N° d'ordre 96 ISAL 0036 Année 1996

THESE

présentée devant

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

pour obtenir

LE GRADE DE DOCTEUR

FORMATION DOCTORALE : DEA Informatique de Lyon

par

Christophe RIGOTTI

Structure des objets et raisonnements dans les langages de clauses. Application à l'optimisation sémantique de requêtes

---

Soutenue le 10 Mai 1996 devant la Commission d'Examen

Jury MM. N. Bidoit (rapporteur)
F. Bry (rapporteur)
G. Gardarin
MS. Hacid
J. Kouloumdjian
L. Vieille

Remerciements

Je souhaite exprimer ma sincère gratitude envers Jacques Kouloumdjian, Professeur à l'INSA de Lyon, qui m'a accueilli dans son équipe et a mis à ma disposition les moyens matériels nécessaires à la réalisation de cette étude.

Je suis très reconnaissant à Nicole Bidoit, Professeur à l'Université Paris-Nord, et à François Bry, Professeur au Ludwig-Maximilians-Universität de Munich, d'avoir été rapporteur de mon mémoire. Je les remercie vivement pour le travail qu'ils ont consacré à son évaluation.

Je désire également remercier Georges Gardarin, Professeur à l'Université de Versailles, et Laurent Vieille, Responsable Technique - Bases de Données Avancées, à la société BULL, qui m'ont fait l'honneur d'accepter de participer à mon jury de thèse.

Je tiens à souligner tout le mérite de Mohand-Saïd Hacid, Maître de conférences à l'Université Claude Bernard de Lyon, responsable de ma thèse, dont les qualités scientifiques et l'honnêteté intellectuelle font du projet ψ-tool un réel terrain de formation par la recherche.

Ce travail ne serait pas non plus le même sans Jean-François Boulicaut, Maître de conférences à l'INSA de Lyon, dont la curiosité et la rigueur scientifique ont aidé le projet à franchir les étapes importantes.

Je remercie les autres personnes ayant participé à la relecture de se mémoire, et tout particulièrement Patrick Marcel, dont j'ai apprécié la critique constructive et l'enthousiasme dans le travail.

Je termine par un salut amical à tous les membres de l'équipe ECRIN.

Au Funambule de l'utile
et de l'inutile,
qui à la lumière de ses doutes,
jongle avec quelques signes
au parfum d'absolu.

Sommaire

Introduction

1 Problématique

1.1 Contexte de travail

Plusieurs paradigmes ont été étudiés en programmation, en représentation de connaissances ou encore pour la modélisation de données. Nous pouvons citer par exemple les paradigmes *fonctionnel*, *logique* et *objet*, qui ont été des thèmes de recherche importants des deux dernières décennies.

La communauté de recherche montre un intérêt particulier pour l'exploration des combinaisons des paradigmes connus. En effet, chaque formalisme de représentation, chaque principe de calcul possède des champs d'application privilégiés. En dehors de ceux-ci, ils peuvent devenir inappropriés, voire franchement inexploitables. Résoudre des problèmes compliqués nécessitent souvent l'utilisation conjointe de paradigmes existants. Un cas typique est la combinaison d'un principe de résolution et d'une technique de propagation de contraintes permettant de résoudre le problème de la construction de grilles de mots croisées avec un dictionnaire réaliste de 25.000 mots [39].

Cette tendance apparaît au travers des conférences et des divers ateliers de travail spécialisés ayant trait à l'intégration de plusieurs paradigmes.

Elle trouve aussi un écho au niveau du Laboratoire d'Ingénierie des Systèmes d'Information, au sein du projet ψ-tool dont la vocation est l'étude de telles combinaisons et de leurs applications dans le domaine des bases de données. Les efforts au sein de ce projet sont actuellement concentrés sur les paradigmes *objet* et *logique*, et les domaines d'application étudiés plus particulièrement sont : l'optimisation sémantique de requêtes [20] et les langages de requêtes avec contraintes [34].

1.2 Objectifs

Ce travail s'inscrit dans le cadre du projet ψ-tool, et vise à en développer deux aspects :

- étude de l'utilisation de la structure des objets dans les raisonnements associés à des langages de clauses,
- application de ces raisonnements en optimisation sémantique de requêtes dans les bases de données déductives orienté-objet.

1.3 Démarche

Lors de la première étape, qui consiste en l'étude de différentes formes de raisonnement, il est nécessaire d'aller plus loin que le simple recensement des travaux existants. En effet, même si l'on se concentre sur le sujet spécifique de *l'utilisation de la structure des objets dans les raisonnements associés à des langages de clauses*, nous nous trouvons face à une bibliographie importante.

Construire une vision très synthétique nous a semblé le passage clef pour obtenir une image claire du domaine d'étude.

Ensuite, une phase de travail plus exploratoire, concernant l'utilisation des raisonnements identifiés, est apparue nécessaire. Il s'agissait plus particulièrement pour nous d'envisager dans quelle mesure ces raisonnements pouvaient être utiles en optimisation sémantique, et de quelles façons nous pouvions les réaliser.

L'étape suivante, fut alors d'établir une spécification déclarative formelle de l'outil de calcul désiré, puis de construire une sémantique opérationnelle correspondante. Sachant qu'il n'est pas facile de reconstruire de toute pièce un canevas de démonstration de l'équivalence entre une sémantique déclarative et une sémantique opérationnelle, nous avons adapté un canevas existant.

Enfin, l'outil de calcul réalisé a été inséré dans un cadre d'optimisation sémantique de requêtes. L'essentiel fut alors de montrer que les requêtes optimisées étaient bien équivalentes aux requêtes de départ.

2 Travail réalisé

2.1 Déroulement

La collecte et la lecture de la bibliographie relative à *l'utilisation de la structure des objets dans les raisonnements associés à des langages de clauses* a nécessité un effort important. En effet, elle se répartie sur trois domaines : l'intelligence artificielle, la programmation logique, et les bases de données. Ceci a demandé un investissement allant des logiques terminologiques[1] à la F-Logique[2], en passant par des langages de programmation logique comme LIFE ou L&O et des langages pour bases de données déductives tels que COL.

Nous avons commencé notre effort de synthèse en nous concentrant tout d'abord sur les aspects opérationnels [48]. Nous l'avons ensuite poursuivi et étendu aux aspects déclaratifs. Face à l'ampleur du travail nécessaire, nous n'avons pas étudié les aspects non-monotones tels que la surcharge, les changements d'état, ou encore la prise en compte de la négation. Ceci n'a pas été génant pour l'application de notre travail à l'optimisation sémantique, et ces aspects non-monotones constituent autant de possibilités d'extensions futures.

La phase suivante (non présentée dans le mémoire) a consisté en un travail de réflexion concernant l'utilisation et la combinaison de ces raisonnements [49]. Nous avons notamment étudié les aspects opérationnels à

1. Support formel des bases de connaissances de la famille KL-ONE.

2. Une logique pour les objets et les *frames* utilisées dans le domaine des bases de données.

partir d'un prototype exploratoire. Nous avons ensuite mis en évidence l'intérêt d'une combinaison de deux des formes de raisonnement pour détecter certains cas de requêtes ne pouvant pas avoir de réponse [51]. Ceci laissait entrevoir des applications possibles en optimisation sémantique et marquait ainsi la fin de cette phase du travail.

L'étape suivante était de construire un outil formel combinant ces raisonnements. Ceci a imposé un investissement important, qui passait notamment par le choix d'un canevas existant pour la démonstration de l'équivalence entre sémantique déclarative et sémantique opérationnelle. Un travail de rapprochement [50] entre la combinaison que nous avions envisagée et la programmation logique avec contraintes, nous ont conduit à nous inspirer du canevas proposé par Höhfeld et Smolka[3]. Nous avons alors spécifié les notions de conséquence logique et de réponse correcte, puis nous avons décrit une résolution par réduction des buts pour laquelle nous avons établi la justesse et la complétude.

Après avoir montré l'intérêt des possibilités de calculs de notre outil en optimisation sémantique [28], nous avons construit un cadre formel de reformulation des requêtes [29]. Ce cadre peut être vu comme une passerelle entre l'optimisation sémantique effectuée par résolution dans les langages de clauses très généraux d'une part, et les optimisations plus fines permises par subsomption dans des langages plus restreints d'autre part. La principale limitation de notre cadre est de ne pas fournir de stratégie globale de reformulation des requêtes. La proposition d'une telle stratégie dépasse le cadre de cette thèse et en constitue une voie d'extension intéressante.

2.2 Résultats et plan du mémoire

Ce mémoire comprend trois parties, chacune correspondant à l'un des principaux résultats:

- une synthèse originale des différentes utilisations de la structure des objets dans les raisonnements associés à des langages de clauses,
- un outils formel permettant de combiner abduction et déduction de la structure des objets,
- un cadre permettant de faire cohabiter deux techniques classiques employées pour l'optimisation sémantique des requêtes: le calcul de résolvantes et la simplification basée sur une relation de subsomption décidable.

3. Ce canevas de démonstration a été proposé dans le cadre de travaux de généralisation de la programmation logique avec contraintes [30].

Première partie
Synthèse

1 Introduction

Nous présentons dans cette partie les résultats d'une synthèse bibliographique portant sur l'utilisation de la structure des objets dans les raisonnements associés à des langages de clauses. Face à l'abondante littérature, il semble illusoire de vouloir présenter une revue exhaustive des travaux existants. Nous avons donc opté pour une présentation synthétique dans laquelle nous soulignons les principaux axes de recherche au moyen de points de repères bibliographiques importants et de travaux typiques.

Nous avons identifié trois familles de langages, faisant chacune intervenir d'une façon caractéristique la structure des objets dans les raisonnements utilisés pour des langages de clauses. Nous adoptons pour chaque famille, le canevas de présentation suivant :

1. motivations liées au développement de la famille et principes généraux communs aux différents travaux apparentés.
2. description d'un langage de la famille, pour lequel nous donnons une sémantique déclarative et une sémantique opérationnelle. Ce langage, bien que ne correspondant pas directement à un langage existant, demeure typique de la famille.
3. illustration sur un exemple de l'utilisation de la structure des objets dans des raisonnements simples.
4. présentation des différents axes de recherche et des points d'entrées bibliographiques correspondants.

Cette synthèse se concentre sur les raisonnements monotones et les aspects tels que la négation, les changements d'états ou la surcharge ne sont pas considérés.

Avant de présenter les trois familles dans les sections 3, 4 et 5, précisons quelques notions communes aux différentes descriptions.

2 Notions préliminaires

Nous indiquons dans cette section les notions communes employées pour formuler les sémantiques déclaratives et opérationnelles. Nous introduisons ensuite les aspects liés à la description de la structure des objets.

2.1 Atomes et clauses

Nous utilisons deux sortes d'atomes :

- des atomes "classiques" tels que ceux constituant les clauses en programmation logique, et que nous appelons *liens atomiques*. Dans les raisonnements, ces atomes représenteront des liens (entre termes) devant être déduits.
- des atomes appelés *contraintes atomiques*, pour lesquels nous considérerons simplement la satisfaisabilité.

Soient $A_1, \ldots, A_n$ des liens atomiques, leur conjonction est notée :

$$A_1 \ \& \ \ldots \ \& \ A_n$$

Nous notons de la même façon les conjonctions de contraintes atomiques.

Les contraintes atomiques suivantes sont utilisées dans chacune des familles :

- $T_1 = T_2$, avec T_1 et T_2 des termes du premier ordre. Une telle contrainte est satisfaisable *ssi* T_1 et T_2 peuvent être unifiés.
- ϵ, qui représente la conjonction de contraintes *vide*, toujours satisfaite.

Par abus de langage, nous appelons *programme* un ensemble fini de clauses.

Dans la suite de cette présentation, A représente un lien atomique, F une conjonction de liens atomiques, H une conjonction de contraintes atomiques et C une clause. Nous notons A_g, F_g, H_g et C_g les formes correspondantes ne contenant pas de variable.

2.1.1 Sémantique déclarative

Une interprétation est un *modèle* d'un programme P *ssi* toutes les clauses de P sont satisfaites.

Une clause C est satisfaite *ssi* toutes les instances sans variable de C sont satisfaites.

Pour chaque famille de langages nous précisons dans la section correspondante la forme des interprétations, la satisfaction des clauses sans variable et la notion de réponse correcte pour un but et un programme donnés.

2.1.2 Sémantique opérationnelle

Nous décrivons les sémantiques opérationnelles sous la forme de machines abstraites non-déterministes du type "machine Prolog III" [22]. On notera que ces machines fixent la règle de choix de l'atome à effacer tout en laissant ouvert le choix de la clause candidate.

Ces machines abstraites sont définies en termes d'états et de transitions possibles entre ces états.

Un état est une paire $\langle G, S\rangle$ où G représente la conjonction d'atomes restant à effacer et S est une conjonction de contraintes devant être satisfaisable.

L'état initial de la machine représente le but à effacer.

Un état final est de la forme $\langle \epsilon, S_f\rangle$.

Les changements d'états possibles sont décrits par une ou plusieurs règles de transition sous la forme : $\langle$état de départ$\rangle \longrightarrow \langle$état d'arrivée$\rangle$. Voici, par exemple, l'unique règle d'une machine abstraite décrivant un interprète Prolog standard :

$\langle A_d \ \& \ F_d, S_d\rangle \longrightarrow \langle F_c \ \& \ F_d, S_a\rangle$
si il existe C, une clause du programme renommée avec
de nouvelles variables, de la forme $(A_c :\!- \ F_c)$
telle que la conjonction $(S_d \ \& \ A_d = A_c)$ notée S_a soit satisfaisable.

Introduisons la notion de *réponse calculée*. Pour un programme et un but donnés, l'ensemble des réponses calculées est l'ensemble des conjonctions de contraintes S_f telles que l'état final $\langle \epsilon, S_f\rangle$ peut être atteint par une séquence de transitions depuis l'état initial.

Nous préciserons pour chaque famille le contenu des états initiaux et la ou les règles de transition.

2.2 Description des objets

La notion d'objet sera définie de façon précise dans la partie 2. Pour l'instant, nous employons le sens informel, qui lui est attribué de façon usuel dans le domaine des bases de données et dans celui des langages de programmation.

Précisons cependant que, par convention :

- la notion d'objet recouvre celles d'individu et de classe.
- un même attribut d'un objet peut avoir plusieurs valeurs différentes.

Nous utilisons un langage de contraintes pour décrire la structure des objets. Ce langage provient des travaux réalisés en intelligence artificielle, dans le domaine des bases de connaissances de la famille KL-ONE[4] [13]. Une sémantique déclarative ainsi qu'un test de satisfaisabilité pour un langage de ce type sont décrits, par exemple, dans [16].

Nous définirons précisément la sémantique d'un langage de contraintes apparenté dans la partie 2. La présentation du langage de contraintes est ici informelle.

Les contraintes que nous utilisons sont :

- l'individu X appartient à l'ensemble d'individus E, notée $X : E$.
- pour l'individu X, une des valeurs de l'attribut R est Y, notée $X.R \rightarrow Y$.
- l'ensemble d'individus E_1 est inclus dans l'ensemble d'individus E_2, notée $E_1 \sqsubseteq E_2$.
- X et Y dénotent le même individu, notée $X = Y$.

Les ensembles d'individus sont décrits dans ces contraintes sous la forme de *concepts*[5]. Les concepts employés sont :

- *les noms de classe*, chaque nom étant alors interprété comme l'ensemble des individus instances de la classe.
- $and(E_1, \ldots, E_n)$, qui est interprété comme l'intersection des concepts E_1 à E_n.
- $all(R, E)$, qui est l'ensemble des individus dont toutes les valeurs de l'attribut R sont dans le concept E.
- $at\text{-}most(n, R)$, interprété comme l'ensemble des individus dont l'attribut R a au plus n valeurs distinctes.
- $not(\textit{nom de classe})$, qui représente l'ensemble des individus qui ne sont pas instances de la classe mentionnée.

Une interprétation associe à chaque nom de classe un ensemble d'individus et à chaque attribut R une relation binaire entre individus. Elle fixe donc ainsi l'ensemble d'individus associé à chaque concept.

4. Telles que CLASSIC, BACK ou encore LOOM. Un aperçu de cette famille de systèmes peut être trouvé dans [40].

5. La notion de *concept* utilisée ici, est celle des langages de la famille KL-ONE: un concept est une expression interprétée comme un ensemble d'individus.

Pour une interprétation donnée, la satisfaction des contraintes sur la structure des objets est définie de la façon suivante :

- $X : E$ est satisfaite *ssi* l'individu X appartient à l'ensemble d'individus associé au concept E.
- $X.R \rightarrow Y$ est satisfaite *ssi* $\langle X, Y \rangle$ est dans la relation associée à R.
- $E_1 \sqsubseteq E_2$ est satisfaite *ssi* l'ensemble d'individus associé à E_1 est inclus dans l'ensemble d'individus associé à E_2.
- une conjonction de contraintes est satisfaite *ssi* toutes les contraintes qu'elle contient sont satisfaites

Soient les conjonctions de contraintes H_1 et H_2. Nous dirons que H_1 implique H_2 *ssi* toute interprétation qui satisfait H_1 satisfait aussi H_2. Un test d'implication correspondant peut être construit à partir de certains tests de satisfaisabilité connus, comme par exemple celui de [16].

Notons les deux points suivants :

- la simplification des systèmes de contraintes est indispensable car elle permet d'augmenter l'efficacité des tests de satisfaisabilité et d'obtenir des systèmes de contraintes plus lisibles. Certaines formes de simplifications seront étudiées dans la partie 3, dans un cadre d'optimisation de requêtes. Mais dès à présent, dans le but de faciliter la lecture, nous présenterons dans les exemples des systèmes simplifiés.
- nous utiliserons simultanément des contraintes sur la structure des objets et des contraintes d'égalité entre termes du premier ordre, ces dernières étant employées pour décrire de façon concise les contraintes d'unification. Afin de ne pas compliquer le langage de contraintes, nous considérons que ces contraintes d'égalité entre termes du premier ordre ne sont qu'un sucre syntaxique. En effet, elles peuvent être codées sous la forme de contraintes sur la structure des objets. Ainsi la contrainte $f(X, a) = f(b, c)$ peut être codée par la conjonction :

$$
\begin{aligned}
& terme \sqsubseteq and(\textit{at-most}(1, foncteur), \\
& \qquad \textit{at-most}(1, arg_1), \textit{at-most}(1, arg_2)) \\
& \& \; T_1 : terme \; \& \; T_2 : terme \\
& \& \; T_1.foncteur \rightarrow f \; \& \; T_2.foncteur \rightarrow f \\
& \& \; T_1.arg_1 \rightarrow X \; \& \; T_2.arg_1 \rightarrow b \\
& \& \; T_1.arg_2 \rightarrow a \; \& \; T_2.arg_2 \rightarrow c \; \& \; T_1 = T_2
\end{aligned}
$$

  Cette conjonction de contraintes n'est bien sûr pas satisfaisable puisque $f(X, a) = f(b, c)$ ne l'est pas non plus[6].

6. Un codage similaire est utilisé dans LIFE pour décrire les termes du premier ordre sous la forme de ψ-termes [4].

3 Famille I : Objets et hypothèses

Pour les langages de cette première famille, la notion d'objet à été intégrée dans les clauses, au niveau des termes. La structure des objets est alors décrite au moyen de contraintes portant sur ces termes, et utilisée comme un ensemble d'hypothèses devant être satisfaites.

3.1 Présentation

Deux préoccupations sont à l'origine des langages de cette famille. La première est de rendre plus aisé le codage des structures de données. En programmation logique classique, les structures de données sont décrites sous la forme de termes du premier ordre. Ceci présente l'avantage d'une grande homogénéité mais nécessite souvent des efforts lors de la lecture et de l'écriture des programmes. Il faut ainsi se rappeler que personne(madison, ramakrishnan, raghu) représente Raghu Ramakrishnan de Madison (USA). L'introduction de la notion d'objet au niveau des termes permet d'effectuer cette description au moyen de la conjonction de contraintes ramakrishnan:personne & ramakrishnan.prénom → raghu & ramakrishnan.habite → madison.

Autre avantage, ceci facilite la modification des programmes. Supposons que l'on désire ajouter la profession des personnes. Le codage sous forme de termes personne(Lieu, Nom, Prénom) nécessite de reprendre toutes les parties du programme accèdant à cette structure de données pour les réécrire en personne(Lieu, Nom, Prénom, Profession). L'utilisation de contraintes décrivant des objets au niveau des termes permet, elle, de ne mentionner que les composantes manipulées dans la structure de données. Un accès à la profession peut par exemple être codé par ramakrishnan.profession → Y et n'impose pas de modifier les parties du programme réalisant des accès au reste de la structure.

La seconde motivation pour l'utilisation de la structure des objets sous la forme de contraintes au niveau des termes est de gagner en efficacité. Ces gains sont réalisés tous d'abord en utilisant sur les contraintes des algorithmes de calculs spécialisés (et donc plus efficaces) plutôt que d'avoir à réaliser ces calculs à l'aide de pas de résolution. D'autres gains sont obtenus en réduisant l'espace de recherche par une programmation du type *contraindre et générer*. Celle-ci consiste à limiter le nombre des combinaisons générées (essayées) lors de la résolution d'un problème, en restreignant par des contraintes les combinaisons possibles.

Les deux motivations, qui ont conduit à cette introduction de la notion d'objet au niveau des termes, sont très proches de celles qui ont encouragées le développement des langages de programmation logique avec contraintes

(CLP). La famille de langages, décrite ici, peut d'ailleurs être vue comme une généralisation des schémas CLP(X) de [31].

La forme générale des clauses est $A :\!- F \mathbin{/\!/} H$ où A est un lien atomique, F est une conjonction de liens atomiques et H une conjonction de contraintes atomiques. Intuitivement, la lecture déclarative d'une telle clause est :
Si les termes de $A :\!- F$ *vérifient* H *et si* F *est vraie alors* A *est vrai.*

Un but de la forme $F \mathbin{/\!/} H$ se lira quant à lui :
Peut-on prouver que lorsque H *est vérifiée,* F *est vraie ?*

Une réponse R pour ce but, est une conjonction de contraintes telle que :
on peut prouver que lorsque R *et* H *sont vérifiées,* F *est vraie.*

Cette lecture est à l'origine d'une vision en terme d'abduction de la CLP [41] et d'une façon plus générale de la résolution sur des clauses avec contraintes [19]. En effet, R peut être vue comme un ensemble d'hypothèses, portant sur les termes du but, tel que si ces hypothèses sont satisfaites alors $F \mathbin{/\!/} H$ peut être déduit du programme.

La sémantique opérationnelle associée peut ainsi être apparentée à une procédure d'abduction. Elle consiste, lors de l'effacement d'un but, à accumuler les contraintes des clauses utilisées, et à ne poursuivre l'exploration d'une branche de résolution que si ces contraintes forment un ensemble d'hypothèses acceptables, c'est-à-dire pouvant être satisfaites.

Lorsque tous les liens atomiques ont pu être effacés du but, l'ensemble des contraintes collectées constitue une hypothèse-réponse.

Dans notre cas, les contraintes décrivent les structures des objets. Les hypothèses-réponses portent donc, elles aussi, sur ces structures.

Nous décrivons à présent un langage typique de cette famille. La sémantique déclarative et la sémantique opérationnelle sont formulées en donnant seulement les définitions complétant celles de la section 2.

3.2 Sémantique déclarative

Interprétations

Une interprétation est une paire $\langle \mathcal{I}, \mathcal{B} \rangle$. $\mathcal{I}$ est utilisé pour interpréter les conjonctions de liens atomiques alors que $\mathcal{B}$ est employé pour interpréter les conjonctions de contraintes atomiques.

Soit une interprétation $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$.

Satisfaction des clauses sans variable

Une conjonction de liens atomiques sans variable, de la forme A_{g_1} & ... & A_{g_n} est vraie dans $\mathcal{I}$ *ssi* $A_{g_1}, \ldots, A_{g_n}$ sont vrais dans $\mathcal{I}$.

Une formule sans variable, de la forme F_g // H_g est vraie dans Γ *ssi* F_g est vraie dans $\mathcal{I}$ et H_g est vraie dans $\mathcal{B}$.

Une clause sans variable A_g :– F_g // H_g est satisfaite par Γ *ssi* F_g // H_g est fausse dans Γ ou A_g est vrai dans $\mathcal{I}$.

Réponses correctes

Soient une conjonction de contraintes R et un but G. La formule $R \Rightarrow G$ est satisfaite par Γ *ssi* pour toute instance sans variable $R_g \Rightarrow G_g$ de $R \Rightarrow G$, R_g est fausse dans $\mathcal{B}$ ou G_g est vrai dans Γ.

Soient un programme P et un but G. Une réponse correcte R pour G est une conjonction de contraintes telle que, pour toute interprétation Ω modèle de P, Ω satisfait $R \Rightarrow G$.

3.3 Sémantique opérationnelle

Soit le but ?– F // H.
L'état initial de la machine abstraite est alors $\langle F, H \rangle$.

L'unique règle de transition est :

$\langle A_d \ \& \ F_d, S_d \rangle \longrightarrow \langle F_c \ \& \ F_d, S_a \rangle$
si il existe $C = (A_c$:– F_c // $H_c)$
une clause du programme renommée avec de nouvelles variables
telle que $S_a = (S_d \ \& \ H_c \ \& \ A_d = A_c)$ soit satisfaisable.

3.4 Illustration

Commentons, sur des exemples, l'utilisation du langage qui vient d'être défini.

Soit le programme :

observe(cathy,tweety).

calme(john).

calme(X) :– observe(X,Y) // X:personne.

apprécie(X,X) :– // X:personne. [7]

apprécie(X,Y) :– calme(Y) // X:femme & X.enfant → Y. [8]

7. D'après [5] ...
8. Adaptation d'une clause de [47] ...

Rappelons, à partir de la dernière de ces clauses, leur sens informel. Nous pouvons lire cette clause simplement de la façon suivante : *Si l'on peut faire l'hypothèse que* X *est une femme ayant au moins un enfant* Y *et si l'on peut déduire que ce* Y *est calme alors on déduit que* X *apprécie* Y.

Pour un but donné, les réponses contiennent des contraintes d'égalité décrivant des unifications nécessaires à l'effacement de ce but, et des contraintes sur la structure des objets qui ont été collectées lors de l'utilisation des clauses du programme.
Ainsi, pour le but simple :

?– calme(X).

nous obtenons les réponses triviales (et simplifiées) :

X = john
X = cathy & cathy:personne[9]

Les réponses peuvent être perçues comme des hypothèses portant sur la structure des objets mentionnés dans le programme.
Au but :

?– apprécie(mary,john).

nous avons comme réponse :

mary:femme & mary.enfant → john

que nous pouvons lire : *Sous l'hypothèse que* mary *soit une femme ayant pour enfant* john, *on peut déduire du programme que* mary *apprécie* john.

Les hypothèses permettent de formuler des réponses quasiment intentionnelles. Par exemple, considérons le but :

?– apprécie(X,Y).

Les réponses sont :

X:personne & X = Y
X:femme & X.enfant → Y & Y = john
X:femme & X.enfant → Y & Y = cathy & cathy:personne

Le caractère intentionnel des réponses se retrouve notamment dans la seconde, qui peut se lire : *Toute femme ayant pour enfant* john *apprécie* john.

Les contraintes peuvent aussi être utilisées directement au niveau des buts, comme dans :

?– apprécie(cathy,Y) // cathy:at-most(0,enfant).

que nous pouvons lire : *Peut-on déduire que* cathy *apprécie quelqu'un, sous l'hypothèse qu'elle n'ait pas d'enfant ?*
Puisque cathy ne doit pas avoir d'enfant nous obtenons une seule réponse :

cathy:at-most(0,enfant) & cathy:personne & cathy = Y

9. Nous ne nous intéressons pas à l'ordre d'obtention des réponses. Cet ordre n'est pas fixé par la machine abstraite.

C'est-à-dire : *Sous l'hypothèse que* cathy *n'ait pas d'enfant et qu'elle soit une personne identique à* Y, *nous pouvons déduire que* cathy *apprécie* Y.

Des contraintes plus complexes peuvent faire intervenir le type des attributs. Soit $\mathcal{CT}$ la conjonction des contraintes suivantes : cathy *et* mary *sont des femmes,* mary *n'a que des fils,* femme *et* homme *sont deux classes disjointes.* Elle s'écrit :

$\mathcal{CT}$ = cathy:femme & mary:and(femme,all(enfant,homme)) & femme $\sqsubseteq$ not(homme)

Soit le but :

Quels sont les X *pour lesquels nous pouvons déduire qu'ils sont calmes et appréciés par* mary, *sous l'hypothèse* $\mathcal{CT}$ *?*

Il s'écrit :

?– calme(X) & apprécie(mary,X) // $\mathcal{CT}$.

L'unique réponse est :

mary:femme & mary.enfant $\rightarrow$ X & X = john & $\mathcal{CT}$

En effet, calme(X) impose X = john ou X = cathy. X = cathy n'est pas possible pour apprécie(mary,X) car cathy est une femme et mary n'a que des fils. En revanche, pour X = john il est possible que X soit un des fils de mary, car rien n'indique que john soit une femme.

Enfin, voici un exemple illustrant la propagation des contraintes vers les sous-classes. Ajoutons à l'ensemble des clauses le fait calme(clyde), et modifions $\mathcal{CT}$:

$\mathcal{CT}$ = cathy:femme & mary:and(femme,all(enfant,homme)) & clyde:éléphant & personne $\sqsubseteq$ not(éléphant) & femme $\sqsubseteq$ and(personne, not(homme)) & homme $\sqsubseteq$ personne

Conservons le but :

?– calme(X) & apprécie(mary,X) // $\mathcal{CT}$.

Bien que clyde et john apparaîssent de la même façon dans les faits du programme, nous obtenons toujours une seule et même réponse :

mary:femme & mary.enfant $\rightarrow$ X & X = john & $\mathcal{CT}$

En effet puisque personne et éléphant sont des classes disjointes, homme et éléphant le sont aussi (propagation des contraintes vers les sous-classes), et donc clyde ne peut être un enfant de mary !

3.5 Famille

Cette famille regroupe des langages dans lesquels des contraintes particulières sont utilisées pour décrire les termes sous la forme d'objets. Ces contraintes, lorsqu'elles sont fournies en tant que réponses, peuvent être perçues comme des hypothèses. Nous avons décrit une sémantique déclarative et une sématique opérationnelle, qui, bien qu'elles ne correspondent à aucun langage existant réellement, sont typiques de cette famille. Il ne s'agit

pas, dans cette section, de répertorier tous les langages apparentés à celui que nous venons de présenter, mais plutôt d'indiquer les membres les plus représentatifs de cette famille et de souligner les principaux axes de travail.

A l'origine de cette famille nous trouvons le langage LOGIN [5]. Dans ce langage, les descriptions de la structure des objets sont effectuées sous la forme de termes, appelés ψ-termes, et sont assez différentes de celles que nous avons présentées. Les travaux qui ont suivi LOGIN se sont faits selon trois axes privilégiés :

1. *l'extension des descriptions faites au niveau des termes.* Nous citerons par exemple, dans cette voie, l'ajout d'une forme de typage des attributs aux ψ-termes de LOGIN réalisée dans U-Log [27].

   Ces travaux à partir de LOGIN, ont abouti actuellement à un langage stable appelé LIFE [6]. Il faut noter que les possibilités de LIFE vont au-delà de celles associées à cette famille de langages, puisqu'il intègre aussi le paradigme fonctionnel.

2. *l'intégration entre résolution SLD et satisfaction de contraintes.* Nous trouvons ici notamment les travaux de Höhfeld et Smolka [30] qui ont montré les liens entre cette famille de langages et le schéma CLP(X) [31], ou encore ceux de Mamede et Montero [42]. Une présentation de la correspondance entre ces formes de résolutions avec contraintes et la génération d'hypothèses a été réalisée par Bürckert et Nutt [19].

3. *la représentation de connaissances.* Cet axe vise à exploiter les deux niveaux de descriptions permis par l'utilisation conjointe des clauses et des contraintes. Les clauses servent alors à décrire la connaissance liée à la façon de résoudre un problème et les contraintes sont employées pour décrire la connaissance liée au domaine dans lequel le problème doit être résolu.

   Des langages, tels que TaxLog [1] et $\mathcal{AL}$-log [25], permettant d'exploiter cette complémentarité, ont été développés au sein de la communauté de recherche en intelligence artificielle. Un cadre formel correspondant a été proposé par Baader et al. [7]. Notons que les descriptions de la structure des objets dans ces langages sont très proches de celles que nous avons présentées.

4 Famille II : Objets et contextes

Dans cette famille de langages, les programmes sont décomposés en théories, chaque théorie ne pouvant s'appliquer que dans un contexte particulier. La structure des objets est alors utilisée pour décrire ces contextes.

4.1 Présentation

Simplifier l'écriture des programmes et gagner en efficacité ont été les motivations principales de développement de la famille précédente. Elles vont se trouver encore à l'origine de cette seconde famille. Cependant, ces préoccupations ne sont plus prises en compte lors du codage et de la manipulation des structures de données (au niveau des termes), elles interviennent maintenant de façon globale au niveau de la structuration des programmes.

Les travaux sur les langages et méthodes de programmation, ont montré que le concept de module était fondamental pour faciliter l'écriture, la compréhension et la mise au point des programmes.

Dans les langages de clauses, ces modules sont apparus via le découpage des programmes en plusieurs théories logiques. Le principe est simple, chaque module contient une théorie, et peut être utilisé pour décrire les propriétés de telle ou telle structure de données, ou bien regrouper les théorèmes utiles pour résoudre un type particulier de problèmes. Ensuite, dans le corps des clauses, au niveau de chaque atome, il est possible d'indiquer le module (et donc la théorie) devant être utilisé pour prouver cet atome.

On notera que cette forme de modules répond aussi (en partie) aux préoccupations liées à l'efficacité. En effet, lors de l'effacement d'un atome, si la théorie (et donc la partie du programme) à utiliser est connue, ceci restreint le nombre de clauses candidates.

Afin de rendre possible la description d'une théorie particulière par raffinement d'une théorie plus générale, la notion de spécialisation entre modules a été introduite. Pour cela, chaque module est associé à une étiquette qui décrit les objets pour lesquels la théorie du module s'applique.

Ces modules sont de la forme :

[étiquette]
clause 1.
⋮
clause n.

Un module M_1 spécialise un module M_2 si l'étiquette de M_1 décrit des objets plus spécifiques que l'étiquette de M_2. Dans le corps des clauses, chaque atome A est lui aussi associé à une étiquette E. Une telle association,

notée $[E]A$ indique que A devra être prouvé en utilisant uniquement les modules dont l'étiquette décrit des objets plus généraux que ceux décrits par E.

Dans cette présentation, nous utilisons comme étiquettes les conjonctions de contraintes sur la structure des objets introduites dans la section 2.2.

Nous décidons alors que les objets décrits par une étiquette E_1 sont plus spécifiques (moins généraux) que ceux décrits par une étiquette E_2 *ssi* E_1 implique E_2.

Sur le plan opérationnel, l'étiquette d'un module impose le contexte[10] minimal nécessaire pour que les clauses du module puissent être utilisées. Par exemple, les clauses d'un module mère (décrivant les propriétés d'une mère) ne peuvent être sélectionnées que pour prouver des propriétés liées à une femme ayant au moins un enfant. Ce contexte minimal requis pour accéder aux clauses du module, est décrit au moyen de l'étiquette X:femme & X.enfant $\rightarrow$ Y.

Considérons un but G devant être effacé dans le contexte T et un atome étiqueté $[E]A$ de G.

Dans ce cas, les clauses du module mère ne pourront être utilisées pour effacer A que si l'étiquette du module mère décrit des objets plus généraux que E & T. C'est à dire si E & T représente un contexte plus spécifique que X:femme & X.enfant $\rightarrow$ Y comme par exemple X:femme & X.enfant $\rightarrow$ Y & Y:homme.

En ce qui concerne la sémantique déclarative, elle peut être perçue de façon intuitive à l'aide de la modalité *savoir*.

Le sens d'un atome et d'une étiquette associée, $[E]A$, est alors :
Si je sais que E est vraie alors A est vrai[11].

Une clause C d'un module d'étiquette E se lit :
Si je sais que E est vraie alors C est vrai[12].

Et enfin, le sens d'un but simple comme ?– $[E]A$ est :
Peut-on prouver que si je sais que E est vraie alors A est vrai ?

L'idée d'une approche modale, pour décrire une sémantique déclarative typique de cette famille de langage, nous vient de [26]. Cependant, les sémantiques (déclarative et opérationnelle) que nous présentons maintenant, sont à notre connaissance originales.

10. Dans le sens général de contexte d'exécution.
11. Ce que l'on peut traduire approximativement en logique modale S4 par $\Box E \supset A$.
12. Que l'on peut traduire par $\Box E \supset C$.

4.1.1 Sémantique déclarative

Interprétations

Une interprétation Γ est un triplet $\langle \mathcal{W}, r, e \rangle$. Où $\mathcal{W}$ est un ensemble de mondes, r est une relation binaire sur $\mathcal{W}$, réflexive et transitive, représentant une relation d'accessibilité entre les mondes, et e est une fonction qui associe à chaque $w \in \mathcal{W}$ un ensemble d'atomes clos. $e(w)$ représente l'ensemble des atomes clos qui ont pour valeur de vérité *vrai* dans le monde w.

Une conjonction d'atomes clos est vraie dans w *ssi* tous les atomes de la conjonction sont membres de $e(w)$.

Satisfaction des clauses sans variable

Soient une interprétation $\Gamma = \langle \mathcal{W}, r, e \rangle$ et $w \in \mathcal{W}$ un monde.

Une formule sans variable, de la forme $[H_g]A_g$, est vraie dans w *ssi* H_g est fausse dans un monde $w' \in \mathcal{W}$ tel que $r(w, w')$ ou A_g est vrai dans w.

Une conjonction de liens atomiques avec étiquettes, sans variable, de la forme $[H_{g_1}]A_{g_1}$ & ... & $[H_{g_n}]A_{g_n}$ est vraie dans w *ssi* $[H_{g_1}]A_{g_1}, \ldots, [H_{g_n}]A_{g_n}$ sont vrais dans w.

Soit F_g une conjonction de liens atomiques avec étiquettes, sans variable. Une formule sans variable, de la forme A_g :– F_g, est vraie dans w *ssi* F_g est fausse dans w ou A_g est vrai dans w.

Considérons qu'une instance d'une clause d'un module M est étiquetée avec l'instance de l'étiquette de M.

Alors, une clause sans variable $[H_g]A_g$:– F_g est vraie dans w *ssi* H_g est fausse dans un monde $w' \in \mathcal{W}$ tel que $r(w, w')$ ou A_g :– F_g est vraie dans w.

Une clause sans variable $[H_g]A_g$:– F_g est satisfaite par $\Gamma = \langle \mathcal{W}, r \rangle$ *ssi* elle est vraie pour tout $w \in \mathcal{W}$.

Réponses correctes

Soient un programme P et un but G. Une réponse correcte σ pour G est une substitution telle que pour toute interprétation Ω modèle de P, Ω satisfait $\sigma(G)$.

4.2 Sémantique opérationnelle

Soit le but ?– F, l'état initial de la machine abstraite est alors $\langle F, \epsilon \rangle$.

Pour simplifier la présentation, nous considérons qu'à chaque clause est associée l'étiquette de son module.

La règle de transition est alors
$\langle [H_d]A_d \ \& \ F_d, S_d\rangle \longrightarrow \langle F_a, S_a\rangle$
avec $F_a = ([H_d \ \& \ H_1]A_1 \ \& \ \ldots \ \& \ [H_d \ \& \ H_n]A_n \ \& \ F_d)$
si il existe $C = ([H_c]A_c :\!- [H_1]A_1 \ \& \ \ldots \ \& \ [H_n]A_n)$
une clause du programme renommée avec de nouvelles variables[13]
telle que pour $S_a = (S_d \ \& \ A_d = A_c)$, nous avons $S_a \ \& \ H_d$ satisfaisable
et $S_a \ \& \ H_d$ implique $S_a \ \& \ H_c$.

Toujours afin de simplifier la présentation, nous considérons que les réponses calculées sont des conjonctions de contraintes d'égalités entre des termes du premier ordre alors que les réponses correctes sont des substitutions. Il serait cependant aisé de définir des correspondances entre les deux.

4.3 Illustration

Soient les quatre modules suivants, décrivant respectivement des propriétés applicables aux personnes, aux mères (femmes ayant au moins un enfant), à Cathy et à John :

```
[X:personne]
        calme(X) :- [ϵ]observe(X,Y).
        apprécie(X,X).

[X:femme & X.enfant → Y]
        apprécie(X,Y) :- [Y:personne] calme(Y).

[cathy:femme]
        observe(cathy,tweety).

[john:homme]
        calme(john).
```

Un tel programme ne permet de déduire des liens atomiques que lorsque nous *savons* suffisamment de choses concernant les termes sur lesquels portent ces liens atomiques.

Considérons le but suivant :

```
?- calme(john).
```

Puisque l'on ne *sait* rien sur john au niveau du but, aucun module ne peut être utilisé, et calme(john) ne peut pas être déduit du programme.

En revanche, le but :

```
?- [john:homme] calme(john)
```

contient des informations autorisant la sélection de la clause du module [john:homme] permettant de déduire calme(john).

13. $[H_c]$ est l'étiquette du module de C, après renommage.

Les informations fournies ne sont pas toujours suffisantes. Tel est le cas lorsque l'étiquette employée au niveau de l'atome à prouver ne décrit pas des objets assez spécifiques. Soit le but : *Peut-on prouver que si je sais que* cathy *est une personne alors* cathy *observe un* Y *?*
Il s'écrit :

[cathy:personne] observe(cathy,Y).

Ce but n'a pas de réponse correcte. En effet on ne peut prouver observe(cathy,Y) que si l'on sait que cathy est une femme. Pour obtenir un ensemble non vide de réponses il faut poser le but :

[cathy:femme] observe(cathy,Y).

Si il est nécessaire de fournir, au niveau du but, suffisamment d'informations concernant la structure des objets, en revanche, l'identité des individus peut, elle, ne pas être indiquée. En effet, elle peut être obtenue à partir des contraintes d'égalités décrivant les unifications. Ainsi pour le but :

[X:femme] observe(X,Y).

Nous avons comme réponse :

X = cathy & Y = tweety

On conserve donc, même en présence de variables dans les étiquettes, la possibilité d'effacer un but en réalisant une preuve constructive, et donc un calcul de réponse.

Grâce à cette particularité, le but :

?– [john:homme] calme(john)

utilisé précédemment, peut être posé sous la forme plus générale :

?– [X:homme] calme(X)

Il a alors pour réponse :

X = john

Les contraintes d'inclusion entre classes permettent d'imposer des spécialisations entre les modules. Ainsi le but :

?– [X:femme & femme $\sqsubseteq$ personne] calme(X)

peut être prouvé à l'aide des clauses des modules ayant pour étiquettes [X:personne] et [cathy:femme]. La réponse obtenue est alors :

X = cathy

Si les contraintes utilisées comme étiquettes dans les buts permettent d'accéder aux clauses des modules, des contraintes trop spécifiques peuvent interdire l'utilisation de certaines clauses.
Considérons la conjonction :
$\mathcal{CT}$ = femme $\sqsubseteq$ personne & homme $\sqsubseteq$ and(personne, not(femme))
et le but :

?– [X:personne & $\mathcal{CT}$] apprécie(X,Y).

Nous avons la réponse :

$X = Y$

Alors qu'une étiquette décrivant des objets plus spécifiques, comme par exemple dans le but :

?– [X:femme & X.enfant $\rightarrow$ Y & Y:homme & $\mathcal{CT}$] apprécie(X,Y).

ne permet plus d'obtenir cette réponse. En effet, un même individu ne peut pas être simultanément instance de femme et homme. En revanche, dans ce dernier exemple, les contraintes ajoutées permettent d'obtenir une nouvelle réponse :

$Y = \text{john}$

4.4 Famille

Nous situons dans cette famille les langages permettant d'une part de regrouper les clauses en modules, et d'autre part de construire de nouveaux modules par spécialisation et combinaison de modules plus généraux déjà définis. Cette fois encore, le langage décrit ne correspond pas à un langage particulier existant, mais il reste cependant un représentant typique de cette famille. Les langages apparentés ont été développés selon deux courants majeurs :

1. *l'intégration de la programmation orienté-objet et de la programmation logique.* Cette lignée posséde des représentants assez bien diffusés tels que L&O [43] et plus récemment Prolog++ [45]. Il faut remarquer que ces langages intègrent aussi certains des aspects impératifs de la programmation orienté-objet, que nous ne considérons pas dans cette étude. Il s'agit notamment d'extensions permettant de gérer les changements d'états des objets.

2. *la définition de modules et la combinaison de théories en programmation logique.* Cette voie a donné lieu à de nombreux travaux (voir [18]), et elle semble être la seule issue vers la structuration des grands programmes logiques et d'une façon plus générale vers celle des bases de connaissances importantes. Le travail que nous avons réalisé pour formuler une sémantique déclarative typique de cette famille, a pour origine l'approche modale de la structuration des programmes logiques décrite dans [26]. Les relations entre héritage et combinaison de théories logiques sont, quant à elles, discutées par exemple dans [44].

5 Famille III : Objets et faits connus ou déduits

Les deux familles précédentes trouvent leurs origines dans le domaine de la programmation logique. Cette troisième famille, elle, s'est plutôt développée dans la communauté des bases de données. La structure des objets est utilisée comme un ensemble de données connues (la partie extentionnelle de la base) et comme un ensemble de données déduites (la partie intentionnelle).

5.1 Présentation

L'intégration entre clauses et structure des objets réalisée dans cette famille peut être perçue selon deux points de vue :

1. ajout de possibilités de déductions aux bases de données orienté-objet.
2. prise en compte du paradigme objet dans les bases de données déductives.

Comme pour les deux familles déjà présentées, nous évoquons seulement les aspects concernant la structure des objets, et nous ne mentionnons pas les motivations liées à l'utilisation des méthodes et à la prise en compte de la dynamique.

Dans la première perspective, celle de l'addition de capacités déductives à une bases de données orienté-objet, on cherche des retombées similaires à celles des travaux effectués dans le domaine des bases de données déductives classique, c'est-à-dire notamment :

- l'augmentation du pouvoir d'expression des langages de requêtes.
- l'approfondissement des fondements théoriques.
- le développement des moyens de vérification de l'intégrité, en utilisant les méthodes d'évaluation et de maintenance de vues.

Ici l'utilisation de la structure des objets comme faits connus ou déduits apparaît de façon directe. En effet, la déduction d'information s'appuie sur le contenu de la base extentionnelle (EDB) décrivant la structure d'un ensemble d'objets, et permet d'obtenir dans la partie intentionnelle (IDB) de nouvelles structures.

Lorsque l'on considère la seconde vision, celle d'une intégration de la notion d'objet dans les clauses décrivant l'IDB, là encore la transformation semble assez naturelle. On peut d'ailleurs la percevoir simplement comme une adaptation de la sémantique de façon à prendre en compte l'usage fréquent de certaines descriptions.
Par exemple, on rencontre rarement en tête de clause le lien

$personne(Nom, Enfant, Observe, Apprécie, Est\text{-}calme, Est\text{-}une\text{-}mère)$.

Il est en revanche plus habituel de trouver ces informations décrites à partir de plusieurs clauses dont les têtes pourraient être :

$personne(Nom)$
$calme(Nom)$
$mère(Nom)$
$enfant(Nom1, Nom2)$
$observe(Nom1, Nom2)$
$apprécie(Nom1, Nom2)$

Ou encore, sous une forme plus générale :

$est\text{-}un(Nom, personne)$
$est\text{-}un(Nom, calme)$
$est\text{-}un(Nom, mère)$
$sorte\text{-}de(mère, personne)$
$attribut(Nom1, enfant, Nom2)$
$attribut(Nom1, observe, Nom2)$
$attribut(Nom1, apprécie, Nom2)$

L'emploi répété de ce type de descriptions suggère de considérer trois formes de liens atomiques spécifiques pour décrire la structure des objets :

- $X : Y$, X est instance de Y ($est\text{-}un/2$).
- $X < Y$, X hérite de Y[14] ($sorte\text{-}de/2$).
- $X.R \rightarrow Y$, Y est l'une des valeurs de l'attribut R pour X ($attribut/3$).

Ces liens doivent respecter certaines propriétés. Les deux principales sont la transitivité de l'héritage et l'appartenance des instances d'une classe à ses super-classes.

La sémantique déclarative doit intégrer ces liens particuliers. Ceci peut être réalisé de trois façons :

- en indiquant comment traduire un ensemble de clauses du langage sous la forme d'une théorie de la logique des prédicats, et en incluant éventuellement des axiomes spécifiques comme :
  $\forall X, \forall Y, \forall Z,\ X : Y \wedge Y < Z \Rightarrow X : Z$.
- en plaçant des restrictions sur l'interprétation de ces liens. Ainsi, il est possible d'imposer que le prédicat décrivant les liens d'héritages ne soit interprété que par une relation transitive.

14. Attention, un lien d'héritage n'est pas équivalent à une contrainte d'inclusion $X \sqsubseteq Y$.

– en changeant l'opérateur de conséquence immédiate T_P associé à un programme P pour générer des liens implicites. Par exemple pour un ensemble Γ de liens atomiques sans variable, si $a : b$ et $b < c$ sont dans Γ, alors $T_P(\Gamma)$ doit contenir $a : c$.

En ce qui concerne la sémantique opérationnelle, trois solutions peuvent être envisagées :

– ajouter au programme des clauses décrivant les propriétés des liens $X : Y$, $X < Y$ et $X.R \rightarrow Y$, comme par exemple $X : Y$:– $X : Z$ & $Z < Y$, et utiliser des sémantiques opérationnelles classiques.

– utiliser un nouveau système d'inférence pour raisonner sur les clauses et sur les buts, en tenant compte directement des propriétés de $X : Y$, $X < Y$ et $X.R \rightarrow Y$.

– évaluer les requêtes à l'aide d'une procédure naïve basée sur le nouvel opérateur T_P.

Il est ensuite possible d'adapter des méthodes telles que celle des *ensembles magiques* [9] ou la méthode *Requêtes-Sous-Requêtes* [54], ou encore de se placer dans un cadre général d'évaluation tel que [14].

Pour conserver une présentation homogène vis-à-vis des familles précédentes, nous donnons une sémantique déclarative typique en termes d'interprétations et de satisfaction des clauses, puis une sémantique opérationnelle permettant de raisonner sur les buts.

5.2 Sémantique déclarative

Interprétations

Nous pouvons utiliser des interprétations classiques, en nous restreignant aux interprétations Γ telles que :
si les relations binaires $:^\Gamma$ et $<^\Gamma$ sont les interprétations respectives de : et $<$, alors (1) $<^\Gamma$ est une relation d'ordre strict, et (2) pour tous d_1, d_2 et d_3 éléments du domaine d'interprétation, si $a :^\Gamma b$ et $b <^\Gamma c$ alors $a :^\Gamma c$.

Satisfaction des clauses sans variable

Une conjonction de liens atomiques sans variable, de la forme
A_{g_1} & ... & A_{g_n} est vraie dans Γ *ssi* $A_{g_1}, \ldots, A_{g_n}$ sont vrais dans Γ.

Une clause sans variable A_g :– F_g est satisfaite par Γ *ssi* F_g est fausse dans Γ ou A_g est vrai dans Γ.

Réponses correctes

Soient un programme P et un but G. Une réponse correcte σ pour G est une substitution telle que, pour toute interprétation Ω modèle de P, Ω satisfait $\sigma(G)$.

5.3 Sémantique opérationnelle

Soit le but ?– F, l'état initial de la machine abstraite est alors $\langle F, \epsilon \rangle$.

Les règles de transition sont :

$\langle A_d \ \& \ F_d, S_d \rangle \longrightarrow \langle F_c \ \& \ F_d, S_a \rangle$
si il existe $C = (A_c \text{ :- } F_c)$
une clause du programme renommée avec de nouvelles variables
telle que $S_a = (S_d \ \& \ A_d = A_c)$ soit satisfaisable.

$\langle X : Y_d \ \& \ F_d, S_d \rangle \longrightarrow \langle X : Z \ \& \ F_c \ \& \ F_d, S_a \rangle$
si il existe $C = (Z < Y_c \text{ :- } F_c)$
une clause du programme renommée avec de nouvelles variables
telle que $S_a = (S_d \ \& \ Y_d = Y_c)$ soit satisfaisable.

$\langle X < Y_d \ \& \ F_d, S_d \rangle \longrightarrow \langle X < Z \ \& \ F_c \ \& \ F_d, S_a \rangle$
si il existe $C = (Z < Y_c \text{ :- } F_c)$
une clause du programme renommée avec de nouvelles variables
telle que $S_a = (S_d \ \& \ Y_d = Y_c)$ soit satisfaisable.

Comme dans la section 4, afin de simplifier la présentation, les réponses calculées sont des conjonctions de contraintes et les réponses correctes sont des substitutions.

5.4 Illustration

Le schéma, l'IDB et l'EDB sont décrits au moyen de clauses, et la différenciation de ces trois parties n'est pas essentielle pour la présentation de cette synthèse. Cependant, nous les présentons séparément de façon à rappeler que l'origine des langages de cette famille se situe dans la communauté des bases de données.

Soit le schéma :

femme $<$ personne.

mère $<$ femme.

homme $<$ personne.

Les clauses décrivant l'IDB sont :

X:calme :– X.observe → Y.

X:mère :– X.enfant → Y.

X.apprécie → X :– X:personne.

X.apprécie → Y :– X:femme & X.enfant → Y & Y:calme.

Enfin l'EDB utilisée est :

cathy:femme.

cathy.observe → tweety.

john:homme.

john:calme.

mary.enfant → john.

La perception intuitive que l'on peut avoir d'une base de données déductive classique reste applicable. Ainsi le but :
Quels sont les X *pour lesquels on peut déduire qu'ils sont calmes?*
qui s'écrit :

?– X:calme.

a bien les réponses attendues :

X = john
X = cathy

Toutefois, en raison de l'inclusion de l'extension d'une classe dans ses super-classes, des réponses supplémentaires peuvent exister. Ainsi, pour le but :

?– X:personne.

nous avons les réponses :

X = cathy
X = mary
X = john

dûes respectivement aux liens d'héritage
femme < personne, mère < femme et homme < personne.

De même, grâce à ces liens, pour le but :

?– X.apprécie → Y.

nous obtenons :

X = cathy, Y = cathy
X = john, Y = john
X = mary, Y = mary
X = mary, Y = john

La description des schémas au moyen de clauses permet de les interroger, comme si il s'agissait de données. Ainsi la question :

Quelles sont les sous-classes de personne *?*
peut être formulée par le but :

?– X < personne.

Les trois réponses sont :

X = femme
X = mère
X = homme

Ce type de formulation homogène permet d'interroger simultanément le schéma et le contenu proprement dit de la base. Par exemple, le but :
Quelles sont les sous-classes de personne *dont* mary *est une instance?*
peut s'écrire sous la forme :

?– X < personne & mary:X.

Les différentes réponses sont alors :

X = femme
X = mère

Enfin, la capacité de déduction de nouvelles propriétés pour les objets, peut être mise à profit pour restructurer les informations. Ainsi la clause :

X.parent → Y :– Y.enfant → X.

permet de déduire un attribut parent inverse de enfant. Cette forme de restructuration par déduction peut aussi être employée au niveau des classes. Par exemple, la clause :

C < ecologique :– C < animal & C < moyen-de-transport.

permet de déduire de nouveaux liens d'héritage, à partir d'un schéma existant.

5.5 Famille

Nous plaçons dans cette famille les langages dans lesquels la structure des objets est utilisée comme un ensemble de données connues ou déduites. Ces langages sont pour la plupart issus de travaux menés dans la communauté des bases de données. Comme pour les autres familles, notre ambition n'est pas de citer tous les travaux mais plutôt d'indiquer des repères essentiels.

Dans cette famille, nous pouvons isoler deux tendances principales :

1. *conception de langages pour bases de données déductives orienté-objet.* Nous rencontrons ici, notamment, deux représentants majeurs : COL [3] et LDL [11], ainsi que leurs extensions (par exemple [2] et [56]). Un des aspects étudiés de façon plus particulière dans ces travaux, et qui n'est pas présent dans le langage que nous avons décrit, est la construction d'ensembles d'objets, et l'utilisation de termes dénotant de tels ensembles.

2. *construction de cadres logiques pour la structure des objets.* Le représentant le plus élaboré de cette branche est sans doute actuellement la F-logique [36] [37], qui a déjà été réutilisée comme support pour la définition de différents langages[15]. Citons par exemple le langage de requêtes pour bases de données orienté-objet XSQL [35], et le langage graphique DOODLE [23].

15. Notons que les clauses du langage que nous avons présenté, peuvent être vues simplement comme une forme restreinte de clauses de la F-logique.

6 Conclusion

Nous avons identifié trois familles de travaux, utilisant de façons différentes la structure des objets dans des raisonnements associés à des langages de clauses. Pour chaque famille nous avons expliqué les motivations sous-jacentes aux différents travaux, puis nous avons décrit un langage typique pour lequel nous avons donné une sémantique déclarative et une sémantique opérationnelle. Nous avons ensuite illustré l'utilisation de la structure des objets dans des raisonnements simples. Enfin nous avons indiqué les points de repères bibliographiques importants et souligné les principaux axes de recherche. Ces résultats sont présentés sous une forme synthétique dans la table de la figure 1.

	1	2	3
Travaux typiques	LOGIN [5]	L&O [43]	COL [3], F-logique [37]
Utilisations de la structure des objets	Hypothèses et contraintes	Sélecteurs de théories	Informations connues et déduites
Calculs de ces structures	Oui par abduction	Structures non calculées	Oui par déduction
Spécificités opération-nelles	Résolution type CLP	Règle de choix des clauses	Nouvelles règles d'inférences
Domaines	Prog. logique et IA	Prog. logique	BD déductives

FIG. 1 - *Trois familles de langages*

Les structures des objets sont utilisées de façons différentes dans chacune des trois familles. Dans la première, ces structures sont des contraintes por-

tant sur les termes. Lorsqu'elles sont fournies comme réponse à un but, elles ont le statut d'hypothèses devant être satisfaites pour que ce but soit une conséquence des clauses du programme. La seconde famille fait intervenir ces structures pour décrire les contextes dans lesquels les clauses doivent être utilisées, ainsi que les contextes dans lesquels les atomes d'un but doivent être prouvés. Enfin, dans la troisième famille, la structure des objets est décrite sous la forme de liens entre les objets, ces liens étant, eux, employés en tant qu'atomes. On s'intéresse alors à leurs déductibilités à partir de l'ensemble des clauses.

De façon schématique, on peut dire que la première famille est celle qui concerne la communauté de recherche la plus diversifiée. En effet, elle regroupe des travaux allant de la programmation logique à la représentation de connaissances en intelligence artificielle. Dans le domaine de la programmation logique, l'intérêt de tels langages est de permettre une description plus aisée des structures de données, ainsi que leurs manipulations de façon plus efficace par l'intermédiaire d'algorithmes dédiés. En ce qui concerne la représentation de connaissances, les langages de cette famille sont utilisés pour leurs deux niveaux de descriptions : les contraintes pour représenter la connaissance d'un domaine d'application et les clauses pour décrire les méthodes de résolution du problème dans ce domaine.

La seconde famille de travaux concerne, elle, de façon plus spécifique la communauté de recherche en programmation logique modulaire. En effet, les langages qui en sont issus permettent tout d'abord de regrouper les clauses applicables dans un même contexte pour former des modules, et ensuite d'utiliser la notion de contexte plus spécifique pour spécialiser ces modules.

Enfin la troisième famille est quant à elle fortement ancrée dans le domaine des base de données déductives. Les travaux qu'elle regroupe ont été développés pour permettre l'utilisation de la structure des objets comme informations connues et comme informations déduites, et ainsi notamment autoriser des descriptions et des interrogations plus naturelles.

Il est important de noter que les spécificités opérationnelles de chaque famille concernent différents aspects des procédures de calcul sur les clauses. En effet, elles résident respectivement :

- au niveau de l'unification qui est remplacée, dans la première famille, par la construction d'une conjonction de contraintes devant être satisfaisable.
- au niveau de la règle de choix des clauses, pour la seconde famille, en faisant intervenir une notion de contexte.
- et, dans la troisième famille, au niveau de nouvelles règles d'inférences qui permettent de prendre en compte des prédicats spécifiques décrivant les liens entre les objets.

Sur le plan opérationnel, ces trois niveaux sont relativement indépendants. Ceci permet de penser que la combinaison des mécanismes de calcul mis en jeux est possible. La partie suivante décrit une construction de ce type, sous la forme d'un langage intégrant des caractéristiques de la première et de la troisième des familles.

Deuxième partie

Outil

1 Introduction

Dans la partie 1, nous avons montré que la structure des objets a été utilisée de trois façons différentes dans les raisonnements portant sur des clauses :

1. comme des hypothèses devant être satisfaites pour garantir la validité d'une déduction.

2. en tant que contexte permettant de sélectionner les ensembles de clauses à utiliser lors d'un raisonnement.

3. sous la forme de données utilisées comme faits connus ou déduits.

Des travaux exploratoires, dans le domaine des bases de données, nous ont permis de montrer qu'une combinaison des approches 1 et 3 permettait de détecter certaines des requêtes ne pouvant pas avoir de réponses. Cette technique basée sur l'exploitation des contraintes d'intégrité, a été présentée dans [51]. Nous verrons dans la partie 3 comment une telle combinaison peut être complétée et mise à profit dans le cadre de l'optimisation sémantique de requêtes pour les BDOO déductives.

Nous présentons dans cette partie le formalisme supportant une combinaison des approches 1 et 3, que nous utiliserons dans la partie 3. Il permet de décrire des liens entre objets (instanciations, héritages, valorisations d'attributs), des contraintes sur ces objets (types, cardinalités, disjonctions), et il autorise une forme de raisonnement intégrant déduction et abduction de la structure des objets.

Ce formalisme est un langage de clauses avec contraintes au sens de la programmation logique avec contraintes [31]. Sa sémantique a été inspirée par la F-logique [37], les logiques terminologiques [46] et la généralisation de la programmation logique avec contraintes proposée par Höhfeld et Smolka [30].

Dans la section 2, à partir d'un exemple d'école sur la recherche de chemins dans un graphe, nous allons construire par étapes, et informellement, la sémantique déclarative et la sémantique opérationnelle de notre langage. Syntaxes et sémantiques sont ensuite détaillées de façon formelle dans les sections 3, 4 et 5. Puis nous établissons la validité (section 6) et la complétude (section 7) de la réduction utilisée pour calculer les réponses.

2 Présentation du langage

2.1 Faits connus et déduits

Nous débutons notre construction à partir d'un langage de clauses de la troisième des familles de la partie 1. C'est donc un langage dans lequel la structure des objets est exprimée au moyen de liens d'instanciation, d'héritage, et de valorisation d'attribut.

Nous utilisons, dans un premier temps, simplement la sémantique déclarative et la sémantique opérationnelle décrites respectivement dans les sections 5.2 et 5.3 de la partie 1. Dans les raisonnements, la stucture des objets correspond alors à des faits connus et à des faits déduits.

Nous illustrons l'emploi de ce langage en décrivant des graphes et des chemins.

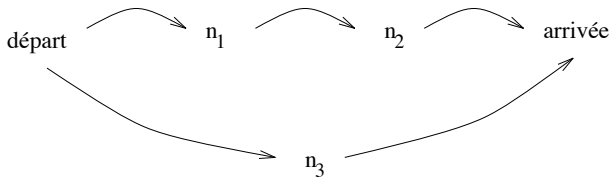

FIG. 2 - *Graphe sans cycle.*

Un arc d'un sommet n_i vers un sommet n_j est représenté par le fait :

$$\mathsf{n}_i\text{.passage-vers} \rightarrow \mathsf{n}_j.$$

Soient les arcs du graphe de la figure 2 :

départ.passage-vers $\rightarrow$ n_1.

n_1.passage-vers $\rightarrow$ n_2.

n_2.passage-vers $\rightarrow$ arrivée.

départ.passage-vers $\rightarrow$ n_3.

n_3.passage-vers $\rightarrow$ arrivée.

Nous notons $\mathcal{G}$ l'ensemble de faits clos correspondant.

Nous définissons alors les sommets à partir desquels on peut atteindre l'arrivée. Un tel sommet est décrit comme étant une valeur d'un attribut passe-par d'un objet chemin.

De l'arrivée on peut atteindre l'arrivée.

chemin.passe-par $\rightarrow$ arrivée.

Si il y a un arc de X *vers* Y*, et si de* Y *on peut atteindre l'arrivée, alors de* X *aussi on peut atteindre l'arrivée.*

chemin.passe-par → X :– X.passage-vers → Y & chemin.passe-par → Y.

Ce programme ne construit pas vraiment des chemins. Il détermine cependant l'ensemble des sommets à partir desquels on peut joindre l'arrivée.

Si nous considérons qu'un modèle est décrit par un ensemble de faits clos, nous avons comme unique modèle minimal :
{ chemin.passe-par → départ, chemin.passe-par → n_1, chemin.passe-par → n_2, chemin.passe-par → n_3, chemin.passe-par → arrivée } ∪ $\mathcal{G}$.

Voici deux exemples de buts :

Peut-on déduire que du sommet départ *on peut atteindre le sommet* arrivée *?*

?– chemin.passe-par → départ.

Réponse : oui.

Pour quels sommets X *peut-on déduire que l'on peut atteindre l'arrivée en partant de* X *?*

?– chemin.passe-par → X.

Réponses différentes :
{ X = départ }, { X = n_1 }, { X = n_2 }, { X = n_3 }, { X = arrivée }

Ce premier programme autorise les chemins non élémentaires[16]. Dans la section suivante nous modifierons notre exemple et interdirons ce type de chemin.

2.2 Contraintes et hypothèses

Nous poursuivons notre construction en ajoutant au langage les contraintes sur objets présentées dans la section 3 de la partie 1. Les clauses sont maintenant de la forme :

Tête :– conjonction de liens // conjonction de contraintes.

Une telle clause signifie que si la conjonction de contraintes est satisfaite et si la conjonction de liens est vraie alors la tête de la clause doit aussi être vraie.

En ce qui concerne la machine abstraite, il faut la modifier de façon à collecter les contraintes rencontrées dans le corps des clauses. Seules les

16. Un chemin *élémentaire* est un chemin tel qu'en le parcourant, on ne rencontre pas deux fois le même sommet.

transitions pour lesquelles l'ensemble des contraintes accumulées reste satisfaisable sont autorisées.

Complétons maintenant notre exemple, afin d'interdire les chemins non élémentaires. La propriété que nous utilisons est :
Un chemin C est élémentaire si chacun des sommets de C est l'extrémité terminale d'au plus un arc de C, et C contient un sommet qui n'est l'extrémité terminale d'aucun arc de C (ce sommet est l'extrémité initiale de C).

Aux faits décrivant le graphe précédent nous ajoutons :

arrivée.passage-vers $\rightarrow$ n_2.

arrivée.passage-vers $\rightarrow$ départ.

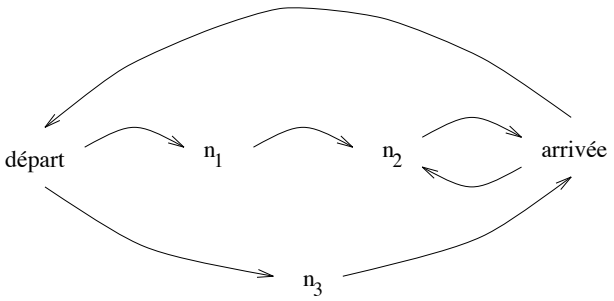

FIG. 3 - *Graphe avec cycles.*

Le graphe correspondant est celui de la figure 3. Nous notons toujours $\mathcal{G}$ le nouvel ensemble de faits clos correspondant.

Le reste du programme est modifié de la façon suivante :
Un sommet d'un chemin est l'extrémité terminale d'au plus un arc de ce chemin. Un arc X.passage-vers $\rightarrow$ Y *n'est donc utilisé dans un chemin que si* Y *peut avoir pour prédécesseur immédiat* X*, sachant que* Y *a au plus un prédécesseur immédiat.*

chemin.passe-par $\rightarrow$ arrivée.

chemin.passe-par $\rightarrow$ X :– X.passage-vers $\rightarrow$ Y & chemin.passe-par $\rightarrow$ Y
// Y.précédé-par $\rightarrow$ X & Y:at-most(1,précédé-par).

L'extrémité initiale d'un chemin est un sommet sans prédécesseur. Peut-on déduire que départ *est effectivement l'extrémité initiale d'un chemin menant à* arrivée *?*

?– chemin.passe-par $\rightarrow$ départ // départ:at-most(0,précédé-par).

Réponse : oui.

Une interprétation pour le langage que nous venons de construire est constituée de deux parties $\mathcal{I}$ et $\mathcal{B}$ que nous appelons semi-interprétations. La semi-interprétation $\mathcal{I}$ est utilisée pour interpréter les liens entre objets et $\mathcal{B}$ pour interpréter les contraintes.

Voici, pour illustrer cette notion, trois exemples d'interprétations qui sont des modèles de notre programme :

1. $\mathcal{I}$ = { chemin.passe-par → départ, chemin.passe-par → n_1, chemin.passe-par → n_2, chemin.passe-par → arrivée } ∪ $\mathcal{G}$.
   $\mathcal{B}$ = { arrivée.précédé-par → n_2, n_2.précédé-par → n_1, n_1.précédé-par → départ }.
2. $\mathcal{I}$ = { chemin.passe-par → départ, chemin.passe-par → n_3, chemin.passe-par → arrivée } ∪ $\mathcal{G}$.
   $\mathcal{B} = \emptyset$.
3. $\mathcal{I}$ = { chemin.passe-par → départ, chemin.passe-par → n_3, chemin.passe-par → arrivée } ∪ $\mathcal{G}$.
   $\mathcal{B}$ = { arrivée.précédé-par → n_3, n_3.précédé-par → départ, départ:at-most(0,précédé-par) }.

Intéressons nous aux modifications induites par cette nouvelle forme d'interprétation sur les réponses correctes. Pour cela, considérons le but :
Quels sont les sommets pour lesquels on peut déduire qu'ils sont l'extrémité initiale d'un chemin menant à arrivée.
?– chemin.passe-par → X // X:at-most(0,précédé-par).

{ X = n_3 } était une réponse correcte pour le but ?– chemin.passe-par → X dans l'exemple précédent (section 2.1). Mais à présent { X = n_3 }[17] n'est pas une réponse correcte pour le but ?– chemin.passe-par → X // X:at-most(0,précédé-par). En effet chemin.passe-par → n_3 n'est pas vrai dans tous les modèles du nouveau programme. chemin.passe-par → n_3 est faux, par exemple, dans le modèle 1 donné ci-dessus, et ne peut donc plus être une conséquence logique du programme.

En revanche, considérons la conjonction suivante :
H_1 = (arrivée.précédé-par → X & arrivée:at-most(1,précédé-par)
& X:at-most(0,précédé-par) & X = n_3)
H_1, vue comme une hypothèse, est une réponse correcte. En effet pour tous les modèles du programme lorsque l'hypothèse H_1 est satisfaite, le but chemin.passe-par → X // X:at-most(0,précédé-par) est aussi satisfait.

17. Ou toute autre réponse de l'exemple précédent.

Nous dirons que ce but est la conséquence logique du programme sous l'hypothèse H_1.

Une autre réponse correcte, pour ce même but, est
H_2 = (arrivée.précédé-par → X & arrivée:at-most(1,précédé-par)
& X:at-most(0,précédé-par) & X = n_3 & n_2.précédé-par → n_1).

Cependant sur le plan opérationnel, nous voulons calculer les réponses les plus générales possibles. Une réponse calculée sera donc plutôt H_1 que H_2, car H_1 est une hypothèse "moins forte" que H_2.

Une autre réponse calculée (la plus générale) sera par exemple
H_3 = (arrivée.précédé-par → n_3 & arrivée:at-most(1,précédé-par)
& n_3.précédé-par → X & n_3:at-most(1,précédé-par)
& X:at-most(0,précédé-par) & X = départ).

Les réponses calculées sont obtenues de la façon suivante : une réponse calculée correspond à l'ensemble des contraintes accumulées pour atteindre un état final de la machine abstraite. De façon à préserver la lisibilité des exemples, comme dans la partie 1, nous écrivons de préférence les réponses sous une forme simplifiée.

Le calcul de H_1 et de H_3 prouve l'existence de chemins, et il est intéressant de noter que les contenus de H_1 et de H_3 décrivent ces chemins.

2.3 Contraintes et faits déduits

Nous allons reformuler l'exemple précédent, en employant des contraintes pour restreindre directement les liens pouvant être déduits entre les objets.

Nous allons ainsi imposer que lors de la construction d'un chemin, il ne soit possible de déduire du programme qu'un seul arc sortant (passage-vers) pour chaque sommet.

Puisque nous considérons les arcs sortants, la propriété que nous utilisons est maintenant :
Un chemin C est élémentaire si chacun des sommets de C est l'extrémité initiale d'au plus un arc de C, et C contient un sommet qui n'est l'extrémité initiale d'aucun arc de C (ce sommet est l'extrémité terminale de C).

Nous conservons les faits du programme précédent (section 2.2) auxquels nous ajoutons :

chemin.passe-par → départ.

chemin.passe-par → X :– Y.passage-vers → X & chemin.passe-par → Y
// Y:at-most(1,passage-vers).

L'extrémité terminale d'un chemin est un sommet qui n'est l'extrémité initiale d'aucun arc de ce chemin. Peut-on déduire que arrivée *est effectivement l'extrémité terminale d'un chemin débutant par* départ *?*

?– chemin.passe-par $\rightarrow$ arrivée // arrivée:at-most(0,passage-vers).

Réponse (souhaitée) :
L'hypothèse sous laquelle le but est conséquence logique du programme.

Sur le plan opérationnel, une telle réponse est obtenue si nous modifions légèrement la sémantique opérationnelle précédente (section 2.2). En effet, il suffit lors de chaque transition de la machine abstraite, *d'ajouter la tête de la clause utilisée aux contraintes collectées*, et de tester la satisfaisabilité de cet ensemble. A présent, nous utiliserons la sémantique opérationnelle ainsi modifiée.

D'un point de vue déclaratif, considérons les deux semi-interprétations constituant une interprétation : $\mathcal{I}$ pour les liens et $\mathcal{B}$ pour les contraintes. Un atome A décrivant un lien entre objets, ne peut être vrai dans $\mathcal{I}$ que si A (vu comme une contrainte) est satisfait dans $\mathcal{B}$. Ceci nous conduit à imposer que toute interprétation $\langle \mathcal{I}, \mathcal{B} \rangle$ soit telle que $\mathcal{I} \subseteq \mathcal{B}$.

Mais cette modification seule ne suffit pas. En effet, nous allons donner un exemple de réponse désirée, qui est bien une réponse calculée par la sémantique opérationnelle, mais qui n'est pas une réponse correcte pour la nouvelle sémantique déclarative.

Soit le but :

?– chemin.passe-par $\rightarrow$ arrivée // arrivée:at-most(0,passage-vers).

une des réponses calculées est :

$H =$ (départ.passage-vers $\rightarrow$ n_3 & n_3.passage-vers $\rightarrow$ arrivée
& chemin.passe-par $\rightarrow$ départ & chemin.passe-par $\rightarrow$ n_3
& chemin.passe-par $\rightarrow$ arrivée & départ:at-most(1,passage-vers)
& n_3:at-most(1,passage-vers) & arrivée:at-most(0,passage-vers))

Or le programme contient les deux faits suivants :

départ.passage-vers $\rightarrow$ n_1.

départ.passage-vers $\rightarrow$ n_3.

Ceci signifie que les interprétations $\langle \mathcal{I}, \mathcal{B} \rangle$ qui sont des modèles du programme vérifient :
{départ.passage-vers $\rightarrow$ n_1, départ.passage-vers $\rightarrow$ n_3 } $\subseteq \mathcal{I}$.
Et puisque $\mathcal{I} \subseteq \mathcal{B}$, nous avons alors pour tous les modèles :
{départ.passage-vers $\rightarrow$ n_1, départ.passage-vers $\rightarrow$ n_3 } $\subseteq \mathcal{B}$.

Puisque l'hypothèse H contient la contrainte départ:at-most(1,passage-vers), H ne peut jamais être satisfaite par un modèle du programme.

Comme il apparait raisonnable de ne considérer que les hypothèses-réponses pouvant être satisfaites par au moins un modèle du programme, H ne peut pas être acceptée en tant que réponse correcte.

Cependant, H est une réponse désirée que nous savons calculer. Nous modifions donc la sémantique déclarative afin d'accepter de telles réponses comme étant correctes.

En plus de la restriction $\mathcal{I} \subseteq \mathcal{B}$ placée sur toutes les interprétations $\langle \mathcal{I}, \mathcal{B} \rangle$, nous allons "affaiblir" la sémantique déclarative des clauses. Le sens d'une clause sans variable telle que :

Tête :– conjonction de liens // conjonction de contraintes.

devient maintenant :

Soit $\langle \mathcal{I}, \mathcal{B} \rangle$ une interprétation. La clause est satisfaite[18] dans cette interprétation *ssi* la tête de la clause est vraie dans $\mathcal{I}$ *ou* la conjonction de contraintes n'est pas satisfaite dans $\mathcal{B}$ *ou* la conjonction de liens n'est pas vraie dans $\mathcal{I}$ *ou* la tête de la clause n'est pas satisfaite dans $\mathcal{B}$.

Ainsi, lorsque la tête vue comme une contrainte n'est pas satisfaite, la clause est quant à elle forcément satisfaite et donc d'une certaine façon "neutralisée".

Nous avons alors des modèles de notre programme qui satisfont H, comme par exemple $\langle \mathcal{I}, \mathcal{B} \rangle$ avec :

$\mathcal{I}$= { départ.passage-vers → n_3, n_3.passage-vers → arrivée, chemin.passe-par → départ, chemin.passe-par → n_3, chemin.passe-par → arrivée }

$\mathcal{B}$= { départ.passage-vers → n_3, n_3.passage-vers → arrivée, chemin.passe-par → départ, chemin.passe-par → n_3, chemin.passe-par → arrivée, départ:at-most(1,passage-vers), n_3:at-most(1,passage-vers), arrivée:at-most(0,passage-vers) }

Ainsi H est bien une réponse correcte.

2.4 Réduction de l'espace de recherche

Dans le but de fournir une vision plus précise du comportement opérationnel attendu, nous situons les réductions de l'espace de recherche permises par rapport à celles rencontrées en programmation logique avec contraintes.

Reconsidérons les faits décrivant le graphe, et les clauses :

chemin.passe-par → départ.

chemin.passe-par → X :– Y.passage-vers → X & chemin.passe-par → Y
// Y:at-most(1,passage-vers).

18. Dans cette introduction informelle nous ne considérons que le cas sans variable.

Soit le but :

Quels sont les sommets Z *situés sur un chemin dont l'extrémité terminale est* arrivée*, et tels que ces* Z *soient des prédécesseurs immédiats de* arrivée *?*

?– chemin.passe-par → arrivée & chemin.passe-par → Z &
Z.passage-vers → arrivée // arrivée:at-most(0,passage-vers).

En terme de construction de solutions, l'effacement (de gauche à droite) de ce but se lit comme suit :

1. chemin.passe-par → arrivée génère tous les chemins C de départ à arrivée.

2. chemin.passe-par → Z génère pour chaque C différentes valeurs pour Z.

   Sans contraintes, pour chaque C tous les Z situés sur un chemin dont l'extrémité initiale est départ seraient générés.

   Ici, des contraintes Y:at-most(1,passage-vers) ont été placées sur les liens Y.passage-vers → X lors de la génération de chaque chemin C. Ainsi pour chaque C seuls les Z situés sur C sont générés par chemin.passe-par → Z.

3. Enfin Z.passage-vers → arrivée ne conserve que les Z liés par un arc à arrivée.

La réduction de l'espace de recherche obtenue dans l'étape 2 est proche de celles permises par les langages de programmation logique avec contraintes de la famille CLP(X) [31].

Apportons à présent une modification apparemment légère au but précédent, et considérons l'espace de recherche pour :

?– chemin.passe-par → arrivée & chemin.passe-par → Z
// Z.passage-vers → arrivée & arrivée:at-most(0,passage-vers).

Dans ce nouveau but, l'information Z.passage-vers → arrivée, qui n'était utilisée qu'à la fin de la génération, va être prise en compte dès le début de la construction des solutions, sous la forme d'une contrainte devant être satisfaite.

Les solutions sont alors générées de la façon suivante :

1. Pas de changement, chemin.passe-par → arrivée génère tous les chemins C de départ à arrivée.

2. Cette fois, chemin.passe-par → Z génère pour chaque C un seul Z. En effet, pour un chemin C donné, tous les sommets Y situés sur C sont tels que Y:at-most(1,passage-vers), et il n'existe alors qu'un seul Z sur C tel que les contraintes Z.passage-vers → arrivée et Z:at-most(1,passage-vers) puissent être satisfaites.

La réduction de l'espace de recherche obtenue dans l'étape 2 est du type de celle permise par les contraintes de propagation en programmation logique [39]. Elle consiste à utiliser des atomes devant être effacés (dans un but) comme si il s'agissait de contraintes.

En règle générale, ces atomes vus comme des contraintes permettent de réduire l'espace de recherche mais doivent tout de même être effacés à leur tour. L'exemple décrit ici est un cas particulier dans lequel l'effacement de Z.passage-vers $\rightarrow$ arrivée n'est même plus nécessaire.

3 Syntaxe

Nous allons à présent décrire de façon formelle la syntaxe, puis la sémantique déclarative (section 4), ainsi qu'une sémantique opérationnelle associée (section 5).

Soient $\mathcal{K}$ un ensemble décidable de symboles de constantes et $\mathcal{V}$ un ensemble infini décidable de symboles de variables, tels que $\mathcal{K} \cap \mathcal{V} = \emptyset$. Les **termes** sont de deux sortes: des **noms d'objets** (noms de classes, d'instances ou d'attributs) et des **concepts** (ensembles regroupant des objets ayant certaines propriétés communes).

La syntaxe de formation des termes est la suivante (v dénote une variable, k une constante, X et R des noms d'objets, C et C_i (pour $i = 1, \ldots, m$) dénotent des concepts et n un entier naturel) :

$$
\begin{array}{lcl}
X & \xrightarrow{def} & v \mid k \\
C & \xrightarrow{def} & X \mid \mathit{anything} \mid \mathit{nothing} \mid \mathit{and}(C_1, \ldots, C_m) \mid \\
 & & \mathit{all}(R, C) \mid \mathit{at\text{-}most}(n, R) \mid \mathit{not}(X)
\end{array}
$$

Les **atomes** sont également de deux sortes: les **liens atomiques** et les **contraintes atomiques.** La syntaxe de formation des atomes est la suivante (X, Y et R dénotent des noms d'objets, C dénote un concept, Al est un lien atomique et Ac une contrainte atomique):

$$
\begin{array}{lcl}
Al & \xrightarrow{def} & X < Y \mid X : Y \mid X.R \rightarrow Y \\
Ac & \xrightarrow{def} & X < Y \mid X : C \mid X.R \rightarrow Y \mid X \sqsubseteq C \mid X = Y
\end{array}
$$

On remarquera qu'un lien atomique est aussi une contrainte atomique.

Les syntaxes de formation des **buts** et des **clauses** sont les suivantes :

$$
\begin{aligned}
F &\xrightarrow{def} \epsilon \mid Al \mid Al_1 \ \& \ \ldots \ \& \ Al_n \\
H &\xrightarrow{def} \epsilon \mid Ac \mid Ac_1 \ \& \ \ldots \ \& \ Ac_m \\
But &\xrightarrow{def} F \ /\!/ \ H \\
Clause &\xrightarrow{def} Al :\!- \ F \ /\!/ \ H
\end{aligned}
$$

F est une **conjonction de liens atomiques**, et H est une **conjonction de contraintes atomiques.** ϵ est une **conjonction vide.** Pour une clause $Al :\!- \ F \ /\!/ \ H$ nous utilisons les appellations habituelles de **corps** pour $F \ /\!/ \ H$ et **tête** pour Al.

Un **langage $\mathcal{L}$** est l'ensemble des buts et des clauses construits à partir de deux ensembles de symboles $\mathcal{K}$ et $\mathcal{V}$.

Conventions de notation : les symboles de variables débutent par une majuscule et les symboles de constantes par une minuscule. De plus, dans les exemples, nous simplifions l'écriture des buts et des clauses en omettant les ϵ ainsi que les $/\!/ \ \epsilon$, et en remplaçant les clauses $Al :\!- \ \epsilon \ /\!/ \ \epsilon$ par Al.

4 Sémantique déclarative

Soit un langage $\mathcal{L}$ construit à partir de l'ensemble de variables $\mathcal{V}$ et de l'ensemble de constantes $\mathcal{K}$.

Nous utilisons deux niveaux d'interprétation : les **semi-interprétations** pour les conjonctions de liens atomiques et les conjonctions de contraintes atomiques, et les **interprétations** pour les buts et les clauses.

Une semi-interprétation est de la forme $\mathcal{J} = \langle D, id, membre, herite, attribut \rangle$ avec :

- D un ensemble non vide appelé domaine de $\mathcal{J}$.
- id une fonction injective totale $id : \mathcal{K} \rightarrow D$, associant à chaque constante un élément du domaine.
- $membre \subseteq D \times D$, une relation liant les instances à leurs classes.
- $herite \subseteq D \times D$, une relation d'ordre strict liant les classes à leurs super-classes.
- $attribut \subseteq D \times D \times D$, une relation dont chaque occurrence lie une instance, un nom d'attribut et une valeur pour cet attribut.

– *membre* et *herite* vérifiant:
$\forall x_1, x_2, x_3 \in D, \langle x_1, x_2\rangle \in membre \wedge \langle x_2, x_3\rangle \in herite \Rightarrow \langle x_1, x_3\rangle \in membre$

(i.e., les instances d'une classe sont aussi des instances de ses superclasses.)

La définition du tuple $\langle D, id, membre, herite, attribut\rangle$ est inspirée de la sémantique déclarative de la F-logique [37].

Soit $\mathcal{J}$ une semi-interprétation. Une **$\mathcal{J}$-assignation** α est une fonction totale $\alpha : \mathcal{V} \to D$, qui permet d'associer aux variables du langage des éléments du domaine de $\mathcal{J}$. Nous notons $ASG^{\mathcal{J}}$ l'ensemble des $\mathcal{J}$-assignations. Toutes les $\mathcal{J}$-assignations $\alpha \in ASG^{\mathcal{J}}$ sont étendues à $\mathcal{V} \cup \mathcal{K}$, c'est-à-dire à l'ensemble des noms d'objets, par *id*: $\forall k \in \mathcal{K}, \alpha(k) = id(k)$.

Pour une semi-interprétation $\mathcal{J} = \langle D, id, membre, herite, attribut\rangle$ et une $\mathcal{J}$-assignation α, nous définissons une fonction $\boldsymbol{ext_{\mathcal{J}\alpha}}$ qui associe à chaque concept un ensemble d'éléments du domaine. Cette fonction est définie par les équations suivantes (avec X et R des noms d'objets, C et C_i (pour $i = 1, \ldots, m$) des concepts et n un entier naturel) :

$$\begin{aligned}
ext_{\mathcal{J}\alpha}(X) &= \{d \in D \mid \langle d, \alpha(X)\rangle \in membre\} \\
ext_{\mathcal{J}\alpha}(anything) &= D \\
ext_{\mathcal{J}\alpha}(nothing) &= \emptyset \\
ext_{\mathcal{J}\alpha}(and(C_1, \ldots, C_m)) &= ext_{\mathcal{J}\alpha}(C_1) \cap \ldots \cap ext_{\mathcal{J}\alpha}(C_m) \\
ext_{\mathcal{J}\alpha}(all(R, C)) &= \{d_1 \in D \mid \forall d_2 \in D, \\
&\qquad \langle d_1, \alpha(R), d_2\rangle \in attribut \Rightarrow d_2 \in ext_{\mathcal{J}\alpha}(C)\} \\
ext_{\mathcal{J}\alpha}(at\text{-}most(n, R)) &= \{d_1 \in D \mid card(\{d_2 \mid \langle d_1, \alpha(R), d_2\rangle \in attribut\}) \leq n\} \\
ext_{\mathcal{J}\alpha}(not(X)) &= \{d \in D \mid d \notin ext_{\mathcal{J}\alpha}(X)\}
\end{aligned}$$

$ext_{\mathcal{J}\alpha}$ correspond à l'interprétation des concepts dans les logiques terminologiques [46].

La fonction $ext_{\mathcal{J}\alpha}$ nous permet de définir la notion d'ensemble de **$\mathcal{J}$-solutions** [30] d'un atome et d'une conjonction d'atomes. L'ensemble des $\mathcal{J}$-solutions d'un atome A sera noté $\boldsymbol{[A]^{\mathcal{J}}}$.

Pour une semi-interprétation $\mathcal{J}$ donnée, les ensembles de $\mathcal{J}$-solutions des différentes formes d'atomes sont :

$$\begin{aligned}
[X < Y]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \langle \alpha(X), \alpha(Y)\rangle \in herite\} \\
[X \sqsubseteq C]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid ext_{\mathcal{J}\alpha}(X) \subseteq ext_{\mathcal{J}\alpha}(C)\} \\
[X : C]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \alpha(X) \in ext_{\mathcal{J}\alpha}(C)\}
\end{aligned}$$

$$\begin{aligned}
[X.R \to Y]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \langle \alpha(X), \alpha(R), \alpha(Y)\rangle \in \mathit{attribut}\} \\
[X = Y]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \alpha(X) =_D \alpha(Y)\}\ ^{19} \\
[\epsilon]^{\mathcal{J}} &:= ASG^{\mathcal{J}}
\end{aligned}$$

La notion d'ensemble de $\mathcal{J}$-solutions est étendue aux conjonctions d'atomes :

$$[A_1 \ \& \ \ldots \ \& \ A_n]^{\mathcal{J}} := [A_1]^{\mathcal{J}} \cap \ldots \cap [A_n]^{\mathcal{J}}.$$

Les semi-interprétations d'un même langage $\mathcal{L}$ sont partiellement ordonnées de la façon suivante :

$$\mathcal{J}_1 \leq \mathcal{J}_2 \Leftrightarrow \text{pour tout atome } A \text{ de } \mathcal{L}, [A]^{\mathcal{J}_1} \subseteq [A]^{\mathcal{J}_2}.$$

Une **interprétation** Γ de $\mathcal{L}$ est de la forme $\Gamma = \langle \mathcal{I}, \mathcal{B}\rangle$ avec $\mathcal{I}$ et $\mathcal{B}$ des semi-interprétations de $\mathcal{L}$ telles que : $\mathcal{I} \leq \mathcal{B}$. La semi-interprétation $\mathcal{B}$ est appelée la base de Γ. L'ensemble des Γ-assignations est $ASG^{\Gamma} = ASG^{\mathcal{B}}$.

Nous définissons une relation d'ordre partiel entre les interprétations : soient deux interprétations $\Gamma_1 = \langle \mathcal{I}_1, \mathcal{B}_1\rangle$ et $\Gamma_2 = \langle \mathcal{I}_2, \mathcal{B}_2\rangle$,

$$\Gamma_1 \leq \Gamma_2 \Leftrightarrow \mathcal{I}_1 \leq \mathcal{I}_2 \wedge \mathcal{B}_1 = \mathcal{B}_2.$$

Nous pouvons à présent étendre la notion de solution aux buts et aux clauses. Soit une interprétation $\Gamma = \langle \mathcal{I}, \mathcal{B}\rangle$, nous définissons les ensembles de Γ-solutions suivants :

$$\begin{aligned}
[F \ /\!/ \ H]^{\Gamma} &:= [F]^{\mathcal{I}} \cap [H]^{\mathcal{B}} \\
[A \text{ :- } F \ /\!/ \ H]^{\Gamma} &:= (ASG^{\Gamma} - [F \ /\!/ \ H \ \& \ A]^{\Gamma}) \cup [A]^{\mathcal{I}}
\end{aligned}$$

Une conjonction de contraintes atomiques H est **satisfaisable** *ssi* il existe au moins une semi-interprétation $\mathcal{J}$ telle que $[H]^{\mathcal{J}} \neq \emptyset$.

Une interprétation Γ est un **modèle** d'un ensemble de clauses S *ssi* $\forall C \in S, [C]^{\Gamma} = ASG^{\Gamma}$.

Soient S un ensemble de clauses, H une conjonction de contraintes atomiques et G un but. S a pour **conséquence logique** G **sous hypothèse** H (noté $S \models H \Rightarrow G$) *ssi* pour tout modèle $\Gamma = \langle \mathcal{I}, \mathcal{B}\rangle$ de S, nous avons $[H]^{\mathcal{B}} \subseteq [G]^{\Gamma}$.

Nous pouvons maintenant définir la notion de réponse correcte. Une **réponse** est une conjonction de contraintes atomiques. Une **réponse correcte** au but G pour S est une conjonction de contraintes atomiques H, telle que H soit satisfaisable et $S \models H \Rightarrow G$.

19. $=_D$ dénote l'égalité dans le domaine D.

5 Sémantique opérationnelle

Nous formalisons ici la résolution utilisée en définissant la relation de réduction des buts [30].

Pour la suite, nous notons A les liens atomiques, F les conjonctions de liens atomiques, H les conjonctions de contraintes atomiques, G les buts, et C les clauses.

5.1 Notions préliminaires

Soit *Formule* une conjonction d'atomes, un but ou une clause. Nous notons $\boldsymbol{var(Formule)}$ l'ensemble des variables de *Formule*.

Deux liens atomiques A_1 et A_2 sont dits **compatibles** *ssi* :

(1) $A_1 = (X_1 : Y_1)$ et $A_2 = (X_2 : Y_2)$.

ou (2) $A_1 = (X_1 < Y_1)$ et $A_2 = (X_2 < Y_2)$.

ou (3) $A_1 = (X_1.R_1 \rightarrow Y_1)$ et $A_2 = (X_2.R_2 \rightarrow Y_2)$.

Dans les cas (1) et (2) nous noterons $\boldsymbol{(A_1 \overset{arg}{=} A_2)}$ la conjonction de contraintes $(X_1 = X_2)$ & $(Y_1 = Y_2)$. Dans le cas (3), la notation $(A_1 \overset{arg}{=} A_2)$ représentera la conjonction de contraintes $(X_1 = X_2)$ & $(R_1 = R_2)$ & $(Y_1 = Y_2)$.

Une clause C_1 obtenue en renommant les variables d'une clause C_2 est appelée une **variante** de C_2.

5.2 Réduction des buts

Pour un langage $\mathcal{L}$ donné, soient S un ensemble de clauses et V un ensemble fini de variables.

La $\boldsymbol{(S,V)}$**-réduction** est une relation binaire sur l'ensemble des buts. Une occurrence de cette relation signifie qu'un but G est transformé en un but G', en utilisant une variante d'une clause de S et sans utiliser les variables de V. Nous notons alors $G \xrightarrow{r}_{S,V} G'$. La relation de réduction est définie par les trois règles suivantes. Dans chacune de ces règles, C dénote une variante d'une clause de S telle que $(V \cup var(G)) \cap var(C) = \emptyset$ [20].

20. Condition de standardisation des variables séparées.

Réduction directe

$A_1 \;\&\; F_1 \;/\!/\; H_1 \;\xrightarrow{r}_{S,V}\; F_2 \;\&\; F_1 \;/\!/\; H_3$
avec $C = (A_2 \text{:-} F_2 \;/\!/\; H_2)$ et
$H_3 = (H_1 \;\&\; H_2 \;\&\; A_2 \;\&\; (A_1 \overset{arg}{=} A_2))$,
si A_1 et A_2 sont compatibles
et H_3 est satisfaisable.

Réduction par inclusion

$X : Y_1 \;\&\; F_1 \;/\!/\; H_1 \;\xrightarrow{r}_{S,V}\; X : Z \;\&\; F_2 \;\&\; F_1 \;/\!/\; H_3$
avec $C = (Z < Y_2 \text{:-} F_2 \;/\!/\; H_2)$ et
$H_3 = (H_1 \;\&\; H_2 \;\&\; Z < Y_2 \;\&\; Y_1 = Y_2)$,
si H_3 est satisfaisable.

Réduction par héritage

$X < Y_1 \;\&\; F_1 \;/\!/\; H_1 \;\xrightarrow{r}_{S,V}\; X < Z \;\&\; F_2 \;\&\; F_1 \;/\!/\; H_3$
avec $C = (Z < Y_2 \text{:-} F_2 \;/\!/\; H_2)$ et
$H_3 = (H_1 \;\&\; H_2 \;\&\; Z < Y_2 \;\&\; Y_1 = Y_2)$,
si H_3 est satisfaisable.

Le lecteur intéressé par la réalisation des tests de satisfaisabilité pour les conjonctions de contraintes atomiques pourra se référer aux travaux réalisés dans le domaine des bases de connaissances terminologiques (e.g., [16]). En effet, la sémantique déclarative du langage de contraintes que nous utilisons est très proche de celle des langages employés dans ces systèmes. Les tests de satisfaisabilité proposés pour ces langages semblent donc pouvoir être adaptés aisément.

Soit $\xrightarrow{r}{}^{*}_{S,V}$ la fermeture réflexo-transitive de $\xrightarrow{r}_{S,V}$.

Etant donné S, une **réponse calculée** à un but G est une conjonction de contraintes atomiques H satisfaisable, telle que :

$$G \xrightarrow{r}{}^{*}_{S,V} \epsilon \;/\!/\; H.$$

On remarque que si H est obtenue par au moins une étape de réduction alors H est satisfaisable par construction.

Pour les sections suivantes, nous étendons les assignations aux atomes. Soient $T_1, \ldots, T_n$ les termes d'un atome A, dans leur ordre d'apparition dans A. Soit γ une assignation, alors $\gamma(A) = \langle \gamma(T_1), \ldots, \gamma(T_n) \rangle$. Nous étendons aussi l'égalité dans D ($=_D$) aux tuples de D^n.

6 Validité

Nous allons tout d'abord démontrer que chaque pas de réduction est valide. Puis nous montrerons (théorème 6.4) que toute réponse calculée est une réponse correcte.

Lemme 6.1 *Si $G \xrightarrow{r}_{S,V} G'$ à l'aide d'une clause de tête A, alors pour tout modèle $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ de S, $[G']^\Gamma \subseteq [A]^\mathcal{I}$.*

Preuve :

Soient les buts G et $G' = (F_2 \ \& \ F_1 \ /\!/ \ H_1 \ \& \ H_2 \ \& \ A \ \& \ H_4)$ tels que $G \xrightarrow{r}_{S,V} G'$ (par n'importe laquelle des trois règles de réduction), à l'aide de la clause $C = (A \text{ :- } F_2 \ /\!/ \ H_2)$. Soient $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle de S et $\alpha \in [G']^\Gamma$.

Γ est un modèle de S et C est une variante d'une clause de S, donc Γ est un modèle de $\{C\}$. Par définition, toute Γ-assignation est Γ-solution de C, d'où $\alpha \in [C]^\Gamma$. C'est-à-dire $\alpha \in [A \text{ :- } F_2 \ /\!/ \ H_2]^\Gamma$. Par définition d'une Γ-solution $\alpha \in (ASG^\Gamma - [F_2 \ /\!/ \ H_2 \ \& \ A]^\Gamma) \cup [A]^\mathcal{I}$.

Puisque $\alpha \in [G']^\Gamma$, nous avons $\alpha \in [F_2 \ /\!/ \ H_2 \ \& \ A]^\Gamma$. Donc $\alpha \notin (ASG^\Gamma - [F_2 \ /\!/ \ H_2 \ \& \ A]^\Gamma)$. On en déduit que $\alpha \in [A]^\mathcal{I}$. □

Théorème 6.2 *Si $G \xrightarrow{r}_{S,V} G'$, alors pour tout modèle Γ de S, $[G']^\Gamma \subseteq [G]^\Gamma$.*

Preuve :

Soient les buts G et G' tels que $G \xrightarrow{r}_{S,V} G'$. Soit $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle de S avec $\mathcal{I} = \langle D, id, membre_\mathcal{I}, herite_\mathcal{I}, attribut_\mathcal{I} \rangle$. Soit $\alpha \in [G']^\Gamma$.

Trois cas de réduction peuvent se présenter.

Cas 1 : G' est obtenu par la règle de réduction directe.

Posons $G = (A \ \& \ F_1 \ /\!/ \ H_1), G' = (F_2 \ \& \ F_1 \ /\!/ \ H_1 \ \& \ H_2 \ \& \ A_2 \ \& \ (A_1 \stackrel{arg}{=} A_2))$
et $C = (A_2 :\!- \ F_2 \ /\!/ \ H_2)$ la clause utilisée pour la réduction.

Puisque $\alpha \in [G']^\Gamma$ nous avons $\alpha \in [A_1 \stackrel{arg}{=} A_2]^{\mathcal{B}}$ et donc $\alpha(A_1) =_D \alpha(A_2)$. Or, par le lemme 6.1, nous avons $\alpha \in [A_2]^{\mathcal{I}}$, d'où $\alpha \in [A_1]^{\mathcal{I}}$.

Puisque $\alpha \in [G']^\Gamma$, nous avons également $\alpha \in [F_1 \ /\!/ \ H_1]^\Gamma$ d'où $\alpha \in [A_1]^{\mathcal{I}} \cap [F_1 \ /\!/ \ H_1]^\Gamma$. Donc $\alpha \in [A_1 \ \& \ F_1 \ /\!/ \ H_1]^\Gamma$, c'est-à-dire $\alpha \in [G]^\Gamma$.

Cas 2 : G' est obtenu par la règle de réduction par inclusion.

Posons $G = (X : Y_1 \ \& \ F_1 \ /\!/ \ H_1)$,
$G' = (X : Z \ \& \ F_2 \ \& \ F_1 \ /\!/ \ H_1 \ \& \ H_2 \ \& \ Z < Y_2 \ \& \ Y_1 = Y_2)$
et $C = (Z < Y_2 :\!- \ F_2 \ /\!/ \ H_2)$ la clause utilisée pour la réduction.

Par le lemme 6.1, $\alpha \in [Z < Y_2]^{\mathcal{I}}$. Donc par définition des ensembles de $\mathcal{I}$-solutions, $\langle \alpha(Z), \alpha(Y_2) \rangle \in herite_{\mathcal{I}}$. $\mathcal{I}$ étant une semi-interprétation $\forall x_1, x_2, x_3 \in D, \langle x_1, x_2 \rangle \in membre_{\mathcal{I}} \wedge \langle x_2, x_3 \rangle \in herite_{\mathcal{I}} \Rightarrow \langle x_1, x_3 \rangle \in membre_{\mathcal{I}}$.

D'où $\{x_1 | \langle x_1, \alpha(Z) \rangle \in membre_{\mathcal{I}}\} \subseteq \{x_1 | \langle x_1, \alpha(Y_2) \rangle \in membre_{\mathcal{I}}\}$.

Nous pouvons donc déduire que $ext_{\mathcal{I}\alpha}(Z) \subseteq ext_{\mathcal{I}\alpha}(Y_2)$.

Puisque $\alpha \in [G']^\Gamma$ nous avons $\alpha \in [X : Z]^{\mathcal{I}}$ et donc $\alpha(X) \in ext_{\mathcal{I}\alpha}(Z)$. Sachant que $ext_{\mathcal{I}\alpha}(Z) \subseteq ext_{\mathcal{I}\alpha}(Y_2)$, on en déduit que $\alpha \in [X : Y_2]^{\mathcal{I}}$.

Puisque $\alpha \in [G']^\Gamma$ nous avons également $\alpha \in [Y_1 = Y_2]^{\mathcal{B}}$ et donc $\alpha(Y_1) =_D \alpha(Y_2)$. Sachant que $\alpha \in [X : Y_2]^{\mathcal{I}}$ on en déduit que $\alpha \in [X : Y_1]^{\mathcal{I}}$.

Enfin, puisque $\alpha \in [G']^\Gamma$ nous avons $\alpha \in [F_1 \ /\!/ \ H_1]^\Gamma$. Donc $\alpha \in [X : Y_1 \ \& \ F_1 \ /\!/ \ H_1]^\Gamma$ c'est-à-dire $\alpha \in [G]^\Gamma$.

Cas 3 : G' est obtenu par la règle de réduction par héritage.

Posons $G = (X < Y_1 \ \& \ F_1 \ /\!/ \ H_1)$,
$G' = (X < Z \ \& \ F_2 \ \& \ F_1 \ /\!/ \ H_1 \ \& \ H_2 \ \& \ Z < Y_2 \ \& \ Y_1 = Y_2)$
et $C = (Z < Y_2 :\!- \ F_2 \ /\!/ \ H_2)$ la clause utilisée pour la réduction.

Par le lemme 6.1 $\alpha \in [Z < Y_2]^{\mathcal{I}}$. De plus $\alpha \in [X < Z]^\Gamma$ (car $\alpha \in [G']^\Gamma$). D'où $\langle \alpha(Z), \alpha(Y_2) \rangle \in herite_{\mathcal{I}}$ et $\langle \alpha(X), \alpha(Z) \rangle \in herite_{\mathcal{I}}$. $\mathcal{I}$ étant une semi-interprétation $herite_{\mathcal{I}}$ est transitive. Nous pouvons donc déduire que $\langle \alpha(X), \alpha(Y_2) \rangle \in herite_{\mathcal{I}}$ et donc que $\alpha \in [X < Y_2]^{\mathcal{I}}$

Nous pouvons poursuivre comme dans la cas précédent.

$\alpha \in [Y_1 = Y_2]^{\mathcal{B}}$ (car $\alpha \in [G']^\Gamma$). D'où $\alpha(Y_1) =_D \alpha(Y_2)$. Or $\alpha \in [X < Y_2]^{\mathcal{I}}$ donc $\alpha \in [X < Y_1]^{\mathcal{I}}$.

De plus $\alpha \in [F_1 \ /\!/ \ H_1]^\Gamma$ (car $\alpha \in [G']^\Gamma$) d'où $\alpha \in [X < Y_1 \ \& \ F_1 \ /\!/ \ H_1]^\Gamma$ c'est-à-dire $\alpha \in [G]^\Gamma$.

Fin des cas □

Corollaire 6.3 *Si* $G \xrightarrow{r}{}^*_{S,V} G'$*, alors pour tout modèle* Γ *de* S, $[G']^\Gamma \subseteq [G]^\Gamma$.

Esquisse de la preuve : par induction sur la longueur de la séquence de réductions, en utilisant le théorème 6.2 □

Théorème 6.4 (Validité) *Si* $G \xrightarrow{r}{}^*_{S,V} \epsilon \mathbin{/\!/} H$ *alors* $S \models H \Rightarrow G$.

Preuve :

Pour tout modèle $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ de S, par le corollaire 6.3 nous avons $[\epsilon \mathbin{/\!/} H]^\Gamma \subseteq [G]^\Gamma$.

Par définition des ensembles de Γ-solutions, $[H]^\mathcal{B} = [\epsilon \mathbin{/\!/} H]^\Gamma$ et donc $[H]^\mathcal{B} \subseteq [G]^\Gamma$.

Cette propriété étant vérifiée par tous les modèles de S, nous avons alors $S \models H \Rightarrow G$. □

7 Complétude

Nous allons démontrer la complétude de la réduction. Nous montrerons (théorème 7.9) que toute solution d'une réponse correcte est solution d'une réponse calculée.

La technique de démonstration utilisée ici est dûe à Höhfeld et Smolka [30]. Elle utilise une mesure de complexité basée sur un ordre bien fondé, défini sur un ensemble de multi-ensembles.

7.1 Principe de la démonstration

1. On se dote d'un moyen itératif permettant d'obtenir des modèles minimaux $\langle \mathcal{T}, \mathcal{B} \rangle$ en construisant une chaîne $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_i, \ldots$ à partir de $\mathcal{B}$, et en prenant $\mathcal{T} = \bigcup_{i \geq 0} \mathcal{T}_i$.

2. Pour un atome A et une solution $\alpha \in [A]^\mathcal{T}$, on définit la *complexité* de α comme étant le nombre minimal d'itérations i tel que $\alpha \in [A]^{\mathcal{T}_i}$.

3. Pour une solution α d'un but, la complexité associée est le multi-ensemble composé des complexités des atomes de ce but.

4. On introduit un ordre bien fondé $\ll$ sur la complexité des solutions des buts.

5. On montre que pour un but $G = (A \ \& \ \ldots \ /\!/ \ \ldots)$, si G possède une solution α alors il existe une réduction $G \xrightarrow{r}_{S,V} G'$ telle que α soit solution de G' et telle que la complexité de α pour G' soit strictement inférieure à la complexité de α pour G.

6. La complétude de $\xrightarrow{r}{}^{*}_{S,V}$ s'obtient alors par induction sur la complexité des buts, en utilisant l'ordre bien fondé $\ll$.

Dans [30], cette technique est utilisée pour montrer la complétude d'un système de réduction comportant une seule règle, vis-à-vis d'une sémantique déclarative dans laquelle l'interprétation des contraintes atomiques est indépendante de celle des liens atomiques.

Dans notre cas, le principe global de démonstration reste applicable, mais le raisonnement détaillé est différent et plus complexe. En effet, il convient de tenir compte de trois règles de réduction et d'une sémantique déclarative dans laquelle l'interprétation des liens atomiques est dépendante de celle des contraintes atomiques.

Nous considérons pour la suite que l'ordre partiel $\leq$ sur l'ensemble des semi-interprétations et la définition des ensembles de $\mathcal{J}$-solutions pour une semi-interprétation $\mathcal{J}$, sont étendus à tous les tuples $\mathcal{J} = \langle D, id, membre, herite, attribut \rangle$ pour lesquels $D \neq \emptyset$, $id : \mathcal{K} \to D$, $membre \subseteq D \times D$, $herite \subseteq D \times D$ et $attribut \subseteq D \times D \times D$.

7.2 Construction itérative d'un modèle minimal

Nous nous dotons d'une méthode itérative permettant, pour un ensemble de clauses S et une semi-interprétation $\mathcal{B}$ donnés, de construire un modèle minimal $\Gamma = \langle \mathcal{T}, \mathcal{B} \rangle$ de S.

Soient un ensemble de clauses S et une semi-interprétation $\mathcal{B} = \langle D, id, membre_{\mathcal{B}}, herite_{\mathcal{B}}, attribut_{\mathcal{B}} \rangle$.

Considérons les tuples $\mathcal{T}_{i,(i \geq 0)}$, définis par :

$$\mathcal{T}_i \ := \ \langle D, id, membre_i, herite_i, attribut_i \rangle$$

Avec $membre_0 := \emptyset, herite_0 := \emptyset, attribut_0 := \emptyset$,
et pour $i > 0$

$$\begin{aligned} attribut_i \ := \ & \{\langle \alpha(X), \alpha(R), \alpha(Y) \rangle | ((X.R \to Y) \text{ :- } F \ /\!/ \ H) \in S \wedge \\ & \alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \ \& \ X.R \to Y]^{\mathcal{B}}\} \end{aligned}$$

$$herite_i \ := \ her\_dir_i \cup her\_ind_i$$

$$
\begin{aligned}
her\_dir_i &:= \{\langle \alpha(X), \alpha(Y)\rangle | (X < Y :\!\!- F /\!\!/ H) \in S \wedge \\
&\quad \alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \,\&\, X < Y]^{\mathcal{B}}\}
\end{aligned}
$$

$$
\begin{aligned}
her\_ind_i &:= \{\langle d, \alpha(Y)\rangle | (X < Y :\!\!- F /\!\!/ H) \in S \wedge \\
&\quad \alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \,\&\, X < Y]^{\mathcal{B}} \wedge \langle d, \alpha(X)\rangle \in herite_{i-1}\}
\end{aligned}
$$

$$
membre_i := mem\_dir_i \cup mem\_ind_i
$$

$$
\begin{aligned}
mem\_dir_i &:= \{\langle \alpha(X), \alpha(Y)\rangle | (X : Y :\!\!- F /\!\!/ H) \in S \wedge \\
&\quad \alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \,\&\, X : Y]^{\mathcal{B}}\}
\end{aligned}
$$

$$
\begin{aligned}
mem\_ind_i &:= \{\langle d, \alpha(Y)\rangle | (X < Y :\!\!- F /\!\!/ H) \in S \wedge \\
&\quad \alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \,\&\, X < Y]^{\mathcal{B}} \wedge \langle d, \alpha(X)\rangle \in membre_{i-1}\}
\end{aligned}
$$

De façon informelle, on peut dire que $attribut_i$ est l'ensemble des valorisations d'attributs pouvant être déduites directement à partir d'une clause et des relations de $\mathcal{T}_{i-1}$ et $\mathcal{B}$.

Dans le même ordre d'idée, her_dir_i et mem_dir_i sont respectivement les ensembles de liens d'héritage et de liens d'instanciation pouvant être déduits directement à partir d'une clause et des relations de $\mathcal{T}_{i-1}$ et $\mathcal{B}$.

Enfin, her_ind_i (resp. mem_ind_i) est l'ensemble des liens d'héritage (resp. d'instanciation) produits indirectement à l'aide des liens d'héritage pouvant être déduits à partir d'une clause, que l'on combine avec les liens de $herite_{i-1}$ (resp. $membre_{i-1}$). Ceci correspond en fait à la transitivité de l'héritage (resp. à l'inclusion des instances d'une classe dans ses super-classes).

Soit $\mathcal{T} := \langle D, id, membre, herite, attribut\rangle$ avec $membre := \bigcup_{i\geq 0} membre_i$, $herite := \bigcup_{i\geq 0} herite_i$ et $attribut := \bigcup_{i\geq 0} attribut_i$.

Soit $\Gamma = \langle \mathcal{T}, \mathcal{B}\rangle$. Γ est appelée **extension par itérations de $\mathcal{B}$ pour S.** La séquence $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_i, \ldots$ est appelée **chaîne de construction de Γ.**

Nous allons tout d'abord démontrer que Γ est un modèle de S. Puis nous montrerons que si $\Omega = \langle \mathcal{T}, \mathcal{B}\rangle$ est un modèle minimal de S alors Ω est l'extension par itérations de $\mathcal{B}$ pour S.

Lemme 7.1 *Soit $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_i, \ldots$ la chaîne de construction d'une extension par itérations de $\mathcal{B}$ pour S, alors pour tout $i \geq 0$, $\mathcal{T}_i \leq \mathcal{B}$.*

Esquisse de la preuve : Par induction sur i on montre que $membre_i \subseteq membre_{\mathcal{B}}, herite_i \subseteq herite_{\mathcal{B}}$ et $attribut_i \subseteq attribut_{\mathcal{B}}$. Donc $\mathcal{T}_i \leq \mathcal{B}$.
□

Lemme 7.2 *Soit $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_i, \ldots$ la chaîne de construction d'une extension par itérations de $\mathcal{B}$ pour S, alors pour tout $i > 0$, $\mathcal{T}_{i-1} \leq \mathcal{T}_i$.*

Esquisse de la preuve : Par induction sur i, on montre que $membre_{i-1} \subseteq membre_i$, $herite_{i-1} \subseteq herite_i$ et $attribut_{i-1} \subseteq attribut_i$. D'où $\mathcal{T}_{i-1} \leq \mathcal{T}_i$.
□

Soit $herite^+$ la fermeture transitive de $herite$.

Soit $inter(d_1, d_2) := \{e | \langle d_1, e\rangle \in herite^+ \wedge \langle e, d_2\rangle \in herite^+\}$.

Informellement, $inter(d_1, d_2)$ est l'ensemble des éléments situés *entre* d_1 et d_2 pour $herite^+$.

Lemme 7.3 $\forall d_1, d_2 \in D$, *si* $card(inter(d_1, d_2)) \geq 1$ *alors* $\exists i \in I\!N, \exists e_1 \ldots \exists e_n$ *tels que* $\{\langle d_1, e_1\rangle, \langle e_1, e_2\rangle, \ldots, \langle e_{n-1}, e_n\rangle, \langle e_n, d_2\rangle\} \subseteq her\_dir_i$

Preuve :

La preuve est réalisée par induction sur $card(inter(d_1, d_2))$.

Pour tout d_1 et tout d_2 de D tels que $card(inter(d_1, d_2)) = 1$
la propriété est vérifiée.

Hypothèse d'induction : pour $k \in \mathbb{N}^*$ donné, $\forall d_1, d_2 \in D$,
si $k \geq card(inter(d_1, d_2)) \geq 1$ alors $\exists i \in \mathbb{N}, \exists e_1 \ldots \exists e_n$ tels que :
$\{\langle d_1, e_1\rangle, \langle e_1, e_2\rangle, \ldots, \langle e_{n-1}, e_n\rangle, \langle e_n, d_2\rangle\} \subseteq her\_dir_i$

Soient $d_1, d_2 \in D$, tels que $k + 1 = card(inter(d_1, d_2))$. Il nous faut démontrer que $\exists i \in \mathbb{N}, \exists e_1 \ldots \exists e_n$ tels que $\{\langle d_1, e_1\rangle, \langle e_1, e_2\rangle, \ldots, \langle e_{n-1}, e_n\rangle, \langle e_n, d_2\rangle\} \subseteq her\_dir_i$.

Soit $e \in inter(d_1, d_2)$, e existe car $card(inter(d_1, d_2)) = k + 1 (> 1)$.
Par construction, $inter(d_1, e) \subseteq inter(d_1, d_2)$ et $inter(e, d_2) \subseteq inter(d_1, d_2)$.

A partir du lemme 7.1 nous pouvons déduire que $herite \subseteq herite_{\mathcal{B}}$. Donc, puisque $herite_{\mathcal{B}}$ est transitive nous avons $herite^+ \subseteq herite_{\mathcal{B}}$.

Puisque $herite^+ \subseteq herite_{\mathcal{B}}$ avec $herite_{\mathcal{B}}$ un ordre strict, nous avons alors les inclusions strictes suivantes :
$inter(d_1, e) \subset inter(d_1, d_2)$ et $inter(e, d_2) \subset inter(d_1, d_2)$.

Donc $card(inter(d_1, e)) < card(inter(d_1, d_2))$ et $card(inter(e, d_2)) < card(inter(d_1, d_2))$.

Ceci signifie que $card(inter(d_1, e)) \leq k$ et $card(inter(e, d_2)) \leq k$.

Par hypothèse d'induction nous avons:
$\exists j_1 \in \mathbb{N}, \exists a_1 \ldots \exists a_p$ tels que
$\{\langle d_1, a_1\rangle, \langle a_1, a_2\rangle, \ldots, \langle a_{p-1}, a_p\rangle, \langle a_p, e\rangle\} \subseteq her\_dir_{j_1}$
et $\exists j_2 \in \mathbb{N}, \exists b_1 \ldots \exists b_q$ tels que
$\{\langle e, b_1\rangle, \langle b_1, b_2\rangle, \ldots, \langle b_{q-1}, b_q\rangle, \langle b_q, d_2\rangle\} \subseteq her\_dir_{j_2}$

Soient $j_1, j_2 \in \mathbb{N}$, $a_1, \ldots, a_p$ et $b_1, \ldots, b_q \in D$ vérifiant ces deux propriétés.

Du lemme 7.2 et de la définition de *her_dir*, on déduit que $\forall j > 0$, $her\_dir_{j-1} \subseteq her\_dir_j$.

Donc, pour $j_3 = sup(j_1, j_2)$, nous avons
$\{\langle d_1, a_1\rangle, \langle a_1, a_2\rangle, \ldots, \langle a_{p-1}, a_p\rangle, \langle a_p, e\rangle\} \subseteq her\_dir_{j_3}$
et $\{\langle e, b_1\rangle, \langle b_1, b_2\rangle, \ldots, \langle b_{q-1}, b_q\rangle, \langle b_q, d_2\rangle\} \subseteq her\_dir_{j_3}$
D'où $\exists i \in \mathbb{N}, \exists e_1 \ldots \exists e_n$ tels que
$\{\langle d_1, e_1\rangle, \langle e_1, e_2\rangle, \ldots, \langle e_{n-1}, e_n\rangle, \langle e_n, d_2\rangle\} \subseteq her\_dir_i$.
□

Lemme 7.4 *Soit $\langle \mathcal{T}, \mathcal{B}\rangle$ l'extension par itérations de $\mathcal{B}$ pour S, alors $\mathcal{T}$ est une semi-interprétation.*

Preuve:

Nous allons montrer que 1) *herite* est une relation d'ordre strict et 2) $\forall d_1, d_2, d_3 \in D, \langle d_1, d_3\rangle \in membre \wedge \langle d_3, d_2\rangle \in herite \Rightarrow \langle d_1, d_2\rangle \in membre$ (i.e. les instances d'une classe sont aussi des instances de ses super-classes).

Propriété 1) Pour tout $i \geq 0$, $herite_i \subseteq herite_{\mathcal{B}}$ (lemme 7.1), et par suite $herite \subseteq herite_{\mathcal{B}}$. Or $herite_{\mathcal{B}}$ est irréflexive et antisymétrique, puisque $\mathcal{B}$ est une semi-interprétation. Donc *herite* est irréflexive et antisymétrique.

Montrons que *herite* est aussi transitive.

Soient $d_1, d_2, d_3 \in D$ tels que $\langle d_1, d_2\rangle \in herite \wedge \langle d_2, d_3\rangle \in herite$.

Cas 1 : $card(inter(d_1, d_2)) \geq 1 \wedge card(inter(d_2, d_3)) \geq 1$

En appliquant le lemme 7.3, nous avons:

$\exists j_1 \in \mathbb{N}, \exists a_1 \ldots \exists a_p$ tels que

$\{\langle d_1, a_1\rangle, \langle a_1, a_2\rangle, \ldots, \langle a_{p-1}, a_p\rangle, \langle a_p, d_2\rangle\} \subseteq her\_dir_{j_1}$

et $\exists j_2 \in \mathbb{N}, \exists b_1 \ldots \exists b_q$ tels que

$\{\langle d_2, b_1\rangle, \langle b_1, b_2\rangle, \ldots, \langle b_{q-1}, b_q\rangle, \langle b_q, d_3\rangle\} \subseteq her\_dir_{j_2}$

Du lemme 7.2 et de la définition de *her_dir*, on déduit que $\forall j > 0$, $her\_dir_{j-1} \subseteq her\_dir_j$.

Donc, pour $j_3 = sup(j_1, j_2)$, nous avons

$\{\langle d_1, a_1\rangle, \langle a_1, a_2\rangle, \ldots, \langle a_{p-1}, a_p\rangle, \langle a_p, d_2\rangle\} \subseteq her\_dir_{j_3}$

et $\{\langle d_2, b_1\rangle, \langle b_1, b_2\rangle, \ldots, \langle b_{q-1}, b_q\rangle, \langle b_q, d_3\rangle\} \subseteq her\_dir_{j_3}$

D'ou $\exists i \in \mathbb{N}, \exists e_1 \ldots \exists e_n$ tels que

$\{\langle d_1, e_1\rangle, \langle e_1, e_2\rangle, \ldots, \langle e_{n-1}, e_n\rangle, \langle e_n, d_3\rangle\} \subseteq her\_dir_i$.

Pour un tel i et un tel n, à partir du lemme 7.2 et de la définition de *her_ind* on déduit que $\langle d_1, d_3\rangle \in her\_ind_{i+n}$.

D'où $\langle d_1, d_3\rangle \in herite_{i+n}$. Comme $herite_{i+n} \subseteq herite$, nous avons $\langle d_1, d_3\rangle \in herite$.

Cas 2 : $card(inter(d_1, d_2)) = 0 \vee card(inter(d_2, d_3)) = 0$

Ce cas est proche du précédent, avec $\exists j_1, \langle d_1, d_2\rangle \in her\_dir_{j_1}$ ou $\exists j_2, \langle d_2, d_3\rangle \in her\_dir_{j_2}$ (ou les deux). Par un raisonnement similaire nous avons aussi dans ce cas $\langle d_1, d_3\rangle \in herite$.

Fin des cas

herite est donc transitive.

Ceci permet de conclure que *herite* est une relation d'ordre strict.

Propriété 2) Montrons que la contrainte liant *herite* et *membre* pour une semi-interprétation est aussi satisfaite. Soient $d_1, d_2, d_3 \in D$ tels que $\langle d_1, d_2\rangle \in membre \wedge \langle d_2, d_3\rangle \in herite$. Nous allons montrer que $\langle d_1, d_3\rangle \in membre$.

Cas 1 : $card(inter(d_2, d_3)) \geq 1$.

D'après le lemme 7.3, nous avons :

$\exists j_1 \in \mathbb{N}, \exists e_1 \ldots \exists e_n$ tels que

$\{\langle d_2, e_1\rangle, \langle e_1, e_2\rangle, \ldots, \langle e_{n-1}, e_n\rangle, \langle e_n, d_3\rangle\} \subseteq her\_dir_{j_1}$.

De plus par définition de *membre*, $\exists j_2 \in \mathbb{N}, \langle d_1, d_2\rangle \in membre_{j_2}$.

Soient de tels entiers j_1, j_2 et n. Soit $i = sup(j_1, j_2)$. A partir du lemme 7.2 et de la définition de *mem_ind*, on déduit que $\langle d_1, d_3\rangle \in mem\_ind_{i+n}$.

Comme $membre_{i+n} \subseteq membre$, nous avons $\langle d_1, d_3\rangle \in membre$.

Cas 2 : $card(inter(d_2, d_3)) = 0$.

Ce cas est similaire au précédent, avec $\exists j_1, \langle d_2, d_3\rangle \in her\_dir_{j_1}$.

Fin des cas

Les instances d'une classe sont donc aussi des instances de ses super-classes.

Les propriétés **1)** et **2)** obtenues montrent que $\mathcal{T}$ satisfait à la définition d'une semi-interprétation.

□

Théorème 7.5 *Soit* Γ *l'extension par itérations de* $\mathcal{B}$ *pour* S. Γ *est un modèle de* S.

Preuve :

Soit $\Gamma = \langle \mathcal{T}, \mathcal{B} \rangle$ l'extension par itérations de $\mathcal{B}$ pour S. $\mathcal{B}$ est une semi-interprétation, $\mathcal{T}$ est aussi une semi-interprétation (lemme 7.4) et $\mathcal{T} \leq \mathcal{B}$ (lemme 7.1). Γ est donc une interprétation.

Pour montrer que Γ est un modèle de S, il suffit de montrer que pour toute clause C de S et tout $\alpha \in ASG^{\Gamma}$, nous avons $\alpha \in [C]^{\Gamma}$.

Soient $C = (A :\!\!- F \;/\!/\; H)$ une clause de S et $\alpha \in ASG^{\Gamma}$.
Par définition, $[C]^{\Gamma} = (ASG^{\Gamma} - [F \;/\!/\; H \;\&\; A]^{\Gamma}) \cup [A]^{\mathcal{T}}$. Nous avons deux cas possibles : $\alpha \in (ASG^{\Gamma} - [F \;/\!/\; H \;\&\; A]^{\Gamma})$ ou $\alpha \notin (ASG^{\Gamma} - [F \;/\!/\; H \;\&\; A]^{\Gamma})$.

Cas 1 : $\alpha \in (ASG^{\Gamma} - [F \;/\!/\; H \;\&\; A]^{\Gamma})$.

Alors $\alpha \in [C]^{\Gamma}$.

Cas 2 : $\alpha \notin (ASG^{\Gamma} - [F \;/\!/\; H \;\&\; A]^{\Gamma})$.

Alors $\alpha \in [F \;/\!/\; H \;\&\; A]^{\Gamma}$. Nous avons donc $\alpha \in [F]^{\mathcal{T}}$ et $\alpha \in [H \;\&\; A]^{\mathcal{B}}$, et par construction de $\mathcal{T}$, nous savons qu'il existe $\mathcal{T}_i$ tel que $\alpha \in [F]^{\mathcal{T}_i}$.

Si A est de la forme $X : Y$ (resp. $X < Y$, $X.R \rightarrow Y$), puisque $\alpha \in [F]^{\mathcal{T}_i}$ et que $\alpha \in [H \;\&\; A]^{\mathcal{B}}$, par construction de $\mathcal{T}_{i+1}$, à partir de C nous avons $\langle \alpha(X), \alpha(Y) \rangle \in mem\_dir_{i+1}$ (resp. $\langle \alpha(X), \alpha(Y) \rangle \in her\_dir_{i+1}$, $\langle \alpha(X), \alpha(R), \alpha(Y) \rangle \in attribut_{i+1}$).

Donc quelle que soit la forme de A, $\alpha \in [A]^{\mathcal{T}_{i+1}}$. D'où $\alpha \in [A]^{\mathcal{T}}$.

On a alors $\alpha \in [C]^{\Gamma}$.

Fin des cas

Nous pouvons alors conclure que Γ est un modèle de S.

□

Théorème 7.6 *Soit* Ω *un modèle minimal de base* $\mathcal{B}$ *d'un ensemble de clauses* S. Ω *est l'extension par itérations de* $\mathcal{B}$ *pour* S.

Preuve :

Soit $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle minimal de base $\mathcal{B}$ d'un ensemble de clauses S, avec $\mathcal{I} = \langle D, id, membre_{\mathcal{I}}, herite_{\mathcal{I}}, attribut_{\mathcal{I}} \rangle$.

Soit $\Gamma = \langle \mathcal{T}, \mathcal{B} \rangle$ l'extension par itérations de $\mathcal{B}$ pour S, avec $\mathcal{T} = \langle D, id, membre_{\mathcal{T}}, herite_{\mathcal{T}}, attribut_{\mathcal{T}} \rangle$ et $membre_{\mathcal{T}} = \bigcup_{i \geq 0} membre_i, herite_{\mathcal{T}} = \bigcup_{i \geq 0} herite_i$ et $attribut_{\mathcal{T}} = \bigcup_{i \geq 0} attribut_i$.

Γ est un modèle de S (théorème 7.5). Puisque Ω est un modèle minimal, nous ne pouvons pas avoir $\Gamma < \Omega$. Donc pour montrer que $\Gamma = \Omega$, il suffit de montrer que $\Gamma \leq \Omega$.

Par définition, Γ a même base que Ω et nous savons aussi que $\mathcal{T}$ et $\mathcal{I}$ ont même domaine D et même fonction id. Il nous reste donc à montrer que $membre_{\mathcal{T}} \subseteq membre_{\mathcal{I}}$, $herite_{\mathcal{T}} \subseteq herite_{\mathcal{I}}$ et $attribut_{\mathcal{T}} \subseteq attribut_{\mathcal{I}}$.

Pour cela nous montrons par récurrence que pour tout $i \geq 0$, $attribut_i \subseteq attribut_{\mathcal{I}}, herite_i \subseteq herite_{\mathcal{I}}$ et $membre_i \subseteq membre_{\mathcal{I}}$.

Pour $i = 0$, nous avons bien $attribut_0 \subseteq attribut_{\mathcal{I}}, herite_0 \subseteq herite_{\mathcal{I}}$ et $membre_0 \subseteq membre_{\mathcal{I}}$.

Hypothèse de récurrence : $(i > 0)$ $attribut_{i-1} \subseteq attribut_{\mathcal{I}}, herite_{i-1} \subseteq herite_{\mathcal{I}}$ et $membre_{i-1} \subseteq membre_{\mathcal{I}}$.

- Montrons que pour tout $\langle d_1, d_2, d_3 \rangle \in attribut_i$, nous avons $\langle d_1, d_2, d_3 \rangle \in attribut_{\mathcal{I}}$.

  Soit $\langle d_1, d_2, d_3 \rangle \in attribut_i$.

  Par définition de $attribut_i$, il existe une clause de la forme $(X.R \rightarrow Y) \text{ :- } F \ // \ H$ dans S et une assignation α telles que $\alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \ \& \ X.R \rightarrow Y]^{\mathcal{B}} \wedge \langle d_1, d_2, d_3 \rangle =_D \langle \alpha(X), \alpha(R), \alpha(Y) \rangle$.

  F ne contient que des liens atomiques, donc si $\mathcal{J}_1$ et $\mathcal{J}_2$ sont des semi-interprétations telles que $\mathcal{J}_1 \leq \mathcal{J}_2$, alors $[F]^{\mathcal{J}_1} \subseteq [F]^{\mathcal{J}_2}$. C'est pourquoi, de $\alpha \in [F]^{\mathcal{T}_{i-1}}$ et d'après l'hypothèse de récurrence, on déduit que $\alpha \in [F]^{\mathcal{I}}$.

  Nous savons que $((X.R \rightarrow Y) \text{ :- } F \ // \ H) \in S$ avec $\alpha \in [F]^{\mathcal{I}}$ et $\alpha \in [H \ \& \ X.R \rightarrow Y]^{\mathcal{B}}$. Comme Ω est un modèle de S, on en déduit que $\alpha \in [X.R \rightarrow Y]^{\mathcal{I}}$, c'est-à-dire $\langle \alpha(X), \alpha(R), \alpha(Y) \rangle \in attribut_{\mathcal{I}}$. Nous avons donc $\langle d_1, d_2, d_3 \rangle \in attribut_{\mathcal{I}}$.

- Montrons que pour tout $\langle d_1, d_2 \rangle \in herite_i$, nous avons $\langle d_1, d_2 \rangle \in herite_{\mathcal{I}}$.

  Soit $\langle d_1, d_2 \rangle \in herite_i$. Par définition de $herite_i$, soit $\langle d_1, d_2 \rangle \in her\_dir_i$ soit $\langle d_1, d_2 \rangle \in her\_ind_i$.

Cas 1 : $\langle d_1, d_2 \rangle \in her\_dir_i$.

Le même raisonnement que celui employé ci-dessus pour démontrer que $\langle d_1, d_2, d_3\rangle \in attribut_{\mathcal{I}}$ permet de montrer que $\langle d_1, d_2\rangle \in herite_{\mathcal{I}}$ à partir de la définition de her_dir_i.

Cas 2 : $\langle d_1, d_2\rangle \in her\_ind_i$.

Alors il existe une clause $(X < Y :\!\!- F \mathbin{/\!\!/} H) \in S$ et une assignation α telles que $\alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \;\&\; X < Y]^{\mathcal{B}} \wedge \alpha(Y) =_D d_2$ et il existe $d_3 \in D$ tel que $\langle d_1, d_3\rangle \in herite_{i-1} \wedge \alpha(X) =_D d_3$.

Le même raisonnement que celui employé ci-dessus pour démontrer que $\langle d_1, d_2, d_3\rangle \in attribut_{\mathcal{I}}$ permet de montrer que $\langle d_3, d_2\rangle \in herite_{\mathcal{I}}$.

De plus, $\langle d_1, d_3\rangle \in herite_{i-1}$ permet de déduire à l'aide de l'hypothèse de récurrence que $\langle d_1, d_3\rangle \in herite_{\mathcal{I}}$.

Puisque $herite_{\mathcal{I}}$ est transitive (car $\mathcal{I}$ est une semi-interprétation), on déduit alors que $\langle d_1, d_2\rangle \in herite_{\mathcal{I}}$.

Fin des cas

– Montrons que pour tout $\langle d_1, d_2\rangle \in membre_i$, nous avons $\langle d_1, d_2\rangle \in membre_{\mathcal{I}}$.

Soit $\langle d_1, d_2\rangle \in membre_i$. Par définition de $membre_i$, soit $\langle d_1, d_2\rangle \in mem\_dir_i$ soit $\langle d_1, d_2\rangle \in mem\_ind_i$.

Cas 1 : $\langle d_1, d_2\rangle \in mem\_dir_i$.

Le même raisonnement que celui employé ci-dessus pour démontrer que $\langle d_1, d_2, d_3\rangle \in attribut_{\mathcal{I}}$ permet de montrer que $\langle d_1, d_2\rangle \in membre_{\mathcal{I}}$ à partir de la la définition de mem_dir_i.

Cas 2 : $\langle d_1, d_2\rangle \in mem\_ind_i$.

Alors il existe une clause $(X < Y :\!\!- F \mathbin{/\!\!/} H) \in S$ et une assignation α telles que $\alpha \in [F]^{\mathcal{T}_{i-1}} \wedge \alpha \in [H \;\&\; X < Y]^{\mathcal{B}} \wedge \alpha(Y) =_D d_2$ et il existe $d_3 \in D$ tel que $\langle d_1, d_3\rangle \in membre_{i-1} \wedge \alpha(X) =_D d_3$.

Le même raisonnement que celui employé ci-dessus pour démontrer que $\langle d_1, d_2, d_3\rangle \in attribut_{\mathcal{I}}$ permet de montrer que $\langle d_3, d_2\rangle \in herite_{\mathcal{I}}$.

De plus, $\langle d_1, d_3\rangle \in membre_{i-1}$ permet de déduire à l'aide de l'hypothèse de récurrence que $\langle d_1, d_3\rangle \in membre_{\mathcal{I}}$.

Puisque $herite_{\mathcal{I}}$ est transitive (car $\mathcal{I}$ est une semi-interprétation), on déduit alors que $\langle d_1, d_2\rangle \in herite_{\mathcal{I}}$.

Puisque les instances d'une classe sont aussi des instances de ses super-classes (car $\mathcal{I}$ est une semi-interprétation), on déduit alors que $\langle d_1, d_2\rangle \in membre_{\mathcal{I}}$.

Fin des cas

Donc pour tout i, $attribut_i \subseteq attribut_{\mathcal{I}}, herite_i \subseteq herite_{\mathcal{I}}$ et $membre_i \subseteq mem_{\mathcal{I}}$. Nous avons alors $\Gamma \leq \Omega$, et donc $\Gamma = \Omega$.

□

7.3 Mesure de Complexité

Nous utilisons la mesure de complexité de [30].

Soit $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle minimal d'un ensemble de clauses S. Ω est l'extension par itérations de $\mathcal{B}$ pour S (théorème 7.6). Soit $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_i, \ldots$ la chaîne de construction de Ω.

Soient A un lien atomique et $\alpha \in [A]^{\Omega}$. La **complexité** de α pour A dans Ω est $comp(\alpha, A, \Omega) := min\{i \mid \alpha \in [A]^{\mathcal{T}_i}\}$.

Soientt $Q = A_1 \ \& \ \ldots \ \& \ A_n \ /\!/ \ H$ un but et $\alpha \in [Q]^{\Omega}$. La **complexité** de α pour Q dans Ω est le multi-ensemble composé de la complexité de tous les liens atomiques de Q :

$comp(\alpha, Q, \Omega) := \{\!\{comp(\alpha, A_1, \Omega), \ldots, comp(\alpha, A_n, \Omega)\}\!\}$.

Nous utilisons sur ces multi-ensembles l'ordre total $\underline{\ll}$ et l'ordre bien fondé $\ll$ de [30]. Soient $cpxt_1$ et $cpxt_2$ deux multi-ensembles représentant des complexités pour deux buts,

$cpxt_1 \underline{\ll} cpxt_2 \Leftrightarrow$

$\exists cpxt'_1 \subseteq cpxt_1$ et $\exists cpxt'_2 \subseteq cpxt_2$ tels que
$cpxt_1 = (cpxt_2 - cpxt'_2) \cup cpxt'_1$
et $\forall x \in cpxt'_1 \ \exists x' \in cpxt'_2, x < x'$.

$cpxt_1 \ll cpxt_2 \Leftrightarrow$

$cpxt_1 \underline{\ll} cpxt_2$ et $cpxt_1 \neq cpxt_2$.

Informellement, nous avons $cpxt_1 \underline{\ll} cpxt_2$ lorsque $cpxt_1$ peut être obtenu à partir de $cpxt_2$ de la façon suivante : on supprime des éléments de $cpxt_2$ (éventuellement aucun) et on ajoute des éléments strictement inférieurs à un des éléments supprimés.

Le lecteur trouvera une étude détaillée concernant les relations d'ordre sur multi-ensembles dans [24].

Soient β une assignation et V un ensemble de variables, nous notons $\beta|_V$ la restriction de β aux variables de V. On étend cette restriction aux ensembles de Ω-solutions de la façon suivante :
$[L]^{\Omega}_V := \{\beta|_V \mid \beta \in [L]^{\Omega}\}$ avec L un but ou une clause.
La restriction est aussi définie pour les ensembles de $\mathcal{I}$-solutions :

$[L]^{\mathcal{I}}_V := \{\beta|_V \mid \beta \in [L]^{\mathcal{I}}\}$ avec L une conjonction de contraintes atomiques ou de liens atomiques.

Soit Ω un modèle minimal d'un ensemble de clauses S. Soient $Q = (F \;/\!/\; H)$ un but et $\alpha \in [Q]^{\Omega}_V$. La **$V$-complexité** de α pour Q dans Ω est :
$comp_V(\alpha, Q, \Omega) := min\{comp(\beta, Q, \Omega) \mid \beta \in [Q]^{\Omega} \wedge \alpha = \beta|_V\}$ [21].

Théorème 7.7 *Soient Ω un modèle minimal d'un ensemble de clauses S, G un but de la forme A_1 & F_1 // H_1, et une assignation $\alpha \in [G]^{\Omega}_V$; il existe alors une clause C de S telle que :*

1. *une (S,V)-réduction de G est possible à partir d'une variante de C.*
2. *si $G \xrightarrow{r}_{S,V} G'$ à l'aide d'une variante de C alors $\alpha \in [G']^{\Omega}_V$ et $comp_V(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$.*

Preuve :

Soit $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle minimal de S. Soient G un but de la forme A_1 & F_1 // H_1 et $\alpha \in [G]^{\Omega}_V$ une assignation.

Par le théorème 7.6 nous savons que Ω est l'extension par itérations de $\mathcal{B}$ pour S. Soit $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_i, \ldots$ la chaîne de construction de Ω. Chaque tuple $\mathcal{T}_i$ sera noté $\langle D, id, membre_i, herite_i, attribut_i \rangle$.

Soit une assignation $\beta \in [G]^{\Omega}$ telle que $\alpha = \beta|_V$ et $comp_V(\alpha, G, \Omega) = comp(\beta, G, \Omega)$.
Soit $i = comp(\beta, A_1, \Omega)$. Nous avons donc $\beta \in [A_1]^{\mathcal{T}_i}$.

Rappels : si $T_1, \ldots, T_n$ sont les termes de l'atome A_1, dans leur ordre d'apparition dans A_1 nous notons $\beta(A_1)$ le tuple $\langle \gamma(T_1), \ldots, \gamma(T_n) \rangle$. L'égalité dans D ($=_D$) est étendue aux tuples de D^n.

Puisque $\beta \in [A_1]^{\mathcal{T}_i}$, par définition des $\mathcal{T}_i$-solutions pour les liens atomiques, $\beta(A_1) \in attribut_i \vee \beta(A_1) \in herite_i \vee \beta(A_1) \in membre_i$.

D'après les définitions de $attribut_i$, $herite_i$ et $membre_i$, trois cas sont alors possibles :

1. $\beta(A_1) \in attribut_i \vee \beta(A_1) \in her\_dir_i \vee \beta(A_1) \in mem\_dir_i$.
2. $\beta(A_1) \in her\_ind_i$.
3. $\beta(A_1) \in mem\_ind_i$.

21. Minimal pour $\ll$.

Cas 1 : $\beta(A_1) \in attribut_i \vee \beta(A_1) \in her\_dir_i \vee \beta(A_1) \in mem\_dir_i$

Il existe alors une clause $C = (A_2 :\!- F_2 \;/\!/\; H_2)$ de S et une assignation γ telles que $\gamma \in [F_2]^{\mathcal{T}_{i-1}} \wedge \gamma \in [H_2 \;\&\; A_2]^{\mathcal{B}} \wedge \beta(A_1) =_D \gamma(A_2)$.

Par construction, le langage utilisé pour S et G est clos par renommage et contient un nombre infini de variables. Puisque V, $var(G)$ et $var(C)$ sont des ensembles finis, il existe dans le langage une variante C' de C telle que $(V \cup var(G)) \cap var(C') = \emptyset$.

Soit π le renommage correspondant, nous avons alors $C' = (\pi(A_2) :\!- \pi(F_2) \;/\!/\; \pi(H_2))$.

Soit $H_3 = (H_1 \;\&\; \pi(H_2) \;\&\; \pi(A_2) \;\&\; (A_1 \stackrel{arg}{=} \pi(A_2)))$. Une (S,V)-réduction de G à partir de C' est possible (par réduction directe) si H_3 est satisfaisable.

Pour montrer que H_3 est satisfaisable, nous allons construire une solution de H_3.

Soit une assignation $\delta \in ASG^{\mathcal{B}}$ telle que :

$$\begin{aligned} \delta|_{V \cup var(G)} &= \beta|_{V \cup var(G)} \\ \delta|_{var(C')} &= (\gamma \circ \pi^{-1})|_{var(C')} \end{aligned}$$

δ existe car $(V \cup var(G)) \cap var(C') = \emptyset$, par construction de C'.

Pour montrer que H_3 est satisfaisable, nous montrons maintenant que $\delta|_{var(H_3)} \in [H_3]^{\mathcal{B}}_{var(H_3)}$.

$$\begin{aligned} \beta \in [G]^{\Omega} &\Rightarrow \beta \in [H_1]^{\mathcal{B}} \\ &\Rightarrow \delta|_{var(H_1)} \in [H_1]^{\mathcal{B}}_{var(H_1)} \qquad (1) \end{aligned}$$

$$\begin{aligned} \gamma \in [H_2 \;\&\; A_2]^{\mathcal{B}} &\Rightarrow (\gamma \circ \pi^{-1}) \in [\pi(H_2) \;\&\; \pi(A_2)]^{\mathcal{B}} \\ &\Rightarrow \delta|_{var(\pi(H_2 \;\&\; A_2))} \in [\pi(H_2) \;\&\; \pi(A_2)]^{\mathcal{B}}_{var(\pi(H_2 \;\&\; A_2))} \qquad (2) \end{aligned}$$

$$\begin{aligned} \beta(A_1) =_D \gamma(A_2) &\Rightarrow \beta(A_1) =_D (\gamma \circ \pi^{-1} \circ \pi)(A_2) \\ &\Rightarrow \beta(A_1) =_D (\gamma \circ \pi^{-1})(\pi(A_2)) \\ &\text{or } \delta|_{var(A_1)} = \beta|_{var(A_1)} \wedge \delta|_{var(\pi(A_2))} = (\gamma \circ \pi^{-1})|_{var(\pi(A_2))} \\ &\Rightarrow \delta|_{var(A_1 \;\&\; \pi(A_2))} \in [A_1 \stackrel{arg}{=} \pi(A_2)]^{\mathcal{B}}_{var(A_1 \;\&\; \pi(A_2))} \qquad (3) \end{aligned}$$

$$(1), (2) \text{ et } (3) \Rightarrow \delta|_{var(H_3)} \in [H_3]^{\mathcal{B}}_{var(H_3)} \qquad (4)$$

H_3 est donc satisfaisable.

Donc une (S,V)-réduction de G est possible par réduction directe à partir de C'.

Soit $G' = (\pi(F_2)\ \&\ F_1\ /\!/\ H_3)$, le but obtenu par cette réduction.

Dans ce cas 1, il nous reste à montrer que $\alpha \in [G']^{\Omega}_{V}$
et que $comp_V(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$.

$$\beta|_{var(F_1)} \in [F_1]^{\mathcal{I}}_{var(F_1)} \quad \Rightarrow \quad \delta|_{var(F_1)} \in [F_1]^{\mathcal{I}}_{var(F_1)} \tag{5}$$

$$\begin{aligned} \gamma|_{var(F_2)} \in [F_2]^{\mathcal{T}_{i-1}}_{var(F_2)} \quad &\Rightarrow \quad \delta|_{var(F_2)} \in [F_2]^{\mathcal{I}}_{var(F_2)} \\ \Rightarrow \ (\gamma \circ \pi^{-1})|_{var(\pi(F_2))} &\in [\pi(F_2)]^{\mathcal{I}}_{var(\pi(F_2))} \\ \Rightarrow \quad \delta|_{var(\pi(F_2))} &\in [\pi(F_2)]^{\mathcal{I}}_{var(\pi(F_2))} \end{aligned} \tag{6}$$

Sachant que $G' = (\pi(F_2)\ \&\ F_1\ /\!/\ H_3)$ et $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$,

(4), (5) et (6) $\quad \Rightarrow \quad \delta|_{var(G')} \in [G']^{\Omega}_{var(G')}$.

Or, par construction, $\delta|_V = \beta|_V = \alpha$

D'où $\boxed{\alpha \in [G']^{\Omega}_{V}}$

Pour montrer que $comp(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$, nous posons $j = comp_V(\alpha, G', \Omega)$.

$j \lll comp(\delta, G', \Omega)$

car j est minimal par définition de $comp_V$

$$\begin{aligned} &\Rightarrow \quad j \lll comp(\delta, \pi(F_2), \Omega) \cup comp(\delta, F_1, \Omega) \\ &\Rightarrow \quad j \lll comp(\delta, \pi(F_2), \Omega) \cup comp(\beta, F_1, \Omega) \qquad (7) \\ &\qquad \text{car } \delta|_{var(F_1)} = \beta|_{var(F_1)} \end{aligned}$$

$$\begin{aligned} \gamma|_{var(F_2)} \in [F_2]^{\mathcal{T}_{i-1}}_{var(F_2)} \quad &\Rightarrow \quad \delta|_{var(\pi(F_2))} \in [\pi(F_2)]^{\mathcal{T}_{i-1}}_{var(\pi(F_2))} \\ &\Rightarrow \quad comp(\delta, \pi(F_2), \Omega) \ll \{\!|i|\!\} \end{aligned} \tag{8}$$

$$
\begin{array}{lll}
\text{(7) et (8)} & \Rightarrow & j \ll \{\!\{i\}\!\} \cup comp(\beta, F_1, \Omega) \\
& \Rightarrow & j \ll \{\!\{comp(\beta, A_1, \Omega)\}\!\} \cup comp(\beta, F_1, \Omega) \\
& & \text{car } i = comp(\beta, A_1, \Omega) \\
& \Rightarrow & j \ll \{\!\{comp(\beta, A_1, \Omega)\}\!\} \cup comp(\beta, F_1 \mathbin{/\!/} H_1, \Omega) \\
& \Rightarrow & j \ll comp(\beta, G, \Omega) \\
& \Rightarrow & j \ll comp_V(\alpha, G, \Omega) \\
& & \text{car nous avons choisi l'assignation } \beta \text{ telle que} \\
& & \qquad comp(\beta, G, \Omega) = comp_V(\alpha, G, \Omega) \\
& \Rightarrow & \boxed{comp(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)}
\end{array}
$$

Cas 2 : $\beta(A_1) \in her\_ind_i$

Dans ce cas A_1 est de la forme $A_1 = (X < Y_1)$. Posons $\beta(A_1) = \langle d_1, d_2 \rangle$. Il existe alors une clause $C = (Z < Y_2 \text{ :- } F_2 \mathbin{/\!/} H_2)$ de S et une assignation γ telles que $\gamma \in [F_2]^{\mathcal{T}_{i-1}} \wedge \gamma \in [H_2 \ \& \ Z < Y_2]^{\mathcal{B}} \wedge \langle d_1, \gamma(Z) \rangle \in herite_{i-1} \wedge \gamma(Y_2) = d_2$.

Comme dans le cas 1, il existe dans le langage une variante C' de C telle que $(V \cup var(G)) \cap var(C') = \emptyset$.

Soit $C' = (\pi(Z < Y_2) \text{ :- } \pi(F_2) \mathbin{/\!/} \pi(H_2))$.

Soit $H_3 = (H_1 \ \& \ \pi(H_2) \ \& \ \pi(Z < Y_2) \ \& \ (Y_1 = \pi(Y_2)))$. Une (S, V)-réduction de G à partir de C' est possible (réduction par héritage) si H_3 est satisfaisable.

Comme dans le cas 1, on considère l'assignation $\delta \in ASG^{\mathcal{B}}$ telle que :

$$
\begin{array}{rcl}
\delta|_{V \cup var(G)} & = & \beta|_{V \cup var(G)} \\
\delta|_{var(C')} & = & (\gamma \circ \pi^{-1})|_{var(C')}
\end{array}
$$

Nous avons toujours :

$$\delta|_{var(H_1)} \in [H_1]^{\mathcal{B}}_{var(H_1)} \qquad (1)$$

$$\delta|_{var(\pi(H_2 \ \& \ A_2))} \in [\pi(H_2) \ \& \ \pi(A_2)]^{\mathcal{B}}_{var(\pi(H_2 \ \& \ A_2))} \qquad (2)$$

Avec $A_2 = (Z < Y_2)$.

De plus :

$$\begin{aligned}
\beta(Y_1) =_D \gamma(Y_2) \quad &\Rightarrow \quad \beta(Y_1) =_D (\gamma \circ \pi^{-1} \circ \pi)(Y_2) \\
&\Rightarrow \quad \beta(Y_1) =_D (\gamma \circ \pi^{-1})(\pi(Y_2)) \\
\text{or } \delta|_{\{Y_1\}} = \beta|_{\{Y_1\}} \wedge \delta|_{\{\pi(Y_2)\}} &= (\gamma \circ \pi^{-1})|_{\{\pi(Y_2)\}} \\
&\Rightarrow \quad \delta|_{\{Y_1,\pi(Y_2)\}} \in [Y_1 \stackrel{arg}{=} \pi(Y_2)]^{\mathcal{B}}_{\{Y_1,\pi(Y_2)\}} \qquad (3)
\end{aligned}$$

$$(1), (2) \text{ et } (3) \quad \Rightarrow \quad \delta|_{var(H_3)} \in [H_3]^{\mathcal{B}}_{var(H_3)}$$

H_3 est donc satisfaisable.

Donc une (S,V)-réduction de G est possible, par réduction par héritage à partir de C'.

Soit $G' = (X < \pi(Z)\ \&\ \pi(F_2)\ \&\ F_1 \ /\!/\ H_3)$ le but obtenu par cette réduction.

Dans ce cas 2, il nous reste à montrer que $\alpha \in [G']^{\Omega}_V$
et que $comp_V(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$.

Comme dans le cas 1, nous avons :

$$\delta|_{var(F_1)} \in [F_1]^{\mathcal{I}}_{var(F_1)} \qquad (4)$$

$$\delta|_{var(\pi(F_2))} \in [\pi(F_2)]^{\mathcal{I}}_{var(\pi(F_2))} \qquad (5)$$

$$\delta|_{var(H_3)} \in [H_3]^{\mathcal{B}}_{var(H_3)} \qquad (6)$$

De plus :

$$\begin{aligned}
\langle d_1, \gamma(Z)\rangle \in herite_{i-1} \quad &\Rightarrow \quad \langle d_1, \delta(\pi(Z))\rangle \in herite_{i-1} \\
&\Rightarrow \quad \langle \beta(X), \delta(\pi(Z))\rangle \in herite_{i-1} \\
&\Rightarrow \quad \langle \delta(X), \delta(\pi(Z))\rangle \in herite_{i-1} \qquad (7) \\
&\Rightarrow \quad \delta|_{\{X,\pi(Z)\}} \in [X < \pi(Z)]^{\mathcal{I}}_{\{X,\pi(Z)\}} \qquad (8)
\end{aligned}$$

Sachant que $G' = (X < \pi(Z)\ \&\ \pi(F_2)\ \&\ F_1 \ /\!/\ H_3)$ et $\Omega = \langle \mathcal{I}, \mathcal{B}\rangle$

$$(4), (5), (6) \text{ et } (8) \quad \Rightarrow \quad \delta|_{var(G')} \in [G']^{\Omega}_{var(G')}$$

$$\Rightarrow \quad \boxed{\alpha \in [G']^{\Omega}_V} \ (\text{car } \delta|_V = \beta|_V = \alpha)$$

Pour montrer que $comp(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$, nous posons $j = comp_V(\alpha, G', \Omega)$.

$j \lll comp(\delta, G', \Omega)$ car j est minimal par définition de $comp_V$

$$\Rightarrow \quad j \lll \{\!\{comp(\delta, X < \pi(Z), \Omega)\}\!\} \cup comp(\delta, \pi(F_2), \Omega) \cup comp(\delta, F_1, \Omega)$$

$$\Rightarrow \quad j \lll \{\!\{comp(\delta, X < \pi(Z), \Omega)\}\!\} \cup comp(\delta, \pi(F_2), \Omega) \cup comp(\beta, F_1, \Omega) \qquad (9)$$

car $\delta|_{var(F_1)} = \beta|_{var(F_1)}$

Nous avons toujours :

$$\gamma|_{var(F_2)} \in [F_2]^{\mathcal{T}_{i-1}}_{var(F_2)} \quad \Rightarrow \quad \delta|_{var(\pi(F_2))} \in [\pi(F_2)]^{\mathcal{T}_{i-1}}_{var(\pi(F_2))}$$
$$\Rightarrow \quad comp(\delta, \pi(F_2), \Omega) \ll \{\!\{i\}\!\} \qquad (10)$$

De plus :

$$(7) \quad \Rightarrow \quad \delta|_{\{X,\pi(Z)\}} \in [X < \pi(Z)]^{\mathcal{T}_{i-1}}_{\{X,\pi(Z)\}}$$
$$\Rightarrow \quad comp(\delta, X < \pi(Z), \Omega) \leq i - 1$$
$$\Rightarrow \quad \{\!\{comp(\delta, X < \pi(Z), \Omega)\}\!\} \ll \{\!\{i\}\!\} \qquad (11)$$

(9), (10) et (11) $\Rightarrow j \ll \{\!\{i\}\!\} \cup comp(\beta, F_1, \Omega)$

Par le même raisonnement que dans le cas 1, on déduit

$$\boxed{comp(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)}$$

Cas 3 : $\beta(A_1) \in mem\_ind_i$

Dans ce cas A_1 est de la forme $A_1 = (X : Y_1)$. Posons $\beta(A_1) = \langle d_1, d_2 \rangle$. Il existe alors une clause $C = (Z < Y_2 :\!\!- F_2 \;/\!/\; H_2)$ de S et une assignation γ telles que $\gamma \in [F_2]^{\mathcal{T}_{i-1}} \wedge \gamma \in [H_2 \;\&\; Z < Y_2]^{\mathcal{B}} \wedge \langle d_1, \gamma(Z) \rangle \in membre_{i-1} \wedge \gamma(Y_2) = d_2$.

Le raisonnement est identique à celui du cas 2, en utilisant la règle de réduction par inclusion à la place de de la règle de réduction par héritage.

Comme dans le cas 1, il existe dans le langage une variante C' de C telle que $(V \cup var(G)) \cap var(C') = \emptyset$.

Soit $C' = (\pi(Z < Y_2) \text{ :- } \pi(F_2) \ // \ \pi(H_2))$.

Soit $H_3 = (H_1 \ \& \ \pi(H_2) \ \& \ \pi(Z < Y_2) \ \& \ (Y_1 = \pi(Y_2)))$.

Comme dans le cas 1, on considère l'assignation $\delta \in ASG^{\mathcal{B}}$ telle que :

$$\begin{aligned} \delta|_{V \cup var(G)} &= \beta|_{V \cup var(G)} \\ \delta|_{var(C')} &= (\gamma \circ \pi^{-1})|_{var(C')} \end{aligned}$$

De la même façon que dans le cas 2, on déduit que H_3 est satisfaisable en montrant que $\delta|_{var(H_3)} \in [H_3]^{\mathcal{B}}_{var(H_3)}$.

Donc une (S, V)-réduction de G est possible, par réduction par inclusion à partir de C'.

Soit $G' = (X : \pi(Z) \ \& \ \pi(F_2) \ \& \ F_1 \ // \ H_3)$, le but obtenu par cette réduction.

Dans ce cas 3, il nous reste à montrer que $\alpha \in [G']^{\Omega}_V$
et que $comp_V(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$.

Comme dans le cas 1, nous avons :

$$\delta|_{var(F_1)} \in [F_1]^{\mathcal{I}}_{var(F_1)} \tag{1}$$

$$\delta|_{var(\pi(F_2))} \in [\pi(F_2)]^{\mathcal{I}}_{var(\pi(F_2))} \tag{2}$$

$$\delta|_{var(H_3)} \in [H_3]^{\mathcal{B}}_{var(H_3)} \tag{3}$$

De plus :

$$\begin{aligned} \langle d_1, \gamma(Z)\rangle \in membre_{i-1} &\Rightarrow \langle d_1, \delta(\pi(Z))\rangle \in membre_{i-1} \\ &\Rightarrow \langle \beta(X), \delta(\pi(Z))\rangle \in membre_{i-1} \\ &\Rightarrow \langle \delta(X), \delta(\pi(Z))\rangle \in membre_{i-1} \quad (4) \\ &\Rightarrow \delta|_{\{X,\pi(Z)\}} \in [X < \pi(Z)]^{\mathcal{I}}_{\{X,\pi(Z)\}} \quad (5) \end{aligned}$$

Sachant que $G' = (X : \pi(Z) \ \& \ \pi(F_2) \ \& \ F_1 \ // \ H_3)$ et $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$

$(1), (2), (3)$ et $(5) \quad \Rightarrow \quad \delta|_{var(G')} \in [G']^{\Omega}_{var(G')}$

$$\Rightarrow \quad \boxed{\alpha \in [G']^{\Omega}_V} \ (\text{car } \delta|_V = \beta|_V = \alpha)$$

Pour montrer que $comp(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$, nous posons $j = comp_V(\alpha, G', \Omega)$.

$j \lll comp(\delta, G', \Omega)$ car j est minimal par définition de $comp_V$

$$\Rightarrow \quad j \lll \{\!\{comp(\delta, X : \pi(Z), \Omega)\}\!\} \cup comp(\delta, \pi(F_2), \Omega) \cup comp(\delta, F_1, \Omega)$$

$$\Rightarrow \quad j \lll \{\!\{comp(\delta, X : \pi(Z), \Omega)\}\!\} \cup comp(\delta, \pi(F_2), \Omega) \cup comp(\beta, F_1, \Omega) \qquad (6)$$

car $\delta|_{var(F_1)} = \beta|_{var(F_1)}$

Nous avons toujours :

$$\gamma|_{var(F_2)} \in [F_2]^{\mathcal{T}_{i-1}}_{var(F_2)} \quad \Rightarrow \quad \delta|_{var(\pi(F_2))} \in [\pi(F_2)]^{\mathcal{T}_{i-1}}_{var(\pi(F_2))}$$
$$\Rightarrow \quad comp(\delta, \pi(F_2), \Omega) \ll \{\!\{i\}\!\} \qquad (7)$$

De plus :

$$(4) \quad \Rightarrow \quad \delta|_{\{X,\pi(Z)\}} \in [X : \pi(Z)]^{\mathcal{T}_{i-1}}_{\{X,\pi(Z)\}}$$
$$\Rightarrow \quad comp(\delta, X : \pi(Z), \Omega) \leq i - 1$$
$$\Rightarrow \quad \{\!\{comp(\delta, X : \pi(Z), \Omega)\}\!\} \ll \{\!\{i\}\!\} \qquad (8)$$

(6), (7) et (8) $\Rightarrow j \ll \{\!\{i\}\!\} \cup comp(\beta, F_1, \Omega)$

Par le même raisonnement que dans le cas 1, on déduit

$$\boxed{comp(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)}$$

Fin des cas

□

Rappelons que $\xrightarrow{r}{}^*_{S,V}$ est la fermeture reflexo-transitive de $\xrightarrow{r}_{S,V}$.

Théorème 7.8 *Soient $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle minimal d'un ensemble de clauses S, G un but et $\alpha \in [G]^{\Omega}_{V}$; il existe alors une conjonction de contraintes H telle que $G \xrightarrow{r}{}^*_{S,V} \epsilon \;/\!/\; H$ et $\alpha \in [\epsilon \;/\!/\; H]^{\Omega}_{V}$*

Preuve :

Cette démonstration s'effectue par induction sur $comp_V(\alpha, G, \Omega)$ en utilisant l'ordre bien fondé $\ll$.

Soit $cpxt_m$ une complexité (pour un but). Nous supposons que la propriété est vraie lorsque $comp_V(\alpha, G, \Omega) \ll cpxt_m$ et nous montrons alors qu'elle est aussi vraie lorsque $comp_V(\alpha, G, \Omega) = cpxt_m$.

Hypothèse d'induction :
Pour tous les $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$ qui sont des modèles minimaux d'un ensemble de clauses S, pour tous les buts G et tous les $\alpha \in [G]^{\Omega}_V$, si $comp_V(\alpha, G, \Omega) \ll cpxt_m$ alors il existe une conjonction de contraintes H telle que $G \stackrel{r}{\rightarrow}^*_{S,V} \epsilon \mathbin{/\!/} H$ et $\alpha \in [\epsilon \mathbin{/\!/} H]^{\Omega}_V$.

Montrons que la propriété est vraie aussi pour les complexités égales à $cpxt_m$.

Soient $\Omega = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle minimal d'un ensemble de clauses S, G un but et $\alpha \in [G]^{\Omega}_V$, tels que $comp_V(\alpha, G, \Omega) = cpxt_m$.

Cas 1 : G est de la forme $\epsilon \mathbin{/\!/} H_1$.

Nous avons $G \stackrel{r}{\rightarrow}^*_{S,V} G$ et $\alpha \in [G]^{\Omega}_V$. C'est-à-dire $G \stackrel{r}{\rightarrow}^*_{S,V} \epsilon \mathbin{/\!/} H_1$ et $\alpha \in [\epsilon \mathbin{/\!/} H_1]^{\Omega}_V$. H_1 est la conjonction H désirée.

Cas 2 : G est de la forme $A \,\&\, F \mathbin{/\!/} H_1$.

D'après le théorème 7.7, nous savons qu'il existe G' tel que $G \stackrel{r}{\rightarrow}_{S,V} G'$ avec $\alpha \in [G']^{\Omega}_V$ et $comp_V(\alpha, G', \Omega) \ll comp_V(\alpha, G, \Omega)$.

Par hypothèse d'induction, il existe une conjonction de contraintes H telle que $G' \stackrel{r}{\rightarrow}^*_{S,V} \epsilon \mathbin{/\!/} H$ et $\alpha \in [\epsilon \mathbin{/\!/} H]^{\Omega}_V$.

Enfin, puisque $G \stackrel{r}{\rightarrow}^*_{S,V} G'$, nous avons $G \stackrel{r}{\rightarrow}^*_{S,V} \epsilon \mathbin{/\!/} H$.

Fin des cas

□

Nous montrons maintenant que toutes les solutions d'une réponse correcte sont solutions de réponses calculées.

Théorème 7.9 (Complétude) *Soient S un ensemble de clauses, G un but et H_1 une conjonction de contraintes. Soient V un ensemble de variables et $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle de S.*

Si $S \models H_1 \Rightarrow G$ et H_1 est satisfaisable, alors pour tous les $\alpha \in [H_1]^{\mathcal{B}}_V$, il existe une conjonction de contraintes H_2 telle que $G \stackrel{r}{\rightarrow}^*_{S,V} \epsilon \mathbin{/\!/} H_2$ et $\alpha \in [H_2]^{\mathcal{B}}_V$

Preuve :

Soient S un ensemble de clauses, G un but et H_1 une conjonction de contraintes tels que $S \models H_1 \Rightarrow G$ et H_1 satisfaisable.

Soient V un ensembles de variables, $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ un modèle de S, et $\alpha \in [H_1]^{\mathcal{B}}_V$.

Nous montrons tout d'abord qu'il existe un modèle minimal Ω de S basé sur $\mathcal{B}$ (i.e., Ω est de la forme $\langle \mathcal{I}_m, \mathcal{B} \rangle$).

Nous définissons un ordre bien fondé $<$ sur les interprétations.

Soient Γ_1 et Γ_2 deux interprétations, alors $\Gamma_1 < \Gamma_2 \Leftrightarrow \Gamma_1 \leq \Gamma_2 \wedge \Gamma_1 \neq \Gamma_2$ (avec $\leq$ la relation d'ordre partiel définie sur les interprétations à la section 4).

$<$ est bien fondé car $\subset$ est bien fondé.

Soit Δ l'ensemble des modèles de S basés sur $\mathcal{B}$. Δ n'est pas vide car $\Gamma \in \Delta$. Puisque $<$ est bien fondé sur l'ensemble des interprétations, il existe un élément $\Omega \in \Delta$ qui est "minimal" pour $<$ (i.e., $\forall \Omega' \in \Delta, \Omega' \not\leq \Omega$). Ω est aussi minimal pour $\leq$.

Donc il existe un modèle minimal Ω de S basé sur $\mathcal{B}$.

Puisque $\alpha \in [H_1]^{\mathcal{B}}_V$ et $S \models H_1 \Rightarrow G$, nous avons alors $\alpha \in [G]^{\Omega}_V$.

D'après le théorème 7.7 nous savons qu'il existe H_2 tel que $G \xrightarrow{r}{}^{*}_{S,V} \epsilon \;/\!/\; H_2$ et $\alpha \in [H_2]^{\mathcal{B}}_V$. □

Troisième partie

Application

This part is an extended version of [29].

Abstract

We propose a framework for semantic query optimization in deductive object-oriented databases. The intentional database is described by means of clauses and a more restricted language is used for the integrity constraints. We apply a specific resolution and a classification mechanism to rewrite a query into a less expensive yet equivalent one. The main contribution of this work is to show how resolution and classification can be used together within a common framework to perform complementary semantic query optimizations in deductive object-oriented databases.

1 Introduction

In object-oriented database management systems, query optimization is performed by transforming algebraic expressions and by taking into account the physical representation of data [53][21]. Semantic query optimization takes advantage of the semantic knowledge (e.g., integrity constraints) about the content of databases to reformulate a query into a less expensive yet equivalent query.

A number of techniques have been followed to perform semantic query optimization in classical deductive databases (see [20]). Recently, this problem has been addressed in the context of deductive object-oriented databases. Two different approaches have been proposed: the first, described in [55], is an extension of [20] for deductive object-oriented databases. It uses a specific resolution to reason about clauses. The main advantage of this approach is that these clauses allow to take into account general descriptions of the intensional database. The second approach [10][12][17] uses a restricted language to describe the intensional database and the integrity constraints. This permits the use of a classification mechanism based on a decidable subsumption relation. This decidable relation helps to discover equivalences and simplifications that cannot be computed in the first framework, where this subsumption relation is undecidable in general [52].

To our knowledge, no formal framework has been proposed to combine these two promising approaches. This work is a contribution in this direction. We present a reformulation scheme that uses a resolution method and a classification-based rewriting in complementary ways. We propose three types of reformulation:

Resolution: we reduce some atoms referring to the intensional database to atoms referring to the extensional database, and we remove parts of

the query that cannot have any answers with respect to the integrity constraints.

Factorization and classification [40]: we simplify the query by eliminating some redundant selection conditions. In addition, we reformulate the query so that it takes advantage of some materialized views.

Propagation: we make explicit a part of the knowledge which is implicitly available in the query. In this way, we have more information about objects to be retrieved. This allows the search space of objects which will be considered during the query evaluation process to be reduced.

These reformulations can be applied in stages or iteratively. They can also be integrated within a query evaluation process. These aspects will not be developed.

We use a specific language which will be described in Section 4. However, the reformulation scheme can be applied to other languages for deductive object-oriented databases.

The kind of databases we consider is outlined in the next Section. In Section 3 we give an example of a semantic query optimization in our framework. Section 4 formally defines the language used in this part. The different query reformulations are detailed in Section 5. Then, we conclude in Section 6. Proofs of the theorems of Section 5 are given in the Appendix.

2 Context

In this section, by means of an example, we delimit the kind of database we consider. We also give an informal introduction to the knowledge representation language we use.

A deductive object-oriented database is split into three parts: $\mathcal{SCH}$, $\mathcal{EDB}$ and $\mathcal{IDB}$.
$\mathcal{SCH}$ is the database schema. It describes the classes used in the extensional database, the integrity constraints, and the materialized views. It will be seen as a set of constraints to be satisfied.
$\mathcal{EDB}$ corresponds to the extensional database.
$\mathcal{IDB}$ is a set of clauses that defines the intensional part of the database.

Basically, two forms of atoms are used in the description of the schema: $X \sqsubseteq C$ (the instances of class X have the properties C), $X := C$ (definition of the materialized view X).

Three forms of atoms are used in the clauses: $X < Y$ (X inherits from Y), $X : Y$ (X is an instance of Y), $X.R \rightarrow Y$ (for X, one of the values of the attribute R is Y).

A clause has the form :
A :– A_{idb_1} & ... & A_{idb_n} // A_{edb_1} & ... & A_{edb_m}.
In the body of the clause, we make a syntactic difference between a conjunction of atoms A_{idb_1} & ... & A_{idb_n}, that refers to the intensional database, and a conjunction of atoms A_{edb_1} & ... & A_{edb_m}, that refers to the extensional database.

We shall now comment on the following example, which will be used throughout this part. It describes the possible structure of a laboratory in a university.

$\mathcal{SCH}$:

(s1) person $\sqsubseteq$ anything

(s2) permanent_staff_member $\sqsubseteq$ person

(s3) teacher $\sqsubseteq$ person

(s4) student $\sqsubseteq$ and(person, all(works_in_project, project), mono(works_in_project))

(s5) professor $\sqsubseteq$ and(permanent_staff_member, teacher)

(s6) lecturer $\sqsubseteq$ and(permanent_staff_member, teacher, not(professor))

(s7) project $\sqsubseteq$ and(all(managed_by, permanent_staff_member), exist(managed_by, permanent_staff_member))

(s8) theme $\sqsubseteq$ and(all(managed_by, professor), exist(managed_by, permanent_staff_member), mono(managed_by))

(s9) assistant := and(student, teacher)

$\mathcal{IDB}$:

(r1) E:satisfied :– Z:important // Z.managed_by $\rightarrow$ E & Z:theme.

(r2) E:satisfied :– X.supervised_by $\rightarrow$ E // X:teacher.

(r3) very_satisfied $<$ satisfied.

(r4) X.supervised_by $\rightarrow$ E :– E:entitled // X.works_in_project $\rightarrow$ Z & Z.managed_by $\rightarrow$ E & X:student.

(r5) E:very_satisfied :– ...

(r6) E:entitled :– ...

(r7) Z:important :– ...

$\mathcal{EDB}$:

(e1) deductive_databases:theme

(e2) deductive_databases.managed_by $\rightarrow$ sam

(e3) sam:professor

The schema $\mathcal{SCH}$ asserts that person is a subclass of the universal class anything *(s1)* and that permanent_staff_member, teacher and student are subclasses of person *(s2)(s3)(s4)*. *(s4)* also tells us that for a student, the attribute works_in_project is of type project (constructor all) and it has at most one value (constructor mono).

A professor is a permanent_staff_member and a teacher *(s5)*. A lecturer is also a permanent_staff_ member and a teacher, but not a professor *(s6)*. A research project is managed by at least one (constructor exist) permanent_staff_member and only by permanent_staff_members *(s7)*. A theme is managed by one and only one permanent_staff_member, who must be a professor *(s8)*.

The description *(s9)* defines the materialized view assistant as the intersection of the classes student and teacher.

In the intensional database, the instances of the class satisfied are all the managers of the important themes *(r1)* and all the supervisors of the teachers *(r2)*. The Rule *(r4)* defines the derived attribute supervised_by. It states that a student's supervisor is an entitled manager of a project in which the student is involved. *(r3)* is a clause with an empty body that describes an inheritance link between two classes (very_satisfied and satisfied) of the intensional database.

The derivation rules of classes very_satisfied, entitled and important *(r5) (r6) (r7)* are left blank since they will not be used.

Finally, the extensional database tells us that deductive_databases is a research theme *(e1)* which is managed by sam *(e2)* who is a professor *(e3)*. Note that $\mathcal{EDB}$ will be in fact of no use during semantic query optimization.

3 Reformulation Principle

In our approach, the main query reformulations are performed using a resolution and a classification mechanism. Before we develop the technical details, it is important that we give the reader an informal description of the query reformulations.

A query has the same form as a clause body. Let Q be the following query:

E:satisfied // E:lecturer

which retrieves all satisfied lecturers. Figure 4 shows some possible reformulations of Q.

The resolution uses the clauses of $\mathcal{IDB}$ and the schema $\mathcal{SCH}$ to rewrite Q into a disjunction of queries (Q_1 *or* ... *or* Q_n), which is semantically equivalent[22] to Q.

By reformulating Q with clause *(r1)* we should obtain the query Q_1. But when we look at the part of Q_1 that refers to the extensional database:

$$\text{E:lecturer} \ \& \ \text{Z:theme} \ \& \ \text{Z.managed\_by} \rightarrow \text{E}$$

and since the schema imposes that all themes are managed by professors *(s8)* and that a professor is not a lecturer *(s6)*, Q_1 will always have an empty

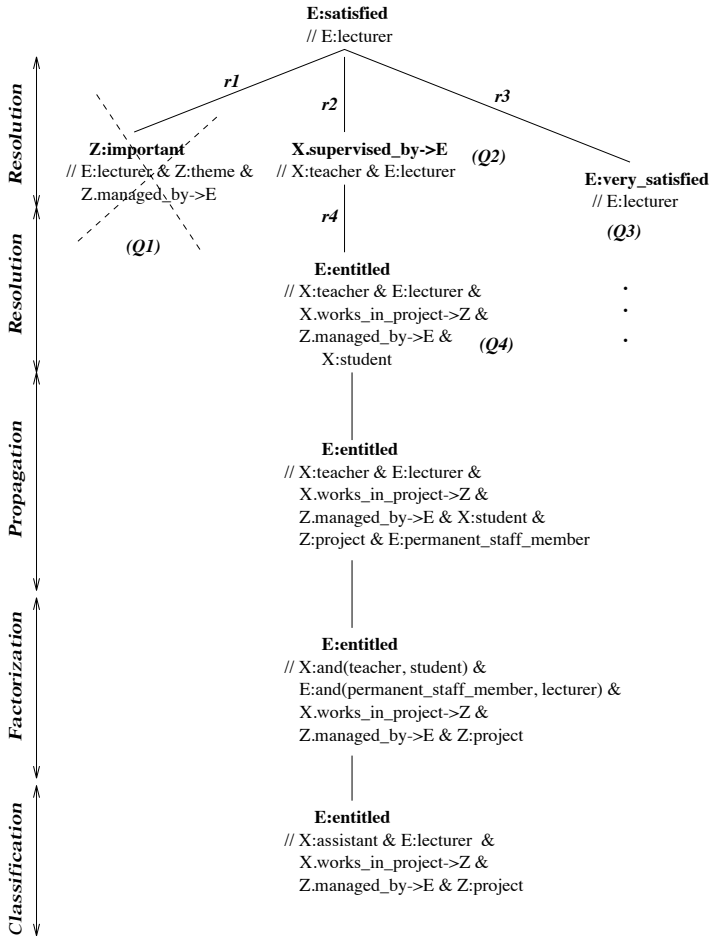

Figure 4: Reformulation of a query.

set of answers. A satisfiability test (with respect to the schema) applied to this part of the query during the resolution step allows us to detect such contradictions, and then to eliminate Q_1.

[22]Informally, two queries are semantically equivalent (with respect to $\mathcal{IDB}$ and $\mathcal{SCH}$) if they have the same answers for all databases constructed from $\mathcal{IDB}$ and $\mathcal{SCH}$ [20]

The reformulation of the query Q with the clause *(r2)* generates the query Q_2. From Q we obtain Q_3 using clause *(r3)* and a specific resolution rule that handles inheritance links.

No other direct resolution steps can be performed to reformulate Q with a clause of $\mathcal{IDB}$. So Q is equivalent to the disjunction (Q_2 *or* Q_3).

Another resolution-based reformulation step can be performed. For example, the reformulation of the query Q_2 with the clause *(r4)* produces Q_4:

E:entitled // X:teacher & X:student & E:lecturer &
X.works_in_project → Z & Z.managed_by → E

The propagation-based reformulation is used on Q_4 to add selection information to the part that refers to the $\mathcal{EDB}$, and, thus, helps to reduce the search space of objects. For example, when looking at the schema, we see that a student is involved in a project *(s4)*, which is managed by permanent_staff_members *(s7)*. Thus, using the atoms X.works_in_project → Z and Z.managed_by → E in the query, we can add Z:project and E:permanent_staff_member.

Finally, the example illustrates the classification-based reformulation. This reformulation is performed in two steps: factorization and classification-based rewriting. The key idea behind the factorization is to put together, for each variable, instantiation links in which the variable appears. For example, from X:teacher and X:student, we obtain X:and(teacher, student).

The classification consists in finding the simplest and most specific description of the objects to be retrieved. The classification of the description and(permanent_staff_member, lecturer) allows us to deduce that E:and(permanent_staff_member, lecturer) can be simplified by rewriting it as E:lecturer. The classification of and(teacher, student) leads to a replacement of X:and(teacher, student) by X:assistant, and, thus, takes advantage of the materialized view assistant.

The ordering of the reformulations used in this example is not determined by our reformulation scheme. This ordering is out of the scope of this work.

4 Language

We use a knowledge representation language designed to allow the combination of two complementary kinds of reasoning: abduction and deduction of objects structural properties (see part 2). The attentive reader will notice that the syntax and the semantics are slightly richer than those used in the examples of this part. This is a language of clauses with constraints

in the sense of constraint logic programming [31]. Its semantics is drawn from F-logic [37], terminological logics [46] and the generalized framework for constraint logic programming proposed by Höhfeld and Smolka [30].

First, we present its syntax, its declarative semantics, and a resolution method. Then, we use the language to describe databases and queries. Informally, in a logical framework for deductive databases, one can see a model as a set Ω of true atoms. In this our case, we see such a model Ω as two sets $\mathcal{I}$ and $\mathcal{B}$, where $\mathcal{I} \subseteq \Omega \subseteq \mathcal{B}$.

$\mathcal{B}$ is a set of possibly true atoms with respect to the database $\mathcal{SCH}$+$\mathcal{IDB}$ (hence $\Omega \subseteq \mathcal{B}$) and $\mathcal{I}$ is a set of necessarily true atoms with respect to the intensional database $\mathcal{IDB}$ (hence $\mathcal{I} \subseteq \Omega$).

This will be used in two ways, when reasoning to reformulate a query Q. Firstly, since we do not want to access the $\mathcal{EDB}$, we will use $\mathcal{B}$ as an hypothetical denotation of the database. And secondly, since we allow to access the $\mathcal{IDB}$, we will use $\mathcal{I}$, as a necessary denotation of the $\mathcal{IDB}$ such that the query Q will have answers.

4.1 Syntax

Let $\mathcal{K}$ be a decidable set of symbols of constants and $\mathcal{V}$ be a decidable, infinite set of symbols of variables. *Terms* are of two kinds: *object names* (names of classes, of instances or of attributes) and *concepts*.

Terms are obtained according to the following abstract syntax:

$$\begin{array}{lcl} X & \xrightarrow{def} & v \mid k \\ C & \xrightarrow{def} & X \mid \mathit{anything} \mid \mathit{nothing} \mid \mathit{and}(C_1, \ldots, C_n) \mid \\ & & \mathit{all}(R, C) \mid \mathit{mono}(R) \mid \mathit{exist}(R, C) \mid \mathit{not}(X) \end{array}$$

where v denotes a variable, k denotes a constant, X and R denote object names, whereas C and C_i (for $i = 1, \ldots, n$) denote concepts.

Atoms are also of two kinds: *atomic links* and *atomic constraints*. The syntax for forming atoms is the following (X, Y and R denote object names, C denotes a concept, Al is an atomic link and Ac is an atomic constraint):

$$\begin{array}{lcl} Al & \xrightarrow{def} & X < Y \mid X : Y \mid X.R \rightarrow Y \\ Ac & \xrightarrow{def} & X < Y \mid X : C \mid X.R \rightarrow Y \mid X := C \mid X \sqsubseteq C \mid X = Y \end{array}$$

Sometimes, we will put atoms in brackets in order to facilitate reading.

Goals and clauses are built according to the following syntax:

$$
\begin{aligned}
F &\xrightarrow{def} \epsilon \mid Al \mid Al_1 \ \& \ \ldots \ \& \ Al_n \\
H &\xrightarrow{def} \epsilon \mid Ac \mid Ac_1 \ \& \ \ldots \ \& \ Ac_m \\
Goal &\xrightarrow{def} F \ /\!/ \ H \\
Clause &\xrightarrow{def} Al \ :\!- \ F \ /\!/ \ H
\end{aligned}
$$

where Al and Al_i (for $i = 1, \ldots, n$) denote atomic links, Ac and Ac_i (for $i = 1, \ldots, m$) denote atomic constraints, F denotes a conjunction of atomic links, H denotes a conjunction of atomic constraints and ϵ the empty conjunction of constraints.

We use the conventional designations of clause *body* and *head*. A fact is a clause whose body is $\epsilon \ /\!/ \ \epsilon$. It is abbreviated by writing only the head of the clause. Finally, variable symbols start with an upper case letter and constant symbols start with a lower case letter.

A *language* $\mathcal{L}$ is the set of goals and clauses constructed from the two sets of symbols $\mathcal{K}$ and $\mathcal{V}$.

4.2 Declarative Semantics

Let $\mathcal{L}$ be a language constructed from the set $\mathcal{V}$ of variables and the set $\mathcal{K}$ of constants.

We use two levels of interpretation: *semi-interpretations* for conjunctions of atoms and *interpretations* for goals and clauses.

A semi-interpretation $\mathcal{J}$ of $\mathcal{L}$ is of the form:

$$\mathcal{J} = \langle D, id, member, inherit, attribute \rangle, \text{ with}$$

- D a non-empty set called the domain of $\mathcal{J}$.
- id a total injection $id : \mathcal{K} \to D$. It assigns an element of the domain to each constant (name of a class, of an instance or of an attribute).
- $member \subseteq D \times D$, a relation linking instances to their classes.
- $inherit \subseteq D \times D$, a strict ordering relation linking classes to their superclasses.
- $attribute \subseteq D \times D \times D$, a relation where every occurrence links an instance, an attribute name and a value for this attribute.
- $member$ and $inherit$ satisfying:

$\forall d_1, d_2, d_3 \in D, \langle d_1, d_2 \rangle \in member \wedge \langle d_2, d_3 \rangle \in inherit \Rightarrow \langle d_1, d_3 \rangle \in member$

(i.e., instances of a class are also instances of its superclasses.)

Let $\mathcal{J}$ be a semi-interpretation of $\mathcal{L}$. A *$\mathcal{J}$-assignment* α is a total function $\alpha : \mathcal{V} \to D$, which assigns elements of the domain of $\mathcal{J}$ to the variables of the language. We write $ASG^{\mathcal{J}}$ for the set of all $\mathcal{J}$-assignments. All the $\mathcal{J}$-assignments $\alpha \in ASG^{\mathcal{J}}$ are extended to $\mathcal{V} \cup \mathcal{K}$ by *id*: $\forall k \in \mathcal{K}, \alpha(k) := id(k)$.

For a semi-interpretation $\mathcal{J} = \langle D, id, member, inherit, attribute \rangle$ and an assignment α, we define a function $ext_{\mathcal{J}\alpha}$ which assigns a set of elements of the domain to every concept. This function is defined by the following equations:

$$\begin{aligned}
ext_{\mathcal{J}\alpha}(X) &= \{d \in D \mid \langle d, \alpha(X) \rangle \in member\} \\
ext_{\mathcal{J}\alpha}(anything) &= D \\
ext_{\mathcal{J}\alpha}(nothing) &= \emptyset \\
ext_{\mathcal{J}\alpha}(and(C_1, \ldots, C_n)) &= ext_{\mathcal{J}\alpha}(C_1) \cap \ldots \cap ext_{\mathcal{J}\alpha}(C_n) \\
ext_{\mathcal{J}\alpha}(all(R, C)) &= \{d_1 \in D \mid \forall d_2 \in D, \langle d_1, \alpha(R), d_2 \rangle \in attribute \Rightarrow \\
&\qquad d_2 \in ext_{\mathcal{J}\alpha}(C)\} \\
ext_{\mathcal{J}\alpha}(mono(R)) &= \{d_1 \in D \mid \forall d_2, d_3 \in D, \langle d_1, \alpha(R), d_2 \rangle \in attribute \wedge \\
&\qquad \langle d_1, \alpha(R), d_3 \rangle \in attribute \Rightarrow d_2 =_D d_3\} \\
ext_{\mathcal{J}\alpha}(exist(R, C)) &= \{d_1 \in D \mid \exists d_2 \in D, \langle d_1, \alpha(R), d_2 \rangle \in attribute \wedge \\
&\qquad d_2 \in ext_{\mathcal{J}\alpha}(C)\} \\
ext_{\mathcal{J}\alpha}(not(X)) &= \{d \in D \mid d \notin ext_{\mathcal{J}\alpha}(X)\}
\end{aligned}$$

where X and R are object names, C and C_i (for $i = 1, \ldots, n$) are concepts.
$ext_{\mathcal{J}\alpha}$ corresponds to the interpretation of concepts in terminological logics [46].

The function $ext_{\mathcal{J}\alpha}$ allows us to define the notion of *$\mathcal{J}$-solutions* [30] of an atom and of a conjunction of atoms. The set of $\mathcal{J}$-solutions of an atom A will be written $[A]^{\mathcal{J}}$.

For a given semi-interpretation $\mathcal{J}$, the sets of $\mathcal{J}$-solutions of the different forms of atoms are:

$$\begin{aligned}
[X : C]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \alpha(X) \in ext_{\mathcal{J}\alpha}(C)\} \\
[X.R \to Y]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \langle \alpha(X), \alpha(R), \alpha(Y) \rangle \in attribute\} \\
[X < Y]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \langle \alpha(X), \alpha(Y) \rangle \in inherit\} \\
[X \sqsubseteq C]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid ext_{\mathcal{J}\alpha}(X) \subseteq ext_{\mathcal{J}\alpha}(C)\} \\
[X := C]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid ext_{\mathcal{J}\alpha}(X) = ext_{\mathcal{J}\alpha}(C)\} \\
[X = Y]^{\mathcal{J}} &:= \{\alpha \in ASG^{\mathcal{J}} \mid \alpha(X) =_D \alpha(Y)\}
\end{aligned}$$

$$[\epsilon]^{\mathcal{J}} \quad := \quad ASG^{\mathcal{J}}$$

The notion of set of $\mathcal{J}$-solutions is extended to conjunctions of atoms:

$$[A_1 \ \& \ \ldots \ \& \ A_n]^{\mathcal{J}} = [A_1]^{\mathcal{J}} \cap \ldots \cap [A_n]^{\mathcal{J}}.$$

The semi-interpretations of a given language $\mathcal{L}$ are partially ordered by $\leq$ as follows:

$$\mathcal{J}_1 \leq \mathcal{J}_2 \Leftrightarrow \text{for every atom } A \text{ of } \mathcal{L}, [A]^{\mathcal{J}_1} \subseteq [A]^{\mathcal{J}_2}.$$

An *interpretation* Γ of $\mathcal{L}$ is of the form $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ where $\mathcal{I}$ and $\mathcal{B}$ are semi-interpretations of $\mathcal{L}$ such that: $\mathcal{I} \leq \mathcal{B}$. The semi-interpretation $\mathcal{B}$ is called the base of Γ. The set of Γ-*assignments* is $ASG^{\Gamma} = ASG^{\mathcal{B}}$.

We define a partial ordering relation on interpretations: let $\Gamma_1 = \langle \mathcal{I}_1, \mathcal{B}_1 \rangle$ and $\Gamma_2 = \langle \mathcal{I}_2, \mathcal{B}_2 \rangle$ be two interpretations, then $\Gamma_1 \leq \Gamma_2$ iff $\mathcal{I}_1 \leq \mathcal{I}_2$ and $\mathcal{B}_1 = \mathcal{B}_2$.

We now extend the notion of solution to goals and clauses. Let $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ be an interpretation, we define the following sets of Γ-*solutions*:

$$\begin{array}{rcl} [F \ /\!/ \ H]^{\Gamma} & := & [F]^{\mathcal{I}} \cap [H]^{\mathcal{B}} \\ [A \ :\!- \ F \ /\!/ \ H]^{\Gamma} & := & (ASG^{\Gamma} - [F \ /\!/ \ H \ \& \ A]^{\Gamma}) \cup [A]^{\mathcal{I}} \end{array}$$

A more classical point of view for the second definition would have been:

$$[A \ :\!- \ F \ /\!/ \ H]^{\Gamma} \quad := \quad (ASG^{\Gamma} - [F \ /\!/ \ H]^{\Gamma}) \cup [A]^{\mathcal{I}}$$

which requires that if $\alpha \in [A \ :\!- \ F \ /\!/ \ H]^{\Gamma}$ is a solution of $F \ /\!/ \ H$, then α must be a solution of A.

In our definition, the requirement is weaker: for $\alpha \in [A \ :\!- \ F \ /\!/ \ H]^{\Gamma}$, α must be a solution of A only if α is a solution of $F \ /\!/ \ H \ \& \ A$ (i.e., the body is satisfied and the head seen as a constraint is also satisfied).

A conjunction of atomic constraints H is *satisfiable* iff there exists at least one semi-interpretation $\mathcal{J}$ such that $[H]^{\mathcal{J}} \neq \emptyset$.

An interpretation Γ is a *model* of a set of clauses S iff $\forall C \in S, [C]^{\Gamma} = ASG^{\Gamma}$.

The notion of logical consequence is defined as follows:
let S be a set of clauses, H be a conjunction of atomic constraints and

Q be a goal. Q is a *logical consequence of* S *under hypothesis* H (written $S \models H \Rightarrow Q$) iff for every model $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ of S, we have $[H]^{\mathcal{B}} \subseteq [Q]^{\Gamma}$.

We can now define the notion of answer. Let S be a set of clauses, then an *answer* for a goal Q is a conjunction of atomic constraints H, such that H is satisfiable and $S \models H \Rightarrow Q$.

Informally, an answer is often seen as a substitution on the variables of the goal. Here, an answer can be seen as a formula that constrains these variables. For example, when a classical answer is the substitution $\{X/a, Y/Z\}$, we have an answer of the form $X = a \ \& \ Y = Z$. In this aspect, our definitions of answer and of logical consequence are in the spirit of [19] and are also related to constraint logic programming[23] [31].

4.3 Resolution Method

We now formalize our resolution method by defining a relation of goal reduction [30].

Preliminary Notions

Let F be a conjunction of atoms, a goal or a clause. We write $var(F)$ the set of variables in F.

Two atoms A and A' are said to be *compatible* iff :

(1) $A = (X : Y)$ and $A' = (X' : Y')$. *In this case we write* $(A \stackrel{arg}{=} A')$ *for the conjunction of constraints:* $(X = X') \ \& \ (Y = Y')$.

or (2) $A = (X < Y)$ and $A' = (X' < Y')$. *We also write in this case* $(A \stackrel{arg}{=} A')$ *for the conjunction of constraints:* $(X = X') \ \& \ (Y = Y')$.

or (3) $A = (X.R \rightarrow Y)$ and $A' = (X'.R' \rightarrow Y')$. *In this last case we write* $(A \stackrel{arg}{=} A')$ *for the conjunction of constraints:* $(X = X') \ \& \ (R = R') \ \& \ (Y = Y')$.

A clause C obtained by a renaming of the variables of a clause C' is called a *variant* of C'.

Resolution

For a given language $\mathcal{L}$, let S be a set of clauses, and V be a finite set of variables.

[23] A classical constraint logic programming scheme uses a fixed interpretation for the atomic constraints, and, thus, will not be appropriate here.

The (S,V)-*reduction* is a binary relation $\stackrel{r}{\rightarrow}_{S,V}$ on the set of goals. It is defined by the following rules, of the form $Q \stackrel{r}{\rightarrow}_{S,V} Q'$ (Where C denotes a variant of a clause of S such that $(V \cup var(Q)) \cap var(C) = \emptyset$):

Direct Reduction

$$A \ \& \ G \ /\!/ \ H_1 \ \stackrel{r}{\rightarrow}_{S,V} \ F \ \& \ G \ /\!/ \ H_3$$
with $C = (A' \coloneq F \ /\!/ \ H_2)$ and
$H_3 = (H_1 \ \& \ H_2 \ \& \ A' \ \& \ (A \stackrel{arg}{=} A'))$,
if A and A' are compatible
and H_3 is satisfiable.

Reduction-by-inclusion

$$X : Y \ \& \ G \ /\!/ \ H_1 \ \stackrel{r}{\rightarrow}_{S,V} \ X : Z \ \& \ F \ \& \ G \ /\!/ \ H_3$$
with $C = (Z < Y' \coloneq F \ /\!/ \ H_2)$ and
$H_3 = (H_1 \ \& \ H_2 \ \& \ Z < Y' \ \& \ Y = Y')$,
if H_3 is satisfiable.

Reduction-by-inheritance

$$X < Y \ \& \ G \ /\!/ \ H_1 \ \stackrel{r}{\rightarrow}_{S,V} \ X < Z \ \& \ F \ \& \ G \ /\!/ \ H_3$$
with $C = (Z < Y' \coloneq F \ /\!/ \ H_2)$ and
$H_3 = (H_1 \ \& \ H_2 \ \& \ Z < Y' \ \& \ Y = Y')$,
if H_3 is satisfiable.

In the reduction rules, the head A' of clause C is pushed in the conjunction of constraints H_3. This is because A' must be satisfiable w.r.t. the other constraints.

The reader interested in satisfiability tests for conjunctions of atomic constraints may refer to the works done in the field of terminological knowledge bases (e.g., [16]).

Properties

The proofs[24] of the two following theorems can be found in part 2 (theorems 7.8 and 6.2). The completeness has been shown using a technique due to Höhfeld and Smolka [30].

[24]The proofs have been done for a slightly different set of atomic constraints, but these minor changes have no impact on them.

Let $\stackrel{r}{\rightarrow}^*_{S,V}$ be the reflexive and transitive closure of $\stackrel{r}{\rightarrow}_{S,V}$.

Theorem 4.1 (Soundness) *If* $Q \stackrel{r}{\rightarrow}^*_{S,V} Q'$, *then for every model* Γ *of* S, $[Q']^\Gamma \subseteq [Q]^\Gamma$.

Let Γ be an interpretation, α be an assignment and V be a finite set of variables. Then $\alpha|_V$ is the restriction of α to $V \cup \mathcal{K}$.

Let Q be a goal, then $[Q]^\Gamma_V := \{\alpha|_V \mid \alpha \in [Q]^\Gamma\}$.

Theorem 4.2 (Completeness) *Let* Γ *be a minimal model of* S *(wrt* $\leq$*),* Q *be a goal and* $\alpha \in [Q]^\Gamma$*; then there exists a conjunction of atomic constraints* H *such that* $Q \stackrel{r}{\rightarrow}^*_{S,V} \epsilon \ /\!/ \ H$ *and* $\alpha|_V \in [\epsilon \ /\!/ \ H]^\Gamma_V$.

4.4 Database and Query

In our framework, a *database* is a triple $\langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ where $\mathcal{SCH}$ (the schema) is a conjunction of variable-free atomic constraints, $\mathcal{EDB}$ (the extensional database) is a conjunction of variable-free atomic links and $\mathcal{IDB}$ (the intensional database) is a set of clauses. A *query* is a goal.

An interpretation $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ is a *model of a database* iff $[\mathcal{EDB}]^\mathcal{B} = [\mathcal{SCH}]^\mathcal{B} = ASG^\mathcal{B}$ and Γ is a model of $\mathcal{IDB}$.

Informally, the conditions $[\mathcal{EDB}]^\mathcal{B} = ASG^\mathcal{B}$ and $[\mathcal{SCH}]^\mathcal{B} = ASG^\mathcal{B}$ ensure that both $\mathcal{EDB}$ and the schema are true in the model.

Let $\mathcal{DB}$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and H be a conjunction of atomic constraints.

Q is a *logical consequence of* $\mathcal{DB}$ *under hypothesis* H *on* V (written $\mathcal{DB} \models H \Rightarrow_V Q$) iff for every model $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ of $\mathcal{DB}$, $\forall \alpha \in ASG^\Gamma, \alpha \in [H]^\mathcal{B} \Rightarrow \alpha|_V \in [Q]^\Gamma_V$.

This extends to set of queries in the following way:

Let $\mathcal{R}$ be a set of queries. $\mathcal{R}$ is a *logical consequence of* $\mathcal{DB}$ *under hypothesis* H *on* V (written $\mathcal{DB} \models H \Rightarrow_V \mathcal{R}$) iff for every model $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ of $\mathcal{DB}$, $\forall \alpha \in ASG^\Gamma, \alpha \in [H]^\mathcal{B} \Rightarrow (\exists Q' \in \mathcal{R}$ such that $\alpha|_V \in [Q']^\Gamma_V)$.

Informally, this interprets a set of queries as a disjunction of queries.

Finally, H is a *V-answer* to Q (resp. $\mathcal{R}$) iff H is satisfiable and $\mathcal{DB} \models H \Rightarrow_V Q$ (resp. $\mathcal{DB} \models H \Rightarrow_V \mathcal{R}$).

This definition is more general than the classical notion of answer. Here an answer is a hypothesis H such that if H is satisfied then the query is also satisfied.

5 Reformulation

This section presents the various reformulations outlined in Section 3. Proofs of the theorems are given in the Appendix.

5.1 Resolution-Based Reformulation

In this kind of reformulation, we perform resolution steps involving the query and the $\mathcal{IDB}$, by application of the reduction rules. Then, we obtain a new query in which the disjunctive parts that cannot have any answers with respect to the integrity constraints of the schema have been removed.

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, Q be a query and V be a finite set of variables. Then, a set $\mathcal{R}$ of queries obtained by (V, r)-*reformulation* of Q is a set that satisfies the following four conditions:

1. $\forall C \in \mathcal{IDB}$,
   let $\mathcal{R}_C := \{Q' \mid Q \xrightarrow{r}_{\mathcal{IDB},V} Q'$ using a variant of C and the direct reduction rule$\}$,
   if $\mathcal{R}_C \neq \emptyset$, then $\mathcal{R}_C \cap \mathcal{R} \neq \emptyset$.

2. $\forall C \in \mathcal{IDB}$,
   let $\mathcal{R}_C := \{Q' \mid Q \xrightarrow{r}_{\mathcal{IDB},V} Q'$ using a variant of C and the reduction-by-inclusion rule$\}$,
   if $\mathcal{R}_C \neq \emptyset$, then $\mathcal{R}_C \cap \mathcal{R} \neq \emptyset$.

3. $\forall C \in \mathcal{IDB}$,
   let $\mathcal{R}_C := \{Q' \mid Q \xrightarrow{r}_{\mathcal{IDB},V} Q'$ using a variant of C and the reduction-by-inheritance rule$\}$,
   if $\mathcal{R}_C \neq \emptyset$, then $\mathcal{R}_C \cap \mathcal{R} \neq \emptyset$.

4. $\mathcal{R}$ is minimal with respect to set inclusion.

Informally, $\mathcal{R}$ is the set of all queries that can be obtained from Q by a reduction step, with the following restriction: when a reduction can be performed with different variants of a clause, only one of the variants is used. It should be noted that for a given query Q, there exists several reformulated queries, depending on the different variants one can use in a reduction step.

In order to discard the queries that cannot have any answer with respect to the integrity constraints, we incorporate the schema into the reformulation process. For a given query $Q = F \mathbin{/\!/} H$, this is achieved if we perform the (V, r)-reformulation of $F \mathbin{/\!/} H \ \& \ \mathcal{SCH}$. Informally, this leads to allow only reductions by which $\mathcal{SCH}$ remains satisfiable.

Theorem 5.1 (Soundness of Reformulation by Resolution)
Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and $\mathcal{R}$ obtained by (V, r)-reformulation of $(Q \ \& \ \mathcal{SCH})$. If H is a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$, then H is a V-answer to Q wrt $\mathcal{DB}$.

Theorem 5.2 (Completeness of Reformulation by Resolution)
Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and $\mathcal{R}$ obtained by (V, r)-reformulation of $(Q \ \& \ \mathcal{SCH})$. For every V-answer H to Q wrt $\mathcal{DB}$, H is also a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$.

The equality constraints generated by the reduction rules, like in constraint logic programming language (e.g., Prolog III [22]), play a role similar to unification in classical resolutions. In order to facilitate reading, in what follows, the queries have been simplified with respect to these constraints.

Moreover, the presence of $\mathcal{SCH}$ and the form of the reduction rules lead to obtain other additional atomic constraints that will not be given. However, these atomic constraints do not represent accesses to the extensional part of the database and can be easily filtered out.

Consider the example of Section 3. Let $V = \{E\}$ be the set of variables of interest in Q. We use $(\mathcal{IDB}, V)$-reduction to reformulate this query.

A direct reduction of $(Q \ \& \ \mathcal{SCH})$ using a variant of *(r2)* gives Q_2. A reduction-by-inclusion of $(Q \ \& \ \mathcal{SCH})$ using a variant of *(r3)* gives Q_3. The other reduction rules do not apply to Q using a variant of *(r2)* or *(r3)*.

A reduction of Q using a variant of *(r1)* is not possible, since it would lead to a set of atomic constraints which is not satisfiable.

Since no other reduction is possible using other clauses of $\mathcal{IDB}$, $\mathcal{R} = \{Q_2, Q_3\}$ is obtained by (V, r)-reformulation of Q. By soundness and completeness of this reformulation, $\mathcal{R}$ is semantically equivalent to Q.

5.2 Propagation-Based Reformulation

Let $Q = F \ // \ H$ be a query. In order to simplify the notation, we view $H = Ac_1 \ \& \ \ldots \ \& \ Ac_n$ as a set of atomic constraints, that is $H = \{Ac_1, \ldots, Ac_n\}$. To make the presentation easier, we will also work with an expanded version of the schema, obtained by replacing all atomic constraints of the form $C \sqsubseteq and(D_1, \ldots, D_m)$ by the conjunction of the atomic constraints $C \sqsubseteq D_1, \ldots, C \sqsubseteq D_m$

The purpose of this reformulation is to complete H by introducing additional selection information about objects to be retrieved. To achieve this,

we add constraints to H by means of so-called propagation rules. Each application of a rule will lead to a new set of constraints.

$H[X/Y]$ denotes the set of constraints obtained from H by substituting each occurrence of the variable X by Y.

Let X, R and Y be object names, C and D be concepts, V_1 and V_2 be two distinct variables and k be a constant. The propagation rules are:

- Variable simplification:

  (p1) $H \longrightarrow (H - \{V_1 = V_2\})[V_1/V_2] \cup \{V_1 = V_2\}$ if $(V_1 = V_2) \in H$.

  (p2) $H \longrightarrow (H - \{k = V_1\})[V_1/k] \cup \{V_1 = k\}$ if $(k = V_1) \in H$.

  (p3) $H \longrightarrow (H - \{V_1 = k\})[V_1/k] \cup \{V_1 = k\}$ if $(V_1 = k) \in H$.

- Propagation of class properties at the instance level:

  (p4) $H \longrightarrow \{X : D\} \cup H$

  if $(X : C) \in H$ and the schema contains $C \sqsubseteq D$.

- Propagation of the attribute properties:

  (p5) $H \longrightarrow \{Y : C\} \cup H$

  if $(X.R \rightarrow Y) \in H$ and $(X : all(R, C)) \in H$.

These propagation rules are based on those employed in satisfiability tests for terminological knowledge bases [16]. This set of propagation rules is not exhaustive. Since this framework does not focus on propagation-based reformulation, only a restricted set of rules is given.

Theorem 5.3 (Correctness of Propagation-Based Reformulation) *Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, $Q = F \;/\!/\; H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by a finite sequence of applications of the propagation rules; then for every model Γ of $\mathcal{DB}$, $[F \;/\!/\; H_1]^\Gamma = [F \;/\!/\; H_2]^\Gamma$.*

Consider the example of Section 3. The reformulation by propagation is applied to the part of Q_4 which refers to the extensional database:

X:teacher & E:lecturer & X.works_in_project → Z &
Z.managed_by → E & X:student

The class student inherits from the concept all(works_in_project, project) *(s4)* and the query contains the atomic constraint X:student. Thus, by the application of the rule *(p4)* we obtain the new atomic constraint X:all(works_in_project, project).

Rule *(p5)* applies to X.works_in_project → Z and X:all(works_in_project, project). It generates Z:project.

The rule *(p4)* uses all(managed_by,permanent_staff_member) inherited by the class project *(s7)* and Z:project to add Z:all(managed_by,permanent_staff_member).

Finally, rule *(p5)* can be used to obtain E:permanent_staff_member from Z.managed_by $\rightarrow$ E and Z:all(managed_by,permanent_staff_member).

The rules *(p1) (p2) (p3)* are required to handle the equality constraints generated by resolution. As this kind of constraint has already been simplified (to facilitate reading), these rules are not used in the example. However, their application is straightforward.

5.3 Factorization and Classification-Based Reformulation

Let $Q = F \;/\!/\; H$ be a query. Like in the Section 5.2, in order to simplify the notation, we view $H = Ac_1 \;\&\; \ldots \;\&\; Ac_n$ as a set of atomic constraints, that is $H = \{Ac_1, \ldots, Ac_n\}$.

H may contain redundant parts, and conjunctions of atomic constraints whose evaluation can be replaced by accesses to materialized views. So, it is useful to retain only the most specific constraints, and to detect parts of the query corresponding to the evaluation of materialized views. To do this, we rewrite H by means of a technique based on concept classification.

First we put together constraints related to the same object. This factorization process is performed by means of the following factorization rule (used iteratively until it can no longer be applied):

$$H \longrightarrow \{X : and(C1, C2)\} \cup (H - \{X : C1, X : C2\})$$
$$\text{if } (X : C1) \in H \text{ and } (X : C2) \in H \text{ and } (X : and(C1, C2)) \notin H.$$

Theorem 5.4 (Termination of the Factorization Process)
Any sequence of application of the factorization rule is finite.

Theorem 5.5 (Correctness of Factorization)
Let $Q = F \;/\!/\; H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by the factorization process; then for every model Γ of any database, $[F \;/\!/\; H_1]^\Gamma = [F \;/\!/\; H_2]^\Gamma$.

Before describing the classification used, we need to define a partial ordering on concepts, called subsumption relation. Let C_1 and C_2 be two concepts: C_1 subsumes C_2 iff for each semi-interpretation $\mathcal{J}$ such that $[\mathcal{SCH}]^\mathcal{J} = ASG^\mathcal{J}$ (i.e., the schema is satisfied by $\mathcal{J}$) and for each $\mathcal{J}$-assignment α we have $ext_{\mathcal{J}\alpha}(C_2) \subseteq ext_{\mathcal{J}\alpha}(C_1)$.

C_1 is called subsumer of C_2, and C_2 is said to be subsumed by C_1.

Let Ω be the set of all concepts which are not of the form $and(D_1, \ldots, D_m)$.

The classification of a concept C consists in finding in Ω its least subsumers and its greatest subsumees.

The reader interested in decidability and complexity of concept subsumption, or in concept classification may refer to [46].

Let H' be obtained from H by the factorization process. The core of the classification-based reformulation is performed as follows: for every atoms of the form $X : C$ in H', first classify C and then replace it in $X : C$ by the conjunction of its least subsumers.

For example, with respect to a schema containing only one atomic constraint $C_1 \sqsubseteq C_2$, for the concept $and(C_1, C_2, C_3)$ we obtain the two most-specific subsumers C_1 and C_3 in Ω. Thus, the concept $and(C_1, C_2, C_3)$ is simplified and replaced by $and(C_1, C_3)$.

In the case of another schema containing only the definition of a materialized view $C_4 := and(C_1, C_2)$, the classification of $and(C_1, C_2, C_3)$ gives the two most-specific subsumers C_4 and C_3 in Ω. Then, $and(C_1, C_2, C_3)$ is replaced by $and(C_4, C_3)$, which takes advantage of the materialized view.

Theorem 5.6 (Correctness of Classification-Based Reformulation)

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, $Q = F \;/\!/\; H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by a classification-based reformulation; then for every model Γ of $\mathcal{DB}$, $[F \;/\!/\; H_1]^\Gamma = [F \;/\!/\; H_2]^\Gamma$.

Consider the example of Section 3. The factorization and classification-based reformulation is applied to the following query part which refers to the extensional database:

X:teacher & E:lecturer & X.works_in_project $\rightarrow$ Z & Z.managed_by $\rightarrow$ E & X:student & Z:project & E:permanent_staff_member

The result of the factorization process is:

X:and(teacher, student) & E:and(permanent_staff_member, lecturer) &
X.works_in_project $\rightarrow$ Z & Z.managed_by $\rightarrow$ E & Z:project

Using the atomic constraint assistant := and(student, teacher) contained in the schema *(s9)*, the classification of the concept and(teacher, student) gives a single least subsumer: the concept assistant. Likewise, because of *(s6)* lecturer $\sqsubseteq$ and(permanent_staff_member, teacher, not(professor)) the classification of and(permanent_staff_member, lecturer) gives lecturer as the only least subsumer.

Then, X:and(teacher, student) and E:and(permanent_staff_member, lecturer) are replaced respectively by X:assistant and E:lecturer. So, we have

eliminated a redundant constraint and taken into account a materialized view to simplify the query.

5.4 Reformulation Heuristics

A query can often be reformulated in several ways, and it is important to choose the most interesting reformulation. Chakravarthy et al. [20] have suggested three solutions: (1) generate several equivalent queries and let the evaluation plan generator (of the database) select the optimal one. (2) use heuristics based on an evaluation cost model to guide the query reformulation. (3) combine (1) and (2) by tightly integrating the plan generation and the query reformulation.

These aspects are out of the scope of this work. However, we mention a preferential ordering for the different kinds of reformulations.

The transformations based on resolution and those based on classification are complementary. The former helps to simplify the part of the query that refers to the intensional database, while the latter removes redundant atoms that refer to the extensionnal database. Since a resolution step can add new atoms that refer to the extensional database, it is preferable to perform a classification-based reformulation after a reformulation by resolution.

The propagation rules simplify equality constraints (e.g., those generated by resolution) and also adds atoms to the query. This can be useful to perform a more accurate classification. Thus a propagation-based reformulations should be done before a classification process and after a resolution-based reformulation.

6 Conclusion

Two techniques have already been proposed for semantic query optimization in deductive object oriented databases:

- The first uses a highly expressive language for database description and a specific resolution for query reformulation. This technique is an extension of that of Chakravarthy et al. [20]. It has been developed by Yoon and Kerschberg [55] to exploit the inheritance links of an object-oriented database and the clauses describing its intensional part.

- The second technique is based on a restricted language to describe the schema and the deduction rules. This turns a particular subsumption relation to be decidable and then provides a classification process. This computation leads to simplifications of the query that cannot be obtained in the first approach, where this subsumption relation is undecidable in general [52]. This technique has been explored by Beck et al. [10], Borgida et al. [12] and Buchheit et al. [17].

Our proposal is a reconciliation of these two approaches. Indeed, we use clauses to describe the intensional database and more restricted formulae to specify the database schema. Thus, on one hand, we can describe derived classes and derived attributes with enough generality. And, on the other hand, we can take into account conventional integrity constraints of object-oriented databases (e.g., types of attributes, single-valuations of attributes, mandatory valuations, disjunctions of classes), and a restricted form of materialized views to perform simplifications by means of a decidable subsumption relation.

Important problems not covered by the proposal are the handling of negation in the clause bodies and the ordering of the reformulations.

The main contribution of this work is to show that resolution-based reasoning and classification-based reasoning can be used together in a common framework to perform complementary semantic query optimizations.

To simplify the presentation we have used only the core of an integrity constraint language. However, it can be easily extended without any change to the framework, if we take care to keep the subsumption relation decidable. The interested reader may find such extensions in [16] for cardinality constraints and in [8] for numerical constraints. Another possibility is to allow also clauses describing integrity constraints. In this case, the classical technique of Chakravarthy et al.[20] can be adapted, but this kind of integrity constraints will not be used in the classification-based reformulation.

7 Appendix

This appendix presents proofs of the theorems of Section 5.

Theorem 5.1

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and $\mathcal{R}$ obtained by (V, r)-reformulation of $(Q \ \& \ \mathcal{SCH})$. If H is a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$, then H is a V-answer to Q wrt $\mathcal{DB}$.

Proof:

Let H be a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$.

Let $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ be a model of the database $\mathcal{DB}$ and α be an Γ-assignment such that $\alpha \in [H]^{\mathcal{B}}$. Since $\mathcal{DB} \models H \Rightarrow_V \mathcal{R}$, there exists $Q' \in \mathcal{R}$ such that $\alpha|_V \in [Q']^{\Gamma}_V$. The definition of $\mathcal{R}$ implies $(Q \ \& \ \mathcal{SCH}) \xrightarrow{r}_{\mathcal{IDB},V} Q'$. Then by theorem 4.1 $\alpha|_V \in [Q \ \& \ \mathcal{SCH}]^{\Gamma}_V$. Hence $\alpha|_V \in [Q]^{\Gamma}_V$.

Thus $\mathcal{DB} \models H \Rightarrow_V Q$.

H is satisfiable because H is a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$.
Hence, H is a V-answer to Q wrt $\mathcal{DB}$.

□

Theorem 5.2

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and $\mathcal{R}$ obtained by (V, r)-reformulation of $(Q \ \& \ \mathcal{SCH})$. For every V-answer H to Q wrt $\mathcal{DB}$, H is also a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$.

Proof:

Let H be a V-answer to Q wrt $\mathcal{DB}$.

Let $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ be a model of the database $\mathcal{DB}$ and α be an Γ-assignment such that $\alpha \in [H]^{\mathcal{B}}$. Since $\mathcal{DB} \models H \Rightarrow_V Q$ it follows that $\alpha|_V \in [Q]^{\Gamma}_V$.

We now show that there exists a minimal model Ω (wrt $\leq$) of the $\mathcal{IDB}$ based on $\mathcal{B}$.

We define a well-founded ordering $<$ on interpretations.

Let Γ_1 and Γ_2 be two interpretations then $\Gamma_1 < \Gamma_2$ iff $\Gamma_1 \leq \Gamma_2$ and $\Gamma_1 \neq \Gamma_2$ (where $\leq$ is the partial ordering relation on interpretations defined in Section 4).
$<$ is well-founded since $\subset$ is well-founded.

Let Δ be the set of all interpretations that are models of $\mathcal{IDB}$ based on $\mathcal{B}$. Δ is not empty since it contains Γ. $<$ is well-founded on interpretations, hence there exists a "minimal" element Ω wrt $<$ in Δ (i.e., for all Ω' in Δ, $\Omega' \not\leq \Omega$). Ω is also minimal wrt $\leq$.

Hence, there exists a minimal model Ω (wrt $\leq$) of $\mathcal{IDB}$ based on $\mathcal{B}$.

Γ based on $\mathcal{B}$ is a model of $\mathcal{DB}$ implies $[\mathcal{EDB}]^{\mathcal{B}} = [\mathcal{SCH}]^{\mathcal{B}} = ASG^{\mathcal{B}}$. Then, since Ω based on $\mathcal{B}$ is a model of $\mathcal{IDB}$, Ω based on $\mathcal{B}$ is a model of $\mathcal{DB}$. Then, because $\alpha \in [H]^{\mathcal{B}}$ and $\mathcal{DB} \models H \Rightarrow_V Q$, $\alpha|_V \in [Q]_V^{\Omega}$.

By theorem 4.2, there exists H_r, a conjunction of atomic constraints such that $Q \xrightarrow{r}{}^*_{\mathcal{IDB},V} \epsilon \mathbin{/\!/} H_r$ and $\alpha|_V \in [\epsilon \mathbin{/\!/} H_r]_V^{\Omega}$. Since Ω is based on $\mathcal{B}$, then $\alpha|_V \in [H_r]_V^{\mathcal{B}}$.

Thus, $\alpha|_V \in [\epsilon \mathbin{/\!/} H_r]_V^{\Gamma}$ because Γ is based on $\mathcal{B}$.

Let Q' be a query such that $(Q \;\&\; \mathcal{SCH}) \xrightarrow{r}_{\mathcal{IDB},V} Q' \xrightarrow{r}{}^*_{\mathcal{IDB},V} (\epsilon \mathbin{/\!/} H_r)$. Q' exists because $(Q \;\&\; \mathcal{SCH}) \xrightarrow{r}{}^*_{\mathcal{IDB},V} (\epsilon \mathbin{/\!/} H_r)$

From $\alpha|_V \in [\epsilon \mathbin{/\!/} H_r]_V^{\Gamma}$ and theorem 4.1 it follows that $\alpha|_V \in [Q']_V^{\Gamma}$.

Let C' be the variant of a clause $C \in \mathcal{IDB}$, such that C' is used in the (S,V)-reduction $(Q \;\&\; \mathcal{SCH}) \xrightarrow{r}_{\mathcal{IDB},V} Q'$. By definition of $\mathcal{R}$, there exists $Q'' \in \mathcal{R}$ such that $(Q \;\&\; \mathcal{SCH}) \xrightarrow{r}_{\mathcal{IDB},V} Q''$ using a variant C'' of C.
From the form of the reduction rules and the conditions $(V \cup var(Q \;\&\; \mathcal{SCH})) \cap var(C') = \emptyset$ and $(V \cup var(Q \;\&\; \mathcal{SCH})) \cap var(C'') = \emptyset$ satisfied by the variants used in the reductions, it follows that Q'' is a variant of Q'. Additionally, let ρ be a renaming such that $Q'' = \rho(Q')$, then $\forall X \in V, \rho(X) = X$.
Thus $\alpha|_V \in [Q']_V^{\Gamma}$ implies $\alpha|_V \in [Q'']_V^{\Gamma}$.

Then, for any model $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ of the database $\mathcal{DB}$ and for any $\alpha \in [H]^{\mathcal{B}}$, there exists $Q'' \in \mathcal{R}$ such that $\alpha|_V \in [Q'']_V^{\Gamma}$. Whence $\mathcal{DB} \models H \Rightarrow_V \mathcal{R}$.

H is satisfiable because H is a V-answer to Q wrt $\mathcal{DB}$.
Hence H is a V-answer to $\mathcal{R}$ wrt $\mathcal{DB}$.

□

Theorem 5.3

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, $Q = F \mathbin{/\!/} H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by a finite sequence of applications of the propagation rules; then for every model Γ of $\mathcal{DB}$, $[F \mathbin{/\!/} H_1]^\Gamma = [F \mathbin{/\!/} H_2]^\Gamma$.

Proof sketch:
Let $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ be a model of $\mathcal{DB}$ then it is sufficient to show that $[H_1]^\mathcal{B} = [H_2]^\mathcal{B}$.

Let H_3 be a set of atomic constraints and H_4 obtained from H_3 by one application of a propagation rule. From the forms of the propagation rules and the definition of the sets of $\mathcal{B}$-solutions, we deduce that $[H_3]^\mathcal{B} = [H_4]^\mathcal{B}$ (a detailed version of this reasoning step is given in proof of theorem 5.5 for the factorization rule).

Then, by induction on the length of the sequence of applications of the propagation rules, we have $[H_1]^\mathcal{B} = [H_2]^\mathcal{B}$. □

Theorem 5.4

Any sequence of application of the factorization rule is finite.

Proof sketch: An application of the factorization rule strictly decreases the cardinality of the set of atomic constraints. Thus, by induction on the length of the sequence, any sequence of applications of the rule strictly decreases the cardinality of the set of atomic constraints. Since the factorization process applies to a finite set of atomic constraints, this implies that any sequence of application of the factorization rule is finite. □

Theorem 5.5

Let $Q = F \mathbin{/\!/} H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by the factorization process; then for every model Γ of any database, $[F \mathbin{/\!/} H_1]^\Gamma = [F \mathbin{/\!/} H_2]^\Gamma$.

Proof:

Let $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ be a model of a database. Since $[F \mathbin{/\!/} H_1]^\Gamma = [F]^\mathcal{I} \cap [H_1]^\mathcal{B}$ and $[F \mathbin{/\!/} H_2]^\Gamma = [F]^\mathcal{I} \cap [H_2]^\mathcal{B}$, it is sufficient to show that $[H_1]^\mathcal{B} = [H_2]^\mathcal{B}$.

Let H_3 be a set of atomic constraints and H_4 obtained from H_3 by one application of the factorization rule, we first show that $[H_3]^{\mathcal{B}} = [H_4]^{\mathcal{B}}$.

Let $H_5 := H_3 \cap H_4$. Then, $H_3 = H_5 \cup \{X : C_1, X : C_2\}$ and $H_4 = H_5 \cup \{X : and(C_1, C_2)\}$. Thus it is sufficient to show that $[X : C_1 \ \& \ X : C_2]^{\mathcal{B}} = [X : and(C_1, C_2)]^{\mathcal{B}}$.

$$\begin{aligned}
[X : C_1 \ \& \ X : C_2]^{\mathcal{B}} &= [X : C_1]^{\mathcal{B}} \cap [X : C_2]^{\mathcal{B}} \\
&= \{\alpha \in ASG^{\mathcal{B}} \mid \alpha(X) \in ext_{\mathcal{B}\alpha}(C_1)\} \\
&\quad \cap \{\alpha \in ASG^{\mathcal{B}} \mid \alpha(X) \in ext_{\mathcal{B}\alpha}(C_2)\} \\
&= \{\alpha \in ASG^{\mathcal{B}} \mid \alpha(X) \in (ext_{\mathcal{B}\alpha}(C_1) \cap ext_{\mathcal{B}\alpha}(C_2))\} \\
&= \{\alpha \in ASG^{\mathcal{B}} \mid \alpha(X) \in ext_{\mathcal{B}\alpha}(and(C_1, C_2))\} \\
&= [X : and(C_1, C_2)]^{\mathcal{B}}
\end{aligned}$$

H_2 is obtained from H_1 by a finite sequence of applications of the factorization rule. By induction on the sequence length we have $[H_1]^{\mathcal{B}} = [H_2]^{\mathcal{B}}$.

□

Theorem 5.6

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, $Q = F \ /\!/ \ H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by a classification-based reformulation; then for every model Γ of $\mathcal{DB}$, $[F \ /\!/ \ H_1]^{\Gamma} = [F \ /\!/ \ H_2]^{\Gamma}$.

Proof:

Let $\Gamma = \langle \mathcal{I}, \mathcal{B} \rangle$ be a model of $\mathcal{DB}$. Let H_3 be the set of atomic constraints obtained from H_1 by means of the factorization process, during the classification-based reformulation that has generated H_2. Since the factorization is correct (theorem 5.5) we have $[F \ /\!/ \ H_1]^{\Gamma} = [F \ /\!/ \ H_3]^{\Gamma}$. Thus we only have to prove that $[F \ /\!/ \ H_3]^{\Gamma} = [F \ /\!/ \ H_2]^{\Gamma}$. It is sufficient to show that $[H_3]^{\mathcal{B}} = [H_2]^{\mathcal{B}}$.

Let $X : C$ be an atomic constraint and $\mathcal{LS} = \{C_1, \ldots, C_n\}$ be the set of the least subsumers of C. We will show that $[X : C]^{\mathcal{B}} = [X : and(C_1, \ldots, C_n)]^{\mathcal{B}}$.

$[\mathcal{SCH}]^{\mathcal{B}} = ASG^{\mathcal{B}}$, since Γ is a model of $\mathcal{DB}$. Then, by definition of a subsumer, $\forall C_i \in \mathcal{LS}, \forall \alpha \in ASG^{\mathcal{B}}, ext_{\mathcal{B}\alpha}(C) \subseteq ext_{\mathcal{B}\alpha}(C_i)$. Thus, $\forall \alpha \in ASG^{\mathcal{B}}, ext_{\mathcal{B}\alpha}(C) \subseteq ext_{\mathcal{B}\alpha}(and(C_1, \ldots, C_n))$. Whence, $[X : C]^{\mathcal{B}} \subseteq [X : and(C_1, \ldots, C_n)]^{\mathcal{B}}$.

We now show by contradiction that $[X : C]^{\mathcal{B}} \supseteq [X : and(C_1, \ldots, C_n)]^{\mathcal{B}}$.

Assume that $\exists \alpha \in ASG^{\mathcal{B}}, \exists x \in ext_{\mathcal{B}\alpha}(and(C_1, \ldots, C_n))$ such that $x \notin ext_{\mathcal{B}\alpha}(C)$. Let such α and x be given.

Let Ω be the set of all concepts which are not of the form $and(D_1, \ldots, D_m)$.

Case 1: $C \in \Omega$.

C is a subsumer of C. For all $C_i \in \mathcal{LS}$, since C_i is smaller than any other subsumer of C (by definition of $\mathcal{LS}$), we have $ext_{\mathcal{B}\alpha}(C_i) \subseteq ext_{\mathcal{B}\alpha}(C)$. Hence $ext_{\mathcal{B}\alpha}(and(C_1, \ldots, C_2)) \subseteq ext_{\mathcal{B}\alpha}(C)$, contradicting the assumption.

Case 2: $C \notin \Omega$.

Then C has the following form: $C = and(D_1, \ldots, D_m)$. Since $x \notin ext_{\mathcal{B}\alpha}(C)$, there exists a D_j in $and(D_1, \ldots, D_m)$ such that $x \notin ext_{\mathcal{B}\alpha}(D_j)$. Furthermore, $C = and(D_1, \ldots, D_m)$ implies that D_j is a subsumer of C.

Since $\mathcal{LS}$ is the set of the least subsumers of C, for all $C_i \in \mathcal{LS}$, we have $ext_{\mathcal{B}\alpha}(C_i) \subseteq ext_{\mathcal{B}\alpha}(D_j)$. Thus, for all $C_i \in \mathcal{LS}$, $x \notin ext_{\mathcal{B}\alpha}(D_j)$ implies $x \notin ext_{\mathcal{B}\alpha}(C_i)$. Hence $x \notin ext_{\mathcal{B}\alpha}(and(C_1, \ldots, C_n))$, contradicting the assumption.

Thus, $\forall \alpha \in ASG^{\mathcal{B}}, \forall x \in ext_{\mathcal{B}\alpha}(and(C_1, \ldots, C_n)), x \in ext_{\mathcal{B}\alpha}(C)$.
And then $[X : C]^{\mathcal{B}} \supseteq [X : and(C_1, \ldots, C_n)]^{\mathcal{B}}$

Hence $[X : C]^{\mathcal{B}} = [X : and(C_1, \ldots, C_n)]^{\mathcal{B}}$.

Let H_4 be a conjunction of atomic constraint containing $X : C$ and H_5 be obtained from H_4 by replacing $X : C$ by $X : and(C_1, \ldots, C_2)$. Since $[X : C]^{\mathcal{B}} = [X : and(C_1, \ldots, C_n)]^{\mathcal{B}}$, then $[H_4]^{\mathcal{B}} = [H_5]^{\mathcal{B}}$.

By induction on the number of replacements performed in H_3 to obtain H_2 we have $[H_3]^{\mathcal{B}} = [H_2]^{\mathcal{B}}$

□

Quatrième partie

Conclusion

1 Contribution

Le travail de synthèse présenté dans la première partie a mis en évidence trois familles de langages, correspondant chacune à une forme d'utilisation de la structure des objets dans les raisonnements sur des clauses. Le travail réalisé ensuite visait à intégrer des notions rencontrées dans deux de ces familles. La synthèse préliminaire nous a alors permis d'entreprendre une intégration à partir de paradigmes de représentations et de raisonnements clairement identifiés. Il s'agit de l'utilisation de la structure des objets d'une part comme contrainte au niveau des termes et d'autre part comme information connue ou déduite.

L'outil-langage que nous avons ainsi construit, a été décrit sous deux angles fondamentaux: (1) sa sémantique déclarative au travers de la notion de réponse correcte et (2) sa sémantique opérationnelle en définissant la notion de réponse calculée.
Les deux approches que nous avons intégrées sont suffisament orthogonales, sur le plan opérationnel, pour que les mécanismes de raisonnement mis en jeu aient pu être combinés sans trop de difficultés. En revanche, en ce qui concerne les aspects déclaratifs, le travail de construction a été plus délicat car il a nécessité d'intégrer ensemble deux niveaux différents de description de la structure des objets: (1) constraintes sur les termes et (2) prédicats définis par des clauses. Ceci nous a conduit à une sémantique qui fournie une spécification déclarative [38] de la notion de réponse correcte. C'est par référence à cette spécification que nous avons alors établi la justesse et la complétude de notre sémantique opérationnelle.

Cet outil-langage de représentation et de raisonnement a ensuite été utilisé pour la reformulation des requêtes dans les bases de données déductives orienté-objet. Ceci nous a permis de proposer un cadre original d'optimisation sémantique de requêtes, dont l'intérêt principal est d'autoriser la combinaison de deux schémas d'optimisation complémentaires existants.

2 Perspectives

Le cadre d'optimisation sémantique proposé se concentre sur les reformulations de requêtes à un niveau élémentaire et ne fixe pas leur enchaînement. Il serait donc intéressant de l'étendre vers un schéma de transformation plus général, soit en guidant ces reformulations élémentaires à partir d'heuristiques d'optimisation, soit en les insérant dans une procédure globale d'évaluation de requêtes.

Si de tels développements semblent naturels à partir du travail déjà réalisé, certains des aspects originaux de la sémantique déclarative offrent des

perspectives d'applications plus inattendues. Ainsi, elle semble permettre d'aborder le problème de la mise à jour aux travers des vues par abduction [33] [15].

Rappelons sous une forme simplifiée une des clefs de l'utilisation de cet outil-langage en optimisation sémantique. Soient S la description de la partie intentionnelle de la base de données et Q une requête (ou une sous-requête obtenue au cours d'une évaluation). Nous pouvons savoir s'il existe un ensemble d'hypothèses H à faire sur les valeurs des variables de Q et sur le partie extentionnelle de la base (EDB), pour que Q soit la conséquence logique de S sous les hypothèses H. Si un tel ensemble H n'existe pas, ceci signifie que quelque soit l'EDB, la requête (ou la sous-requête) ne peut avoir de réponse et son évaluation ne nécessite donc pas d'être poursuivie.

Dans cette application, nous ne considérons bien sûr que les ensembles d'hypothèses qui respectent les contraintes d'intégrité. En revanche, nous ne tenons pas compte de l'EDB courante de la base, car il est en effet souhaitable de ne pas avoir à accéder au contenu extentionnel durant une phase d'optimisation de requêtes.

Si à présent nous utilisons aussi l'EDB courante, un problème de mise à jour aux travers de vues peut s'exprimer de la façon suivante : soient S la description des vues et M la mise à jour souhaitée, quelle est H la nouvelle EDB construite à partir de l'EDB courante, telle que S ait pour conséquence logique M sous les hypothèses H ?

Appliquons ce principe à un exemple tiré de [32].

Soit l'EDB décrivant les mères et les pères d'un ensemble d'individus. Soient trois vues indiquant (1) que la mère de X est un de ses parents, (2) que le père de X est aussi un de ses parents et (3) que si Z est un parent de X et de Y alors X et Y ont un parent commun.

Comme en optimisation sémantique, nous utilisons des contraintes dans le corps des clauses pour décrire les accès à l'EDB. Ceci nous conduit à représenter les vues de la façon suivante :

X.parent $\rightarrow$ Z :– // X.mère $\rightarrow$ Z.

X.parent $\rightarrow$ Z :– // X.père $\rightarrow$ Z.

X.même-parent $\rightarrow$ Y :– X.parent $\rightarrow$ Z & Y.parent $\rightarrow$ Z.

Soient les contraintes d'intégrité indiquant qu'une personne a au plus une mère et au plus un père, que les mères sont des femmes, que les pères sont des hommes, et enfin que homme et femme sont deux sous-classes disjointes de personne. Soit l'EDB mentionnant quatre individus : une femme marie dont la mère est linda et le père est john, ainsi qu'un homme arsène.

A la différence de ce que nous avons fait en optimisation sémantique, nous utilisons maintenant l'EDB comme une source de contraintes. La conjonction globale de contraintes devant être satisfaite est alors :
C =

personne $<$ and(at-most(1,mère), at-most(1,père),
all(mère,femme), all(père,homme))
& femme $<$ personne
& homme $<$ and(personne, not(femme))
& marie : femme
& marie.mère $\rightarrow$ linda
& marie.père $\rightarrow$ john
& arsène : homme

Pour une mise à jour demandée M, nous pouvons alors chercher la conjonction H telle que : les vues S permettent de déduire M sous l'hypothèse H, en maintenant la conjonction de contraintes C satisfaite.

Considérons par exemple la mise à jour suivante :

Marie et Arsène ont un parent commun.

Elle s'écrit M = marie.même-parent $\rightarrow$ arsène.

Soit le but $G = M \mathbin{/\!/} C$. De façon opérationnelle, si nous appliquons à G la réduction des buts développée dans la partie 2 nous obtenons deux réponses : l'une contenant arsène.mère $\rightarrow$ linda et l'autre arsène.père $\rightarrow$ john [25] correspondant à deux mises à jour possibles au niveau de l'EDB. L'utilisation des contraintes d'intégrité et des informations provenant de l'EDB a permis d'éliminer lors de la réduction les mises à jour qui seraient la source d'une inconsistance, comme par exemple arsène.mère $\rightarrow$ john.

25. Comme dans le cas des reformulations en optimisation sémantique, les réponses obtenues contiennent aussi d'autre contraintes (par exemples la conjonction C) qu'il faudra éliminer.

Références bibliographiques

[1] Abecker A. and Hanschke P. – TaxLog: A flexible architecture for logic programming with structured types and constraints. *In : Proc. of the Workshop on Constraint Processing held in conjunction with CSAM'93*, Petersburg, July 1993, pp. 77–91.

[2] Abiteboul S. – Towards a deductive object-oriented database language. *Data & Knowledge Engineering*, 1990, vol. 5, pp. 263–287.

[3] Abiteboul S. and Grumbach S. – A rule-based language with functions and sets. *ACM Trans. on Database Systems*, 1991, vol. 16, n° 1, pp. 1–30.

[4] Aït-Kaci H., Meyer R., Van Roy P. and Dumant B. – *Wild LIFE 1.0 User Manual (Draft) Proteus Project*, Paris : DEC-PRL, 1992, 81p.

[5] Aït-Kaci H. and Nasr R. – LOGIN: a logic programming language with built-in inheritance. *Journal of Logic Programming*, 1986, vol. 3, n° 3, pp. 185–215.

[6] Aït-Kaci H. and Podelski A. – Towards a meaning of LIFE. *Journal of Logic Programming*, 1993, vol. 16, n° 3 and 4, pp. 195–234.

[7] Baader F., Bürckert H.-J., Hollunder B., Nutt W. and Siekmann J. – Concept logics. *In : Computational Logic*, edited by Lloyd J. W., Berlin : Springer Basic Research Series, 1990, pp. 177–202.

[8] Baader F. and Hanschke P. – A scheme for integrating concrete domains into concept languages. *In : Proc. of the 12th Int. Joint Conf. on Artificial Intelligence*, Sydney, Australia, August 1991, pp. 452–457.

[9] Bancilhon F., Maier D., Sagiv Y. and Ullman J. – Magic sets and other strange ways to implement logic programs. *In : Proc. of the 5th ACM Symp. on Principles of Database Systems*, Cambridge, Massachusetts, March 1986, pp. 1–16.

[10] Beck H., Gala S. and Navathe S. – Classification as a query processing technique in the CANDIDE semantic data model. *In : Proc. of the Fifth International Conference on Data Engineering*, Los Angeles, California, USA, February 1989, pp. 572–581.

[11] Beeri C., Naqvi S., Shmueli O. and Tsur S. – Set constructors in a logic database language. *Journal of Logic Programming*, 1991, vol. 10, pp. 181–232.

[12] Borgida A., Brachman R., McGuinness D. and Resnick L. – CLASSIC : a structural data model for objects. *In : Proc. of the ACM SIGMOD*

Int. Conf. on Management of Data, Portland, Oregon, June 1989, pp. 58–67.

[13] Brachman R. J. and Schmolze J. G. – An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 1985, vol. 9, n° 2, pp. 171–216.

[14] Bry F. – Query evaluation in recursive databases: Bottom-up and top-down reconciled. *In: Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 1989, pp. 25–44.

[15] Bry F. – Intensional updates: Abduction via deduction. *In: Proc. of the 7th Int. Conf. on Logic Programming*, Jerusalem, June 1990, pp. 561–575.

[16] Buchheit M., Donini F. and Schaerf A. – Decidable reasoning in terminological knowledge representation systems. *In: Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*, Chambery, France, August 1993, pp. 704–709.

[17] Buchheit M., Jeusfeld M. A., Nutt W. and Staudt M. – Subsumption between queries to object-oriented databases. *Information Systems*, 1994, vol. 19, n° 1, pp. 33–54.

[18] Bugliesi M., Lamma E. and Mello P. – Modularity in logic programming. *Journal of Logic Programming*, 1994, vol. 19-20, pp. 443–502.

[19] Bürckert H.-J. and Nutt W. – *On Abduction and Answer Generation through Constrained Resolution*, Saarbrücken, Germany: German Research Center for Artificial Intelligence (DFKI), October 1992, 20p., Rapport technique n° RR-92-51.

[20] Chakravarthy U. S., Grant J. and Minker J. – Logic-based approach to semantic query optimization. *ACM Trans. on Database Systems*, 1990, vol. 15, n° 2, pp. 162–207.

[21] Cluet S. and Delobel C. – A general framework for the optimization of object-oriented queries. *In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, San Diego, California, USA, June 1992, pp. 383–392.

[22] Colmerauer A. – An introduction to prolog III. *Communications of the ACM*, 1990, vol. 33, n° 7, pp. 69–90.

[23] Cruz I. – Doodle: A visual language for object-oriented databases. *In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, San Diego, California, June 1992, pp. 71–80.

[24] Dershowitz N. and Manna Z. – Proving termination with multiset orderings. *Communications of the ACM*, 1979, vol. 22, n° 8, pp. 465–476.

[25] Donini F., Lenzerini M., Nardi D. and Schaerf A. – A hybrid system with datalog and concept languages. *In : Proc. of the 2nd Italian Conf. on Artificial Intelligence*, Palermo, Italy, October 1991, pp. 88–97.

[26] Giordano L. and Martelli A. – Structuring logic programs: a modal approach. *Journal of Logic Programming*, 1994, vol. 21, n° 2, pp. 59–94.

[27] Gloess P. Y. – U-Log, an ordered sorted logic with typed attributes. *In : Proc. of the 4th Int. Symp., PLILP'91*, Passau, Germany, August 1991, pp. 275–286.

[28] Hacid M. S., Mansouri F. and Rigotti C. – Résolution et classification pour l'optimisation sémantique des requêtes dans les BDOO déductives. *In : Actes des 11èmes journées Bases de Données Avancées, BDA'95*, edited by Jomier G., Nancy, France, September 1995, pp. 323–342.

[29] Hacid M. S. and Rigotti C. – Combining resolution and classification for semantic query optimization in DOOD. *In : Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases*, Singapore, December 1995, pp. 447–466.

[30] Höhfeld M. and Smolka G. – *Definite Relations over Constraint Languages*, Stuttgart, Germany : IBM Deutschland, October 1988, 26p., LILOG Report n° 53.

[31] Jaffar J. and Lassez J.-L. – Constraint logic programming. *In : Proc. of the 14th ACM Symposium on Principles of Programming Languages*, Munich, Germany, January 1987, pp. 111–119.

[32] Kakas A. C., Kowalski R. and Toni F. – Abductive logic programming. *Journal of Logic and Computation*, 1993, vol. 2, n° 6, pp. 719–770.

[33] Kakas A. C. and Mancarella P. – Database updates through abduction. *In : Proc. of the 16th Int. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, pp. 650–661.

[34] Kanellakis P. C., Kuper G. M. and Revesz P. Z. – Constraint query langages. *In : Proc. of 9th ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, April 1990, pp. 299–313.

[35] Kifer M., Kim W. and Sagiv Y. – Querying object-oriented databases. *In : Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, San Diego, California, June 1992, pp. 393–402.

[36] Kifer M. and Lausen G. – F-logic: A higher-order language for reasoning about objects, inheritance and scheme. *In : Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Portland, Oregon, June 1989, pp. 134–146.

[37] Kifer M., Lausen G. and Wu J. – Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 1995, vol. 42, n° 4, pp. 741–843.

[38] Kowalski R. – *Logic without Model Theory*, London, England : Imperial College, 1993, 37p., Report of the logic programming group.

[39] Le Provost T. and Wallace M. – Generalized constraint propogation over the CLP scheme. *Journal of Logic Programming*, 1993, vol. 16, pp. 319–359.

[40] Mac Gregor R. – The evolving technology of classification-based knowledge representation systems. *In : Principles of Semantic Networks*, edited by Sowa J., San Mateo, California : Morgan Kaufmann, 1991, pp. 385–400.

[41] Maïm E. – Abduction and constraint logic programming. *In : Proc. of the 10th European Conf. on Artificial Intelligence*, Vienna, Austria, August 1992, pp. 149–153.

[42] Mamede M. and Monteiro L. – A constraint logic programming scheme for taxonomic reasoning. *In : Proc. of the Joint Int. Conf. and Symp. on Logic Programming*, Washington, USA, November 1992, pp. 255–269.

[43] McCabe F. – *Logic and Objects*, New York : Prentice Hall, 1992, 289p.

[44] Mello P. – Inheritance as combination of horn clause theories. *In : Inheritance Hierarchies in Knowledge Representation and Programming Languages*, New York : John Wiley and Sons, 1991, pp. 275–289.

[45] Moss C. – *Prolog++*, New York : Addison-Wesley, 1994, 312p.

[46] Nebel B. – *Reasoning and Revision in Hybrid Representation Systems*, Berlin : Springer-Verlag, 1990, *LNAI*, 270p.

[47] Nilsson U. and Małuszyński J. – *Logic Programming and Prolog*, New York : John Wiley and Sons, 1990, 289p.

[48] Rigotti C., Boulicaut J.-F. and Hacid M. S. – Vers une typologie des sémantiques opérationnelles pour les extensions objets de prolog. *In : Actes des 3ièmes Journées Francophones de Programmation en Logique, JFPL'94*, edited by Corsini M. M., Bordeaux, France, May 1994, pp. 223–237.

[49] Rigotti C., Hacid M. S. and Boulicaut J.-F. – Does a deductive object-oriented system produce object-oriented answers? a hybrid reasoning approach. *In : Actes du 10ième congrès INFORSID'94*, Aix-en-Provence, France, May 1994, pp. 99–110.

[50] Rigotti C., Hacid M. S. and Boulicaut J. F. – F-logic programming and terminological constraints. *In : Proceedings of the Workshop on the Integration of Declarative Paradigms (ICLP'94 Post-Conference Workshop)*, edited by Hanus M., Santa Margherita Ligure, Italy, June 1994, pp. 3–12.

[51] Rigotti C., Hacid M. S. and Boulicaut J.-F. – Une approche multi-paradigmes pour tester la cohérence d'applications BDOO. *In : Actes des 10èmes journées Bases de Données Avancées, BDA'94*, edited by Bidoit N., Clermont-Ferrand, France, September 1994, pp. 55–74.

[52] Shmueli O. – Equivalence of DATALOG queries is undecidable. *Journal of Logic Programming*, 1993, vol. 15, n° 3, pp. 231–241.

[53] Straube D. D. and Özsu M. T. – Queries and query processing in object-oriented database systems. *ACM Transactions on Information Systems*, 1990, vol. 8, n° 4, pp. 387–430.

[54] Vieille L. – Recursive axioms in deductive databases: the query/subquery approach. *In : Proc. of the First Int. Conf. on Expert Database Systems*, Charleston, South Carolina, USA, April 1986, pp. 253–267.

[55] Yoon J. P. and Kerschberg L. – Semantic query optimization in deductive object-oriented databases. *In : Proc. of the 3th Int. Conf. on Deductive and Object-Oriented Databases*, Phoenix, Arizona, USA, December 1993, pp. 169–182.

[56] Zaniolo C. – Object identity and inheritance in deductive databases–an evolutionary approach. *In : Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 1989, pp. 7–24.