \wedge U_n$.

  Soit $A_1 = Var(S)$ (A_1 est l'ensemble des variables libres de S)

- La partie Information Demandées I de Q est la formule $t'_1 \wedge ... \wedge t'_k$ où chaque t'_j est obtenue de la façon suivante :

  Si t_j est une variable tuple indexée $r_n[i]$ alors :

  $$t'_j = A_{n',i}(r'_n, x_{n',i})$$

  avec n' tel que $P_{n'}(r'_n)$ est une formule apparaissant dans A.

  Si t_j est une variable tuple r_n alors :

  $$t'_j = A_{n',1}(r'_n, x_{n',1}) \wedge ... \wedge A_{n',N}(r'_n, x_{n',N})$$

  avec n' tel que $P_{n'}(r'_n)$ est une formule apparaissant dans A.

  Soit $A_2 = Var(Information\ Demandée)$.

- La partie Condition C de Q est la formule V' obtenue en appliquant la transformation T à la formule V et en quantifiant existentiellement toutes les variables libres de cette formule qui n'apparaissent pas dans A_1 ou A_2.

A une expression Alpha générale A, on fait correspondre une requête Q de L de la façon suivante :

· Soit A'_1 et soit A'_2 les requêtes ayant respectivement pour sujet S_1 et S_2, condition C_1 et C_2 et information demandée I (identique pour les deux requêtes), obtenue en transformant les expressions $t : w_1$ et $t : w_2$ alors :

- si $A = t : (w_1 \vee w_2)$, alors W est la requête ayant pour sujet $S_1 \vee S_2$, condition $(S_1 \wedge C_1) \vee (S_2 \wedge C_2)$ et information demandée I.
- si $A = t : (w_1 \wedge \neg w_2)$, alors W est la requête ayant pour sujet S_1, condition $C_1 \wedge \neg(S_2 \wedge C_2)$ et information demandée I.
- si $A = t : (w_1 \wedge w_2)$, alors W est la requête ayant pour sujet $S_1 \wedge S_2$, condition $C_1 \wedge C_2$ et information demandée I.

Théorème 1 Soit A une expression quelconque du langage Alpha, et soit Q la requête obtenue après transformation de A. Alors Q est bien une requête appartenant au langage L.

Ce résultat se vérifie trivialement. Il est en particulier facile de vérifier que la requête Q est une formule indépendante du domaine, au sens défini dans [26].

Théorème 2 Soit A une expression quelconque du langage Alpha, et soit Q la requête obtenue après transformation de A. Soit $\mathcal{R}$ l'ensemble des n-uplets réponses à l'expression A et soit $\mathcal{R}'$ l'ensemble des n'-uplets de constantes $C = \{c_1, ..., c'_n\}$ tels que :

$$T \vdash_L S(\sigma(C_1/X_1)) \wedge C(\sigma(C_2/X_2)) \wedge I(\sigma(C/X))$$

où $\sigma(C/X)$ représente la substitution de X par C.

Alors : $\mathcal{R} = \Pi\mathcal{R}'$ où Π est la projection sur chacune des composantes introduites pour numéroter les tuples des relations.

Ce résultat se vérifie facilement par construction de la transformation.

Ces deux théorèmes établissent la correspondance entre une expression du langage Alpha et une requête du langage L.

Exemple

Considérons une base de données incluant les deux relations suivantes (Cf. [15]) :

Symbole	Relation	Domaine 1	Domaine 2	Domaine 3
S	Fournisseur	Fournisseur#	Nom	Addresse
T	Fournir	Fournisseur#	Piece#	

et soit la requête :

Quel est le nom et l'addresse des fournisseurs qui fournissent la pièce 15.

L'expression Alpha correspondant à cette requête est la suivante :

$$(r_1[2], r_1[3]) : P_1(r_1) \wedge \exists P_2(r_2)(r_2[2] = 15 \wedge r_2[1] = r_1[1])$$

et la requête L correspondante est la suivante :

Sujet : $Fournisseur(x)$

Condition :

$\exists(y, y_3, y_4, y_5)$

$Fournir(y) \wedge Piece\#(y, y_3) \wedge y_3 = 15 \wedge$

$Fournisseur\#(x, y_4) \wedge Fournisseur\#(y, y_5) \wedge y_4 = y_5$

Information demandée : $Nom(x, y_1) \wedge Addresse(x, y_2)$

Annexe D

Construction d'une algèbre permettant de fournir des formules comme réponses

D.1 Conditions de satisfaisabilité

Pour définir les conditions de satisfaisabilité, on utilise la notation :

$$Sat([A(X), F'], F(X))$$

qui signifie que le couple $[A(X), F']$, où $A(X)$ est un n-uplet d'assignation (x_i, a'_i) avec x_i une variable libre de F et a'_i une constante et où F' est une formule fermée sans quantificateur, satisfait la formule $F(X)$ où X $= < x_1, x_2, ..., x_n >$ sont toutes les variables libres de F.

La condition $Sat(A(X), F(X))$ est définie en fonction d'une autre condition

$$NSat(A(X), F(X))$$

avec des notations similaires.

Les conditions $Sat(A(X), F(X))$ et $NSat(A(X), F(X))$ sont définies ci-dessous.

D.1.1 Conditions de satisfaisabilité

Une interprétation est définie par $I =< D, F >$ où :

- D est un ensemble de n individus $a'_1, ..., a'_n$.

- F est une fonction d'interprétation :
  - Pour tout symbole de constante a :

    $F(a) = a'$, où a' est un individu de D.
  - Pour tout symbole de prédicat p-aire :

    $F(P) = \{< a'_1, ..., a'_p >\}$ où chaque a'_i est un élément de D.

La fonction d'assignation As est une fonction de l'ensemble des symboles de variables dans D.

Dans un modèle M, nous avons $Sat([A, F'], F)$ si et seulement les conditions suivantes sont vérifiées :

1. Si $F = P(X)$ est une formule atomique :

   $Sat([A, P(t'_1, t'_2, ..., t'_n)], P(t_1, t_2, ..., t_n))$ ssi

   $< t'_1, t'_2, ..., t'_n >$ appartient à l'interprétation de P
   et $t'_i = F(t_i)$ si t_i est un symbole de constante
   et $t'_i = As(t_i)$ si t_i est un symbole de variable
   et $A =< (t_{i_1}, As(t_{i_1})), ..., (t_{i_p}, As(t_{i_p})) >$
   où $< t_{i_1}, ..., t_{i_p} >$ représente l'ensemble des variables libres de $P(t_1, ..., t_n)$.

2. Si $F = F_1 \wedge F_2$:

   $Sat([A, F'_1 \wedge F'_2], F_1 \wedge F_2)$ ssi $Sat([A_1, F'_1], F_1)$ et $Sat([A_2, F'_2], F_2)$
   où A_1 (resp. A_2) est définie de la façon suivante :
   Soit $A =< (x_{i_1}, a_{i_1}), ..., (x_{i_n}, a_{i_n}) >$
   Alors $A_1 =< (x_{j_1}, a_{j_1}), ..., (x_{j_p}, a_{i_p}) >$
   et $A_2 =< (x_{k_1}, a_{k_1}), ..., (x_{k_q}, a_{k_q}) >$
   où $< x_{j_1}, ..., x_{j_p} >$ (resp. $< x_{k_1}, ..., x_{k_q} >$) sont toutes les variables libres de F_1 (resp. F_2), les assignations dans A_1 (resp. A_2) restant identiques à celles de A.

3. Si $F = F_1 \vee F_2$:

   $Sat([A, F'], F_1 \vee F_2)$ ssi
   Si $Sat([A_1, F'_1], F_1)$ et $Sat([A_2, F'_2], F_2)$ et $F' = F'_1 \wedge F'_2$
   Sinon $Sat([A_1, F'], F_1)$ ou $Sat([A_2, F'], F_2)$

4. Si $F = \exists x P(X, x)$

   $Sat([A, P'], \exists x P(X, x))$ ssi il existe a dans le domaine D tel que :
   $Sat([< A, (x, a) >, P'], P(X, x))$

5. Si $F = \forall x P(X, x)$

$Sat([A, P'], \forall x P(X, x))$ ssi pour tout a de D on a :

$Sat([< A, (x, a) >, P'], P(X, x))$

6. Si $F = \neg F_1$

$Sat([A, F'], \neg F_1)$ ssi $NSat([A, F'], F_1)$

D.1.2 Conditions de non satisfaisabilité

Dans un modèle M, nous avons $NSat([A, F'], F)$ si et seulement les conditions suivantes sont vérifiées :

1. Si $F = P(X)$ est une formule atomique :

$NSat([A, \neg P(t'_1, t'_2, ..., t'_n)], P(t_1, t_2, ..., t_n))$ ssi

$< t'_1, t'_2, ..., t'_n >$ n'appartient pas à l'interprétation de P
et $t'_i = F(t_i)$ si t_i est un symbole de constante
et $t'_i = As(t_i)$ si t_i est un symbole de variable
et $A =< (t_{i_1}, As(t_{i_1})), ..., (t_{i_p}, As(t_{i_p})) >$
où $< t_{i_1}, ..., t_{i_p} >$ représente l'ensemble des variables libres de $P(t_1, ..., t_n)$.

2. Si $F = F_1 \wedge F_2$:

$NSat([A, F'], F_1 \wedge F_2)$ ssi
Si $NSat([A_1, F'_1], F_1)$ et $NSat([A_2, F'_2], F_2)$ et $F' = F'_1 \wedge F'_2$
Sinon $NSat([A_1, F'], F_1)$ ou $NSat([A_2, F'], F_2)$

3. Si $F = F_1 \vee F_2$:

$NSat([A, F'_1 \wedge F'_2], F_1 \vee F_2)$ ssi $NSat([A_1, F'_1], F_1)$ et $NSat([A_2, F'_2], F_2)$

4. Si $F = \exists x P(X, x)$

$NSat([A, P'], \exists x P(X, x)$ ssi pour tout a de D on a :

$NSat([< A, (x, a) >, P'], P(X, x))$

5. Si $F = \forall x P(X, x)$

$NSat([A, P'], \forall x P(X, x))$ ssi il existe a dans le domaine D tel que :

$NSat([< A, (x, a) >, P'], P(X, x))$

6. Si $F = \neg F_1$:

$NSat([A, F'], \neg F_1)$ ssi $Sat([A, F'], F_1)$

D.2 Définition d'une algèbre relationnelle pour un modèle

Nous essayons ici de définir une algèbre permettant de remplacer une déduction dans une théorie de la forme T, par une evaluation dans un modèle M.

La définition des conditions de satisfaisabilité et de non satisfaisabilité est une étape pour atteindre cet objectif, mais il ne fournit pas une méthode constructive qui calcule tous les théorèmes de la forme $F(X)$ pour une interprétation des prédicats dans un modèle de M.

Cette méthode est donnée par les deux fonctions $Eval$ et $Neval$ définie ci-dessous.

On définit d'abord les opérateurs de l'algèbre relationnelle.

Le résultat de $Eval(F(X))$ et de $NEval(F(X))$ est considéré comme un ensemble de couple $[A, F']$ où A est un n-uplet d'assignation de variables et F une formule fermée sans quantificateur.

D.2.1 Les opérateurs de l'algèbre relationnel

Assignation As_X

Soit A un ensemble de n-uplets de constantes.

Soit $X =< x_1, x_2, ..., x_n >$ un n-uplet de symboles de variables.

Alors $As_X(A) = \{< (x_1, a_1),(x_2, a_2), ..., (x_n, a_n) > /$
$< a_1, a_2, ..., a_n >$ appartient à $A \}$

Intersection $\cap'$

Considérons $Eval(F(X)) \cap' Eval(G(Y))$

où $X =< x_1, x_2, ..., x_m >$ et $Y =< y_1, y_2, ..., y_n >$

Soit :

$$Eval(F(X)) = \{[< (x_1, a_{i_1}), (x_2, a_{i_2}), ..., (x_m, a_{i_m}) >, F_i]\}$$

$$Eval(G(Y)) = \{[< (y_1, a_{j_1}), (y_2, a_{j_2}), ..., (y_n, a_{j_n}) >, F_j]\}$$

Alors :

$Eval(F(X)) \cap' Eval(G(Y))$
$= \{[< (z_1,a_{k_1}),(z_2,a_{k_2}),...,(z_p,a_{k_p}) >, F_k] /$
z_t appartient à $X \cup Y$ et il existe i et j tels que
si $z_t = x_{t_1} = y_{t_2}$ est un élement de X et de Y
alors $a_{k,t} = a_{i,t_1} = a_{j,t_2}$
sinon
si $z_t = x_{t_1}$ est un élément de X
alors $a_{k,t} = a_{i,t_1}$
si $z_t = y_{t_2}$ est un élément de Y
alors $a_{k,t} = a_{i,t_2}$
et $F_k = F_i \wedge F_j\}$

Par exemple :

Soit $A_1 = \{[< (x,a),(y,b) >, P(a,b)], [< (x,b),(y,c) >, P(b,c)]\}$
et $A_2 = \{[< (x,a),(z,d) >, Q(a,d)], [< (x,b),(z,e) >, R(b,e)]\}$
alors $A_1 \cap' A_2 = \{[< (x,a),(y,b),(z,d) >, P(a,b) \wedge Q(a,d)]$
$[< (x,b),(y,c),(z,e) >, P(b,c) \wedge R(b,e)]\}$

Union $\cup'$

Considérons $Eval(F(X)) \cup' Eval(G(Y))$

où $X =< x_1, x_2, ..., x_m >$ et $Y =< y_1, y_2, ..., y_n >$

Soit :

Si $X = Y$ alors :

$Eval(F(X)) \cup' Eval(G(Y))$
$= \{ [A, F'] /$ Si $[A, F'_1]$ appartient à $Eval(F(X))$ et
$[A, F'_2]$ appartient à $Eval(G(Y))$
alors $F' = F'_1 \wedge F'_2$
Sinon
$[A, F']$ appartient à $Eval(F(X))$ ou
$[A, F']$ appartient à $Eval(G(Y))$ $\}$

Si $X \neq Y$ alors :

Soit $Y' = Y - (X \cap Y) =< x_{i_1}, x_{i_2}, ..., x_{i_p} >$
$X' = X - (X \cap Y) =< x_{j_1}, x_{j_2}, ..., x_{j_q} >$

La signification de $\cup'$ est :

$C_{Y'}(Eval(F(X))) \cup' C_{X'}(Eval(G(Y)))$

où $C_{Y'}$ est un opérateur similaire à celui de cylindrification de l'algèbre relationnelle.

La définition de $C_{Y'}$ est la suivante :

$C_{Y'}(Eval(F(X)))$
$= \{[< A,(x_{i_1}, a_{i_1}),(x_{i_2}, a_{i_2}),...,(x_{i_p}, a_{i_p}) >, F'] /$
$[A, F']$ appartient à $Eval(F(X))$ et
$< a_{i_1}, a_{i_2}, ..., a_{i_p} >$ appartient à D^p }

Projection p'_X

Considérons $p'_X(Eval(F(Y)))$

où $Y =< x_{i_1}, ..., x_{i_n} >$ et $X =< x_{j_1}, ..., x_{j_p} >$.

Soit $Y - X =< x_{k_1}, ..., x_{k_m} >$

$p'_X(Eval(F(Y)))$
$= \{[A, F'] / [< A,(x_{k_1}, a_{k_1}),...,(x_{k_m}, a_{k_m}) >, F']$ appartient à $Eval(F(Y)))$

Différence –

L'opérateur – est l'opérateur classique de l'algèbre relationnel réalisant la différence entre deux ensembles.

L'opérateur *Couple*

$Couple(E, F(X))$ réalise l'opération suivante :

F est une formule, X est n-uplet de symboles et E un ensemble d'assignation sur le n-uplet de variables X.

$Couple(E, F(X)) = \{[A, F(A')] / A$ appartient à E et
$F(A')$ est la formule obtenue en substituant dans F
chaque variable de X par son assignation dans A }

L'opérateur fusion F_x

$F_x(Eval(F(X, x)))$ réalise l'opération suivante :

Soit $E_1 = \{A/[A, F']$ appartient à $Eval(F(X, x))\}$
$E_2 = As_X(D^n) - p_x(As_{<X,x>}(D^{n+1}) - E_1)$
où p_x est l'opérateur standard de projection sur un ensemble d'assignation.
$E_3 = p'_x(Eval(F(X, x)))$

Alors $F_x(Eval(F(X,x)))$
$= \{[A,F'] \ / \ [A,F']$ appartient à E_3 et
A appartient à E_2 $\}$

D.2.2 Définition des fonctions $Eval$ et $NEval$

1. Si H est une formule atomique $P(X)$:
   $Eval(P(X)) = Couple(As_X(P'), P(X))$

   où $P' = \{A \ / \ A$ appartient à l'interprétation de P dans $M\}$

   $NEval(P(X)) = Couple(As_X(D^n - P'), \neg P(X))$

   où n est l'arité de P.

2. Si $H = F \wedge G$ alors
   $Eval(F \wedge G) = Eval(F) \cap' Eval(G)$
   $NEval(F \wedge G) = NEval(F) \cup' NEval(G)$

3. Si $H = F \vee G$ alors
   $Eval(F \vee G) = Eval(F) \cup' Eval(G)$
   $NEval(F \vee G) = NEval(F) \cap' NEval(G)$

4. Si $H = \neg F$ alors
   $Eval(\neg F) = NEval(F)$
   $NEval(\neg F) = Eval(F)$

5. $H = \exists x F(X,x)$ alors
   $Eval(\exists x F(X,x)) = p'_X(Eval(F(X,x)))$
   $NEval(\exists x F(X,x)) = F_x(NEval(F(X,x)))$

6. $H = \forall x F(X,x)$ alors :
   $Eval(\forall x F(X,x)) = F_x(Eval(F(X,x)))$
   $NEval(\forall x F(X,x)) = p'_X(NEval(F(X,x)))$

D.3 Extension des définitions pour prendre en compte le pseudo-implique

Dans cette partie, nous montrons comment étendre la définition de la satisfaisabilité Sat ainsi que celle de la fonction $Eval$ pour prendre en compte l'opérateur pseudo-implique $\Longrightarrow$.

Dans ces définitions, nous faisons l'hypothèse que cet opérateur n'apparait jamais sous le champ d'une négation; c'est la raison pour laquelle nous ne définissons pas la non satsifaisabilité pour cet opérateur.

Par suite, si $F = F_1 \Longrightarrow F_2$ alors :

$Sat([A, F'], F_1 \Longrightarrow F_2)$ ssi

Si $Sat([A_1, F'_1], F_1)$ et $Sat([A_2, F'_2], F_2)$ et $F' = F_2$
Sinon $Sat([A_1, F'_1], \neg F_1)$ et $F' = Vrai$.

et :

$$Eval(F_1 \Longrightarrow F_2) = (Couple(Ev(F_1), Vrai) \cap' Eval(F_2)) \cup' Couple(Ev(\neg F_2), Vrai)$$

où Ev est la fonction classique d'évaluation de l'algèbre relationnelle dont le résultat est un ensemble d'assignation de variables.

D.4 Définition de la réponse à une requête

Soit Q une requête de notre langage dont les trois parties sont :

$$Sujet \wedge Condition \wedge Info$$

La réponse à la requête Q est l'ensemble des formules fermées F telles que :

$Sat([A, Q'], Q)$ et $Sat([A_1, F'1], Sujet)$ et $Sat([A_2, F'_2], Info)$ et $F = F'_1 \wedge F'_2$

Par exemple, considerons la requête Q :

$Requête(Q)$

$Sujet\text{-}i(Q, \text{“}Vol(x) \vee Train(x)\text{”})$

$Condition\text{-}i(Q, \text{“}Ville\text{-}de\text{-}Départ(x, Paris) \wedge$
$Ville\text{-}d'Arrivée(x, Bruxelles) \wedge$
$Heure\text{-}de\text{-}Départ(x, y) \wedge (8 < y) \wedge (y < 12)\text{”})$

$Info\text{-}i(Q, \text{“}Heure\text{-}de\text{-}Départ(x, y) \wedge$
$Heure\text{-}d'Arrivée(x, y) \wedge$
$(Vol(x) \wedge Jour(y) \wedge \neg Opère(x, y))$
$\Longrightarrow \neg Opère(x, y)\text{”})$

Une traduction de cette question en langue naturelle pourrait être :

Pour les vols ou les trains qui partent de Paris entre 8 heures et 12 heures et qui arrivent à Bruxelles fournir l'heure de départ et la liste des jours pour lesquels le vol n'opère pas, s'il y en a.

Considérons tout d'abord le vol AF638 qui part de Paris à 8 heures 5, arrive à 8 heures 55 à Bruxelles et qui ne marche pas le dimanche. Pour ce vol on va fournir la réponse suivante :

$$Vol(AF638) \quad \wedge \quad \begin{array}{l} Heure\text{-}de\text{-}Départ(AF638, 08h05) \wedge \\ Heure\text{-}d'Arrivée(AF638, 08h55) \wedge \\ \neg Opère(AF638, Dimanche) \end{array}$$

Pour le vol AF642 qui part à 9 heures 35, arrive à 10 heures 25 et opère tous les jours de la semaine, on va fournir la réponse suivante :

$$Vol(AF642) \quad \wedge \quad \begin{array}{l} Heure\text{-}de\text{-}Départ(AF642, 09h35) \wedge \\ Heure\text{-}d'Arrivée(AF642, 10h25) \end{array}$$

Enfin pour le train SNCF483 qui part à 10 heures 8 et arrive à 13 heures 4, on fournit la réponse :

$$Train(SNCF483) \quad \wedge \quad \begin{array}{l} Heure\text{-}de\text{-}Départ(SNCF483, 10h08) \wedge \\ Heure\text{-}d'Arrivée(SNCF483, 13h04) \end{array}$$

Annexe E

Exemples d'exécution de programmes

Dans cette annexe, nous reprenons les exemples de transformation que nous avons étudié dans les chapitres 7, 8 et 9, et nous présentons les résultats obtenus en exécutant notre programme sur chaque requête.

Le premier exemple reprend la requête QT1 de la page 10 et lui applique le processus de transformation des informations demandées décrit au chapitre 7. Dans le deuxième exemple, on applique sur cette même requête le processus de transformations des conditions présenté au chapitre 8. Le troisième exemple reprend la requête QT5 de la page 24 et lui applique successivement les processus de transformation des informations demandées et de transformation de sujet (décrit au chapitre 9). Enfin, un quatrième exemple illustre le problème des réponses complétives.

Pour chaque exemple, nous présentons la requête initiale ainsi que la requête transformée qui permet de fournir des réponses coopératives.

Comme nous l'avons indiqué au chapitre 5, les réponses sont présentées par des formules plutôt que par des tuples.

Nous utilisons la syntaxe du calcul des prédicats : "&" représente la conjonction, "v" la disjonction, "==>" le pseudo-implique, "Not" la négation et "Ex" la quantificateur existentiel.

E.1 Exemple de transformation des informations demandées

```
=> (question)                                                    ; Question de l'utilisateur :
= Entites => Vol(*x)                                             ; Pour les vols
= Conditions => Ville-de-depart(*x Paris) &                      ; de Paris
                Ville-d'arrivee(*x New-York) &                   ; a New-York
                Heure-de-depart(*x *y) & < (08h00 *y) & < (*y 12h00)   ; partant entre 8 a.m et 12 a.m
= Informations demandees => Heure-de-depart(*x *y)               ; fournir : l'heure de depart.

Question transformee fournissant des attributs supplementaires :
-----------------------------------------------------------------
                                                                 ; Question transformee :
Entites : Vol (*x)                                               ; Pour les vols
Conditions : Ville-de-depart (*x Paris) &                        ; de Paris
             Ville-d'arrivee (*x New-York) &                     ; a New-York
             Heure-de-depart (*x *y) & < (08h00 *y) & < (*y 12h00)   ; partant entre 8 a.m et 12 a.m,
Informations demandees : Heure-de-depart (*x *y) &               ; fournir : l'heure de depart,
                         (Jour (*x1) & Not (Opere (*x *x1)))     ; les jours pour lesquels le vol n'opere pas,
                             ==> (Not (Opere (*x *x1))) &        ; s'il y en a,
                         (Cher (*x)) ==> (Prix (*x *x2)) &       ; le prix des vols chers,
                         (Not (Periode (*x 01/01 31/12)))        ; la periode de validite
                             ==> (Periode (*x *x3 *x4)) &        ; pour les vols qui n'operent pas toute l'annee,
                          Heure-d'arrivee (*x *x5)               ; et l'heure d'arrivee.
**************************

Reponses :
----------

                                                                 ; Premiere reponse : le vol AF001,
Vol (AF001) & Heure-de-depart (AF001 10h00) &                    ; qui part a 10.00 a.m,
              Heure-d'arrivee (AF001 08h45) &                    ; arrive a 8.45 a.m,
              Prix (AF001 27715FF) &                             ; coute 27.715 francs,
              Periode (AF001 01/01 27/09)                        ; et marche du 01/01 au 27/09.
                                                                 ; Seconde reponse : le vol AF001,
Vol (AF001) & Heure-de-depart (AF001 11h00) &                    ; qui part a 11.00 a.m,
              Heure-d'arrivee (AF001 09h45) &                    ; arrive a 9.45 a.m,
              Prix (AF001 27715FF) &                             ; coute 27.715 francs,
              Periode (AF001 28/09 31/12)                        ; et marche du 28/09 au 31/12.
                                                                 ; Troisieme reponse : le vol AF015,
Vol (AF015) & Heure-de-depart (AF015 10h30) &                    ; qui part a 10.30 a.m,
              Heure-d'arrivee (AF015 14h55) &                    ; arrive a 14.55 p.m,
              Periode (AF015 01/01 27/09) &                      ; marche du 01/01 au 27/09,
              Not (Opere (AF015 Mardi)) &                        ; et ne marche pas
              Not (Opere (AF015 Jeudi)) &                        ; le Mardi, le Jeudi,
              Not (Opere (AF015 Vendredi)) &                     ; le Vendredi et le Dimanche.
              Not (Opere (AF015 Dimanche))
                                                                 ; Quatrieme reponse : le vol AF015,
Vol (AF015) & Heure-de-depart (AF015 10h30) &                    ; qui part a 10.30 a.m,
              Heure-d'arrivee (AF015 14h55) &                    ; arrive a 14.55 p.m,
              Periode (AF015 28/09 31/12) &                      ; marche du 28/09 au 31/12,
              Not (Opere (AF015 Mardi)) &                        ; et ne marche pas,
              Not (Opere (AF015 Jeudi)) &                        ; le Mardi, le Jeudi et le Vendredi.
              Not (Opere (AF015 Vendredi))
```

E.2 Exemple de transformation des conditions

```
Question transformee fournissant d'autres entites interessantes :        ; Premiere question transformee qui
-------------------------------------------------------------------      ; fournit des entites voisines.
Entites : Vol (*x)
Conditions : Heure-de-depart (*x *y) & < (08h00 *y) & < (*y 12h00) &
             Ville-de-depart (*x *v1) & Ville-d'arrivee (*x *v2) &       ; Condition voisine sur
             Trajet-Voisin (*v1 *v2 Paris New-York) &                    ; les villes de depart et d'arrivee.
             Prix (*x *x5) &
             Not (Ex (*x6 *y2 *x4)
                     (Vol (*x6) & Prix (*x6 *y2) & Not (< (*x5 *y2)) &
                      Ville-de-depart (*x6 Paris) &                      ; Critere d'optimisation sur
                      Ville-d'arrivee (*x6 New-York) &                   ; le prix des vols.
                      Heure-de-depart (*x6 *x4) &
                      < (08h00 *x4) & < (*x4 12h00)))
Informations demandeess : Prix (*x *x8) & Heure-de-depart (*x *y) &      ; Fournir : Prix (info. comparative), heure de dep.
                          Ville-de-depart (*x *v1) &                     ; ville de depart
                          Ville-d'arrivee (*x *v2)                       ; et ville d'arrivee (conditions modifiees).
**************************

Question transformee fournissant d'autres entites interessantes :        ; Deuxieme question transformee.
-------------------------------------------------------------
Entites : Vol (*x)
Conditions : Ville-de-depart (*x Paris) & Ville-d'arrivee (*x New-York) &
             Heure-de-depart (*x *x11) & < (07h45 *x11) & < (*x11 12h15) & ; Condition voisine sur
             Prix (*x *x13) &                                            ; l'heure de depart.
             Not (Ex (*x14 *x15 *x12)
                     (Vol (*x14) &
                      Prix (*x14 *x15) & Not (< (*x13 *x15)) &           ; Critere d'optimisation sur
                      Ville-de-depart (*x14 Paris) &                     ; le prix des vols.
                      Ville-d'arrivee (*x14 New-York) &
                      Heure-de-depart (*x14 *x12) &
                      < (08h00 *x12) & < (*x12 12h00)))
Information demandees : Prix (*x *x17) & Heure-de-depart (*x *x11) &
                        Ville-de-depart (*x *x19) & Ville-d'arrivee (*x *x20)
**************************

Reponses :
----------
Vol (AF054) & Prix (AF054 08500FF) &                                     ; Premiere reponse : le vol AF054
              Heure-de-depart (AF054 12h10) &                            ; qui satisfait
              Ville-de-depart (AF054 Paris) &                            ; des conditions voisines
              Ville-d'arrivee (AF054 New-York)                           ; sur l'heure de depart.

Vol (AF020) & Prix (AF020 07850FF) &
              Heure-de-depart (AF020 09h30) &
              Ville-de-depart (AF020 Paris) &
              Ville-d'arrivee (AF020 Washington)

Vol (BA017) & Prix (BA017 06750FF) &                                     ; Trois autres reponses
              Heure-de-depart (BA017 09h00) &                            ; qui satisfont
              Ville-de-depart (BA017 Londres) &                          ; des conditions voisines
              Ville-d'arrivee (BA017 New-York)                           ; sur la ville de depart
                                                                         ; et sur la ville d'arrivee.
Vol (BA142) & Prix (BA142 06500FF) &
              Heure-de-depart (BA142 10h10) &
              Ville-de-depart (BA142 Londres) &
              Ville-d'arrivee (BA142 Boston)
```

E.3 Exemple de transformation du sujet

```
=> (question)                                                    ; Question de l'utilisateur :
= Entites => Vol(*x)                                             ; Pour les vols
= Conditions => Ville-de-depart(*x Paris) &                      ; de Paris
                Ville-d'arrivee(*x Bruxelles) &                  ; a Bruxelles
                Heure-de-depart(*x *y) & < (07h00 *y) & < (*y 11h00)   ; partant entre 7 a.m et 11 a.m
= Informations demandees => Heure-de-depart(*x *y)               ; fournir : l'heure de depart.

Question transformee fournissant des attributs supplementaires :
-----------------------------------------------------------------
                                                                 ; Question transformee :
Entites : Vol (*x)                                               ; Pour les vols
Conditions : Ville-de-depart (*x Paris) &                        ; de Paris
             Ville-d'arrivee (*x Bruxelles) &                    ; a Bruxelles
             Heure-de-depart (*x *y) & < (07h00 *y) & < (*y 11h00)   ; partant entre 7 a.m et 11 a.m,
Informations demandees : Heure-de-depart (*x *y) &               ; fournir : l'heure de depart,
                         (Jour (*x1) & Not (Opere (*x *x1)))     ; les jours pour lesquels le vol n'opere pas,
                             ==> (Not (Opere (*x *x1))) &        ; s'il y en a,
                         (Not (Periode (*x 01/01 31/12)))        ; la periode de validite
                             ==> (Periode (*x *x2 *x3)) &        ; pour les vols qui n'operent pas toute l'annee,
                          Prix (*x *x4) &                        ; le prix (information comparative),
                          Heure-d'arrivee (*x *x5)               ; et l'heure d'arrivee.
**************************

Reponses :
----------

                                                                 ; Premiere reponse : le vol AF638,
Vol (AF638) & Heure-de-depart (AF638 08h05) &                    ; qui part a 8.05 a.m,
              Heure-d'arrivee (AF638 08h55) &                    ; arrive a 8.55 a.m,
              Prix (AF638 1850FF) &                              ; coute 1.850 francs,
              Not (Opere (AF638 Dimanche))                       ; mais ne marche pas le dimanche.
                                                                 ; Seconde reponse : le vol AF642,
Vol (AF642) & Heure-de-depart (AF642 09h35) &                    ; qui part a 9.35 a.m,
              Heure-d'arrivee (AF642 10h25) &                    ; arrive a 10.25 a.m,
              Prix (AF642 1850FF)                                ; et coute 1.850 francs.

Question transformee fournissant d'autres entites interessantes :
-----------------------------------------------------------------
                                                                 ; Deuxieme question transformee :
Entites : Train (*x)                                             ; Pour les trains
Conditions : Ville-de-depart (*x Paris) &                        ; de Paris
             Ville-d'arrivee (*x Bruxelles) &                    ; a Bruxelles
             Heure-de-depart (*x *y) & < (07h00 *y) & < (*y 11h00)   ; partant entre 7 a.m et 11 a.m,
Informations demandees : Heure-de-depart (*x *y) &               ; fournir : l'heure de depart,
                         Prix (*x *x1) &                         ; le prix (information comparative),
                         Heure-d'arrivee (*x *x2)                ; et l'heure d'arrivee.
**************************

Reponses :
----------

                                                                 ; Premiere reponse : le train SNCF281,
Train (SNCF281) & Heure-de-depart (SNCF281 07h48) &               ; qui part a 7.48 a.m,
                  Heure-d'arrivee (SNCF281 10h41) &              ; arrive a 10.41 a.m,
                  Prix (SNCF281 250FF)                           ; et coute 250 francs.
                                                                 ; Seconde reponse : le train SNCF483,
Train (SNCF483) & Heure-de-depart (SNCF483 10h08) &              ; qui part a 10.08 a.m,
                  Heure-d'arrivee (SNCF483 13h04) &              ; arrive a 13.04 p.m,
                  Prix (SNCF483 250FF)                           ; et coute 250 francs.
```

E.4 Exemple de réponses complétives

```
=> (question)                                                    ; Question de l'utilisateur :
= Entites => Trajet(*x)                                          ; Pour les trajets
= Conditions => Ville-de-depart(*x Paris) &                      ; de Paris
                (Ville-d'arrivee(*x Bruxelles) v                 ; a Bruxelles
                 Ville-d'arrivee(*x Londres)) &                  ; ou Londres
                Heure-de-depart(*x *y) & < (07h00 *y) & < (*y 10h00)   ; partant entre 7 a.m et 10 a.m
= Informations demandees => Heure-de-depart(*x *y)               ; fournir : l'heure de depart.

Question transformee fournissant des attributs supplementaires :
-----------------------------------------------------------------
                                                                 ; Question transformee :
Entites : Trajet (*x)                                            ; Pour les trajets
Conditions : Ville-de-depart (*x Paris) &                        ; de Paris
             (Ville-d'arrivee (*x Bruxelles) v                   ; a Bruxelles
              Ville-d'arrivee (*x Londres)) &                    ; ou Londres
             Heure-de-depart (*x *y) & < (07h00 *y) & < (*y 10h00)   ; partant entre 7 a.m et 10 a.m,
Informations demandees : Heure-de-depart (*x *y) &               ; fournir : l'heure de depart,
                         (Vol (*x))
                            ==> ((Jour (*x1) & Not (Opere (*x *x1)))   ; les jours pour lesquels le vol n'opere pas,
                                    ==> (Not (Opere (*x *x1)))) &      ; s'il y en a,
                         (Vol (*x))
                            ==> ((Not (Periode (*x 01/01 31/12)))      ; la periode de validite
                                    ==> (Periode (*x *x2 *x3))) &      ; pour les vols qui n'operent pas toute l'annee,
                         (Cher (*x) ==> (Prix (*x *x4)) &          ; le prix des trajets chers,
                         (Ville-d'arrivee (*x Bruxellss))          ; la ville d'arrivee,
                            ==> (Ville-d'arrivee (*x Bruxelles)) & ; s'il s'agit de Bruxelles,
                         (Ville-d'arrivee (*x Londres))            ; ou de Londres,
                            ==> (Ville-d'arrivee (*x Londres))
                          Heure-d'arrivee (*x *x5)                 ; et l'heure d'arrivee.
**************************

Reponses :
----------
                                                                 ; Premiere reponse : le vol AF638,
Vol (AF638) & Heure-de-depart (AF638 08h05) &                    ; qui part a 8.05 a.m,
              Heure-d'arrivee (AF638 08h55) &                    ; arrive a 8.55 a.m,
              Ville-d'arrivee (AF638 Bruxelles) &                ; a Bruxelles,
              Not (Opere (AF638 Dimanche))                       ; mais ne marche pas le dimanche.
                                                                 ; Seconde reponse : le vol AF642,
Vol (AF642) & Heure-de-depart (AF642 09h35) &                    ; qui part a 9.35 a.m,
              Heure-d'arrivee (AF642 10h25) &                    ; arrive a 10.25 a.m,
              Ville-d'arrivee (AF642 Bruxelles)                  ; a Bruxelles.
                                                                 ; Troisieme reponse : le vol AF657,
Vol (AF657) & Heure-de-depart (AF657 09h10) &                    ; qui part a 9.10 a.m,
              Heure-d'arrivee (AF657 10h05) &                    ; arrive a 10.05 a.m,
              Ville-d'arrivee (AF657 Londres)                    ; a Londres.
                                                                 ; Quatrieme reponse : le train SNCF281,
Train (SNCF281) & Heure-de-depart (SNCF281 07h48) &              ; qui part a 7.48 a.m,
                  Heure-d'arrivee (SNCF281 10h41) &              ; arrive a 10.41 a.m,
                  Ville-d'arrivee (SNCF281 Bruxelles)            ; a Bruxelles.
```