Nouvelles techniques pour la construction de modèles finis ou infinis en déduction automatique

Nicolas Peltier

THÈSE

présentée par :

Nicolas PELTIER

pour obtenir le titre de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
(Arrêté ministériel du 30 mars 1992)

en INFORMATIQUE

Nouvelles techniques pour la construction de modèles finis et infinis en déduction automatique

Date de soutenance : 10 octobre 1997

Jury :

Ricardo Caferra	directeur
Hubert Comon	rapporteur
Jean Della Dora	président
Alexander Leitsch	rapporteur
David A. Plaisted	examinateur
Michaël Rusinowitch	examinateur

Thèse préparée au sein du Laboratoire Leibniz de l'Imag

A ma famille,
A mes amis.

"Pour l'artiste scrupuleux, l'œuvre réalisée, quelle qu'en puisse être la valeur, n'est jamais que la scorie de son rêve".

José Maria DE HEREDIA.
(Les Lettres et les Arts, 1886).

Abstract

Although model building is widely recognized as a very important topic in Automated Deduction and Computer Science, the number of works dedicated to this problem is negligible compared to the amount of work devoted to refutational methods. In this thesis, we present several new techniques for model building in Automated Deduction. The thesis is divided in five main parts.

In the first part, we propose a general method for building *finite* models. It uses detection of counter-models and symmetries to prune the search space and favourably compares with the most powerful existing finite model builders (FINDER, SEM, etc.). In the second part, we investigate methods for simultaneous search for refutations and (infinite, Herbrand) models. We improve the methods RAMC[1] and RAMCET[2] defined by R. CAFERRA and N. ZABEL by defining new rules for building models and by defining new strategies. These extensions strictly increase the capabilities of the methods *both* for model building *and* unsatisfiability detection. We also define a new method called EQMC combining enumeration and derivation approaches. We show that some of the proposed methods are *uniform decision procedures* for a wide range of decidable classes including (but not limited to) *all classes decidable by hyperresolution* (e.g. the classes PVD, OCC1N etc.) and more generally any classe decidable by semantic resolution (guided by an interpretation representable by symbolic constraints). In the third part, we show the limits of the formalism of equational constraints, previously used for representing Herbrand models, and we propose to extend it by including *terms with integer exponents (I-terms)* and *tree automata*. As a new result, we prove the decidability of the first-order theory of I-terms. This theoretical result allows us to include I-terms into our method. It has also many applications in Computer Science, different from model building. Fourthly, we study some applications of our work: we present a new approach for discovering and using *analogy* in simultaneous search for refutations and models, and we show how to use the method RAMC in Logic Programming (for extending logic program interpreters, detecting and correcting errors in logic programs etc.). Finally we describe the system $\text{RAMC}_{\text{ATINF}}$ implementing our approach and we report some experiments showing the practical capabilities of our method.

1. **R**efutation **A**nd **M**odel **C**onstruction.
2. **R**efutation **A**nd **M**odel **C**onstruction with **E**quational **T**ableaux.

Remerciements

Je tiens à remercier en premier lieu les membres du jury, qui ont bien voulu apporter à ces travaux le soutien de leur renommée internationale : Jean Della Dora, Professeur à l'Institut National Polytechnique de Grenoble, qui nous fait l'honneur de présider le jury, Alexander Leitsch, Professeur de la *Technische Universität Wien* et Hubert Comon, Directeur de Recherche au CNRS, qui ont accepté la tâche difficile de rapporteur, ainsi que Michaël Rusinowitch, Directeur de Recherche à l'INRIA et David A. Plaisted, Professeur à l'*University of North Carolina*.

Je remercie également Philippe Jorrand, Directeur du laboratoire Leibniz, ainsi que toutes les personnes qui travaillent au sein de ce laboratoire ou de l'INPG, sans lesquelles cette recherche n'aurait pu avoir lieu dans d'aussi bonnes conditions.

Merci également à tous les membres (passés ou présents) du projet Atinf, pour l'ambiance de travail agréable, les conseils et les discussions profitables : Christophe Bourely, Thierry Boy de la Tour, Ricardo Caferra, Gilles Défourneaux, Stéphane Demri, Michaël Dierkes, Stéphane Fèvre, Denis Lugiez, et Dong-Ming Wang.

Je suis tout particulièrement redevable envers toutes les personnes qui ont bien voulu m'aider dans la rédaction de cette thèse, par une relecture attentive de mon manuscrit : Thierry Boy de la Tour, Ricardo Caferra, ainsi que mes parents. Leurs critiques constructives et toujours pertinentes ont grandement contribué à améliorer le fond et la forme de ce travail.

Je tiens surtout à remercier ici mon Directeur de thèse, Ricardo Caferra pour m'avoir accueilli au sein de son équipe, et pour les encouragements et conseils prodigués tout au long de ce travail. Il a su me faire profiter de son experience et de ses grandes connaissances scientifiques, qui m'ont été d'une aide précieuse dans la préparation de cette thèse.

Chapitre 1

Introduction

1.1 Logique et mécanisation du raisonnement

> "Computers ought to produce in the long run some fundamental change in the nature of all mathematical activity"
>
> Wang (1960).

La mécanisation du raisonnement (ou *Déduction Automatique*) se situe au cœur de ce que l'on nomme communément l'Intelligence Artificielle. Son objet est de chercher à doter l'ordinateur de capacités de raisonnement similaires à celles de l'homme, ou encore de mécaniser la logique (Davis, 1983; Loveland, 1984). La mécanisation du raisonnement passe par l'étude du raisonnement humain et par sa formalisation, qui constitue une étape préliminaire indispensable à toute réalisation sur machine. Ainsi, les préoccupations et contraintes des chercheurs en Démonstration Automatique rejoignent largement celles des logiciens qui, bien avant l'apparition de l'ordinateur, se sont intéressés à l'étude du raisonnement en tant que tel, c'est-à-dire en dehors de tout domaine d'application. Ces aspirations trouvent leur origine dans la plus haute antiquité : on attribue généralement à Aristote le mérite de s'être le premier intéressé au raisonnement lui-même, en tant qu'objet d'étude à part entière. En introduisant la notion de *syllogisme*, il cherche à fournir des critères permettant de caractériser de manière précise la correction d'un raisonnement. Il formule des règles permettant d'accepter certaines propositions (*conclusions*) comme valides, à partir d'autres propositions considérées comme vraies (*prémisses*). Un exemple typique de syllogisme (formulé par l'école Stoïcienne) est le *Modus Ponens* qui énonce que si les assertions A et $A \Rightarrow B$ ("A implique B") sont vraies alors l'assertion B est vraie.

Leibniz fut l'un des premiers à introduire l'idée d'un "langage mathématique universel" dont le pouvoir d'expression serait équivalent à celui du langage naturel mais qui serait exempt de toute ambiguïté ou imprécision. Cette formalisation permettrait de définir les propositions et les règles de raisonnement de façon aussi précise que le langage mathématique, et de réduire ainsi le raisonnement à un *calcul*, qui ne laisserait aucune place à l'opinion ou au jugement personnel, mais permettrait d'établir sans conteste la validité - ou l'inexactitude - de toute assertion :

"Je crois qu'on ne pourra jamais mettre fin aux contreverses [...] si l'on ne revient pas des raisonnements compliqués à des calculs simples, et des mots ayant des significations vagues et incertaines à des caractères bien déterminés. Car il arrivera alors que tout paralogisme ne sera rien de plus qu'une erreur de calcul ..." (Leibniz)

Les travaux de Boole, en particulier, constituent la première tentative de définition d'un langage répondant aux exigences de Leibniz. Les travaux de Gödel d'abord, complétés par

ceux de TARSKI et CHURCH, mettent rapidement en évidence les limites de la conception de LEIBNIZ, du moins dans sa version universelle.

La formalisation du raisonnement passe habituellement par la définition d'un *système formel*, c'est-à-dire par la définition d'*axiomes* (propositions arbitrairement supposées valides) et de *règles d'inférence*, permettant de déduire de nouvelles formules valides à partir des formules valides connues. Un système formel permet, en fournissant une définition inductive de l'ensemble des formules valides, d'énumérer systématiquement cet ensemble en partant des axiomes et en appliquant les règles d'inférence de toutes les façons possibles. Il fournit ainsi une définition *syntaxique* des formules valides, ce qui permet de ramener le raisonnement à un calcul, à une manipulation de symboles suivant des règles algorithmiques formellement définies, qui peut alors être reproduit sur machine. Cette approche permet de modéliser l'aspect purement *déductif* du raisonnement, qui consiste à déduire de nouvelles formules d'un ensemble de formules connues. Mais il ne s'agit là que d'un aspect du raisonnement. Un autre aspect tout aussi important, et tout aussi couramment utilisé, concerne la construction de *modèles*.

1.2 Rôle des modèles dans la mécanisation du raisonnement

Un *modèle* (ou *contre-exemple*) d'une formule logique est une structure particulière vérifiant (ou ne vérifiant pas) la formule. Considérons, par exemple, la formule $\mathcal{F}$ suivante :

$$\exists x . \forall y . f(y) \neq x .$$

Un *modèle* de $\mathcal{F}$ est l'ensemble $\mathbb{N}$, où la fonction f est interprétée comme la fonction successeur : $f(y) = y + 1$. $\mathcal{F}$ exprime alors le fait qu'il existe un entier x tel que pour tout y, le successeur de y est distinct de x ($x = 0$). En revanche, si f est interprétée (toujours sur le domaine $\mathbb{N}$) comme la fonction prédécesseur $f(y) = y - 1$, alors $\mathcal{F}$ devient fausse (car pour tout x il existe un y tel que $y - 1 = x$). La structure correspondante est alors appelée un *contre-exemple* de $\mathcal{F}$.[1]

Les modèles permettent de donner une définition de la notion de validité qui est indépendante de la syntaxe de la formule. Une formule sera dite *valide* si elle ne possède pas de contre-exemple, *insatisfaisable* si elle ne possède pas de modèle.

Les modèles ont de nombreuses applications dans des domaines aussi variés que la logique, les mathématiques, l'Informatique ou l'Intelligence Artificielle. L'utilisation de modèles se situe au cœur du raisonnement humain et s'avère souvent indispensable à la compréhension et à la démonstration d'un théorème. En effet, il semble que la recherche d'une preuve d'une formule par un être humain ne s'effectue pas sur des critères purement *syntaxiques* (c'est-à-dire sur la façon dont la formule est écrite), mais plutôt sur des critères essentiellement *sémantiques* (c'est-à-dire dépendants du sens attribué à la formule). Un modèle fournit des informations d'ordre sémantique sur une formule. Ainsi en géométrie, la construction d'une figure est souvent indispensable à la compréhension et à la résolution d'un problème. Le modèle permet également de réfuter une conjecture : la manière la plus simple et la plus convaincante de prouver qu'une formule n'est pas valide est d'en donner un contre-exemple. Celui-ci permet en outre de comprendre *pourquoi* la conjecture est fausse et éventuellement de la corriger pour en faire un théorème (en rajoutant des hypothèses supplémentaires ou en affaiblissant la conclusion). Cette technique est utilisée par les mathématiciens dans l'étude des conjectures.

Disposer de systèmes capables non seulement de prouver un théorème mais également de construire, dans le cas où la formule proposée n'est pas valide, un contre-exemple de la formule

1. Dans la suite, les termes "contre-exemple" ou "modèle" seront souvent employés indifféremment l'un à la place de l'autre, un contre-exemple d'une formule A étant un modèle de la négation de A et réciproquement. De même, une *preuve* de la formule A est une *réfutation* de la négation de A et réciproquement.

serait donc d'un grand intérêt pratique pour de nombreuses applications en logique ou en mathématiques (pour établir l'indépendance d'axiomes, réfuter des conjectures, etc.) mais également en programmation (pour *détecter* et éventuellement *corriger* des erreurs dans des programmes ou dans des spécifications), en vérification de circuit, etc. L'utilisation de contre-exemples étant au cœur de l'activité mathématique, un constructeur automatique ou interactif de modèles apparaît comme une composante essentielle de tout système d'aide à la démonstration.

Les modèles ont également des applications en Déduction Automatique : ils permettent en particulier de *guider la recherche d'une preuve*. Pour la plupart des problèmes, la simple application des règles d'inférence ne permet pas de démontrer la validité du théorème, à cause de la taille souvent très importante de l'espace de recherche. Il est nécessaire de définir des *stratégies* ou *heuristiques* afin de guider l'application des règles. L'utilisation d'un modèle du théorème à démontrer permet, en fournissant des informations sémantiques sur ce théorème, de guider le choix des règles à appliquer, ou l'ordre d'application de ces règles, de façon à la fois naturelle et efficace. Cette idée a été rapidement reconnue comme très importante en Démonstration Automatique : par exemple dès 1960, le démonstrateur GTM pour la géométrie plane utilise des figures géométriques comme heuristiques afin de guider l'exploration de l'espace de recherche (GELERNTER ET AL., 1983). Quelques années plus tard, SLAGLE propose une stratégie de restriction de la méthode de résolution fondée sur l'utilisation d'un modèle (SLAGLE, 1967). Dans les deux cas, les modèles considérés devaient être fournis par l'utilisateur, ce qui limitait sensiblement l'intérêt de ce type d'approche. Disposer de systèmes permettant de construire *automatiquement* de tels modèles serait donc d'un grand intérêt pour l'utilisation de ces stratégies. Comme le remarque SLAGLE lui-même :

"A disadvantage of semantic strategy is that the program must at present be given a model along with the theorem to be proved. One might hope that someday the program would devise its own models" (SLAGLE, 1967).

La recherche de modèles introduit un aspect *sémantique* dans le raisonnement qui s'avère être un complément indispensable aux approches déductives classiques. Les travaux de SLANEY (SLANEY, 1993) illustrent par ailleurs la complémentarité de ces deux types de méthodes en pratique. La construction de modèles s'affirme de plus en plus comme un élément essentiel de tout système de démonstration automatique ou interactive.

En dépit de ces remarques, il n'existe que très peu de travaux qui se sont confrontés à ce problème, en regard de ceux portant sur la conception et l'amélioration des procédures de recherche de preuves. Ce déséquilibre peut sembler surprenant, mais n'est en fait qu'une conséquence de la difficulté intrinsèque du sujet. En effet, la construction de modèles pose des problèmes théoriques et pratiques considérables : l'existence d'un modèle pour une formule logique du premier ordre est évidemment un problème *indécidable* (c'est-à-dire que l'on peut prouver qu'il n'existe aucun algorithme permettant de résoudre ce problème dans le cas général). Si on se restreint à des modèles finis, ce problème devient semi-décidable (il est alors possible d'énumérer toutes les interprétations et de tester pour chacune d'elles si elles constituent un modèle de la formule initiale), mais on se heurte à des problèmes pratiques liés à la taille souvent prohibitive de l'espace de recherche, même pour des formules possédant des modèles de cardinalité faible.

Le problème de prouver qu'une formule n'est pas valide, ainsi que celui de chercher un modèle de la formule ont été largement délaissés aussi bien par les logiciens que par les chercheurs en Déduction Automatique. A cause des insurmontables limites théoriques, les démonstrateurs de théorèmes sont souvent incapables de prouver la non-validité d'une formule : en effet, l'espace de recherche à explorer étant, dans la plupart des cas, infini, les démonstrateurs ne terminent pas en général lorsque la formule proposée n'est pas un théorème. Dans certains cas très particuliers, un démonstrateur peut, en explorant la totalité de l'espace de recherche, établir la non-validité de la formule proposé. Cela consiste à réfuter la formule proposé, en montrant – par une exploration

systématique – que celui-ci *ne peut pas être démontré*. On obtient ainsi une *procédure de décision* pour une certaine classe de formules. Cependant, cette approche n'apporte aucune information supplémentaire sur la formule initiale, en particulier il est extrêmement difficile – sinon impossible – pour un utilisateur de comprendre la raison pour laquelle la formule est non-valide.

Quelques approches doivent pourtant être mentionnées. Dans la section suivante, nous présenterons brièvement les différentes approches existantes dans le domaine de la construction de modèles. Nous ne chercherons pas à présenter de façon exhausitive et formelle toutes les méthodes mais nous nous attacherons simplement à préciser leur principe, ainsi que leur principaux avantages et limites.

REMARQUE. Dans cette section, nous serons amenés à utiliser quelques termes techniques. Le lecteur peu familier avec les notions usuelles en logique et démonstration automatique est invité à se reporter au chapitre 2 où toutes les définitions importantes sont précisément rappelées.

1.3 Historique

1.3.1 Preuve de la non-validité d'une formule

Bien que l'on sache qu'ARISTOTE fut le premier à formuler des règles permettant de définir précisément la correction d'un raisonnement, on oublie souvent qu'il a également été le premier à introduire l'idée de règles permettant de *rejeter* certaines propositions à partir d'autres, c'est-à-dire de montrer que ces propositions *ne sont pas valides* (ŁUKASIEWICZ, 1972). Par la suite, ŁUKASIEWICZ (SLUPECKI ET AL., 1971) introduit le terme de *règles de rejet* (rules of rejection) pour qualifier ce type de règles analogues aux règles d'inférence. Ses travaux portaient essentiellement sur le calcul propositionnel classique, bien que des extensions soient considérées dans le cadre des logiques modales pour lesquelles il propose un système à quatre valeurs de vérité. ŁUKASIEWICZ a également proposé d'utiliser les règles de rejet pour établir la non-validité en logique intuitioniste des théorèmes classiques considérés comme faux par les intuitionistes. Cette approche a été reprise par l'école Polonaise, qui s'est intéressée à l'étude de la notion de règles de rejet.

Dans (CAICEDO, 1978), un système formel pour la déduction de non-théorèmes dans le calcul propositionnel classique est donné.

Beaucoup de logiciens et de chercheurs en Démonstration Automatique se sont intéressés à la recherche de *procédures de décision* (voir par exemple (DREBEN ET GOLDFARB, 1979; BÖRGER ET AL., 1997)), c'est-à-dire de procédures terminant nécessairement et permettant d'établir la validité ou la non-validité d'une formule. A cause des résultats d'indécidabilité mentionnés plus haut, cela n'est possible que pour des logiques particulières ou pour certaines classes particulières de formules. Néanmoins, si ces approches permettent de prouver qu'une formule n'est pas valide, elles ne permettent pas en général d'en donner explicitement un contre-exemple.

Dans (TIOMKIN, 1988), TIOMKIN définit un système formel pour établir la non-validité d'une formule logique du calcul des prédicats du premier ordre (sans symbole de fonction). Ce système, inspiré du calcul des séquents, est complet pour l'ensemble des formules *possédant un contre-exemple fini*, c'est-à-dire qu'il permet d'énumérer l'ensemble des formules ayant un contre-exemple sur un domaine fini. TIOMKIN montre également comment construire explicitement un contre-exemple pour ces formules. A cet égard, il est intéressant de noter que l'ensemble des formules possédant un modèle fini est énumérable (ce qui signifie qu'il existe un algorithme permettant de construire un tel modèle s'il existe, et ne se terminant pas sinon) mais *non co-énumérable*, autrement dit que l'ensemble des formules n'admettant pas de modèle fini n'est pas énumérable. Le système proposé par TIOMKIN ne permet pas de prouver qu'une formule n'admettant pas de contre-exemple fini est non-valide. L'intérêt pratique de ses travaux reste très limité dans la mesure où les règles proposées reviennent en pratique à rechercher de façon

systématique (par énumération) une interprétation sur un domaine fini falsifiant la formule. Son approche peut être rapprochée de la méthode des tableaux sémantiques, dont le principe est de chercher à construire systématiquement un contre-exemple de la formule.

1.3.2 Construction de modèles en Déduction Automatique

L'un des premiers résultats pratiques importants en construction de modèles par ordinateur fut la démonstration assistée par ordinateur de l'indépendance des axiomes de l'algèbre ternaire. Ce résultat est resté longtemps un problème ouvert (entre 1947 et 1977). Il fut obtenu par Wos et Winker (Winker, 1982; Wos, 1993a) avec l'aide d'un démonstrateur par résolution. Néanmoins, cette approche ne définissait pas une méthode générale et automatique de construction de modèles, dans la mesure où toutes les informations importantes étaient données par l'utilisateur. Ainsi Wos reconnait en 1993 :

"I also note that our programs are still not effective for model generation".

Ce n'est qu'à partir des années 1990 que des méthodes *purement automatiques* de construction de modèles ont vu le jour. Elles peuvent être classées en deux catégories : méthodes par énumération et méthodes fondées sur l'utilisation de procédures de preuves (résolution, tableaux, hyper-linking).

Construction de modèles par énumération

Une première façon d'essayer de construire un modèle d'une formule logique donnée est d'énumérer l'ensemble des interprétations sur un domaine fini. Plusieurs approches sont alors envisageables. La plus simple d'un point de vue pratique consiste à fixer le domaine et à traduire la formule considérée sur le domaine en une formule propositionnelle "équivalente" (sur le domaine) à la formule de départ. Pour cela, il suffit de se débarrasser des symboles de fonctions en introduisant de nouveaux symboles de prédicats (par l'équivalence : $P(x_1, \ldots, x_n, y) \leftrightarrow f(x_1, \ldots, x_n) = y$). Il suffit alors d'ajouter pour chacun de ces nouveaux prédicats P la formule : $\forall x_1, \ldots, x_n . \exists! x . P(x_1, \ldots, x_n, x)$ assurant que l'interprétation de P est une fonction. Puisque le domaine est fini, il est possible de remplacer les quantificateurs existentiels et universels par des disjonctions et des conjonctions de formules.

On est ainsi ramené à une formule ne contenant aucune variable, c'est-à-dire à une formule propositionnelle, et le problème est alors équivalent au problème **SAT** (recherche d'un modèle d'un ensemble de clauses propositionnelles) qui est évidemment décidable (mais très difficile, puisqu'il s'agit d'un problème NP-complet) et pour lequel des systèmes performants ont été développés et continuent à l'être actuellement (tels que Ddpp, Sato, Mace...). L'inconvénient de cette approche est que la taille de l'ensemble de la formule propositionnelle obtenue par traduction, ainsi que le nombre de variables à considérer, croît exponentiellement avec la taille du domaine, ce qui limite l'intérêt de ce type d'approche.

Il est également possible d'utiliser une approche *directe* afin d'éviter une telle traduction. Des méthodes spécifiques pour la construction de modèles ont ainsi été proposées (Slaney, 1993; Zhang, 1993; Zhang, 1994; Zhang et Zhang, 1995). Elles sont fondées sur des principes similaires à ceux mis en œuvre pour résoudre les problèmes de satisfaction de contraintes : propagation et renforcement de contraintes et retour-arrière.

Cependant dans tous les cas, la taille de l'espace de recherche reste souvent prohibitif et des heuristiques puissantes doivent être utilisées pour restreindre l'espace de recherche. La taille des modèles constructibles par ce type de méthode reste pour l'instant limitée à 15-20 éléments.

Construction de modèles et procédures de preuves

Plusieurs approches ont été proposées afin d'utiliser des méthodes existantes de recherche de preuve pour construire un modèle d'une formule.

La méthode des tableaux sémantiques La méthode des *tableaux sémantiques* est — dans son principe même — bien adaptée à la recherche de modèles. En effet, le principe de cette approche est précisément d'énumérer l'ensemble des modèles de Herbrand de la formule. Chaque branche non fermée du tableau correspond à un modèle de Herbrand de la formule. En général, les branches non fermées sont infinies, ce qui limite l'intérêt pratique d'une telle énumération. Le démonstrateur par construction de modèles SATCHMO (MANTHEY ET BRY, 1988), combinant l'utilisation de l'hyper-résolution, de la décomposition et du retour-arrière peut être considéré comme un raffinement de la méthode des tableaux sémantiques (BRY ET YAHYA, 1996). SATCHMO permet dans certains cas très particuliers de construire un modèle de Herbrand de la formule. Ce modèle est spécifié sous forme d'un ensemble fini de littéraux (positifs) fermés.

La représentation choisie ne permet naturellement pas de représenter des ensembles *infinis* de littéraux. Elle ne peut donc être utilisée que pour traiter des cas très particuliers de formules. Par exemple, des formules sous forme clausale ne contenant pas de symbole de fonction d'arité 1 ou plus (la classe dite de *Bernays-Schönfinkel* i.e. forme prénexe avec préfixe de la forme $\exists z_1, \ldots, z_n, \forall y_1, \ldots, y_m$), pour lesquelles les modèles de Herbrand sont nécessairement finis (puisque l'univers de Herbrand est fini). Il est à noter qu'une version plus puissante (parallèle) de SATCHMO, nommée MGTP est décrite dans (FUJITA ET HASEGAWA, 1991). La méthode utilisée par les démonstrateurs SATCHMO et MGTP est restreinte à une classe de formules, dite "range-restricted". Néanmoins, il est très facile de voir que toute formule peut être transformée en une formule équivalente de cette classe.

La méthode des Hyper-Tableaux proposée par BAUMGARTER, FURBACH et NIEMELA (BAUMGARTNER ET AL., 1996) est une généralisation de SATCHMO dont le principal intérêt est d'éviter la transformation en formules "range-restricted" ce qui semble améliorer les performances pratiques du système. Du point de vue de la construction de modèles, les hyper-tableaux utilisent des variables universelles, ce qui permet dans certains cas de représenter des ensembles d'atomes infinis et étend donc la classe de modèles constructibles.

La méthode de résolution La méthode de *résolution* proposée par ROBINSON (ROBINSON, 1965) est l'un des calculs des prédicats les plus employés en Déduction Automatique (voir également (FERMÜLLER ET AL., 1993; LEITSCH, 1997)). Conçu pour être mis en œuvre sur machine, ce calcul se caractérise par sa simplicité et son uniformité. A cause du succès obtenu par la méthode de résolution et du nombre considérable de travaux portant sur cette approche, il est très naturel de chercher à utiliser cette méthode pour construire des modèles de formules logiques. Les solutions proposées consistent à utiliser des raffinements de la méthode de résolution (c'est-à-dire des stratégies particulières permettant de restreindre ou de guider l'application des règles). Elles procèdent essentiellement par *saturation*, c'est-à-dire en générant l'ensemble S des conséquences pouvant être déduites de la formule initiale, puis en extrayant un modèle de l'ensemble obtenu. Ce type d'approche reste cependant limité — dans son principe même — à certaines classes particulières de formules, car l'ensemble S est en général infini. D'autre part même si S est fini, sa taille peut être très importante, ce qui limite l'intérêt pratique de ce type d'approche.

- Dans (TAMMET, 1991), TAMMET propose d'utiliser une stratégie d'ordonnancement (fondée sur l'utilisation d'un *ordre sur les termes*) comme procédure de décision pour une classe particulière de formules : la classe *Ackermann-monadique* (classe AM), qui est une extension de la classe Ackermann (forme prénexe avec préfixe $\forall y.\exists x_1, \ldots, x_n$) et de la classe monadique (i.e. ne contenant que des prédicats d'arité 1). Il montre ensuite comment extraire un modèle fini de la formule à partir des ensembles obtenus par saturation. La méthode proposée consiste à

ajouter à l'ensemble de clauses des équations permettant de "fermer" le domaine de Herbrand (c'est-à-dire l'ensemble des termes en général infini) en un domaine fini. Par exemple, l'équation $f(f(a)) = a$ permet de transformer le domaine infini : $\{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$ en $\{a, f(a)\}$. La cohérence de ces nouvelles équations avec la formule initiale est vérifiée par la procédure de résolution ordonnée et par des techniques de réécriture.

Notons que la méthode proposée permet de construire l'interprétation des symboles de fonctions mais pas celle des symboles de prédicats qui doit être construit par une autre méthode (non précisée). D'autre part, la méthode ne peut semble-t-il pas être étendue facilement à d'autre classes que la classe AM.

- Dans (FERMÜLLER ET LEITSCH, 1996), est proposée une méthode fondée sur l'utilisation de l'*hyper-résolution*, qui est une variante de la résolution sémantique introduite par SLAGLE (SLAGLE, 1967). Leur méthode est restreinte à une classe particulière de formules S appelée *clauses positivement décomposées* (PVD), vérifiant les deux conditions suivantes.

  - L'hyper-résolution se termine sur S.
  - Pour toute clause C positive obtenue par hyper-résolution sur S et pour tout couple de littéraux (L_1, L_2) de C, L_1 et L_2 ne partagent aucune variable.

  Elle peut être décomposée en deux phases distinctes.

  1. La première consiste à utiliser l'hyper-résolution et la décomposition pour construire un modèle de Herbrand de la formule. Le modèle est exprimé sous forme d'un ensemble fini d'*atomes*. Ce type de représentation est plus général que celui utilisé par les démonstrateurs comme SATCHMO car les atomes peuvent contenir des variables. Ainsi l'ensemble infini : $\{P(a), P(f(a)), P(f(f(a))), \ldots\}$ peut être exprimé par l'ensemble d'atomes : $\{\forall x.P(x)\}$ (sur la signature $\{a, f\}$).
  2. La deuxième phase transforme le modèle de Herbrand obtenu en un modèle fini. Cette deuxième phase suppose que le modèle vérifie une condition de *linéarité* imposant qu'une variable ne peut avoir qu'une seule occurrence dans un atome donné (par exemple l'atome $\forall x.P(x, x)$ est exclu).

La méthode de FERMÜLLER et LEITSCH a par la suite été étendue à une classe de formule contenant des littéraux équationnels (fermés) (FERMÜLLER ET LEITSCH, 1996).

Hyper-linking La méthode d'*hyper-linking* a été mise au point par H. LEE et D. PLAISTED (LEE ET PLAISTED, 1992; LEE ET PLAISTED, 1994a). Il s'agit d'une application du théorème de Herbrand (qui énonce que tout ensemble de clauses fermées insatisfaisable contient un sous-ensemble *fini* insatisfaisable). Ce théorème suggère une méthode pour réfuter un ensemble de clauses donné : elle consiste à énumérer toutes les instances possibles de ces clauses et à tester au fur et à mesure la satisfaisabilité de l'ensemble de clauses fermées obtenu (en utilisant une procédure de décision pour le calcul propositionnel). Evidemment, cette méthode ne s'avère pas d'une grande utilité pratique si aucune méthode n'est utilisée afin de guider la génération des instances fermées (GILMORE, 1960). L'idée de la méthode d'hyper-linking est d'utiliser l'*unification* pour générer de nouvelles instances fermées. Une instance $C\theta$ d'une clause C sera générée si et seulement si pour chaque littéral L de C il existe un littéral $\neg R$ dans l'ensemble de clauses tel que $L\theta = R\theta$. (LEE ET PLAISTED, 1994b) montre comment utiliser cette approche pour construire des modèles de certains ensembles de clauses. Là encore, les modèles sont représentés par des ensembles d'atomes fermés, donc seuls des modèles de Herbrand *finis* peuvent être obtenus. Dans (LEE ET PLAISTED, 1994a) est proposée une variante de la méthode de Hyper-Linking, nommée *Hyper-Linking sémantique* où le choix de la substitution θ est guidé par l'utilisation

d'une interprétation $\mathcal{I}$. Dans (Chu et Plaisted, 1997), est décrit un démonstrateur fondé sur une combinaison de l'hyper-résolution sémantique avec la règle de résolution.

Dans (Paramasivam et Plaisted, 1997), une technique utilisant un construction de modèles finis pour réalisée des tests de subsomption est proposée.

Conclusion

Quelques remarques générales se dégagent de l'étude de ces différentes méthodes. D'une part, on peut souligner que la plupart des méthodes existantes cherchent à construire un modèle *fini*, et sont donc inutilisables pour des classes non finiment contrôlables[2]. L'espace de recherche augmente très rapidement avec la taille du domaine, ce qui rend la plupart des méthodes inutilisables pour des formules n'admettant que des modèles de cardinalité importante. L'utilisation de représentations atomiques (comme dans (Fermüller et Leitsch, 1996)) permet de surmonter cette limite. Cependant, le pouvoir d'expression des ensembles d'atomes reste très limité. D'autre part, ces méthodes sont souvent spécifiques à une classe particulière de formules, et ne peuvent apparemment pas être généralisées facilement, puisqu'elles sont fondées sur des propriétés particulières de ces classes. Elles ne constituent donc pas une méthode générale de construction de modèles.

Notons également que tous les travaux recensés s'intéressent uniquement à la logique du premier ordre (ou à des fragments de la logique du premier ordre). La construction de modèles dans des logiques plus expressives n'a (à notre connaissance) fait l'objet d'aucune étude.

Recherche simultanée de réfutation et de modèles

A partir de 1990, Caferra et Zabel ont proposé une approche générale et originale permettant de rechercher simultanément un contre-exemple et une preuve d'une formule du premier ordre (Caferra et Zabel, 1990; Caferra et Zabel, 1992). Cette approche, nommée Ramc[3], cherche à formaliser et à reproduire l'attitude usuelle d'un mathématicien confronté à une nouvelle conjecture : chercher *en même temps à construire un contre-exemple et une preuve de la formule*.

Le principe est d'exprimer, parallèlement à l'application des règles de réfutation habituelles, les conditions qui *empêchent* leur application. Pour cela, on utilise des contraintes qui permettent de restreindre le domaine des clauses classiques. Ces contraintes constituent des "sortes dynamiques" qui sont raffinées successivement jusqu'à obtenir des littéraux purs, pouvant faire partie du modèle. On introduit ainsi le concept de règles de "disinférence" similaire à celui de règles d'inférence pour la construction de modèles et qui peut être rapproché de la notion de "règles de rejet" introduit par Lukasiewitcz.

Cette approche peut être utilisée pour étendre les méthodes classiques de recherche de réfutation : résolution (Caferra et Zabel, 1992), tableaux sémantiques (Caferra et Zabel, 1993), méthode des connections etc. Nous reviendrons plus en détail par la suite sur le principe de Ramc. Nous nous contentons ici de préciser les avantages et les limites de la méthode proposée. Les principaux avantages de la méthode par rapport à celles précédemment mentionnées résident dans sa généralité. Elle peut être utilisée pour étendre plusieurs types de procédures de preuves. Elle n'est pas restreinte à une classe particulière de formules et permet de construire des modèles finis ou infinis d'une formule logique. D'autre part, elle permet de combiner la construction de modèles et la recherche de réfutation, ce qui permet souvent d'élaguer l'arbre de recherche. La méthode, initialement définie pour des clauses sans égalité (Caferra et Zabel, 1992), a par la suite été étendue au cas avec égalité dans (Bourely et al., 1994) (voir aussi (Bourely, 1992; Bourely, 1998)). Cependant, à cause de la présence de nouvelles règles et de l'absence de stratégie ou d'heuristique pour guider l'application des règles, l'espace de recherche

2. Une classe de formules est dite *finiment contrôlable* si et seulement si toute formule satisfaisable de cette classe admet un modèle fini

3. Pour **R**efutation **A**nd **M**odel **C**onstruction

est souvent très important. Les propriétés théoriques de la méthode (terminaison, classes de formules pour lesquelles on peut construire des modèles, etc.) sont d'ailleurs très peu connues, en dehors des propriétés usuelles de correction et complétude pour la réfutation.

L'étude et l'amélioration de la méthode RAMC constitue une part importante de ce travail.

1.4 Objectif et organisation du mémoire

L'étude de l'existant (section 1.3) montre que, bien que la construction de modèles soit reconnue comme un problème d'une grande importance théorique et pratique, les travaux dans ce domaine restent peu nombreux et d'une portée limitée. Elle montre également la nécessité d'une approche plus générale et plus systématique de la construction de modèles.

L'objectif de ce travail est double. D'une part, il s'agit de définir des méthodes ou techniques pour la représentation et la construction de modèles et d'évaluer leurs possibilités et leurs limites. D'autre part, nous nous intéressons aux applications possibles de notre étude en Déduction Automatique, Logique, Intelligence Artificielle ou Informatique. De par la difficulté intrinsèque du sujet, aucune méthode générale pour la construction de modèles n'est envisageable. Nous avons donc cherché à définir un large éventail d'approches différentes s'avérant utiles en pratique sur certaines classes de problèmes particuliers, avec l'objectif d'étendre autant que possible la classe de modèles constructibles. La principale difficulté est de trouver un compromis entre le *pouvoir d'expression* des formalismes utilisés pour représenter les modèles et l'*efficacité* des procédures de vérification et de construction de modèles associées. Par ailleurs, nous avons cherché à *combiner* notre approche avec des techniques couramment utilisées en Déduction Automatique, Intelligence Artificielle, Programmation...(résolution sémantique, analogie, négation en Programmation Logique...), avec le double objectif d'améliorer les capacités et les performances de notre méthode et d'utiliser ses capacités pour étendre la portée de ces techniques.

Ce travail théorique s'est concrétisé par la réalisation d'un système opérationnel de recherche simultanée de preuves et de contre-exemples appelé $\text{RAMC}_{\text{ATINF}}$ qui constitue une partie du système ATINF[4] développé au sein du projet ATINF (pour plus de détails sur le projet ATINF, voir (CAFERRA ET HERMENT, 1993; CAFERRA ET HERMENT, 1995; CAFERRA ET AL., 1991)).

Précisons maintenant l'organisation et le contenu de la thèse.

Construction de modèles finis

Nous nous intéressons dans un premier temps à la recherche de modèles *finis*. La nécessité de cette étude est motivée par les remarques suivantes.

- D'une part, la recherche de modèles finis est un problème intrinséquement plus simple que celui d'un modèle infini (puisque les interprétations peuvent être facilement représentées sur ordinateur), et se révèle d'une grande utilité dans l'étude d'une conjecture (voir par exemple (FUJITA ET AL., 1993)). Ainsi, un constructeur de modèles finis nous paraît être une part essentielle de tout système de construction de modèles.

- D'autre part, l'étude spécifique des structures finies est d'une grande importance en Informatique et en Logique. Dans certaines applications, on considère *uniquement* des structures finies : par exemple en théorie des graphes, ou encore programmation (par exemple pour la synthèse de programmes à partir de spécifications en logique temporelle.

Nous proposons donc une méthode de construction de modèles sur un domaine fini, fondée sur des techniques d'énumération de l'ensemble des interprétations. Le critère primordial du succès de ce type d'approche est la réduction de l'espace de recherche, aussi proposons-nous des stratégies permettant de réduire de façon importante le nombre d'interprétations à considérer. Nous décrivons précisément le principe de notre méthode et donnons quelques éléments de

4. **At**elier d'**Inf**érence.

comparaison avec les approches existantes. Nous montrons également comment étendre cette méthode à d'autres logiques.

Construction de modèles infinis

Dans une seconde partie, le problème de la construction de modèles *infinis* et plus précisément de modèles de Herbrand (modèles dont le domaine est l'algèbre des termes sur la signature considérée) est abordé. Cette extension est nécessaire car, d'une part, il existe des formules satisfaisables *n'admettant pas de modèles finis* et, d'autre part, les méthodes de construction de modèles par énumération s'avèrent inutilisables en pratique, si le modèle est de cardinalité importante.

La construction de modèles infinis pose un double problème.

- D'une part, celui de la *représentation* des interprétations. Un modèle fini peut être représenté par des tables donnant la valeur de chacun des symboles de fonction. En revanche, il n'existe aucun formalisme permettant de représenter un modèle infini (et en particulier un modèle de Herbrand) puisque le nombre de modèles possibles n'est pas dénombrable. Il convient donc de rechercher des formalismes ayant un pouvoir d'expression suffisamment important pour pouvoir représenter les interprétations utiles en pratique, mais étant dans le même temps suffisamment simples pour permettre une manipulation aisée des interprétations. Dans cette thèse, nous utilisons des *contraintes* pour représenter les interprétations de Herbrand (plus précisément le domaine de validité des symboles relationnels).
- D'autre part, il s'agit de résoudre le problème de la construction *automatique* de telles représentations.

Ce travail est limité à la logique du premier ordre, même s'il est probable que certaines des méthodes présentées peuvent être étendues à d'autres logiques[5].

Etude et extension de la méthode RAMC

L'extension de la méthode RAMC proposée par Ricardo CAFERRA et Nicolas ZABEL a constitué le point de départ de ce travail. Nous montrons les possibilités et les limites de l'approche initiale et nous proposons de nouvelles règles de dis-inférence et des extensions de certaines des règles-clefs (en particulier la règle GPL) de la méthode afin de remédier à ces difficultés.

L'un des inconvénients de la version initiale de RAMC est que la taille de l'espace de recherche est importante (c'est une conséquence du fait qu'elle recherche *en même temps* un modèle et une réfutation). Cela est dû, d'une part, à la présence de nouvelles règles (règles dites de *dis-inférence*), et, d'autre part, à l'absence de stratégies pour guider l'application de ces règles. Afin d'apporter une réponse partielle à ces problèmes, nous proposons d'utiliser des stratégies *sémantiques*, c'est-à-dire des stratégies destinées à réduire l'espace de recherche, fondées sur l'utilisation d'un modèle (éventuellement construit automatiquement) de la formule à démontrer. L'approche proposée permet d'améliorer de façon importante les performances de la méthode à la fois pour la recherche de preuve et pour la recherche de contre-exemple.

Nous introduisons une nouvelle classe de formules nommée $\mathfrak{C}_{\text{eq-model}}$, définie comme l'ensemble des formules admettant un modèle exprimable dans le formalisme des contraintes équationnelles. Nous montrons qu'il existe des ensembles de c-clauses dans $\mathfrak{C}_{\text{eq-model}}$ pour lesquels RAMC ne permet pas de construire des modèles.

EQMC

Nous proposons une méthode permettant de surmonter ces limitations. Elle est fondée sur une combinaison de techniques d'énumération des eq-interprétations avec l'utilisation de méthodes

5. Voir (CAFERRA ET ZABEL, 1993) pour une extension (par traduction) à certaines logiques modales.

déductives. Nous montrons que cette méthode permet de construire des modèles pour toute formule de la classe $\mathfrak{C}_{\text{eq-model}}$.

RAMCET

Nous nous intéressons ensuite à la méthode RAMCET[6]. Nous proposons une extension de la méthode des tableaux sémantiques fondée sur l'utilisation d'une règle de simplification (qui peut être vue comme une généralisation des règles de subsumption et résolution à des formules du premier ordre) et de stratégies sémantiques similaires à celles utilisées pour la méthode RAMC. Nous définissons une nouvelle méthode permettant d'extraire un modèle d'une branche potentiellement infinie. L'un de ses avantages est qu'elle peut être combinée facilement avec l'utilisation des formalismes étudiés dans cette thèse (tels que les I-termes, grammaires d'arbres...). Notre méthode est fondée sur l'utilisation de règles de généralisation inductive.

Nous montrons que notre approche est strictement plus puissante que la méthode proposée par (CAFERRA ET ZABEL, 1993). En particulier, elle permet de réduire considérablement l'espace de recherche et d'augmenter strictement la classe de modèles constructibles. Les démonstrateurs de théorème par génération de modèles tels que SATCHMO, MGTP ou les HYPER-TABLEAUX, etc. (voir section 1.3) peuvent être considérés comme un cas particulier de notre approche.

Nous montrons que RAMCET est une procédure de décision pour la classe $\mathfrak{C}_{\text{eq-model}}$.

Construction de modèles et procédure de décision

Les rapports entre la procédure RAMC et la méthode de résolution sémantique (utilisée comme procédure de décision pour certaines classes de formules) sont étudiés. Nous montrons en particulier que les procédures obtenues peuvent être utilisées comme procédures *uniformes* de décision pour une large classe de formules logiques (incluant par exemple toutes les classes décidables par hyper-résolution, sans subsomption, les classes OCC1N, PVD, la classe monadique, etc.), en prouvant que toutes ces classes sont inclues dans $\mathfrak{C}_{\text{eq-model}}$. Nous montrons en outre que la classe $\mathfrak{C}_{\text{eq-model}}$ *n'est pas syntaxiquement caractérisable*, c'est-à-dire qu'il n'existe pas d'algorithme décidant de l'appartenance à la classe $\mathfrak{C}_{\text{eq-model}}$.

Nouveaux formalismes de représentation

Il apparaît rapidement que le formalisme initialement utilisé, fondé sur l'utilisation de contraintes équationnelles interprétées dans l'algèbre des termes, s'avère très insuffisant pour de nombreuses applications. L'exemple le plus simple est sans doute la formule (déjà citée dans (CAFERRA ET ZABEL, 1992)) : $\forall x.P(x) \; - \; \neg P(f(x))$ qui est satisfaisable, mais n'admet pas de modèle de Herbrand exprimable par des contraintes équationnelles. Afin d'étendre la classe de modèles constructibles, il est nécessaire d'utiliser de nouveaux formalismes pour exprimer et construire les modèles. Nous avons donc cherché à utiliser des formalismes plus expressifs, permettant d'exprimer une plus large classe de modèles. Pour cela, il suffit de changer la théorie dans laquelle les contraintes sont interprétées. Il est clair qu'aucune approche générale n'est possible[7]. Nous avons donc étudié plusieurs démarches différentes pour résoudre ce problème.

Une première idée consiste à construire des modèles dans une théorie non vide E, construite au fur et à mesure de la recherche du modèle (CAFERRA ET PELTIER, 1995b). Il s'agit d'utiliser les informations déduites pendant la résolution des contraintes pour générer des théories équationnelles dans lesquelles le modèle peut être construit. L'inconvénient de cette approche est qu'elle reste presque entièrement interactive. D'autre part, la résolution des contraintes dans une algèbre quelconque est indécidable. Nous n'avons pas retenu cette solution (décrite plus en détail dans (BOURELY, 1998)).

6. **R**efutation **A**nd **M**odel **C**onstruction with **E**xtended **T**ableaux

7. L'ensemble des interprétations de Herbrand n'est pas (en général) énumérable, donc il n'existe aucun formalisme capable de représenter toutes les interprétations de Herbrand

Nous nous sommes intéressés dans un deuxième temps à l'utilisation de *schématisations* qui permettent d'exprimer des séquences infinies de termes (comme par exemple l'ensemble $\{a, f(a), f(f(a)), \ldots, f^n(a), \ldots\}$). Ces formalismes ont été développés essentiellement pour éviter la divergence des méthodes de démonstration automatique ou de réécriture. Nous proposons une méthode de résolution des contraintes du premier ordre dans un formalisme de schématisations particulier, appelé "termes avec exposants entiers" (I-termes), initialement proposé par H. COMON (il s'agit d'une extension des *termes récurrents* de (CHEN ET AL., 1990)). Ce résultat est nécessaire pour l'intégration des termes avec exposants entiers dans la méthode RAMC. Il étend de façon importante les résultats existants (seule la décidabilité du problème d'unification – qui est un *cas particulier* de formule équationnelle – était connue jusqu'alors) et a de nombreuses applications en dehors de la construction de modèles. Nous avons ensuite montré comment utiliser ce formalisme pour représenter et construire des modèles de Herbrand dans le cas où la méthode RAMC échoue.

Nous étudions enfin un autre type de formalisme, fondé sur l'utilisation d'*automates d'arbres*. Nous montrons l'intérêt et les limites de ce formalisme en le comparant avec les I-termes.

Analogie en recherche simultanée de réfutation et de modèles

Une des raisons principales de l'inefficacité des démonstrateurs est leur manque de mémoire. Ils ne sont pas capables de reconnaître un problème déjà résolu, ni d'utiliser les informations pouvant être déduites de la résolution d'un problème pour guider la démonstration d'un nouveau problème similaire. Nous avons donc proposé une méthode permettant d'utiliser le raisonnement par analogie pour rechercher *simultanément* une réfutation ou un modèle d'une formule. L'idée principale de l'approche proposée consiste à utiliser les informations déduites de la réfutation ou du processus de construction de modèles d'une formule afin de transformer la formule en une formule "plus générale" (en un sens à préciser). Lors de la présentation d'une nouvelle formule, le démonstrateur recherche alors parmi les formules déjà traitées, s'il en existe une dont le nouveau problème soit une instance. Dans ce cas, il est possible d'en reconstruire la réfutation ou le modèle de façon immédiate. Lorsque l'analogie échoue, notre approche permet d'obtenir des informations intéressantes sur la nouvelle formule (par exemple un modèle partiel, ou des lemmes à démontrer pour compléter la preuve). A notre connaissance, aucune autre méthode de recherche systématique de construction de contre-exemples par analogie n'a été publiée[8].

Construction de modèles et programmation logique

Nous nous intéressons à l'application de RAMC à la programmation (essentiellement à la programmation logique). Nous montrons comment étendre les capacités des interpréteurs logiques existants et nous illustrons l'intérêt de la méthode pour la vérification interactive ou automatique de programmes et pour la détection et la correction interactive d'erreurs dans un programme donné.

Implémentation

Enfin, nous terminons par une brève description du système $\text{RAMC}_{\text{ATINF}}$ dont l'objet est d'expérimenter certaines des idées introduites et de fournir un logiciel interactif de recherche simultanée de réfutation et de modèles. Des expérimentations confirment l'intérêt pratique de certaines des idées introduites dans ce mémoire. En particulier, $\text{RAMC}_{\text{ATINF}}$ est utilisé pour construire un modèle d'une formule donnée par (GOLDFARB, 1984), qui est satisfaisable, mais qui ne posséde pas de modèle fini, pour laquelle aucune autre solution n'est connue.

Nous concluons en mentionnant quelques voies de recherche à explorer dans le futur.

8. Ce type de résultats est une conséquence indirecte de la philosophie du projet ATINF, conduisant à la recherche d'améliorations *qualitatives* des démonstrateurs automatiques.

Les chapitres 3, 4 (construction de modèles finis), 5 (pour la caractérisation des propriétés des eq-interprétations), 7, 8, 9, 10, (extension des méthodes RAMC et RAMCET, définition de la méthode EQMC), 11, 12 (nouveaux formalismes de représentation des interprétations), 13 (analogie en construction de modèles[9]) et 14 (application en programmation logique) ainsi que la partie V (implémentation et expérimentations) constituent l'essentiel de la contribution originale de nos travaux, le reste étant des rappels de l'existant, qui s'avèrent indispensables à la compréhension du présent mémoire. Notons que la plupart des résultats décrits dans ce mémoire ont été également publiés, ou sont sur le point d'être publiés, dans des revues ou actes de conférences en langue anglaise (sous une forme parfois différente) : construction de modèles finis (PELTIER, 1997b), extension de RAMC (BOURELY ET AL., 1994; CAFERRA ET PELTIER, 1995b; CAFERRA ET PELTIER, 1995a; CAFERRA ET PELTIER, 1996a), application en Programmation Logique (CAFERRA ET PELTIER, 1996b; CAFERRA ET PELTIER, 1997b), analogie (BOURELY ET AL., 1996; BOURELY ET AL., 1997; DÉFOURNEAUX ET PELTIER, 1997b; DÉFOURNEAUX ET PELTIER, 1997a), I-termes (PELTIER, 1997a), construction de modèles et automates d'arbres (PELTIER, 1997d), méthode EQMC (CAFERRA ET PELTIER, 1997a), méthode des tableaux (PELTIER, 1997c).

9. Les résultats décrits dans ce chapitre ont été obtenus en collaboration avec Christophe BOURELY et Gilles DÉFOURNEAUX.

Chapitre 2

Préliminaires

"Il[1] pose des définitions exactes qui le privent de l'agréable liberté d'abuser des termes dans les occasions"

Bernard le Bovier de Fontenelle.

L'objet de ce chapitre est de rappeler les définitions et notations nécessaires à la compréhension de ce travail (syntaxe et sémantique de la logique du premier ordre, forme clausale, etc.). On pourra aussi consulter, par exemple, (Gallier, 1986; Caferra, 1986; Lalement, 1990).

2.1 Relations et ordres

Rappelons ci-dessous quelques définitions importantes. Une relation binaire R est dite :

- *réflexive* si et seulement si $\forall x.(x,x) \in R$.
- *symétrique* si et seulement si $\forall x,y.(x,y) \in R - (y,x) \in R$.
- *antisymétrique* si et seulement si $\forall x,y.x \neq y \Rightarrow (x,y) \notin R \vee (y,x) \notin R$.
- *transitive* si et seulement si $\forall x,y,z.((x,y) \in R \wedge (y,z) \in R) \Rightarrow (x,z) \in R$.

Une relation binaire R est une *relation d'équivalence* si et seulement si R est transitive, réflexive et symétrique. La *fermeture transitive* R^* d'une relation R est définie comme la plus petite relation réflexive et transitive contenant R. Un *pré-ordre* est une relation reflexive et transitive. Un ordre est un pré-ordre antisymétrique. Un ordre strict $<$ est une relation transitive, anti-symétrique vérifiant $\forall x.x \not< x$.

Un ordre $<$ strict est dit *bien-fondé* s'il n'existe pas de suite infinie $(u_i)_{i\in\mathbb{N}}$ vérifiant $\forall i \in \mathbb{N}.u_i > u_{i+1}$. Un ordre $\leq$ est dit *total* si pour tout x,y $x \leq y$ ou $y \leq x$.

Dans la suite, un n-uplet d'éléments de E sera souvent noté $\overline{x}$ à la place de $(x_1,\ldots,x_n)$. Par abus de notation, nous noterons également $\overline{x}$ pour dénoter un sous-ensemble de E.

Extension lexicographique

Soient $D_1,\ldots,D_n$, n ensembles munis chacun d'un ordre $<_i$, l'extension lexicographique $<_{lex}$ des ordres $<_i$ est la relation d'ordre définie sur $D_1 \times \ldots \times D_n$ par

$$(a_1,\ldots,a_n) <_{lex} (b_1,\ldots,b_n) - \exists i \in [1..n]/\forall j < i.a_j = b_j \ et \ a_i <_i b_i.$$

1. Leibniz

Multi-ensemble

Soit D un ensemble muni d'une relation d'ordre $<$. Un *multi-ensemble* d'éléments de D est une fonction de D vers $\mathbb{N}$. On note: $x \in X$ pour $X(x) > 0$.

On définit les opérateurs suivants sur les multi-ensembles.

- $M_1 \cup M_2(x) = max(M_1(x), M_2(x))$
- $M_1 \cap M_2(x) = min(M_1(x), M_2(x))$
- $M_1 + M_2(x) = M_1(x) + M_2(x)$
- $M_1 - M_2(x) = max(M_1(x) - M_2(x), 0)$

On définit l'extension multi-ensemble de l'ordre $\leq$ par

$$X \leq_m Y$$

si et seulement si l'une des conditions suivantes est satisfaite.

- $X = \emptyset$.
- $\exists x \in Y \cap X$ tel que $X - \{x\} \leq_m Y - \{x\}$.
- $\exists x \in Y$ tel que $X - \{y / y < x\} \leq_m Y - \{x\}$.

THÉORÈME 2.1.1. – *Si les* $<_i$ $(1 \leq i \leq n)$ *sont totaux alors* $<_{lex}$ *est total.*

- *Si les* $<_i$ $(1 \leq i \leq n)$ *sont bien fondés alors* $<_{lex}$ *est bien fondé.*
- *Si* $<$ *est total alors* $<_m$ *est total.*
- *Si* $<$ *est bien fondé alors* $<_m$ *est bien fondé.*

2.2 Logique des prédicats du premier ordre

Nous définissons ici la logique des prédicats du premier ordre, en précisant tout d'abord la *syntaxe* puis la *sémantique* du langage. Par souci de généralité, nous introduisons la présence de sortes (sans relation d'inclusion entre sortes).

2.2.1 Syntaxe

Un alphabet du premier ordre comprend plusieurs ensembles supposés disjoints : un ensemble de connectifs logiques $\{\vee, \wedge, \Rightarrow, -, \neg, \forall, \exists\}$, un ensemble de symboles de sorte $\mathcal{S}$, une famille d'ensembles infinis dénombrables $(\mathcal{V}_s)_{s \in \mathcal{S}}$ de symboles de variables, un ensemble Σ de symboles de fonction, un ensemble Ω de symboles de prédicat et *profil* une fonction associant à tout élément f de $\Sigma \cup \Omega$ une séquence (non vide si $f \in \Sigma$) finies de symboles de sorte. *profil*(f) est appelé le *profil* de f. Si $f \in \Sigma$, *profil*$(f) = s_1, \ldots, s_n, s$ est noté : $f : s_1, \ldots, s_n \rightarrow s$. n sera appelé l'*arité* de f et noté $a(f)$. De même si $P \in \Omega$ et si *profil*$(P) = s_1, \ldots, s_n$, alors on note $P : s_1, \ldots, s_n$, n est l'arité de P et $a(P) = n$. On notera parfois $f^{(i)}$ pour désigner un symbole d'arité i. Pour tout entier n, nous noterons Σ_n (respectivement Ω_n) l'ensemble des éléments f de Σ (respectivement Ω) tels que $a(f) = n$. L'ensemble des *variables* est $\mathcal{V} = \bigcup_{s \in \mathcal{S}} \mathcal{V}_s$.

Termes

DÉFINITION 2.2.1. *L'ensemble des termes $\tau_s(\Sigma, X)$ de sorte s formés sur la signature Σ et sur l'ensemble de variables $X \subseteq \mathcal{V}$ est le plus petit ensemble de séquences de symboles vérifiant les conditions suivantes.*

Si $x \in X$ et $x \in \mathcal{V}_s$ alors $x \in \tau_s(\Sigma, X)$

Si $f : s_1, \ldots, s_n \to s$ et $\forall i \leq n . t_i \in \tau_{s_i}(\Sigma, X)$ alors $f(t_1, \ldots, t_n) \in \tau_s(\Sigma, X)$

◇

Pour tout terme t, il existe une unique sorte $s = sort(t)$ telle que $t \in \tau_s(\Sigma, \mathcal{V})$. Nous supposons en outre qu'aucun élément $\tau_s(\Sigma)$ est vide. L'ensemble des termes $\tau(\Sigma, X)$ est :

$$\tau(\Sigma, X) = \bigcup_{s \in \mathcal{S}} \tau_s(\Sigma, X)$$

Si $X = \emptyset$, alors $\tau_s(\Sigma, X)$ (resp. $\tau(\Sigma, X)$) sera simplement noté $\tau_s(\Sigma)$ (resp. $\tau(\Sigma)$) et appelé *univers de Herbrand*. Les termes dans $\tau(\Sigma)$ sont appelés *termes fermés* .

La notion de position permet de repérer les sous-termes apparaissant à l'intérieur d'un terme.

DÉFINITION 2.2.2. *Une position p est une chaîne (éventuellement vide) d'entiers. La chaîne vide sera notée Λ. Le signe "." dénote l'opérateur de concaténation entre chaînes. L'ensemble des positions d'un terme t est inductivement défini de la façon suivante.*

$\mathcal{P}os(t) = \Lambda$ *(si* $t \in \mathcal{V}$*)*

$\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\Lambda\} \cup \bigcup_{i=1}^{n} i.\mathcal{P}os(t_i)$

Si $t \in \tau(\Sigma, X)$ et si $p \in \mathcal{P}os(t)$, nous définirons le terme $t_{|p}$ de la façon suivante.

Si $p = \Lambda$, alors $t_{|p} = t$.

Si $p = i.q$ et $t = f(t_1, \ldots, t_n)$, $t_{|p} = t_{i|q}$.

Un terme t est un sous-terme *(respectivement un sous-terme propre) de s si et seulement si il existe une position p (resp. une position $p \neq \Lambda$) dans $\mathcal{P}os(s)$ telle que : $s_{|p} \equiv t$. Cela est noté : $t \preceq s$ (resp. $t \prec s$).* ◇

L'ensemble (parfois le n-uplet, avec un ordre quelconque) des variables apparaissant dans un terme t sera noté $\mathcal{V}ar(t)$. Un terme t est dit *linéaire* si toute variable de t apparaît une seule fois dans t (c'est-à-dire si pour toute variable x il existe au plus une position p telle que $t_{|p} = x$).

Formules

DÉFINITION 2.2.3.

L'ensemble des formules bien formées est le plus petit ensemble de séquences de symboles $\mathcal{F}bf$ vérifiant les conditions suivantes.

$\top \in \mathcal{F}bf$

$\bot \in \mathcal{F}bf$

Si $P : s_1, \ldots, s_n \in \Omega_n$ et $\forall i \leq n . t_i \in \tau_{s_i}(\Sigma, X)$ alors $P(t_1, \ldots, t_n) \in \mathcal{F}bf$

Si $f \in \mathcal{F}bf$ alors $\neg f \in \mathcal{F}bf$

Si f_1 et f_2 appartiennent à $\mathcal{F}bf$

alors $f_1 \vee f_2, f_1 \wedge f_2, f_1 \Rightarrow f_2$ et $f_1 - f_2$ appartiennent à $\mathcal{F}bf$.

Si $f \in \mathcal{F}bf$ et $x \in \mathcal{V}$, $\exists x . f \in \mathcal{F}bf$ et $\forall x . f \in \mathcal{F}bf$.

◇

Les formules de la forme $P(t_1, \ldots, t_n)$ sont dites *atomiques*. Nous supposerons en outre que Ω contient pour chaque sorte s un symbole d'égalité "$=_s : s, s$" (noté plus simplement $=$ et en notation infixé). La formule $\neg(t_1 = t_2)$ sera notée $t_1 \neq t_2$.

Les notions de *position* et de *sous-formule* se définissent exactement de la même manière que pour les termes.

DÉFINITION 2.2.4. *L'ensemble des positions d'une formule $\mathcal{F}$ est inductivement défini de la façon suivante.*

$\mathcal{P}os(P(\overline{s})) = \Lambda$.

$\mathcal{P}os(\exists x.F) = \mathcal{P}os(\forall x.\mathcal{F}) = \mathcal{P}os(\neg\mathcal{F}) = \{\Lambda\} \cup 1.\mathcal{P}os(\mathcal{F})$

$\mathcal{P}os(\mathcal{F}_1 \star \mathcal{F}_2) = \{\Lambda\} \cup 1.\mathcal{P}os(\mathcal{F}_1) \cup 2.\mathcal{P}os(\mathcal{F}_2)$ $(\star \in \{\vee, \wedge, \Rightarrow, -\})$.

Pour $\mathcal{F} \in \mathcal{F}bf$ et $p \in \mathcal{P}os(\mathcal{F})$, nous définirons la formule $\mathcal{F}_{|p}$ de la façon suivante.

Si $p = \Lambda$, alors $\mathcal{F}_{|p} = \mathcal{F}$.

Si $p = 1.q$ et $\mathcal{F} = \neg\mathcal{F}_1$ ou $\mathcal{F} = \exists x.\mathcal{F}_1$ ou $\mathcal{F} = \forall x.\mathcal{F}_1$, $\mathcal{F}_{|p} = \mathcal{F}_{1|q}$.

Si $p = i.q$ $(i \in \{1,2\})$ et $\mathcal{F} = \mathcal{F}_1 \star \mathcal{F}_2$, $\mathcal{F}_{|p} = \mathcal{F}_{i|q}$.

$\mathcal{F}$ est dite sous-formule (resp. propre) de $\mathcal{G}$, noté $\mathcal{F} \preceq \mathcal{G}$ (resp. $\mathcal{F} \prec \mathcal{G}$) si et seulement si il existe une position p (resp. une position non vide) telle que : $\mathcal{G}_{|p} = \mathcal{F}$. Un terme t apparaît *dans une formule $\mathcal{F}$ (noté $t \preceq \mathcal{F}$) si et seulement s'il existe une sous-formule $P(t_1, \ldots, t_n)$ de $\mathcal{F}$ et un entier $i \in [1..n]$ tel que : $t \prec t_i$.* ◇

Les formules de la forme $P(t_1, \ldots, t_n)$, $\neg P(t_1, \ldots, t_n)$ sont appelées des *littéraux*. Un littéral est dit *positif* si il est de la forme : $P(t_1, \ldots, t_n)$, négatif sinon.

DÉFINITION 2.2.5. *Une variable x apparaissant dans une formule est dite* liée *si elle n'apparaît que dans une sous-formule de la forme $\forall x.\mathcal{F}$ ou $\exists x.\mathcal{F}$. L'ensemble des variables liées apparaissant dans une formule $\mathcal{F}$ est noté $\mathcal{V}ar(\mathcal{F})$ et $\mathcal{V}ar_B(\mathcal{F})$.*

L'ensemble $\mathcal{V}ar(\mathcal{F})$ des variables libres d'une formule est inductivement défini de la façon suivante :

- $\mathcal{V}ar(P(\overline{t})) = \mathcal{V}ar(\overline{t})$.
- $\mathcal{V}ar(\mathcal{F} \wedge \mathcal{G}) = \mathcal{V}ar(\mathcal{F} \vee \mathcal{G}) = \mathcal{V}ar(\mathcal{F}) \cup \mathcal{V}ar(\mathcal{G})$.
- $\mathcal{V}ar(\neg\mathcal{F}) = \mathcal{V}ar(\mathcal{F})$.
- $\mathcal{V}ar(\exists x.\mathcal{F}) = \mathcal{V}ar(\forall x.\mathcal{F}) = \mathcal{V}ar(\mathcal{F}) \setminus \{x\}$.

Une formule ne contenant pas de variables libres est dite fermée. ◇

Dans la suite les formules seront supposées *rectifiées*, c'est-à-dire que deux occurrences distinctes de quantificateurs lient deux variables distinctes et que $\mathcal{V}ar(\mathcal{F}) \cap \mathcal{V}ar_B(\mathcal{F}) = \emptyset$.

Substitutions

DÉFINITION 2.2.6. *Une* substitution *est une application σ de $\mathcal{V}$ dans $\tau(\Sigma, \mathcal{V})$ à support fini et telle que pour toute variable x : $sort(x) = sort(\sigma(x))$.* ◇

Une substitution σ est dite fermée si et seulement si pour toute variable x telle que $\sigma(x) \neq x$, $\sigma(x)$ est fermé. Une substitution peut être étendue en un endomorphisme de $\tau(\Sigma, \mathcal{V})$ par la relation

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)).$$

De même, une substitution peut être étendue en une fonction de $\mathcal{F}bf$ dans $\mathcal{F}bf$. L'image d'un terme (resp. d'une formule) E par une substitution σ est notée $E\sigma$ ou $\sigma(E)$.

La substitution associant à chaque variable x_i (pour $i \in [1..n]$) un terme t_i sera notée $\{x_i \rightarrow t_i / i \in [1..n]\}$. Si σ et θ sont deux substitutions à support disjoint, $\sigma \cup \theta$ dénote la composition $\sigma\theta$.

2.2.2 Sémantique

Pour donner une sémantique aux formules de la logique des prédicats, on définit la notion d'interprétation, dont l'objet est de donner une signification aux symboles de fonction et de prédicat de la signature.

DÉFINITION 2.2.7. *Une* structure d'interprétation *est un couple* $((\mathcal{D}_s)_{s\in\mathcal{S}}, \mathcal{I})$ *où* $\mathcal{D}_s$ *est un ensemble non vide d'objets (le* domaine *de la sorte* s*),* $\mathcal{I}$ *est une fonction associant à tout élément* P *de* Ω *de profil :* $s_1, \ldots, s_n$ *une relation* $\mathcal{P}_\mathcal{I}$ *sur* $\mathcal{D}_{s_1} \times \ldots \times \mathcal{D}_{s_n}$ *et à tout symbole fonctionnel* f *de profil* $s_1, \ldots, s_n \to s$ *de* Σ *une fonction de* $\mathcal{D}_{s_1} \times \ldots \times \mathcal{D}_{s_n}$ *dans* $\mathcal{D}_s$*.* ◇

DÉFINITION 2.2.8. *Une interprétation* $((\mathcal{D})_{s\in\mathcal{S}}, \mathcal{I})$ *est dite* normale *si et seulement si l'interprétation de* $=$ *est l'égalité, c'est-à-dire si et seulement si,* $\forall s.\forall x, y \in \mathcal{D}_s, (x, y) \in \mathcal{I}(=_s) - x = y$*. Dans la suite, nous supposerons que toutes les interprétations considérées sont normales.* ◇

La fonction $\mathcal{I}$ peut être étendue en une fonction de $\tau_s(\Sigma)$ dans $\mathcal{D}_s$ par la relation suivante :

$$\mathcal{I}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$$

DÉFINITION 2.2.9.

Une assignation $\mathcal{A}$ *d'une formule* $\mathcal{F}$ *est un couple* $(((\mathcal{D}_s)_{s\in\mathcal{S}}), \mathcal{I}), \sigma)$ *où* $((\mathcal{D}_s)_{s\in\mathcal{S}}, \mathcal{I})$ *est une interprétation et* σ *une fonction de* $\mathcal{V}ar(\mathcal{F})$ *dans* $\bigcup_{s\in\mathcal{S}} \mathcal{D}_s$ *telle que* $\forall x \in \mathcal{V}_s \cap \mathcal{V}ar(\mathcal{F}).\sigma(x) \in \mathcal{D}_s$*. Là encore,* $\mathcal{A}$ *peut être étendue en une fonction de* $\tau_s(X, \Sigma)$ *dans* $\mathcal{D}_s$ *de la façon suivante.*

$$\mathcal{A}(x) = \sigma(x) \text{ Si } x \in \mathcal{V}.$$

$$\mathcal{A}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n))$$

◇

On définit alors la notion de validité d'une formule logique de la façon habituelle.

DÉFINITION 2.2.10. *Une assignation* $A : (((\mathcal{D}_s)_{s\in\mathcal{S}}, \mathcal{I}), \sigma)$ *de* $\mathcal{F}$ valide $\mathcal{F}$ *(noté* $\mathcal{A} \models \mathcal{M}$*) si et seulement si l'une des conditions suivantes est vérifiée.*

$\mathcal{F} = P(t_1, \ldots, t_n)$ *et* $(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_n)) \in \mathcal{I}(P)$.

$\mathcal{F} = \neg\mathcal{F}'$ *et* $\mathcal{A} \not\models \mathcal{F}'$.

$\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$ *et* $\mathcal{A}$ *valide* $\mathcal{F}_1$ *ou* $\mathcal{F}_2$.

$\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$ *et* $\mathcal{A}$ *valide* $\mathcal{F}_1$ *et* $\mathcal{F}_2$.

$\mathcal{F} = \mathcal{F}_1 \Rightarrow \mathcal{F}_2$ *et* $\mathcal{A}$ *valide* $\neg\mathcal{F}_1$ *ou* $\mathcal{F}_2$.

$\mathcal{F} = \mathcal{F}_1 - \mathcal{F}_2$ *et* $\mathcal{A}$ *valide* $\mathcal{F}_1 \Rightarrow \mathcal{F}_2$ *et* $\mathcal{F}_2 \Rightarrow \mathcal{F}_1$.

$\mathcal{F} = \forall x.\mathcal{F}'$, $x \in \mathcal{V}_s$ *et pour tout* a *dans* $\mathcal{D}_s$, $(\mathcal{I}, \sigma \cup \{x \to a\}) \models \mathcal{F}'$.

$\mathcal{F} = \exists x.\mathcal{F}'$, $x \in \mathcal{V}_s$ *et il existe* a *dans* $\mathcal{D}_s$ *tel que* $(\mathcal{I}, \sigma \cup \{x \to a\}) \models \mathcal{F}'$.

Une interprétation $\mathcal{I}$ valide *une formule logique* $\mathcal{F}$ *(noté* $\mathcal{I} \models \mathcal{F}$*) si et seulement si pour toute assignation* $(\mathcal{I}, \sigma)$, $(\mathcal{I}, \sigma) \models \mathcal{F}$. ◇

DÉFINITION 2.2.11. *Une formule* $\mathcal{F}$ *est dite* valide *si et seulement si elle est validée par toute interprétation* $\mathcal{I}$*. Une formule* $\mathcal{F}$ *est dite* insatisfaisable *si et seulement si elle n'est validée par aucune interprétation.* ◇

DÉFINITION 2.2.12. *Soit* $\mathcal{I}$ *une interprétation et* $\mathcal{F}$ *une formule. Si* $\mathcal{I} \models \mathcal{F}$ *alors* $\mathcal{I}$ *est un* modèle *de* $\mathcal{F}$*. Si* $\mathcal{I} \not\models \mathcal{F}$ *alors* $\mathcal{I}$ *est un* contre-exemple *(ou* contre-modèle*) de* $\mathcal{F}$. ◇

REMARQUE. $\mathcal{F}$ est valide si et seulement si $\neg\mathcal{F}$ est insatisfaisable. De même, un modèle de $\mathcal{F}$ est un contre-exemple de $\neg\mathcal{F}$ et réciproquement. Ces notions de validité/insatisfaisabilité et contre-exemple/modèle sont donc duales.

DÉFINITION 2.2.13. *Soit deux formules* $\mathcal{F}_1$ *et* $\mathcal{F}_2$*. Si* $\mathcal{I} \models (\mathcal{F}_1 - \mathcal{F}_2)$*, on note* $\mathcal{F}_1 \equiv_\mathcal{I} \mathcal{F}_2$*.* $\mathcal{F}_1$ *et* $\mathcal{F}_2$ *sont dites* équivalentes *(noté* $\mathcal{F}_1 \equiv \mathcal{F}_2$*) si et seulement si pour toute interprétation* $\mathcal{I}$*,* $\mathcal{F}_1 \equiv_\mathcal{I} \mathcal{F}_2$. ◇

Solutions d'une formule

DÉFINITION 2.2.14. *Une substitution fermée σ est appelée une* solution *d'une formule $\mathcal{F}$ dans une interprétation $\mathcal{I}$ si et seulement si*

$$\mathcal{I} \models \mathcal{F}\sigma.$$

L'ensemble des solutions d'une formule $\mathcal{F}$ dans une interprétation $\mathcal{I}$ sera noté $\mathcal{S}_{\mathcal{I}}(\mathcal{F})$. ◇

Semi-décidabilité de la logique du premier ordre

Rappelons brièvement quelques résultats sur la semi-décidabilité de la logique du premier ordre. Le problème de savoir si une formule $\mathcal{F}$ est satisfaisable (resp. valide) est *indécidable*, c'est-à-dire qu'il n'existe aucun algorithme qui se termine toujours et qui retourne 1 si $\mathcal{F}$ est insatisfaisable (resp. valide) 0 sinon. Ce problème est néanmoins *semi-décidable*, c'est-à-dire qu'une telle procédure existe mais ne termine pas nécessairement si la formule est satisfaisable (resp. non valide). Ainsi, l'ensemble des formules insatisfaisables (resp. valide) est récursivement énumérable mais pas co-récursivement énumérable.

2.2.3 Formes normales d'une formule

Forme clausale

Plusieurs procédures de recherche de preuve (en particulier la méthode de *résolution*) se restreignent à une forme particulière de formule, appelée *forme clausale*.

DÉFINITION 2.2.15. *Une formule fermée est dite sous forme* clausale *si et seulement si elle est de la forme :*

$$\bigwedge_{i=1}^{n} \forall \overline{x}_i \bigvee_{j=1}^{k_i} L_{ij}$$

où les formules L_{ij} (pour $1 \leq i \leq n$ et $1 \leq j \leq k_i$) sont des littéraux.

Les formules fermées de la forme : $\forall \overline{x}. \bigvee_{j=1}^{k} .L_j$, où pour tout $j \in [1..k]$, L_j est un littéral, sont appelées des clauses.

◇

REMARQUE. Une clause est souvent représentée comme un multi-ensemble de littéraux. Puisque toutes les variables sont quantifiées universellement, on peut omettre les quantificateurs universels. Ainsi la clause $\forall x, y.(P(x) \vee Q(x,y))$ sera notée $P(x) \vee Q(x,y)$ ou encore $\{P(x), Q(x,y)\}$. De même, une formule sous forme clausale est souvent représentée comme un *ensemble (ou multi-ensemble) de clauses* plutôt que comme une conjonction.

Il est possible de transformer toute formule en un ensemble de clauses en préservant la satisfaisabilité de la formule initiale.

THÉORÈME 2.2.1. *Il existe un algorithme transformant toute formule $\mathcal{F}$ en un ensemble de clauses noté clause($\mathcal{F}$) tel que :*

$$\mathcal{F} \text{ est satisfaisable} - \text{clause}(\mathcal{F}) \text{ est satisfaisable}$$

D'autre part, tout modèle de *clause*($\mathcal{F}$) est un modèle de $\mathcal{F}$[2].

2. Cette propriété est évidemment cruciale pour l'utilité de cette transformation dans le contexte de la construction de modèles !

Nous ne rappellerons pas ici le principe de cette transformation, qui utilise des règles de transformation fondées sur les propriétés usuelles des connectifs logiques $\vee, \wedge, \Rightarrow \ldots$ (distributivité...) ainsi qu'une opération de *skolémisation* permettant de remplacer les quantificateurs existentiels par de nouveaux symboles de fonction. Plusieurs algorithmes de transformation existent. L'algorithme "naïf", le plus simple, peut induire une augmentation exponentielle de la taille de la formule. En utilisant des techniques de *renommage* (Plaisted et Greenbaum, 1986; Boy de la Tour, 1992), il est possible d'obtenir une transformation *polynômiale en taille* (quadratique). Une étude détaillée des algorithmes utilisant le renommage et un algorithme optimal[3] (pour des formules *linéaires*, c'est-à-dire ne contenant pas le symbole "$-$") de transformation sous forme clausale peuvent être trouvés dans (Boy de la Tour, 1992).

Autres formes normales

Définition 2.2.16.

- *Une formule $\mathcal{F}$ est dite en* forme normale négative *(fnn) si et seulement si pour toute sous-formule de $\mathcal{F}$ de la forme $\neg\mathcal{G}$, $\mathcal{G}$ est atomique.*
- *Une formule est dite en* forme normale prénexe *si elle est de la forme $Q_1x_1 \cdots Q_nx_n.\mathcal{M}$, où $\forall i \leq n.Q_i \in \{\exists, \forall\}$ et où $\mathcal{M}$ ne contient aucun quantificateur. $\mathcal{M}$ est appelée la* matrice *de $\mathcal{F}$.*

- *Une formule $\mathcal{F}$ en forme prénexe est dite en* forme normale disjonctive *(fnd) si et seulement si sa matrice est de la forme $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} \mathcal{F}_{ij}$, où $\mathcal{F}_{ij}$ est un littéral.*
- *Une formule $\mathcal{F}$ en forme prénexe est dite en* forme normale conjonctive *(fnc) si et seulement si sa matrice est de la forme $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \mathcal{F}_{ij}$, où $\mathcal{F}_{ij}$ est un littéral.*

◇

2.2.4 Modèles de Herbrand

Définition 2.2.17. *Une interprétation (resp. modèle) $((\mathcal{D}_s)_{s\in\mathcal{S}}, \mathcal{I})$ est appelée* interprétation (resp. *modèle*) de Herbrand *si et seulement si $\forall s \in \mathcal{S}.\mathcal{D}_s = \tau_s(\Sigma)$ et si pour tout $f \in \Sigma$, $\mathcal{I}(f)(t_1, \ldots, t_n) = f(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$.* ◇

Théorème 2.2.2. *Soit un ensemble de clauses S sans égalité. Alors, soit S est insatisfaisable, soit S admet un modèle de Herbrand.*

Définition 2.2.18. *L'ensemble : $\{P(t_1, \ldots, t_n)/P \ : s_1, \ldots, s_n \in \Omega, \forall i \leq n.t_i \in \tau_{s_i}(\Sigma)\}$ est appelé la* base de Herbrand *et noté $\mathcal{BH}$.* ◇

2.3 Problèmes équationnels

Quelques résultats importants de (Comon et Lescanne, 1989) concernant les formules équationnelles (définition et résolution dans la théorie vide) sont rappelés dans cette section. Intuitivement, un *problème équationnel* est une formule dans laquelle n'apparaît que le prédicat "$=$".

Définition 2.3.1. *Une formule équationnelle est une formule du premier ordre dans laquelle toutes les sous-formules atomiques sont de la forme : $s = t$, où s et t sont deux termes.* ◇

Définition 2.3.2. *Une* solution *d'une formule équationnelle dans la théorie vide est une solution de la formule dans l'interprétation de Herbrand (unique ici puisque la formule ne contient pas de symbole de prédicat). L'ensemble des solutions d'une formule $\mathcal{F}$ est noté $\mathcal{S}(\mathcal{F})$.* ◇

Les formules équationnelles ont fait l'objet d'études théoriques poussées (Mal'cev, 1971; Comon et Lescanne, 1989). Celles-ci ont permis de montrer que le problème de savoir si

3. au sens : "minimisant le nombre de clauses obtenues par traduction".

une formule équationnelle admet des solutions dans la théorie vide est décidable. Ce résultat a de nombreuses applications en réécriture, programmation, déduction automatique...et en particulier, il est de la plus haute importance pour l'approche mise en œuvre dans cette thèse. Il a été établi initialement dans (Mal'cev, 1971) puis redécouvert de façon indépendante dans (Kunen, 1987) (pour des signatures infinies) et (Maher, 1988; Comon, 1988; Comon et Lescanne, 1989) (signatures finies ou infinies). Dans (Comon et Lescanne, 1989) est proposé un algorithme permettant de transformer (en préservant l'ensemble des solutions) les formules équationnelles en un type particulier de formules appelées *formules résolues avec contraintes* dont les solutions peuvent être obtenues de façon immédiate. La définition 2.3.4 précise la forme générale des formes résolues avec contraintes.

Définition 2.3.3. *Une variable $x \in \mathcal{V}_s$ est dite* infinitaire *si $\tau_s(\Sigma)$ est infini.* ◇

Définition 2.3.4. *Une formule équationnelle $\mathcal{P}$ est dite* sous forme résolue avec contraintes *si et seulement si $\mathcal{P}$ est sous l'une des formes suivantes :*

- $\top$
- $\bot$
- $\exists \overline{w}.[\bigwedge_{j=1}^{m} x_j = s_j] \wedge [\bigwedge_{i=1}^{k} x'_i \neq s'_i]$, *où les x_j, x'_j sont des variables, chaque x_j apparaît une seule fois dans $\mathcal{P}$, tout x'_i n'apparaît pas dans s'_i et x'_i est* infinitaire.

◇

Théorème 2.3.1. *[voir (Comon, 1988; Comon et Lescanne, 1989)] Toute formule sous forme résolue avec contraintes différente de $\bot$ admet au moins une solution.*

Théorème 2.3.2. *[voir (Comon, 1988; Comon et Lescanne, 1989)] Il existe un algorithme transformant toute formule équationnelle $\mathcal{F}$ en une formule de la forme :*

$$\bigvee_{i=1}^{k} \mathcal{F}_i$$

telle que :

- $\mathcal{S}(\mathcal{F}) = \mathcal{S}(\bigvee_{i=1}^{k} \mathcal{F}_i)$.
- $\mathcal{F}_i$ $(1 \leq i \leq k)$ *est sous forme résolue avec contraintes.*

2.4 Présentation des algorithmes

Les algorithmes présentés dans cette thèse seront décrits soit en pseudo-pascal, soit par des systèmes à base de règles (Floyd, 1960). Chaque technique a ses avantages et ses inconvénients. Nous choisirons dans chaque cas la présentation qui nous paraît la mieux adaptée et la plus naturelle suivant la procédure considérée.

Pour tout système de règles $\mathcal{R}$ on écrit $q \rightarrow_{\mathcal{R}} u$ ou $u = \mathcal{R}(q)$ quand il existe une règle ρ de $\mathcal{R}$ transformant q en u.

Si $\mathcal{R}$ est a terminaison finie, on note $\mathcal{R}^*(u)$ une forme normale de u par rapport à $\mathcal{R}$.

Partie I

Une vision unifiée de la construction de modèles finis

Chapitre 3

Description de la méthode FMC

La théorie des modèles finis (traduction de l'anglais *finite model theory*) est d'une importance croissante en Logique et en informatique (voir par exemple (Fagin, 1974; Vardi, 1982; Gurevich, 1982; Immerman, 1983; Fagin, 1993)). En effet, elle fournit un cadre naturel pour l'étude des classes de complexité. L'importance donnée aux structures finies en informatique est souvent justifiée par le fait qu'elles peuvent être facilement représentées sur ordinateur.

Dans de nombreuses applications, on ne considère que des interprétations finies : c'est le cas par exemple en théorie des graphes, ou encore en programmation (en vérification de programmes, en synthèse de programmes concurrents à partir de spécifications en logique temporelle etc.). Il est par ailleurs intéressant de constater que la plupart des théorèmes importants en théorie des modèles ne sont plus valides en théorie des modèles finis. Un des exemples les plus frappants est le théorème affirmant la semi-décidabilité du problème de la validité d'une formule de la logique du premier ordre, qui n'est plus un théorème lorsque l'on se limite à des structures finies (ce qui signifie que l'ensemble des formules n'ayant pas de modèles finis n'est pas énumérable). Cette remarque illustre la spécificité de la théorie des modèles finis.

Dans cette partie, nous nous intéressons à la construction de modèles finis. Nous proposons une méthode générale pour construire un modèle fini d'une formule, fondée sur des techniques d'énumération efficaces des interprétations. La méthode proposée énumère l'ensemble des interprétations finies jusqu'à obtenir un modèle de la formule considérée. Naturellement, un tel algorithme s'avère impraticable dans les cas non-triviaux, à cause de la taille importante de l'espace de recherche. Nous définissons donc plusieurs stratégies à la fois naturelles et efficaces, permettant de réduire de façon importante le nombre d'interprétations à considérer. Ces stratégies sont fondées sur deux idées essentielles.

1. La première est d'utiliser les informations déduites de l'échec des tests déjà effectués pour réduire l'espace de recherche. Plus précisément, il s'agit d'identifier, pendant l'évaluation de la formule, les équations qui sont responsables de l'échec de l'interprétation testée et d'utiliser ces informations pour éliminer certaines interprétations.

2. La seconde est d'utiliser la symétrie naturelle existant (éventuellement) entre les éléments du domaine pour réduire le nombre d'interprétations à considérer.

Contrairement aux approches existantes (Slaney, 1992; Zhang, 1993; Zhang et Zhang, 1995) nous n'utilisons pas de techniques de propagation de contraintes. Ainsi, la méthode proposée peut traiter des formules logiques du premier ordre et ne nécessite aucune transformation préliminaire de la formule initiale. En particulier, elle ne nécessite pas de traduction préalable de la formule en un ensemble de clauses. Cette dernière caractéristique est particulièrement importante, car la mise sous forme clausale a pour effet, soit d'augmenter exponentiellement la taille de la formule, soit d'ajouter de nouveaux symboles de prédicats dans la signature, ce qui augmente l'espace de recherche. L'approche proposée est très modulaire et peut être étendue de façon très naturelle à d'autres logiques. En effet, elle est *indépendante* de la méthode utilisée pour évaluer la formule.

Le chapitre présente en détail les algorithmes de la méthode et établit certaines de leurs propriétés (correction et terminaison).

3.1 Définitions préliminaires

Les définitions et notations supplémentaires ci-dessous facilitent la présentation de la méthode.

Par souci de simplicité, nous ne considérons dans ce chapitre que des formules purement équationnelles, c'est-à-dire que nous supposons que $\Omega = \emptyset$ (voir chapitre 2). Cette condition n'est évidemment pas restrictive. En effet, si la formule contient un symbole de prédicat P de profil $s_1, \ldots, s_n$, nous pouvons la transformer en une formule équationnelle en introduisant un nouveau symbole de fonction f_P de profil $f_P : s_1, \ldots, s_n \rightarrow \mathit{Boolean}$ et en remplaçant chaque formule atomique $P(t_1, \ldots, t_n)$ par $f_P(t_1, \ldots, t_n) = \mathit{true}$[1].

Dans ce chapitre, nous supposerons donnée une fonction $\mathcal{S}ize$ qui associe à chaque symbole de sorte un entier strictement positif représentant la cardinalité du domaine de la sorte. Sans perte de généralité, les éléments du domaine de la sorte s seront notés $1, \ldots, \mathcal{S}ize(s)$. Nous notons $\mathcal{D}om(s) = \{1, \ldots, \mathcal{S}ize(s)\}$.

Nous conserverons autant que possible la terminologie de (Slaney, 1992; Zhang et Zhang, 1995), ce qui permettra de mieux situer notre travail par rapport à l'existant. Un terme[2] de la forme $f(i_1, \ldots, i_n)$ où $f : s_1, \ldots, s_n \rightarrow s$ est un symbole fonctionnel et $\forall j \leq n . i_j \in \mathcal{D}om(s_j)$ est appelé une *cellule* de sorte s (notée $\mathit{sort}(f(i_1, \ldots, i_n))$). Nous noterons $\mathcal{D}om(c)$ le domaine de la sorte de c et $\mathcal{S}ize(c)$ la cardinalité du domaine de la sorte de c. L'ensemble des cellules est noté $\mathcal{C}$ (il s'agit évidemment d'un ensemble fini), et l'ensemble des cellules de sorte s est noté $\mathcal{C}_s$.

Dans toute la suite, nous supposerons donné un ordre total strict $<$ sur $\mathcal{C}$. Si la cardinalité du domaine de chaque sorte est fixée, alors l'interprétation $f_{\mathcal{I}}$ d'un symbole fonctionnel $f : s_1, \ldots, s_n \rightarrow s$ est entièrement spécifiée en donnant la valeur de la fonction $f_{\mathcal{I}}$ sur chaque n-uplet $(i_1, \ldots i_n) \in \mathcal{D}om(s_1) \times \ldots \times \mathcal{D}om(s_n)$, i.e. en donnant la valeur de chaque cellule. Nous appellerons *interprétation partielle finie* une *fonction partielle* de $\mathcal{C}$ dans $\mathbb{N}$ (l'ensemble des entiers naturels) telle que pour tout $c, \mathcal{I}(c) \in \mathcal{D}om(c)$. $\mathit{cells}(\mathcal{I})$ dénote le domaine de la fonction $\mathcal{I}$. Une interprétation partielle sera habituellement notée $\{c = v / c \in \mathit{cells}(\mathcal{I}), \mathcal{I}(c) = v\}$.

Remarque. Si $\mathit{cells}(\mathcal{I}) = \mathcal{C}$, $\mathcal{I}$ correspond à une interprétation totale (au sens du chapitre 2).

L'ensemble des interprétations partielles sur un ensemble de cellules peut être totalement ordonné en utilisant l'extension lexicographique de l'ordre $<$ sur $\mathcal{C}$. Plus précisément, nous notons $\mathcal{I} < \mathcal{J}$ si et seulement si $\mathit{cells}(\mathcal{I}) = \mathit{cells}(\mathcal{J})$ et si il existe $c \in \mathit{cells}(\mathcal{I})$ telle que $\mathcal{I}(c) < \mathcal{J}(c)$ et pour tout $c' < c$, $\mathcal{I}(c') = \mathcal{J}(c')$. Pour toute interprétation $\mathcal{I}$ on note $\mathit{succ}(\mathcal{I})$ le successeur de $\mathcal{I}$ par rapport à cet ordre (i.e. le plus petit élément plus grand que $\mathcal{I}$), si celui-ci existe (sinon $\mathit{succ}(\mathcal{I})$ n'est pas défini). Donnons un exemple afin d'illustrer les notions que nous venons d'introduire.

Exemple 3.1.1. Soit $\mathcal{S} = \{T\}$ un ensemble de sortes. Soit $\Sigma = \{a, f\}$ avec $a :\rightarrow T$, $f : T \rightarrow T$. Supposons que $\mathcal{S}ize(T) = 2$. On a alors $\mathcal{D}om(T) = \{1, 2\}$. L'ensemble des cellules est $\mathcal{C} = \{a, f(1), f(2)\}$ (nous choisissons l'ordre $a < f(1) < f(2)$).

L'ensemble $\{a = 1, f(1) = 2, f(2) = 1\}$ est une interprétation sur $\mathcal{C}$.

Les ensembles $\mathcal{I}_1 \equiv \{a = 1, f(1) = 1, f(2) = 1\}$, et $\mathcal{I}_2 \equiv \{a = 1, f(1) = 2, f(2) = 2\}$ sont deux interprétations sur $\mathcal{C}$.

---

1. *Boolean* est un nouveau symbole de sorte, et *true*, *false* sont deux symboles de fonction de profil $\rightarrow \mathit{Boolean}$ (avec l'axiome $\mathit{true} \neq \mathit{false}$).

2. Nous autorisons dans les termes la présence d'éléments des domaines. Plus précisément, les éléments du domaine $\mathcal{D}om(s)$ seront considérés comme des constantes de sorte s (i.e. des membres de Σ de profil $\rightarrow s$). Ceci nous permet de considérer des termes de la forme $f(g(1), a, 2)$ ou $f(g(2), g(1), f(a, b, c))$, où $1, 2$ sont des éléments du domaine.

Nous avons $\mathcal{I}_1 < \mathcal{I}_2$, car $\mathcal{I}_1$ est plus petit que $\mathcal{I}_2$ (par rapport à l'extension lexicographique sur l'ordre sur $\mathcal{C}$) : on a en effet $\mathcal{I}_1(f(1)) < \mathcal{I}_2(f(1))$ et $\mathcal{I}_1(a) = \mathcal{I}_2(a)$.

Nous avons $succ(\mathcal{I}_1) = \{a = 1, f(1) = 1, f(2) = 2\}$ et $succ(\mathcal{I}_2) = \{a = 2, f(1) = 1, f(2) = 1\}$.

◇

Introduisons maintenant quelques notations utiles. Soit E un ensemble de cellules. Soit $\mathcal{I}$ et $\mathcal{J}$ deux interprétations telles que $E \subseteq cells(\mathcal{I})$ et $E \subseteq cells(\mathcal{J})$. On note $\mathcal{I}_{|E}$ l'ensemble $\{c = v \in \mathcal{I} / c \in E\}$. On note $\mathcal{I} <_{|E} \mathcal{J}$ si et seulement si $\mathcal{I}_{|E} < \mathcal{J}_{|E}$. Similairement, $\mathcal{I} =_{|E} \mathcal{J}$, si et seulement si $\mathcal{I}_{|E} = \mathcal{J}_{|E}$. $succ_E$ dénote la fonction successeur par rapport à cet ordre, i.e. la plus petite (par rapport à $<$) interprétation $\mathcal{J}$ telle que $\mathcal{I} <_{|E} \mathcal{J}$.

La cellule minimale de $\mathcal{C}$ est notée c_0. La cellule maximale est notée c_{max}.

L'ensemble de toutes les interprétations possibles est noté $\mathcal{I}nt$. L'interprétation *minimale*, c'est-à-dire l'interprétation $\{c = 1 / c \in \mathcal{C}\}$, est notée $\mathcal{I}_0$.

$\max(E)$ est la plus grande cellule de E. $trunc(E)$ est l'ensemble $E \setminus \{\max(E)\}$.

3.2 La méthode FMC

3.2.1 Algorithme de base

Une méthode très simple (évidemment très inefficace) pour construire un modèle fini d'une formule fermée est la méthode naïve consistant à énumérer toutes les interprétations sur le domaine et à tester (avec une procédure de décision évidente) si l'une d'entre elles satisfait $\mathcal{F}$. Pour cela, nous pouvons simplement énumérer les interprétations en utilisant l'ordre total $<$: il suffit de commencer par l'interprétation $\mathcal{I}_0$ définie par $\forall c \in \mathcal{C}.\mathcal{I}(c) = 1$ et de vérifier si $\mathcal{I}$ est un modèle de $\mathcal{F}$. Si $\mathcal{I} \not\models \mathcal{F}$, alors on considère l'interprétation $succ(\mathcal{I})$ et on répète cette opération jusqu'à obtenir un modèle ou jusqu'à atteindre l'interprétation maximale par rapport à $<$ (qui n'a pas de successeur).

Evidemment, cette méthode n'est pas utilisable en général[3]. Ainsi le but principal de ce chapitre est de définir des stratégies permettant d'éliminer certaines interprétations. Pour cela, nous remplaçons la fonction *succ* par une autre fonction ϕ satisfaisant certaines propriétés assurant à la fois la terminaison et la correction de l'algorithme. De façon informelle, ϕ est une fonction donnant, pour chaque interprétation $\mathcal{I}$, l'interprétation suivante $\phi(\mathcal{I})$ à considérer.

DÉFINITION 3.2.1.

Une fonction de saut *est une fonction partielle de $\mathcal{I}nt$ dans $\mathcal{I}nt$, qui satisfait les propriétés suivantes.*

($\mathcal{P}_1$) *Si $\mathcal{I} \not\models \mathcal{F}$ et si $\phi(\mathcal{I})$ existe, alors $\phi(\mathcal{I}) > \mathcal{I}$.*

($\mathcal{P}_2$) *Si $\mathcal{I} \not\models \mathcal{F}$, et si $\phi(\mathcal{I})$ existe, alors pour toute interprétation $\mathcal{J}$ telle que $\phi(\mathcal{I}) > \mathcal{J} \geq \mathcal{I}$, $\mathcal{J} \not\models \mathcal{F}$.*

($\mathcal{P}_3$) *Si $\mathcal{I} \not\models \mathcal{F}$, et si $\phi(\mathcal{I})$ n'existe pas alors pour toute interprétation $\mathcal{J} \geq \mathcal{I}$, $\mathcal{J} \not\models \mathcal{F}$.*

◇

L'algorithme pour la construction de modèle fini utilisant la fonction ϕ est donné dans la figure 3.1.

Intuitivement, ($\mathcal{P}_1$) garantit la *terminaison* de l'algorithme et ($\mathcal{P}_2$) et ($\mathcal{P}_3$) assurent la *complétude* (i.e. assurent que tous les modèles possibles peuvent être trouvés). L'utilisation de la fonction ϕ réduit de façon importante le nombre d'interprétations à considérer. Plus précisément, cela permet d'éviter de considérer les interprétations $\mathcal{J}$ telles que $\mathcal{J} > \mathcal{I}$ et $\mathcal{J} < \phi(\mathcal{I})$, si $\mathcal{I}$ a été considérée auparavant. Evidemment, notre objectif est de trouver une fonction permettant d'éliminer le plus d'interprétations possibles, i.e. $\phi(\mathcal{I})$ doit être aussi grande que possible pour

3. Le nombre d'interprétations à considérer est d^c ou c est le nombre de cellules et d la taille du domaine.

FMC

INPUT:
A first order formula $\mathcal{F}$
A function $\phi : \mathcal{I}nt \rightarrow \mathcal{I}nt$
OUTPUT:
A model $\mathcal{M}$ of $\mathcal{F}$
or a message **"no model found"**

begin
$\mathcal{I} = \{c = 1/c \in \mathcal{C}\}$
while $\phi(\mathcal{I})$ exists and $\mathcal{I} \not\models \mathcal{F}$
$\mathcal{I} = \phi(\mathcal{I})$
if $\mathcal{I} \models \mathcal{F}$ **then return**($\mathcal{I}$)
else return("no model found")
end

FIG. 3.1 - : ***F**inite **M**odel **C**onstruction: algorithme de base*

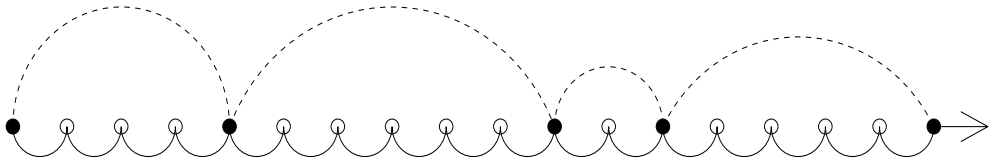


FIG. 3.2 - : *Effet de la fonction de saut*

chaque $\mathcal{I}$. La figure 3.2 illustre les effets de la fonction de saut. Les cercles (vides et pleins) représentent des interprétations. La ligne pleine représente le graphe de la fonction *succ* et la ligne discontinue représente le graphe de la fonction de saut ϕ. Les cercles vides représentent les interprétations éliminées grâce à l'utilisation de la fonction de saut ϕ.

THÉORÈME 3.2.1. *Soit ϕ une fonction de saut et $\mathcal{F}$ une formule du premier ordre. $FMC(\mathcal{F}, \phi)$ se termine. De plus si FMC retourne une interprétation $\mathcal{M}$ alors $\mathcal{M} \models \mathcal{F}$. Si FMC retourne* ***"no model found"*** *alors il n'existe aucun modèle de $\mathcal{F}$ sur le domaine considéré.*

PREUVE. $\mathcal{I}$ augmente strictement (propriété $(\mathcal{P}_1)$). Donc *FMC* se termine.

Si *FMC* retourne $\mathcal{M}$, $\mathcal{M}$ est évidemment un modèle de $\mathcal{F}$. Supposons que *FMC* retourne **"no model found"**. Soit $\mathcal{J}$ la plus grande interprétation telle que $\mathcal{J} \leq \mathcal{I}$ et telle qu'il existe n vérifiant $\mathcal{J} = \phi^n(\mathcal{I}_0)$ (n existe puisque on a $\mathcal{I}_0 \leq \mathcal{I}$ et $\mathcal{I}_0 = \phi^0(\mathcal{I}_0)$). Nous avons $\mathcal{J} \not\models \mathcal{F}$ (sinon *FMC* s'arrêterait à l'étape n).

Deux cas doivent être distingués.

1. Soit $\phi(\mathcal{J})$ existe. Alors par définition de $\mathcal{J}$, $\mathcal{J} \leq \mathcal{I} < \phi(\mathcal{J})$. Par $(\mathcal{P}_2)$, $\mathcal{I} \not\models \mathcal{F}$.
2. Soit $\phi(\mathcal{J})$ n'existe pas. Alors par définition de $\mathcal{J}$, $\mathcal{J} \leq \mathcal{I}$ et par $(\mathcal{P}_3)$, $\mathcal{I} \not\models \mathcal{F}$.

C.Q.F.D.

De nombreuses stratégies différentes peuvent être définies pour l'algorithme *FMC*, simplement en changeant la définition de la fonction de saut ϕ. Dans la suite de ce chapitre, nous donnons des exemples de fonctions possibles (respectivement notées $\phi_{ref}, \phi_{cref}, \phi'_{cref}$). L'objet de ces fonctions est d'utiliser l'information extraite de l'échec des interprétations considérées auparavant afin d'éliminer certaines interprétations.

3.2.2 La notion de réfutation

La première fonction de saut que nous proposons est très simple. Dans sa définition, on utilise la notion de réfutation d'une formule fermée $\mathcal{F}$ dans une interprétation $\mathcal{I}$. Dans un premier temps, nous introduisons informellement cette notion par un exemple.

EXEMPLE 3.2.1. Considérons la formule $\mathcal{F} : \forall x.x \neq f(x)$ sur le domaine $\{1, 2\}$. L'ensemble des cellules est $\{f(1), f(2)\}$. L'interprétation minimale (par rapport à $<$) est $\mathcal{I} : \{f(1) = 1, f(2) = 1\}$.

Nous avons de façon évidente $\mathcal{I} \not\models \mathcal{F}$. Si nous appliquons notre algorithme "naïf" (*FMC*), nous devons considérer ensuite les interprétations $(f(1) = 1, f(2) = 2)$ (i.e. $succ(\mathcal{I})$) et $(f(1) = 2, f(2) = 1)$ (i.e. $succ(succ(\mathcal{I}))$)...et tester pour chacune de ces interprétations si elle est un modèle de $\mathcal{F}$. Néanmoins, si on considère avec plus d'attention la première interprétation $\mathcal{I}$, il est facile de voir que l'équation $f(1) = 1$ suffit à prouver que $\mathcal{I} \not\models \mathcal{F}$. En effet, $f(1) = 1$ contredit l'axiome $\forall x.f(x) \neq x$. Il est donc inutile de considérer la valeur associée à la cellule $f(2)$ dans ce cas. Ainsi, il est inutile de considérer d'autres interprétations *si celles-ci ne changent pas la valeur de la cellule* $f(1)$. ◇

Par conséquent, au lieu de considérer le successeur de l'interprétation $\mathcal{I}$, nous prenons le premier successeur $\mathcal{J}$ de $\mathcal{I}$ tel que la valeur de $f(1)$ dans $\mathcal{J}$ est différente de 1, i.e. nous ne considérons pas l'interprétation $succ(\mathcal{I})$ mais seulement $succ(succ(\mathcal{I}))$.

De façon plus formelle, nous définissons la notion de réfutation comme suit.

DÉFINITION 3.2.2. *Une* réfutation[4] $\mathcal{R}$ *d'une formule (non valide) fermée du premier ordre* $\mathcal{F}$ *dans une interprétation* $\mathcal{I}$ *est un ensemble fini de cellules telles que pour toute interprétation* $\mathcal{J}$, *si* $\mathcal{I} =_{|\mathcal{R}} \mathcal{J}$, *alors* $\mathcal{J} \not\models \mathcal{F}$. ◇

Présentons en premier lieu un algorithme pour trouver les réfutations d'une formule donnée $\mathcal{F}$ dans une interprétation $\mathcal{I}$ (telle que $\mathcal{I} \not\models \mathcal{F}$). Evidemment, nous pourrions choisir l'ensemble $cells(\mathcal{I})$ lui-même (puisque $\mathcal{I} \not\models \mathcal{F}$, $cells(\mathcal{I})$ est de façon évidente une réfutation de $\mathcal{F}$) mais cela ne serait d'aucune utilité. En effet, nous cherchons des critères permettant d'identifier dès que possible qu'une interprétation falsifie une formule, donc nous avons besoin d'une réfutation qui soit *aussi petite que possible*, (par rapport au nombre de cellules). Trouver la plus petite réfutation est très difficile[5], donc nous ne chercherons pas à trouver une solution optimale pour ce problème.

Nous donnons ci-contre un algorithme permettant de trouver une réfutation pour une interprétation $\mathcal{I}$. Le principe de cet algorithme est tout simplement d'évaluer la formule $\mathcal{F}$ dans l'interprétation $\mathcal{I}$ et de conserver – pendant le processus d'évaluation – l'ensemble des cellules responsables de la valeur de $\mathcal{F}$ dans $\mathcal{I}$. Cet algorithme est défini par induction structurelle sur l'ensemble des termes et formules.

Sans perte de généralité, nous supposerons que la formule $\mathcal{F}$ est *rectifiée* (voir chapitre 2) et que $\mathcal{F}$ contient uniquement les symboles logiques $\neg, \vee$ et $\exists$. Nous définissons en premier lieu la procédure **EvaluateTerm** qui calcule la valeur d'un terme donné dans une interprétation, puis la procédure **Evaluate** qui retourne la valeur de vérité de la formule et la réfutation correspondante. Ces procédures sont respectivement spécifiées par les figures 3.3 et 3.4.

THÉORÈME 3.2.2. *Si* ***EvaluateTerm***$(t, \sigma, \mathcal{I})$= $(s, \mathcal{R})$ *alors*

– $\mathcal{I}(t\sigma) = s$;

4. Une réfutation correspond en fait à un contre-modèle partiel de la formule. Nous avons cependant préférer conserver la terminologie introduite par d'autres auteurs en construction de modèles finis.

5. $\emptyset$ est une réfutation de $\mathcal{F}$ si et seulement si la formule $\mathcal{F}$ est insatisfaisable sur le domaine considéré. Plus précisément, si nous nous restreignons à des formules propositionnelles, le problème de déterminer si une formule admet une réfutation minimale non vide est équivalent au problème SAT donc est NP-complet.

Procedure EvaluateTerm
INPUT:
A term t de sort s
A function $\sigma : \mathcal{V} \rightarrow \mathcal{D}om(s)$
An interpretation $\mathcal{I}$
OUTPUT:
A pair $(v, \mathcal{R})$
where v is the value of $t\sigma$ dans $\mathcal{I}$
and $\mathcal{R}$ is a refutation of $t\sigma \neq v$ in $\mathcal{I}$
begin
if $t \in \mathcal{V}$ **then return**$(t\sigma, \emptyset)$
else $\{\ t \equiv f(t_1, \ldots, t_n)\ \}$
for each $i \in \{1, \ldots, n\}$
$(v_i, \mathcal{R}_i)$ =**EvaluateTerm**$(t_i, \sigma, \mathcal{I})$
$v = \mathcal{I}(f(v_1, \ldots, v_n))$
$\mathcal{R} = \{f(v_1, \ldots, v_n)\} \cup \bigcup_{i=1}^{n} \mathcal{R}_i$
end

FIG. 3.3 - : *La procédure* ***EvaluateTerm***

– *$\mathcal{R}$ est une réfutation dans $\mathcal{I}$ de $t\sigma \neq s$.*

Si ***Evaluate***$(\mathcal{F}, \sigma, \mathcal{I})$= $(v, \mathcal{R})$ *alors*

– *Si v = false, alors $\mathcal{I} \not\models \mathcal{F}\sigma$ et $\mathcal{R}$ est une réfutation de $\mathcal{F}\sigma$ dans $\mathcal{I}$;*

– *Si v = true, alors $\mathcal{I} \models \mathcal{F}\sigma$ et $\mathcal{R}$ est une réfutation de $\neg\mathcal{F}\sigma$ dans $\mathcal{I}$.*

PREUVE. Par induction structurelle sur l'ensemble des termes et formules (voir les algorithmes pour les notations).

1. – Soit $t \in \mathcal{V}$. Soit $t\sigma = v$. Pour toute interprétation $\mathcal{J}$, nous avons par définition $\mathcal{J}(t\sigma) = v$. Donc $\mathcal{I}(t\sigma) = v$ et $\mathcal{R} = \emptyset$ est une réfutation de $t \neq v$.
   – Soit $t = f(t_1, \ldots, t_n)$. Soit $\mathcal{R} = \bigcup_{i=1}^{n} \mathcal{R}_i$. Par hypothèse d'induction $\mathcal{I}(t_i\sigma) = v_i$. D'où $\mathcal{I}(t\sigma) = \mathcal{I}(f(v_1, \ldots, v_n)) = v$. Alors, soit $\mathcal{J}$ une interprétation telle que $\mathcal{J} =_{|\mathcal{R}} \mathcal{I}$. Par définition de $\mathcal{R}$, pour tout $i \leq n$, on a $\mathcal{I} =_{|\mathcal{R}_i} \mathcal{J}$. Alors par hypothèse d'induction, $\mathcal{J}(t_i\sigma) = v_i$. De plus $\mathcal{J}(f(v_1, \ldots, v_n)) = v$, d'où $\mathcal{J}(t\sigma) = v$.
2. – Soit $\mathcal{F} \equiv t = s$. Par 1, nous savons que $\mathcal{I}(t) = v_1$ et $\mathcal{I}(s) = v_2$. D'où $\mathcal{I} \models \mathcal{F}$ si et seulement si $v_1 = v_2$. Soit $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Alors, pour tout $\mathcal{J}$ telle que $\mathcal{J} =_{|\mathcal{R}} \mathcal{I}$ nous avons $\mathcal{J} =_{|\mathcal{R}_i} \mathcal{I}$ donc par hypothèse d'induction, $\mathcal{J}(t) = v_1$ et $\mathcal{J}(s) = v_2$. Donc $\mathcal{J} \models \mathcal{F}$ si et seulement si $v_1 = v_2$.
   – Soit $\mathcal{F} \equiv \mathcal{F}_1 \vee \mathcal{F}_2$. Soit $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Deux cas sont à distinguer. Soit v_1 est vrai soit v_1 est faux. Si v_1 est vrai, alors pour tout $\mathcal{J}$ telle que $\mathcal{J} =_{|\mathcal{R}_1} \mathcal{I}$, nous avons par hypothèse d'induction, $\mathcal{J} \models \mathcal{F}_1\sigma$, donc $\mathcal{J} \models \mathcal{F}\sigma$. Si v_1 est faux alors pour tout $\mathcal{J}$ telle que $\mathcal{J} =_{|\mathcal{R}_1 \cup \mathcal{R}_2} \mathcal{I}$, $\mathcal{J} =_{|\mathcal{R}_1} \mathcal{I}$ donc la valeur de vérité de $\mathcal{F}_1$ dans $\mathcal{J}$ est v_1. De plus $\mathcal{J} =_{|\mathcal{R}_2} \mathcal{I}$ donc la valeur de vérité de $\mathcal{F}_2$ dans $\mathcal{J}$ est v_2. Donc la valeur de vérité de $\mathcal{F}$ dans $\mathcal{J}$ est $v_1 \vee v_2$, donc v_2.
   – Soit $\mathcal{F} = \exists x.\mathcal{F}'$. Soit $(v_i, \mathcal{R}_i)$ la valeur retournée par **Evaluate** sur $\mathcal{F}', \sigma \cup \{x \rightarrow i\}$. Considérons deux cas. Soit il existe un $i \in \mathcal{D}om(s)$ tel que v_i est vrai. Alors, par hypothèse d'induction, nous avons, pour tout $\mathcal{J}$ telle que $\mathcal{J} =_{|\mathcal{R}_i} \mathcal{I}$: $\mathcal{J} \models \mathcal{F}'\sigma\{x \rightarrow v_i\}$ donc $\mathcal{J} \models \exists x.\mathcal{F}'\sigma$. Sinon pour tout $i \in \mathcal{D}om(s)$, v_i est faux. Soit $\mathcal{R} = \bigcup_i \mathcal{R}_i$. Soit $\mathcal{J}$ telle que, $\mathcal{J} =_{|\mathcal{R}} \mathcal{I}$. Par hypothèse d'induction et puisque $\mathcal{J} =_{|\mathcal{R}_i} \mathcal{I}$, $\mathcal{J} \not\models \mathcal{F}\sigma\{x \rightarrow i\}$. Alors $\mathcal{J} \not\models \exists x.\mathcal{F}'\sigma$.
   – Soit $\mathcal{F} = \neg\mathcal{F}'$. Soit $\mathcal{J}$ une interprétation telle que $\mathcal{J} =_{|\mathcal{R}} \mathcal{I}$. Alors $\mathcal{J} \models \mathcal{F}'$ si et seulement si v est vrai. Donc $\mathcal{J} \models \mathcal{F}$ si et seulement si v est faux.

Procedure Evaluate:
INPUT:
A formula $\mathcal{F}$
A function $\sigma : \mathcal{V} \rightarrow \mathcal{D}om$
An interpretation $\mathcal{I}$
OUTPUT:
A pair $(v, \mathcal{R})$
where v is the truth value of $\mathcal{F}\sigma$ in $\mathcal{I}$
and if $v = false$ then $\mathcal{R}$ is a refutation of $\mathcal{F}$ in $\mathcal{I}$.
else $\mathcal{R}$ is a refutation of $\neg\mathcal{F}$ in $\mathcal{I}$.
begin
if $\mathcal{F} \equiv s = t$
then
$(v_1, \mathcal{R}_1)$ =**EvaluateTerm**$(t, \sigma, \mathcal{I})$
$(v_2, \mathcal{R}_2)$ =**EvaluateTerm**$(s, \sigma, \mathcal{I})$
return$(v_1 \equiv v_2, \mathcal{R}_1 \cup \mathcal{R}_2)$
else if $\mathcal{F} \equiv \neg\mathcal{F}'$
$(v, \mathcal{R})$ ≡**Evaluate**$(\mathcal{F}', \sigma, \mathcal{I})$
return$(\neg v, \mathcal{R})$
else if $\mathcal{F} \equiv \mathcal{F}_1 \vee \mathcal{F}_2$
$(v_1, \mathcal{R}_1)$ =**Evaluate**$(\mathcal{F}_1, \sigma, \mathcal{I})$
if v_1 **then**
return$(v_1, \mathcal{R}_1)$
else
$(v_2, \mathcal{R}_2)$ =**Evaluate**$(\mathcal{F}_2, \sigma, \mathcal{I})$
if v_2 **then**
return$(v_2, \mathcal{R}_2)$
else **return**$(v_1 \vee v_2, \mathcal{R}_1 \cup \mathcal{R}_2)$
else $\{\mathcal{F} \equiv \exists x.\mathcal{F}_1\}$
Let s be the sort symbol such that: $x \in \mathcal{V}_s$.
$\mathcal{R} = \emptyset$
for each $v \in \mathcal{D}om(s)$
$(v', \mathcal{R}')$ =**Evaluate**$(\mathcal{F}, \sigma \cup \{x \rightarrow v\}, \mathcal{I})$
if v' **then return**$(v', \mathcal{R}')$
$\mathcal{R} = \mathcal{R} \cup \mathcal{R}'$
return$(false, \mathcal{R}')$
end

FIG. 3.4 - : *La procédure* ***Evaluate***

C.Q.F.D.

Nous pouvons maintenant définir la fonction de saut ϕ_{ref}. Son principe est très simple. Au lieu de considérer le successeur immédiat de l'interprétation courante $\mathcal{I}$, nous considérons la plus petite interprétation $\mathcal{J}$ supérieure à $\mathcal{I}$ et telle que $\mathcal{I} \neq_{|\mathcal{R}} \mathcal{J}$ (où $\mathcal{R}$ est la réfutation calculée par **Evaluate**). Plus formellement :

DÉFINITION 3.2.3. *Pour toute interprétation $\mathcal{I}$ telle que $\mathcal{I} \not\models \mathcal{F}$, alors*

$$\phi_{ref}(\mathcal{I}) = succ_{\mathcal{R}}(\mathcal{I}).$$

Si: ***Evaluate***$(\mathcal{F}, \emptyset, \mathcal{I}) = (false, \mathcal{R})$ *(ϕ_{ref} est non définie sinon).* ◇

THÉORÈME 3.2.3. *ϕ_{ref} est une fonction de saut.*

PREUVE. Nous montrons que ϕ_{ref} satisfait $(\mathcal{P}_1), (\mathcal{P}_2)$ et $(\mathcal{P}_3)$.

$(\mathcal{P}_1)$ la preuve est évidente.

$(\mathcal{P}_2)$ Soit $\mathcal{J}$ une interprétation telle que $\mathcal{I} \leq \mathcal{J} < \phi_{ref}(\mathcal{I})$. Par définition $\phi_{ref}(\mathcal{I})$ est la plus petite interprétation $\mathcal{H}$ telle que $\mathcal{H} \neq_{|\mathcal{R}} \mathcal{I}$. Donc $\mathcal{I} =_{|\mathcal{R}} \mathcal{J}$, donc $\mathcal{J} \not\models \mathcal{F}$ (puisque $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$).

$(\mathcal{P}_3)$ Soit $\mathcal{J}$ une interprétation telle que $\mathcal{I} \leq \mathcal{J}$. Par définition, si $\phi_{ref}(\mathcal{I})$ n'existe pas, alors pour tout $\mathcal{H} > \mathcal{I}$, nous avons $\mathcal{H} =_{|\mathcal{R}} \mathcal{I}$, donc $\mathcal{H} \not\models \mathcal{F}$.

C.Q.F.D.

3.2.3 Réfutation couvrante

Cette fonction de saut peut être améliorée en considérant non plus une réfutation dans l'interprétation courante (générée par **Evaluate**) mais plutôt l'ensemble des cellules utilisées depuis le début de la recherche, i.e. l'union de toutes les réfutations générées dans le passé. Pendant la recherche d'un modèle, nous allons conserver toutes les cellules appartenant à une des réfutations que nous avons déjà générées. Intuitivement, cet ensemble de cellules sera l'ensemble de cellules responsable de l'échec de toutes les interprétations déjà testées. Nous montrerons par la suite que cet ensemble possède certaines propriétés intéressantes permettant de simplifier au fur et à mesure la réfutation obtenue. Pour décrire cette idée d'une façon plus formelle, il faut introduire la notion nouvelle de *réfutation couvrante*. Pour cela, nous devons introduire quelques notations. Pour toute cellule c, nous notons c^- l'ensemble $[c_0, c[$ (i.e. l'ensemble de cellules x telles que $c_0 \leq x < c$) et c^+ l'ensemble $[c, c_{max}]$ (i.e. l'ensemble de cellules x telles que $c \leq x \leq c_{max}$).

DÉFINITION 3.2.4. *Soit $\mathcal{I}$ une interprétation. Un ensemble de cellules $\mathcal{R}$ est appelé une* réfutation couvrante *de $\mathcal{F}$ dans $\mathcal{I}$, si et seulement si pour toute cellule c et pour toute interprétation $\mathcal{J}$, si $\mathcal{I} =_{|c^- \cap \mathcal{R}} \mathcal{J}$ et $\mathcal{J} <_{|c^+} \mathcal{I}$ alors $\mathcal{J} \not\models \mathcal{F}$.* ◇

REMARQUE. Si $\mathcal{R}$ est une réfutation couvrante de $\mathcal{F}$, dans $\mathcal{I}$, alors pour tout $\mathcal{J} < \mathcal{I}$, $\mathcal{J} \not\models \mathcal{F}$ (il suffit de prendre $c = c_0$ dans la définition ci-dessus).

Les lemmes suivants énoncent quelques propriétés intéressantes des réfutations couvrantes.

DÉFINITION 3.2.5. *Une réfutation couvrante $\mathcal{R}$ de $\mathcal{F}$ dans $\mathcal{I}$ est dite* normalisée *si et seulement si $\mathcal{R} = \emptyset$ ou si $\mathcal{I}(\max(\mathcal{R})) \neq \mathcal{S}ize(\max(\mathcal{R}))$.* ◇

LEMME 3.2.1. *Soit $\mathcal{R}$ une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$. Si $\mathcal{R}$ n'est pas normalisée alors $trunc(\mathcal{R})$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$. Si de plus $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$ alors $trunc(\mathcal{R})$ est également une réfutation de $\mathcal{F}$ dans $\mathcal{I}$.*

PREUVE. Nous notons $m = \max(\mathcal{R})$ et $\mathcal{R}' = trunc(\mathcal{R})$.

Supposons que $\mathcal{R}'$ n'est pas une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$. Alors il existe une interprétation $\mathcal{J}$ et une cellule c telles que $\mathcal{J} =_{|c^- \cap \mathcal{R}'} \mathcal{I}$, $\mathcal{J} <_{|c^+} \mathcal{I}$ et $\mathcal{J} \models \mathcal{F}$.

Supposons que $c \leq m$. Alors $c^- \cap \mathcal{R} = c^- \cap \mathcal{R}'$. Donc $\mathcal{J} =_{|c^- \cap \mathcal{R}} \mathcal{I}$. Puisque $\mathcal{J} <_{|c^+} \mathcal{I}$ nous en déduisons que $\mathcal{J} \not\models \mathcal{F}$, ce qui est impossible. Donc $c > m$. Si $\mathcal{J}(m) = \mathcal{D}om(m)$, alors $\mathcal{J}_{|c^- \cap \mathcal{R}} = \mathcal{I}_{|c^- \cap \mathcal{R}}$. Mais puisque $\mathcal{J} <_{|c^+} \mathcal{I}$, $\mathcal{J} \not\models \mathcal{F}$ (puisque $\mathcal{R}$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$). Donc $\mathcal{J}(m) \neq \mathcal{D}om(m)$, i.e. $\mathcal{J}(m) < \mathcal{D}om(m)$. On a $\mathcal{J} <_{|m^+} \mathcal{I}$ et $\mathcal{J} =_{|m^- \cap \mathcal{R}} \mathcal{I}$, donc $\mathcal{J} \not\models \mathcal{F}$, ce qui est impossible.

Maintenant supposons que $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$. Supposons que $\mathcal{J} =_{|\mathcal{R}'} \mathcal{I}$. Alors soit $\mathcal{J}(m) = \mathcal{I}(m)$, et dans ce cas $\mathcal{J} =_{|\mathcal{R}} \mathcal{I}$ donc $\mathcal{J} \not\models \mathcal{F}$, ou $\mathcal{J} <_{|m^+} \mathcal{I}$ (puisque $\mathcal{I}(m) = \mathcal{D}om(m)$) donc $\mathcal{J} \not\models \mathcal{F}$. C.Q.F.D.

Donc nous pouvons supposer sans perte de généralité que toute réfutation couvrante est normalisée. En effet, si $\mathcal{R}$ n'est pas normalisée, nous pouvons supprimer l'élément $\max(\mathcal{R})$ de $\mathcal{R}$ et obtenir ainsi une nouvelle réfutation couvrante. En répétant ce processus, il est possible d'obtenir une réfutation couvrante normalisée, notée $\mathcal{N}(\mathcal{R})$.

LEMME 3.2.2. *Si $\mathcal{R}$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$, et si $trunc(\mathcal{R})$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$ alors $trunc(\mathcal{R})$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$.*

PREUVE. $\mathcal{R}' = trunc(\mathcal{R})$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$. Supposons que $\mathcal{R}'$ n'est pas une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$. Alors il existe une interprétation $\mathcal{J}$ et une cellule $c \in \mathcal{R}$ telles que $\mathcal{J} =_{|c^- \cap \mathcal{R}'} \mathcal{I}$, $\mathcal{J} <_{|c^+} \mathcal{I}$ et $\mathcal{J} \models \mathcal{F}$.

Si $c \leq m$, $c^- \cap \mathcal{R} = c^- \cap \mathcal{R}'$. Donc $\mathcal{J} =_{|c^- \cap \mathcal{R}} \mathcal{I}$ donc (puisque $\mathcal{R}$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$) $\mathcal{J} \not\models \mathcal{F}$. Donc $c > m$. $\mathcal{R}' = \mathcal{R}' \cap c^-$, donc $\mathcal{R}' \cap c^-$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$, donc $\mathcal{J} \not\models \mathcal{F}$.

C.Q.F.D.

REMARQUE. Les propriétés énoncées par les lemmes 3.2.1 et 3.2.2 sont très importantes car elles permettent de *simplifier les réfutations pendant la recherche*. Remarquons que le lemme 3.2.1 n'en est plus un si nous considérons des réfutations au lieu des réfutations couvrantes, ce qui montre l'intérêt de cette nouvelle notion.

L'utilisation du lemme 3.2.2 pour simplifier une réfutation nécessite un test pour décider si un ensemble de cellules est une réfutation, ce qui est en pratique très coûteux. Nous nous contenterons donc d'une condition plus forte, donnée par l'opérateur de simplification suivant.

$$\mathcal{N}_{\mathcal{R}'}(\mathcal{R}) = \mathcal{N}_{\mathcal{R}'}(trunc(\mathcal{R})) \text{ si } \mathcal{R} \text{ n'est pas normalisée ou si } \mathcal{R}' \subseteq trunc(\mathcal{R}).$$

$$\mathcal{N}_{\mathcal{R}'}(\mathcal{R}) = \mathcal{R} \text{ sinon}$$

LEMME 3.2.3. *Si $\mathcal{R}$ est une réfutation couvrante et $\mathcal{R}'$ une réfutation, alors $\mathcal{N}_{\mathcal{R}'}(\mathcal{R})$ est une réfutation couvrante. Si de plus $\mathcal{R}$ est une réfutation alors $\mathcal{N}_{\mathcal{R}'}(\mathcal{R})$ est une réfutation.*

PREUVE. C'est une conséquence immédiate des lemmes 3.2.1 et 3.2.2. C.Q.F.D.

LEMME 3.2.4. *Si $\mathcal{R}$ est une réfutation couvrante de $\mathcal{I}$ par rapport à $\mathcal{F}$ alors pour tout ensemble de cellules E, $\mathcal{R} \cup E$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$.*

PREUVE. (triviale) Si $\mathcal{J} =_{|c^- \cap (\mathcal{R} \cup E)} \mathcal{I}$ et $\mathcal{J} <_{c^+} \mathcal{I}$ alors $\mathcal{J} =_{|c^- \cap \mathcal{R}} \mathcal{I}$ donc $\mathcal{J} \not\models \mathcal{F}$. C.Q.F.D.

LEMME 3.2.5. *Si $\mathcal{R}$ est une réfutation couvrante normalisée de $\mathcal{F}$ dans $\mathcal{I}$ et si $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$, alors*

1. *Si $succ_{\mathcal{R}}(\mathcal{I})$ existe, alors $trunc(\mathcal{R})$ est une réfutation couvrante de $\mathcal{F}$ dans $succ_{\mathcal{R}}(\mathcal{I})$.*

2. *Si $succ_{\mathcal{R}}(\mathcal{I})$ n'existe pas, alors $\mathcal{F}$ est insatisfaisable sur le domaine considéré.*

PREUVE. 1. Soit $m = \max(\mathcal{R})$. Soit $\mathcal{J}$ une interprétation. Nous notons $\mathcal{I}' = succ_{\mathcal{R}}(\mathcal{I})$ et $\mathcal{R}'' = trunc(\mathcal{R})$.
Supposons qu'il existe une cellule c et une interprétation $\mathcal{J}$ telles que $\mathcal{I}' =_{|c^- \cap \mathcal{R}''} \mathcal{J}$, $\mathcal{J} <_{|c^+} \mathcal{I}'$ et $\mathcal{J} \models \mathcal{F}$. Par définition de $succ_{\mathcal{R}}$, $\forall c > m$, $\mathcal{I}'(c) = 1$. Donc si $c > m$, alors $\mathcal{I}'_{|c^+} = \{c' = 1/c' \geq c\}$, donc $\mathcal{I}'_{|c^+}$ est minimal par rapport à $<$, ce qui est impossible puisque $\mathcal{J}_{|c^+} < \mathcal{I}'_{|c^+}$. Donc $c \leq m$.
Donc $\mathcal{R}'' \cap c^- = \mathcal{R} \cap c^-$.
Par définition de $succ_{\mathcal{R}}$, puisque $\mathcal{I}(m) \neq \mathcal{D}om(m)$, $\mathcal{I} =_{|m^-} \mathcal{I}'$. Par conséquent $\mathcal{J} =_{|\mathcal{R} \cap c^-} \mathcal{I}$.
Si $\mathcal{J} <_{|c^+} \mathcal{I}$, $\mathcal{J} \not\models \mathcal{F}$ (puisque $\mathcal{R}$ est une réfutation couvrante dans $\mathcal{I}$). Si $\mathcal{J} =_{|c^+} \mathcal{I}$, alors $\mathcal{I} =_{|\mathcal{R}} \mathcal{J}$, donc $\mathcal{J} \not\models \mathcal{F}$ (en effet, $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$). Donc $\mathcal{I} <_{|c^+} \mathcal{J} <_{|c^+} \mathcal{I}'$
Puisque $\mathcal{R}$ est normalisée, $\mathcal{I}(m) \neq \mathcal{D}om(m)$, donc par définition de $succ_{\mathcal{R}}$: $\mathcal{I}'(m) = \mathcal{I}(m) + 1$ (puisque $\mathcal{I}' =_{m^-} \mathcal{I}$). De plus, $\mathcal{J}_{|c^+}$ doit être de la forme $\mathcal{I}_{|[c,m]} \cup E$. où $E >_{|]m,c_{max}]} \mathcal{I}$.
Alors $\mathcal{J} =_{\mathcal{R}} \mathcal{I}$ et $\mathcal{J} \not\models \mathcal{F}$.

2. Puisque $succ_{\mathcal{R}}(\mathcal{I})$ n'existe pas, on a (puisque $\mathcal{R}$ est normalisée), $\mathcal{R} = \emptyset$. Puisque $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$ nous en déduisons que pour toute interprétation $\mathcal{J}$: $\mathcal{J} \not\models \mathcal{F}$.

C.Q.F.D.

Les lemmes 3.2.1 et 3.2.5 permettent de calculer – pendant la recherche du modèle – une réfutation couvrante $\mathcal{R}$ de $\mathcal{F}$ dans l'interprétation courante $\mathcal{I}$. Cette réfutation couvrante $\mathcal{R}$ sera notée α. La réfutation $\mathcal{R}'$ générée par la procédure **Evaluate** est notée β.

L'utilisation des réfutations couvrantes permet d'éliminer certaines interprétations pendant la recherche. De façon plus précise, la formule $\mathcal{F}$ est fausse dans toutes les interprétations entre $\mathcal{I}$ et $succ_{\alpha}(\mathcal{I})$, donc ces interprétations n'ont pas besoin d'être considérées. Formellement, nous définissons la fonction ϕ_{cref} comme suit.

DÉFINITION 3.2.6. *Définissons la suite suivante (rappelons que $\mathcal{I}_0$ est l'interprétation minimale) :*

$$
\begin{array}{ll}
\alpha_0 = \mathcal{N}_{\beta_0}(\beta_0) & \\
(v_i, \beta_i) = Evaluate(\mathcal{F}, \emptyset, \mathcal{I}_i) & \textit{si } \mathcal{I}_i \textit{ est définie} \\
\alpha_{i+1} = \mathcal{N}_{\beta_{i+1}}(trunc(\alpha_i) \cup \beta_{i+1}) & \textit{si } \mathcal{I}_i \textit{ est définie} \\
\mathcal{I}_{i+1} = succ_{\alpha_i}(\mathcal{I}_i) & \textit{si } v_i = \textit{false} \\
\mathcal{I}_{i+1} = \textit{undefined} & \textit{sinon}
\end{array}
$$

La fonction ϕ_{cref} est alors définie par :

$$\phi_{cref}(\mathcal{I}) = \mathcal{I}_{i+1} \textit{ si } \mathcal{I} = \mathcal{I}_i$$

Si pour tout i, $\mathcal{I}$ n'est pas égale à $\mathcal{I}_i$, $\phi_{cref}(\mathcal{I})$ n'est pas définie. ◇

Le nouvel algorithme est spécifié dans la figure 3.5.

THÉORÈME 3.2.4. *L'algorithme FMC se termine. De plus si FMC retourne $\mathcal{M}$ alors $\mathcal{M} \models \mathcal{F}$ et si FMC retourne* ***"no model found"*** *alors $\mathcal{F}$ est insatisfaisable sur le domaine.*

PREUVE. D'après le théorème 3.2.2, β est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$. Par les lemmes 3.2.2, 3.2.4 et 3.2.5, nous savons que, à chaque étape de l'algorithme, α est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$. Montrons que ϕ_{cref} est une fonction de saut.
ϕ_{cref} satisfait de façon évidente ($\mathcal{P}_1$). Soit $\mathcal{I}$ une interprétation.

– Si $\phi_{cref}(\mathcal{I})$ existe, alors d'après le lemme 3.2.5, il existe une réfutation couvrante de $\phi_{cref}(\mathcal{I})$. Alors pour tout $\mathcal{J} < \phi_{cref}(\mathcal{I})$, $\mathcal{J} \not\models \mathcal{F}$.

Procedure *FMC*{ **F**inite **M**odel **C**onstruction }

INPUT:
A formula $\mathcal{F}$
OUTPUT
A model M de $\mathcal{F}$
or un message: **"no model found"**

begin
$\mathcal{I} = \{v = 1/v \in \mathcal{C}\}$
$\alpha = \emptyset$ { α is a covering refutation of $\mathcal{F}$ w.r.t. $\mathcal{I}$ }
(v, β) =**Evaluate**$(\mathcal{F}, \emptyset, \mathcal{I})$
{ β is a refutation of $\mathcal{F}$ w.r.t. $\mathcal{I}$ }
while $v \equiv$*false*
$\alpha = \mathcal{N}_\beta(\alpha \cup \beta)$
if $succ_\alpha(\mathcal{I})$ exists
then
$\mathcal{I} = \phi_{cref}(\mathcal{I})$
$\alpha = trunc(\alpha)$
else return(**"no model found"**)
(v, β) =**Evaluate**$(\mathcal{F}, \emptyset, \mathcal{I})$
end{ **while** }
return$(\mathcal{I})$
end

FIG. 3.5 - : ***F**inite **M**odel **C**onstruction*

– Si $\phi_{cref}(\mathcal{I})$ n'existe pas, alors d'après le lemme 3.2.5, $\mathcal{J} \not\models \mathcal{F}$, pour tout $\mathcal{J} \geq \mathcal{I}$.

Donc d'après le théorème 3.2.1,

– *FMC* se termine ;

– si *FMC* retourne $\mathcal{M}$ alors $\mathcal{M} \models \mathcal{F}$;

– si *FMC* retourne **"no model found"** alors $\mathcal{F}$ est insatisfaisable sur le domaine.

C.Q.F.D.

3.2.4 Une amélioration

Une analyse plus profonde de la preuve du lemme 3.2.5 amène à définir une fonction de saut ϕ'_{cref} plus efficace, permettant, dans certains cas particuliers, d'éliminer un plus grand nombre d'interprétations que la fonction ϕ_{cref}.

Cette nouvelle fonction de saut est fondée sur le lemme suivant.

LEMME 3.2.6. *Soit $\mathcal{F}$ une formule. Soit $\mathcal{I}$ une interprétation, $\mathcal{R}$ une réfutation couvrante normalisée et une réfutation de $\mathcal{F}$ dans $\mathcal{I}$. Soit $\mathcal{R}' = trunc(\mathcal{R})$. Soit $m = \max(\mathcal{R})$*

Soit $\mathcal{J}$ une interprétation identique à $\mathcal{I}$ excepté pour la cellule m dont la valeur est incrémentée de 1. $\mathcal{J}$ est formellement définie par

$\mathcal{J} = \mathcal{I}_{|\mathcal{C}\setminus\{m\}} \cup \{m = \mathcal{I}(m) + 1\}$.

Si $\mathcal{R}'$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$, alors $\mathcal{R}'$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{J}$.

PREUVE. Soit c une cellule. Soit $\mathcal{H}$ une interprétation telle que $\mathcal{H} <_{|c^+} \mathcal{J}$ et $\mathcal{H} =_{|c^- \cap \mathcal{R}'} \mathcal{J}$. Supposons que $\mathcal{H} \models \mathcal{F}$.

Si $c \leq m$, alors $\mathcal{J} =_{|c^- \cap \mathcal{R}'} \mathcal{I}$. Donc $\mathcal{H} =_{|c^- \cap \mathcal{R}'} \mathcal{I}$. De plus, $\mathcal{H} <_{c^+} \mathcal{J}$. Donc il existe $c' \geq c$ tel que $\mathcal{H}(c') < \mathcal{J}(c')$ et $\forall c'' \in [c..c'[.\mathcal{J}(c) = \mathcal{H}(c)$. Si $c' > m$, $\mathcal{H}_{\mathcal{R}} = \mathcal{I}_{\mathcal{R}}$. Donc $\mathcal{H} \not\models \mathcal{F}$. Si $c' \leq m$, on a $\mathcal{H} =_{|c'^- \cap \mathcal{R}'} \mathcal{I}$ et $\mathcal{H} <_{c'^+} \mathcal{I}$ donc $\mathcal{H} \not\models \mathcal{F}$ (puisque $\mathcal{R}'$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$).

Alors nous devons avoir $c > m$.

Si $\mathcal{H} <_{|c^+} \mathcal{I}$, $\mathcal{H} \not\models \mathcal{F}$. (puisque $\mathcal{R}'$ est une réfutation couvrante de $\mathcal{F}$ dans $\mathcal{I}$.). De façon similaire si $\mathcal{H} =_{|c^+} \mathcal{I}$, nous avons $\mathcal{H} =_{|\mathcal{R}} \mathcal{I}$, donc $\mathcal{H} \not\models \mathcal{F}$.

Donc on a $\mathcal{J} >_{|c^+} \mathcal{H} >_{|c^+} \mathcal{I}$. Ce qui est impossible puisque $\mathcal{J} =_{|c^+} \mathcal{I}$ (car $c > m$). C.Q.F.D.

DÉFINITION 3.2.7. *Soit $\mathcal{I}$ une interprétation.*

Définissons la fonction ϕ'_{cref} comme suit.

Modifions tout d'abord la définition de la suite $(\mathcal{I}_i)$.

- $m_i = \max(\alpha_i)$.
- $\mathcal{I}_{i+1} = \mathcal{I}_{i|\mathcal{C}\setminus\{m\}} \cup \{m = \mathcal{I}_i(m) + 1\}$ *si* $\mathcal{I}_i \not\models \mathcal{F}$ *et si* $m_i \notin \alpha_{i-1}$.
- $\mathcal{I}_{i+1} = succ_{\alpha_i}(\mathcal{I}_i)$, *si* $\mathcal{I}_i \not\models \mathcal{F}$, *et si* $m \in \alpha_{i-1}$.
- *sinon* $\mathcal{I}_{i+1}$ *n'est pas définie.*

$$\phi'_{cref}(\mathcal{I}) = \mathcal{I}_{i+1} \text{ si } \mathcal{I} = \mathcal{I}_i$$

Si pour tout i, $\mathcal{I}$ n'est pas égale à $\mathcal{I}_i$, $\phi'_{cref}(\mathcal{I})$ n'est pas définie. ◇

THÉORÈME 3.2.5. *$\phi'_{cref}(\mathcal{I})$ est une fonction de saut.*

PREUVE. C'est une conséquence triviale des lemmes 3.2.6 et 3.2.5. C.Q.F.D.

3.2.5 Détection des symétries

Présentation informelle

La stratégie que nous proposons dans cette section utilise la symétrie naturelle entre les éléments du domaine d'une sorte s donnée. Considérons une sorte s et une permutation σ de $\mathcal{D}om(s)$. Nous avons $\mathcal{I}\sigma \models \mathcal{F}\sigma$ si et seulement si $\mathcal{I} \models \mathcal{F}$.

Donc, si aucun élément de $\mathcal{D}om(s)$ n'apparaît dans la formule initiale $\mathcal{F}$, alors pour toute interprétation $\mathcal{I}$ de $\mathcal{F}$,

$$\mathcal{I} \models \mathcal{F} \text{ si et seulement } \mathcal{I}\sigma \models \mathcal{F}$$

Ceci montre que pour une interprétation $\mathcal{I}$ donnée, il existe de nombreuses interprétations isomorphes à $\mathcal{I}$, où $\mathcal{F}$ a la même valeur de vérité que dans $\mathcal{I}$. Il est inutile de considérer toutes ces possibilités mais seulement l'une d'entre elles.

Illustrons ce fait par un exemple (délibérément trivial).

EXEMPLE 3.2.2. Soit Σ la signature $\{a :\rightarrow T, b :\rightarrow T\}$ avec $\mathcal{S}ize(T) = 3$. Soit $\mathcal{F}$ la formule $a = b \wedge a \neq b$.

La procédure *FMC* considère successivement les interprétations $\mathcal{I}_1 \equiv \{a = 1, b = 1\}$,$\mathcal{I}_2 \equiv \{a = 1, b = 2\}$ et $\mathcal{I}_3 \equiv \{a = 1, b = 3\}$. Ici $\mathcal{I}_2$ et $\mathcal{I}_3$ sont de façon évidente symétriques: il suffit en effet d'échanger les éléments 2 et 3. Il est donc inutile d'évaluer la formule $\mathcal{F}$ dans l'interprétation $\mathcal{I}_3$. On peut directement affirmer que $\mathcal{I}_3 \not\models \mathcal{F}$, puisque $\mathcal{I}_2 \not\models \mathcal{F}$ et $\mathcal{I}_3 = \sigma(\mathcal{I}_2)$. Par la suite, *FMC* teste l'interprétation $\mathcal{I}_4 = \{a = 2, b = 1\}$. Là encore $\mathcal{I}_4$ et $\mathcal{I}_2$ sont symétriques puisque $\mathcal{I}_4 = \sigma\mathcal{I}_2$. (avec $\sigma(2) = 1$ et $\sigma(1) = 2$). Donc $\mathcal{I}_4 \not\models \mathcal{F}$. De façon similaire, toutes les interprétations restantes $\mathcal{I}_5 = \{a = 2, b = 2\}$ $\mathcal{I}_6 = \{a = 2, b = 3\}$, $\mathcal{I}_7 = \{a = 3, b = 1\}$...sont symétriques à une interprétation déjà testée $\mathcal{I}_1$ ou $\mathcal{I}_2$. Donc nous pouvons arrêter la recherche et retourner **"no model found"**.

Ceci permet de considérer seulement deux interprétations, alors qu'une exploration naïve de l'espace de recherche aurait donné $3 \times 3 = 9$ interprétations possibles. ◇

Définition des symétries

Afin de présenter formellement notre méthode, nous introduisons le théorème suivant.

LEMME 3.2.7. *Soit σ une permutation de $\mathcal{D}om(s)$. Soit $\mathcal{F}$ une formule. Soit $\mathcal{I}$ un modèle de $\mathcal{F}$. Alors $\sigma(\mathcal{I})$ est un modèle de $\sigma(\mathcal{F})$.*

PREUVE. $\sigma(\mathcal{I})$ est de façon évidente une interprétation, puisque σ est bijective. De plus, nous avons pour tout terme t, $(\sigma(\mathcal{I}))(t) = \sigma(\mathcal{I}(t))$.

Nous montrons par induction structurelle sur l'ensemble des formules que pour toute substitution σ, $\sigma(\mathcal{I}) \models \sigma(\mathcal{F})$.

1. Supposons que $\mathcal{F}$ est de la forme $t = s$. Alors $\mathcal{I} \models (t = s)$ i.e. $\mathcal{I}(t) = \mathcal{I}(s)$, donc $\sigma(\mathcal{I}(t)) = \sigma(\mathcal{I}(s))$ i.e. $\sigma(\mathcal{I}) \models \sigma(\mathcal{F})$.
2. Supposons que $\mathcal{F}$ est de la forme $\neg\mathcal{F}_1$. Alors $\mathcal{I} \not\models \mathcal{F}$. Supposons que $\sigma(\mathcal{I}) \models \sigma(\mathcal{F}_1)$. Par hypothèse d'induction (puisque σ^{-1} est une permutation) cela signifie que $\mathcal{I} \models \mathcal{F}$, ce qui est impossible. Donc $\sigma(\mathcal{I}) \not\models \sigma(\mathcal{F}_1)$ donc $\sigma(\mathcal{I}) \models \sigma(\mathcal{F})$.
3. Si $\mathcal{F} \equiv \mathcal{F}_1 \vee \mathcal{F}_2$, alors $\mathcal{I}$ est soit un modèle de $\mathcal{F}_1$ soit un modèle de $\mathcal{F}_2$. Par hypothèse d'induction nous avons, soit $\sigma(\mathcal{I}) \models \mathcal{F}_1$, soit $\sigma(\mathcal{I}) \models \mathcal{F}_2$, donc $\sigma(\mathcal{I}) \models \sigma(\mathcal{F})$.
4. Si $\mathcal{F} \equiv \exists x.\mathcal{F}_1$. Alors $\mathcal{F}$ est équivalent à $\bigvee_{c \in \mathcal{D}om(x)} \mathcal{F}_1\{x \to c\}$. D'après le cas 3 ci-dessus, nous en déduisons que $\sigma(\mathcal{I}) \models \sigma(\bigvee_{c \in \mathcal{D}om(x)} \mathcal{F}_1\{x \to c\})$, i.e. $\sigma(\mathcal{I}) \models \bigvee_{c \in \mathcal{D}om(x)} \sigma(\mathcal{F}_1\{x \to c\})$, i.e. $\sigma(\mathcal{I}) \models \exists x.\sigma(\mathcal{F}_1)$. Par conséquent $\sigma(\mathcal{I}) \models \sigma(\mathcal{F})$.

C.Q.F.D.

LEMME 3.2.8. $\sigma(\mathcal{I}_{|\mathcal{R}}) = \sigma(\mathcal{I})_{|\sigma(\mathcal{R})}$.

PREUVE. $c = e \in \sigma(\mathcal{I}_{\mathcal{R}})$. si et seulement $\sigma^{-1}(c) = \sigma^{-1}(e) \in \mathcal{I}$ et $\sigma^{-1}(c) \in \mathcal{R}$ i.e. $c \in \sigma(\mathcal{R})$. Ceci est équivalent à $c = e \in \sigma(\mathcal{I})$ et $c \in \sigma(\mathcal{R})$ (par composition par σ), donc à $c = e \in \sigma(\mathcal{I})_{|\sigma(\mathcal{R})}$. C.Q.F.D.

Nous en déduisons le théorème suivant.

THÉORÈME 3.2.6. *Si $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$ alors $\sigma(\mathcal{R})$ est une réfutation de $\sigma(\mathcal{F})$ dans $\sigma(\mathcal{I})$.*

PREUVE. Soit $\mathcal{K}$ une interprétation telle que $\mathcal{K} =_{|\sigma(\mathcal{R})} \sigma(\mathcal{I})$. i.e $\sigma^{-1}(\mathcal{K}_{|\sigma(\mathcal{R})}) = \sigma^{-1}(\sigma(\mathcal{I})_{|\sigma(\mathcal{R})})$. D'après le lemme 3.2.8, nous en déduisons que $\sigma^{-1}(\mathcal{K})_{|\mathcal{R}} = \mathcal{I}_{|\mathcal{R}}$, donc $\sigma^{-1}(\mathcal{K}) \not\models \mathcal{F}$ (puisque $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$).

Par conséquent $\mathcal{K} \not\models \sigma(\mathcal{F})$ (par le lemme 3.2.7). C.Q.F.D.

La notion de symétrie entre interprétations est définie comme suit.

DÉFINITION 3.2.8. *Deux interprétations $\mathcal{I}$ et $\mathcal{J}$ sont dites* symétriques *s'il existe une permutation σ telle que $\sigma(\mathcal{I}) = \mathcal{J}$ et $\sigma(\mathcal{F}) = \mathcal{F}$.* ◇

Cette définition peut être modifiée afin de tirer parti de la notion de réfutation. Cela permet d'affaiblir les conditions entre les interprétations $\mathcal{I}$ et $\mathcal{J}$, tout en garantissant que $\mathcal{F}$ a même valeur de vérité dans ces deux interprétations.

DÉFINITION 3.2.9. *Deux interprétations $\mathcal{I}$ et $\mathcal{J}$ sont dites $\mathcal{R}$-*symétriques *s'il existe une permutation σ telle que $\sigma(\mathcal{I}) =_{|\mathcal{R}} \mathcal{J}$ et $\sigma(\mathcal{F}) = \mathcal{F}$.* ◇

Remarquons que la notion de $\mathcal{R}$-symétrie est *plus faible* que celle de symétrie, puisque nous nous restreignons à un sous-ensemble particulier de cellules.

REMARQUE. Les relations de *$\mathcal{R}$-symétrie* et *symétrie* sont bien entendus réflexives.

Le corollaire suivant est une conséquence immédiate du théorème 3.2.6.

COROLLAIRE 3.2.1. *Si $\mathcal{I}$ et $\mathcal{J}$ sont $\mathcal{R}$-symétriques (par rapport à une substitution σ) et si $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$, alors $\sigma(\mathcal{R})$ est une réfutation de $\mathcal{F}$ par rapport à $\mathcal{J}$.*

Utilisation pratique des symétries

Selon le corollaire 3.2.1, nous savons que si $\mathcal{I}$ et $\mathcal{J}$ sont $\mathcal{R}$-symétriques dans une permutation σ, et si $\mathcal{R}$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$, alors $\mathcal{J} \not\models \mathcal{F}$ et $\mathcal{R}\sigma$ est une réfutation de $\mathcal{F}$ dans $\mathcal{I}$.

Par souci d'efficacité, nous ne cherchons pas à identifier toutes les symétries possibles entre les interprétations, mais nous cherchons plutôt un critère simple (c'est-à-dire pouvant être vérifié avec un coût de calcul très faible) permettant de détecter qu'une interprétation donnée est symétrique à une des interprétations précédentes.

Pour cela, nous introduisons la notion de *symétrie directe*.

DÉFINITION 3.2.10. *Deux interprétations sont dites* directement $\mathcal{R}$-symétriques *si et seulement si les conditions suivantes sont satisfaites.*

- *Il existe au plus une cellule c dans $\mathcal{R}$ telle que $\mathcal{I}(c) \neq_{\mathcal{R}} \mathcal{J}(c)$.*
- *$\mathcal{I} =_{|\mathcal{R}} \sigma(\mathcal{J})$, où σ est la permutation définie par:*
  - *$\sigma(\mathcal{I}(c)) = \mathcal{J}(c)$.*
  - *$\sigma(\mathcal{J}(c)) = \mathcal{I}(c)$.*
  - *$\sigma(e) = e$, si $e \neq \mathcal{I}(c)$ et $e \neq \mathcal{J}(c)$.*
- *$\sigma(\mathcal{R}) = \mathcal{R}$.*
- *Ni $\mathcal{I}(c)$ ni $\mathcal{J}(c)$ n'apparaît dans $\mathcal{F}$.*

◇

De façon évidente, si $\mathcal{I}$ et $\mathcal{J}$ sont directement symétriques alors $\mathcal{I}$ et $\mathcal{J}$ sont symétriques. De plus, vérifier si deux interprétations sont directement $\mathcal{R}$-symétriques peut se faire très facilement dans notre algorithme *FMC* (en un temps linéaire par rapport à la taille de $\mathcal{R}$). Utiliser ce test très simple au lieu de la procédure **Evaluate** pour détecter qu'une interprétation donnée est $\mathcal{R}$-symétrique à la précédente peut étendre de façon significative les performances de notre approche. De façon évidente, en nous restreignant à des symétries directes, nous limitons le pouvoir de notre stratégie. Néanmoins, le test correspondant peut être réalisé très facilement, ce qui ne serait pas le cas en général.

Cette approche est une façon très simple et relativement restrictive de traiter les symétries. Ce procédé pourrait être amélioré dans le futur. En particulier, il existe des travaux sur les symétries en calcul propositionnel qui pourraient être combinés avec notre méthode. On trouve par exemple dans (BOY DE LA TOUR ET DEMRI, 1995) une analyse théorique de la complexité du calcul des symétries syntaxiques et dans (BOY DE LA TOUR, 1996) une méthode permettant d'*exploiter* des symétries *sémantiques* pour résoudre des problèmes propositionnels. En outre, il serait également intéressant de pouvoir définir un algorithme permettant d'énumérer les classes d'équivalence par la relation de symétrie au lieu d'énumérer l'ensemble des interprétations.

L'utilisation des symétries permet d'éliminer un grand nombre d'interprétations. Par exemple, considérons le cas d'une signature contenant un unique symbole de fonction d'arité 1 de profil $s \rightarrow s$. Si la taille du domaine de s est 5, il existe $5^5 = 3125$ interprétations possibles. En utilisant la stratégie proposée, on obtient uniquement 600 interprétations différentes, soit un gain de plus de 80 %. Cela reste cependant largement supérieur au nombre de classes d'équivalence par la relation de symétrie.

Si la taille du domaine est de 6, le gain est encore plus important : 4320 contre 46656 soit une amélioration de plus de 99 %. Notons que ces résultats peuvent encore être améliorés par l'utilisation des réfutations et réfutations couvrantes (le gain dépend alors étroitement de la formule considérée, donc est plus difficile à estimer).

3.2.6 Choix de l'ordre

Afin d'obtenir une procédure déterministe, il reste à spécifier l'ordre sur les cellules. Plusieurs ordres différents peuvent être utilisés pour un problème donné, et les performances de notre procédure peuvent dépendre de façon importante du choix de l'ordre. Trouver l'ordre le mieux adapté à un problème donné semble être une question très difficile. Il doit être choisi en général expérimentalement.

Dans les exemples du chapitre 4, l'ordre choisi est le suivant. Tout d'abord, les symboles fonctionnels sont ordonnés suivant leur arité (les symboles fonctionnels avec la plus grande arité sont prioritaires). Ensuite, les arguments des cellules sont ordonnés avec l'extension lexicographique de l'ordre sur le domaine.

Par exemple, si $\Sigma = \{f, g, a\}$, où f, g et a sont d'arité $2, 1, 0$ respectivement et si la taille du domaine est 2, alors l'ensemble des cellules sera ordonné comme suit.

$f(0,0) < f(0,1) < f(1,0) < f(1,1) < g(0) < g(1) < a$.

De façon évidente, d'autres ordres sont possibles, par exemple

$f(0,0) < g(0) < a < f(0,1) < f(1,0) < f(1,1) < g(1)$.

3.2.7 Choix de la réfutation

Soit $\mathcal{I}$ une interprétation et $\mathcal{F}$ une formule. Supposons que $\mathcal{I} \not\models \mathcal{F}$. Alors **Evaluate** retourne une réfutation de $\mathcal{F}$ dans $\mathcal{I}$. Mais, pour une interprétation donnée $\mathcal{I}$, plusieurs réfutations sont possibles, suivant l'ordre dans lequel les éléments du domaine sont considérés dans la procédure **Evaluate** et suivant l'ordre dans lequel les sous-formules de $\mathcal{F}$ sont évaluées.

Ces réfutations ne sont pas équivalentes, au sens où certaines peuvent permettre d'éliminer plus d'interprétations que d'autres. En particulier, il est clair qu'il est préférable de chercher une réfutation aussi petite que possible (suivant l'ordre d'inclusion entre ensembles). D'autre part, le nombre d'interprétations qu'une réfutation donnée $\mathcal{R}$ permet d'éliminer est lié au rang des cellules de $\mathcal{R}$ suivant l'ordre $<$. En effet, considérons deux réfutations $\mathcal{R}$ et $\mathcal{R}'$. Supposons que les cellules de $\mathcal{R}$ sont inférieures à celles de $\mathcal{R}'$. Alors on a (par définition de *succ*): $succ_{\mathcal{R}}(\mathcal{I}) > succ_{\mathcal{R}'}(\mathcal{I})$. Donc l'interprétation $\mathcal{I}'$ considérée après $\mathcal{I}$, si $\mathcal{R}$ est choisie, est supérieure (par rapport à l'ordre $<$ sur les interprétations) que celle considérée dans le cas où $\mathcal{R}'$ est retenue. Ainsi, $\mathcal{R}$ peut être considérée comme "meilleure" que $\mathcal{R}'$, puisque elle permet d'éliminer plus d'interprétations que $\mathcal{R}'$. Il est donc préférable en général d'évaluer en premier lieu les sous-formules contenant les cellules les plus petites. Ainsi, l'ordre dans lequel les sous-formules sont évaluées doit dépendre de l'ordre choisi sur les cellules.

3.3 Comparaison avec des travaux existants et discussion

Dans cette section, nous donnons un bref aperçu des méthodes utilisées en construction de modèles finis et nous donnons quelques éléments de comparaison avec notre approche.

Comme le mentionne l'introduction (voir section 1.3.2) les démonstrateurs existants pour la logique propositionnelle (DDPP, SATO, MACE...) ou les démonstrateurs par construction de modèles (SATCHMO, MGTP) peuvent être utilisés pour construire des modèles finis pour des formules du premier ordre. Néanmoins, la taille du problème propositionnel obtenu par traduction dépasse très rapidement les capacités de ces systèmes, en particulier si la formule contient des termes très profonds (ZHANG ET ZHANG, 1995).

Les trois programmes qui sont les plus proches de notre système sont FINDER, FALCON (une première version de SEM) et SEM.

FINDER (SLANEY, 1993) est un générateur de modèles fondé sur des techniques d'énumération et de retour-arrière. Au contraire de *FMC*, il n'utilise aucune technique particulière pour réduire l'espace de recherche, mais construit une base de données contenant les réfutations obtenues précédemment, ce qui permet d'éviter d'échouer deux fois pour la même raison.

FALCON (anciennement connu sous le nom de MOD/E) (ZHANG, 1993; ZHANG, 1994) et SEM (ZHANG ET ZHANG, 1995) sont également deux systèmes extrêmement efficaces pour la construction de modèles finis. Une heuristique très simple appelée "the Least Number Heuristic" (LNH) est utilisée pour réduire l'espace de recherche. LNH est fondée sur des idées similaires (mais moins générales) à celles de notre règle de détection des symétries, i.e. éviter de considérer deux fois de suite deux interprétations symétriques. Notre méthode est cependant plus générale, parce qu'elle ne dépend pas de l'ordre sur les éléments du domaine et que le critère utilisé pour rejeter les interprétations symétriques est plus faible que celui fourni par LNH. FALCON et SEM sont restreints à des formules en forme clausale. Ils utilisent des techniques d'implémentation efficaces pour la propagation des contraintes.

Les résultats expérimentaux présentés au chapitre 4 donnent des éléments de comparaison pratique entre ces systèmes et montrent l'intérêt de notre approche.

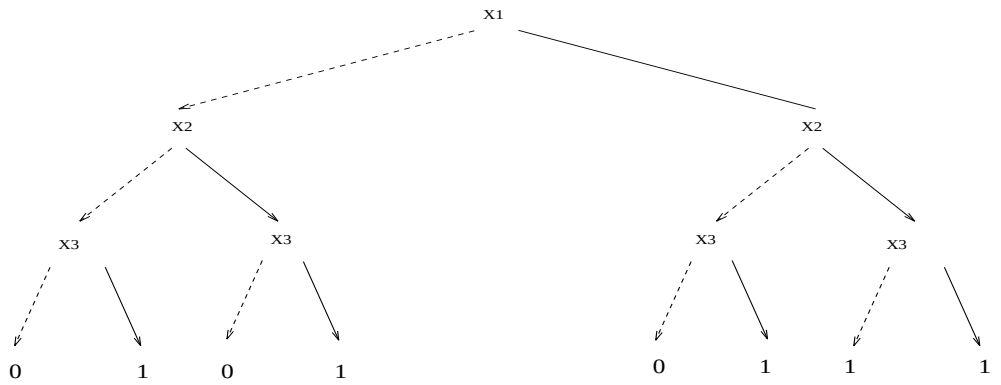
Le principal inconvénient de notre méthode est que ses performances dépendent fortement de l'ordre sur les cellules, qui doit être choisi avant le début de la recherche et ne peut changer par la suite. Afin de résoudre ce problème, il est possible d'utiliser certaines stratégies ou heuristiques afin de trouver un ordre adapté à un problème donné (par exemple il existe de nombreuses heuristiques utilisées pour résoudre les problèmes de satisfaction de contraintes qui pourraient certainement être combinées avec notre approche) et éventuellement pour modifier cet ordre pendant la recherche tout en préservant la complétude. Il serait également utile d'étudier des heuristiques permettant de guider la recherche d'une réfutation (dans la procédure **Evaluate**) i.e. pour guider le choix des sous-formules à évaluer en premier.

L'un des avantages les plus intéressants de notre méthode est qu'elle peut – au contraire des approches existantes – traiter des formules du premier ordre quelconques et ne nécessite *aucune transformation de la formule initiale*. C'est très important, car la transformation d'une formule en forme clausale peut augmenter de façon exponentielle la taille de la formule. Il est bien entendu possible d'utiliser le *renommage* pour obtenir une transformation quadratique mais c'est au prix de l'introduction de nouveaux symboles de prédicats, ce qui augmente le nombre de cellules à considérer.

De plus la généralité de notre approche doit également être soulignée. En effet, pour étendre notre méthode à d'autres logiques, il suffit de modifier la définition de la procédure **Evaluate** qui calcule la valeur de vérité d'une formule et la réfutation correspondante. *L'algorithme principal, ainsi que les stratégies proposées ne changent pas.*

Comparaison avec les BDD

Les BDD (**B**inary **D**ecision **D**iagrams) sont une méthode souvent utilisée pour répresenter des fonctions booléennes (voir par exemple (LEE, 1959; AKERS, 1978; BRYANT, 1992)). Elle consiste, étant donné un ensemble de variables propositionnelles $\{x_1, \ldots, x_n\}$, à construire un arbre sémantique dont chaque noeud est étiquetté par un x_i et possède 2 fils, correspondant respectivement aux valeurs 0 ou 1. Les feuilles de l'arbre sont étiquettés par la valeur de la fonction booléenne correspondant aux valeurs choisie pour les variables. Par exemple, il est ainsi possible de représenter la fonction $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee x_3$ par l'arbre suivant (les lignes pointillées correspondent à la valeur 0, les lignes pleines à la valeur 1).



Il est possible de réduire de façon importante la taille des BDD en transformant les arbres sémantiques en des graphes, obtenus en factorisant certains sous-arbres commun. On applique pour cela certaines règles de transformation visant à éliminer les noeuds (terminaux ou non-terminaux) identiques et à supprimer les tests inutiles. Cette technique fournit une méthode pour représenter de façon concise et efficace des fonctions booléennes. Si, de plus, l'ordre dans lequel les variables sont considérées lors de la construction de l'arbre sématique est fixé, on parle de *Ordered BDD* (OBDD).

Il est possible d'étendre la définition des BDD en considérant non plus des variables booléennes, mais des variables avec un nombre fini (plus grand que 2) de valeurs possibles. Le facteur de branchement pour un noeud x donné est la cardinalité de l'ensemble des valeurs possibles pour x. Cette technique permet d'énumérer de façon économique l'ensemble des modèles associés à une formule donnée. Notons que la technique utilisée par *FMC* ou par SEM pour énumérer les interprétations sur un domaine fini est très similaire à celle utilisé pour la construction des OBDD. En outre, l'utilisation des "refutations" semble proche de règles de transformation utilisees pour reduire la taille des BDD.

Les différences essentielles avec notre travail sont les suivantes.

- D'une part, les symétries traités par les BDD concernent des symétries sur la fonction booléenne (donc sur l'arbre sémantique obtenu). Ici la notion de symétrie est très différente, puisqu'il s'agit d'un isomorphisme sur les éléments du domaine. Le groupe des symétrie ne dépend pas de la formule considérée et est connu à priori. Dans le cas plus général des BDD, il n'y a pas de telles relations sur les variables considérées. Notons qu'en pratique, l'exploitation de cet isomorphisme (par exemple l'utilisation de la règle de détection des $\mathcal{R}$-symétries de *FMC* ou l'heuristique LNH de SEM) est d'une importance cruciale pour la réduction de l'espace de recherche.

- D'autre part, contrairement aux BDD, nous ne cherchons pas ici à représenter l'ensemble des modèles de la formule, mais simplement à trouver le ou les modèles possibles. Par conséquent, nous n'avons pas besoin de construire explicitement l'arbre sémantique associé à la formule. Cela représente une différence essentielle avec les BDD, dont le but est précisément de permettre une représentation concise et aisément manipulable de ces fonctions.

Chapitre 4

Résultats expérimentaux

4.1 FMC_{ATINF}: un système pour construire des modèles finis

FMC_{ATINF} est le logiciel qui implémente notre approche. Il s'agit d'un prototype dont le but est d'évaluer l'intérêt pratique de la méthode proposée. Il est écrit en C++ et fait partie du système RAMC$_{\text{ATINF}}$ (voir chapitre 15).

FMC_{ATINF} offre à l'utilisateur la possibilité de spécifier le problème à traiter comme un ensemble de formules du premier ordre (avec sorte). L'algorithme est contrôlé par un ensemble de *paramètres*, permettant par exemple de spécifier l'ordre sur les cellules, de choisir une stratégie particulière, ou de fixer des limites à la recherche (temps de calcul, nombre d'interprétations testées, nombre de modèles générés, etc.).

Nous donnons ci-dessous une liste d'exemples provenant de domaines très différents afin de montrer l'utilité de la méthode. Pour chacun d'entre eux, nous donnons une brève description du problème (ainsi que des références vers une description plus complète) et le temps de calcul obtenu avec FMC_{ATINF}.

REMARQUE. Presque tous ces exemples sont tirés de (ZHANG, 1994), où une description claire et complète de nombreux problèmes pour la construction de modèles finis est fournie (voir également (WOS ET AL., 1990; WOS, 1993b) pour plus de détails). Certains d'entre eux font également partie de la bibliothèque de problèmes TPTP (SUTTNER ET SUTCLIFFE, 1995).

4.2 Description des problèmes

Nous donnons ci-dessous une courte description des exemples considérés. Ils ont été choisis parmi les problèmes les plus utilisés comme "benchmark" pour les constructeurs de modèles finis.

Algèbre ternaire (TPTP BOO019-1)

Une *algèbre booléenne ternaire* est une structure satisfaisant les axiomes suivants.
(T_1) $\forall x.f(x,x,y) = x$
(T_2) $\forall x,y.f(g(x),x,y) = y$
(T_3) $\forall x,y.f(x,y,g(y)) = x$
(T_4) $\forall x,y,u,v.f(z,u,v) = f(f(x,y,z),u,f(x,y,v))$
(T_5) $\forall x,y.f(x,y,y) = y$

Il s'agit de montrer l'indépendance de l'axiome T_5 par rapport aux axiomes T_1-T_4. On utilise la technique standard, i.e. on tente de construire un modèle de T_1-T_4 qui ne soit *pas* un modèle de T_5, donc qui satisfait $\neg T_5 : \exists x,y.f(x,y,y) \neq y$.

Comme nous l'avons mentionné dans l'introduction, ce problème a été interactivement résolu pour la première fois par WOS et WINKER (WINKER, 1982) avec *l'aide* d'un démonstrateur par

résolution. Par la suite, des solutions purement automatiques ont été données (voir par exemple (Zhang, 1993; Bourely et al., 1994)).

Théorie des groupes.

La formule suivante est un axiome unique pour la théorie des groupes (Zhang, 1994; McCune, 1993) (c'est-à-dire que toute la théorie peut être dérivée de cet axiome):

(G_1) $f(x, i(f(y, f(f(f(z, i(z)), i(f(u, y))), x)))) = u$

Il s'agit de prouver que l'axiome suivant (qui est une instance de G_1, avec $x = u$) n'est pas un axiome unique de la théorie des groupes.

(G_2) $f(u, i(f(y, f(f(f(z, i(z)), i(f(u, y))), u)))) = u$

Pour cela, il suffit de construire un modèle de G_2 qui ne soit pas un modèle de G_1.

Le calcul d'équivalence

Les axiomes du calcul d'équivalence exprimés en logique du premier ordre (traduction de l'anglais "equivalence calculus") (Wos et al., 1990; Zhang, 1994) sont les suivants.

(E_1) $\forall x.P(e(x, x))$

(E_2) $\forall x, y.P(e(e(x, y), e(y, x)))$

(E_3) $\forall x, y, z.P(e(e(x, y), e(e(y, z), e(x, z))))$

La seule règle d'inférence utilisée est une règle appelée "condensed detachment", qui s'écrit comme suit.

(CD) $\forall x, y.P(e(x, y)) \wedge P(x) \rightarrow P(y)$

Le problème est de montrer que la formule suivante n'est pas un axiome unique pour le calcul d'équivalence.

(XBB) $P(e(x, e(e(e(x, e(y, z)), y), z)))$

Par cela, il suffit de construire un modèle de $(XBB), (CD)$ qui n'est pas un modèle de (E_1).

Arithmétique Heap

L'arithmétique Heap est décrite dans (Slaney, 1992). Les axiomes sont similaires à la théorie standard de l'arithmétique, sauf que l'on suppose qu'il existe un élément $Heap$ qui est son propre successeur i.e. tel que $succ(Heap) = Heap$. Plus précisément, l'arithmétique Heap est définie par les axiomes suivants.

$succ(Heap) = Heap$

$x < Heap \rightarrow x < succ(x)$

$x < y \rightarrow succ(x) = y \vee succ(x) < y$

$x + 0 = x$

$x + succ(y) = succ(x + y)$

$x \times 0 = 0$

$x \times succ(y) = (x * y) + x$

$x^0 = succ(0)$

$x^{succ(y)} = x^y \times x$

Les deux premiers axiomes définissent la relation de successeur *succ*, et les suivants définissent inductivement l'addition, la multiplication et l'exponentiation. Le problème est de trouver un modèle fini de cet ensemble d'axiomes.

La logique combinatoire

La logique combinatoire (pour la compréhension de ce travail, on pourra consulter (Wos, 1993b; Zhang, 1994)) est définie par un ensemble de *combinateurs* (notés en majuscules) définis par une série d'axiomes. Par exemple :

$$a(a(a(B,x),y),z) = a(x,a(y,z))$$
$$a(a(a(N_1,x),y),z) = a(a(a(x,y),y),z)$$
$$a(a(a(S,x),y),z) = a(a(x,z),a(y,z))$$
$$a(a(W,x),y) = a(a(x,y),y)$$
$$a(a(K,x),y) = x$$
$$a(a(L,x),y) = a(x,a(y,y))$$
$$a(a(a(Q,x),y),z) = a(y,a(x,z)).\ a(M,x) = a(x,x)$$

Un ensemble donné de combinateurs a la *propriété du point fixe faible (WFP)* si et seulement si pour tout x il existe un combinateur y tel que $y = a(x,y)$. Il possède la *propriété du point fixe fort (SFP)* si et seulement s'il existe un combinateur y tel que pour tout x, $a(y,x) = a(x,a(y,x))$.

Le problème est de déterminer quel(s) ensemble(s) de combinateurs possède(nt) ces propriétés. Afin de montrer qu'un ensemble *ne possède pas* les propriétés (WFP) ou (SFP) il suffit d'exhiber un modèle de ces combinateurs qui ne satisfait pas (WFP) ou (SFP). Ici, nous nous limiterons à trouver des modèles finis des ensembles $\{B, N_1, \neg(SFP)\}$ (logique combinatoire 1), $\{L, Q, \neg(SFP)\}$ (logique combinatoire 2) et $\{S, W, \neg(WFP)\}$ (logique combinatoire 3) qui sont les plus intéressants en pratique.

Problème de permutation

Le but est de trouver une fonction injective p de $[1..n]$ dans $[1..n]$ telle que pour tout $(i,j,k,l) \in [1..n]^4$ vérifiant $i < j < k < l$, on a

$$\neg(p(i) < p(j) < p(k) < p(l))$$

et

$$\neg(p(l) < p(k) < p(j) < p(i)).$$

Ce problème a une solution pour $n < 10$. Pour $n = 10$, il n'a plus de solution.

Problèmes de décidabilité (Church, 1956)

Dans (Church, 1956) Church donne une liste de formules satisfaisables ou insatisfaisables. Certaines de ces formules appartiennent à une classe décidable de la logique du premier ordre, la classe Am (Ackermann-Monadique). La question de la validité d'une formule de cette classe peut être décidée par exemple en utilisant une stratégie d'ordonnancement pour la procédure de résolution (voir (Fermüller et al., 1993) pour plus de détails). Certains de ces problèmes sont très simples, mais d'autres peuvent être considérés comme difficiles. Tous ces problèmes peuvent également être trouvés dans la bibliothèque de formules Tptp.

Ici nous considérons les problèmes SYN324-1 et SYN348-1 (selon le code utilisé par Tptp), qui sont représentatifs de l'ensemble des formules.

4.3 Exemple d'utilisation du logiciel

Nous donnons comme exemple d'utilisation de FMC_{ATINF} le problème de l'algèbre ternaire, sous la forme de sorties d'écran d'une session de travail avec le logiciel $\text{RAMC}_{\text{ATINF}}$(voir chapitre 15).

```
% Specify the order on the cells
order = 1.
symbol_precedence = greater_arity.
parameter_precedence = lex.
```

```
refutation_precedence = first.

% strategy

set(detect_symmetry).
set(cref).
set(prolog_style_variables).

list(tba).
f(X,X,Y) = X.
f(g(Y),Y,X) = X.
f(X,Y,g(Y)) = X.
f(U,V,f(X,Y,Z)) = f(f(U,V,X),Y,f(U,V,Z)).
(exists X,Y.f(X,Y,Y) != Y).
end.

set_size(3).
fmc(tba).
quit.
```

Le programme retourne le modèle suivant.

```
Model:

g(0)=1
g(1)=0
g(2)=0

f(0,0,0)=0
f(0,0,1)=0
f(0,0,2)=0
f(0,1,0)=0
f(0,1,1)=1
f(0,1,2)=2
f(0,2,0)=0
f(0,2,1)=1
f(0,2,2)=2
f(1,0,0)=0
f(1,0,1)=1
f(1,0,2)=2
f(1,1,0)=1
f(1,1,1)=1
f(1,1,2)=1
f(1,2,0)=1
f(1,2,1)=1
f(1,2,2)=1
f(2,0,0)=0
f(2,0,1)=2
f(2,0,2)=2
f(2,1,0)=2
f(2,1,1)=2
f(2,1,2)=2
f(2,2,0)=2
```

Problèmes	Taille	FMC_{ATINF}	SEM	FINDER
Algèbre ternaire	3	0.06	0.04	-[1]
Algèbre ternaire	4	0.3	0.1	-[1]
Algèbre ternaire	5	1.18	0.33	-[1]
Algèbre ternaire	6	4.22	0.78	-[1]
Théorie des groupes	2	0.02	0.02	0.03
Théorie des groupes	3	0.44	0.65	1.02
Théorie des groupes	4	13.11	-	> 1000
Arithmétique Heap	10	0.53	-	0.87 (114.82[2])
Arithmétique Heap	15	4.02	-	6.08
Le calcul d'équivalence (XBB)	4	0.78	-	2.82
Logique combinatoire 1	4	2.36	0.04	12.45
Logique combinatoire 1	5	24.5	0.04	267.55
Logique combinatoire 1	6	239.79	0.07	> 12000
Logique combinatoire 2	4	574.62	0.05	171.65
Logique combinatoire 3	5	0.04	0.03	0.08
Logique combinatoire 3	10	0.28	0.17	0.32
Logique combinatoire 3	20	1.74	1.39	2.62
Problème de Church SYN324-1	10	0.01	0.03	0.1
Problème de Church SYN324-1	20	0.02	0.11	0.35
Problème de Church SYN324-1	40	0.16	1.19	-
Problème de Church SYN348-1	6	0.06	0.12	0.12
Problème de Church SYN348-1	8	0.11	0.2	0.32
Problème de Church SYN348-1	12	0.25	0.41	1.57
Problème de Church SYN348-1	18	0.58	0.96	8.08
Permutation	5	0.01	-[3]	0.07
Permutation	6	0.03	-[3]	0.2
Permutation	7	0.18	-[3]	1.3
Permutation	8	4.38	-[3]	13.77
Permutation	9	54.12	-[3]	91.98

FIG. 4.1 - : *Résultats expérimentaux*

```
f(2,2,1)=2
f(2,2,2)=2

Run Time:     0.04
```

4.4 Expérimentations

Le tableau 4.1 résume les résultats obtenus avec notre système. Afin de faciliter les comparaisons avec les systèmes existants, nous donnons également le temps de calcul obtenu avec les deux constructeurs de modèles les plus efficaces (à notre connaissance) disponibles dans le domaine public : FINDER (FINDER.3.0) et SEM (SEM 1.0.6). Voir (SLANEY, 1993; ZHANG ET ZHANG, 1995), ou section 3.3. Les temps de calcul sont donnés sur une station SUN-4 en secondes. FMC_{ATINF} utilise l'algorithme FMC (fonction ϕ'_{cref}) et la stratégie de détection des symétries.

La "taille" est la cardinalité du domaine de l'interprétation. "-" signifie que le programme ne trouve pas de modèle dans un temps "raisonnable" (par rapport aux temps obtenus avec les autres systèmes) ou que le programme n'accepte pas ces exemples.

Notes (du tableau 4.1)

1. FINDER est actuellement restreint à des symboles de fonction d'arité inférieure à 2, donc ne peut traiter cet exemple.

2. Le premier temps est obtenu avec le choix de paramètre `pre-test = 4`. Voir (SLANEY, 1992) pour plus de détails.

3. Pour ces problèmes, SEM retourne "Exit due to the limit of memory". Cela peut s'expliquer par le fait qu'ils contiennent de nombreuses variables différentes, et donc que la taille de la formule obtenue par instanciation dépasse les capacités de l'ordinateur.

Comme on pouvait s'y attendre, aucun des trois programmes n'est meilleur que les autres partout. FINDER est plus rapide que FMC_{ATINF} sur l'un des problèmes de la logique combinatoire, tandis que notre système est équivalent ou meilleur sur les autres problèmes considérés. SEM dépasse de loin les deux autres systèmes sur certains problèmes, en particulier sur des formules constituées de conjonctions de littéraux équationnels (logique combinatoire et algèbre ternaire), mais semble beaucoup moins efficace sur d'autres problèmes (théorie des groupes, arithmétique Heap, etc.). Selon (ZHANG ET ZHANG, 1995), l'efficacité de SEM est essentiellement liée à la puissance de l'algorithme de propagation de contraintes.

Ainsi aucune des méthodes étudiées n'est uniformément supérieure aux autres. Certaines stratégies semblent donc mieux adaptées à certains cas particuliers. Malheureusement, il est très difficile d'identifier à priori la méthode la mieux adaptée à un problème donné. Des expérimentations supplémentaires, mais surtout des études théoriques plus poussées sont donc nécessaires afin d'identifier avec précision les avantages et les limites des approches existantes sur différentes classes de formules. Cette étude dépasse le cadre de cette thèse.

REMARQUE. Nous tenons à signaler que nous avons comparé notre prototype avec des systèmes ayant bénéficié d'un investissement de programmation très supérieur.

Un exemple très étudié : le "pigeonhole"

Il est intéressant d'observer le comportement de notre système sur un problème particulièrement bien connu et largement étudié : le principe dit du "pigeonhole".

Il s'agit de prouver qu'il n'existe aucune fonction injective d'un ensemble de cardinalité n vers un ensemble de cardinalité $n-1$. Ceci revient à chercher une relation in d'un ensemble S_1 vers un ensemble S_2 telle que

$$\forall x.\exists y.in(x,y) \wedge \forall x,y,z.(in(x,y) \wedge in(x,z)) \Rightarrow y = z$$

et $card(S_1) = n$, $card(S_2) = n-1$.

La figure 4.2 donne les résultats obtenus avec différentes valeurs de n.

Afin de comparer notre approche avec les méthodes existantes, il convient de rappeler que les systèmes de démonstration pour le calcul propositionnel ne disposant pas de mécanismes spécifiques permettant de traiter les symétries s'avèrent incapables de résoudre le problème en un temps raisonnable pour $n \geq 15$. Dans (BENHAMOU ET SAIS, 1994) une méthode pour exploiter les symétries dans les problèmes propositionnels est présentée. Elle permet de prouver l'insatisfaisabilité du "pigeonhole" pour des valeurs de $n \leq 30$. Nous comparons ci-dessous les résultats de FMC_{ATINF} avec ceux obtenus avec la méthode proposée par (BENHAMOU ET SAIS, 1994) (les temps de calcul sont tirés de (BENHAMOU ET SAIS, 1994), et le système est implémenté sur une SUN4/110). Nous donnons également les temps obtenus avec les systèmes FINDER et SEM.

REMARQUE. "-" signifie "non disponible" (pour $n \geq 40$, FINDER retourne "Search space too big to fit in vector length."; pour $n \geq 50$, SEM retourne "Exit due to the limit of memory").

n	FMC_{ATINF}	SEM	FINDER	BS
5	0	0.01	0.03	-
10	0.02	0.05	0.17	-
14	0.04	0.1	0.63	4.8
16	0.06	0.15	1.02	7.73
18	0.08	0.22	1.52	13.5
20	0.11	0.29	2.3	22.36
22	0.15	0.38	3.12	37.11
24	0.18	0.49	4.3	55.58
26	0.23	0.67	5.9	96.0
28	0.30	0.83	8.42	122.0
30	0.34	1	10.35	184.0
40	0.81	2.46	-	-
50	1.52	-	-	-
75	4.35	-	-	-
100	11.13	-	-	-

FIG. 4.2 - : *Le pigeonhole*

Le gain de temps de calcul obtenu avec FMC_{ATINF} est important, en particulier par rapport à BS. La raison est que FMC_{ATINF} tire parti du fait que le groupe de symétrie est connu avant le début de la recherche et ainsi n'a pas besoin d'être recalculé par la suite, et que la *représentation* du problème est très différente.

Partie II

Construction de modèles de Herbrand infinis

Chapitre 5

Représentation des interprétations

5.1 Objectif

Avant de chercher à construire des modèles de formules logiques, il est nécessaire d'en connaître le *langage de représentation*. Naturellement, il convient de rechercher des formalismes permettant de représenter de façon *finie*, efficace et naturelle ces interprétations. L'importance de ce problème est très clairement mise en évidence dans (Fermüller et Leitsch, 1996). Lorsque l'on cherche à représenter et à construire des modèles finis (comme au chapitre 3), il est possible en principe de représenter les interprétations par des tables donnant pour chaque élément du modèle $a_1, \ldots, a_n$ et pour chaque symbole de fonction ou de prédicat f la valeur de $f(a_1, \ldots, a_n)$. Cette technique permet de représenter facilement n'importe quel modèle *fini*, à condition que la taille du domaine considéré n'excède pas les capacités de la machine. En revanche, il est clair qu'il ne peut exister aucun formalisme permettant de représenter une interprétation quelconque sur un domaine *infini*, et en particulier qu'il ne peut exister aucun formalisme permettant de représenter n'importe quel modèle de Herbrand. En effet, l'ensemble des d'interprétations sur un domaine infini n'est pas dénombrable. Il est alors nécessaire de trouver des formalismes de représentation qui permettent à la fois de représenter une large classe d'interprétations et d'effectuer de façon efficace un certain nombre d'opérations de base sur les interprétations ainsi représentées.

Selon (Fermüller et Leitsch, 1996), un formalisme destiné à représenter des interprétations doit offrir les fonctionalités suivantes.

- **Problème d'évaluation atomique**. Il doit exister un algorithme "*efficace*" pour trouver la valeur de vérité d'un atome dans l'interprétation.

- **Problème d'équivalence**. Il doit exister un algorithme permettant de tester si deux représentations données sont équivalentes, c'est-à-dire représentent la même interprétation.

- **Problème d'évaluation clausale**. Il doit être possible de décider de la valeur de vérité d'une *clause* dans l'interprétation. Ceci est particulièrement important en Déduction Automatique où la règle de résolution est largement appliquée. En effet, les interprétations sont souvent utilisées pour guider la recherche d'une réfutation. On utilise pour cela des stratégies de restriction des règles d'inférence, appelées *stratégies sémantiques* qui, si la formule est en forme clausale, nécessitent justement de décider de la valeur de vérité d'une clause dans l'interprétation. Il est donc important que l'évaluation d'une clause soit décidable et aussi efficace que possible.

Dans une vision plus large de la déduction automatique, on doit également ajouter à ces conditions les suivantes.

- Il doit exister un algorithme permettant d'évaluer une *formule* quelconque dans l'interprétation.

– La représentation de l'interprétation doit être "compréhensible" par l'utilisateur.

Pour représenter un modèle infini, il est nécessaire de représenter :

– Le domaine de l'interprétation. Dans le cas où l'interprétation est définie sur l'univers de Herbrand, son domaine est implicite. Sinon, il doit être défini, par exemple, de façon inductive.

– L'ensemble des atomes fermés vrais dans l'interprétation.

Dans la suite de cette thèse, nous introduirons plusieurs formalismes pour représenter les modèles et nous les comparerons. Nous montrerons les avantages et les limites de chacun d'entre eux, essentiellement du point de vue de la construction de modèles. Notre étude nous conduira à la recherche d'un compromis entre le pouvoir d'expression des formalismes considérés et les limites théoriques sur la décidabilité et l'efficacité éventuelle des problèmes mentionnés ci-dessus. Là encore, nous ne rechercherons pas le formalisme "idéal", qui de toute façon n'existe probablement pas, mais nous nous efforcerons de définir plusieurs types de formalismes permettant chacun de résoudre certaines classes de problèmes particuliers et ayant chacun leurs avantages et leurs limites.

5.2 Représentation par des ensembles d'atomes

Dans la mesure où il n'existe que peu de recherches sur la construction de modèles infinis, l'étude de formalismes capables de représenter ces modèles n'a pas fait l'objet de beaucoup de travaux. A notre connaissance, les seuls travaux existants dans ce domaine sont ceux de (Fermüller et Leitsch, 1996) où les modèles (éventuellement infinis) sont représentés par des ensembles d'atomes et (Matzinger, 1997) où des grammaires régulières d'arbres sont utilisées (ce qui correspond à une généralisation des ensembles d'atomes linéaires). Evidemment, il existe beaucoup de travaux en théorie des langages qui fournissent des outils permettant de représenter des ensembles infinis de termes sur un alphabet donné (grammaires d'arbres, automates, etc.). Ils sont particulièrement utiles dans le cadre de la construction de modèles.

Une première idée pour représenter des modèles de Herbrand consiste à représenter explicitement l'ensemble des atomes vrais dans l'interprétation. C'est ce type de représentation, très simple, qui est utilisé par des démonstrateurs par construction de modèles (*model generation theorem prover*) tels que Satchmo (Manthey et Bry, 1988) ou Mgtp (Fujita et Hasegawa, 1991). De façon évidente, cela n'est possible que si cet ensemble est fini. Si l'ensemble d'atomes est infini, il est alors possible d'étendre le formalisme en autorisant la présence de variables dans les ensembles d'atomes positifs. Il devient alors possible de représenter l'ensemble infini $\{P(a), P(f(a)), \ldots\}$, par l'ensemble d'atomes $\{P(x)\}$. C'est ce type de représentation — appelé *représentation atomique* — qui est utilisé dans (Fermüller et Leitsch, 1996).

Définition 5.2.1. *Soit un ensemble satisfaisable d'atomes E. Le* modèle de Herbrand $\mathcal{M}$ associé à E *est l'ensemble des atomes positifs fermés $P(\overline{t})$ tels qu'il existe $P(\overline{s}) \in E$ et une substitution σ fermée tels que $P(\overline{s})\sigma = P(\overline{t})$.* ◇

L'avantage principal des représentations atomiques est qu'il est très facile de trouver la valeur de vérité d'un atome : il suffit d'utiliser un algorithme de filtrage (linéaire par rapport à la taille du terme considéré). D'autre part, Fermüller et Leitsch montrent (Fermüller et Leitsch, 1996) que le problème de savoir si deux représentations sont équivalentes est décidable. Ils fournissent en outre un algorithme permettant d'évaluer une clause dans les interprétations ainsi représentées.

Dans (Fermüller et Leitsch, 1996), les représentations atomiques sont étendues au cas avec égalité. Une *représentation atomique équationnelle* est un ensemble satisfaisable d'atomes E tel que tout littéral équationnel de E est fermé. L'interprétation dénotée par E est définie d'une part par la théorie $\mathcal{T}$ constituée par tous les littéraux équationnels de E et d'autre part,

par l'ensemble des atomes fermés L unifiables avec un atome de E modulo $\mathcal{T}$. Remarquons que le problème de la validité d'une formule équationnelle du premier ordre dans $\mathcal{T}$ est décidable, puisque $\mathcal{T}$ ne contient que les atomes fermés (Comon, 1993). Notons que cette technique peut s'étendre facilement à toute théorie décidable.

5.3 Représentation par contraintes

Dans ce travail, nous représentons les interprétations de Herbrand par des *contraintes*. L'utilisation de contraintes peut être vue comme une généralisation des représentations atomiques. Elle consiste à représenter l'ensemble des atomes $P(\overline{s})$ vrais dans le modèle par une *contrainte*, i.e. les images des symboles relationnels par l'interprétation sont définies par une formule équationnelle (éventuellement interprétée dans une théorie $\mathcal{T}$).

Précisons le principe général de la méthode. Nous considérons une théorie $\mathcal{T}$. Soit P un symbole de prédicat. L'extension de P, c'est-à-dire l'ensemble de n-uplets $x_1, \ldots, x_n$ tels que $P(x_1, \ldots, x_n)$ est vrai, est représentée par une formule $\mathcal{F}_P$ du premier ordre interprétée dans la théorie $\mathcal{T}$ contenant n variables libres $x_1, \ldots, x_n$ et telle que

$$\mathcal{F}_P \text{ est vraie dans } \mathcal{T} \text{ si et seulement si } P(x_1, \ldots, x_n) \text{ est vrai.}$$

L'utilisation de représentations par contraintes permet de résoudre facilement la plupart des problèmes présentés en introduction. En effet, l'une des propriétés de ce type de représentation est que l'ensemble des solutions d'une formule logique du premier ordre dans une interprétation $\mathcal{I}$ représentée par contraintes peut lui-même être dénoté par une contrainte. De nombreux problèmes concernant les interprétations (évaluation, équivalence etc.) se ramènent ainsi au problème de décider de la satisfaisabilité d'une contrainte. Il suffit donc de disposer d'un algorithme (si possible efficace) pour résoudre les contraintes dans la théorie $\mathcal{T}$, afin de pouvoir résoudre la plupart des problèmes mentionnés ci-dessus. L'un des avantages de cette approche est sa généralité: elle permet d'étendre facilement la méthode en changeant la théorie $\mathcal{T}$.

La représentation par contraintes fournit donc un cadre naturel particulièrement bien adapté à la représentation des interprétations. Dans la section suivante, nous introduisons la notion d'*interprétation partielle* et nous montrons comment représenter de telles interprétations par des contraintes équationnelles interprétées dans l'algèbre des termes.

5.4 eq-interprétations

5.4.1 Interprétation partielle

Dans (Hsiang et Rusinowitch, 1986; Caferra et Zabel, 1992), la notion d'*interprétation partielle de Herbrand* est introduite. Informellement, il s'agit d'interprétations $\mathcal{I}$ dans lesquelles les images des symboles relationnels ne sont pas entièrement spécifiées. Ainsi une interprétation partielle n'affecte pas une valeur vraie ou fausse à toutes les formules, mais seulement à une partie de celles-ci. Par exemple, si on considère la formule

$$\mathcal{F} = \forall x \exists y.(P(x,y) \vee Q(x,y))$$

une interprétation partielle validant $\mathcal{F}$ est, par exemple, l'interprétation définie par $\forall x, y.P(x,y) = \mathit{true}$. L'interprétation du prédicat Q n'est pas spécifiée, ainsi la formule $\exists x, y.Q(x,y)$ n'est ni vraie ni fausse dans l'interprétation. Une interprétation partielle peut également être considérée comme une interprétation dans une logique à trois valeurs de vérités: *true*, *false*, et *undefined* (non définie).

Nous étendrons ici la définition de (Caferra et Zabel, 1992) à des interprétations quelconques (non nécessairement définies sur l'univers de Herbrand). Nous donnerons également une définition de la relation "$\models$" à la fois plus générale et correspondant mieux à l'idée intuitive

de la notion de validité. En effet, la définition donnée dans (Caferra et Zabel, 1992) pose plusieurs problèmes : elle est limitée à des clauses (ou ensemble de clauses) et n'attribue pas la valeur vraie à des clauses de la forme $A \vee \neg A$ par exemple. En particulier, ce dernier point rend la règle d'élimination des tautologies incorrecte (au sens habituel du terme, i.e. préservation de la valeur de vérité dans toute interprétation) puisque $A \vee \neg A$ n'est pas valide dans l'interprétation partielle vide (i.e. attribuant à tous les littéraux la valeur *undefined*) alors que l'ensemble vide est valide dans toute interprétation.

Définition 5.4.1.

Une interprétation partielle *est un couple* $\langle (D_s)_{s \in \mathcal{S}}, \mathcal{I} \rangle$ [1] *où* D_s *est un ensemble non vide (domaine de la sorte* s*) et où* $\mathcal{I}$ *est une fonction qui, à tout symbole fonctionnel* $f : s_1, \ldots, s_n \to s$ *de* Σ*, associe une fonction* **partielle** $\mathcal{I}(f)$ *de* $D_{s_1} \times \ldots \times D_{s_n}$ *dans* D_s*, et à tout symbole prédicat de* Ω $P : s_1, \ldots, s_n$ *associe* ***deux sous-ensembles*** *de* $D_{s_1} \times \ldots \times D_{s_n}$*,* $\mathcal{I}(P)^+$ *et* $\mathcal{I}(P)^-$ *tels que* $\mathcal{I}(P)^+ \cap \mathcal{I}(P)^- = \emptyset$. ◇

Il est intéressant de considérer quelques cas particuliers.

Définition 5.4.2.

Une interprétation partielle $\langle (D_s)_{\mathcal{S}}, \mathcal{I} \rangle$ *est dite* totale *si et seulement si pour tout* $f \in \Sigma$*,* $\mathcal{I}(f)$ *est totale et pour tout* $P : s_1, \ldots, s_n$ *de* Ω*, on a* $\mathcal{I}(P)^+ \cup \mathcal{I}(P)^- = D_{s_1} \times \ldots \times D_{s_n}$. ◇

On retrouve dans ce cas la notion classique d'interprétation.

Définition 5.4.3. *On appelle interprétation* partielle de Herbrand, *une interprétation* $\langle (D_s)_{\mathcal{S}}, \mathcal{I} \rangle$ *telle que* $D_s = \tau_s(\Sigma)$ *et* $\mathcal{I}(f(\overline{t})) = f(\mathcal{I}(\overline{t}))$ *(*$\mathcal{I}(f)$ *est totale sur* $\mathcal{D}_s$*).* ◇

Remarque. Dans la suite, toutes les interprétations considérées seront des interprétations de Herbrand.

Une interprétation partielle de Herbrand $\mathcal{I}$ peut également être considérée comme un ensemble de littéraux fermés vérifiant la condition

$$\forall x . x \in \mathcal{I} \Rightarrow \neg x \notin \mathcal{I}.$$

On a alors
$\mathcal{I}(b) = \mathit{true}$ si $b \in \mathcal{I}$
$\mathcal{I}(b) = \mathit{false}$ si $\neg b \in \mathcal{I}$
$\mathcal{I}(b) = \mathit{undefined}$ sinon

Nous noterons $\mathcal{I}^+$ l'ensemble des littéraux positifs fermés de $\mathcal{I}$ et $\mathcal{I}^-$ l'ensemble des littéraux négatifs fermés de $\mathcal{I}$.

Définition 5.4.4. *L'*interprétation partielle vide de Herbrand *(notée* $\emptyset$*) est l'interprétation de Herbrand définie par*

$$\forall P \in \Omega \, \mathcal{I}^+(P) = \mathcal{I}^-(P) = \emptyset.$$

◇

Nous pouvons introduire un ordre partiel sur les interprétations partielles de la façon suivante.

Définition 5.4.5.

Une interprétation $\mathcal{I}$ est inclue *dans une interprétation partielle de même domaine* $\mathcal{J}$ *(cela est noté* $\mathcal{I} \subseteq \mathcal{J}$*) si et seulement si*

$$\forall f \in \Sigma . \mathcal{I}(f)(x_1, \ldots, x_n) = a \Rightarrow \mathcal{J}(x_1, \ldots, x_n) = a$$

1. Pour simplifier, une interprétation partielle $\langle (D_s)_{s \in \mathcal{S}}, \mathcal{I} \rangle$ sera souvent notée également $\mathcal{I}$ si les domaines D_s sont implicites.

$$\forall P \in \Omega.(\mathcal{I}(P)^+ \subseteq \mathcal{J}(P)^+ \wedge \mathcal{I}(P)^- \subseteq \mathcal{J}(P)^-).$$

$\mathcal{I} \subset \mathcal{J}$ si et seulement si $\mathcal{I} \subseteq \mathcal{J}$ et $\mathcal{I} \neq \mathcal{J}$. ◇

DÉFINITION 5.4.6.

Une interprétation partielle $\mathcal{I}$ valide *une formule $\mathcal{F}$ si et seulement si toute interprétation totale contenant $\mathcal{I}$ valide $\mathcal{F}$.*

◇

Cette définition diffère sensiblement de celle présentée dans (CAFERRA ET ZABEL, 1992) qui ne considérait que des interprétations partielles sur l'univers de Herbrand et ne définissait la notion de validité que pour des ensembles de clauses (avec contraintes). Enfin, la définition de (CAFERRA ET ZABEL, 1992) n'attribuait pas la valeur de vérité **vrai** à toutes les formules valides dans toute interprétation totale. Par exemple, la tautologie $P \vee \neg P$ n'était pas **vraie** dans toute interprétation.

Notation 5.4.1 *Soit $\mathcal{I}$ une interprétation partielle et $\mathcal{F}$ une formule. $\mathcal{I}(\mathcal{F})$ est définie par*

- *$\mathcal{I}(\mathcal{F}) = false$, si $\mathcal{I} \models \neg\mathcal{F}$.*
- *$\mathcal{I}(\mathcal{F}) = true$, si $\mathcal{I} \models \mathcal{F}$.*
- *$\mathcal{I}(\mathcal{F}) = undefined$, sinon.*

5.4.2 Représentation des interprétations partielles

Le formalisme de représentation utilisé dans la deuxième partie de cette thèse est de considérer des interprétations où l'image des symboles relationnels est définie par une *formule équationnelle* interprétée dans l'algèbre des termes. C'est un cas particulier du principe de représentation décrit à la section 5.3, avec $\mathcal{T} = \emptyset$.

DÉFINITION 5.4.7. *Un sous-ensemble E de $\tau(\Sigma)^n$ est appelé un eq-ensemble si et seulement si il existe une formule équationnelle $\mathfrak{F}(E)$ contenant n variables libres $x_1, \ldots, x_n$ vérifiant*

$$(t_1, \ldots, t_n) \in E - \{x_i \rightarrow t_i / 1 \leq i \leq n\} \in \mathcal{S}(\mathfrak{F}(E)).$$

◇

DÉFINITION 5.4.8. *Une interprétation partielle de Herbrand $\mathcal{I}$ est appelée* une eq–interprétation *("eq" pour "*partial *interpretation definable by* **eq**uational problems"*) si et seulement si pour tout prédicat n-aire P, les ensembles $\mathcal{I}(P)^+$ et $\mathcal{I}(P)^-$ sont des eq-ensembles.* ◇

5.5 Vérification de modèle

Beaucoup de problèmes sur les eq-interprétations se ramènent de façon immédiate à la résolution des formules équationnelles. Par exemple, le problème de *l'équivalence* qui consiste à décider si deux représentations données $\mathcal{A}$ et $\mathcal{B}$ définissent la même interprétation se ramène au problème de décider si pour tout $P \in \Omega$, les eq-ensembles correspondants $\mathcal{A}(P)^+$ et $\mathcal{B}(P)^+$ (resp. $\mathcal{A}(P)^-$ et $\mathcal{B}(P)^-$) sont égaux, c'est-à-dire si les formules correspondantes sont équivalentes. L'équivalence de $\mathcal{A}$ et $\mathcal{B}$ se ramène donc au problème de l'équivalence de deux formules équationnelles qui peut être décidé par l'algorithme de (COMON ET LESCANNE, 1989). Cette section montre comment résoudre, en utilisant cette technique, le problème de l'évaluation d'une formule du premier ordre dans une eq-interprétation.

Le problème de savoir si une eq-interprétation de Herbrand valide une formule du premier ordre $\mathcal{F}$ est *indécidable*. En effet, par définition, $\mathcal{I} \models \mathcal{F}$ si et seulement si pour toute extension totale $\mathcal{J}$ de $\mathcal{I}$, $\mathcal{J} \models \mathcal{F}$. Or si $\mathcal{I} = \emptyset$, toute interprétation totale est une extension de $\mathcal{I}$, donc $\emptyset \models \mathcal{F}$? est équivalent à $\mathcal{F} \equiv \top$? qui est un problème indécidable.

En revanche, il devient décidable si on ajoute l'une des conditions suivantes.

– $\mathcal{I}$ est totale ;

– ou $\mathcal{F}$ ne contient pas de quantificateur existentiel (voir section 6.5).

DÉFINITION 5.5.1. *Soit $\mathcal{I}$ une interprétation. Soit $\mathcal{F}$ une formule. L'ensemble des* solutions *de $\mathcal{F}$ (noté $\mathcal{S}_{\mathcal{I}}(\mathcal{F})$) dans $\mathcal{I}$ est l'ensemble des substitutions fermées σ telles que $\mathcal{I} \models \mathcal{F}\sigma$.* ◇

Nous proposons ci-dessous un algorithme permettant de calculer certaines des solutions d'une formule $\mathcal{F}$ dans une eq-interprétation $\mathcal{I}$. L'algorithme est *incomplet* c'est-à-dire qu'il ne calcule pas toutes les solutions. Il devient complet si l'interprétation $\mathcal{I}$ est totale. Le principe de la méthode proposée est de traduire la formule $\mathcal{F}$ en une formule équationnelle équivalente (c'est-à-dire possédant le même ensemble de solutions).

Plus précisément, la définition 5.5.2 ci-dessous introduit deux formules $\phi^+_{\mathcal{M}}(\mathcal{F})$ et $\phi^-_{\mathcal{M}}(\mathcal{F})$ exprimant des conditions *suffisantes – mais non nécessaires* – pour que $\mathcal{F}$ soit vraie (resp. fausse) dans l'interprétation $\mathcal{M}$.

DÉFINITION 5.5.2. *Soit $\mathcal{M}$ une eq-interprétation et $\mathcal{F}$ une formule. Soit $\phi^+_{\mathcal{M}}(\mathcal{F})$ et $\phi^-_{\mathcal{M}}(\mathcal{F})$ les formules équationnelles définies comme suit.*

– *Si $\mathcal{F}$ est de la forme $P(\overline{t})$, alors*

$$\phi^+_{\mathcal{M}}(\mathcal{F}) = \exists \overline{x}.\overline{t} = \overline{x} \wedge \mathfrak{F}(\mathcal{I}(P)^+)$$

$$\phi^-_{\mathcal{M}}(\mathcal{F}) = \exists \overline{x}.\overline{t} = \overline{x} \wedge \mathfrak{F}(\mathcal{I}(P)^-)$$

(où $\overline{x}$ sont les variables correspondant à $\mathfrak{F}(\mathcal{I}(P)^+)$ et $\mathfrak{F}(\mathcal{I}(P)^-)$).

– *Si $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$:*

$$\phi^+_{\mathcal{M}}(\mathcal{F}) = \phi^+_{\mathcal{M}}(\mathcal{F}_1) \vee \phi^+_{\mathcal{M}}(\mathcal{F}_2)$$

$$\phi^-_{\mathcal{M}}(\mathcal{F}) = \phi^-_{\mathcal{M}}(\mathcal{F}_1) \wedge \phi^-_{\mathcal{M}}(\mathcal{F}_2)$$

– *Si $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$:*

$$\phi^+_{\mathcal{M}}(\mathcal{F}) = \phi^+_{\mathcal{M}}(\mathcal{F}_1) \wedge \phi^+_{\mathcal{M}}(\mathcal{F}_2)$$

$$\phi^-_{\mathcal{M}}(\mathcal{F}) = \phi^-_{\mathcal{M}}(\mathcal{F}_1) \vee \phi^-_{\mathcal{M}}(\mathcal{F}_2)$$

– *Si $\mathcal{F} = \exists x.\mathcal{F}_1$:*

$$\phi^+_{\mathcal{M}}(\mathcal{F}) = \exists x.\phi^+_{\mathcal{M}}(\mathcal{F}_1)$$

$$\phi^-_{\mathcal{M}}(\mathcal{F}) = \forall x.\phi^-_{\mathcal{M}}(\mathcal{F}_1)$$

– *Si $\mathcal{F} = \forall x.\mathcal{F}_1$:*

$$\phi^+_{\mathcal{M}}(\mathcal{F}) = \forall x.\phi^+_{\mathcal{M}}(\mathcal{F}_1)$$

$$\phi^-_{\mathcal{M}}(\mathcal{F}) = \exists x.\phi^-_{\mathcal{M}}(\mathcal{F}_1)$$

– *Si $\mathcal{F} = \neg\mathcal{F}_1$:*

$$\phi^+_{\mathcal{M}}(\mathcal{F}) = \phi^-_{\mathcal{M}}(\mathcal{F}_1)$$

$$\phi^-_{\mathcal{M}}(\mathcal{F}) = \phi^+_{\mathcal{M}}(\mathcal{F}_1)$$

◇

THÉORÈME 5.5.1. *Soit $\mathcal{M}$ une eq-interprétation et $\mathcal{F}$ une formule. Soit σ une substitution. Si $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}))$, alors $\mathcal{M} \models \mathcal{F}\sigma$. Si $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}))$, alors $\mathcal{M} \models \neg\mathcal{F}\sigma$.*

PREUVE. Par induction structurelle sur l'ensemble des formules $\mathcal{F}\sigma$.

- Soit $\mathcal{F} = P(\overline{t})$. Supposons que $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}))$. Alors, par définition, $\sigma \in \mathcal{S}(\exists \overline{x}.\overline{t} = \overline{s} \wedge \mathcal{X})$ donc $\{\overline{x} \rightarrow \overline{t}\sigma\}$ est une solution de $\mathfrak{F}(\mathcal{I}(P)^+)$, d'où $P(\overline{t})\sigma$ appartient à $\mathcal{I}(P)^+$. Donc $\mathcal{M} \models \mathcal{F}\sigma$. La preuve est similaire pour $\mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}))$
- $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$. Si $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}))$, alors soit $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}_1))$ soit $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}_2))$. Par hypothèse d'induction soit $\mathcal{M} \models \mathcal{F}_1\sigma$ soit $\mathcal{M} \models \mathcal{F}_2\sigma$. Donc $\mathcal{M} \models \mathcal{F}_1\sigma \vee \mathcal{F}_2\sigma$, i.e. $\mathcal{M} \models \mathcal{F}\sigma$. Si $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}))$, alors $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}_1))$ et $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}_2))$. Par hypothèse d'induction, $\mathcal{M} \models \neg\mathcal{F}_1\sigma$ et $\mathcal{M} \models \neg\mathcal{F}_2\sigma$. Donc $\mathcal{M} \models \neg\mathcal{F}_1\sigma \wedge \neg\mathcal{F}_2\sigma$, i.e. $\mathcal{M} \models \neg(\mathcal{F}_1 \vee \mathcal{F}_2)\sigma = \neg\mathcal{F}\sigma$.
- La preuve est similaire si $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$.
- Si $\mathcal{F} = \neg\mathcal{F}_1$. Si $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}))$, alors $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}_1))$, donc $\mathcal{M} \models \neg\mathcal{F}_1\sigma$, i.e. $\mathcal{M} \models \mathcal{F}\sigma$. Si $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}))$, alors $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}_1))$, donc $\mathcal{M} \models \mathcal{F}_1\sigma$, i.e. $\mathcal{M} \models \neg\neg\mathcal{F}_1\sigma$.
- Si $\mathcal{F} = \exists x.\mathcal{F}_1$. Si $\sigma \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}))$, alors $\sigma \in \mathcal{S}(\exists x.\phi^+(\mathcal{F}_1))$, i.e. il existe un terme fermé t tel que $\sigma \cup \{c \rightarrow t\} \in \mathcal{S}(\phi^+_{\mathcal{M}}(\mathcal{F}_1))$. Donc $\mathcal{M} \models \mathcal{F}_1\{x \rightarrow t\}\sigma$. Donc $\mathcal{M} \models (\exists x.\mathcal{F}_1)\sigma$. Si $\sigma \in \mathcal{S}(\phi^-_{\mathcal{M}}(\mathcal{F}))$, alors $\sigma \in \mathcal{S}(\forall x.\phi^-(\mathcal{F}_1))$, i.e. pour tout terme fermé t, $\sigma\{x \rightarrow t\} \in \mathcal{S}(\phi^+(\mathcal{F}_1))$. Donc $\forall t.\mathcal{M} \models \neg\mathcal{F}_1\{x \rightarrow t\}\sigma$. Et $\mathcal{M} \models (\forall x.\neg\mathcal{F}_1)\sigma$. i.e. $\mathcal{M} \models \neg\mathcal{F}\sigma$.
- La preuve est similaire si $\mathcal{F} = \forall x.\mathcal{F}_1$.

C.Q.F.D.

Dans le cas où $\mathcal{I}$ est totale, la réciproque est également vraie. C'est une conséquence du théorème 5.5.2.

THÉORÈME 5.5.2. *Si $\mathcal{I}$ est totale alors $\phi^+_{\mathcal{I}}(\mathcal{F}) \vee \phi^-(\mathcal{F}) \equiv \top$.*

PREUVE. Par induction structurelle sur $\mathcal{F}$.

- Formule atomique. Par définition $\sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$ si et seulement si $\overline{t}\sigma \in \mathcal{I}^+(P)$. Puisque $\mathcal{I}$ est totale, ceci est équivalent à $\overline{t}\sigma \notin \mathcal{I}^-(P)$, donc à $\sigma \notin \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}))$.
- Si $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$, et si $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$, alors $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}_1))$ et $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}_1))$. Par hypothèse d'induction, ceci implique $\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}_1))$ et $\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}_1))$, c'est-à-dire $\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}))$.
- La preuve est similaire si $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$
- Si $\mathcal{F} = \neg\mathcal{F}'$, alors $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$ implique $\sigma \notin \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}_1))$ donc (par hypothèse d'induction) $\sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}_1))$ d'où $\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}))$.
- Si $\mathcal{F} = \exists x.\mathcal{F}'$. Si $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$ alors par définition pour tout terme de $\tau_s(\Sigma)$, (où $x \in \mathcal{V}_s$), on a $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}'\{x \rightarrow t\}))$. Donc, par hypothèse d'induction $\forall t \in \tau_s(\Sigma).\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}\{x \rightarrow t\}))$, i.e. $\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}))$
- La preuve est similaire pour $\mathcal{F} = \forall x.\mathcal{F}'$.

C.Q.F.D.

COROLLAIRE 5.5.1. *Soit $\mathcal{I}$ une eq-interprétation totale et $\mathcal{F}$ une formule. Soit σ une substitution de $\mathcal{V}ar(\mathcal{F})$.*

$$\mathcal{I} \models \mathcal{F}\sigma - \sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$$
$$\mathcal{I} \not\models \mathcal{F}\sigma - \sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}))$$

PREUVE. Il suffit de montrer que $\mathcal{I} \models \mathcal{F}\sigma$ implique que $\sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$ (la réciproque étant établie par le théorème 5.5.1). Supposons que $\mathcal{I} \models \mathcal{F}\sigma$ et que $\sigma \notin \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$. Alors, d'après le théorème 5.5.2, on a $\sigma \in \mathcal{S}(\phi^-_{\mathcal{I}}(\mathcal{F}))$. D'où $\mathcal{I} \models \neg\mathcal{F}\sigma$ d'après le théorème 5.5.1. Donc $\mathcal{I}$ valide à la fois $\mathcal{F}\sigma$ et $\neg\mathcal{F}\sigma$, ce qui est impossible. C.Q.F.D.

La recherche des solutions de la formule $\mathcal{F}$ est donc ramenée à la résolution d'un problème équationnel qui est décidable d'après le théorème 2.3.2.

Ces résultats établissent la décidabilité du problème d'évaluation dans une eq-interprétation totale (c'est-à-dire du problème de trouver la valeur de vérité d'une formule dans une interprétation donnée) qui est d'une importance cruciale pour de nombreuses applications. Ils étendent ainsi les résultats de (FERMÜLLER ET LEITSCH, 1996) dans deux directions.

1. D'une part, la classe des interprétations représentables est plus large que celle de (FERMÜLLER ET LEITSCH, 1996).
2. D'autre part, les problèmes que nous considérons sont plus généraux (évaluation d'une formule du premier ordre au lieu d'un ensemble de clauses).

Nous verrons par la suite l'utilité pratique des formules ϕ^+ et ϕ^- que nous venons de définir. Elles sont au cœur de l'extension de la résolution sémantique proposée au chapitre 7, de l'extension de la méthode des tableaux sémantiques (chapitre 9), etc.

5.6 Simplification d'une formule dans un contexte

Une interprétation partielle peut être utilisée pour simplifier une formule.

THÉORÈME 5.6.1. *[Simplification d'une formule] Soit A une formule, $\mathcal{I}$ une interprétation partielle de Herbrand. Soit $Simplify_{\mathcal{I}}(A) = (\neg\phi^-_{\mathcal{I}}(A)) \wedge (\phi^+_{\mathcal{I}}(A) \vee A)$.*

$$A \equiv_{\mathcal{I}} Simplify_{\mathcal{I}}(A))$$

PREUVE. Soit $\mathcal{J}$ une interprétation totale contenant $\mathcal{I}$. Soit une solution σ de A dans $\mathcal{J}$. Par définition, $\sigma \notin \mathcal{S}(\phi^-_{\mathcal{I}}(A))$ et $\sigma \in \mathcal{S}_{\mathcal{J}}(A)$ donc $\sigma \in \mathcal{S}_{\mathcal{J}}(\text{Simplify}_{\mathcal{I}}(A))$. Réciproquement soit $\sigma \in \mathcal{S}_{\mathcal{J}}(\text{Simplify}_{\mathcal{I}}(A))$. On a soit $\sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(A))$ donc $\sigma \in \mathcal{S}_{\mathcal{J}}(A)$, soit $\sigma \in \mathcal{S}_{\mathcal{J}}(A)$. C.Q.F.D.

DÉFINITION 5.6.1. *Soit $\mathcal{F}$ une formule, $\mathcal{I}$ une eq-interprétation partielle de Herbrand. On note $Normalize_{\mathcal{I}}(\mathcal{F})$ la formule obtenue en remplaçant tous les littéraux A de la formule $\mathcal{F}$ par*

$$\neg\phi^-_{\mathcal{I}}(A) \wedge (\phi^+_{\mathcal{B}}(A) \vee A).$$

◇

THÉORÈME 5.6.2. *[Simplification d'une formule] Soit $\mathcal{F}$ une formule, $\mathcal{I}$ une eq-interprétation partielle de Herbrand. On a :*

$$\mathcal{F} \equiv_{\mathcal{I}} Normalize_{\mathcal{I}}(\mathcal{F}).$$

D'autre part, si $\mathcal{I}$ est totale, $Normalize_{\mathcal{I}}(\mathcal{F})$ est purement équationnelle.

PREUVE. La première partie de la preuve est une simple conséquence du théorème 5.6.1. Si $\mathcal{I}$ est totale, on a pour toute substitution σ, $\sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(A) \vee \phi^-_{\mathcal{I}}(A))$, donc $\neg\phi^-_{\mathcal{I}}(A) \wedge (\phi^+_{\mathcal{I}}(A) \vee A) \equiv_{\mathcal{I}} \neg\phi^-_{\mathcal{I}}(A)$. C.Q.F.D.

REMARQUE. Les résultats de cette section, qui ont été présentés par souci de clarté dans la théorie vide, se généralisent de façon immédiate à toute théorie non-vide $\mathcal{T}$ *à condition qu'il existe une procédure pour décider de la validité d'une formule de $\mathcal{T}$*. Nous ne redonnerons pas ici toutes les définitions correspondantes.

Chapitre 6

La méthode RAMC

6.1 Introduction

La méthode RAMC, initialement définie dans (CAFERRA ET ZABEL, 1990) est décrite dans ce chapitre. Le rappel de la méthode est en effet indispensable à la compréhension de notre travail. Nous introduisons également quelques définitions et théorèmes supplémentaires (non donnés dans (CAFERRA ET ZABEL, 1990; CAFERRA ET ZABEL, 1992)) qui seront utiles par la suite et précisons quelque peu le principe de la méthode.

L'idée générale de la méthode est simple : elle consiste à étendre la notion de conséquence immédiate correspondant à l'application des règles d'inférence avec la notion de *non-conséquence*, correspondant à l'application de règles dites de *dis-inférence*. La méthode exprime des conditions permettant l'application des règles d'inférence *ou empêchant leur application*. Ces conditions sont codées sous forme de *contraintes*, associées aux formules et restreignant le domaine des variables. On peut considérer ces contraintes comme des *sortes dynamiques*, raffinées successivement jusqu'à obtenir dans certains cas un *modèle* de la formule initiale si celle-ci est satisfaisable. Cette approche peut être utilisée pour étendre n'importe quelle procédure de preuve. Elle a été au départ utilisée pour étendre la méthode des tableaux sémantiques (CAFERRA ET ZABEL, 1993), puis la méthode de résolution (CAFERRA ET ZABEL, 1992). Les deux procédures ainsi étendues ont été appelées respectivement RAMCET et RAMC. Dans un premier temps, notre étude portera essentiellement sur la méthode RAMC. Ce chapitre a donc pour objet de définir formellement cette méthode et de préciser ses fondements théoriques. Nous énoncerons et démontrerons également un certain nombre de propriétés supplémentaires.

6.2 Clauses contraintes

DÉFINITION 6.2.1. *Une* clause contrainte *(ou c-clause) notée* $[\![\mathcal{C} \ : \mathcal{X}]\!]$ *est un couple formé d'une clause* $\mathcal{C}$ *de la logique du premier ordre (partie* clause*), et d'une formule équationnelle* $\mathcal{X}$ *(partie* contrainte*).*

Si $\mathcal{C}$ *est unitaire alors* $[\![\mathcal{C} \ : \mathcal{X}]\!]$ *est appelée un* littéral contraint *(ou* c-littéral*). Si* $\mathcal{C}$ *est la clause vide et si* $\mathcal{X}$ *est satisfaisable alors* $[\![\mathcal{C} \ : \mathcal{X}]\!]$ *est notée* $\square$ *(*c-clause vide*). La* profondeur *d'une c-clause* $[\![C \ : \mathcal{X}]\!]$ *est la somme des profondeurs maximales des termes de* C *et de* $\mathcal{X}$*. La* profondeur *d'un ensemble de c-clauses est la profondeur maximale des c-clauses qu'il contient.*
◇

Notation 6.2.1 *On note* $\boldsymbol{Unit}(S)$ *l'ensemble des c-clauses unitaires de* S*.*

REMARQUE. Si $\mathcal{X} \equiv \top$, $[\![C \ : \mathcal{X}]\!]$ sera plutôt notée C (par souci de lisibilité et de concision).

DÉFINITION 6.2.2. *Pour toute c-clause* $[\![C \ : \mathcal{X}]\!]$ *on note* $\mathcal{S}([\![C \ : \mathcal{X}]\!])$*, l'ensemble*

$$\{C\sigma / \sigma \in \mathcal{S}(\mathcal{X})\}$$

De même, si S est un ensemble de c-clauses, l'ensemble $\bigcup_{C \in S} \mathcal{S}(C)$ sera noté $\mathcal{S}(S)$. Pour toute c-clause C les éléments de $\mathcal{S}(C)$ seront appelés des instances closes *de C.* ◇

La définition suivante permet d'étendre la notion de validité (et par là même la notion de satisfaisabilité) aux clauses contraintes et ensembles de clauses contraintes. Informellement, une c-clause C sera équivalente (par définition) à l'ensemble de ses instances closes $\mathcal{S}(C)$.

DÉFINITION 6.2.3. *Soit $\mathcal{I}$ une interprétation et C une c-clause. $\mathcal{I}$* valide la c-clause C *(noté $\mathcal{I} \models C$) si et seulement si pour toute clause fermée $C\sigma \in \mathcal{S}(C)$, $\mathcal{I} \models C\sigma$.* ◇

REMARQUE. Remarquons que les contraintes $\mathcal{X}$ de C sont interprétées dans la théorie vide. Cette définition diffère donc de celle utilisée dans (BOURELY ET AL., 1994), où les contraintes sont interprétées dans une théorie non vide (et non fixée à priori).

On voit donc que pour toute c-clause $[\![C \ : \mathcal{X}]\!]$:

- Si $\mathcal{X} = \top$, alors $[\![C \ : \mathcal{X}]\!]$ est équivalent à C : les clauses sont des cas particuliers des c-clauses. Ce résultat montre que pour toute formule $\mathcal{F}$, il est possible de trouver un ensemble fini de c-clauses S tel que S est satisfaisable si et seulement si $\mathcal{F}$ l'est, et tout modèle de S est modèle de $\mathcal{F}$. Il permet également d'utiliser les algorithmes existants de transformation de formules en ensembles de clauses (voir chapitre 2).
- Si $\mathcal{X} = \bot$ alors $[\![C \ : \mathcal{X}]\!]$ est équivalent à $\top$ (toute c-clause dont les contraintes sont insatisfaisables est une tautologie, elle peut donc être éventuellement supprimée de l'ensemble de c-clauses).

DÉFINITION 6.2.4. *Deux c-clauses C et D sont dites* équivalentes *si et seulement si elles représentent le même ensemble de clauses fermées, c'est-à-dire si*

$$\mathcal{S}(C) = \mathcal{S}(D)$$

Cette relation sera notée $C \cong D$. De même, si S_1 et S_2 sont deux ensembles de c-clauses, nous noterons $S_1 \cong S_2$ si $\forall C_1 \in S_1 \exists C_2 \in S_2 . C_1 \cong C_2$ et $\forall C_2 \in S_2 \exists C_1 \in S_1 . C_1 \cong C_2$. ◇

Par la suite, nous ne considérerons pas une c-clause en tant que telle mais plutôt sa *classe d'équivalence par la relation* $\cong$. Plus précisément, nous identifierons parfois une c-clause C (ou un ensemble de c-clauses S) et l'ensemble correspondant $\mathcal{S}(C)$ (ou $\mathcal{S}(S)$). Nous écrirons par exemple $C \subseteq D$, pour $\mathcal{S}(C) \subseteq \mathcal{S}(D)$, ou $C \cap D$ pour $\mathcal{S}(C) \cap \mathcal{S}(D)$. Cet abus de notation se justifie par le théorème suivant.

THÉORÈME 6.2.1. *1. Le problème du vide pour un ensemble de c-clauses S donné est décidable (il existe un algorithme qui permet de décider si $S \cong \emptyset$).*

2. Pour tout ensemble de c-clauses S_1, S_2, il est possible de calculer des ensembles de c-clauses S_3, S_4, S_5 tels que

(a) $\mathcal{S}(S_3) = \mathcal{S}(S_1) \cup \mathcal{S}(S_2)$;

(b) $\mathcal{S}(S_4) = \mathcal{S}(S_1) \cap \mathcal{S}(S_2)$;

(c) $\mathcal{S}(S_5) = \mathcal{S}(S_1) \setminus \mathcal{S}(S_2)$.

PREUVE. 1. Il suffit de vérifier que pour toute c-clause $[\![C \ : \mathcal{X}]\!]$ de S, $\mathcal{S}(\mathcal{X}) = \emptyset$, ce qui est décidable d'après le théorème 2.3.2.

2(a) la preuve est immédiate : il suffit de prendre $S_3 = S_1 \cup S_2$.

(b) Sans perte de généralité, nous supposerons que les ensembles S_1 et S_2 sont unitaires (il suffit ensuite d'utiliser le résultat ci-dessus et les propriétés de distributivité des opérations $\cup, \cap$ pour étendre ce résultat à des ensembles quelconques). Soit $S_1 \equiv \{[\![C_1 \ : \mathcal{X}_1]\!]\}$ et $S_2 \equiv \{[\![C_2 \ : \mathcal{X}_2]\!]\}$. C_1 est de la forme $\bigvee_{i=1}^{l_1} P_{1,i}(\overline{t}_{1,i})$ et C_2 de la forme $\bigvee_{i=1}^{l_2} P_{2,i}(\overline{t}_{2,i})$ Si $l_1 \neq l_2$ alors on a de façon évidente $\mathcal{S}(S_1) \cap \mathcal{S}(S_2) = \emptyset$, donc il suffit de prendre $S_4 = \emptyset$. Supposons donc que $l_1 = l_2$. Soit s l'ensemble des permutations σ de $[1..l_1]$ telles que $P_{1,i} = P_{2,\sigma(i)}$. Soit $\mathcal{F}$ la formule

$$\bigvee_{\sigma \in s} \bigwedge_{i=1}^{l_1} \overline{t}_{1,i} = \overline{t}_{2,\sigma(i)} \wedge \mathcal{X}_2$$

Soit $S_4 \equiv \{[\![C_1 \ : \mathcal{X}_1 \wedge \mathcal{F}]\!]\}$. On a $\mathcal{S}(S_4) = \mathcal{S}(S_1) \cap \mathcal{S}(S_2)$. En effet, pour toute clause fermée c, $c \in \mathcal{S}(S_4)$ si et seulement si c est de la forme $C_1\theta$ où $\theta \in \mathcal{S}(\mathcal{X}_1 \wedge \mathcal{F})$ c'est-à-dire si et seulement si c est de la forme $\bigvee_{i=1}^{l_1} .P_{1,i}(\overline{t}_{1,i})\theta$ avec $\theta \in \mathcal{S}(\mathcal{X}_1)$ et $\theta \in \mathcal{S}(\mathcal{F})$, i.e. ssi $C_1\theta \in \mathcal{S}(S_1)$ et il existe une permutation σ telle que $\forall i \in [1..n].P_{1,i} = P_{2,\sigma(i)}$, et $\theta \in \mathcal{S}(\overline{t}_{1,i} = \overline{t}_{2,\sigma(i)})$ i.e. ssi $c \in \mathcal{S}(S_1)$ et $\overline{t}_{1,i}\theta = \overline{t}_{2,\sigma(i)}\theta$ et $\theta \in \mathcal{S}(\mathcal{X}_2)$, i.e. : $c \in \mathcal{S}(S_1)$, $C_1\theta = C_2\theta$ (modulo la commutativité et l'associativité de $\vee$) et $\theta \in \mathcal{S}(\mathcal{X}_2)$, i.e. $c \in \mathcal{S}(S_1)$ et $c \in \mathcal{S}(S_2)$.

(c) La preuve est similaire en considérant cette fois la formule $\mathcal{F}$ définie par

$$\neg \bigvee_{\sigma \in s} \exists \overline{x}_2 . \bigwedge_{i=1}^{l_1} \overline{t}_{1,i} = \overline{t}_{2,\sigma(i)} \wedge \mathcal{X}_2$$

où $\overline{x}_2 = \mathcal{V}ar(S_2)$.

C.Q.F.D.

Le théorème 6.2.1 permet d'identifier un ensemble de c-clauses et sa classe d'équivalence par rapport à la relation $\cong$. De même l'égalité "=" entre clauses contraintes désignera l'égalité sur les ensembles de clauses fermées associés aux c-clauses. Par exemple les c-clauses $[\![P(x) \ : x = a]\!]$, $[\![P(a) \ : \top]\!]$ ou $[\![P(y) \ : \exists x . y = x \wedge x = a]\!]$ seront considérées comme égales[1].

Les notations $\cap, \setminus, \cup$, seront souvent utilisées avec des c-clauses unitaires. Le tableau suivant donne la signification de ces opérateurs sur les c-clauses unitaires, plus simple que dans le cas général, donné par le théorème 6.2.1 (voir également (Comon, 1988)).

Notation	Définition
$[\![P(\overline{t}) \ : \mathcal{X}]\!] \cup [\![P(\overline{s}) \ : \mathcal{Y}]\!]$	$[\![P(\overline{x}) \ : \exists \overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X} \vee \exists \overline{z}.\overline{x} = \overline{s} \wedge \mathcal{Y}]\!]$ où $\overline{y} = \mathcal{V}ar([\![P(\overline{t}) \ : \mathcal{X}]\!])$ et $\overline{z} = \mathcal{V}ar([\![P(\overline{s}) \ : \mathcal{Y}]\!])$
$[\![P(\overline{t}) \ : \mathcal{X}]\!] \cap [\![P(\overline{s}) \ : \mathcal{Y}]\!]$	$[\![P(\overline{x}) \ : \exists \overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X} \wedge \exists \overline{z}.\overline{x} = \overline{s} \wedge \mathcal{Y}]\!]$ où $\overline{y} = \mathcal{V}ar([\![P(\overline{t}) \ : \mathcal{X}]\!])$ et $\overline{z} = \mathcal{V}ar([\![P(\overline{s}) \ : \mathcal{Y}]\!])$
$[\![P(\overline{t}) \ : \mathcal{X}]\!] \setminus [\![P(\overline{s}) \ : \mathcal{Y}]\!]$	$[\![P(\overline{x}) \ : \exists \overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X} \wedge \neg(\exists \overline{z}.\overline{x} = \overline{s} \wedge \mathcal{Y})]\!]$ où $\overline{y} = \mathcal{V}ar([\![P(\overline{t}) \ : \mathcal{X}]\!])$ et $\overline{z} = \mathcal{V}ar([\![P(\overline{s}) \ : \mathcal{Y}]\!])$
$[\![P(\overline{t}) \ : \mathcal{X}]\!] \setminus [\![P'(\overline{s}) \ : \mathcal{Y}]\!]$	$[\![P(\overline{t}) \ : \mathcal{X}]\!]$ où $P \neq P'$

Nous introduisons les notations suivantes.

Définition 6.2.5. *Un c-littéral L* appartient à *une c-clause $C : [\![P_1 \vee \ldots \vee P_n \ : \mathcal{X}]\!]$ si et seulement s'il existe i tel que $L = [\![P_i \ : \mathcal{X}]\!]$ (noté $L \in C$).* ◇

Définition 6.2.6. *Pour tout c-littéral $L = [\![P(\overline{t}) \ : \mathcal{X}]\!]$ on note $\neg L$ le littéral $[\![\neg P(\overline{t}) \ : \mathcal{X}]\!]$.* ◇

Définition 6.2.7. *L et L' sont dit* complémentaires *si et seulement si $L \cap \neg L' \neq \emptyset$.* ◇

1. Cette relation d'égalité est évidemment décidable : il suffit de décider si $C \setminus D = \emptyset$ et $D \setminus C = \emptyset$.

Règles de simplification des ensembles de c-clauses

Rappelons quelques règles utiles de transformation des c-clauses. Ces règles préservent l'équivalence des ensembles de c-clauses. Elles permettent de simplifier les contraintes apparaissant dans les c-clauses, au prix d'une *augmentation de la taille de l'ensemble.*

$\vee$-**Elimination**	$\{[\![C \ : \mathcal{P}_1 \vee \mathcal{P}_2]\!]\} \cup S$	$\rightarrow$	$\{[\![C \ : \mathcal{P}_1]\!], [\![C \ : \mathcal{P}_2]\!]\} \cup S$
$=$-**Elimination**	$\{[\![C \ : x = t \wedge \mathcal{P}]\!]\} \cup S$	$\rightarrow$	$\{[\![C\{x \rightarrow t\} \ : \mathcal{P}\{x \rightarrow t\}]\!]\} \cup S$
$\top$-**Introduction**	$\{[\![C \ : \mathcal{P}]\!]\} \cup S$	$\rightarrow$	$\{[\![C \ : \top]\!]\} \cup S$
	Si $\neg\mathcal{P}$ est insatisfaisable.		

THÉORÈME 6.2.2. *Les règles $\vee$-Elimination, $=$-Elimination, $\top$-Introduction sont correctes.*

PREUVE. C'est une conséquence immédiate de la définition 6.2.3. C.Q.F.D.

Forme normale d'un ensemble de c-clauses

DÉFINITION 6.2.8. *Un ensemble de c-clauses S est dit en* forme normale *si les contraintes des c-clauses S sont de la forme*

$$\bigwedge_{i=1}^{n} x_i \neq t_i$$

où les x_i $(1 \leq i \leq n)$ sont des variables telles que $x_i \neq t_i$. $\diamond$

THÉORÈME 6.2.3. *Tout ensemble de c-clauses est équivalent à un ensemble en forme normale.*

PREUVE. Soit une c-clause $[\![C \ : \mathcal{X}]\!]$. D'après le théorème 2.3.2, il existe une formule $\mathcal{X}'$ équivalente à $\mathcal{X}$ de la forme $\bigvee_{i=1}^{k} \mathcal{F}_i$ telle que chaque $\mathcal{F}_i$ est de la forme :

- $\top$
- $\bot$
- $\exists \overline{w}.[\bigwedge_{j=1}^{m} x_j = s_j] \wedge [\bigwedge_{i=1}^{l} x'_i \neq t_i]$, où les x_j, x'_j sont des variables.

Par la règle $\vee$-Elimination $[\![C \ : \mathcal{X}]\!]$ est équivalente à l'ensemble $\{[\![C \ : \mathcal{F}_i]\!] / 1 \leq i \leq k\}$. Par la règle $=$-Elimination, $[\![C \ : \mathcal{F}_i]\!]$ est équivalente à $\{[\![C\{x_j \rightarrow s_j / 1 \leq j \leq m\} \ : \bigwedge_{i=1}^{l} .x'_i \neq t_i]\!]$ (car $\forall j.\ x_j$ n'apparaît qu'une seule fois dans $\mathcal{F}_i$), qui est en forme normale. C.Q.F.D.

Représentation des eq-interprétations par des ensembles de c-clauses

Toute eq-interprétation partielle de Herbrand peut être représentée de façon très naturelle par un ensemble fini de c-clauses unitaires (théorème 6.2.4).

THÉORÈME 6.2.4. *Soit S un ensemble de c-clauses unitaires. Si S est satisfaisable, l'ensemble $\mathcal{S}(S)$ est une eq-interprétation. Réciproquement, pour toute eq-interprétation $\mathcal{I}$ il existe un ensemble fini S de c-clauses unitaires tel que $\mathcal{S}(S) = \mathcal{I}$.*

PREUVE. – Supposons S satisfaisable. Alors pour tout littéral fermé L, si $L \in S$ alors $\neg L \notin S$. D'où S est une interprétation de Herbrand (voir chapitre 5). Il s'agit alors de montrer que S est une eq-interprétation. Pour cela, il suffit de montrer que les ensembles $S^+(P)$ et $S^-(P)$ (pour tout $P \in \Omega$) sont des eq-ensembles. Considérons les formules équationnelles suivantes.

$$\mathcal{F}^+ = \bigvee_{[\![P(\overline{t}) \ : \mathcal{X}]\!] \in S} \exists \overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X}$$

et

$$\mathcal{F}^- = \bigvee_{[\![\neg P(\overline{t}) \ : \mathcal{X}]\!] \in S} \exists \overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X}$$

où $\overline{y} = \mathcal{V}ar([\![P(\overline{t}) \ : \mathcal{X}]\!])$.
On a par définition : $\sigma \in \mathcal{S}(\mathcal{F}^+)$ si et seulement si $P(\overline{x})\sigma \in S$, i.e. $\overline{x}\sigma \in S^+(P)$. De même $\sigma \in \mathcal{S}(\mathcal{F}^-)$ si et seulement si $\neg P(\overline{x})\sigma \in S$ i.e. $\overline{x}\sigma \in S^-(P)$. D'où S est une eq-interprétation.

– Réciproquement, soit une eq-interprétation $\mathcal{I}$. Soit S l'ensemble de c-clauses (fini, si Ω est fini) suivant :

$$\{[\![P(\overline{x}) \ : \mathfrak{F}(\mathcal{I}^+(P))]\!]/P \in \Omega\} \cup \{[\![\neg P(\overline{x}) \ : \mathfrak{F}(\mathcal{I}^-(P))]\!]/P \in \Omega\}$$

Alors on a de façon évidente, par définition de S pour tout littéral fermé L:

$$L \in S - \mathcal{I} \models L.$$

D'où $\mathcal{S}(S) = \mathcal{I}$.

C.Q.F.D.

Ce théorème nous permet d'identifier eq-interprétations et ensembles satisfaisables de c-clauses unitaires.

6.3 La méthode RAMC

Nous pouvons maintenant présenter les règles de la méthode. Elles peuvent être divisées en 3 catégories.

6.3.1 Règles d'inférence

Les *règles d'inférence*, ou *règles de réfutation*, ont pour objet de rechercher une preuve de l'ensemble de c-clauses, c'est-à-dire de chercher à dériver la c-clause vide $\square$. Il s'agit simplement des règles classiques de résolution et de factorisation, adaptées aux clauses avec contraintes.

renommage :

$$\frac{C}{C\sigma}$$

Si σ est un renommage des variables de C.

c-résolution :

$$\frac{[\![P(\overline{t}_1) \vee a \ : \mathcal{X}]\!] \qquad [\![\neg P(\overline{t}_2) \vee b \ : \mathcal{Y}]\!]}{[\![a \vee b \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2]\!]}$$

On note $Res(C, D)$ la c-clause obtenue par c-résolution entre C et D.

c-factorisation :

$$\frac{[\![P(\overline{t}_1) \vee P(\overline{t}_2) \vee a \ : \mathcal{X}]\!]}{[\![P(\overline{t}_1) \vee a \ : \mathcal{X} \wedge \overline{t}_1 = \overline{t}_2]\!]}$$

6.3.2 Règles de dis-inférence

Les règles de construction de modèles ou règles de *dis-inférence* ont pour objet de rechercher un modèle de l'ensemble de c-clauses. Elles cherchent à générer des conditions empêchant l'application des règles de réfutation.

(unit) bc-disrésolution :

$$\frac{c_1 \ : [\![P(\overline{t}_1) \vee a \ : \mathcal{X}]\!] \qquad c_2 \ : [\![\neg P(\overline{t}_2) \ : \mathcal{Y}]\!]}{c_3 \ : [\![P(\overline{t}_1) \vee a \ : \mathcal{X} \wedge (\forall \overline{x}.\neg \mathcal{Y} \vee \overline{t}_1 \neq \overline{t}_2)]\!]}$$

avec $\overline{x} = var(\overline{t}_2) \cup var(\mathcal{Y})$

Soit $c_4 \ : [\![a \ : \mathcal{X} \wedge \mathcal{Y} \wedge t_1 = t_2]\!]$, la c-clause obtenue par c-résolution à partir de c_1 et c_2. Il est facile de voir que $c_1 \wedge c_2$ est équivalent à $c_2 \wedge c_3 \wedge c_4$. Si la disrésolution et la c-résolution sont appliquées conjointement, la clause c_1 pourra donc être supprimée de l'ensemble de clauses.

Nous proposons d'étendre la règle à des clauses quelconques (non unitaires). Le principe est de diviser le domaine D d'une c-clause c_1 en deux sous-domaines D_1 et D_2 tels que l'application de la résolution entre c_1 et c_2 soit possible sur le domaine D_1, impossible sur D_2. Cela revient à remplacer une c-clause c_1 par deux c-clauses c_3 et c_4 obtenues à partir de c_1 en rajoutant aux contraintes de c_1 des conditions supplémentaires empêchant ou permettant l'application de la règle de bc-résolution. Formellement la règle de disrésolution se définit de la façon suivante.

bc-disrésolution généralisée :

$$\frac{c_1 \ : [\![P(\overline{t}_1) \vee a \ : \mathcal{X}]\!] \qquad c_2 \ : [\![\neg P(\overline{t}_2) \vee b \ : \mathcal{Y}]\!]}{c_3 \ : [\![P(\overline{t}_1) \vee a \ : \mathcal{X} \wedge (\forall \overline{x}.\neg \mathcal{Y} \vee \overline{t}_1 \neq \overline{t}_2)]\!] \qquad c_4 \ : [\![P(\overline{t}_1) \vee a \ : \mathcal{X} \wedge (\exists \overline{x}.\mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2)]\!]}$$

avec $\overline{x} = var(\overline{t}_2) \cup var(\mathcal{Y})$

THÉORÈME 6.3.1. *Pour toutes c-clauses c_1, c_2 et pour toutes c-clauses c_3, c_4 générées à partir de c_1 et c_2 par la règle de disrésolution généralisée, $c_1 \equiv (c_3 \wedge c_4)$*

PREUVE. On a l'équivalence :

$$\forall \overline{x}.\neg \mathcal{Y} \vee \overline{t}_1 \neq \overline{t}_2 \equiv \neg(\exists \overline{x}.\mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2).$$

Donc σ sera solution de $\mathcal{X}$ si et seulement si σ valide $\mathcal{X} \wedge \forall \overline{x}.\neg \mathcal{Y} \vee \overline{t}_1 \neq \overline{t}_2$ ou si σ valide $\mathcal{X} \wedge \exists \overline{x}.\mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2$. D'où $c_1 \equiv (c_3 \wedge c_4)$, ce qui permet de supprimer la c-clause c_1 et de la remplacer par c_3 et c_4.

C.Q.F.D.

c-disfactorisation :

$$\frac{[\![P(\overline{t}_1) \vee P(\overline{t}_2) \vee a \ : \mathcal{X}]\!]}{[\![P(\overline{t}_1) \vee P(\overline{t}_2) \vee a \ : \mathcal{X} \wedge \overline{t}_1 \neq \overline{t}_2]\!]}$$

c-dissubsomption :

$$\frac{c_1 \ : [\![\bigvee_{i=1}^{n} L_i(\overline{s_i}) \ : \mathcal{X}]\!] \qquad c_2 \ : [\![\bigvee_{i=1}^{n} L_i(\overline{t_i}) \vee c'_2 \ : \mathcal{Y}]\!]}{c_3 \ : [\![L_1(\overline{t_1}) \vee \ldots \vee L_n(\overline{t_n}) \vee c'_2 \ : \mathcal{Y} \wedge \forall \overline{x}[\neg \mathcal{X} \vee \overline{s_1} \neq \overline{t_1} \vee \ldots \vee \overline{s_n} \neq \overline{t_n}]]\!]}$$

avec $\overline{x} = var(c_1)$

De plus, $c_1 \wedge c_2$ est équivalent à $c_1 \wedge c_3$. La clause c_2 peut donc être supprimée et remplacée par c_3.

c-distautologie :

$$\frac{c_1 \ : [\![P(\overline{t}_1) \vee \neg P(\overline{t}_2) \vee a \ : \mathcal{X}]\!]}{c_2 \ : [\![P(\overline{t}_1) \vee \neg P(\overline{t}_2) \vee a \ : \mathcal{X} \wedge \overline{t}_1 \neq \overline{t}_2]\!]}$$

De même c_1 est équivalent à c_2.

GPL :

La règle GPL(**G**enerating **P**ure **L**iterals) permet de rendre un littéral pur.

DÉFINITION 6.3.1. *Un c-littéral* $[\![P(\overline{t}) \ : \mathcal{X}]\!]$ *est dit* pur *dans un ensemble de c-clauses* S *si et seulement si pour toute c-clause* $C \in S$ *et pour tout c-literal* $L \in C$*, tel que* L *est de la forme* $[\![\neg P(\overline{s}) \ : \mathcal{Y}]\!]$*, on a :*

$$\mathcal{X} \wedge \mathcal{Y} \wedge \overline{s} = \overline{t} \equiv \bot.$$

◇

REMARQUE. Si P est pur dans S alors S est satisfaisable si et seulement si $S \cup \{P\}$ l'est.

L'idée intuitive est la suivante. Un littéral P appartenant à une c-clause C de S pourra faire partie du modèle s'il ne peut se résoudre avec des littéraux des clauses de S. La règle GPL exprime donc les contraintes empêchant l'application de la résolution entre P et les littéraux de S.

La définition formelle est la suivante.

$$\frac{[\![l(\overline{t}) \vee c' \ : \mathcal{X}]\!] \qquad S}{[\![l(\overline{t}) \ : \mathcal{X}_{pure}]\!]}$$

où $\mathcal{X}_{pure} = \bigwedge \{\forall \overline{y}.[\neg \mathcal{Y} \vee \overline{s} \neq \overline{t}] \ : [\![k \ : \mathcal{Y}]\!] \in S \ \textit{et} \ l^c(\overline{s}) \in k\} \wedge \mathcal{X}$ où $\overline{y}$ sont les variables de $\mathcal{Y}$ et de k.

6.3.3 Règles de simplification

Ces règles n'ont d'autre but que de simplifier l'ensemble de c-clauses, en résolvant la partie contrainte ou en simplifiant l'écriture des c-clauses.

Simplification.

$$\frac{[\![c \ : \mathcal{P}]\!]}{[\![c \ : \mathcal{P}']\!]}$$

où $\mathcal{P}'$ *est obtenu par application d'une règle de résolution de contraintes sur* $\mathcal{P}$*.*

$$\frac{[\![c \ : \bot]\!]}{true}$$

6.3.4 Autres règles

Nous ajoutons deux règles supplémentaires.

Règle de décomposition

décomposition :

$$\frac{\{[\![P \vee R \ : \mathcal{X} \wedge \mathcal{Y}]\!]\} \cup S}{\{[\![P \ : \mathcal{X}]\!]\} \cup S, \{[\![R \ : \mathcal{Y}]\!]\} \cup S}$$

si $(\mathcal{V}ar(P) \cup \mathcal{V}ar(\mathcal{X})) \cap (\mathcal{V}ar(R) \cup \mathcal{V}ar(\mathcal{Y})) = \emptyset$.

Contrairement aux autres, la règle de **décomposition** introduit un facteur de branchement dans la méthode, puisqu'un ensemble de c-clauses est remplacé par deux ensembles de c-clauses. Cela a pour inconvénient d'introduire une certaine redondance. Néanmoins, la règle permet de simplifier dans certains cas l'ensemble de c-clauses et de faciliter l'obtention du modèle.

LEMME 6.3.1. *Soit S_1, S_2 deux ensembles de c-clauses déduits de S par la règle de **décomposition** sur une c-clause $[\![P \vee R \ : \mathcal{X} \wedge \mathcal{Y}]\!]$. Alors*

$$S \equiv S_1 \cup S_2.$$

PREUVE. Soit $\mathcal{I}$ une interprétation validant $[\![P \vee R \ : \mathcal{X} \wedge \mathcal{Y}]\!]$. Supposons que $\mathcal{I} \not\models [\![P \ : \mathcal{X}]\!]$. Alors il existe $\theta \in \mathcal{S}(\mathcal{X})$ telle que $\mathcal{I} \not\models P\theta$. Soit une solution σ de $\mathcal{Y}$. Puisque $\mathcal{V}ar(\mathcal{X}) \cap \mathcal{V}ar(\mathcal{Y}) = \emptyset$, on a $\sigma\theta \in \mathcal{S}(\mathcal{X} \wedge \mathcal{Y})$. D'où $\mathcal{I} \models P\sigma\theta \vee R\sigma\theta$. Or $P\sigma\theta \equiv P\theta$ et $R\sigma\theta \equiv R\sigma$. D'où $\mathcal{I} \models R\sigma$. On a donc $\mathcal{I} \models [\![P \ : \mathcal{X}]\!] \vee [\![R \ : \mathcal{Y}]\!]$. D'où $S \equiv S_1 \vee S_2$. C.Q.F.D.

Règle de coupure

La règle suivante peut être utilisée afin d'instancier les variables apparaissant au sein des c-clauses.

Coupure sur les contraintes :

$$\frac{[\![C \ : \mathcal{X}]\!]}{[\![C \ : \mathcal{X} \wedge \mathcal{Y}]\!] \qquad [\![C \ : \mathcal{X} \wedge \neg\mathcal{Y}]\!]}$$

6.4 Correction et complétude réfutationnelle

THÉORÈME 6.4.1. *[Correction] Les règles de c-résolution, c-disrésolution, c-factorisation, c-disfactorisation, c-dissubsomption, simplification, décomposition et coupure sont correctes (c'est-à-dire que toute c-clause déduite de S par ces règles est une conséquence logique de S). La règle* GPL *préserve la satisfaisabilité de l'ensemble de c-clauses. Si P est déduite de S par application de* GPL *alors $S \cup \{P\}$ est satisfaisable si et seulement si S l'est.*

THÉORÈME 6.4.2. *[Complétude] Le calcul composé des règles de réfutations: c-résolution et c-factorisation est complet pour la réfutation, c'est-à-dire que pour tout ensemble de c-clauses insatisfaisable S, il existe une dérivation $S \rightarrow \square$ utilisant uniquement les règles de c-résolution et c-factorisation.*

PREUVE. Voir (CAFERRA ET ZABEL, 1992). C.Q.F.D.

Notation 6.4.1 *On note:*

- RAMC *le système:* { ***renommage, c-résolution, c-factorisation, c-disrésolution, c-disfactorisation, c-dissubsomption, simplification, décomposition*** }.

– RAMC$_{CC}$ *le système:* RAMC $\cup \{$***coupure***$\}$.

Une règle ne sera appliquée sur un ensemble S que si l'ensemble de c-clauses S' obtenu est distinct de S, c'est-à-dire $S \neq S'$.

Soit un ensemble S de c-clauses. RAMC s'arrête dans chacun des trois cas suivants.

1. La clause vide a été générée. Ceci prouve que S est insatisfaisable.
2. On obtient un ensemble S' stable de c-clauses unitaires. S' constitue un modèle partiel de Herbrand de S.
3. On obtient un ensemble irréductible par les règles de RAMC, mais certaines clauses ne sont pas unitaires. S est satisfaisable, mais un modèle ne peut être construit.

THÉORÈME 6.4.3. *Le système { dissubsomption, factorisation, disfactorisation, distautologie } est à terminaison finie.*

PREUVE. La preuve est immédiate, en effet le nombre d'applications possibles des règles décroît à chaque fois. C.Q.F.D.

Notation 6.4.2 *On note:*

– *DSub(E, F) l'ensemble obtenu en appliquant la règle de dissubsomption sur F en utilisant les c-clauses de E.*
– *ModelCheck(E, F) l'ensemble obtenu en appliquant les règles de dissubsomption et de distautologie sur F en utilisant les c-clauses de E.*
– *N_{ramc} le système composé des règles de simplification, factorisation, disfactorisation, dissubsomption et distautologie*

Nous introduisons également l'ordre suivant sur les c-clauses.

DÉFINITION 6.4.1. *On note $C \leq_{dissub} D$ si et seulement si $DSub(C, D) = \emptyset$.* ◇

Remarquons que le théorème 6.4.2 n'utilise aucune stratégie particulière, notamment en ce qui concerne les règles de simplification et de dis-inférence. Nous précisons ci-dessous les conditions garantissant la complétude de la méthode. En particulier, nous montrons la complétude d'une stratégie appliquant les règles de normalisation (i.e. le système N_{ramc}) aussitôt que possible.

DÉFINITION 6.4.2. *Une stratégie d'application $\mathfrak{S}$ des règles de* RAMC *est dite* équitable pour la réfutation *si et seulement si pour toute clause fermée non tautologique déductible de S par c-résolution et c-factorisation, il existe une dérivation selon la stratégie $\mathfrak{S}$ conduisant à un ensemble de c-clause S' tel qu'il existe $C' \in S'$ avec $C' \leq_{dissub} C$.* ◇

REMARQUE. Toute stratégie équitable pour la réfutation est compléte pour la réfutation (puisque $C' \leq_{dissub} \square$ si et seulement si $C' = \square$).

Soit $\mathcal{R}(S)$ l'ensemble de c-clauses obtenu en appliquant les règles c-résolution, GPL, et c-disrésolution de toutes les façons possibles sur S. Soit $\mathcal{R}_s(S) = N_{ramc}{}^*(\mathcal{R}(S))$.

Pour tout ensemble de c-clauses S, on définit la suite : $S_0 = S$ et $S_{i+1} = \mathcal{R}_s(S_i)$. Par induction sur la longueur de la dérivation, on montre que si une clause fermée C non tautologique est déductible de S par c-résolution et c-factorisation, alors il existe $i \in \mathbb{N}$ et $C' \in S_i$ tels que $C' \leq_{dissub} C$. Par conséquent, si S est insatisfaisable, alors il existe i tel que $\square \in S_i$.

REMARQUE. Nous introduirons au chapitre 7 une notion similaire à la notion d'équité pour la réfutation, pour l'aspect construction de modèle.

6.5 RAMC et vérification de modèles

Les règles de RAMC peuvent également être utilisées pour *vérifier* qu'une interprétation partielle est un modèle d'un ensemble de c-clauses (et éventuellement pour compléter le modèle).

THÉORÈME 6.5.1. *Pour tout littéral fermé* $P(\overline{s})$ *:*

$$\mathcal{I}(P(\overline{s})) = \begin{cases} true & Si\ \mathcal{I}(\overline{s}) \in \mathcal{I}(P)^+ \\ false & Si\ \mathcal{I}(\overline{s}) \in \mathcal{I}(P)^- \\ undefined & sinon \end{cases}$$

Pour toute clause fermée $c\ :\ L_1(\overline{s_1}) \vee \ldots \vee L_n(\overline{s_n})$ *:*

$$\mathcal{I}(c) = \begin{cases} true & si\ \exists i \in \{1..n\}\ \mathcal{I}(L_i(\overline{s_i})) = true \\ true & ou\ si\ \exists i,j \in \{1..n\}^2\ L_i = \neg L_j\ et\ \mathcal{I}(\overline{s_i}) = \mathcal{I}(\overline{s_j}) \quad \textbf{(1)} \\ false & si\ \forall i \in \{1..n\}\ \mathcal{I}(L_i(\overline{s_i})) = false \\ undefined & sinon \end{cases}$$

Pour toute c-clause $[\![c\ :\mathcal{P}]\!]$*:*

$$\mathcal{I}([\![c\ :\mathcal{P}]\!]) = \begin{cases} true & si\ \forall \sigma \in \mathcal{S}(\mathcal{P})\ \mathcal{I}(\sigma(c)) = true \\ false & si\ \exists \sigma \in \mathcal{S}(\mathcal{P})\ \mathcal{I}(\sigma(c)) = false \\ undefined & sinon \end{cases}$$

REMARQUE. La condition **(1)** n'était pas présente dans (CAFERRA ET ZABEL, 1992). Elle est néanmoins nécessaire afin que les clauses de la forme : $P(\overline{t}) \vee \neg P(\overline{t}) \vee R$ (qui sont des tautologies) soient valides dans toute interprétation. Par exemple, sans la condition **(1)**, la c-clause $P(a) \vee \neg P(a)$, n'est pas valide dans le modèle $\mathcal{I}$ défini par :

$$\mathcal{I}(P)^+ = \mathcal{I}(P)^- = \emptyset$$

PREUVE. C'est une conséquence immédiate de la définition 6.2.3. C.Q.F.D.

Soit un ensemble de c-clauses S, et une eq–interprétation partielle de Herbrand $\mathcal{I}$ (exprimée comme un ensemble de c-clauses unitaires E). Le problème est de vérifier que $\mathcal{I}$ valide S (i.e. que E implique S).

THÉORÈME 6.5.2. *Soit* $\mathcal{I}$ *une eq-interprétation (considérée ici comme un ensemble fini de c-littéraux) et* S *un ensemble de c-clauses. Alors*

$$\mathcal{I} \models S - \mathit{ModelCheck}(\mathcal{I}, S) = \emptyset.$$

Si de plus $\mathcal{I}$ *est totale,*

$$\mathcal{I} \models S - \mathit{DSub}(\mathcal{I}, S) = \emptyset$$

PREUVE. – Soit une c-clause $C = [\![P_1(\overline{t_1}) \vee \ldots \vee P_n(\overline{t_n})\ :\mathcal{X}]\!]$ de S, C' la clause de $Simpl(E,S)$ correspondante. Supposons que $\mathcal{I} \models C$.
Soit une substitution $\sigma \in \mathcal{S}(\mathcal{X})$. On a $\mathcal{I} \models \sigma(P_1(\overline{t_1}) \vee \ldots \vee P_n(\overline{t_n}))$, donc :

- Soit il existe i tel que $P_i = Q$ et $\sigma(\overline{t_i}) \in \mathcal{I}(Q)^+$. Dans ce cas, il existe une c-clause $[\![Q(\overline{s})\ :\mathcal{X}']\!] \in E$ telle que $\mathcal{X}' \wedge \mathcal{X} \wedge \overline{t} = \overline{s} \not\equiv \bot$, donc $\sigma \notin \mathcal{S}(\mathcal{Y})$ (par irréductibilité par dissubsomption)
- Soit il existe i tel que $P_i = \neg Q$ et $\sigma(\overline{t_i}) \in \mathcal{I}(Q)^-$. Idem.
- Soit il existe i,j tels que $P_i(\overline{t_i}) = \neg P_j(\overline{t_j})$. Alors, par irréductibilité par distautologie, $\sigma \notin \mathcal{Y}$

Réciproquement, supposons que $\mathcal{Y} \equiv \bot$. Soit σ une solution de $\mathcal{X}$. $\sigma \notin \mathcal{S}(\mathcal{Y})$ donc, par définition de $\mathcal{Y}$:

- Soit il existe $[\![P_i(\overline{t_i}) \ : \mathcal{X}_i]\!] \in E$ tel que $\mathcal{X} \wedge \forall \overline{x_i}(\neg \mathcal{X}_i \vee \overline{t_i} \neq \overline{s_i}) \equiv \bot$, auquel cas $\sigma(\overline{t_i}) \in \mathcal{I}(P_i)^+$ d'où $\mathcal{I} \models C$.
- De même avec $[\![\neg P_i(\overline{t_i}) \ : \mathcal{X}_i]\!]$
- Soit il existe $i, j \in \{1..n\}$ tels que $P_i = \neg P_j$ et $\sigma \in \mathcal{S}(t_i = t_j)$. Alors $\sigma((P_1(\overline{t_1}) \vee \ldots \vee P_n(\overline{t_n}))$ est une tautologie et $\mathcal{I} \models C$

– La preuve est similaire (dans ce cas on n'a plus besoin de la règle de distautologie, puisque, pour tout littéral P, soit $\mathcal{I} \models P$, soit $\mathcal{I} \models \neg P$).

C.Q.F.D.

Les règles de RAMC permettent donc de vérifier qu'un modèle partiel $\mathcal{I}$ valide un ensemble de c-clauses S. Si l'application des règles de dissubsomption et de distautologie entre S et l'ensemble E associé à $\mathcal{I}$ réduit les contraintes des c-clauses S à $\bot$, (donc la règle de simplification supprime les c-clauses obtenues) alors $\mathcal{I}$ est un modèle de S. Sinon $\mathcal{I}$ n'est pas un modèle de S, et les autres règles de RAMC peuvent alors être utilisées afin de compléter le modèle, ou de prouver que $\mathcal{I}$ est un *contre-modèle* de S. Ce résultat fournit également un algorithme pour décider si une interprétation partielle valide une formule du premier ordre sans quantificateur existentiel (puisque toute formule sans quantificateur existentiel peut être automatiquement transformée en un ensemble de c-clauses équivalent).

REMARQUE. Comme au chapitre 5, les résultats présentés ici se généralisent de façon immédiate à n'importe quelle théorie décidable $\mathcal{T}$. La complétude de la méthode est alors une conséquence de la complétude de la théorie-résolution (STICKEL, 1985).

Chapitre 7

Limites et extensions de la méthode RAMC

Ce chapitre présente plusieurs extensions de la méthode RAMC dont la version initiale est décrite au chapitre 6. De nouvelles règles d'inférence et de disinférence sont définies et certaines règles existantes (notamment la règle GPL) sont étendues de façon importante. Nous montrons l'intérêt de ces extensions en prouvant qu'elles permettent d'étendre la classe de modèles constructibles. Est présentée notamment une extension de la stratégie de résolution sémantique qui permet de réduire l'espace de recherche, à la fois pour la recherche de réfutations et pour la construction de modèles. Enfin, nous concluons en montrant les limites de la méthode.

7.1 Les règles GPL et GMPL

La puissance de la méthode RAMC repose pour une bonne part sur la règle GPL qui permet d'engendrer des c-clauses unitaires constituant un modèle partiel de la formule initiale. L'intérêt principal de GPL est d'engendrer des c-clauses qui ne sont *pas des conséquences logiques de l'ensemble de c-clauses*. Dans la section suivante, la nécessité d'une extension de cette règle est mise en évidence par quelques exemples montrant les limites de sa version initiale.

7.1.1 Limites de la règle GPL

Traitement des clauses auto-résolvantes

EXEMPLE 7.1.1.

Considérons l'ensemble de clauses suivant : $E = \{\neg P(x) \vee P(f(x))\}$, avec $\Sigma = \{a^0, f^1\}$. E contient une seule clause, donc E est satisfaisable. Un modèle $\mathcal{I}$ de E est, par exemple, l'interprétation $\mathcal{I}$ définie par

$$\mathcal{I}(P)^+ = \tau(\Sigma).$$

Or, aucun modèle de E ne peut être engendré par l'application de la règle GPL. En effet, GPL ne peut pas être appliquée sur le littéral $P(f(x))$ à cause de la présence du littéral $\neg P(x)$. D'autre part, si la règle est appliquée sur le littéral $\neg P(x)$, on obtient :

$[\![\neg P(x) \ : \forall y.(x \neq f(y)]\!])$ (GPL)
$[\![\neg P(a) \ : \top]\!]$ (résolution des contraintes)
$[\![\neg P(x) \vee P(f(x)) \ : x \neq a]\!]$ (dissubsomption, à partir de $\neg P(x) \vee P(f(x))$)
$[\![\neg P(f(x)) \vee P(f(f(x))) \ : \top]\!]$ (résolution des contraintes)
$[\![\neg P(f(x)) \ : \forall y.(x \neq f(y)]\!])$ (GPL)
...

On génère donc un nombre infini de clauses de la forme $\neg P(f^n(a))$ (n = 0,1,2,...). Il est facile de voir que ce résultat ne dépend pas de la stratégie utilisée pour l'application des règles. ◇

Dans l'exemple ci-dessus, les contraintes empêchant l'unification de $P(x)$ avec $P(f(x))$ sont ajoutées au littéral $\neg P(x)$, alors que la c-clause $[\![\neg P(x) \vee P(f(x)) \ : \top]\!]$ est subsumée par $\neg P(x)$. La condition garantissant que l'ajout du littéral L préserve la satisfaisabilité (imposant que L soit pur dans l'ensemble de c-clauses) est trop forte pour pouvoir construire des modèles dans le cas de cet exemple. Nous proposons par conséquent d'*affaiblir* cette condition en introduisant la définition suivante.

DÉFINITION 7.1.1. *Un c-littéral L est dit* quasi-pur *dans un ensemble de c-clauses S si et seulement si L est pur dans $DSub(L,S)$.* ◇

REMARQUE. Il est clair que tout c-littéral L pur dans S est également quasi-pur dans S (la réciproque n'étant pas vraie).

LEMME 7.1.1. *Si L est quasi-pur dans S et si S est satisfaisable, alors $S \cup \{L\}$ est satisfaisable.*

PREUVE. Si S est satisfaisable alors $DSub(L,S)$ l'est également. On a $S \cup \{L\} \equiv DSub(L,S) \cup \{L\}$ (par correction de la dissubsomption). D'autre part, L est pur dans $DSub(L,S)$ donc $DSub(L,S) \cup \{L\}$ est satisfaisable. C.Q.F.D.

Il reste à donner un algorithme pour construire automatiquement des c-littéraux quasi-purs dans un ensemble de c-clauses S donné. Ce problème est évidemment beaucoup plus difficile et plus coûteux que la génération de c-littéraux purs. Dans (BOURELY ET AL., 1994) est proposé un algorithme, nommé EGPL (**E**xtended **GPL**), permettant d'extraire un c-littéral quasi-pur à partir d'un ensemble de c-clauses. Nous ne le présenterons pas ici, car il constitue un cas particulier de la règle GMPL présentée plus loin.

Cas d'interdépendance entre 2 littéraux

Appliquer GPL sur un littéral unique ne permet pas toujours de construire un modèle de l'ensemble de c-clauses, car l'interprétation du littéral considéré ne peut pas en général être fixée indépendamment de celle des autres littéraux de l'ensemble. Considérons, par exemple, la formule

$$A(x) \leftrightarrow B(x).$$

L'ensemble de c-clauses correspondant est :

$$\{[\![A(x) \vee \neg B(x) \ : \top]\!], [\![\neg A(x) \vee B(x) \ : \top]\!]\}$$

Appliquons la règle GPL sur le littéral $A(x)$. Nous obtenons

$$[\![A(x) \ : \forall y.y \neq x]\!]$$

c'est-à-dire (par résolution des contraintes)

$$[\![A(x) \ : \bot]\!].$$

Cette c-clause est une tautologie, ce qui n'apporte aucune information supplémentaire. Par symétrie, on obtient le même résultat en appliquant GPL sur $B(x), \neg A(x)$ ou $\neg B(x)$. En effet, A et B étant équivalents, il est impossible de fixer l'interprétation de A sans fixer *en même temps* celle de B.

Afin de prendre en compte cette difficulté, nous proposons de généraliser la définition 7.1.1 à des *ensembles* de c-littéraux.

DÉFINITION 7.1.2. *Un ensemble de c-littéraux E est dit* quasi-pur *dans un ensemble de c-clauses si et seulement si :*

– *E est satisfaisable.*

– *Tout littéral de E est pur dans $DSub(E, S)$.*

◇

THÉORÈME 7.1.1. *Si E est quasi-pur dans S et si S est satisfaisable, alors $E \cup S$ est satisfaisable.*

PREUVE. Si S est satisfaisable, $S' = DSub(E, S)$ l'est, car les c-clauses de S' sont des *restrictions* des clauses de S. Si $S \cup E$ est insatisfaisable, alors $S' \cup E$ l'est également (par correction de la dissubsomption). Si $S' \cup E$ est insatisfaisable, S' l'est, puisque la règle GPL préserve l'équivalence (CAFERRA ET ZABEL, 1992). En effet, deux c-littéraux quelconques de E n'étant pas complémentaires (puisque E est satisfaisable), E peut être obtenu à partir de S' par $card(E)$ applications de la règle GPL. D'où S est satisfaisable implique que $S \cup E$ est satisfaisable. C.Q.F.D.

La section suivante est consacrée à la présentation d'un algorithme engendrant automatiquement des ensembles de c-littéraux quasi-purs.

7.1.2 La règle GMPL

Présentation informelle

Avant de définir formellement la règle GMPL (**G**enerating **M**any **P**ure **L**iterals) donnons une idée intuitive de son principe.

L'objet de la règle GPL est de transformer un c-littéral P en un c-littéral pur P' tel que l'ajout de P' à S préserve la satisfaisabilité de S. Pour cela, on ajoute au c-littéral "candidat" P des contraintes empêchant l'unification entre ce c-littéral et tout littéral complémentaire de l'ensemble de c-clauses. La procédure peut se formaliser par une règle de **Restriction** définie de la façon suivante :

(Restriction) $L \rightarrow L \setminus L'$
Il existe une c-clause C de S telle que : $\neg L' \in C$

REMARQUE. Notons que cette règle (ainsi que les règles d'extension, de contradiction et de simplification introduites dans cette section) l'est *pas* une règle d'inférence, ni une règle de dis-inférence, mais simplement une règle de calcul permettant de trouver des c-littéraux purs.

Il est clair que tout c-littéral L irréductible par la règle **Restriction** est pur dans S. Afin d'engendrer des c-littéraux quasi-purs, nous renforçons les conditions d'application de la règle, pour empêcher l'application de la règle si le littéral candidat subsume la c-clause C. Nous obtenons la règle suivante.

(Restriction) $L \rightarrow L \setminus L'$ S'il existe une c-clause C de S telle que :
$\neg L' \in C$ et $DSub(\{L\}, C) \neq \top$

REMARQUE. Notons que l'opérateur $\setminus$ introduit au chapitre 6 permet d'exprimer la règle (ainsi que les suivantes) de façon très simple.

De façon évidente, si L est irréductible par rapport à la règle **Restriction**, alors L est quasi-pur dans S, *mais pas nécessairement pur* dans S.

Afin d'engendrer simultanément un ensemble de littéraux quasi-purs, on considère non plus des c-littéraux candidats mais des *ensembles de c-littéraux*. La règle de **Restriction** se généralise de façon immédiate :

(Restriction) $E \cup \{L\} \rightarrow E \cup \{L \setminus L'\}$
S'il existe une c-clause C de S telle que : $\neg L' \in C$
et $DSub(E, C) \neq \top$

Considérons à nouveau l'ensemble S :

$$\{[\![A(x) \vee \neg B(x) \ : \top]\!], [\![\neg A(x) \vee B(x) \ : \top]\!]\}$$

Initialement, on a : $E = \{A(x)\}$. On cherche à transformer l'ensemble E en un ensemble E' tel que l'ajout de E' à S préserve la satisfaisabilité de S. Au lieu d'empêcher l'application de la résolution entre $A(x)$ et $\neg A(x) \vee B(x)$ en ajoutant des contraintes sur $A(x)$, nous allons garantir que le résolvant obtenu par résolution sera subsumé par l'ensemble E, en ajoutant à E le littéral $B(x)$. Nous obtenons ainsi l'ensemble $E' : \{A(x), B(x)\}$. Cet ensemble préserve de façon évidente la satisfaisabilité de S, puisque toute c-clause de S est subsumée par une clause de E. L'ajout d'un littéral à l'ensemble E se formalise par la règle **Extension** suivante.

(Extension) $E \rightarrow E \cup \{L'\}$
Si $C \in \mathit{Res}(E, S)$, $\mathit{DSub}(E, C) \not\equiv \top$ et $L' \in C$

L'ensemble E doit être satisfaisable. Or, rien ne garantit que la satisfaisabilité de E est préservée par la règle **Extension** qui peut introduire, dans l'ensemble de c-littéraux-candidats E, le complémentaire d'un littéral appartenant à E. Pour résoudre ce problème, nous proposons une règle **Contradiction** dont l'objet est d'ajouter aux c-littéraux de E des contraintes garantissant la satisfaisabilité de E, c'est-à-dire empêchant l'application de la résolution entre 2 c-clauses de E.

(Contradiction) $E \cup \{L, \neg L'\} \rightarrow E \cup \{\neg L', L \setminus L'\}$

Enfin, nous ajoutons également une règle de **Simplification** destinée à supprimer de l'ensemble E les c-littéraux dont les contraintes sont insatisfaisables.

(Simplification) $E \cup [\![C \ : \mathcal{X}]\!] \rightarrow E$
Si $\mathcal{X} \equiv \bot$

Définition formelle

Nous obtenons donc la règle suivante :

GMPL (**G**enerating **M**any **P**ure **L**iterals) : $\dfrac{S}{E}$

où E est irréductible par le système de règle R_S suivant (E peut être obtenu en appliquant R_S jusqu'à saturation sur un c-littéral $[\![L(\bar{t}) \ : \mathcal{X}]\!]$ de S).

(Simplification) $E \cup [\![C \ : \mathcal{X}]\!] \rightarrow E$
Si $\mathcal{X} \equiv \bot$

(Restriction) $E \cup \{L\} \rightarrow E \cup \{L \setminus L'\}$
Il existe une c-clause C de S telle que : $\neg L' \in C$
et $\mathit{DSub}(E, C) \not\equiv \top$

(Extension) $E \rightarrow E \cup \{L'\}$
Si $C \in \mathit{Res}(E, S)$, $\mathit{DSub}(E, C) \not\equiv \top$ et $L' \in C$

(Contradiction) $E \cup \{L, \neg L'\} \rightarrow E \cup \{\neg L', L \setminus L'\}$

REMARQUE. – La règle GPL correspond à une application des règles précédentes où :

- La règle **Extension** n'est jamais appliquée (E est donc de cardinalité 1 et la règle **Contradiction** n'est jamais appliquée).
- La condition $\mathit{DSub}(E, S) \not\equiv \top$ n'est pas prise en compte dans l'application de la règle de restriction.

– De même, la règle EGPL proposée dans (BOUREL Y ET AL., 1994) correspond à une application de ces règles dans laquelle la règle **Extension** n'est jamais appliquée.

Avant d'établir les propriétés de terminaison et de correction de la règle GMPL, nous donnons un exemple d'application, qui illustre son fonctionnement et montre son utilité.

Exemple

EXEMPLE 7.1.2. Soit $\Sigma = \{a, g\}$. Soit S l'ensemble de c-clauses suivant.

$$\begin{array}{l} 1\ [\![P(x) \vee R(x)\ \ : \top]\!] \\ 2\ [\![P(a) \vee \neg Q(a)\ \ : \top]\!] \\ 3\ [\![\neg P(a) \vee Q(a)\ \ : \top]\!] \\ 4\ [\![\neg P(g(x))\ \ : \top]\!] \end{array}$$

Appliquons la règle GMPL sur le littéral $P(x)$. Initialement, on a $E = \{P(x)\}$. Nous pouvons appliquer la règle de **Restriction** sur $P(x)$ avec la c-clause 3. On obtient l'ensemble de littéraux $\{[\![P(x)\ \ : x \neq a]\!]\}$. Ensuite, nous appliquons la règle de **Restriction** sur $[\![P(x)\ \ : x \neq a]\!]$ avec la c-clause 4. On obtient : $\{[\![P(x)\ \ : x \neq a \wedge \forall y.x \neq g(y)]\!]\}$. Par application de la règle de **Simplification**, cet ensemble est transformé en l'ensemble $\emptyset$ qui est évidemment irréductible par R_S, mais n'apporte aucune information intéressante sur S.

Au lieu d'appliquer la règle **Restriction** à l'étape 1, nous pouvons appliquer la règle **Extension**. On obtient l'ensemble de c-littéraux $\{[\![P(x)\ \ : \top]\!], [\![Q(a)\ \ : \top]\!]\}$

Ensuite, on obtient par application de la règle de **Restriction** sur la c-clause 4 :

$$\{[\![P(x)\ \ : \forall y.x \neq g(y)]\!], [\![Q(a)\ \ : \top]\!]\}$$

d'où :

$$E' = \{[\![P(a)\ \ : \top]\!], [\![Q(a)\ \ : \top]\!]\}$$

E' est irréductible par R_S. ◇

REMARQUE. Contrairement aux autres règles, le calcul de l'ensemble de c-littéraux E nécessite l'utilisation du *retour-arrière* afin de calculer tous les ensembles de c-littéraux quasi-purs possibles (voir exemple 7.1.2). Cependant, aucun retour-arrière n'est nécessaire après l'ajout de E à S puisque le théorème 7.1.3 assure que E préserve la satisfaisabilité de S.

7.1.3 Propriétés

Terminaison

THÉORÈME 7.1.2. *Pour tout S et pour tout E, l'application non-déterministe des règles de R_S sur E se termine si la règle* ***Extension*** *est appliquée au plus un nombre fini de fois (fixé à l'avance) sur chaque littéral.*

PREUVE. Puisque la règle **Extension** n'est appliquée qu'un nombre fini de fois, nous pouvons, sans perte de généralité, supposer que la dérivation ne contient aucune application de cette règle. Soit C_n le nombre de couples de littéraux complémentaires de E et R_n le nombre de couples $(L1, L2)$ de $S \times E$ complémentaires et S_n le nombre de littéraux $[\![L\ \ : \bot]\!]$ de E.

A chaque application d'une règle :

- Si **Restriction** est appliquée, R_n décroît strictement.
- Si **Contradiction** est appliquée, R_n n'augmente pas et C_n décroît strictement.
- Si **Simplification** est appliquée, R_n et C_n n'augmentent pas et S_n décroît strictement.

Il est donc facile de voir que le n-uplet (R_n, C_n, S_n) est strictement décroissant (par rapport à l'extension lexicographique de l'ordre naturel sur les entiers) et donc que l'algorithme se termine. C.Q.F.D.

Correction

THÉORÈME 7.1.3. *Soient S un ensemble de c-clauses, et E un ensemble de c-littéraux irréductible par la règle de restriction de R_S. S est satisfaisable si et seulement si $S \cup E$ l'est.*

PREUVE. D'après le théorème 7.1.1, il suffit de prouver que E est quasi-pur dans S.

E étant irréductible par la règle Contradiction, E est satisfaisable. Il suffit donc de montrer que tout littéral de E est pur dans $S' = DSub(E, S)$. Supposons qu'il existe $L_1 \in C' \in S'$ et $L_2 \in E$, tel que L_1 et $\neg L_2$ soient unifiables et tel que $C' \not\equiv \top$. Puisque les c-clauses de S' sont des restrictions des c-clauses de S, il existe $L_3 \in C \in S$ tel que $L_1 \subseteq L_3$. Alors L_3 et $\neg L_2$ sont unifiables. Par définition, la règle de restriction s'applique sur E, ce qui est impossible. C.Q.F.D.

7.2 Extension de la stratégie de résolution sémantique

7.2.1 Résolution sémantique

En 1967, SLAGLE propose une nouvelle stratégie pour la méthode de résolution. Cette stratégie est fondée sur l'utilisation d'une interprétation de la formule à réfuter, qui est utilisée pour restreindre l'application de la résolution. Le principe est simple : puisque la résolution est une règle correcte, il est inutile de chercher à générer une contradiction ($\square$) en appliquant la résolution entre deux clauses valides dans une même interprétation. En effet, la conclusion sera nécessairement valide dans l'interprétation et donc ne pourra être une contradiction. Intuitivement, nous pouvons donc empêcher l'application de la résolution entre deux clauses valides dans la même interprétation. C'est l'idée de la résolution sémantique. Cette stratégie se révèle particulièrement efficace en pratique (voir par exemple (SLANEY, 1993)). Elle est également très générale : par exemple, l'hyperrésolution peut être vue comme un cas particulier de la résolution sémantique (elle correspond à une interprétation initiale $\mathcal{I}_s$ telle que tout littéral positif fermé est faux dans $\mathcal{I}_s$). Nous montrons dans cette section comment intégrer l'utilisation de la stratégie sémantique dans la méthode RAMC avec un double avantage.

- D'une part, réduire l'espace de recherche de la méthode.
- D'autre part, étendre la résolution sémantique en utilisant les contraintes pour coder les conditions sémantiques sur les c-clauses générées. Cette technique permet de résoudre d'une façon élégante un problème inhérent à l'utilisation de la résolution sémantique : l'existence de dérivation sémantiques non fermées qui ne correspondent à aucune dérivation sémantique fermée (voir (SANDFORD, 1980)).

7.2.2 Principe de l'extension proposée

La méthode RAMC permettant de construire des modèles d'ensembles de c-clauses, il est naturel de chercher à utiliser les modèles ainsi construits pour guider la recherche d'une preuve. Pourtant, nous pouvons aller plus loin : au lieu d'évaluer – avant l'application de la règle de résolution – les c-clauses dans l'interprétation $\mathcal{I}_s$, il est possible de *coder dans les contraintes* du résolvent des conditions imposant que l'une au moins des c-clauses est fausse dans $\mathcal{I}_s$. Il s'agit de traduire en une formule équationnelle la condition : "une au moins des c-clauses parentes est fausse dans $\mathcal{I}_s$", et d'ajouter cette condition au résolvent.

$$[\![C \ : C \text{ est fausse dans } \mathcal{I}_s]\!]$$

Notons que la condition "C est fausse dans $\mathcal{I}_s$" s'exprime de façon immédiate en utilisant la formule ϕ^- définie au chapitre 5.

7.2.3 Une nouvelle règle de réfutation

Règle de c-résolution

Soit $\mathcal{I}_s$ une eq-interprétation, représentée par un ensemble fini de c-clauses unitaires.
La définition suivante étend la notion de *clash*[1] (SLAGLE, 1967) aux clauses contraintes.

DÉFINITION 7.2.1.
Un c-clash *est un ensemble fini de c-clauses:* $\{E_1, \ldots, E_q, C\}$ *satisfaisant les conditions suivantes.*

– *C contient au moins q littéraux* $L_1(\overline{t_1}), \ldots L_q(\overline{t_q})$.

$$C = [\![\bigvee_{i=1}^{q} L_i(\overline{t_i}) \vee R \ : \mathcal{X}]\!].$$

– *Pour tout* $i \in [1..q]$, *E_i contient un littéral* $\neg L_i(\overline{s_i})$.

$$E_i = [\![\neg L_i(\overline{s_i}) \vee R_i \ : \mathcal{X}_i]\!].$$

– $\bigwedge_{i=1}^{n}(\mathcal{X}_i \wedge s_i = t_i \wedge \neg\phi^+_{\mathcal{I}_s}(E_i)) \wedge \neg\phi^+_{\mathcal{I}_s}(R)$ *a au moins une solution.*

Le c-$\mathcal{I}_s$-résolvent *(ou* c-résolvent sémantique*) de* $\{E_1, \ldots, E_q, C\}$ *est la c-clause*

$$[\![R \vee \bigvee_{i=1}^{n} R_i \ : \mathcal{Y}_{res}]\!]$$

où

$$\mathcal{Y}_{res} = \mathcal{X} \wedge \bigwedge_{i=1}^{n}(\mathcal{X}_i \wedge s_i = t_i \wedge \neg\phi^+_{\mathcal{I}_s}(E_i)) \wedge \neg\phi^+_{\mathcal{I}_s}(R)$$

◇

REMARQUE. Nous utilisons la formule $\neg\phi^+_{\mathcal{I}_s}$ pour exprimer les conditions sémantiques au lieu de $\phi^-_{\mathcal{I}_s}$ car cela est nécessaire pour préserver la complétude réfutationnelle dans le cas où $\mathcal{I}_s$ est une interprétation *partielle*[2]. Si $\mathcal{I}_s$ est totale, ces deux formules sont équivalentes, comme nous l'avons vu au chapitre 2.

Propriétés de la méthode : correction et complétude réfutationnelle

THÉORÈME 7.2.1. *[Correction] La c-résolution est correcte.*

PREUVE. Soit $\theta \in Sol(\mathcal{Y}_{res})$. Pour tout $i \leq q$, $\theta(s_i) = \theta(t_i)$. Par conséquent, $S' = \{\theta(\neg L_i(\overline{s_i}) \vee R_i)/i \leq q\} \cup \theta(\bigvee_{i=1}^{q} L_i(\overline{t_i}) \vee R)$ est un *clash* (au sens habituel), et nous avons $S' \models \theta(R \vee \bigvee R_i)$. Puisque $\theta \in \mathcal{S}(\mathcal{X})$ et $\theta \in \mathcal{S}(\mathcal{X}_i), \forall i \leq q$, on a $\theta(\bigvee_{i=1}^{q} L_i(\overline{t_i}) \vee R) \in \mathcal{S}(C)$ et $\forall i \leq q.\theta(\neg L_i(\overline{s_i}) \vee R_i) \in \mathcal{S}(E_i)$. D'où $E_1, \ldots E_n, C \models \theta(R \vee \bigvee R_i)$. La règle de c-résolution est donc correcte. C.Q.F.D.

La preuve de la complétude réfutationnelle nécessite les deux lemmes suivants.

LEMME 7.2.1. *[Lemme d'extension du modèle]*
Soit $\mathcal{I}$ une interprétation partielle. Alors, il existe une interprétation totale $\mathcal{I}'$ telle que, pour toute clause fermée C

$$\mathcal{I} \models C \Rightarrow \mathcal{I}' \models C.$$

1. A notre connaissance, il n'existe pas de terme français équivalent utilisé dans la littérature.
2. Notons que si $\mathcal{I}_s$ est vide, $\phi^+_{\mathcal{I}_s}(C) = \phi^-_{\mathcal{I}_s}(C) = \bot$.

PREUVE. Par définition, puisque $\mathcal{I} \models C$, toute interprétation totale de $\mathcal{I}$ valide C. C.Q.F.D.

LEMME 7.2.2. *[lemme de relèvement]*
Soit

$$\{[\![E_1 \ : \mathcal{X}_1]\!], \ldots [\![E_n \ : \mathcal{X}_n]\!], [\![C \ : \mathcal{X}]\!]\}$$

un ensemble de c-clauses. Soient $\sigma_1, \ldots \sigma_n, \sigma$ *des solutions de* $\mathcal{X}_1, \ldots, \mathcal{X}_n, \mathcal{X}$ *respectivement,* $\mathcal{I}_s$ *une eq-interprétation,* $\mathcal{I}'$ *l'extension totale de* $\mathcal{I}_s$ *définie par le lemme 7.2.1. Soit* R_g *un* $\mathcal{I}'$*-résolvent de* $\sigma_1(E_1), \ldots \sigma_n(E_n), \sigma(C)$*. Il existe un* $\mathcal{I}_s$*-c-résolvent de* $[\![E_1 \ : \mathcal{X}_1]\!], \ldots [\![E_n \ : \mathcal{X}_n]\!], [\![C \ : \mathcal{X}]\!]$*, et une substitution* $\theta \in Sol(\mathcal{Y}_{res})$*, telle que*

$$\theta(\bigvee R_i \vee R) = R_g.$$

PREUVE. Soit $\theta = \sigma_1 \ldots \sigma_n \ldots \sigma$. On a $\theta \in \mathcal{S}(\mathcal{X}_i)$ (pour tout i), et $\theta \in \mathcal{S}(\mathcal{X})$. Puisque $\sigma_1(E_1), \ldots, \sigma_n(E_n)$ est un clash, E_i est de la forme $L_i(t_i) \vee R_i$ (pour tout $i \leq n$) et C est de la forme $\bigvee_{i=1}^{n} \neg L_i(s_i) \vee R$ avec $\sigma_i(L_i(t_i)) = \sigma(L_i(s_i))$, d'où $\theta(L_i(t_i)) = \theta(L_i(s_i))$. En outre pour tout i, $\theta(E_i)$ est fausse dans $\mathcal{I}'$, donc, $\theta \notin \mathcal{S}(\phi^+_{\mathcal{I}_s}(E_i))$. De même, $\mathcal{I}' \not\models \theta(R)$, d'où $\theta \notin \mathcal{S}(\phi^+_{\mathcal{I}_s}(R))$.

Nous avons donc $\theta \models \mathcal{Y}_{res}$. De plus, par définition de R_g, on a $\theta(R_g) = \theta(\bigvee R_i \vee R) = \bigvee(\sigma(R_i)) \vee \sigma(R)) = R_g$ C.Q.F.D.

THÉORÈME 7.2.2. *[Complétude réfutationelle]*
Pour toute eq-interprétation $\mathcal{I}_s$*, et pour tout ensemble insatisfaisable* S *de c-clauses, il existe une réfutation de* S *utilisant seulement les règles de* $\mathcal{I}_s$*-c-résolution et de c-factorisation.*

PREUVE. Par complétude réfutationnelle de la résolution sémantique (SLAGLE, 1967) il existe une réfutation sémantique guidée par $\mathcal{I}'$ et fermée de S. D'après le lemme 7.2.2, elle peut être transformée en une $\mathcal{I}_s$-c-réfutation de S. C.Q.F.D.

REMARQUE. Ce résultat reste valide si les règles de normalisation (le système N_{ramc}) sont appliquées aussitôt que possible (voir chapitre 6).

Contrairement à l'approche classique, la réciproque du lemme de relèvement est également valide comme le montre le théorème suivant.

LEMME 7.2.3. *Soit* $\mathcal{I}_s$ *une eq-interprétation totale. Soit* $E_1, \ldots E_n, C$ *un c-clash. Soit* $[\![R \ : \mathcal{Y}]\!]$ *un c-*$\mathcal{I}_s$*-résolvent de* $E_1, \ldots E_n, C$*. Soit* $\theta \in Sol(\mathcal{Y})$*.* $\theta(R)$ *est un* $\mathcal{I}_s$*-résolvent de* $\theta(E_1), \ldots, \theta(E_n), \theta(C)$*.*

PREUVE. Puisque $E_1, \ldots E_n, C$ est un c-clash, $\theta \in Sol(\mathcal{Y}), \theta(E_1), \ldots, \theta(E_n), \theta(C)$ est un clash (au sens classique). En effet, par définition de $\mathcal{Y}$, $\mathcal{I}_s \not\models R$, et $\forall i \, \mathcal{I}_s \not\models E_i$, et $\theta(L_i(\overline{t}_i)) = \theta(L_i(\overline{s}_i))$. Donc, $\theta(R)$ est un $\mathcal{I}_s$-résolvent de $\theta(E_1), \ldots, \theta(E_n), \theta(C)$. C.Q.F.D.

Ce théorème montre que toute c-dérivation sémantique correspond à une dérivation *fermée* sémantique. Ce théorème n'est pas valide pour la résolution sémantique sans contraintes. Afin de s'en convaincre, il suffit de considérer les deux clauses suivantes et l'interprétation $\{P(a)\}$.

$$c_1 : P(x) \vee Q(x)$$
$$c_2 : \neg P(x) \vee R(x, y)$$

En utilisant la résolution sémantique, nous pouvons inférer la clause $Q(x) \vee R(x, y)$. La clause $Q(a) \vee R(a, b)$ est une instance de $Q(x) \vee R(x, y)$, et *ne peut pas* être déduite d'une instance de c_1 ou c_2 par une dérivation sémantique fermée. Ainsi, il peut exister de nombreuses

dérivations sémantiques qui ne correspondent à aucune dérivation fermée. Notre méthode permet de résoudre ce problème d'une façon très naturelle. En effet, la c-$\mathcal{I}_s$-résolution génère la c-clause $c_3 : [\![Q(x) \vee R(x,y) \ : x \neq a]\!]$. $Q(a) \vee R(a,b)$ n'est pas une instance de c_3.

THÉORÈME 7.2.3. *Soit $\mathcal{I}_s$ une interprétation, S un ensemble de c-clauses et $[\![C \ : \mathcal{X}]\!]$ une c-clause déduite de S en utilisant les règles de c-réfutation. Pour toute substitution $\sigma \in \mathcal{S}(\mathcal{X})$, il existe une dérivation fermée de $C\sigma$ à partir de S.*

PREUVE. La preuve est une conséquence immédiate du lemme 7.2.3 (par induction sur la longueur n de la dérivation). C.Q.F.D.

Ce théorème est un effet de bord intéressant de notre approche. Il montre que la méthode permet de restreindre l'espace de recherche d'une façon plus stricte que la résolution sémantique classique. L'exemple suivant illustre cette possibilité.

$$c_1 \ : P(x) \vee S(x) \qquad c_2 \ : \neg P(x) \qquad c_3 \ : \neg S(a) \vee R(y)$$

EXEMPLE 7.2.1. L'interprétation est

$$\mathcal{I}_s = \{S(a)\}.$$

La dérivation suivante est une dérivation au sens de SLAGLE.

$c_4 \ : S(x)$ (résolution entre c_1 et c_2)
$c_5 \ : R(y)$ (résolution entre c_4 et c_3)

Notre méthode produit les c-clauses suivantes.

$c_4 \ : [\![S(x) \ : x \neq a]\!]$ ($\mathcal{I}_s$-c-res,c_1,c_2)
$c_5 \ : [\![R(y) \ : x \neq a \wedge x = a]\!]$ ($\mathcal{I}_s$-c-res,c_4,c_3)

Les contraintes de c_5 n'ont pas de solution, donc c_5 est une tautologie qui peut être supprimée de l'ensemble de c-clauses. ◇

Ce problème est identifié dans (SANDFORD, 1980), où une méthode est proposée afin d'éviter des inférences qui ne correspondent à aucune dérivation sémantique fermée. Son principe est d'associer à chaque clause une liste de littéraux dont les descendants doivent être falsifiés (appelée la liste FSL). Cette méthode semble cependant peu efficace, d'après les expérimentations rapportées dans (MCCUNE ET HENSCHEN, 1985).

*"The FSL method was implemented but during preliminary experiments we found that it required too much processing time. During deep searches the FSL lists became very long and the improvement was small or non-existent." ((*MCCUNE ET HENSCHEN, *1985), page 256)*

Notre approche permet de coder les conditions sémantiques de façon plus compacte, et semble être mieux adaptée pour résoudre ce problème d'une façon plus naturelle et plus efficace.

7.2.4 Une nouvelle règle de construction de modèles

La disrésolution sémantique

Nous pouvons appliquer à la règle de résolution définie à la section précédente l'idée proposée dans (CAFERRA ET ZABEL, 1992), pour définir une nouvelle règle : la *disrésolution sémantique*. Le principe de cette règle est d'imposer des conditions *empêchant* l'application de la règle de c-résolution sémantique. De façon plus précise, chaque c-clause $C_1, \ldots C_n$ appartenant au c-clash sera remplacée par deux c-clauses C'_i, C''_i, telles que $C_i \equiv C'_i \wedge C''_i$, et telles que la règle de c-résolution ne peut pas être appliquée entre $C''_1, \ldots C''_n$, et peut être appliquée entre $C'_1, \ldots, C'_n$.

DÉFINITION 7.2.2.

Soit $S = [\![C_1 \ : \mathcal{X}_1]\!], \ldots, [\![C_n \ : \mathcal{X}_n]\!]$ un c-clash. Soit $\overline{x}$ l'ensemble des variables de S. Les c-disrésolvents de S sont les c-clauses $\bigcup_{i=1}^{n} \{C'_i, C''_i\}$, où C'_i et C''_i sont définies comme suit.

$$C'_i : [\![C_i \ : \forall \overline{y}.\neg \mathcal{Y}_{res} \wedge \mathcal{X}_i]\!]$$
$$C''_i : [\![C_i \ : \exists \overline{y}.\mathcal{Y}_{res} \wedge \mathcal{X}_i]\!]$$

avec $\overline{y} = \overline{x} \setminus \mathcal{V}ar([\![C_i \ : \mathcal{X}_i]\!])$

$\diamond$

THÉORÈME 7.2.4. *Soit $C_1, \ldots, C_n$ un c-clash. Soit $\{C'_1, C''_1, \ldots, C'_n, C''_n\}$ les c-clauses déduites de $C_1, \ldots C_n$ par la règle de disrésolution. Pour tout $i \leq n$, $C_i \equiv C'_i \wedge C''_i$*

PREUVE. La preuve est immédiate. C.Q.F.D.

Nous notons RAMC$_{\mathcal{I}_s}$ la procédure définie par les règles renommage, dissubsomption, factorisation, disfactorisation, distautologie, GPL, GMPL, $\mathcal{I}_s$-c-résolution sémantique et $\mathcal{I}_s$-c-disrésolution sémantique.

REMARQUE. La procédure RAMC définie au chapitre 6 peut être considérée comme un cas particulier de RAMC$_{\mathcal{I}_s}$ avec $\mathcal{I}_s = \emptyset$.

7.2.5 Construction incrémentale du modèle

L'une des caractéristiques les plus intéressantes de notre approche est de permettre une *construction incrémentale* du modèle. C'est particulièrement important lorsque l'ensemble de c-clauses est d'une taille très importante et utile lorsque l'ajout des nouvelles c-clauses ne modifie pas sensiblement l'interprétation de l'ensemble. Le théorème suivant fournit des conditions permettant d'extraire un modèle d'un ensemble de c-clauses obtenu lorsque RAMC$_{\mathcal{I}_s}$ s'arrête. Ces conditions sont similaires — mais *plus faibles* — que les conditions correspondantes du chapitre 6. En effet, nous pouvons tirer parti de l'interprétation $\mathcal{I}_s$ (qui constitue un modèle partiel de l'ensemble de c-clauses) afin de guider la recherche du modèle.

DÉFINITION 7.2.3. *[Construction incrémentale des interprétations] Soit $\mathcal{I}$ une interprétation de Herbrand et E un ensemble satisfaisable de littéraux fermés. Alors on note $\mathfrak{E}_{\mathcal{I}}(E)$ l'interprétation définie de la façon suivante : pour tout littéral fermé $P(\overline{t})$,*

- $\mathfrak{E}_{\mathcal{I}}(E)(P(\overline{t})) =$ *true si* $P(\overline{t}) \in E$.
- $\mathfrak{E}_{\mathcal{I}}(E)(P(\overline{t})) =$ *false si* $\neg P(\overline{t}) \in E$.
- $\mathfrak{E}_{\mathcal{I}}(E)(P(\overline{t})) = \mathcal{I}(P(\overline{t}))$ *sinon.*

$\diamond$

Informellement, $\mathfrak{E}_{\mathcal{I}}(E)$ coïncide avec $\mathcal{I}$ sauf sur les littéraux de E pour lesquels la valeur de l'interprétation est fixée par E.

REMARQUE. Si $\mathcal{I}$ et E sont des eq-ensembles, alors $\mathfrak{E}_{\mathcal{I}}(E)$ est une eq-interprétation. En effet on a $\mathfrak{E}_{\mathcal{I}}(E) = (\mathcal{I} \setminus \{x / \neg x \in E\}) \cup E$.

THÉORÈME 7.2.5. *Soit S un ensemble de c-clauses irréductible par toutes les règles de* RAMC$_{\mathcal{I}_s}$. *Si toutes les c-clauses non unitaires de S sont* **vraies** *dans $\mathcal{I}_s$, alors un modèle de S peut être construit automatiquement.*

PREUVE. Soit S un ensemble satisfaisable de c-clauses, tel que toute c-clause non unitaire de S est **vraie** dans S. Considérons l'interprétation $\mathcal{I}' = \mathfrak{E}_{\mathcal{I}_s}(\mathit{Unit}(S))$. Nous prouvons que $\mathcal{I}' \models S$.

Pour toute c-clause $c : [\![\bigvee_{i=1}^{n} L_i(\overline{t_i}) \ : \mathcal{X}]\!] \in S$ l'une des deux conditions suivantes est vérifiée.

- $n = 1$ **(C est une c-clause unitaire).** Alors $\forall \sigma \in \mathit{Sol}(\mathcal{X}).\mathcal{I}' \models L_i(\sigma(\overline{t_i}))$ (par définition de $\mathcal{I}'$).

– $n > 1$. Alors C est valide dans $\mathcal{I}'$. Soit $\sigma \in Sol(\mathcal{X})$. Puisque C est stable par distautologie, il existe i tel que $\sigma(L_i(\overline{t_i}))$ est **vraie** dans $\mathcal{I}_s$. Supposons sans perte de généralité, que $\forall i \leq q$, $I_s \models \sigma(L_i(\overline{t}_i))$ et $\forall i > q$, $\mathcal{I}_s \models \neg\sigma(L_i(\overline{t}_i))$. Alors par irréductibilité par dis-résolution $\exists i \leq q$, tel que $\neg L_i(\overline{t}_i) \notin S$. Par définition de $\mathcal{I}'$, on en déduit $\mathcal{I}' \models \sigma(L_i(\overline{t_i}))$, d'où

$$\mathcal{I}' \models C$$

C.Q.F.D.

REMARQUE. La construction de $\mathcal{I}'$ est *incrémentale*, car elle est réalisée en *modifiant* et en *étendant* l'interprétation $\mathcal{I}_s$.

Exemple

L'exemple suivant tiré de (FERMÜLLER ET AL., 1993), illustre cet aspect de la méthode.

$$\begin{array}{l}
1\ [\![P(x) \vee Q(g(x,x)) \ : \top]\!] \\
2\ [\![\neg Q(x) \vee R(y,x) \ : \top]\!] \\
3\ [\![\neg R(a,a) \vee \neg R(f(b),a) \ : \top]\!] \\
4\ [\![R(f(x),y) \ : \top]\!] \\
5\ [\![P(x) \vee \neg P(f(x)) \ : \top]\!] \\
6\ [\![\neg P(a) \ : \top]\!]
\end{array}$$

RAMC construit automatiquement le modèle $\mathcal{I}_s$ suivant.

$$\begin{array}{l}
[\![R(f(x),y) \ : \top]\!] \\
[\![Q(g(x,x)) \ : \top]\!] \\
[\![\neg R(a,a) \ : \top]\!] \\
[\![\neg Q(a) \ : \top]\!] \\
[\![\neg P(f(x)) \ : \top]\!] \\
[\![\neg P(a) \ : \top]\!] \\
[\![R(x,y) \ : (a \neq y) \vee (a \neq x)]\!]
\end{array}$$

Supposons que l'on ajoute ensuite les c-clauses suivantes.

$$\begin{array}{l}
7\ [\![\neg S(x) \vee S(g(x,x)) \ : \top]\!] \\
8\ [\![S(a) \vee S(g(x,x)) \ : \top]\!] \\
9\ [\![\neg S(x) \vee \neg R(b,g(x,x)) \ : \top]\!]
\end{array}$$

RAMC$_{\mathcal{I}_s}$ génère les c-clauses.

$10\ [\![\neg S(x) \vee \neg R(b,g(g(x,x),g(x,x))) \ : \top]\!]$	résolution,9,7
$11\ [\![\neg S(x) \vee \neg Q(g(g(x,x),g(x,x))) \ : \top]\!].$	résolution,10,2
$12\ [\![\neg S(x) \vee P(g(x,x)) \ : \top]\!]$	résolution,11,1
$13\ [\![P(g(x,x)) \ : \top]\!]$	GPL
$14\ [\![P(x) \vee Q(g(x,x)) \ : \forall u.x \neq g(u,u)]\!]$	dissubsomption,13,1
$15\ [\![\neg Q(g(g(x,x),g(x,x))) \ : \top]\!]$	GPL
$16\ [\![\neg Q(x) \vee R(y,x) \ : \forall y.x \neq g(g(y,y),g(y,y))]\!]$	dissubsomption,15,2
$17\ [\![\neg R(b,g(g(x,x),g(x,x))) \ : \top]\!]$	GPL
$18\ [\![\neg S(x) \vee \neg R(b,g(x,x)) \ : \forall y.x \neq g(y,y)]\!]$	dissubsomption,17,9
$19\ [\![S(g(x,x)) \ : \top]\!]$	GPL
$20\ [\![\neg S(x) \ : \forall y.x \neq g(y,y)]\!]$	GPL

Les c-clauses $\{7, 8, 10, 11, 12, 18\}$ restantes sont supprimées par dissubsomption. Nous obtenons un ensemble de c-clauses $\{4, 6, 8, 13, 15, 17, 19, 3, 5, 14, 16, 20\}$ satisfaisant les conditions de la section 7.2.5. Le modèle est le suivant.

C-littéraux obtenus en modifiant le modèle initial :

$$[\![R(f(x), y) \ : \top]\!]$$
$$[\![Q(g(x, x)) \ : \forall y.x \neq g(y, y)]\!]$$
$$[\![\neg R(a, a) \ : \top]\!]$$
$$[\![\neg Q(a) \ : \top]\!]$$
$$[\![\neg P(f(x)) \ : \top]\!]$$
$$[\![\neg P(a) \ : \top]\!]$$
$$[\![R(x, y) \ : ((a \neq y) \vee (a \neq x)) \wedge ((x \neq b) \vee \forall u.y \neq g(g(u, u), g(u, u)))]\!]$$

Nouveaux c-littéraux :

$$[\![P(g(x, x)) \ : \top]\!]$$
$$[\![\neg Q(g(g(x, x), g(x, x))) \ : \top]\!]$$
$$[\![\neg R(b, g(g(x, x), g(x, x))) \ : \top]\!]$$
$$[\![S(g(x, x)) \ : \top]\!]$$
$$[\![\neg S(x) \ : \forall y.x \neq g(y, y)]\!]$$

7.3 Les classes $\mathfrak{C}_{\text{gen}}$, $\mathfrak{C}_{\text{eq-model}}$ et $\mathfrak{C}_{\text{atomic}}$

7.3.1 Classe des formules admettant un eq-modèle

DÉFINITION 7.3.1. *Soit $\mathfrak{C}_{eq\text{-}model}$ la classe des formules possédant un eq-modèle sur la signature Σ.* ◇

Une sous-classe particulière de $\mathfrak{C}_{\text{eq-model}}$ doit être mentionnée.

DÉFINITION 7.3.2. *Une formule équationnelle $\mathcal{F}$ est dite* positive *si et seulement si elle ne contient aucune négation.* ◇

DÉFINITION 7.3.3. *Une eq-interprétation totale $\mathcal{M}$ est appelée une interprétation atomique si et seulement si il existe une partition Ω^+, Ω^- de Ω telle que pour tout $P \in \Omega^+$, $\mathcal{F}^+(P)$ est positive et pour tout $P \in \Omega^-$, $\mathcal{F}^-(P)$ est positive. On note $\mathfrak{C}_{atomic}$ l'ensemble des formules satisfaisables admettant un modèle atomique.* ◇

Les interprétations atomiques sont des interprétations représentables par des ensembles finis de littéraux (non fermés, non linéaires et sans contraintes). En effet, il est clair que la forme normale d'un ensemble de c-clauses dont les contraintes sont positives est un ensemble de clauses (c'est-à-dire un ensemble de c-clauses dont les contraintes sont $\top$). Elles sont utilisées dans (FERMÜLLER ET LEITSCH, 1996) pour représenter et construire des modèles de certaines classes de formules décidables par hyperrésolution.

EXEMPLE 7.3.1. L'eq-interprétation $\{R(x, x), [\![\neg R(x, y) \ : x \neq y]\!]\}$ est atomique. L'eq-interprétation $\{[\![R(x, x, y) \ : x \neq y]\!], [\![\neg R(x, y, z) \ : x \neq y \vee z = y]\!]\}$ n'est *pas* atomique. ◇

7.3.2 La classe $\mathfrak{C}_{\text{gen}}$

Introduisons la notion de c-littéral générable. Informellement, un c-littéral est dit *générable* à partir d'un ensemble de c-clauses donné, s'il peut être obtenu en appliquant les règles de $\text{RAMC}_{\mathcal{I}_s}$.

DÉFINITION 7.3.4. *Un c-littéral L est dit $\mathcal{I}_s$-n-*générable *à partir d'un ensemble de c-clauses S si et seulement si il existe une dérivation de longueur n à partir de S vers S' utilisant les règles de $\text{RAMC}_{\mathcal{I}_s}$ et un ensemble de c-littéraux $L_1, \ldots, L_n$ vérifiant les conditions suivantes.*

– $L \equiv L_1 \wedge \ldots \wedge L_n$.

– *Pour tout $i \leq n$, il existe une c-clause $[\![P \vee R \ : \mathcal{X}]\!] \in S'$ telle que $L_i \equiv [\![P \ : \mathcal{X}]\!]$.*

*Un c-littéral L est $\mathcal{I}_s$-*générable *si et seulement s'il est $\mathcal{I}_s$-n-générable pour un certain n. Une eq-interprétation $\mathcal{M}$ est dite $\mathcal{I}_s$-*générable *si elle admet une représentation sous forme d'un ensemble fini de c-littéraux $\mathcal{I}_s$-générables.* ◇

EXEMPLE 7.3.2. Soit S l'ensemble de c-clauses $\{[\![P(x) \vee Q(x) \ : \top]\!], \neg P(a)\}$. Soit $\Sigma = \{a, b, c\}$ (où a, b, c dénotent des symboles de constantes). L'interprétation $\{\neg P(a), [\![P(x) \ : x \neq a]\!], Q(a)\}$ est générable à partir de S. En effet, en appliquant la règle de c-disrésolution, on obtient la c-clause $[\![P(x) \vee Q(x) \ : x \neq a]\!]$, qui contient le littéral $[\![P(x) \ : x \neq a]\!]$.

L'interprétation $\{[\![\neg R(x) \ : x \neq b]\!], [\![P(x) \ : x = b]\!], [\![Q(x) \ : x \neq b]\!]\}$ est également un modèle de S, mais ce modèle n'est pas générable, car les c-littéraux $[\![P(x) \ : x = b]\!]$ et $[\![Q(x) \ : x \neq b]\!]$ ne peuvent pas être obtenus en utilisant les règles de RAMC. ◇

DÉFINITION 7.3.5. *Soit $\mathfrak{C}_{gen}$ l'ensemble des ensembles de c-clauses S tel que S a un eq-modèle S-générable.* ◇

DÉFINITION 7.3.6. *Soit une stratégie $\mathfrak{S}$ d'application des règles de* RAMC. *$\mathfrak{S}$ est dit* équitable pour la construction de modèle *si pour tout ensemble de c-clauses S et pour toute eq-interprétation $\mathcal{I}_s$ et pour tout c-littéral L $\mathcal{I}_s$-générable à partir de S, toute application des règles de* RAMC *suivant $\mathfrak{S}$ conduit à un ensemble de c-clauses S' dont les c-clauses contiennent n c-littéraux $L_1, \ldots, L_n$ tels que :*

$$L \equiv L_1 \wedge \ldots \wedge L_n.$$

◇

Notons que cette notion d'équité est plus forte que la notion habituelle pour les procédures de preuve. Considérons par exemple l'ensemble de c-clauses suivant.

EXEMPLE 7.3.3.

$$\begin{array}{l} P(x,x) \vee R \\ R \\ \neg P(x,y) \vee Q \end{array}$$

Le littéral $[\![\neg P(x,y) \ : x \neq y]\!]$ est générable (il suffit d'appliquer la c-disrésolution entre les c-clauses 1 et 3). Or, si la c-dissubsomption est appliquée préalablement sur la c-clause 1, elle est réduite à $\top$ par résolution des contraintes, donc supprimée de l'ensemble de c-clauses. $[\![\neg P(x,y) \ : x \neq y]\!]$ n'est plus générable à partir de l'ensemble obtenu. ◇

REMARQUE. Dans (CAFERRA ET PELTIER, 1996a), est proposée une stratégie particulière utilisant des règles de RAMC pour construire de façon systématique tout eq-modèle générable.

La figure 7.1 illustre les différentes classes de formules introduites dans cette section.

7.3.3 Limites des règles d'inférence et de disinférence

De façon évidente, on a $\mathfrak{C}_{\text{gen}} \subseteq \mathfrak{C}_{\text{eq-model}}$ et $\mathfrak{C}_{\text{atomic}} \subseteq \mathfrak{C}_{\text{eq-model}}$. De plus, il est clair qu'il existe des ensembles de c-clauses admettant un eq-modèle mais aucun eq-modèle atomique, donc $\mathfrak{C}_{\text{atomic}} \subset \mathfrak{C}_{\text{eq-model}}$. Par définition, si RAMC se termine sur $\mathcal{S}$ et génère un modèle de $\mathcal{S}$, alors $\mathcal{S} \in \mathfrak{C}_{\text{gen}}$. La question se pose alors de savoir si la réciproque est vraie, c'est-à-dire si tout ensemble de c-clauses admettant un eq-modèle possède également un eq-modèle générable. Un tel résultat serait particulièrement satisfaisant car il fournirait une caractérisation sémantique précise de la classe de formules décidable par la méthode RAMC. Malheureusement, la réponse à cette question est négative : l'exemple suivant montre qu'il existe des ensembles de c-clauses

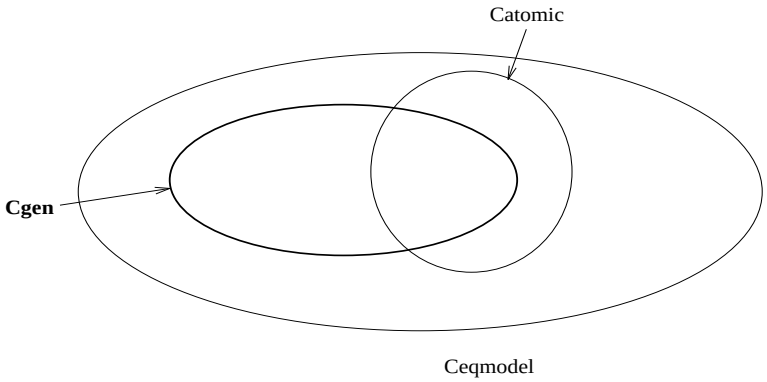


FIG. 7.1 - : *Les classes* $\mathfrak{C}_{eq\text{-}model}$, $\mathfrak{C}_{atomic}$ *et* $\mathfrak{C}_{gen}$

appartenant à la classe $\mathfrak{C}_{\text{eq-model}}$, mais dont le modèle *ne peut être obtenu* en appliquant la méthode RAMC.

EXEMPLE 7.3.4. Soit S l'ensemble de c-clauses défini comme suit.

$$\begin{array}{l}[\![\neg P(f(x),a) \ : \top]\!] \\ [\![\neg P(a,f(x)) \ : \top]\!] \\ [\![P(a,a) \ : \top]\!] \\ [\![P(x,y) \vee \neg P(f(x),f(y)) \ : \top]\!] \\ [\![\neg P(x,y) \vee P(f(x),f(y)) \ : \top]\!]\end{array}$$

De façon évidente, ces clauses définissent l'identité syntaxique sur $\tau(\Sigma)$. S n'admet qu'un unique modèle de Herbrand.

$$P(x,y) \text{ est vrai si et seulement si } x = y.$$

Il est facile de voir que les c-littéraux $P(x,x)$ et $[\![\neg P(x,y) \ : x \neq y]\!]$ ne sont pas générables. Donc $\mathcal{M}$ n'est pas générable. D'où

$$\mathfrak{C}_{\text{eq-model}} \neq \mathfrak{C}_{\text{gen}}$$

REMARQUE. $P(x,x)$ et $[\![\neg P(x,y) \ : x \neq y]\!]$ sont des conséquences *inductives* de S, mais pas des conséquences logiques de S.

◇

L'exemple précédent montre que les règles d'inférence et de disinférence ne sont pas suffisantes pour générer tous les eq-modèles possibles. On a également $\mathfrak{C}_{\text{atomic}} \not\subset \mathfrak{C}_{\text{gen}}$, comme le montre l'exemple suivant.

EXEMPLE 7.3.5. Soit S l'ensemble de c-clauses suivant.

$$\begin{array}{l}[\![P(a,a) \ : \top]\!] \\ [\![\neg P(f(x),a) \ : \top]\!] \\ [\![\neg P(x,y) \vee P(x,f(y)) \ : \top]\!] \\ [\![P(x,y) \vee \neg P(x,f(y)) \ : \top]\!]\end{array}$$

Là encore, S admet un unique modèle de Herbrand sur la signature considérée.

$$[\![P(a,x) \ : \top]\!], [\![\neg P(f(y),x) \ : \top]\!]$$

L'application des règles de RAMC génère les c-littéraux:

$$P(a, f(a)), P(a, f(f(a))), \ldots, P(a, f^n(a))$$

et

$$\neg P(f(x), f(a)), \neg P(f(x), f(f(a))), \ldots, \neg P(f(x), f^n(a)),$$

mais pas les littéraux $P(a, x)$ et $\neg P(f(x), x)$. ◇

Est-il possible de surmonter ces limitations? Une façon de procéder serait d'ajouter aux règles de RAMC de nouvelles règles permettant de génèrer les littéraux qui ne peuvent pas être obtenus par l'application des règles d'inférence ou de disinférence. Nous pouvons évidemment utiliser la règle **coupure** (voir chapitre 6) qui permet d'engendrer n'importe quel c-littéral. Néanmoins une telle approche ne semble pas réaliste, en l'absence de stratégies pour guider le choix des clauses et des formules sur lesquelles appliquer la règle. Elle conduirait en effet à une augmentation du nombre de c-clauses que l'on ne pourrait traiter en pratique. Nous n'avons donc pas cherché à approfondir cette voie de recherche, préférant nous intéresser à d'autres types de méthodes de construction de eq-modèles. Elles sont présentées dans les chapitres suivants.

Chapitre 8

EQMC : construction de eq-modèles par énumération

Les résultats du chapitre 7 montrent les limites de la méthode RAMC. Elles sont dues à l'existence d'ensembles de c-clauses admettant un eq-modèle, *mais ne possédant aucun eq-modèle générable*, c'est-à-dire aucun eq-modèle pouvant être obtenu par application des règles d'inférence ou de disinférence (voir exemples 7.3.4 et 7.3.5, chapitre 7). Dans ce chapitre, nous proposons une méthode permettant de surmonter ces limites.

Il est clair que l'ensemble des eq-interprétations est récursivement énumérable. D'autre part, il est prouvé au chapitre 5 que le problème de la validité d'une formule du premier ordre dans une eq-interprétation (totale) est décidable. Par conséquent, une méthode évidente et systématique pour construire un eq-modèle d'une formule donnée consiste à énumérer toutes les eq-interprétations sur la signature considérée et à tester si l'une d'entre elles satisfait la formule considérée. Cette approche permet de traiter toute formule possédant un eq-modèle. Ainsi les capacités de la méthode ne dépendent plus, à la différence de la méthode RAMC, des propriétés *syntaxiques* de la formule, mais uniquement de propriétés *sémantiques* (i.e. l'appartenance à la classe $\mathfrak{C}_{\text{eq-model}}$). Naturellement, une telle énumération est particulièrement coûteuse, et s'avère inutilisable en pratique. Par conséquent, il est nécessaire de réduire l'espace de recherche, en éliminant certaines interprétations et en guidant le choix des interprétations à considérer. Pour cela, il est naturel de chercher à *combiner* les deux approches, par énumération et déduction. Dans ce chapitre est proposée une méthode, nommée EQMC, fondée sur cette idée. Elle sera décrite en trois étapes.

- Dans un premier temps, nous nous intéresserons au processus d'énumération des eq-interprétations dont nous donnerons une définition formelle. Pour cela, nous supposerons donnée une procédure *Deduce* dont nous préciserons les spécifications.

- Dans un second temps, nous définirons plus précisément la procédure *Deduce*. Informellement, la procédure correspond à la partie *déductive* de la méthode. Son but est double. D'une part, elle est utilisée pour détecter qu'une interprétation donnée valide la formule, et d'autre part, elle réduit l'espace de recherche en détectant les contre-modèles et en générant des conséquences *ou des non-conséquences* de la formule initiale.

- Enfin, nous conclurons en donnant quelques exemples d'heuristiques permettant de guider l'application des règles de la procédure EQMC.

8.1 Enumération des eq-interprétations

8.1.1 Ensembles de représentation

Introduisons tout d'abord le concept d'*ensemble de représentation*

DÉFINITION 8.1.1. *Un ensemble de c-littéraux positifs $\mathcal{D}$ est appelé un* ensemble de représentation *si et seulement si les conditions suivantes sont vérifiées.*

- *Pour tout couple $(L_1, L_2) \in \mathcal{D}^2$, on a $L_1 \cap L_2 = \emptyset$.* Cette condition signifie que les ensembles d'atomes fermés dénotés par deux c-littéraux distincts de $\mathcal{D}$ sont disjoints deux à deux.
- *Pour tout littéral fermé positif L_g il existe un c-littéral $L \in \mathcal{D}$ tel que $L_g \subseteq L$.* Cette condition signifie que tous les littéraux fermés peuvent être capturés par $\mathcal{D}$.

◇

Ces conditions assurent qu'un ensemble de représentation $\mathcal{D}$ définit une *partition* de la base de Herbrand sur la signature Σ, Ω. Il est alors possible de représenter certaines eq-interprétations en donnant une valeur de vérité à chaque élément de la partition. Cela n'est possible que si l'interprétation considérée est *compatible* avec $\mathcal{D}$ c'est-à-dire si deux atomes appartenant à un même élément de $\mathcal{D}$ ont même valeur de vérité.

DÉFINITION 8.1.2. *Une c-clause unitaire C est dite* compatible *avec un ensemble de c-littéraux $\mathcal{D}$ (ou $\mathcal{D}$-*compatible*) si et seulement si pour tout c-littéral L tel que $L \in \mathcal{D}$ ou $\neg L \in \mathcal{D}$ on a : si $L \cap C \neq \emptyset$, alors $L \subseteq C$. Un ensemble de c-clauses unitaires est dit $\mathcal{D}$-*compatible *si et seulement si toute c-clause de S est $\mathcal{D}$-compatible. Une eq-interprétation $\mathcal{M}$ est dite $\mathcal{D}$-*compatible *si et seulement s'il existe une représentation de S (voir chapitre 5) par un ensemble de c-clauses $\mathcal{D}$-compatibles.* ◇

EXEMPLE 8.1.1.

$$\mathcal{D} = \{P(x,y), Q(a), Q(b), Q(f(x,x)), [\![Q(f(x,y)) \ : x \neq y]\!]\}$$

est un ensemble de représentation sur la signature $\Sigma = \{a, b, f\}$ et $\Omega = \{P, Q\}$. L'interprétation

$$\mathcal{I} = \{P(x,y), Q(b), [\![\neg Q(x) \ : x \neq b]\!]\}$$

est $\mathcal{D}$-compatible. En revanche, l'ensemble de c-clauses

$$\mathcal{J} = \{P(a,y), [\![\neg P(x,y) \ : x \neq a]\!], Q(b), [\![\neg Q(x) \ : x \neq b]\!]\}$$

n'est pas $\mathcal{D}$-compatible. En effet, on a, par exemple, $P(x,y) \cap P(a,x) \neq \emptyset$ et $P(x,y) \not\subseteq P(a,x)$. En outre, il est clair que l'eq-interprétation dénotée par $\mathcal{J}$ n'est pas $\mathcal{D}$-compatible. ◇

DÉFINITION 8.1.3. *Un ensemble de c-littéraux $\mathcal{D}$ est dit* compatible *avec une formule $\mathcal{F}$ si et seulement s'il existe un modèle de $\mathcal{F}$ contenant $\mathcal{D}$ (i.e. si $\mathcal{D}$ est une eq-interprétation partielle et $\mathcal{D} \not\models \neg\mathcal{F}$).* ◇

Génération des ensembles de représentation

La génération des ensembles de représentation est réalisée en utilisant les règles suivantes, appliquées sur n'importe quel ensemble de représentation.

GR_Σ $\mathcal{D} \cup \{[\![L \ : \mathcal{X}]\!]\} \rightarrow \mathcal{D} \cup \{[\![L \ : \mathcal{X} \wedge x = f(\overline{z})]\!] / f : \overline{s} \rightarrow s \in \Sigma, sort(x) = s\}$
Si il existe au moins deux symboles f tels que
$\mathcal{X} \wedge x = f(\overline{z})$ est satisfaisable,
et si $x \in \mathcal{V}ar([\![L \ : X]\!])$, et $\overline{z}$ sont de nouvelles variables.

$\mathrm{GR}_=$ $\mathcal{D} \cup \{[\![L \ : \mathcal{X}]\!]\} \rightarrow \mathcal{D} \cup \{[\![L \ : \mathcal{X} \wedge t = s]\!], [\![L \ : \mathcal{X} \wedge t \neq s]\!]\}$
si $\mathcal{X} \wedge t \neq s \not\equiv \bot$ et $\mathcal{X} \wedge t = s \not\equiv \bot$,
si $t, s < [\![L \ : \mathcal{X}]\!]$ et si $sort(t) = sort(s)$.

On note GR le système $\{\mathrm{GR}_\Sigma, \mathrm{GR}_=\}$.

EXEMPLE 8.1.2. [Utilisation du système GR] Soit $\Omega = \{P\}$, $\Sigma = \{a, f\}$, et $\mathcal{D}_0 = \{P(x)\}$. Appliquons la règle GR_Σ sur $\mathcal{D}_0$, en choisissant la variable x. On obtient (après simplification) l'ensemble $\mathcal{D}_1 = \{P(a), P(f(u,v))\}$. Appliquons ensuite la règle $\mathrm{GR}_=$ sur les termes u et v. On obtient (après simplification) : $\mathcal{D}_2 = \{P(a), P(f(u,u)), [\![P(f(u,v)) \ : u \neq v]\!]\}$.

$\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2$ sont des ensembles de représentation. ◇

Soit $\mathcal{D}_0$ un ensemble de représentation quelconque. Nous montrons ci-dessous que, pour toute eq-interprétation $\mathcal{I}$, toute application des règles de GR sur $\mathcal{D}_0$ conduit à un ensemble $\mathcal{D}$ tel que $\mathcal{I}$ est $\mathcal{D}$-représentable. Cette propriété n'est pas vraie en général. Nous devons imposer une condition supplémentaire sur la stratégie d'application de ces règles, afin de garantir que ces règles sont appliquées de façon équitable.

DÉFINITION 8.1.4. *Soit s une stratégie d'application des règles de GR. s est dit* équitable *si et seulement si, pour toute séquence infinie, $\mathcal{D}_1 \rightarrow_s \mathcal{D}_2 \rightarrow_s \ldots \rightarrow_s \mathcal{D}_n \rightarrow_s \ldots$ et pour tout $L_1 \in \mathcal{D}_1$, on a :*

- *Si la règle $GR_=$ est applicable sur L_1 avec deux termes t et s, il existe i tel que, pour tout $L_i \in \mathcal{D}_i$ la règle $GR_=$ n'est pas applicable sur L_i entre t et s.*
- *Si la règle GR_Σ est applicable sur L_1 avec une variable x, il existe i tel que, pour tout $L_i \in \mathcal{D}_i$, GR_Σ n'est pas applicable sur L_i avec la variable x.*

◇

EXEMPLE 8.1.3. Par exemple, la stratégie consistant à appliquer les règles de GR sur des termes apparaissant à des profondeurs minimales est équitable. ◇

THÉORÈME 8.1.1. *Soit $(\mathcal{D}_i)_{i\in\mathbb{N}}$ une dérivation équitable des règles $GR_=, GR_\Sigma$. Soit $\mathcal{I}$ une eq-interprétation. Il existe $k \in \mathbb{N}$ tel que $\mathcal{I}$ est $\mathcal{D}_k$-représentable.*

La preuve du théorème 8.1.1 nécessite quelques définitions et lemmes supplémentaires.

DÉFINITION 8.1.5. *Soit $n \in \mathbb{N}$. Un c-littéral $[\![P \ : \mathcal{X}]\!]$ est dit* n-maximal *si et seulement si les conditions suivantes sont satisfaites.*

- *Pour toute position p de P telle que $|p| \leq n$, $P_{|p}$ existe et n'est pas une variable.*
- *Pour tout couple de positions p_1, p_2 telles que $|p_1| \leq n$ $|p_2| \leq n$, $P_{|p_1} = P_{|p_2} \wedge \mathcal{X} \wedge \mathcal{Y}$ est soit $\top$ soit $\bot$.*

On note $\mathcal{D}_n$ l'ensemble de tous les littéraux n-maximaux distincts. ◇

Les ensembles de littéraux n-maximaux possèdent une propriété intéressante : **toute eq-interprétation est $\mathcal{D}_n$-compatible pour un certain n.** La preuve de cette assertion utilise le lemme suivant.

LEMME 8.1.1. *Soit $[\![L(\overline{t}) \ : \mathcal{X}]\!]$ une c-clause en forme normale de profondeur inférieure à n. Soit $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ un littéral n-maximal. S'il existe une solution σ de $\mathcal{Y} \wedge \overline{t} = \overline{s} \wedge \mathcal{X}$ alors pour toute solution θ_1 de $\mathcal{Y}$, il existe une solution θ_2 de $\mathcal{X}$ telle $\overline{t}\theta_2 = \overline{s}\theta_1$.*

PREUVE. Sans perte de généralité, on suppose que $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ est en forme normale. Soit $\overline{x} = \mathcal{V}ar([\![L(\overline{t}) \ : \mathcal{X}]\!])$ et $\overline{y} = \mathcal{V}ar([\![L(\overline{s}) \ : \mathcal{Y}]\!])$. Supposons que $\overline{t} = \overline{s} \wedge \mathcal{X} \wedge \mathcal{Y}$ n'est pas équivalent à $\bot$. Soit θ_1 une solution de $\mathcal{Y}$. Supposons qu'il existe une solution σ de $\mathcal{X} \wedge \mathcal{Y}$ telle que $\overline{t}\sigma = \overline{s}\sigma$. Soit $x \in \overline{x}$. Soit q_1, q_2 deux positions de $L(\overline{t})$ telles que $L(\overline{t})_{|q_1} = L(\overline{t})_{|q_2} = x$. $\sigma \in \mathcal{S}(\mathcal{Y} \wedge L(\overline{s})_{|q_1} = L(\overline{s})_{|q_2})$. donc $\mathcal{Y} \wedge L(\overline{s})_{|q_1} = L(\overline{s})_{|q_2} \neq \bot$. Puisque $|q_1| \leq n$ et $|q_2| \leq n$, et puisque $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ est

n-maximal, $\mathcal{Y} \wedge L(\overline{s})_{|q_1} = L(\overline{s})_{|q_2}$ est soit $\top$ soit $\bot$, donc égal à $\top$. On note t_x un terme à une position q (arbitrairement choisie) dans $L(\overline{s})$ telle que $L(\overline{t})_{|q} = x$

Soit θ_2 une substitution de $\overline{x}$ définie par $x\theta_2 = t_x\theta_1$. Supposons que $\theta_2 \notin \mathcal{S}(\mathcal{X} \wedge \mathcal{Y} \wedge \overline{t} = \overline{s})$. Soit q une position de $L(\overline{t})$. $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ est n-maximal donc le terme à la position q dans $L(\overline{s})$ n'est pas une variable (car $|q| < n$). Donc $\overline{s} = \overline{t}$ est réduit par décomposition à soit $\bot$ (ce qui est impossible car $\overline{t} = \overline{s}$ a une solution) soit à une conjonction d'équations $x = t$ où $x \in \overline{x}$. Par conséquent, par définition de θ_2 et de t_x, on a $\theta_2 \in \mathcal{S}(\overline{t} = \overline{s}\theta_1 \wedge \mathcal{Y}\theta_1)$.

Soit x une variable de $\overline{x}$ telle que $\mathcal{X}$ contient une diséquation de la forme $x \neq s$. x apparait à une position q dans $L(\overline{t})$. Puisque $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ est n-maximal, $\mathcal{Y} \wedge t_x\theta_1 \neq s\theta_2$ est soit $\bot$ (impossible car σ est une solution) soit équivaut à une disjonction de diséquations de la forme $y\theta_2 \neq t'$ où y est une variable de s et t' un terme de $t_x\theta_1$, i.e. de la forme $t_y\theta_1 \neq t'$. $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ étant n-maximal, $t_y\theta_1 \neq t'$ est soit $\bot$, soit $\top$, donc est $\top$. D'où $\theta_2 \in \mathcal{S}(\mathcal{X})$.

Donc $\theta_2 \in \mathcal{S}(\mathcal{X} \wedge \overline{s} = \overline{t})$. C.Q.F.D.

LEMME 8.1.2. *Soit $\mathcal{I}$ une eq-interprétation. Il existe $n \in \mathbb{N}$ tel que pour tout ensemble n-maximal $\mathcal{D}$, $\mathcal{I}$ est $\mathcal{D}$-représentable.*

PREUVE. Sans perte de généralité, on suppose que $\mathcal{I}$ est en forme normale. Soit n la profondeur maximale des termes de $\mathcal{I}$. Nous allons montrer que $\mathcal{I}$ est $\mathcal{D}_n$-compatible. Soit $[\![L(\overline{s}) \ : \mathcal{Y}]\!]$ un littéral n-maximal. Supposons qu'il existe une substitution $\sigma \in \mathcal{Y}$ telle que $\mathcal{I} \models L(\overline{s})\sigma$. Par définition, il existe une c-clause $[\![L(\overline{t}) \ : \mathcal{X}]\!] \in \mathcal{I}$ telle que σ est solution de $\exists\overline{x}.\mathcal{X} \wedge \mathcal{Y} \wedge \overline{s} = \overline{t}$. D'après le lemme 8.1.1, on en déduit que pour toute solution θ_1 de $\mathcal{Y}$, il existe une solution θ_2 de $\mathcal{X}$ telle que $\overline{s}\theta_1 = \overline{t}\theta_2$. Par définition, $\mathcal{I} \models L(\overline{t})\theta_2$, donc $\mathcal{I} \models L(\overline{s})\theta_1$. C.Q.F.D.

PREUVE. [du théorème 8.1.1] D'après le lemme 8.1.2, il existe n tel que $\mathcal{I}$ est $\mathcal{D}$-représentable, pour tout ensemble n-maximal $\mathcal{D}$. Puisque le nombre de termes à une profondeur inférieure à n est fini, et puisque l'application des règles est équitable, il existe $k \in \mathbb{N}$ tel que la forme normale de $\mathcal{D}_k$ est n-maximale. Donc $\mathcal{I}$ est $\mathcal{D}_k$-représentable. C.Q.F.D.

8.1.2 Présentation de la méthode

Présentation informelle

Le principe de la méthode peut être résumé de la façon suivante.

- On choisit une partition *finie* de la base de Herbrand en eq-ensembles, définie par le choix d'un ensemble de représentation.
- On énumère l'ensemble des interprétations *compatibles* avec cette partition en choisissant une valeur de vérité pour chaque eq-ensemble de la partition.
- Si aucune eq-interprétation n'a pu être obtenue, le processus est réitéré en choisissant une partition plus fine de la base de Herbrand, obtenue en appliquant les règles $\mathrm{GR}_=$ et GR_Σ de façon équitable. Comme nous le verrons par la suite, l'application de ces règles peut être guidée par les informations déduites à l'étape précédente.

Le choix de la valeur de vérité de chaque élément de la partition est réalisé en utilisant une méthode similaire à celle de la procédure de (DAVIS ET PUTNAM, 1960) : elle consiste à décomposer un problème $\mathcal{F}$ en deux sous-problèmes obtenus en rajoutant à $\mathcal{F}$ les c-littéraux respectifs P et $\neg P$, où P appartient à l'ensemble de représentation. Ce processus est réitéré successivement sur chaque c-littéral de l'ensemble, de façon à énumérer l'ensemble des interprétations $\mathcal{D}$-compatibles. Pour réduire l'espace de recherche, il est nécessaire de détecter, aussitôt

que possible, qu'une interprétation partielle est un contre-exemple de la formule. Pour cela, nous utilisons une procédure *Deduce*, opérant sur les formules du premier ordre ou sur des ensembles de c-clauses, et permettant de :

- vérifier la compatibilité du modèle partiel avec la formule ;
- guider l'application de la règle de décomposition, en générant des conséquences ou des non-conséquences de la formule.

Lors de l'énumération des eq-interprétations, il faut tenir compte du fait que la procédure *Deduce ne préserve pas nécessairement l'équivalence*. Ainsi, il peut s'avérer indispensable de modifier, au cours de la recherche, l'ensemble de représentation utilisé, afin de prendre en compte les nouveaux c-littéraux générés par la procédure de déduction. Ce problème est illustré par l'exemple suivant.

EXEMPLE 8.1.4. Soit l'ensemble de c-clauses S suivant.

$$\begin{array}{l} [\![P(x) \vee Q(x) \ : x \neq b]\!] \\ P(a) \\ \neg P(b) \vee Q(x) \end{array}$$

Soit $\mathcal{D} = \{P(x), Q(x)\}$. S admet un eq-modèle $\mathcal{D}$-compatible $\mathcal{I} = \{P(x), Q(x)\}$. Cependant, si nous appliquons la règle GPL afin de simplifier l'ensemble S, nous obtenons la c-clause $\neg P(b)$. $S \cup \{\neg P(b)\}$ n'admet plus de eq-modèle $\mathcal{D}$-compatible. Cependant $S \cup \{\neg P(b)\}$ admet le eq-modèle $\{\neg P(b), [\![P(x) \ : x \neq b]\!], Q(x)\}$ qui est compatible avec l'ensemble $\{[\![P(x) \ : x \neq b]\!], Q(x), \neg P(b)\}$. ◇

La procédure EQMC

Introduisons en premier lieu la notation suivante.

Notation 8.1.1 *Soit $S = \{L_1, \ldots, L_n\}$ un ensemble de c-littéraux et L un c-littéral. On note $\Im_S(L)$ le littéral*

$$L \setminus \neg L_1 \setminus \ldots \setminus \neg L_n.$$

De façon similaire, pour tout ensemble de c-littéraux S', on note $\Im_S(S')$ l'ensemble $\{\Im_S(L)/L \in S'\}$.

Intuitivement $\Im_S(L)$ est le littéral obtenu en enlevant de l'ensemble de littéraux fermés dénoté par L tous les littéraux fermés dont le complémentaire est dans S.

Nous supposons donnée une procédure *Deduce* (à terminaison finie), dont les spécifications sont données par la figure 8.1. Intuitivement, les conditions sur la procédure *Deduce* garantissent les propriétés suivantes.

- La procédure est "correcte" au sens où elle transforme un couple $(\mathcal{F}, \mathcal{I})$ en un couple $(\mathcal{F}', \mathcal{I}')$ tel que :
  - La satisfaisabilité de la formule est préservée.
  - $\mathcal{F} \wedge \mathcal{I}$ est une conséquence logique de $\mathcal{F}' \wedge \mathcal{I}'$ (C_4).
  - Si $\mathcal{F}$ admet un modèle $\Im_{\mathcal{I}}(\mathcal{D})$-compatible, alors $\mathcal{F}'$ admet un modèle $\Im_{\mathcal{I}'}(\mathcal{D})$-compatible (C_2). Cette propriété assure que tout eq-modèle peut être construit, ce qui garantit la **complétude** de la procédure **BuildModel** (voir ci-dessous).
- Tout modèle ou contre-modèle total est identifié par la méthode (C_1). C'est nécessaire pour arrêter la recherche du modèle.

Procedure *Deduce*
INPUT:
a formula $\mathcal{F}$,
a partial eq-interpretation $\mathcal{I}$
OUTPUT:
a formula $\mathcal{F}'$
and a partial eq-interpretation $\mathcal{I}'$ such that:
- if $\mathcal{I}$ is total then either $Deduce(\mathcal{F},\mathcal{I}) = (\bot,\mathcal{I})$ or $Deduce(\mathcal{F},\mathcal{I}) = (\top,\mathcal{I})$ (C_1)
- for all $\mathcal{D}$, if there exists a $\mathfrak{I}_{\mathcal{I}}(\mathcal{D})$-compatible model of $\mathcal{F}$
then there exists a $\mathfrak{I}_{\mathcal{I}'}(\mathcal{D})$-compatible interpretation $\mathcal{J} \supseteq \mathcal{I}'$ with $\mathcal{J} \models \mathcal{F}'$ (C_2)
- $\mathcal{I} \subseteq \mathcal{I}'$ (C_3)
- $\mathcal{F}' \wedge \mathcal{I}' \Rightarrow \mathcal{F} \wedge \mathcal{I}$ (C_4)
- If $\mathcal{F} \wedge \mathcal{I}$ is satisfiable then $\mathcal{F}' \wedge \mathcal{I}'$ is satisfiable. (C_5)

FIG. 8.1 - : *La procédure Deduce*

– **Monotonie.** $\mathcal{I} \subseteq \mathcal{I}'$. Elle garantit la **terminaison** de la procédure **BuildModel** (voir ci-dessous).

Etant donné un ensemble de représentation $\mathcal{D}$ et une formule $\mathcal{F}$, l'algorithme **BuildModel** recherche un modèle $\mathcal{D}$-compatible de $\mathcal{F}$. Il est spécifié par un ensemble de règles appliquées de façon non déterministe : déduction, décomposition et nettoyage. Ces règles s'appliquent sur des ensembles de couples $(\mathcal{F},\mathcal{I})$ où $\mathcal{F}$ est une formule et $\mathcal{I}$ une interprétation partielle. $\mathcal{F}$ est la formule considérée, $\mathcal{I}$ représente l'interprétation partielle déjà construite. Initialement, l'ensemble de couples contient uniquement le couple $(\mathcal{F},\emptyset)$, où $\mathcal{F}$ est la formule à traiter.

1. **Règle de déduction.** Elle consiste à appliquer la procédure *Deduce* sur un couple $(\mathcal{F},\mathcal{I})$.

2. **Règle de décomposition.** Le problème $(\mathcal{F},\mathcal{I})$ est divisé en deux sous-problèmes obtenus par une règle de décomposition ajoutant à $\mathcal{F}$ respectivement les littéraux P et $\neg P$, où P est un c-littéral de $\mathcal{D}$. Comme dans la méthode de Davis et Putnam (DAVIS ET PUTNAM, 1960), le processus peut être répété jusqu'à obtenir un modèle de $\mathcal{F}$ ou un ensemble contradictoire. A la différence de (DAVIS ET PUTNAM, 1960), la règle de décomposition ne préserve pas l'équivalence puisque P peut contenir des variables.

   Remarquons que, comme nous l'avons déjà mentionné, la procédure *Deduce* peut modifier l'interprétation $\mathcal{I}$ et notamment *générer des c-littéraux non $\mathcal{D}$-compatibles*. Par conséquent, il ne suffit pas d'appliquer la règle de décomposition sur chaque c-clause de $\mathcal{D}$, mais il est nécessaire d'enlever au préalable les littéraux dont le complémentaire appartient à $\mathcal{I}$ (c'est l'objet de la définition de $\mathfrak{I}_S(L)$).

3. **Règle de nettoyage.** Elle supprime les couples de la forme $(\bot,\mathcal{I})$.

On note $\mathcal{R}_{\mathcal{D}}$ le système composé des règles { Déduction, Nettoyage, Décomposition } ci-dessous.

Déduction.	$\mathcal{S} \cup \{(\mathcal{F},\mathcal{I})\} \rightarrow \mathcal{S} \cup \{(\mathcal{F}',\mathcal{I}')\}$
	si $Deduce(\mathcal{F},\mathcal{I}) = (\mathcal{F}',\mathcal{I}')$
Nettoyage.	$\{(\bot,\mathcal{I})\} \cup \mathcal{S} \rightarrow \mathcal{S}$
Décomposition.	$\mathcal{S} \cup \{(\mathcal{F},\mathcal{I})\} \rightarrow \mathcal{S} \cup \{(\mathcal{F},\mathcal{I} \cup \{\mathfrak{I}_{\mathcal{I}}(P)\}),(\mathcal{F},\mathcal{I} \cup \{\mathfrak{I}_{\mathcal{I}}(\neg P)\})\}$
	si $P \in \mathcal{D}, \mathfrak{I}_{\mathcal{I}}(P) \neq \emptyset, \mathfrak{I}_{\mathcal{I}}(\neg P) \neq \emptyset$

Procedure BuildModel
INPUT:
a formula $\mathcal{F}$.
a representation set $\mathcal{D}$.
OUTPUT:
a eq-model $\mathcal{M}$ of $\mathcal{F}$,
or a message **no model found**.
begin
$\mathcal{S} := \{(\mathcal{F}, \emptyset)\}$
while $\mathcal{S} \neq \emptyset \wedge \forall (\mathcal{F}', \mathcal{I}) \in \mathcal{S}, \mathcal{F}' \neq \top$
begin
choose a rule ρ in $\mathcal{R}_{\mathcal{D}}$
apply ρ on $\mathcal{S}$
end
if $\mathcal{S} = \emptyset$
then return(**no model found**)
else $\{\exists (\top, \mathcal{I}) \in \mathcal{S}\}$
return$(\mathcal{I})$
end

FIG. 8.2 - : *La procédure* ***BuildModel***

Procedure EQMC { **EQ**-Model **C**onstruction }
INPUT:
a formula $\mathcal{F}$.
OUTPUT(if any):
a eq-model $\mathcal{M}$ of $\mathcal{F}$,
or a message **no model found**.
begin
$\mathcal{D} := \{P(\overline{x})/P \in \Omega\}$
while *true*
begin
or $\mathcal{D} := \mathrm{GR}(\mathcal{D})$
or
$\mathcal{M} :=$ **BuildModel**$(\mathcal{F}, \mathcal{D})$
if $\mathcal{M} \neq$ **no model found**
return$(\mathcal{M})$
if GR is not applicable and **EQMC**$(\mathcal{F}, \mathcal{D}) =$ **no model found**
then return(**no model found**)
end
end

FIG. 8.3 - : *La procédure* EQMC

La procédure EQMC est formellement spécifiée par la figure 8.3.

REMARQUE. Il y a une relation claire entre la règle GMPL et la règle de décomposition. Toutes deux ont pour objet de générer des c-clauses unitaires *compatibles* avec l'ensemble de c-clauses. La différence entre les deux procédures est que GMPL *calcule* les conditions assurant la compatibilité alors que la règle de décomposition suppose ces conditions valides et vérifie à posteriori la compatibilité. Elle nécessite par conséquent l'utilisation du retour-arrière si la c-clause s'avère incompatible avec la formule. En revanche, elle permet de générer des c-clauses ne pouvant être obtenues par la règle GMPL. Remarquons qu'un cas particulier de la règle de décomposition (où P est un littéral fermé) est utilisé dans (PETERSON, 1988) comme une astuce pour obtenir une réfutation courte d'un problème difficile.

8.1.3 Propriétés de la procédure EQMC

THÉORÈME 8.1.2. *Si* ***BuildModel***$(\mathcal{F})$ *retourne* $\mathcal{I}$ *alors* $\mathcal{I}$ *est un eq-modèle de* $\mathcal{F}$.

PREUVE. Par définition de *Deduce* (C_4), et par induction sur le nombre d'applications des règles, on a, à chaque itération et pour tout $(\mathcal{F}',\mathcal{I}) \in \mathcal{S} : \mathcal{I} \wedge \mathcal{F}' \Rightarrow \mathcal{F}$. Si **BuildModel**$(\mathcal{F})$ retourne $\mathcal{I}$, on a $\mathcal{I} \wedge \top \Rightarrow \mathcal{F}$ i.e. $\mathcal{I} \models \mathcal{F}$. C.Q.F.D.

Le théorème suivant prouve que notre méthode permet de construire des modèles pour toute formule de la classe $\mathfrak{C}_{\text{eq-model}}$. Pour cela, démontrons tout d'abord les lemmes suivants.

LEMME 8.1.3. *Toute application équitable de la procédure* ***BuildModel*** *se termine.*

PREUVE. C'est immédiat, car le nombre de c-littéraux distincts de $\mathcal{D}$ est fini. Par conséquent, après un nombre fini d'itérations, tous les couples de $\mathcal{S}$ sont de la forme $(\mathcal{F}',\mathcal{I})$, où $\mathcal{I}$ est totale. Après application de la règle *Deduce*, on obtient soit $(\bot,\mathcal{I})$, soit $(\top,\mathcal{I})$ (d'après (C_1)). C.Q.F.D.

LEMME 8.1.4. *Si* $\mathcal{F}$ *admet un eq-modèle* $\mathcal{D}$*-compatible, alors* ***BuildModel***$(\mathcal{F},\mathcal{D})$ *retourne un modèle de* $\mathcal{F}$.

PREUVE. Par induction sur le nombre d'applications des règles dans la procédure **BuildModel**, et d'après la définition de *Deduce*, il existe $(\mathcal{F}',\mathcal{I}) \in \mathcal{S}$, et une interprétation $\mathfrak{I}_{\mathcal{I}}(\mathcal{D})$-compatible validant $\mathcal{F}'$.

En effet :

- La propriété est vraie au rang 1, car $\mathcal{F}$ admet un modèle $\mathcal{D}$-compatible et $(\mathcal{F},\emptyset) \in \mathcal{S}$.
- Supposons que la propriété est vraie au rang n. Alors il existe $(\mathcal{F}',\mathcal{I}) \in \mathcal{S}$, et une interprétation $\mathcal{J}$ $\mathfrak{I}_{\mathcal{I}}(\mathcal{D})$-compatible validant $\mathcal{F}'$. Si la règle de déduction est appliquée la propriété est préservée, d'après (C_2). Si la règle de nettoyage est appliquée, la propriété est évidemment préservée. Si la règle de décomposition est appliquée, on a $P \in \mathcal{D}$ donc $\mathfrak{I}_{\mathcal{I}}(P) \in \mathfrak{I}_{\mathcal{I}}(\mathcal{D})$. Donc soit $\mathfrak{I}_{\mathcal{I}}(P) \in \mathcal{J}$ soit $\neg\mathfrak{I}_{\mathcal{I}}(P) \in \mathcal{J}$. Supposons par exemple que $P' = \mathfrak{I}_{\mathcal{I}}(P) \in \mathcal{J}$. Alors $(\mathcal{F}' \wedge \mathcal{I} \cup \{P'\})$ admet une eq-interprétation $\mathfrak{I}_{\mathcal{I}\cup\{P'\}}(\mathcal{D})$-compatible. D'où la propriété est vérifiée au rang $n+1$.

Par conséquent, lorsque **BuildModel** se termine, $\mathcal{S}$ contient nécessairement un couple de la forme $(\top,\mathcal{I})$. C.Q.F.D.

THÉORÈME 8.1.3. *Si* $\mathcal{F}$ *admet un eq-modèle, alors toute application équitable de* ***EQMC***$(\mathcal{F})$ *retourne un eq-modèle de* $\mathcal{F}$.

PREUVE. Soit $\mathcal{F}$ une formule de $\mathfrak{C}_{\text{eq-model}}$. L'application des règles de GR étant équitable, on obtient (après un certain nombre d'itérations) un ensemble $\mathcal{D}$ tel que $\mathcal{F}$ admet un modèle $\mathcal{D}$-compatible. D'après le lemme 8.1.4, on en déduit que la procédure **BuildModel** retourne un modèle de $\mathcal{F}$. C.Q.F.D.

8.2 Le choix de la procédure *Deduce*

Il reste à préciser le choix de la procédure *Deduce*. N'importe quelle procédure satisfaisant les spécifications peut être utilisée. Dans cette section, nous supposons que la formule $\mathcal{F}$ considérée est un ensemble de c-clauses et nous montrons que la procédure RAMC satisfait les conditions requises.

Notation 8.2.1 *Pour tout ensemble de c-clauses S, on note*

$$\textbf{\textit{decompose}}(S) = (S \setminus \textbf{\textit{Unit}}(S), \textbf{\textit{Unit}}(S))$$

si $\textbf{\textit{Unit}}(S)$ *est satisfaisable,* $(\bot, \emptyset)$ *sinon.*

THÉORÈME 8.2.1. *La procédure $Deduce(S, \mathcal{I})$ définie par :*

$$Deduce(S,\mathcal{I}) = \textbf{\textit{decompose}}(\text{RAMC}_{\mathcal{I}}(N_{ramc}{}^{*}(S \cup \mathcal{I})))$$

satisfait les conditions de la figure 8.1.

PREUVE. – Soit $\mathcal{I}$ une eq-interprétation totale et $S' = \text{RAMC}_{\mathcal{I}}(S \cup \mathcal{I})$. Si $\mathcal{I} \models S$, on a, d'après le théorème 6.5.2, $DSub(\mathcal{I}, S) = \emptyset$. Sinon $\mathcal{I} \not\models S$, donc $\square$ est déduite de S par c-résolution.

– Supposons qu'il existe une eq-interprétation $\mathcal{J}$ $\mathfrak{I}_{\mathcal{I}}(\mathcal{D})$-compatible validant S. Nous prouvons qu'il existe une eq-interprétation $\mathfrak{I}_{\mathcal{I}'}(\mathcal{D})$-compatible validant S'. Deux cas doivent être distingués. Si la règle ρ considérée préserve l'équivalence, la preuve est immédiate. Par conséquent, il suffit de considérer le cas de la règle GMPL. Soit E un ensemble déduit de S par GMPL. Soit $\mathcal{J}' = \mathfrak{I}_E(\mathcal{J}) \cup E$. Soit une c-clause fermée C de $\mathcal{S}(S)$. $\mathcal{J} \models C$ donc il existe $L \in C$ tel que $L \in \mathcal{J}$. Si $\neg L \in E$, alors on a $L \in \mathcal{J}'$, d'où $\mathcal{J}' \models C$. Sinon, par définition de GMPL, $E \models C$ donc $\mathcal{J}' \models C$. Il reste à prouver que $\mathcal{J}'$ est $\mathfrak{I}_{\mathcal{I}'}(\mathcal{D})$-compatible. Soit deux littéraux fermés L_1, L_2 appartenant à un même c-littéral de $\mathfrak{I}_{\mathcal{I}'}(\mathcal{D})$. Alors L_1, L_2 n'appartiennent pas à E (car $\mathcal{I}'$ contient E, par définition), d'où $\mathcal{J}(L_i) = \mathcal{J}'(L_i)$ $(\forall i = 1,2)$. $\mathcal{J}$ est $\mathfrak{I}_{\mathcal{I}}(\mathcal{D})$ compatible. En outre L_1 et L_2 n'appartiennent pas à $\neg\mathcal{I}$ (puisque $\mathcal{I} \subseteq \mathcal{I}'$). d'où $\mathcal{J}(L_1) = \mathcal{J}(L_2)$ et $\mathcal{J}'(L_1) = \mathcal{J}'(L_2)$. Par conséquent, $\mathcal{J}'$ est $\mathfrak{I}_{\mathcal{I}'}(\mathcal{J})$-compatible.

C.Q.F.D.

8.3 Exemple

L'exemple suivant, très simple, illustre la méthode EQMC.

EXEMPLE 8.3.1. Soit S l'ensemble de c-clauses suivant (déjà donné au chapitre 7), adapté de (BOURELY ET AL., 1994) (problème SYN303-1 de la base de formules TPTP, voir (SUTTNER ET SUTCLIFFE, 1996)).

$$\begin{array}{l}
[\![P(x,y) \vee \neg P(f(x),f(y)) \ : \top]\!] \\
[\![\neg P(x,y) \vee P(f(x),f(y)) \ : \top]\!] \\
[\![\neg P(a,f(x)) \ : \top]\!] \\
[\![\neg P(f(x),a) \ : \top]\!] \\
[\![P(a,a) \ : \top]\!]
\end{array}$$

Soit l'ensemble de représentation $\mathcal{D}_0 = \{P(x,y)\}$. L'application de la règle de décomposition sur $\mathcal{D}_0$ conduit aux ensembles $S \cup \{P(x,y)\}$ et $S \cup \{\neg P(x,y)\}$. Ces ensembles sont insatisfaisables. L'application de la procédure *Deduce* sur S_1 et S_2 retourne $\bot$. Par conséquent, **BuildModel** retourne **no model found**. Le système GR doit être appliqué, afin d'obtenir un ensemble de

représentation permettant de représenter une plus large classe de eq-interprétations. Appliquons par exemple d'abord la règle $\mathrm{GR}_{=}$ sur x, y. On obtient

$$\{P(x,x), [\![P(x,y) \ : x \neq y]\!]\}.$$

En appliquant la règle de décomposition sur $\{P(x,x)\}$, on génère les ensembles $S \cup \{P(x,x)\}$ et $S \cup \{\neg P(x,x)\}$. $S \cup \{\neg P(x,x)\}$ contient deux c-littéraux $P(a,a)$ et $\neg P(x,x)$ contradictoires donc la procédure *Deduce*$(S \cup \{\neg P(x,x)\})$ retourne $\bot$. L'appel de *Deduce* sur $S \cup \{P(x,x)\}$ génère (après simplification par dissubsumption et résolution des contraintes):

$$\begin{array}{l}
[\![P(x,y) \vee \neg P(f(x),f(y)) \ : x \neq y]\!] \\
[\![\neg P(x,y) \vee P(f(x),f(y)) \ : x \neq y]\!] \\
[\![\neg P(a,f(x)) \ : \top]\!] \\
[\![\neg P(f(x),a) \ : \top]\!] \\
[\![P(x,x) \ : \top]\!]
\end{array}$$

En appliquant la règle Gmpl, on obtient alors le c-littéral

$$\{[\![\neg P(x,y) \ : x \neq y]\!]\}.$$

Ce littéral, conjointement avec le littéral $P(x,x)$ introduit précédemment, définit un modèle de S:

$$\{P(x,x), [\![\neg P(x,y) \ : x \neq y]\!]\}.$$

Remarque. N'importe quelle autre stratégie équitable aurait pu être utilisée : par exemple, nous aurions pu utiliser la règle de décomposition pour générer le littéral $[\![\neg P(x,y) \ : x \neq y]\!]$, au lieu d'utiliser Gmpl. Remarquons cependant que ce littéral, pas plus que le littéral $P(x,x)$, ne peut être déduit de l'ensemble initial par Ramc.

A notre connaissance, il n'existe aucune autre méthode capable de construire un modèle pour S (mis à part la méthode Ramcet-$2_{\mathcal{I}_s}$ qui sera introduite au chapitre 9). En effet, S ne possède aucun modèle fini. En outre, toutes les méthodes pour la construction de modèles de Herbrand (telles que celles de (Fermüller et Leitsch, 1996), ou la méthode Ramc) ne terminent pas sur cette formule : elles génèrent, en effet, les ensembles de littéraux fermés $\{P(t,t), \neg P(t,s)/t \neq s\}$, mais ne sont pas capables de générer les c-littéraux $P(x,x)$ et $[\![\neg P(x,y) \ : x \neq y]\!]$ qui sont nécessaires pour exprimer le modèle de façon finie. Notons que ces deux c-clauses sont des conséquences *inductives* de l'ensemble initial (c'est-à-dire vraies dans tout modèle de Herbrand), mais pas des conséquences logiques de la formule (il existe des modèles de la formule falsifiant ces c-clauses). ◇

8.4 Quelques heuristiques pour la méthode EQMC

En pratique, l'application non déterministe de la procédure Eqmc se révèle très coûteuse, notamment à cause des règles de génération des ensembles de représentation, qui augmentent très rapidement la taille des ensembles. Par conséquent, il est nécessaire de disposer de stratégies ou d'heuristiques permettant de guider l'application des règles, et notamment le choix de *l'ensemble de représentation.*

8.4.1 Choix des c-littéraux

Il est clair que tout littéral $L \in \mathcal{D}$ tel que *Deduce*$(\mathcal{F}, \mathcal{I} \cup \Im_{\mathcal{I}}(P))$ est de la forme $(\bot, \mathcal{I}')$ doit être choisi dès que possible pour l'application de la règle de décomposition. En effet, l'un des deux couples générés par la règle sera éliminé par la règle de nettoyage. Ainsi, il peut être intéressant de tester à l'avance quels sont les c-littéraux incompatibles avec une formule $\mathcal{F}$ de $\mathcal{S}$, et appliquer la règle de décomposition sur ces littéraux, ce qui réduit de façon importante l'espace de recherche. Plus généralement, il est également possible de détecter les *ensembles de c-littéraux* de $\mathcal{D}$ incompatibles avec $\mathcal{F}$, de façon à choisir en priorité les c-littéraux apparaissant dans des ensembles de taille *minimale.*

8.4.2 Ordre d'application des règles

Il est nécessaire de donner des critères pour choisir entre l'application des règles décomposition, nettoyage ou déduction. Il est clair que, en pratique, il est préférable d'appliquer en premier lieu la règle de nettoyage et la règle *Deduce*, car elles n'introduisent aucun facteur de branchement à la différence de la règle de décomposition. Evidemment, cette stratégie est incomplète (du point de vue de la construction de modèle) car l'application répétée de la procédure *Deduce* ne se termine pas en général. Par conséquent, il est nécessaire de limiter l'application de cette procédure (par exemple en limitant le temps de calcul, ou le nombre d'application de *Deduce*) afin de préserver la complétude de la méthode.

8.4.3 Stratégie d'application des règles de GR

Le choix de la stratégie d'application des règles $\mathrm{GR}_{=}$ et GR_{Σ} est cruciale en pratique, car l'application non déterministe de ces règles augmente très rapidement la taille de l'ensemble de représentation. Il est nécessaire de définir des heuristiques pour guider le choix du littéral et le choix de la position à laquelle les règles $\mathrm{GR}_{=}$ ou GR_{Σ} sont appliquées. Une heuristique possible est très brièvement décrite ci-dessous. Elle est fondée sur l'utilisation des informations déduites de la formule par la procédure *Deduce* et sur l'étude de la *réfutation* de la formule. Pour simplifier, nous supposons que la procédure RAMC est utilisée (bien que l'heuristique proposée puisse aisément être adaptée à d'autres types de procédures).

Notation 8.4.1 *Soit $\mathcal{D}$ un ensemble de représentation. Soit une formule $\mathcal{F}$ telle que* ***BuildModel***$(\mathcal{F}, \mathcal{D}) =$ ***no model found****. Soit $\{[\![\Box \ : \mathcal{X}_i]\!]/1 \leq i \leq n\}$ l'ensemble des c-clauses vides déduites pendant l'application de la procédure* ***BuildModel****. On note :*

$$\mathcal{Y}_{\mathcal{F},\mathcal{D}} = \bigvee_{i=1}^{n} \mathcal{X}_i \sigma_i$$

où les σ_i sont des renommages distincts des variables de $\mathcal{X}_i$ par de nouvelles variables.

REMARQUE. La formule $\mathcal{Y}_{\mathcal{F},\mathcal{D}}$ peut être vue comme une explication du fait que $\mathcal{D}$ est incompatible avec $\mathcal{F}$.

La règle GR_{Σ}. Soit x une variable de $\mathcal{D}$. Soit $\mathcal{Y}'$ la formule obtenue en remplaçant toutes les occurrences de x dans $\mathcal{Y}_{\mathcal{F},\mathcal{D}}$ (et toutes les occurrences des variables obtenues par renommage à partir de x) par une unique variable y. S'il existe un terme t tel que $\mathcal{Y}'\{y \to t\}$ est satisfaisable, alors il est inutile d'appliquer la règle GR_{Σ} sur x. En effet, il est clair que tout ensemble de littéraux obtenu par application de ces règles sur x contient au moins un littéral incompatible avec la formule de départ.

EXEMPLE 8.4.1. Soit un ensemble de formules $\mathcal{F}$ contenant les c-clauses

$$\{P(u,u), \neg P(u, f(u)), Q(b, f(v)), \neg Q(a, f(a))\}.$$

Soit l'ensemble de représentation $\mathcal{D} = \{P(x,x), [\![P(x,y) \ : x \neq y]\!], Q(x,x), [\![Q(x,y) \ : x \neq y]\!]\}$. Il est clair que $\mathcal{F}$ peut être réduit à $\bot$ par application de la règle de décomposition sur $\mathcal{D}$. En effet, cette règle ajoute soit $[\![Q(x,y) \ : x \neq y]\!]$ (ce qui contredit le c-littéral $\neg Q(a, f(a))$) soit $[\![\neg Q(x,y) \ : x \neq y]\!]$ ce qui contredit $Q(b, f(v))$.

Les littéraux $[\![Q(x,y) \ : x \neq y]\!]\}$ et $[\![\neg Q(x,y) \ : x \neq y]\!]$ sont incompatibles avec $\mathcal{F}$. On obtient par c-résolution les c-clauses suivantes :

$$[\![\Box \ : x = b \wedge y = f(v) \wedge x \neq y]\!]$$

et (après renommage $\{x \to x', y \to y'\}$) :

$$[\![\Box \ : x' = a \wedge y' = f(a) \wedge x' \neq y']\!]$$

On a $\mathcal{Y}_{\mathcal{F},\mathcal{D}} = (x = b \wedge y = f(v) \wedge x \neq y) \wedge (x' = a \wedge y' = f(a) \wedge x' \neq y')$. Remplaçons y et y' par une nouvelle variable y''. On obtient $\mathcal{Y}' = ((x = b \wedge y'' = f(v) \wedge x \neq y'') \wedge (x' = a \wedge y'' = f(a) \wedge x' \neq y''))$. Ici $\mathcal{Y}'$ est satisfaisable, donc la règle GR_Σ n'est pas applicable sur y. En revanche, si on remplace dans la formule $\mathcal{Y}_{\mathcal{F},\mathcal{D}}$, x et x' par une nouvelle variable x'', la formule $\mathcal{Y}'' = (x'' = b \wedge y = f(v) \wedge x'' \neq y) \wedge (x'' = a \wedge y' = f(a) \wedge x'' \neq y')$ obtenue est insatisfaisable.

GR_Σ ne peut être appliquée que sur le littéral $[\![Q(x,y) \ : x \neq y]\!]$ et sur la variable x, ce qui conduit à l'ensemble :

$$\mathcal{D}' = \{P(x,x), [\![P(x,y) \ : x \neq y]\!], Q(x,x), [\![Q(a,y) \ : a \neq y]\!], \\ [\![Q(b,y) \ : b \neq y]\!], [\![Q(f(u),y) \ : y \neq f(u)]\!]\}.$$

En appliquant ensuite les règles de décomposition, on obtient le modèle suivant. $\{P(x,x), [\![\neg P(x,y) \ : x \neq y]\!], [\![\neg Q(a,y) \ : a \neq y]\!], [\![Q(b,y) \ : b \neq y]\!]\}$ $\diamond$

Cette heuristique fournit un critère pour guider le choix de la variable sur laquelle la règle GR_Σ est appliquée.

La règle $\mathrm{GR}_=$. Une heuristique similaire peut être définie pour guider l'application de la règle $\mathrm{GR}_=$. Afin de réduire l'espace de recherche, nous n'appliquons cette règle que sur des couples de termes (s,t) où s et t sont des *variables*. Notons $\mathcal{X}^+$ la contrainte : $\bigwedge_{i=1}^n x\sigma_i = y\sigma_i$ et $\mathcal{X}^-$ la contrainte $\bigwedge_{i=1}^n x\sigma_i \neq y\sigma_i$. Nous appliquons la règle $\mathrm{GR}_=$ si et seulement si $\mathcal{Y}_{\mathcal{F},\mathcal{D}} \wedge \mathcal{X}^-$ et $\mathcal{Y}_{\mathcal{F},\mathcal{D}} \wedge \mathcal{X}^+$ sont simultanément insatisfaisables.

Il resterait cependant à prouver que cette heuristique préserve la complétude de la méthode (c'est-à-dire que tout eq-modèle peut être généré en utilisant cette heuristique).

8.5 Extension au premier ordre

Jusqu'à présent, nous avons supposé que la formule à traiter était sous forme clausale. C'était justifié par le fait que toute formule $\mathcal{F}$ peut être automatiquement transformée en un ensemble de clauses $S = \mathit{clause}(\mathcal{F})$ (voir chapitre 2) tel que

1. S satisfaisable $\iff$ $\mathcal{F}$ satisfaisable ;
2. Si $\mathcal{M}$ est un modèle de S alors $\mathcal{M} \models \mathcal{F}$.

On peut donc toujours se ramener à l'étude d'ensembles de clauses. D'un point de vue déductif, cette transformation ne pose que peu de problèmes, car, d'une part, elle préserve la satisfaisabilité et, d'autre part, il est possible d'utiliser des techniques de renommage pour éviter l'augmentation exponentielle de la taille de la formule et obtenir ainsi une méthode de traduction efficace et utilisable en pratique (voir par exemple (Plaisted et Greenbaum, 1986; Boy de la Tour, 1992; Egly et Rath, 1996)). Il en va tout autrement du point de vue de la construction de modèles. En effet, il est clair que – à cause de l'opération de *skolémisation* – la mise sous forme clausale ne préserve pas les modèles. Par conséquent, il est possible qu'une formule $\mathcal{F}$, admettant un eq-modèle $\mathcal{M}$, soit transformée en un ensemble de c-clauses n'admettant aucun eq-modèle, i.e. on peut avoir $\mathcal{F} \in \mathfrak{C}_{\text{eq-model}}$ et $\mathit{clause}(\mathcal{F}) \notin \mathfrak{C}_{\text{eq-model}}$. En effet, l'opération de skolémisation remplace un quantificateur existentiel par un nouveau symbole de fonction ce qui a pour effet de *réduire* le nombre d'interprétations possibles (puisque la valeur de la variable existentielle est alors fixée).

Exemple 8.5.1. Considérons la formule $\mathcal{F}$ suivante.

$$P(a,a) \wedge \neg P(b,a) \wedge \forall x, x'.\exists y.P(x',x) - \neg P(x',y)$$

$\mathcal{F}$ admet le eq-modèle suivant (sur la signature $\Sigma = \{a, b\}$) :

$$\{P(a,a), \neg P(a,b), \neg P(b,a), P(b,b)\}.$$

La forme clausale de $\mathcal{F}$ est la suivante.

$$\begin{array}{l}
[\![P(a,a) \ : \top]\!] \\
[\![\neg P(a,b) \ : \top]\!] \\
[\![\neg P(x',x) \vee \neg P(x', f(x',x)) \ : \top]\!] \\
[\![P(x',x) \vee P(x', f(x',x)) \ : \top]\!]
\end{array}$$

Cet ensemble de c-clauses *n'admet aucun eq-modèle* (voir chapitre 11 et (KLINGENBECK, 1996)). ◇

Par conséquent, la transformation sous forme c-clausale peut dans certains cas empêcher la construction d'un eq-modèle. Pour résoudre ce problème, nous proposons de généraliser la méthode EQMC à des formules du premier ordre quelconque. Evidemment, la méthode RAMC ne peut plus être appliquée puisqu'elle suppose que la formule est sous forme clausale. L'idée est alors la suivante : au lieu d'employer la méthode RAMC pour évaluer les formules et détecter les contre-modèles partiels, nous allons nous servir de l'opérateur $\text{Normalize}_y(x)$ défini au chapitre 5, qui utilise une interprétation partielle pour simplifier une formule du premier ordre. En particulier, cet opérateur permet de vérifier qu'une interprétation valide une formule ou de détecter qu'une interprétation partielle est incompatible avec une formule. La procédure *Deduce* est définie de la façon suivante :

$$\mathit{Deduce}(\mathcal{F}, \mathcal{I}) = (\text{Normalize}_{\mathcal{I}}(\mathcal{F}), \mathcal{I}).$$

Il est clair que cette définition satisfait les conditions 8.1 (voir chapitre 5). Les propriétés de la méthode (et notamment la complétude par rapport à la classe $\mathfrak{C}_{\text{eq-model}}$) sont donc préservées.

Chapitre 9

RAMCET: extension de la méthode des tableaux

"Les idées vraiment claires n'oublient pas l'ombre dont elles viennent."
Claude Roy

Dans ce chapitre est proposée une extension de la méthode des tableaux sémantiques. Elle est fondée sur une idée similaire à celle utilisée pour étendre la méthode de résolution : tirer parti des contraintes équationnelles afin d'exprimer simultanément les conditions qui permettent ou qui empêchent l'application des règles d'inférence (ici des règles de construction des tableaux). Cette idée a été exploitée dans (Caferra et Zabel, 1993) où la procédure des tableaux sémantiques a été étendue en utilisant des contraintes équationnelles. La méthode obtenue a été nommée Ramcet-1[1]. La méthode des tableaux sémantiques peut être vue comme une procédure énumérant les modèles partiels de la formule "en espérant ne pas en trouver". L'idée de base de l'extension proposée est de poser des conditions permettant de trouver un modèle. Nous proposons une nouvelle méthode, appelée Ramcet-$2_{\mathcal{I}_s}$, dont les avantages principaux sont les suivants.

- En premier lieu, on introduit une nouvelle règle de simplification, permettant d'utiliser les informations générées dans une branche du tableau afin de simplifier les formules de cette branche. Cette règle peut être vue comme une généralisation de la dissubsomption et de la résolution/disrésolution unitaire à des formules du premier ordre. Elle a pour objet de restreindre l'espace de recherche.

- En second lieu, nous définissons une méthode permettant d'introduire l'utilisation de stratégies *sémantiques* (c'est-à-dire fondées sur l'utilisation d'une interprétation $\mathcal{I}_s$ de la formule) dans la méthode des tableaux. Comme dans la méthode de résolution, $\mathcal{I}_s$ est utilisée pour restreindre l'application des règles, limitant ainsi l'espace de recherche. Dans certains cas (voir exemple 9.2.6), la stratégie peut empêcher la construction de branches infinies, détectant ainsi la satisfaisabilité de la formule.

- Enfin, nous définissons une méthode permettant d'extraire un modèle d'une branche non-fermée (éventuellement infinie) du tableau. Elle utilise des techniques de généralisation, visant à construire par induction une représentation du modèle correspondant à la branche à partir du modèle partiel généré lors de la construction du tableau. Nous montrons qu'elle étend de façon significative la classe de modèles constructibles.

La méthode obtenue est très différente de Ramcet-1(Caferra et Zabel, 1993) et beaucoup plus générale, notamment en ce qui concerne le processus d'extraction du modèle. Elle est *complète pour la réfutation* et constitue aussi une *procédure de décision* pour la classe $\mathfrak{C}_{\text{eq-model}}$,

1. Signifiant : **R**efutation **A**nd **M**odel **C**onstruction with **E**quational **T**ableaux, version 1.

c'est-à-dire qu'elle permet de construire un modèle de toute formule satisfaisable possédant un eq-modèle.

Contrairement à l'approche utilisée au chapitre 6 et à (Caferra et Zabel, 1993; Peltier, 1997c) nous n'utiliserons pas de formules ou clauses contraintes. Les conditions sur les variables sont exprimées dans les formules elles-mêmes sous forme de littéraux équationnels. Cette approche allège la manipulation des formules et permet une notation plus unifiée. Les conditions sur les variables libres du tableau sont exprimées sous forme d'une formule équationnelle associée au tableau (on obtient donc la notion de *tableau contraint*).

9.1 Préliminaires

Dans ce chapitre, **toutes les interprétations considérées sont des interprétations de Herbrand**. Les littéraux équationnels apparaissant dans les formules sont supposés interprétés dans la théorie vide (i.e. pour toute interprétation $\mathcal{I}$, on a $\mathcal{I} \models s = t$ si et seulement si $s = t$). Comme au chapitre 7, nous supposons donnée une interprétation partielle $\mathcal{I}_s$ de Herbrand.

Introduisons le système de règles suivant.

$\mathcal{X} \quad \rightarrow \top$ (si $\mathcal{X}$ est purement équationnelle et si $\mathcal{X} \equiv \top$)
$\mathcal{X} \quad \rightarrow \bot$ (si $\mathcal{X}$ est purement équationnelle et si $\mathcal{X} \equiv \bot$)
$\top \wedge \mathcal{F} \rightarrow \mathcal{F}$
$\bot \vee \mathcal{F} \rightarrow \mathcal{F}$
$\top \vee \mathcal{F} \rightarrow \top$
$\bot \wedge \mathcal{F} \rightarrow \bot$

De façon évidente, $\mathcal{R}$ est à terminaison finie. En outre, pour toute formule $\mathcal{F}$, $\mathcal{F} \equiv \mathcal{R}(\mathcal{F})$. **Dans la suite, nous supposerons que toutes les formules considérées sont en forme normale par rapport à $\mathcal{R}$.**

Si une formule $\mathcal{F}$ ne contient que des quantificateurs universels, $\mathcal{F}$ est équivalente à un ensemble (éventuellement infini) de formules fermées noté $\mathcal{S}(\mathcal{F})$ défini de la façon suivante.

Notation 9.1.1 *Soit $\mathcal{F}$ une formule telle que tout sous-formule de la forme $\exists x.\mathcal{G}$ de $\mathcal{F}$ est purement équationnelle. $\mathcal{F}$ est équivalent (après élimination des quantificateurs existentiels et mise sous forme prénexe) à une formule de la forme $\forall \overline{x}.\mathcal{F}'$, où $\mathcal{F}'$ ne contient aucun quantificateur. On note $\mathcal{S}(\mathcal{F})$ l'ensemble des formules $\mathcal{R}^*(\mathcal{F}'\sigma)$ distinctes de $\top$, où σ est une substitution fermée des variables de $\overline{x}$.*

Proposition 9.1.1. *On a*

$$\mathcal{F} \equiv \mathcal{S}(\mathcal{F}).$$

Remarque. L'opérateur $\mathcal{S}$ peut être vu comme une généralisation de l'opérateur du même nom introduit au chapitre 6 sur les ensembles de c-clauses.

9.1.1 Notion de fait

La notion de fait est l'analogue de la notion de c-clause unitaire. Il s'agit de formules permettant d'exprimer des eq-interprétations partielles.

Définition 9.1.1. *Un* fait *est une formule de la forme*

$$\forall \overline{x}.(\mathcal{X} \vee \mathcal{F})$$

où:

– *$\mathcal{X}$ est une formule équationnelle.*

– $\mathcal{F}$ *est un littéral.*

– $\overline{x}$ *est un ensemble de variables.*

◇

LEMME 9.1.1. *[Equivalence avec les c-littéraux] Un fait* $\forall\overline{x}.(\mathcal{X} \vee \mathcal{F})$ fermé *est équivalent à la c-clause*

$$[\![\mathcal{F} \ : \neg\mathcal{X}]\!]$$

PREUVE. Immédiate. C.Q.F.D.

Par conséquent, toutes les propriétés des c-littéraux peuvent être étendues de façon immédiate aux faits. Dans la suite, nous écrirons indifféremment $\forall\overline{x}.(\mathcal{X} \vee \mathcal{F})$ ou $[\![\mathcal{F} \ : \neg\mathcal{X}]\!]$ pour noter un fait fermé. Cependant, la seconde notation sera plutôt utilisée, car elle est plus proche de celle qui a été employée jusqu'ici.

REMARQUE. A la différence des c-clauses, un fait peut contenir des variables libres.

9.1.2 Simplification d'une formule dans un contexte

Les ensembles de faits peuvent être utilisés pour simplifier une formule donnée. Il suffit pour cela de généraliser la définition des formules $\phi^+_{\mathcal{M}}(\mathcal{F})$ et $\phi^-_{\mathcal{M}}(\mathcal{F})$ (voir définition 5.5.2, chapitre 5) en considérant des ensembles de faits au lieu d'ensembles de c-clauses unitaires (i.e. au lieu d'eq-interprétations). Par souci de simplicité, nous ne redonnons pas ici toutes les définitions. Il suffit de préciser la définition des formules $\mathfrak{F}(\mathcal{I}(P)^+)$ et $\mathfrak{F}(\mathcal{I}(P)^-)$, dans le cas où $\mathcal{I}$ est un ensemble de faits.

DÉFINITION 9.1.2. *[Extension de la définition de* $\mathfrak{F}$*] Soit* $\mathcal{I}$ *un ensemble de faits. On note* $\mathfrak{F}(\mathcal{I}(P)^+)$ *la formule*

$$\bigvee_{\forall\overline{y}.\neg\mathcal{X}\vee P(\overline{t})\in\mathcal{I}} \exists\overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X}$$

et $\mathfrak{F}(\mathcal{I}(P)^-)$ *la formule*

$$\bigvee_{\forall\overline{y}.\neg\mathcal{X}\vee\neg P(\overline{t})\in\mathcal{I}} \exists\overline{y}.\overline{x} = \overline{t} \wedge \mathcal{X}$$

(avec $\overline{y} = \mathcal{V}ar(\mathcal{X} \wedge P(\overline{t}))$*).* ◇

La définition de $\phi^+_{\mathcal{I}}$ et $\phi^-_{\mathcal{I}}$ où $\mathcal{I}$ est un ensemble de faits peut être alors déduite directement.

9.2 RAMCET - version 2

Les règles de construction de tableaux indiquées ci-dessous sont similaires à celles données dans (CAFERRA ET ZABEL, 1993) ou dans (PELTIER, 1997c). La présentation est cependant différente (pour des raisons de simplicité). Nous supposons ici que les formules sont sous forme normale négative, skolémisées et que la portée des quantificateurs est minimale. En outre, nous introduisons de nouvelles conditions d'application sémantiques (utilisant l'interprétation $\mathcal{I}_s$) sur les règles de construction du tableaux. Nous montrerons par la suite que ces restrictions préservent la complétude de la méthode.

DÉFINITION 9.2.1. *Un tableau est un couple* $(\mathcal{T}, \mathcal{X})$ *où*

– $\mathcal{T}$ *est un ensemble d'ensembles de formules.*

– $\mathcal{X}$ *est une formule équationnelle.*

Une branche *est un élément de $\mathcal{T}$. Une branche est dite* fermée *si elle contient $\bot$. On note $\mathcal{M}od(b)$ l'ensemble des faits appartenant à une branche b et $Eqpb((\mathcal{T},\mathcal{X}))$ la formule $\mathcal{X}$.* ◇

Les notions de validité, satisfaisabilité, etc. s'étendent de manière immédiate aux tableaux, par la définition suivante.

DÉFINITION 9.2.2. *Un tableau $(\mathcal{T},\mathcal{X})$ est équivalent (par définition) à la formule*

$$\neg\mathcal{X} \vee \bigvee_{b\in\mathcal{T}} \bigwedge_{f\in b} f.$$

◇

9.2.1 Les règles

Nous introduisons les règles suivantes.

$R_\wedge$:

$$\frac{(\mathcal{S}\cup\{\{\mathcal{F}_1\wedge\mathcal{F}_2\}\cup\mathcal{B}\},\mathcal{X})}{(\mathcal{S}\cup\{\{\mathcal{F}_1,\mathcal{F}_2\}\cup\mathcal{B}\},\mathcal{X})}$$

si $\phi^+_{\mathcal{I}_s}(\neg\mathcal{X}\vee(\mathcal{F}_1\wedge\mathcal{F}_2))\neq\top$.

$R_\vee$:

$$\frac{(\mathcal{S}\cup\{\{\mathcal{F}_1\vee\mathcal{F}_2\}\cup\mathcal{B}\},\mathcal{X})}{(\mathcal{S}\cup\{\{\mathcal{F}_1\}\cup\mathcal{B},\{\mathcal{F}_2\}\cup\mathcal{B}\},\mathcal{X})}$$

si $\phi^+_{\mathcal{I}_s}(\neg\mathcal{X}\vee(\mathcal{F}_1\vee\mathcal{F}_2))\neq\top$.

$R_\forall$:

$$\frac{(\mathcal{S}\cup\{\{\forall x.\mathcal{F}\}\cup\mathcal{B}\},\mathcal{X})}{(\mathcal{S}\cup\{\{\forall x.(\mathcal{F}\vee x=y),\mathcal{F}\{x\to y\}\}\cup\mathcal{B}\},\mathcal{X})}$$

si y est un terme (y peut être par exemple une nouvelle variable, ou encore un terme fermé), et si $\phi^+_{\mathcal{I}_s}(\neg\mathcal{X}\vee\forall x.\mathcal{F})\neq\top$.

Notons que la formule $x=y$ est ajoutée à la formule $\forall x.\mathcal{F}$. En effet, puisque la variable x a été instanciée par le terme y, il est inutile d'instancier à nouveau la variable par le même terme dans cette branche.

EXEMPLE 9.2.1. Soit $\forall x.P(x)$ une formule. La règle $R_\forall$ supprime la formule de la branche où elle apparaît et la remplace par la conjonction des deux formules suivantes : $P(x_1)$ (où x_1 est une nouvelle variable) et $\forall x.x=x_1\vee P(x)$. ◇

Le principe de la règle R_{clash} définie ci-dessous est simple. Si une branche donnée contient deux littéraux $P(t_1)$ et $\neg P(t_2)$ alors t_1 et t_2 ne peuvent être égaux si la branche est satisfaisable. Donc $t_1\neq t_2$ est valide dans la branche considérée.

R_{clash}:

$$\frac{(\mathcal{S}\cup\{\{P(\overline{t}),\neg P(\overline{s})\}\cup\mathcal{B}\},\mathcal{X})}{(\mathcal{S}\cup\{\{P(\overline{t}),\neg P(\overline{s}),\overline{t}\neq\overline{s}\}\cup\mathcal{B}\},\mathcal{X})}$$

Notons que l'interprétation $\mathcal{I}_s$ est utilisée pour *restreindre* l'application des règles $R_\vee, R_\wedge, R_\forall$.

Règles de traitement de l'égalité

Les règles suivantes permettent de traiter les formules équationnelles apparaissant dans le tableau (rappelons que ces formules sont interprétées dans l'algèbre des termes).

$R_{=_1}$ permet de fermer une branche contenant une formule équationnelle $\mathcal{Y}$ en ajoutant la négation de $\mathcal{Y}$ aux contraintes globales.

$R_{=_1}$:

$$\frac{(\mathcal{S} \cup \{\{\mathcal{Y}\} \cup \mathcal{B}\}, \mathcal{X})}{(\mathcal{S}, \mathcal{X} \wedge \neg\mathcal{Y})}$$

si $\mathcal{Y}$ est purement équationnelle, et si $\mathcal{X} \wedge \neg\mathcal{Y}$ est satisfaisable.

La règle suivante élimine les formules incompatibles avec le problème équationnel global $\mathcal{X}$.

$R_{=_2}$:

$$\frac{(\mathcal{S} \cup \{\{\mathcal{Y}\} \cup \mathcal{B}\}, \mathcal{X})}{(\mathcal{S} \cup \{\mathcal{B}\}, \mathcal{X})}$$

si $\mathcal{Y}$ est purement équationnelle, $\mathcal{X} \wedge \neg\mathcal{Y}$ est insatisfaisable.

EXEMPLE 9.2.2. Les exemples suivants illustrent le fonctionnement des règles $R_{=_1}$ et $R_{=_2}$.

1. Soit une branche contenant deux littéraux $P(x)$ et $P(b)$ (les contraintes globales sont $\top$). La règle R_{clash} génère la formule $x \neq b$. Puisque $x = b$ est satisfaisable, nous pouvons fermer la branche en ajoutant aux contraintes la formule $x = b$.
2. Supposons maintenant que la branche contient les formules $P(x)$ et $P(b)$, mais que les contraintes globales $\mathcal{X}$ sont équivalentes à $x = a$. La règle R_{clash} génère comme précédemment la formule $x \neq b$. Cependant, $x = b \wedge \mathcal{X} \equiv \bot$ donc est insatisfaisable. Par conséquent, $x \neq b$ est supprimé de la branche par la règle $R_{=_2}$.

◇

Une nouvelle règle de simplification

La règle suivante est nouvelle. Elle utilise les résultats de la section 9.1.2 sur la simplification des formules dans un modèle partiel. Informellement, la règle remplace une formule $\mathcal{F}$ donnée par la formule $\text{Simplify}_{\mathcal{M}od(\mathcal{B})}(\mathcal{F})$.

R_{simp}:

$$\frac{(\mathcal{S} \cup \{\{\mathcal{G}[\mathcal{F}]_p\} \cup \mathcal{B}\}, \mathcal{X})}{(\mathcal{S} \cup \{\{\mathcal{G}[\text{Simplify}_{\mathcal{M}od(\mathcal{B})}(\mathcal{F})]_p\} \cup \mathcal{B}\}, \mathcal{X})}$$

Il est parfois utile d'appliquer la règle sur des sous-formules de formules apparaissant dans la branche et non sur les formules elles-mêmes, comme le montrent les exemples suivants.

EXEMPLE 9.2.3. Soit $\mathcal{F} = \forall x.P(x)$. Soit $\mathcal{B} = \{P(a)\}$. On a $\phi^+_{\mathcal{B}}(\mathcal{F}) = \forall x.x = a$ i.e. $\phi^+_{\mathcal{B}}(\mathcal{F}) \equiv \bot$. Par conséquent, la formule $\phi^+_{\mathcal{B}}(\mathcal{F})$ n'apporte aucune information supplémentaire. Cependant, en reconsidérant la sous-formule $P(x)$, il est facile de voir que si $\mathcal{I} \models \mathcal{B}$ et si $x = a$ alors $\mathcal{I} \models P(x)$ (puisque $P(a) \in \mathcal{B}$). Par conséquent, cette instance de $P(x)$ n'a pas besoin d'être considérée. Plus formellement, nous pouvons ajouter la condition $x \neq a$ à la sous-formule $P(x)$. Donc, $\mathcal{F}$ peut être transformée en $\mathcal{F}' = \forall x.x = a \vee P(x)$. ◇

EXEMPLE 9.2.4. Soit $\mathcal{T}$ un tableau. Supposons que $\mathcal{T}$ contient une branche b avec une formule $\mathcal{F} = \forall x.(P(x) \vee R(a, x))$ et l'ensemble de faits $\{\forall u.P(f(u)), \forall x.R(x, x)\}$. On a $\phi^-_{\mathcal{B}}(\mathcal{F}) \equiv \bot$ et $\phi^+_{\mathcal{B}}(\mathcal{F}) = \forall x.(\exists u.x = f(u)) \vee \exists y.(y = a \wedge y = x) \equiv \forall x.\exists u.x = f(u) \vee x = a$. Si la signature Σ contient uniquement les symboles a et f, tout terme fermé est de la forme $f(u)$ ou a, donc $\phi^+_{\mathcal{B}}(\mathcal{F}) \equiv \top$.

Ainsi, la formule $\mathcal{F}$ est supprimée de la branche et remplacée par $\top \wedge (\top \vee \mathcal{F})$. Puisque cette formule est valide, elle peut être supprimée de la branche (par application de $R_{=_2}$). ◇

9.2.2 Exemples

Avant d'établir les propriétés de la méthode, illustrons son fonctionnement par deux exemples simples mettant en évidence l'intérêt de la stratégie sémantique proposée.

EXEMPLE 9.2.5. Soit $\mathcal{I}_s = \emptyset$. De façon évidente, on a pour toute formule $\mathcal{F}$, $\phi^+_{\mathcal{I}_s}(\mathcal{F}) = \phi^-_{\mathcal{I}_s}(\mathcal{F}) \equiv \bot$. Soit $\mathcal{F} = \forall x.(P(x) \rightarrow (P(f(x)) \vee P(g(x)))) \wedge P(a)$. On note $\mathcal{F}'$ la formule $\forall x.(P(x) \rightarrow (P(f(x)) \vee P(g(x))))$. Appliquons $R_\wedge$ sur $\mathcal{F}$. $\mathcal{F}$ est remplacée par les formules $\mathcal{F}'$ et $P(a)$. Ensuite, appliquons la règle $R_\forall$ sur x. Nous introduisons une nouvelle variable libre x_1 et nous obtenons

$$P(x_1) \rightarrow (P(f(x_1)) \vee P(g(x_1))).$$

Simultanément, $\mathcal{F}'$ est transformée en $\forall x.x = x_1 \vee (P(x) \rightarrow (P(f(x)) \vee P(g(x))))$. En appliquant la règle $R_\vee$, on obtient finalement les deux branches suivantes.

1. $\{\mathcal{F}', \neg P(x_1), P(a)\}$.
2. $\{\mathcal{F}', P(f(x_1)) \vee P(g(x_1)), P(a)\}$.

Appliquons la règle R_{clash} sur la branche 1 (sur les deux littéraux $\neg P(x_1), P(a)$). On obtient la formule $x_1 \neq a$. En appliquant la règle $R_{=_1}$, on peut alors fermer la branche, en ajoutant aux contraintes globales la formule $x_1 = a$.

On obtient la branche $\{\mathcal{F}', P(f(x_1)) \vee P(g(x_1)), P(a)\}$ avec les contraintes $x_1 = a$.

Appliquons ensuite la règle $R_\vee$ sur $P(f(x_1)) \vee P(g(x_1))$. On obtient les branches $\{\mathcal{F}', P(f(x_1)), P(a)\}$ et $\{\mathcal{F}', P(g(x_1)), P(a)\}$. En répétant ce processus, on peut obtenir un nombre non borné de branches contenant des formules de la forme $P(f(a)), P(g(a)), P(f(g(a))), P(f(f(a))), \ldots$ ◇

EXEMPLE 9.2.6. Maintenant, utilisons une interprétation non vide pour restreindre l'espace de recherche. Considérons l'interprétation $\mathcal{I}_s = \{\forall x.P(x)\}$. La formule est valide dans $\mathcal{I}_s$ (on a, en effet, $\phi^+_{\mathcal{I}_s}(\forall x.P(x) \rightarrow (P(f(x)) \vee P(g(x)))) \equiv \top$).

La seule règle applicable est R_{simp}. Cette règle remplace $\mathcal{F}'$ par :

$$\mathcal{F}'' = \forall x.(x = a \vee P(x)) \rightarrow (P(f(x)) \vee P(g(x))).$$

Puisque $\phi^+_{\mathcal{I}_s}(\mathcal{F}'') \equiv \top$, *aucune autre règle ne peut être appliquée*. La procédure s'arrête. **L'utilisation de l'interprétation $\mathcal{I}_s$ permet de restreindre l'espace de recherche et évite la génération des branches infinies.** ◇

9.2.3 Simulation de méthodes existantes

Il est intéressant de mentionner que les démonstrateurs de théorèmes par génération de modèles (tels que les PUHR-tableaux ou les Hyper-Tableaux sont des *cas particuliers* de notre approche. Ils correspondent, en effet, au cas où $\mathcal{I}_s$ est l'interprétation associant à tout atome la valeur de vérité *false*. La règle de *splitting* des PUHR-tableaux correspond exactement à la règle $R_\wedge$, les règles d'hyperrésolution unitaire et de subsomption sont des cas particuliers de la règle R_{simp}.

Notre méthode est cependant beaucoup plus générale puisque $\mathcal{I}_s$ peut être une interprétation quelconque, et à cause de l'utilisation des contraintes et de la règle de simplification.

9.2.4 La règle Model Explosion

La règle suivante n'est pas nécessaire à la complétude réfutationnelle de la méthode, mais peut se révéler utile pour restreindre l'espace de recherche. Elle remplace une formule $\mathcal{F}$ par deux formules $\mathcal{F}_1$ et $\mathcal{F}_2$ telles que $\mathcal{F} \equiv \mathcal{F}_1 \wedge \mathcal{F}_2$ et telles que pour toute substitution σ, $\mathcal{I}_s \models \mathcal{F}_1\sigma$ et $\mathcal{I}_s \not\models \mathcal{F}_2\sigma$. En appliquant cette règle avant d'appliquer les règles $R_\wedge, R_\vee$, il est possible d'éliminer certaines instances des formules considérées, ce qui restreint l'espace de recherche.

Model Explosion: $\dfrac{(\mathcal{S} \cup \{\mathcal{B} \cup \{\mathcal{G}[\mathcal{F}]_p\}\}, \mathcal{X})}{(\mathcal{S} \cup \{\mathcal{B} \cup \{\mathcal{G}[\neg\phi^+_{\mathcal{I}_s}(\mathcal{F}) \vee \mathcal{F} \wedge \phi^+_{\mathcal{I}_s}(\mathcal{F}) \vee \mathcal{F}]_p\}\}, \mathcal{X})}$

La correction de cette règle est évidente.

EXEMPLE 9.2.7. Soit une branche contenant les formules $\forall x.\neg P(x) \vee P(f(x))$ et $\neg P(b)$. Supposons que $\mathcal{I}_s = \{\forall y.y = f(a) \vee P(y)\}$.

On a $\phi^+_{\mathcal{I}_s}(\neg P(x) \vee P(f(x))) = (\forall x.x \neq a) \not\equiv \top$, donc $R_\forall$ et $R_\vee$ sont applicables sur $\forall x.\neg P(x) \vee P(f(x))$, ce qui génère deux branches $\neg P(x_1)$ et $P(f(x_1))$. A cause de la formule $P(b)$, les règles R_{clash} et $R_{=_2}$ peuvent être appliquées pour fermer la branche contenant $\neg P(x_1)$, en unifiant x_1 et b.

On obtient : $P(f(x_1)), P(b)$ avec les contraintes $x_1 = b$. En repétant ce processus, on génère une branche infinie, contenant les littéraux $P(b), P(f(b)), P(f(f(b))), \ldots$

Cependant, en considérant plus précisément l'interprétation $\mathcal{I}_s$, on remarque que si $x_1 = b$, on a $x_1 \neq a$ donc $\phi^+_{\mathcal{I}_s}(\neg P(x_1) \vee P(f(x_1))) \equiv \top$. Par conséquent, cette instance de $\neg P(x_1) \vee P(f(x_1))$ n'a pas besoin d'être prise en compte lors de l'application des règles d'extension (ce problème est similaire à celui identifié au chapitre 7 pour la résolution sémantique).

Plus exactement, en appliquant la règle **Model Explosion** avant $R_\vee$, on supprime la formule $\forall x.\neg P(x) \vee P(f(x))$ et on la remplace par $\forall x.x = a \vee \neg P(x) \vee P(f(x))$ et $\forall x.x \neq a \vee \neg P(x) \vee P(f(x))$. On a $\phi^+_{\mathcal{I}_s}(x = a \vee \neg P(x) \vee P(f(x))) \equiv \top$, donc $R_\vee$ ne peut pas être appliquée sur cette formule. En appliquant les règles $R_\forall$ et $R_\vee$ sur $\forall x.x \neq a \vee \neg P(x) \vee P(f(x))$ on obtient les branches $x_1 \neq a$, et $\neg P(x_1) \vee P(f(x_1))$. On applique la règle $R_{=_2}$ sur la formule $x_1 \neq a$ pour fermer la branche correspondante. On obtient : $P(b), \forall x.x \neq a \vee \neg P(x) \vee P(f(x_1)), \neg P(x_1) \vee P(f(x_1))$, avec $x_1 = a$, donc, après application de $R_\vee$:

- $\{P(b), \forall x.x \neq a \vee \neg P(x) \vee P(f(x_1)), \neg P(a)\}$,
- et $\{P(b), \forall x.x \neq a \vee \neg P(x) \vee P(f(x)), P(f(a))\}$.

Aucune règle d'extension ne s'applique, donc la procédure se termine. L'utilisation de la règle **Model Explosion** évite ici la génération de branches infinies et la non-terminaison de la méthode. ◇

9.2.5 Correction et complétude réfutationnelle

On note RAMCET-$2_{\mathcal{I}_s}$ le calcul composé des règles $R_\vee, R_\wedge, R_{eq}, R_{clash}, R_{simp}, R_\forall$.

Correction

THÉORÈME 9.2.1. RAMCET-$2_{\mathcal{I}_s}$ *est correct : pour tout couple* $(\mathcal{T}, \mathcal{T}')$ *de tableaux tels que* $\mathcal{T} \rightarrow_{\text{RAMCET-}2_{\mathcal{I}_s}} \mathcal{T}'$ *et pour toute interprétation* $\mathcal{I}$ *de Herbrand, on a*

$$\mathcal{I} \models \mathcal{T} \Rightarrow \mathcal{I} \models \mathcal{T}'.$$

PREUVE. La correction des règles $R_\wedge, R_\vee, R_{clash}, R_\forall, R_{=_2}$ est une simple conséquence de la sémantique des symboles $\vee, \wedge, \forall$ et de la définition des tableaux sémantiques. La correction de la règle R_{simp} est une conséquence immédiate du théorème 5.6.1. Il suffit par conséquent de prouver la correction de la règle $R_{=_1}$. Supposons que

$$\mathcal{T} \rightarrow_{R_{=_1}} \mathcal{T}'.$$

Soit $\mathcal{I}$ une interprétation. Si $\mathcal{I} \models \mathcal{T}$ alors pour toute solution σ de $\mathcal{X}$, il existe une branche $\mathcal{B}$ de $\mathcal{T}$ telle que $\mathcal{I} \models \mathcal{B}\sigma$. Soit une solution σ' de $\mathcal{X} \wedge \neg\mathcal{Y}$. σ' est une solution de $\mathcal{X}$, donc il existe une branche $\mathcal{B}$ de $\mathcal{T}$ telle que $\mathcal{I} \models \mathcal{B}\sigma'$. Or $\mathcal{Y}\sigma'$ est faux donc $\mathcal{B}$ ne contient pas $\mathcal{Y}$. D'où $\mathcal{B} \in \mathcal{T}'$. Par conséquent, pour toute solution σ' de $\mathcal{X} \wedge \neg\mathcal{Y}$, il existe une branche $\mathcal{B}$ de $\mathcal{T}'$ telle que $\mathcal{I} \models \mathcal{B}\sigma$. D'où $\mathcal{I} \models \mathcal{T}'$. C.Q.F.D.

On en déduit que si $\mathcal{T}$ est un tableau fermé déduit de $(\{\{\mathcal{F}\}\}, \top)$, $\mathcal{F}$ est insatisfaisable, par induction sur la longueur de la dérivation.

Complétude réfutationnelle

La preuve de la complétude réfutationnelle de la méthode des tableaux utilise habituellement les *ensembles de Hintikka* (SMULLYAN, 1968; FITTING, 1990). Ici, les formules peuvent contenir des littéraux équationnels. D'autre part, les interprétations considérées sont nécessairement de Herbrand (c'est-à-dire que l'interprétation de "=" est nécessairement l'identité syntaxique entre termes). Enfin, nous utilisons la règle R_{simp}, qui a pour objet de simplifier le tableau, et nous imposons des restrictions sur les conditions d'application des règles $R_\wedge$, $R_\vee$ et $R_\forall$. Par conséquent, il est nécessaire d'adapter la définition habituelle des ensembles de Hintikka afin de prendre en compte toutes ces extensions.

DÉFINITION 9.2.3. *Pour toute formule $\mathcal{F}$ en fnn, on note $\delta(\mathcal{F})$ et on appelle* degré *de $\mathcal{F}$ l'entier défini de la façon suivante.*

$$\delta(\mathcal{F}_1 \vee \mathcal{F}_2) = \delta(\mathcal{F}_1 \wedge \mathcal{F}_2) = \delta(\mathcal{F}_1) + \delta(\mathcal{F}_2) + 1$$

$$\delta(\mathcal{X}) = 0$$

$$\delta(\forall x.\mathcal{F}) = \delta(\mathcal{F}) + 1$$

$$\delta(\neg P(\overline{t})) = \delta(P(\overline{t})) = 0$$

si $\mathcal{F}_1, \mathcal{F}_2$ sont des formules non purement équationnelles et $\mathcal{X}$ est une formule équationnelle. $\diamond$

DÉFINITION 9.2.4. *Soit $\mathcal{F}$ une formule sans variable et E un ensemble de formules sans variable. On note $\mathcal{F}_E$ la formule obtenue en remplaçant chaque littéral P de $\mathcal{F}$ tel que $\neg P \in E$ par $\bot$ et chaque littéral P de E par $\top$. On note $F \lhd_E G$ si et seulement si il existe $E' \subseteq E$ tel que $F = \mathcal{G}_{E'}$.* $\diamond$

REMARQUE. De façon évidente, si $\mathcal{F}$ est une formule sans variable et L un littéral de $\mathcal{F}$ et si E est un ensemble de formules sans variable contenant $\neg L$ ou L, alors $\delta(\mathcal{F}_E) < \delta(\mathcal{F})$.

DÉFINITION 9.2.5. *Un ensemble de formules fermées en fnn $\mathfrak{H}$ est appelé un* ensemble de Hintikka *si et seulement si les conditions suivantes sont vérifiées (on note $E = \mathcal{S}(\mathfrak{H})$).*

1. *Toute formule équationnelle de $\mathfrak{H}$ est valide.*
2. *Si $L \in E$ alors $\neg L \notin E$.*
3. *Pour toute formule $d \in E$, soit $\phi^+_{\mathcal{I}_s}(d_E) \equiv \top$, soit il existe un ensemble de formules $S \subseteq E$ tel que $S \models d$ et pour toute formule $d' \in S$, $\delta(d') < \delta(d)$.*

$\diamond$

Il est nécessaire d'adapter la preuve de la satisfaisabilité des ensembles de Hintikka à la nouvelle définition.

LEMME 9.2.1. *Tout ensemble de Hintikka est satisfaisable.*

PREUVE. Soit E l'ensemble des littéraux de $\mathfrak{H}$. $\mathfrak{H}$ ne contient aucun couple de littéraux contradictoires (d'après la condition 2 dans la définition des ensembles de Hintikka), donc E est une interprétation partielle. Considérons l'interprétation de Herbrand $\mathcal{I} = \mathfrak{E}_{\mathcal{I}_s}(E)$.

Prouvons, par induction sur $\delta(\mathcal{F})$ que toute formule $\mathcal{F} \in \mathcal{S}(\mathfrak{H})$ est valide dans $\mathcal{I}$.

- Supposons que $\delta(\mathcal{F}) = 0$. Deux cas sont à distinguer.
  1. Soit $\mathcal{F}$ est un littéral. Alors, $\mathcal{F} \in E$, donc (par définition de $\mathcal{I}$) $\mathcal{I} \models \mathcal{F}$.
  2. Soit $\mathcal{F}$ est une formule équationnelle valide. Alors $\mathcal{I} \models \mathcal{F}$.
- Supposons que $\delta(\mathcal{F}) > 0$. Supposons que $\mathcal{I} \not\models \mathcal{F}$. Alors on a (par définition de $\mathcal{I}$) $\mathcal{I}_s \not\models \mathcal{F}_E$. Par définition des ensembles de Hintikka, on en déduit qu'il existe un ensemble de formules $S \in \mathcal{S}(\mathfrak{H})$ de degré strictement inférieur à $\delta(\mathcal{F})$ tel que $S \models \mathcal{F}$. Par hypothèse d'induction on en déduit que $\mathcal{I} \models S$ donc $\mathcal{I} \models \mathcal{F}$.

C.Q.F.D.

LEMME 9.2.2. *Soit $(\mathcal{T}, \mathcal{X}), (\mathcal{T}', \mathcal{X}')$ deux tableaux tels que $(\mathcal{T}, \mathcal{X}) \rightarrow_{\text{RAMCET-2}_{\mathcal{I}_s}} (\mathcal{T}', \mathcal{X}')$. Alors pour toute branche $\mathcal{B}$ de $\mathcal{T}$ et pour toute solution σ de $\mathcal{X}$, il existe une branche $\mathcal{B}'$ de $\mathcal{T}'$ et une solution σ' de $\mathcal{T}'$ telle que pour toute formule $\mathcal{F} \in \mathcal{B}\sigma$, $\mathcal{B}'\sigma' \models \mathcal{F}$.*

PREUVE. La preuve est immédiate (il suffit de considérer successivement chaque règle). C.Q.F.D.

On note RAMCET-$2_{\mathcal{I}_s ground}$ la méthode définie par les règles de RAMCET-$2_{\mathcal{I}_s}$ telle que :

- Les règles $R_\vee, R_\wedge, R_=, R_{clash}$ sont appliquées avec une priorité maximale (il est clair que ces règles sont à terminaison finie).
- La règle $R_\forall$ est ensuite appliquée une fois sur toutes les formules du tableau en choisissant un terme fermé t tel que la formule obtenue ne soit pas $\bot$ (modulo $\mathcal{R}$).
- La règle R_{simp} est ensuite appliquée une fois sur toute formule $\mathcal{F}$ telle que $\phi_{\mathcal{I}_s}(\mathcal{F}) \neq \top$ (et sur toute sous-formule de $\mathcal{F}$).

LEMME 9.2.3. RAMCET-$2_{\mathcal{I}_s ground}$ *est complet pour la réfutation. Pour toute formule $\mathcal{F}$ insatisfaisable, il existe une séquence d'application des règles de* RAMCET-$2_{\mathcal{I}_s}$ *suivant la stratégie précédente de $(\{\{\mathcal{F}\}\}, \top)$ vers un tableau fermé.*

PREUVE. La preuve est similaire à celle de (FITTING, 1990). Supposons qu'il existe une séquence infinie $(\mathcal{T}_i)_{i \in \mathbb{N}}$ de tableaux non fermés, vérifiant $\mathcal{T}_i \rightarrow_{\text{RAMCET-2}_{\mathcal{I}_s}} \mathcal{T}_{i+1}$. Alors, il existe une séquence infinie de branches $\mathcal{B}_1, \ldots \mathcal{B}_n, \ldots$, où $\mathcal{B}_{i+1}$ est déduite de $\mathcal{B}_i$ par une règle d'extension ou de simplification du tableau. Soit $\mathcal{B} = \bigcup_{i=1}^{\infty} \mathcal{B}_i$. $\mathcal{B}$ est stable par les règles d'extension du tableau. Nous allons montrer que $\mathcal{B}$ est un ensemble de Hintikka.

- Si $\mathcal{B}$ contient une formule équationnelle $\mathcal{X}$ non valide, alors on a $\mathcal{X} \equiv \bot$, donc $R_{=_1}$ s'applique.
- Par irréductibilité par R_{clash} et $R_=$, $\mathcal{B}$ ne contient aucun couple de littéraux contradictoire.
- Soit $E = \mathcal{S}(\mathcal{B})$ et E' l'ensemble des littéraux de E. Soit $d \in E$. Supposons que $\phi^+_{\mathcal{I}_s}(d_E) \not\equiv \top$.
  - S'il existe j tel que $d \in \mathcal{B}_j$, alors si $\phi^+_{\mathcal{I}_s}(d) \equiv \bot$ alors la règle $R_\vee$ ou $R_\wedge$ s'applique, donc il existe dans E un ensemble de formules S tel que $S \models d$ et $\forall d' \in S . \delta(d') < \delta(d)$. Sinon, puisque $\phi^+_{\mathcal{I}_s}(d_E) \not\equiv \top$, il existe un c-littéral L de E, non valide dans $\mathcal{I}_s$ tel que $\neg L \prec d$. Donc, par irréductibilité par R_{simp}, il existe $d' \lhd_E d$ tel que $d' \in E$ (rappelons que l'on considère la forme normale des formules par rapport à $\mathcal{R}$) Or $\delta(d') < \delta(d)$, et $E' \cup \{d'\} \models d$.

– Sinon, par définition de RAMCET-$2_{\mathcal{I}_s ground}$, il existe $d' \lhd_{E'} d$ tel que $d' \in E$. Là encore, on a $\delta(d') < \delta(d)$, et $E' \cup \{d'\} \models d$.

D'où $\mathcal{B}$ est satisfaisable C.Q.F.D.

LEMME 9.2.4. *Soit $(\mathcal{T}, \mathcal{X})$ un tableau. Soit σ une solution de $\mathcal{X}$. Alors, pour tout tableau $\mathcal{T}'$ obtenu par application des règles de* RAMCET-$2_{\mathcal{I}_s}$ *sur $(\mathcal{T}\sigma, \top)$, il existe une application des règles de* RAMCET-$2_{\mathcal{I}_s}$ *sur $(\mathcal{T}, \mathcal{X})$ produisant un tableau $(\mathcal{T}'', \mathcal{X}')$ tel qu'il existe une solution σ' de $\mathcal{X}'$ vérifiant $\mathcal{T}''\sigma' = \mathcal{T}'$.*

PREUVE. La preuve est immédiate (il suffit de considérer chaque règle). C.Q.F.D.

THÉORÈME 9.2.2. *Le calcul* RAMCET-$2_{\mathcal{I}_s}$ *est complet pour la réfutation.*

PREUVE. Soit $\mathcal{F}$ une formule insatisfaisable. Par le lemme 9.2.3, il existe une application des règles de RAMCET-$2_{\mathcal{I}_s ground}$ conduisant à un tableau fermé. D'après le lemme 9.2.4, cette dérivation peut être transformée en une dérivation non fermée utilisant les règles de RAMCET-$2_{\mathcal{I}_s}$. C.Q.F.D.

REMARQUE. Comme pour la méthode classique des tableaux, et contrairement à la méthode des connexions, le calcul que nous avons proposé est *confluent*, par conséquent la recherche de la preuve ne nécessite pas de retour-arrière.

9.3 Construction de modèles

Nous présentons maintenant une nouvelle technique permettant d'extraire un modèle d'une branche non-fermée du tableau. Beaucoup plus générale que celle présentée dans (CAFERRA ET ZABEL, 1993), elle est fondée sur l'utilisation de techniques d'induction et de généralisation. Afin d'extraire un modèle d'une branche $\mathcal{B}$, la méthode *étend* le modèle partiel contenu dans cette branche, en cherchant une interprétation $\mathcal{M}$ contenant toutes les formules atomiques de la branche. Il suffit ensuite de tester si $\mathcal{M}$ est un modèle de la formule initiale. La méthode initiale (CAFERRA ET ZABEL, 1993) considérait pour cela la fermeture universelle des formules atomiques de la branche. La méthode proposée ci-dessous est plus fine. Elle est fondée sur l'utilisation d'*ensembles de représentations* (voir chapitre 8).

9.3.1 Généralisation des ensembles de faits

Intuitivement, la $\mathcal{D}$-généralisation d'un ensemble $\mathcal{S}$ est la plus petite eq-interprétation $\mathcal{D}$-compatible contenant $\mathcal{S}$ (si elle existe).

DÉFINITION 9.3.1. *Soit $\mathcal{S}$ un ensemble de c-littéraux et $\mathcal{D}$ un ensemble de représentation. $\mathcal{S}$ est dit $\mathcal{D}$-généralisable si et seulement si pour tout $\mathcal{F} \in \mathcal{D}$:*

– *Soit $\mathcal{F} \cap S = \emptyset$.*

– *Soit $\neg\mathcal{F} \cap S = \emptyset$.*

La $\mathcal{D}$-généralisation de $\mathcal{S}$ est alors l'ensemble noté generalize$(S, \mathcal{D})$ défini par :

$$generalize(S, \mathcal{D}) = \{\mathcal{F} / \mathcal{F} \in \mathcal{D} \wedge S \cap \mathcal{F} \neq \emptyset\} \cup \{\neg\mathcal{F} / \mathcal{F} \in \mathcal{D} \wedge \neg S \cap \mathcal{F} \neq \emptyset\}$$

◇

EXEMPLE 9.3.1. Soit $\mathcal{D} = \{\forall x.P(x), \forall y.R(y)\}$. Soit $\mathcal{S} = \{P(a), \neg R(b)\}$. $\mathcal{S}$ est $\mathcal{D}$-généralisable et on a $generalize(\mathcal{S}, \mathcal{D}) = \{\forall x.P(x), \forall y.\neg R(y)\}$. En effet, $\forall x.P(x)$ et $P(a)$ ainsi que $\forall y.R(y)$ et $R(b)$ sont unifiables.

En revanche, $\mathcal{F}' = \{P(a), \neg P(b)\}$ n'est pas $\mathcal{B}$-généralisable puisque $\forall x.P(x)$ est unifiable à la fois avec $P(a)$ et $P(b)$. ◇

9.3.2 Tableaux étendus

Nous allons en premier lieu modifier la définition des tableaux de façon à associer à chaque branche un ensemble de représentation.

DÉFINITION 9.3.2. *Un* tableau étendu *est un couple* $(\mathcal{T}, \mathcal{X})$ *où*

- $\mathcal{T}$ *est un ensemble de couples* $(\mathcal{B}, \mathcal{D})$ *où* $\mathcal{B}$ *est un ensemble de formules et* $\mathcal{D}$ *un ensemble de représentation.*
- $\mathcal{X}$ *est une formule équationnelle.*

Une branche *est un ensemble* $\mathcal{B}$ *tel que* $(\mathcal{B}, \mathcal{D}) \in \mathcal{T}$. ◇

Les notions usuelles (branche fermée, $\mathcal{M}od$ etc.) ainsi que les règles d'extension du tableau peuvent être appliquées de façon immédiate aux tableaux étendus, les ensembles de représentation étant simplement préservés par les règles (par souci de concision nous ne redonnons pas toutes les définitions).

9.3.3 Procédure d'extraction du modèle

Nous définissons la procédure *Extract* qui extrait un modèle partiel d'une branche donnée.

Procedure *Extract*:
  % Extracting a **Model** of a given tableau %
INPUT:
  A branch $(\mathcal{B}, \mathcal{D})$
  An equational formula $\mathcal{X}$
OUTPUT:
  A model of $\mathcal{B}$
  or $fail$
begin
  Compute a solution σ of $Eqpb(\mathcal{T})$
  if $\mathcal{M}od(\mathcal{B}\sigma)$ is $\mathcal{D}$-generalizable
  then $\mathcal{M} := generalize(\mathcal{B}\sigma, \mathcal{D})$
  else return(fail)
  if $\exists \mathcal{F} \in \mathcal{B}\sigma . \phi^{-}_{\mathcal{M}}(\mathcal{F}) \not\equiv \bot$
  return(fail)
  else return$(\mathfrak{E}_{\mathcal{I}_s}(\mathcal{M}))$
end

Règle de génération du modèle.

$$R_{mod} \ : \frac{\mathcal{T} : ((\mathcal{B}, \mathcal{D}) \cup S, \mathcal{X})}{\mathcal{M}}$$

Si $\mathcal{M} = Extract((\mathcal{B}, \mathcal{D}), \mathcal{X}) \neq fail$ *et si pour toute formule* $\mathcal{F}$ *de* $\mathcal{B}$, $\mathcal{M} \models \mathcal{F}$.

EXEMPLE 9.3.2. Les exemples suivants illustrent le fonctionnement de la procédure *Extract*. Soit $\mathcal{I}_s = \{\forall x.\neg P(x), \forall x.\neg Q(x)\}$.

1. ***Extract* (cas d'échec 1) :** Soit $\mathcal{B} = \{\neg P(a), P(f(a))\}$. Soit $\mathcal{D} = \{\forall x.P(x)\}$. $\forall x.P(x)$ est unifiable avec $P(a)$ et $P(f(a))$ donc $\mathcal{D}$ n'est pas $\mathcal{D}$-généralisable. *Extract* échoue.
2. ***Extract* (cas d'échec 2) :** Soit $\mathcal{B} = \{\neg Q(a), P(a), Q(b) \vee Q(c)\}$ et $\mathcal{D} = \{\forall x.P(x), \forall x.Q(x)\}$. $\forall x.P(x)$ est unifiable avec $P(a)$ et $\forall x.Q(x)$ est unifiable avec $Q(a)$ donc la $\mathcal{D}$-généralisation de $\mathcal{B}$ est $\mathcal{M} = \{\forall x.P(x), \forall x.\neg Q(x)\}$. Or on a $\phi^{+}_{\mathcal{M}}(Q(b) \vee Q(c)) \equiv \bot$, donc *Extract* échoue à nouveau.

3. ***Extract* (cas de succès) :** Soit $\mathcal{B} = \{\neg Q(a), P(a), Q(b) \vee Q(c) \vee P(c)\}$ et $\mathcal{D} = \{\forall x.P(x), \forall x.Q(x)\}$. A nouveau, la $\mathcal{D}$-généralisation de $\mathcal{B}$ est $\mathcal{M} = \{\forall x.P(x), \forall x.\neg Q(x)\}$. On a $\phi^+_{\mathcal{M}}(Q(b) \vee Q(c) \vee P(c)) \equiv \top$, donc *Extract* retourne $\mathcal{M}$.

$\diamond$

9.3.4 La règle d'énumération

La règle suivante permet de modifier l'ensemble de représentation associé à une branche du tableau. Ceci est réalisé lorsque la procédure *Extract* échoue (c'est-à-dire lorsque le modèle partiel contenu dans la branche n'est pas $\mathcal{D}$-généralisable, ou lorsque le modèle calculé par *Extract* est incompatible avec la formule).

$$R_{rep} : \frac{((\mathcal{B}, \mathcal{D}) \cup S, \mathcal{X})}{((\mathcal{B}, \mathcal{D}') \cup S, \mathcal{X})}$$

Si $\mathcal{M} = Extract((\mathcal{B}, \mathcal{D}), \mathcal{X}) = fail$ *et si* $\mathcal{D} \rightarrow_{GR} \mathcal{D}'$.

9.3.5 La procédure RAMCET-2 étendue

On étend la procédure RAMCET-$2_{\mathcal{I}_s}$ en ajoutant les règles R_{mod} et R_{rep}. En outre, nous affaiblissons les conditions d'application des règles $R_\vee$, $R_\wedge$, $R_\forall$. Plus précisément, ces règles sont appliquées sur une formule $\mathcal{F}$ d'une branche $\mathcal{B}$, associée à $\mathcal{D}$ si :

- $\phi^+_{\mathcal{I}_s}(\mathcal{F}) \neq \top$.
- **ou** si $\mathcal{J} = Extract((\mathcal{B}, \mathcal{D}), \mathcal{X}) \neq fail$ et si $\phi^+_{\mathcal{J}}(\mathcal{F}) \neq \top$.

9.4 Résultats de décidabilité

Le résultat ci-dessous met en évidence l'intérêt de l'approche proposée.

THÉORÈME 9.4.1. *Soit* $\mathcal{F} \in \mathfrak{C}_{eq\text{-}model}$. *Toute application équitable des règles de* RAMCET-$2_{\mathcal{I}_s}$ *se termine sur* $\mathcal{F}$ *et retourne un eq-modèle de* $\mathcal{F}$.

PREUVE. En premier lieu, nous supposons que la règle $R_\forall$ n'introduit aucune variable libre dans le tableau.

Soit $(\mathcal{T}_i)_{i\in\mathbb{N}} = (S_i, \mathcal{X}_i)_{i\in\mathbb{N}}$ une séquence infinie de tableaux étendus vérifiant $\mathcal{T}_i \rightarrow_{\text{RAMCET-}2_{\mathcal{I}_s}} \mathcal{T}_{i+1}$ et $S_0 = (\{\{\mathcal{F}\}\}, \top)$. Soit un eq-modèle $\mathcal{K}$ de $\mathcal{F}$. Il existe une séquence infinie de branches $((\mathcal{B}_i, \mathcal{D}_i))_{i\in\mathbb{N}}$ telle que pour tout i $\mathcal{K} \models \mathcal{B}_i$. Soit $\mathcal{M}_i = \mathcal{M}od(\mathcal{B}_i)$ On note $\mathcal{B}$ et $\mathcal{M}$ les ensembles $\mathcal{B} = \bigcup_{i=1}^{\infty} \mathcal{B}_i$ et $\mathcal{M} = \bigcup_{i=1}^{\infty} \mathcal{M}_i$.

On a $\mathfrak{E}_{\mathcal{I}_s}(\mathcal{M}) \models \mathcal{F}$. On note $\mathcal{I}$ l'interprétation $\mathfrak{E}_{\mathcal{I}_s}(\mathcal{M})$.

Supposons qu'il existe $k \in \mathbb{N}$ tel que $Extract((\mathcal{B}, \mathcal{D}_k), \top) \neq fail$. Puisque $\mathcal{D}_k$ est fini, il existe un sous-ensemble fini E de $\mathcal{M}$ tel que $generalize(E, \mathcal{D}_k) = generalize(\mathcal{M}, \mathcal{D}_k)$. Par définition de $\mathcal{B}$, il existe $j \geq k$ tel que $E \subseteq \mathcal{M}_j$. On a $Extract((\mathcal{B}_j, \mathcal{D}_k), \top) = Extract((\mathcal{B}, \mathcal{D}_k), \top)$. Donc $\mathcal{D}_j = \mathcal{D}_k$ et $Extract((\mathcal{B}_j, \mathcal{D}_j), \top) = Extract((\mathcal{B}, \mathcal{D}_k), \top)$. Notons $\mathcal{J} = Extract((\mathcal{B}, \mathcal{D}_k), \top)$.

Notons E l'ensemble $\mathcal{S}(\mathcal{B})$. Soit $\mathcal{F}$ une formule de E.

Supposons que $\phi^+_{\mathcal{J}}(\mathcal{F}) \neq \top$. Alors, on montre, par induction sur $\delta(\mathcal{F})$ que c'est impossible.

- $\delta(\mathcal{F}) = 0$. $\mathcal{F}$ est soit un littéral, soit une formule équationnelle.
  - Si $\mathcal{F}$ est un littéral alors on a (par définition de $generalize(x, y)$), $\mathcal{F} \in \mathcal{J}$, d'où $\phi^+_{\mathcal{J}}(\mathcal{F}) \equiv \top$.
  - Si $\mathcal{F}$ est une formule équationnelle, $\mathcal{F}$ est nécessairement valide donc $\mathcal{J} \models \mathcal{F}$.

– $\delta(\mathcal{F}) > 0$. Alors, puisque $\phi^+_{\mathcal{J}}(\mathcal{F}) \neq \top$, il existe un ensemble de formules $S \in \mathcal{B}$ de degré strictement inférieur à $\delta(\mathcal{F})$ tel que $S \models \mathcal{F}$. Par hypothèse d'induction, on a, pour tout $\mathcal{G} \in S$, $\phi^+_{\mathcal{J}}(\mathcal{G}) = \top$, d'où $\mathcal{J} \models \mathcal{G}$.

Par conséquent, la règle R_{mod} s'applique et retourne $\mathcal{J}$.

Maintenant, supposons qu'il n'existe aucun k tel que $\mathit{Extract}((\mathcal{B}, \mathcal{D}_k), \top) \neq \mathit{fail}$. Par définition, cela implique que la dérivation contient un nombre infini d'applications des règles $\mathrm{GR}_=$ et GR_Σ. Il existe k tel que $\mathcal{K}$ est $\mathcal{D}_k$-représentable. Soit $\mathcal{K}' = \mathit{generalize}(\mathcal{B}, \mathcal{D}_k)$. On a $\mathcal{K}' \subseteq \mathcal{K}$ donc $\phi^-_{\mathcal{K}'}(\mathcal{B}) \equiv \bot$. Donc $\mathit{Extract}((\mathcal{B}, \mathcal{D}_k), \top) \neq \mathit{fail}$, ce qui est impossible.

Maintenant, considérons le cas où $R_\forall$ introduit des termes non fermés. Soit $(\mathcal{T}_i)_{i \in \mathbb{N}}$ une séquence de tableaux étendus tels que $\mathcal{T}_i \rightarrow_{\text{RAMCET-2}_{\mathcal{I}_s}} \mathcal{T}_{i+1}$. On note $\mathcal{X}_i$ la formule $\mathit{Eqpb}(\mathcal{T}_i)$. Pour toute substitution $\sigma : \mathcal{V} \rightarrow \tau(\Sigma)$ telle que $\forall i. \mathcal{X}_i\sigma$ est valide, $\mathcal{T}_i\sigma_i$ est fermé et $\mathcal{T}_i\sigma \rightarrow_{\text{RAMCET-2}_{\mathcal{I}_s}} \mathcal{T}_{i+1}\sigma$. Par conséquent, il existe $k \in \mathbb{N}$ tel que R_{mod} s'applique sur $\mathcal{T}_i\sigma$. Donc R_{mod} s'applique sur $\mathcal{T}_i$. C.Q.F.D.

En corollaire, on en déduit que la méthode RAMCET-2 est une procédure de décision pour toute classe de formules $\mathcal{C}$ telle que toute formule satisfaisable de $\mathcal{C}$ appartient à $\mathfrak{C}_{\text{eq-model}}$[2]. Cela implique que la méthode est strictement plus puissante que toutes les approches existantes pour la construction de modèles de Herbrand.

REMARQUE. En pratique, associer à chaque branche du tableau un ensemble de représentation différent peut s'avérer coûteux. Néanmoins, il est possible d'utiliser un seul ensemble de représentation pour toutes les branches du tableau (cette stratégie est utilisée dans (PELTIER, 1997c)).

9.5 Exemples

Nous illustrons ci-dessous notre approche sur les deux exemples 7.3.4 et 7.3.5 (chapitre 7). Nous donnons simplement les principaux pas de calcul, c'est-à-dire la génération du modèle à partir de la branche (la construction du tableau ne présente pas d'intérêt ici).

EXEMPLE 9.5.1. Considérons la formule 7.3.4 définie au chapitre 7.

$$\forall x, y. P(x, y) \leftarrow P(f(x), f(y)) \wedge \neg P(a, f(x)) \wedge P(a, a)$$

En appliquant la règle R_{in} sur x, y on génère les formules :

$$P(x_1, y_1) \leftarrow P(f(x_1), f(y_1)), \neg P(a, f(x_1)), P(a, a)$$

L'ensemble de représentation initial est $\mathcal{D} = \{\forall x, y. P(x, y)\}$. Appliquons la procédure *Extract*. Il est clair que *Extract* échoue puisque $P(a, a)$ et $P(a, f(x_1))$ sont simultanément unifiables avec $\forall x, y. P(x, y)$. On applique donc la règle GR. On obtient $\mathcal{D} = \{\forall x, y : x \neq y. P(x, y), \forall x. P(x, x)\}$. La procédure *Extract* retourne alors l'interprétation $\mathcal{M}$ suivante.

$$\{\forall x, y : x \neq y : \neg P(x, y), \forall x. P(x, x)\}.$$

En effet, $\forall x, y : x \neq y : P(x, y)$ est unifiable avec $P(a, f(x_1))$ dont le complémentaire appartient à $\mathcal{D}$ et $\forall x. P(x, x)$ est unifiable avec $P(a, a)$. $\mathcal{M}$ est un modèle de la formule initiale (voir chapitre 7). ◇

EXEMPLE 9.5.2. Considérons la formule 7.3.5 (voir chapitre 7).

2. Nous donnerons au chapitre 10 quelques exemples de classes de formules ayant cette propriété.

$\forall x, y.P(x, y) - P(f(x), y)$
$\wedge P(b, b)$
$\wedge \forall x.\neg P(b, f(x))$

Les règles de RAMCET-2 génèrent (en particulier) une branche $\mathcal{B}$ contenant les faits suivants.

$$\{\forall x.\neg P(b, f(x)), P(b, b), P(f(b), b), P(f(b), f(u))\}$$

où u est une variable libre introduite par la règle $R_\forall$.

Cet ensemble de faits est $\mathcal{D}$-généralisable pour l'ensemble de représentation $\mathcal{D} = \{\forall x.P(b, f(x)), P(b, b), \forall x.P(f(x), b), \forall x, y : x \neq y.P(f(x), f(y)), \forall x.P(f(x), f(x))\}$

L'ensemble $generalize(\mathcal{B}, \mathcal{D})$ est

$$\begin{array}{l} generalize(\mathcal{B}, \mathcal{D}) = \\ \qquad \{\forall x.\neg P(b, f(x)), P(b, b), \forall x.P(f(x), b), \\ \forall x, y : x \neq y.\neg P(f(x), f(y)), \forall x.\neg P(f(x), f(x))\}. \end{array}$$

$\diamond$

9.6 Extension au premier ordre

Comme au chapitre 8, nous pouvons étendre la méthode afin de traiter des formules du premier ordre (c'est-à-dire pouvant contenir des quantificateurs universels) sans utiliser la skolémisation. Pour cela, nous allons utiliser une nouvelle règle destinée à traiter les quantificateurs $\exists$. Cette règle est similaire à la règle $R_\forall$ déjà introduite, à ceci près qu'elle introduit des disjonctions au lieu de conjonctions.

$R_\exists$:

$$\frac{(\mathcal{S} \cup \{\{\exists x.\mathcal{F}\} \cup \mathcal{B}\}, \mathcal{X})}{(\mathcal{S} \cup \{\{\exists x.(\mathcal{F} \wedge x \neq y) \vee \mathcal{F}\{x \to y\}\} \cup \mathcal{B}\}, \mathcal{X})}$$

si y est un terme (cela peut être par exemple une nouvelle variable).

Il est facile de voir que cette règle est correcte. En revanche, il est clair que la méthode ainsi obtenue *n'est pas complète pour la réfutation*, c'est-à-dire qu'il existe des formules ne possédant aucun modèle de Herbrand, mais ne possédant pas de tableau fermé. En effet, il est facile de voir que le problème de savoir si une formule du premier ordre (avec quantificateurs existentiels) admet un modèle de Herbrand *sur une signature donnée* est indécidable, et non semi-décidable.

9.7 Conclusion

L'extension de la méthode des tableaux sémantiques présentée dans ce chapitre est triple.

- D'une part, nous proposons une méthode permettant de simplifier les formules apparaissant dans une branche donnée.

- D'autre part, nous donnons une nouvelle méthode pour extraire un modèle d'une branche ouverte du tableau.

- Enfin, nous proposons une méthode permettant d'intégrer l'utilisation de stratégies sémantiques.

Ces techniques augmentent strictement le pouvoir de la méthode RAMCET-1, c'est-à-dire la classe de modèles constructibles. Il convient de souligner la *modularité* de l'approche. La procédure d'extraction du modèle *Extract* est indépendante de la procédure utilisée pour générer les ensembles de faits. Par conséquent, elle peut être facilement combinée avec n'importe quel algorithme énumérant les modèles de Herbrand (par exemple les démonstrateurs par tableaux, SATCHMO, les Hyper-Tableaux, RAMC, etc.). De plus, les techniques présentées ici peuvent être généralisées à d'autres formalismes de représentation de modèles tels que les *termes avec exposants entiers* ou les *automates d'arbres* étudiés dans la partie III. Il suffit pour cela de modifier la définition des ensembles de représentation, en ajoutant de nouvelles règles au système GR afin de prendre en compte les nouvelles caractéristiques du langage (par exemple dans le cas des I-termes, pour prendre en compte les constructions de la forme $t^n.u$).

Chapitre 10

Construction de modèles et procédures de décision

"Que le poète obscur persévère dans son obscurité s'il veut trouver la lumière"
Jean PAULHAN.

Dans ce chapitre, nous identifions de façon plus précise certaines classes de formules pour lesquelles les méthodes proposées se terminent et sont capables de construire des modèles. Nous donnons quelques exemples de classes dont toutes les formules satisfaisables appartiennent à $\mathfrak{C}_{\text{eq-model}}$. En corollaire, la méthode RAMCET est une procédure de décision *uniforme* pour toutes ces classes.

10.1 Arbres de dérivation

Cette section introduit quelques définitions et notations qui seront utiles par la suite. Le concept d'*arbre de dérivation* sera utilisé pour représenter et manipuler les dérivations utilisant les c-règles de réfutation.

DÉFINITION 10.1.1. *Un arbre de dérivation est un arbre étiqueté par des c-clauses tel que :*

- *Toute c-clause d'un nœud donné qui n'est pas une feuille peut être obtenue à partir de ses fils par application de la règle* c-résolution *ou* c-factorisation.
- *La c-clause à la racine est différente de* $\top$.

L'ensemble des c-clauses d'entrée *d'un arbre de dérivation est l'ensemble des c-clauses contenues dans les feuilles de l'arbre. La* conclusion *d'un arbre de dérivation* $\mathcal{T}$ *est la c-clause* $[\![R \ : \mathcal{X}]\!]$ *apparaissant à la racine. On note* $\mathcal{M}_{\mathcal{T}}$ *la formule* $\mathcal{X}$.

Un arbre de dérivation $\mathcal{T}$ *est appelé un* arbre de réfutation *si et seulement si sa conclusion est* $\square$ *(la c-clause vide). Un arbre de dérivation* $\mathcal{T}$ *est un arbre de dérivation* d'un ensemble de c-clauses S *si toute c-clause d'entrée de* $\mathcal{T}$ *est une variante d'une c-clause de* S. *Un arbre de dérivation est dit* fermé *s'il ne contient que des c-clauses fermées.* ◇

REMARQUE. Sans perte de généralité, on suppose que toutes les c-clauses d'entrée d'un arbre de dérivation ne partagent aucune variable.

Le lemme suivant permet d'effectuer certaines transformations sur les arbres de dérivations.

LEMME 10.1.1. *Soit* $S = \{[\![C_1 \ : \mathcal{X}_1 \wedge \mathcal{Y}_1]\!], \ldots, [\![C_n \ : \mathcal{X}_n \wedge \mathcal{Y}_n]\!]\}$. *Soit* $\mathcal{T}$ *un arbre de dérivation de* S.

$\mathcal{M}_\mathcal{T}$ est de la forme $\mathcal{Z} \wedge \bigwedge_{i=1}^{n} \mathcal{Y}_i$.

Soit
$S_2 = \{[\![C_1 \vee R_1 \ : \mathcal{X}_1 \wedge \mathcal{Y}'_1]\!], \ldots, [\![C_n \vee R_n \ : \mathcal{X}_n \wedge \mathcal{Y}'_n]\!]\}$ *tel que* $\forall i \leq n. \mathcal{F}^+_\mathcal{I}([\![R_i \ : \mathcal{X}_i \wedge \mathcal{Y}'_i]\!]) = \bot$ *et* $\mathcal{Z} \wedge \bigwedge_{i=1}^{n} \mathcal{Y}'_i$ *est satisfaisable.*

Alors, il existe un arbre de dérivation $\mathcal{T}_2$ tel que :

- *L'ensemble de c-clauses d'entrée de $\mathcal{T}_2$ est S_2.*
- $\mathcal{M}_{\mathcal{T}_2} = \mathcal{Z} \wedge \bigwedge_{i=1}^{n} \mathcal{Y}'_i$.
- *La conclusion de $\mathcal{T}_2$ est $[\![R_1 \vee \ldots R_n \vee R \ : \mathcal{M}_{\mathcal{T}_2}]\!]$, où $[\![R \ : \mathcal{M}_\mathcal{T}]\!]$ est la conclusion de $\mathcal{T}$.*

PREUVE. La preuve est par induction sur l'ensemble des arbres de dérivation.

- **Cas de base.** La preuve est immédiate si $\mathcal{T}$ est réduit à un seul nœud.
- **Cas inductif.** $\mathcal{T}$ admet un nœud $c : [\![R \ : \mathcal{Y}]\!]$ dont les fils sont des feuilles. Sans perte de généralité, on suppose que les fils de $[\![R \ : \mathcal{Y}]\!]$ sont $[\![C_1 \ : \mathcal{X}_1 \wedge \mathcal{Y}_1]\!], \ldots, [\![C_q \ : \mathcal{X}_q \wedge \mathcal{Y}_q]\!]$ (avec $q \leq n$). Distinguons deux cas.
  - **c-résolution.** $[\![R \ : \mathcal{Y}]\!]$ est un $\mathcal{I}$-c-résolvent de $[\![C_1 \ : \mathcal{X}_1 \wedge \mathcal{Y}_1]\!], \ldots, [\![C_q \ : \mathcal{X}_q \wedge \mathcal{Y}_q]\!]$. $\mathcal{Y}$ est de la forme $\mathcal{Y}' \wedge \bigwedge_{i=1}^{q} \mathcal{Y}_i$ où $\mathcal{Y}'$ apparaît dans la conjonction $\mathcal{Z}$. Puisque $\forall i \leq n. \mathcal{F}^+_\mathcal{I}([\![R_i \ : \mathcal{X}_i \wedge \mathcal{Y}'_i]\!]) = \bot$ et $\mathcal{Z} \wedge \bigwedge_{i=1}^{n} \mathcal{Y}'_i$ est satisfaisable, il existe un $\mathcal{I}$-c-résolvent $[\![R \vee \bigvee_{i=1}^{q} .R_i \ : \mathcal{Y}' \wedge \bigwedge_{i=1}^{q} \mathcal{Y}'_i]\!]$ de $[\![C_1 \vee R_1 \ : \mathcal{X}_1 \wedge \mathcal{Y}'_1]\!], \ldots . [\![C_q \vee R \ : \mathcal{X}_q \wedge \mathcal{Y}'_q]\!]$.
    Soit $\mathcal{T}'$ l'arbre obtenu en supprimant les fils de $[\![R \ : \mathcal{Y}]\!]$. Par hypothèse d'induction il existe un arbre de dérivation $\mathcal{T}'_2$ tel que :
    - $\mathcal{M}_{\mathcal{T}'_2} \equiv \mathcal{Z} \wedge \bigwedge_{i=q+1}^{n} \mathcal{Y}'_i \wedge \bigwedge_{i=1}^{q} .\mathcal{Y}'_i$.
    - L'ensemble des c-clauses d'entrée de $\mathcal{T}'_2$ est

$$\{[\![R \vee \bigvee_{i=1}^{q} .R_i \ : \mathcal{Y}' \wedge \bigwedge_{i=1}^{q} \mathcal{Y}'_i]\!]\} \cup \{[\![C_i \vee R_i \ : \mathcal{X}_i \wedge \mathcal{Y}'_i]\!] / i > q\}.$$

    - La conclusion de $\mathcal{T}'_2$ est de la forme

$$[\![R_{q+1} \vee \ldots \vee R_n \vee \bigvee_{i=1}^{q} R_i \vee R \ : \mathcal{M}_{\mathcal{T}'_2}]\!].$$

    Considérons l'arbre de dérivation $\mathcal{T}_2$ obtenu en remplaçant le nœud

$$[\![R \vee \bigvee_{i=1}^{q} .R_i \ : \mathcal{Y}' \wedge \bigwedge_{i=1}^{q} \mathcal{Y}'_i]\!]$$

    de $\mathcal{T}'_2$ par l'arbre de racine

$$[\![R \vee \bigvee_{i=1}^{q} .R_i \ : \mathcal{Y}' \wedge \bigwedge_{i=1}^{q} \mathcal{Y}'_i]\!]$$

    et de fils

$$[\![C_1 \vee R_1 \ : \mathcal{X}_1 \wedge \mathcal{Y}'_1]\!], \ldots . [\![C_q \vee R \ : \mathcal{X}_q \wedge \mathcal{Y}'_q]\!].$$

    On a de façon évidente :
    - $\mathcal{M}_{\mathcal{T}_2} = \mathcal{M}_{\mathcal{T}'_2}$, donc $\mathcal{M}_{\mathcal{T}_2} = \mathcal{Z} \wedge \bigwedge_{i=1}^{n} \mathcal{Y}'_i$.
    - L'ensemble des c-clauses d'entrée de $\mathcal{T}_2$ est S_2.
    - La conclusion de $\mathcal{T}_2$ est identique à celle de $\mathcal{T}'_2$ i.e. $[\![R_1 \vee \ldots \vee R_n \vee R \ : \mathcal{M}_{\mathcal{T}_2}]\!]$.
  - **Etape de c-factorisation.** La preuve est similaire.

C.Q.F.D.

10.2 Classes décidables par résolution sémantique

Dans un premier temps, nous comparons notre approche, fondée sur la construction de modèles, avec des techniques de décision utilisant certaines stratégies de résolution sémantique, telles que l'hyperrésolution (FERMÜLLER ET AL., 1993). L'hyperrésolution peut être considérée comme un cas particulier de la résolution sémantique où l'interprétation choisie pour guider la recherche affecte la valeur *false* à tout atome. Dans (FERMÜLLER ET AL., 1993; LEITSCH, 1993), il est montré que l'hyperrésolution peut être utilisée comme *procédure de décision* pour une large classe de formules logiques, contenant en particulier les classes PVD, OCC1N, etc. Dans cette section, nous montrons que si $\mathcal{I}_s$ est une eq-interprétation et S un ensemble de clauses satisfaisable tel que la $\mathcal{I}_s$-résolution se termine sur S, alors S admet un eq-modèle $\mathcal{M}$, donc $S \in \mathfrak{C}_{\text{eq-model}}$.

10.2.1 L'opérateur $\mathcal{R}es$

Notons $\mathcal{R}es_{\mathcal{I}}$ la procédure définie par les règles c-factorisation, Model Explosion et $\mathcal{I}$-c-résolution.

REMARQUE. Il est clair que si la règle de résolution sémantique (au sens de (SLAGLE, 1967)) se termine sur S, alors $\mathcal{R}es^{\infty}_{\mathcal{I}}(S)$ est fini.

On note $DSub_{\subseteq}$ la procédure définie par la règle c-dissubsumption, restreinte par la stratégie suivante.

La règle de c-dissubsomption est appliquée sur une c-clause C si et seulement si il existe n c-clauses $C_1, \ldots, C_n \in S$ vérifiant les propriétés suivantes.

- $DSub(\{C_1, \ldots, C_n\}, S) = \top$.
- Pour tout littéral $L \in C_i$ il existe un c-littéral $L' \in C$ tel que $L' \subseteq L$.

Une c-clause C est dite $\subset$-*subsumée* par un ensemble E si et seulement si $DSub_{\subseteq}{}^*(C \cup E) = E$ (noté $E \leq_{\subset} C$).

EXEMPLE 10.2.1. La clause $P(x) \vee R(x)$ subsume $P(a) \vee R(a) \vee Z(a)$, mais ne $\subset$-subsume pas $P(a) \vee R(a) \vee Z(a)$. En revanche $P(x) \vee R(x)$ $\subset$-subsume $P(x) \vee R(x) \vee Q(x)$. $\diamond$

10.2.2 Résolution sémantique et construction de modèle

THÉORÈME 10.2.1. *Soit $\mathcal{I}$ une eq-interprétation. Soit S un ensemble de c-clauses telles que :*

- *La partie contrainte des clauses de S est $\top$ (donc S est un ensemble de clauses, au sens habituel).*
- *S est satisfaisable.*
- *La résolution $\mathcal{I}$-sémantique se termine sur S.*

Alors S admet un eq-modèle.

PREUVE. Notons $S_0 = DSub_{\subseteq}{}^*(\mathcal{R}es^{\infty}_{\mathcal{I}}(S))$. On a $\square \notin S_0$. Construisons un eq-modèle $\mathcal{M}$ de S. Pour cela, des techniques similaires à celles de (FERMÜLLER ET LEITSCH, 1996) sont utilisées pour extraire un eq-modèle de S. Cependant elles sont plus compliquées, car la règle de décomposition ne peut être appliquée directement, à la différence de (FERMÜLLER ET LEITSCH, 1996).

Soit $I_1(S)$, $I_2(S)$ les invariants suivants.

- $I_1(S)$ est vrai si et seulement s'il existe un ensemble de c-clauses E_S tel que toute c-clause de S est de la forme $[\![R \ : \mathcal{X}_1\theta_1 \vee \ldots \vee \mathcal{X}_n\theta_n]\!]$ où il existe une c-clause $[\![Z_1\theta_1 \vee \ldots \vee Z_n\theta_n \vee R \ : \top]\!] \in S_0$ vérifiant $[\![Z_i \ : \mathcal{X}_i]\!] \in E_S$ ($[\![R \ : \mathcal{X}_1\theta_1 \vee \ldots \vee \mathcal{X}_n\theta_n]\!]$ est dite E_S-*dominée* par $[\![Z_1\theta_1 \vee \ldots \vee Z_n\theta_n \vee R \ : \top]\!]$).
- $I_2(S)$ est vrai si et seulement si $\mathcal{R}es^\infty_{\mathcal{I}}(S)$ est fini.

Le principe de la preuve est le suivant. Nous supposons ces invariants satisfaits pour S, et nous définissons une fonction δ transformant S en un nouvel ensemble de c-clauses "plus simple" (en un sens à préciser) que S. Nous montrons ensuite que les invariants sont préservé lors du processus, c'est-à-dire que si I_1, I_2 sont satisfaits pour S, alors ils le sont également pour $\delta(S)$. Enfin, nous prouvons que ce processus ne peut se poursuivre indéfiniment, c'est-à-dire que l'application répétée de δ finit par générer un modèle de S.

DÉFINITION 10.2.1. *Une sous-c-clause $[\![L \ : \mathcal{X}]\!]$ apparaissant dans une c-clause $[\![L \vee R \ : \mathcal{X}]\!]$ de S est dite* minimale *si et seulement si pour toute c-clause $[\![L' \ : \mathcal{X}']\!] \in S$ on a :*

- *soit L' est une clause unitaire,*
- *soit il existe un c-littéral l' de $[\![L' \ : \mathcal{X}']\!]$ tel que pour tout c-littéral $l \in [\![L \ : \mathcal{X}]\!]$ on a $l' \not\subset l$.*

◇

REMARQUE. Si S est fini, alors S contient au moins une sous-c-clause minimale.

Processus de construction du modèle.

Soit S un ensemble vérifiant $S = DSub^*_{\subseteq}(\mathcal{R}es^\infty(S))$. Distinguons deux cas, suivant la forme de S.

1. Si toute c-clause non unitaire de S est vraie dans $\mathcal{I}$, alors (d'après le théorème 7.2.5), S est la représentation d'un eq-modèle de S.
2. Sinon, il existe $P = [\![L \ : \mathcal{X}]\!]$ une c-clause minimale de S, apparaissant dans une c-clause $[\![L \vee R \ : \mathcal{X}]\!]$ non valide dans $\mathcal{I}_s$.
   Soit $\mathcal{T}$ un arbre de réfutation de $S \cup [\![R \ : \mathcal{X}]\!]$. Alors il existe, d'après le lemme 10.1.1, un arbre de dérivation $\mathcal{T}'$ de S, vers une c-clause de la forme $[\![L\sigma_1 \vee \ldots \vee L\sigma_k \ : \mathcal{M}_{\mathcal{T}}]\!]$. On note (C_1) la condition suivante : *pour tout arbre de réfutation $\mathcal{T}$ de $S \cup [\![R \ : \mathcal{X}]\!]$, la conclusion de l'arbre $\mathcal{T}'$ associé est $\subset$-subsumée par un ensemble de c-clauses unitaires de S.*

   REMARQUE. Soit $\mathcal{T}$ un arbre de réfutation de $S \cup [\![R \ : \mathcal{X}]\!]$. Soit $\{[\![R \ : \mathcal{X}]\!]\sigma_i / 1 \leq i \leq k\}$ l'ensemble des variantes de $[\![R \ : \mathcal{X}]\!]$ appartenant aux c-clauses d'entrée de $\mathcal{T}$. S'il existe $i \in [1..k]$ vérifiant $[\![L \ : \mathcal{X}]\!]\sigma_i \not\equiv [\![L \ : \mathcal{M}_{\mathcal{T}}]\!]\sigma$, alors $[\![L \ : \mathcal{M}_{\mathcal{T}}]\!]\sigma_i \subset [\![L \ : \mathcal{X}]\!]$. Puisque P est minimal, $[\![L\sigma_1 \vee \ldots \vee L\sigma_k \ : \mathcal{M}_{\mathcal{T}}]\!]$ est $\subset$-subsumé par un ensemble de c-clauses unitaires de S. Par conséquent (C_1) est satisfait.

   Deux cas sont à distinguer.

   - **Cas 1.** Tout d'abord, supposons que (C_1) est valide.
     Soit E l'ensemble de c-clauses unitaires L' de S telles que L' est une instance d'un c-littéral de $[\![L \ : \mathcal{X}]\!]$. On note $[\![L \ : \mathcal{Y}]\!] = DSub(E, [\![L \ : \mathcal{X}]\!])$.
     Supposons que $S' = S \cup [\![R \ : \mathcal{Y}]\!]$ est insatisfaisable. Alors il existe un arbre de réfutation $\mathcal{T}'$ de S'. D'où (d'après le lemme 10.1.1), il existe un arbre de dérivation de S générant une c-clause de la forme $[\![L\sigma_1 \vee \ldots \vee L\sigma_n \ : \mathcal{X}']\!]$. D'après (C_1), on doit avoir $E \leq_{sub_\subseteq} [\![L\sigma_1 \vee \ldots \vee L\sigma_n \ : \mathcal{X}']\!]$.
     Soit σ une solution de $\mathcal{M}_{\mathcal{T}'}$. Il existe $i \leq n$ tel que $E \leq_{sub} L\sigma_i\sigma$. Cela signifie que $\sigma \notin \mathcal{S}(\mathcal{Y}\sigma_i)$, ce qui est impossible par définition des arbres de dérivation.
     D'où $S' = S \cup [\![R \ : \mathcal{Y}]\!]$ est satisfaisable. On note $\delta(S) = S'$.

- **Cas 2.** Supposons maintenant qu'il n'existe aucune c-clause telle que la condition (C_1) est satisfaite. Par conséquent, il existe dans $\mathcal{R}es_{\mathcal{I}}^{\infty}(S)$ un c-littéral P tel que $P \not\subseteq S$ et une c-clause C de la forme $[\![P_1 \vee \ldots \vee P_n \ : \mathcal{Y}]\!]$ où $[\![P_i \ : \mathcal{Y}]\!]$ $(1 \leq i \leq n)$ sont des variantes de P. Nous allons prouver que $S \cup \{P\}$ est satisfaisable.
  Si $S \cup \{P\}$ n'est pas satisfaisable, il existe un arbre de réfutation de $S \cup \{P\}$ ne satisfaisant pas (C_1). Soit n le nombre de variantes de P dans les c-clauses d'entrée de l'arbre de réfutation. Puisque $P \not\subseteq S$, on a $n > 1$.
  Soit $\mathfrak{E}'$ l'ensemble des c-clauses C' de $DSub_{\subseteq}{}^{*}(S)$ telles que tout c-littéral de C' est une variante de P. Soit $\mathfrak{E}$ l'ensemble des c-clauses C dérivables de S telles que C est subsumée par une c-clause de $\mathfrak{E}'$.
  Toute c-clause de $\mathfrak{E}$ est dominée par une c-clause de S_0, donc peut être transformée par les règles d'unification (i.e. décomposition et remplacement) en une c-clause de la forme : $[\![L \vee R' \ : \mathcal{X}_1 \wedge \ldots \wedge \mathcal{X}_k]\!]$, où k est borné (car S_0 est fini). . Soit k_{max} l'entier k maximal et C_{max} une c-clause de $\mathfrak{E}$ correspondant à k_{max}. Par définition, il existe $C \in \mathfrak{E}'$ tel que C subsume C_{max}. Il existe un arbre de réfutation de $S \cup \{P\}$ contenant n variantes $P\sigma_1, \ldots P\sigma_n$ de P et ne satisfaisant pas (C_1). Par conséquent, d'après le lemme 10.1.1, et puisque tout littéral de C est une variante de P, il existe un arbre de dérivation de $S \cup \{C\}$ générant une c-clause C'. D'après (C_1), $C' \in \mathfrak{E}'$. En outre, il existe un arbre de dérivation de S vers une c-clause D de la forme:

  $$[\![R\sigma_1 \vee \ldots R\sigma_n \ : (\mathcal{X}_1 \wedge \ldots \wedge \mathcal{X}_{k_{max}})\sigma_1 \wedge \ldots \wedge (\mathcal{X}_1 \wedge \ldots \wedge \mathcal{X}_{k_{max}})\sigma_n \wedge \mathcal{M}_{\mathcal{T}}]\!],$$

  où D est subsumée par C'. D peut être transformée en une clause dont la partie contrainte contient $n \times k_{max}$ sous-formules. Puisque $n > 1$ et $n \times k_{max} \leq k_{max}$, on a $k_{max} = 0$.
  D'après $I_1(S)$, on en déduit que $\mathcal{Y}$ est $\top$. Soit σ une substitution associant à toute variable de sorte s un unique terme fermé t_s de même sorte (arbitrairement choisi). On a $[\![P_1 \vee \ldots \vee P_n \ : \mathcal{Y}]\!]\sigma \equiv P_1\sigma$.
  Il existe un arbre de réfutation $\mathcal{T}$ de $S \cup \{[\![P_1 \ : \mathcal{Y}]\!]\}$ qui ne satisfait pas (C_1). Puisque C_1 est falsifiée, pour toute variante $[\![P_1 \ : \mathcal{Y}]\!]\theta$ dans $\mathcal{T}$, $\exists\overline{x}.\mathcal{M}_{\mathcal{T}}$ est valide (où $\overline{x} = \mathcal{V}ar(\mathcal{M}_{\mathcal{T}}) \setminus \mathcal{V}ar(\mathcal{Y}\theta)$). $\mathcal{M}_{\mathcal{T}}$ est un problème d'unification. Soit $\mathcal{X}'$ la forme normale de $\mathcal{M}_{\mathcal{T}}$ par rapport aux règles d'unification usuelles (décomposition, test d'occurrence, incompatibilité et remplacement). Soit y une variable n'appartenant pas à $\overline{x}$. Si $\mathcal{X}'$ contient une équation de la forme $y = f(\overline{t})$, alors $\exists\overline{x}.\mathcal{M}_{\mathcal{T}}$ n'est pas valide, ce qui est impossible. Donc $\mathcal{X}'$ ne contient que des équations de la forme $x = y$ où x et y sont des variables. Par conséquent, $\mathcal{T}\sigma$ est un arbre de réfutation et $S \cup \{[\![P_1\sigma \ : \mathcal{Y}\sigma]\!]\}$ est insatisfaisable ce qui est impossible, car S est satisfaisable et $S \models [\![P_1\sigma \ : \mathcal{Y}\sigma]\!]$.
  Par conséquent, $S \cup \{P\}$ est satisfaisable. On note $\delta(S) = S \cup \{P\}$.

Nous allons maintenant prouver que les invariants I_1, I_2 sont préservés.

1. $I_1(S)$ **est vérifié.** $I_1(S_0)$ est évidemment satisfait, puisque S_0 contient seulement des clauses. Soit $S' = \delta(S)$. Supposons que $I_1(S)$ est satisfait. Soit C une c-clause de $\mathcal{R}es_{\mathcal{I}}(S')$. C est obtenue à partir de c-clauses de S' par une dérivation de longueur k. La preuve est par induction sur k. Par construction, S' est de la forme $S \cup \{[\![R \ : \mathcal{X} \wedge \mathcal{Y}]\!]\}$ où $[\![P \vee R \ : \mathcal{X}]\!] \in S$. Soit $E_{S'} = E_S \cup \{[\![P \ : \mathcal{Y}]\!]\}$.

   - $k = 0$. Si $C \in S$, la preuve est immédiate, puisque $I_1(S)$ est vrai. Sinon, $C = [\![R \ : \mathcal{X} \wedge \mathcal{Y}]\!]$. Alors par $I_1(S)$, il existe une clause $Z \vee P' \vee R' \in S_0$ E_S-dominant $[\![P \vee R \ : \mathcal{X}]\!]$. Alors $Z \vee P' \vee R'$ $E_{S'}$-domine $[\![R \ : \mathcal{X}]\!]$.
   - $k > 0$. Soit D la dernière c-clause déduite dans la dérivation. D est déduite soit par factorisation, soit par résolution.

     - **factorisation.** D est de la forme : $[\![P(\overline{t}) \vee R \ : \mathcal{X} \wedge \overline{s} = \overline{t}]\!]$ où $D' = [\![P(\overline{t}) \vee P(\overline{s}) \vee R \ : \mathcal{X}]\!]$ est déductible de S

par une dérivation de longueur $k-1$. Par hypothèse d'induction, il existe dans S_0 une clause de la forme $Z \vee P'_1 \vee P'_2 \vee R'$ $E_{S'}$-dominant D'. Par irréductibilité par factorisation, il existe dans S_0 une clause de la forme $Z \vee P'' \vee R''$ $E_{S'}$-dominant D.

- **résolution.** La preuve est similaire.

Par conséquent, I_1 est préservé.

2. $I_2(S)$ **est vérifié.** C'est une conséquence de $I_1(S)$.
   Toute c-clause de $\mathcal{R}es_{\mathcal{I}}(S)$ est de la forme: $[\![R \ : \bigvee_{i=1}^{n} \mathcal{X}_i\theta_i]\!]$ où il existe une clause $Z_1\theta_1 \vee \ldots \vee Z_n\theta_n \vee R \in S_0$ telle que $[\![Z_i \ : \mathcal{X}_i]\!] \in E_S$. Puisque le nombre de c-clauses de S_0 est fini, il existe un nombre fini de formules distinctes $\bigvee_{i=1}^{n} \mathcal{X}_i\theta_i$ (le nombre de substitutions θ et de c-littéraux possibles sont finis). Par conséquent, le nombre de c-clauses de $\mathcal{R}es_{\mathcal{I}}(S)$ est borné.

Soit $\delta' = \delta.\mathcal{R}es$. On peut construire une séquence (éventuellement infinie) d'ensembles de c-clauses satisfaisant I_1 et I_2 et obtenus par des applications répétées de δ':

$$S, \delta'(S), \delta'(\delta'(S)), \ldots, \delta'^{n}(S) \ldots$$

On note S_i l'ensemble $\delta'^{i}(S)$. Il reste à prouver que la séquence (S_i) est finie.

Supposons que (S_i) est infinie. Soit $C_i = [\![R \ : \mathcal{Y}]\!]$ la c-clause choisie à l'étape i durant la construction de (S_i). On note $\phi(n, S)$ la cardinalité de l'ensemble des c-clauses de longueur n dans $DSub_{\subseteq}{}^{*}(S)$. m_i est la longueur maximale des c-clauses C de S_i telles qu'il existe $j > i$ et $D \in S_j$ tels que C domine D.

Pour tout i on note j_i le plus petit j tel que $j > i$ et $\exists D \in S_j, \exists C \in S_i$ tel que C domine D et $|C| = m_i$.

Soit $j > i$. Puisque toute c-clause ajoutée à S_j pendant le processus de construction de (S_j) est dominée par une clause de S_i, on a: $\forall n \geq m_i . \phi(n, S_i) \geq \phi(n, S_j)$. En outre $\phi(m_i, S_i) > \phi(m_i, S_{j_i})$. De plus, si $j > i$, on a par définition $m_j \leq m_i$. Par conséquent, il est possible de construire une séquence infinie $(j_i)_{i \in \mathbb{N}}$ telle que le couple $(m_{j_i}, \phi(m_{j_i}, S_{j_i}))$ décroît strictement. C.Q.F.D.

10.2.3 Portée et limite du théorème précédent

Le théorème 10.2.1 montre que notre méthode est plus générale que la résolution sémantique (guidée par une eq-interprétation). Il nous permet de donner une caractérisation *sémantique* d'une large classe de formules pour laquelle RAMCET est une procédure de décision. En effet, il existe de nombreuses classes de formules pour lesquelles la résolution sémantique (avec une eq-interprétation) se termine. D'après le théorème 10.2.1, notre méthode est une procédure de décision pour toutes ces classes (voir section 10.4).

Mettons maintenant en évidence quelques limites de ce théorème. Tout d'abord, on pourrait chercher à étendre la démonstration à des c-clauses quelconques au lieu de clauses standards. Cela est impossible, comme le montre l'exemple ci-dessous.

EXEMPLE 10.2.2. Soit S l'ensemble de c-clauses suivant sur la signature $\Sigma = \{0, succ\}$.

$$[\![\neg P(x,y) \vee \neg P(y,x) \ : x \neq y]\!]$$
$$[\![P(x,y) \vee P(y,x) \ : x \neq y]\!]$$

De façon évidente, la c-$\emptyset$-résolution se termine sur S et S est satisfaisable (l'interprétation $P(succ^i(0), succ^j) - \ i < j$ est un modèle de S). Cependant, *S n'admet aucun eq-modèle sur la*

signature $\{0, succ\}$. En effet :

PREUVE. Supposons que $S \in \mathfrak{C}_{\text{eq-model}}$. Soit $\mathcal{I}$ un eq-modèle de S. Sans perte de généralité, nous supposons que $\mathcal{I}$ est en forme normale. Soit p la profondeur maximale des termes de $\mathcal{I}$. Soit t_1, t_2 les termes $succ^{p+1}(0)$ et $succ^{2p+1}(0)$. On a $t_1 \neq t_2$ donc, soit $\mathcal{I} \models P(t_1, t_2)$, soit $\mathcal{I} \models P(t_2, t_1)$. Supposons que $\mathcal{I} \models P(t_1, t_2)$ alors $\mathcal{I} \models \neg P(t_2, t_1)$. Donc il existe un c-littéral L de $\mathcal{I}$ tel que $P(t_1, t_2) \subset L$. La partie clause de L est de la forme $P(succ^n(x), succ^m(y))$, avec $n \leq p$, $m \leq p$. En outre, la partie contrainte $\mathcal{X}$ de L est soit $\top$, soit $x = y$, soit $x \neq y$. $\mathcal{I} \models \neg P(t_2, t_1)$ donc $P(t_2, t_1) \not\subseteq L$. Par conséquent, on a $\mathcal{X} \not\equiv \top$. De plus, puisque $P(t_1, t_2) \subset L$, $\mathcal{X} \not\equiv x = y$. Donc $\mathcal{X} \equiv x \neq y$. Dans ce cas $P(t_2, t_1) \subset L$, ce qui est impossible. C.Q.F.D.

REMARQUE. L'interprétation de P peut être exprimée par des contraintes dans les termes avec exposants entiers (voir (PELTIER, 1997a) ou chapitre 11) : il suffit d'utiliser la formule: $P(x, y) - \exists n, m. x = succ^n(0) \wedge y = succ^m(0) \wedge n < m)$, ou par des contraintes d'ordre $(P(x, y) - x < y)$.

$\diamond$

Il est également très naturel de chercher à étendre ce résultat au cas ou la subsomption est utilisée conjointement à la résolution sémantique. Malheureusement, le théorème 10.2.1 n'en est plus un si la règle de subsomption est utilisée pour réduire l'espace de recherche, comme l'illustre l'exemple suivant.

EXEMPLE 10.2.3. Soit $S =$

$$
\begin{array}{l}
E(x, x) \\
E(x, y) \vee \neg E(succ(x), succ(y)) \\
\neg E(0, succ(x)) \\
\neg E(succ(x), 0) \\
E(x, y) \vee \neg P(x, y) \vee \neg P(y, x) \\
P(x, y) \vee P(y, x)
\end{array}
$$

Cette formule n'admet aucun eq-modèle sur la signature $\{0, succ\}$. En effet l'interprétation de E est l'égalité syntaxique sur l'univers de Herbrand. Donc, S est équivalent à

$$\{E(x, x), [\![\neg E(x, y) \ : x \neq y]\!], P(x, x), [\![\neg P(x, y) \vee \neg P(y, x) \ : x \neq y]\!], P(x, y) \vee P(y, x)\}$$

qui ne possède aucun eq-modèle, comme nous l'avons vu précédemment. Or l'hyperrésolution (cas particulier de la résolution sémantique) avec la subsomption se termine sur S. Les seules nouvelles c-clauses générées sont en effet soit $E(x, y) \vee P(x, y) \vee P(y, x)$, qui est subsumée par $P(x, y) \vee P(y, x)$, soit $E(x, x) \vee P(x, x)$ qui est subsumée par $E(x, x)$. Evidemment, l'hyperrésolution ne se termine pas sur S si la subsomption n'est pas utilisée pour réduire l'espace de recherche. $\diamond$

En revanche, il est possible de chercher à étendre ces résultats au cas ou une restriction de la subsomption est utilisée. Par exemple, nous pouvons considérer le cas où seule la subsomption unitaire est utilisée (cas fréquent en pratique, car la subsomption non unitaire est coûteuse à mettre en œuvre). Nous conjecturons que le théorème 10.2.1 est toujours valable dans ce cas.

10.3 La classe monadique

Une formule du premier ordre est dite *monadique*, si elle ne contient aucun symbole de fonction et si tous les symboles de prédicat sont d'arité 1 au plus. Comme nous l'avons vu précédemment, les ensembles de clauses satisfaisables obtenus à partir de formules de la classe monadique ne possèdent pas nécessairement de eq-modèle. Cependant, il est possible de montrer

que toute formule monadique satisfaisable appartient à $\mathfrak{C}_{\text{eq-model}}$. Pour cela, il suffit d'utiliser un algorithme de transformation en forme clausale visant à réduire la portée des quantificateurs avant skolémisation, et permettant de transformer toute formule monadique en un ensemble de clauses ne contenant aucun symbole de fonction d'arité supérieure à 1. Cette transformation est fondée sur le théorème suivant.

THÉORÈME 10.3.1. *Soit $\mathcal{F}$ une formule monadique. $\mathcal{F}$ est équivalente à une combinaison booléenne $\mathcal{F}'$ de formules de la forme $\exists x.\mathcal{M}$ ou $\forall x.\mathcal{M}$, où $\mathcal{M}$ ne contient aucun quantificateur et $\mathcal{V}ar(\mathcal{M}) = \{x\}$.*

PREUVE. Voir par exemple (OPHELDERS ET DE SWART, 1993). C.Q.F.D.

REMARQUE. La preuve du théorème précédent fournit en outre un algorithme pour construire automatiquement la formule $\mathcal{F}'$ correspondant à une formule monadique $\mathcal{F}$.

COROLLAIRE 10.3.1. *Soit $\mathcal{F}$ une formule de la classe monadique. Il existe un ensemble de clauses S sans symbole de fonction d'arité supérieure ou égale à 1 telle que $\mathcal{F}$ est satisfaisable si et seulement si S est satisfaisable, et tout modèle de S est un modèle de $\mathcal{F}$.*

PREUVE. Il suffit d'appliquer la procédure habituelle de transformation sous forme clausale sur la formule $\mathcal{F}'$, en appliquant l'opération de skolémisation avant la réduction en forme prénexe. Par définition de $\mathcal{F}'$, aucun quantificateur n'est sous la portée d'un autre quantificateur. Donc, tous les symboles de fonction introduits dans la formule sont d'arité 0. C.Q.F.D.

REMARQUE. On a alors $S \in \mathfrak{C}_{\text{eq-model}}$, d'où $\mathcal{F} \in \mathfrak{C}_{\text{eq-model}}$.

COROLLAIRE 10.3.2. *Les méthodes* RAMC *et* RAMCET *sont des procédures de décision pour la classe monadique, si la traduction définie plus haut est utilisée.*

Notons toutefois que la transformation des formules de la classe monadique en ensemble de clause sans symbole de fonction exprimée par le théorème 10.3.1 est exponentielle, donc reste de se révéler inutilisable en pratique. Par conséquent, l'utilisation de stratégies d'ordonnancement (FERMÜLLER ET AL., 1993) pour la décision de cette classe de formules semble préférable en général.

10.4 Résumé des classes décidables par notre méthode

Nous donnons ci-dessous une liste de classes de formules dont toute formule satisfaisable appartient à $\mathfrak{C}_{\text{eq-model}}$, pour lesquelles la méthode RAMCET-$2_{\mathcal{I}_s}$ est une procédure de décision. Nous rappelons succintement la définition de certaines de ces classes.

- **Classe monadique :** classe de formules ne contenant que des symboles de prédicat d'arité 1 ou 0, et ne contenant aucun symbole de fonction d'arité 1 ou plus.
- **Classe Bernays–Schönfinkel :** classe de formules de forme prénexe de la forme $\exists\overline{x}.\forall\overline{y}.\mathcal{M}$, où $\mathcal{M}$ ne contient aucun symbole de fonction d'arité 1 ou plus.
- **Classe PVD** (FERMÜLLER ET AL., 1993) : ensembles de clauses telles que pour toute variable x, la profondeur maximale d'occurrence de x dans un littéral positif est inférieure à la profondeur maximal d'occurrence de x dans un littéral négatif (modulo un renommage des littéraux). PVD contient en particulier les sous-classes de Horn KI et KII. PVD peut être décidée par hyper-résolution.

– **Classe OCC1N** (FERMÜLLER ET AL., 1993) : ensembles de clauses C telles que pour toute variable x :

1. le nombre d'occurrence de x dans des littéraux négatifs de C est inférieur ou égal à 1,

2. si x apparaît à la fois dans un littéral négatif et positif de C, alors la profondeur maximale d'occurrence de x dans un littéral positif est inférieur à la profondeur minimal d'occurrence de x dans un littéral négatif.

OCC1N contient en particulier les sous-classes de Horn D2 et D3 (FERMÜLLER ET AL., 1993). L'hyperrésolution (avec l'opération de *condensing*) (qui est un cas particulier de la subsomption) est une procédure de décision pour OCC1N.

– **Classe VED** (FERMÜLLER ET AL., 1993) : ensemble des clauses de Horn telle que toutes les occurrences d'une même variable apparaissent à des profondeurs identiques. Cette classe est décidable par hyper-résolution.

– **Classe RRSD** (KLINGENBECK, 1996).

– **Classe KPOD.** (LEITSCH, 1993). Un ensemble de clauses S appartient à la classe KPOD si toute clause C de S contient exactement 2 littéraux et si pour toute variable x le nombre d'occurrence de x dans les littéraux négatifs de C est strictement inférieur au nombre d'occurrence de x dans les littéraux positifs de C (modulo un renommage des littéraux).

– **Classe des ensembles de clauses décomposées.** Une clause est *décomposée* si deux littéraux distincts ne partagent aucune variable.

10.5 Indécidabilité du problème "S a un eq-modèle"

Afin de montrer que la méthode RAMCET est une procédure de décision pour une classe de formules donnée, il suffit de montrer que toute formule satisfaisable de cette classe admet un eq-modèle. Par conséquent, une question se pose : existe-t-il un algorithme permettant de décider si un ensemble de clauses donné admet un eq-modèle? Un tel algorithme serait d'un grand intérêt pratique, puisqu'il donnerait une caractérisation *syntaxique* de la classe de formules traitable par notre méthode. Malheureusement, ce problème est indécidable, comme le montre le théorème suivant.

THÉORÈME 10.5.1. *Le problème "$\mathcal{F} \in \mathfrak{C}_{eq\text{-}model}$?" est indécidable.*

PREUVE. On réduit le problème "$\mathcal{F} \in \mathfrak{C}_{\text{eq-model}}$?" au problème de correspondance de POST qui est connu comme indécidable.

Soit $a_1, \ldots, a_n$ et $b_1, \ldots, b_m$ deux séquences de chaînes de caractères sur un alphabet V. Soit $\Sigma = \{0, succ\} \cup V$.

Pour toute chaîne s on note $l(s)$ la longueur de l et si $i \leq l(s)$, on note $s(i)$ le i-ème symbole de s.

Considérons l'ensemble de formules suivantes.

1 $P_1(0,0)$.
2 $P_2(0,0)$.
3 $(-E(X,Y) \vee E(I,0)) \wedge P_1(X,I) \wedge P_2(Y,I) -$
$\bigvee_{i\in[1..n]} \bigwedge_{j=1}^{l(a_i)} Q(a_i(j), succ^{j-1}(X)) \wedge \bigwedge_{j=1}^{l(b_i)} Q(b_i(j), succ^{j-1}(Y))$
$\wedge P_1(succ^{l(a_i)}(X), succ(I)) \wedge P_2(succ^{l(b_i)}(Y), succ(I))$
4 $\neg Q(X,Y) \vee \neg Q(Z,Y) \vee E(X,Z)$.
5 $E(X,X)$.
6 $\bigwedge_{f\in\Sigma, a(f)=n} \bigvee_{i=1}^{n} E(X_i,Y_i) - E(f(X_1,\ldots,X_n), f(Y_1,\ldots,Y_n))$
7 $\bigwedge_{f,g\in\Sigma, f\neq g} \neg E(f(\overline{x}), g(\overline{x}))$
8 $E(X,Y) \vee -P_1(X,Z) \vee -P_1(Y,Z)$.
9 $E(X,Y) \vee -P_1(Z,X) \vee -P_1(Z,Y)$.
10 $E(X,Y) \vee -P_2(X,Z) \vee -P_2(Y,Z)$.
11 $E(X,Y) \vee -P_2(Z,X) \vee -P_2(Z,Y)$.
12 $P_1(X,Y) \Rightarrow (R(Y) - \neg R(s(Y)))$

REMARQUE. L'interprétation de E est l'égalité sur $\tau(\Sigma)$.

Nous allons montrer que $S \in \mathfrak{C}_{\text{eq-model}}$ si et seulement s'il existe une séquence d'entiers $i_1,\ldots,i_k$ satisfaisant le problème de correspondance de Post.

- Supposons qu'il existe un eq-modèle $\mathcal{M}$ de S. Sans perte de généralité, nous supposons que $\mathcal{M}$ est en forme normale. A cause de la formule 12, si pour tout n, il existe x tel que $\mathcal{M} \models P_1(x,n)$, alors pour tout n, on a $\mathcal{M} \models R(n) - \neg R(n+1)$. C'est impossible, puisque $\mathcal{M}$ est un eq-modèle. Donc, il existe n tel que $P_1(x,n)$ est faux. Supposons qu'il n'existe aucun (x,i) tel que $i>0$ et $\mathcal{M} \models P_1(x,i) \wedge P_2(x,i)$. Alors, par induction sur $\mathbb{N}$, on montre que pour tout i il existe x,y tel que $\mathcal{M} \models P_1(x,i) \wedge P_2(y,i)$ ce qui est impossible. Par conséquent, il existe $k \neq 0$ et i tels que $\mathcal{M} \models P_1(k,i) \wedge P_2(k,i)$. Par induction sur $\mathbb{N}$ et en utilisant 3 on montre que si $P_2(k,i)$ est vrai, alors pour tout $k' < k$, il existe x tel que $Q(k',x)$. D'après 4, x est unique. Soit s_i le terme tel que $Q(i,s_i)$. Soit s la chaîne $s_1.\ldots.s_{k-1}$. A cause de la formule 3 et puisque $P(0,0)$ est valide dans $\mathcal{M}$, s est une chaîne de la forme $a_{i_1}.a_{i_2}.\ldots.a_{i_k}$ (par induction sur k) et de la forme $b_{i_1}.\ldots.b_{i_k}$. Donc on a $s = a_{i_1}.a_{i_2}.\ldots.a_{i_l} = b_{i_1}.b_{i_2}.\ldots.b_{i_l}$.
- Réciproquement, supposons qu'il existe une séquence satisfaisant le problème de Post $(i_1,\ldots,i_l)$. Soit $s = a_{i_1}.\ldots.a_{i_l}$. Soit $\mathcal{M}$ le eq-modèle suivant.

$$\begin{array}{l}
\forall z < l.Q(z,s(z)).\\
\forall z \leq l.P_1(l(a_{i_1}) + \ldots + l(a_{i_z}), z)\\
\forall z \leq l.P_2(l(b_{i_1}) + \ldots + l(b_{i_z}), z)\\
\forall x,y.E(x,y) - (x \neq y)\\
\forall x \leq l / x = 2 \times k, R(x)
\end{array}$$

Il est très facile de vérifier que cet ensemble de littéraux est satisfaisable et que le modèle correspondant satisfait $S(a,b)$.

C.Q.F.D.

Le théorème 10.5.1 montre qu'il n'est pas possible de donner une caractérisation syntaxique de $\mathfrak{C}_{\text{eq-model}}$. Puisque le eq-modèle de $S(a,b)$ donné dans la preuve est également une interprétation atomique, il n'existe aucune caractérisation syntaxique de la classe $\mathfrak{C}_{\text{atomic}}$.

Partie III

Nouveaux formalismes de représentation des interprétations

Chapitre 11

Termes avec exposants entiers

11.1 Limite du pouvoir d'expression des formules équationnelles

Dans la partie précédente, nous avons proposé et étudié diverses méthodes de construction de modèles de Herbrand. Ces interprétations étaient représentées par des formules équationnelles interprétées dans la théorie vide. Mais, comme nous l'avons vu au chapitre 7, les formules équationnelles ont un pouvoir d'expression limité. Considérons la formule S suivante :

$$P(x) \vee P(f(x))$$
$$\neg P(x) \vee \neg P(f(x))$$
$$P(a)$$

S admet un unique modèle de Herbrand (sur la signature $\Sigma = \{a, f\}$) qui est :

$P(f^n(a))$ est vrai si et seulement si n est pair.

Cette interprétation ne peut pas être exprimée par une formule équationnelle, i.e. elle n'est pas une eq-interprétation (voir par exemple (Klingenbeck, 1996) pour une démonstration formelle). Cet exemple montre qu'il existe des formules satisfaisables très simples ne possèdant pas de eq-modèle. Il est donc naturel de chercher un formalisme plus expressif, permettant de représenter une plus large classe d'interprétations. Pour cela, une solution consiste à étendre la classe de formules utilisées dans les contraintes en introduisant des langages plus expressifs que les formules équationnelles du premier ordre. Les langages considérés doivent néanmoins préserver les propriétés de l'approche initiale, c'est-à-dire qu'il doit exister un algorithme pour la résolution des contraintes dans la théorie considérée. Plusieurs solutions sont envisageables afin d'étendre le langage utilisé dans la représentation du modèle. Nous pouvons tout d'abord représenter le modèle par une formule équationnelle interprétée dans une théorie non-vide E, inconnue à priori, et calculée pendant le processus de construction. Il s'agirait alors de résoudre les contraintes dans une algèbre non libre (l'algèbre initiale de la théorie considérée). Par exemple, un modèle de l'ensemble S ci-dessus peut être exprimé facilement dans la théorie $f(f(x)) = x$. Deux problèmes se posent.

1. D'une part, la résolution des contraintes dans une théorie non vide est connue comme indécidable (en général). Il faudrait alors imposer des conditions très fortes pour préserver la décidabilité de la théorie.

2. D'autre part, il faut également donner une méthode pour générer *automatiquement* de telles théories (pendant le processus de construction de modèles).

Dans (Caferra et Peltier, 1995b), nous avons proposé une méthode pour générer interactivement de telles théories, et montré comment elles peuvent être utilisées pour construire des modèles de formules pour lesquelles la méthode initiale échoue. Cependant, cette technique a l'inconvénient d'être essentiellement interactive, que ce soit pour la génération de théories (les

critères permettant le calcul de la théorie peuvent être vérifiés automatiquement, mais l'espace de recherche est tel que l'aide de l'utilisateur s'avère souvent indispensable), ou pour la résolution des contraintes. Nous n'avons donc pas retenu cette méthode (utilisée dans (Bourely, 1992; Bourely, 1998)).

Une seconde possibilité consiste à étendre le langage utilisé dans les contraintes, en choisissant une théorie particulière pour laquelle le problème de la validité est décidable. En particulier, il existe de nombreux travaux sur la représentation d'ensembles infinis de termes, qui fournissent des outils appropriés pour la représentation des interprétations de Herbrand.

11.2 Schématisation d'ensembles infinis de termes

De nombreuses procédures de calcul symbolique (telles que la méthode de résolution ou la procédure de Knuth-Bendix) sont susceptibles de diverger en générant une infinité de termes ou de formules structurellement similaires, obtenus par répétition d'un certain contexte le long d'un chemin de longueur variable. Par exemple, étant données les clauses :

$$\{P(a), \neg P(x) \vee P(f(x))\},$$

la procédure de résolution permet d'engendrer l'ensemble de c-clauses

$$\{P(f^n(a))/x \in \mathbb{N}\}.$$

Pour traiter ces ensembles, de nombreux chercheurs ont cherché des formalismes permettant d'exprimer des séquences infinies de termes similaires. Les premiers travaux dans ce domaine (Chen et al., 1990), introduisent la notion de *termes recurrents*. Il s'agit de termes dans lesquels il est possible d'itérer n fois un certain contexte. Ils permettent par exemple de représenter des ensembles de termes du type $f^n(a)$, où n est une *variable*. Par la suite, de nombreuses améliorations ont été apportées au formalisme initial : ω-termes (Chen et Hsiang, 1991), I-termes (Comon, 1992), R-termes (Salzer, 1992), et les grammaires primales (Hermann, 1992; Hermann, 1994). Une très large bibliographie, ainsi qu'une étude détaillée de l'ensemble de ces formalismes et de leurs pouvoirs d'expression respectifs peuvent être trouvés dans (Hermann, 1994; Amaniss, 1996; Amaniss et al., 1993).

Ce chapitre traite d'un formalisme de schématisation particulier : *les termes avec exposants entiers*, ou *I-termes* (Comon, 1992; Comon, 1995) ou plus exactement une extension de ce formalisme, qui est un bon compromis entre le pouvoir d'expression et la faisabilité des opérations élémentaires. En particulier, nous prouvons que la résolution des contraintes équationnelles dans les I-termes est décidable. Ce résultat de décidabilité est nouveau (seule la décidabilité du fragment positif et existentiel était connue jusqu'alors). Il est nécessaire à l'utilisation des termes avec exposants entiers pour la construction et la représentation des modèles. Il a également un intérêt en dehors de toute application à la construction de modèles. Nous définissons ensuite une nouvelle règle d'inférence permettant d'introduire de façon automatique de tels termes dans un ensemble de c-clauses du premier ordre. Cette règle permet de tirer parti du pouvoir d'expression des termes avec exposants entiers pour augmenter de façon importante la classe de modèles constructibles.

11.3 Définition et notations

11.3.1 Syntaxe des I-termes

Les termes avec exposants entiers sont une extension naturelle des termes du premier ordre. Cette extension consiste à autoriser l'itération d'un contexte t, n fois suivant un ensemble de chemins.

Nous supposons que $\mathcal{S}$ contient un symbole particulier $\mathbb{N}$. Les éléments de $\mathcal{V}_{\mathbb{N}}$ sont appelés *variables entières*. Σ est supposé contenir les symboles $0 :\rightarrow \mathbb{N}$, $succ : \mathbb{N} \rightarrow \mathbb{N}$ et $+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.

Le symbole $\diamond$ est un symbole spécial appelé "trou" et supposé distinct des éléments de $\mathcal{V}, \Sigma, \Omega$. En outre, nous supposons que Ω contient les symboles $<$ et $\leq$ de profil $\mathbb{N} \times \mathbb{N}$ et que, pour tout $f : \overline{s} \to s$, soit f est $+$ ou *succ*, soit s n'est pas $\mathbb{N}$[1].

DÉFINITION 11.3.1. *Les ensembles* $\tau_{I^i_{(s,s')}}(\Sigma, \mathcal{X})$ *des* termes de sorte s avec i trous de sorte s' *sont les plus petits ensembles vérifiant les relations suivantes.*

- $\mathcal{V}_s \cap \mathcal{X} \subseteq \tau_{I^0_{(s,s')}}(\Sigma, \mathcal{X})$.
- *Si* $\forall k \leq n$, $t_k \in \tau_{I^{i_k}_{s_k,s'}}(\Sigma, \mathcal{X})$ *et* $f : s_1, \ldots, s_n \to s$, $i = i_1 + \ldots + i_n$, *alors* $f(t_1, \ldots, t_n) \in \tau_{I^i_{(s,s')}}(\Sigma, \mathcal{X})$.
- $\diamond \in \tau_{I^1_{s,s}}(\Sigma, \mathcal{X})$
- $t^n.u \in \tau_{I^0_{(s,s'')}}(\Sigma, \mathcal{X})$ *si* n *est un terme entier,* $t \in \tau_{\mathcal{I}^i_{(s,s)}}(\Sigma, \mathcal{X})$, $i > 0$, $u \in \tau_{I^0_{(s,s')}}(\Sigma, \mathcal{X})$.

Un I-terme *est un élément de* $\tau_{I^0_{(s,s')}}(\Sigma, \mathcal{X})$ *(il est indépendant de* s'*). Pour plus de simplicité, l'ensemble des* I*-termes (resp. de sorte* s*) sera noté* $\tau_I(\Sigma, \mathcal{X})$ *(resp.* $\tau_{I_s}(\Sigma, \mathcal{X})$*).* $\diamond$

REMARQUE. La définition des I-termes introduite ci-dessus est en fait une généralisation de celle de (COMON, 1995), car elle autorise la présence de plus d'un trou dans un terme. Le langage ainsi obtenu est strictement équivalent au langage des R-termes de (SALZER, 1992) ou aux grammaires primales faibles de (HERMANN, 1994). Nous avons utilisé la notation de (COMON, 1995) car elle nous semble plus naturelle.

Par souci de clarté, nous introduisons quelques notations, qui seront utiles dans les sections suivantes.

La notion de position dans un terme est étendue aux termes de $\tau_{I^i_{(s,s')}}(\Sigma, \mathcal{X})$, par la relation $\mathcal{P}os(t^n.u) = \{\Lambda\}$. Nous notons $T_\diamond$ l'ensemble des termes de $\tau_{I^i_{(s,s')}}$. Pour tout terme $t \in T_\diamond$, $P_\diamond(t)$ dénote l'ensemble $\{p \in \mathcal{P}os(t)/t_{|p} = \diamond\}$. Les I-termes de la forme $t^n.u$ sont appelés des *N-termes*. Pour tout ensemble de positions P, $Min(P)$ dénote une position p dans P telle que pour tout $q \in P.|q| \geq |p|$ (par exemple, on peut prendre le minimum de toutes les positions de longueur minimale par rapport à l'ordre lexicographique). En particulier, $Min(t)$ dénote la position $Min(P_\diamond(t))$. Pour tout terme t, $P_N(t)$ dénote l'ensemble des positions p telles que $t_{|p}$ est un N-terme.

Nous modifions la définition de l'ordre de sous-terme sur les termes de la façon suivante.

DÉFINITION 11.3.2.

Un terme s *est dit sous-terme d'un terme* t *(noté* $s \preceq t$*) si et seulement si l'une des conditions suivantes est vérifiée.*

- $s = t$.
- t *est de la forme* $f(t_1, \ldots, t_n)$ *et il existe* i $(1 \leq i \leq n)$ *tel que* s *est un sous-terme de* t_i.
- t *est de la forme* $t_1^n.t_2$ *et* s *est un sous-terme de* t_1 *ou un sous-terme de* t_2.

s *est appelé un* sous-terme propre *(noté* $s \prec t$*) de* t *si et seulement si* s *est un sous-terme de* t *et* $s \neq t$.

$\diamond$

Les formules (resp. clauses, c-clauses) contenant des I-termes sont appelées des I-formules (resp. I-clauses, I-c-clauses).

1. Cette hypothèse est nécessaire à la décidabilité de la théorie.

11.3.2 Sémantique des I-termes

Les formules considérées sont interprétées dans la théorie obtenue par l'union de l'arithmétique de Presburger et de la théorie :

$$\{t^0.u = u, t^{n+1}.u = t(t^n.u)\}.$$

Plus précisément :

DÉFINITION 11.3.3. *Une* I-interprétation *est une interprétation* $\mathcal{I}$. *telle que :*

- *Le domaine de la sorte* $\mathbb{N}$ *est l'ensemble des entiers naturels* $0, succ(0), succ(succ(0)), \ldots$
- *L'interprétation de* $+$ *est l'addition usuelle sur* $\mathbb{N}$.
- *L'interprétation de* $<$ *et* $\leq$ *est l'ordre habituel sur* $\mathbb{N}$.
- $\forall x.\mathcal{I}(succ(x)) = succ(\mathcal{I}(x))$.
- $\mathcal{I}(0) = 0$.

Une I*-assignation est une assignation* $(\mathcal{I}, \sigma)$ *telle que* $\mathcal{I}$ *est une* I*-interprétation. L'image d'un* I*-terme par une* I*-assignation* $\mathcal{A}$ *est définie par les relations suivantes.*

- $\mathcal{A}(x) = \sigma(x)$.
- $\mathcal{A}(f(\overline{t})) = \mathcal{I}(f)(\mathcal{A}(\overline{t}))$.
- $\mathcal{A}(t^0.u) = \mathcal{A}(u)$.
- $\mathcal{A}(t^{succ(n)}.u) = \mathcal{A}(t(t^n.u))$.
- $\mathcal{A}(t^n.u) = \mathcal{A}(t^{\mathcal{A}(n)}.u)$, *si* $n \neq 0$ *et* $\forall k.n \neq succ(k)$.

La sémantique des I*-termes et des* I*-formules s'en déduit alors de façon immédiate (cf. chapitre 2). On appellera interprétation* partielle de Herbrand *une interprétation* $\langle (D_s)_{\mathcal{S}}, \mathcal{I} \rangle$ *telle que pour tout* $s \neq \mathbb{N}$, $D_s = \tau_s(\Sigma)$ *et* $\forall t \in D_s.\mathcal{I}(t) = t$.

◇

La notion de substitution s'étend de façon immédiate aux I-termes par la relation :

$$\sigma(s^n.u) = \sigma(s)^{\sigma(n)}.\sigma(u)$$

11.4 Décidabilité du problème d'unification sur les I-termes

Avant de nous intéresser à la résolution des formules du premier ordre sur les I-termes, nous allons étudier un cas particulier : les problèmes d'unification (formules positives, purement existentielles). Nous montrons la décidabilité du problème d'unification sur le langage des I-termes en étendant l'algorithme de (COMON, 1995), défini pour des termes ne contenant qu'un seul trou, à des I-termes quelconques.

DÉFINITION 11.4.1. *Un* problème d'unification *est une formule purement équationnelle du premier ordre de la forme :*

$$\exists \overline{x}. \bigwedge_{i=1}^{n} s_i = t_i.$$

◇

DÉFINITION 11.4.2. *La* frontière *de* $s = t$ *sera notée* $\mathcal{F}r(s = t)$ *et définie comme suit.*

$$
\begin{array}{ll}
\mathcal{F}r(f(t_1,\ldots,t_n) = g(s_1,\ldots s_n)) & = \bot \ (si\ f \neq g) \\
\mathcal{F}r(f(t_1,\ldots,t_n) = f(s_1,\ldots s_n)) & = \bigwedge_{i=1}^{n} \mathcal{F}r(t_i = s_i) \\
\mathcal{F}r(s = t) & = s = t \\
 & (si\ s\ ou\ t\ est\ une\ variable\ ou\ un\ N\text{-}terme \\
 & et\ s \not\preceq t\ et\ t \not\preceq s) \\
\mathcal{F}r(s = t) & = \bot \\
 & (si\ s\ ou\ t\ est\ une\ variable\ ou\ un\ N\text{-}terme \\
 & et\ s \prec t\ ou\ t \prec s) \\
\mathcal{F}r(s = s) & = \top
\end{array}
$$

◇

EXEMPLE 11.4.1. Soient deux termes $t = f(g(x,y), g(a,b))$ et $s = f(u, g(a,y))$. La frontière de $t = s$ est :

$$\mathcal{F}r(t = s) = (g(x,y) = u \wedge \top \wedge y = a)$$

i.e.

$$\mathcal{F}r(t = s) = (g(x,y) = u \wedge y = a)$$

◇

REMARQUE. La définition 11.4.2 implique que $\mathcal{F}r(s = t)$ ne contient aucune équation $s_{|q} = t_{|q}$, où $t_{|q} \prec s_{|q}$ ou $s_{|q} \prec t_{|q}$. Cette condition simplifiera l'écriture des règles d'unification.

Il est facile de voir que pour toute équation $s = t$, si toutes les expressions entières apparaissant dans les N-termes ont une valeur différente de 0 :

$$\mathcal{S}(s = t) = \mathcal{S}(\mathcal{F}r(s = t)).$$

11.4.1 Résumé de la procédure d'unification

Comme dans (COMON, 1995), nous supposons que toutes les expressions entières apparaissant dans un N-terme ont des valeurs distinctes de 0. Le cas où ces expressions sont égales à 0 doit être considéré séparément[2]. Le résumé de la procédure d'unification que nous proposons est similaire à celui de (COMON, 1995).

1. En premier lieu, on utilise les règles d'unification classiques : **Occur Check**, **Decompose**, **Clash**, **Trivial**, **Replacement**, ainsi que des règles spécifiques aux I-termes, permettant de simplifier les termes de la forme $t^0.u$ ou $t^{n+1}.u$. Les règles d'unification peuvent être étendues de façon immédiate aux I-termes. Il est clair (voir par exemple (JOUANNAUD ET KIRCHNER, 1991)) que le système de règles correspondant (noté R) est correct et à terminaison finie. Par conséquent, nous pouvons supposer sans perte de généralité que les problèmes d'unification sont irréductibles par R. Les seules équations qui restent à considérer après application des règles de R (c'est-à-dire qui sont irréductibles par R et non résolues) sont de la forme suivante.

   $$s = t_2^{n_2}.u_2$$

   où s n'est pas une variable (si une équation n'est pas de cette forme, il est facile de vérifier qu'au moins une des règles de R s'applique).

2. Nous introduisons une règle **Unfold 1** qui a pour objet de réduire le problème au cas particulier où s contient un N-terme $t_1^{n_1}.u_1$, tel que pour tout $q_i \in P_\diamond(t_2)$, il existe un préfixe p_i de q_i, tel que $s_{|p_i} = t_1^{n_1}.u_1$. La forme générale des équations obtenues après application de cette règle est illustrée par la figure 11.1.

2. Nous verrons à la section 11.6.1 que cette condition n'est pas restrictive pour l'application des I-termes à la construction de modèles.

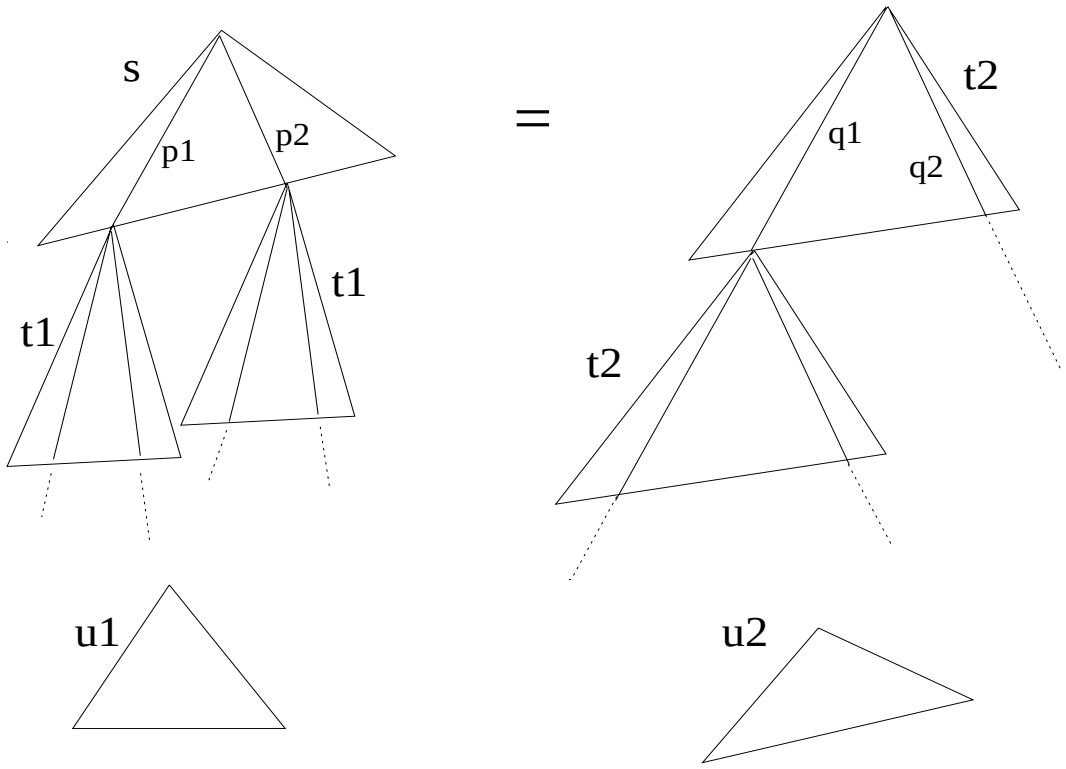


FIG. 11.1 - : *équations qui restent à considérer après* ***Unfold 1*** *(on a* $p_i <_{pref} q_i$*)*

3. Ensuite, nous déplions les termes des équations afin de nous assurer que la longueur de la position la plus courte de $P_\diamond(t_1)$ est la même que celle de la position la plus courte de $P_\diamond(t_2)$, i.e. où $|Min(t_1)| = |Min(t_2)|$, en utilisant la règle **Unfold 2**. Pour cela, il suffit de "déplier" d_1 fois le terme t_1 et d_2 fois le terme t_2, avec $d_1 \times |Min(t_1)| = d_2 \times |Min(t_2)| = d$ (d est le plus petit multiple commun de $|Min(t_1)|$ et $|Min(t_2)|$). Puis, on utilise les règles **Unfold 3**, **Clash 3**, **Replace 1, 2** pour éliminer les équations dont les positions ne sont pas comparables.

4. Enfin, nous prouvons que toutes les équations qui restent à considérer satisfont une propriété particulière permettant de définir une règle de décomposition "infinie". La forme générale des équations obtenues avant l'étape de décomposition est donnée par la figure 11.2.

   Cette règle de décomposition nous permet de simuler par une règle unique k applications successives de la règle de décomposition classique. Par exemple, l'équation $f(\diamond)^n.x = f(\diamond)^m.y$ peut être transformée en

$$(n = m \wedge x = y) \vee \exists k.(n = m + k \wedge f(\diamond)^k.x = y) \vee \exists k'.(m = n + k' \wedge x = f(\diamond)^{k'}.y).$$

Si t et s sont deux termes tels que $t \prec s$, on notera $s\{t \to t'\}$ le terme obtenu en remplaçant toutes les occurrences de t dans s par t'.

11.4.2 Règles d'unification

Dans cette section, nous définissons les règles d'unification et prouvons leur correction. Des exemples sont également donnés afin d'illustrer le fonctionnement des règles.

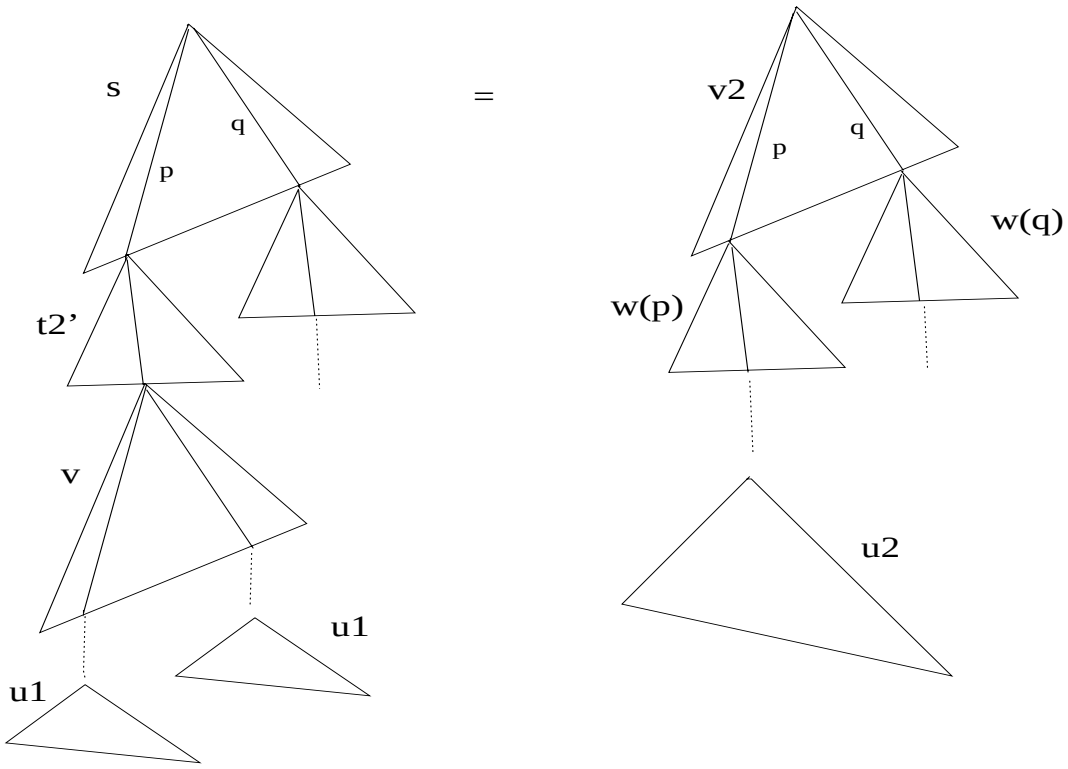


FIG. 11.2 - : *équations restant à considérer avant* ***Decompose 2***

Règles d'unification classiques

Trivial $s = s \rightarrow \top$

Decompose $f(\overline{t}) = f(\overline{s}) \rightarrow \overline{t} = \overline{s}$

Clash $f(\overline{t}) = g(\overline{s}) \rightarrow \bot$ (si $f \neq g$)

Variable Elimination $x = s \wedge P \rightarrow x = s \wedge P\{x \leftarrow s\}$
si $x \notin Var(s), x \in Var(P), x \notin \mathcal{V}_{\mathbb{N}}$
et soit s n'est pas une variable, soit $s \in Var(P)$.

Occur Check $s = t \rightarrow \bot$ (si $s < t$)

Règles de simplification des I-termes

Base case $t^0.u \rightarrow u$

Induction case $t^{n+1}.u \rightarrow t(t^n.u)$

Unfold 1 :

$$t^n.u = s \rightarrow (n = 1 \wedge t(u) = s) \vee (\exists m.n = m + 1 \wedge \mathcal{F}r(t(t^m).u = s)\{t_1 \leftarrow t_2\} \wedge t_1 = t_2)$$

si m est une nouvelle variable entière, **Clash 2** ne s'applique pas et au moins l'une des conditions suivantes est vérifiée.

1. $\mathcal{F}r(t(t^m.u) = s) = \bot$
2. $(t_1 = t_2) \in \mathcal{F}r(s = t(t^m.u))$ et $t_1 = t^m.u$.

3. $(t_1 = t_2) \in \mathcal{F}r(s = t(t^m.u))$ et t_1 est une variable.
4. $(t_1 = t_2) \in \mathcal{F}r(s = t(t^m.u))$ et t_1 est un N-terme et il existe un N-terme t' dans s tel que t' n'apparaît pas dans $t_1 = t_2$.

REMARQUE. Si $\mathcal{F}r(t(t^m.u) = s) = \bot$, la règle peut s'écrire $t^n.u = s \rightarrow n = 1 \wedge t(u) = s$. Les termes t_1 et t_2 sont inutiles dans ce cas. Néanmoins, nous considérons simultanément ces deux cas par souci de simplicité.

LEMME 11.4.1. ***Unfold 1*** *est correcte.*

PREUVE. Soit σ une solution de $t^n.u = s$, telle que $\sigma(n) > 1$. On a $\sigma(t^n.u) = \sigma(s)$. Soit $M \in V_N$ tel que $\sigma(M) = \sigma(n) - 1$. $\sigma(t(t^M.u)) = \sigma(s)$ donc $\mathcal{F}r(t(t^m.u) = s) \neq \bot$, et par décomposition, on a, pour tout $t_1 = t_2 \in \mathcal{F}r(t(t^m).u = s)$, $\sigma(t_1) = \sigma(t_2)$. Par substitutivité, on a $\sigma(t(t^M.u)\{t_1 \leftarrow t_2\}) = \sigma(s\{t_1 \leftarrow t_2\})$. La réciproque est immédiate. C.Q.F.D.

EXEMPLE 11.4.2. Considérons le problème $f(f(\diamond,\diamond)^{k_1}.a, f(\diamond,\diamond)^{k_2}.a) = f(\diamond,\diamond)^n.a$. La règle **Clash 2** ne s'applique pas. Afin de décider si **Unfold 1** s'applique, nous calculons la frontière $\mathcal{F}$ de $f(f(\diamond,\diamond)^{k_1}.a, f(\diamond,\diamond)^{k_2}.a) = f(f(\diamond,\diamond)^m.a, f(\diamond,\diamond)^m.a)$. De façon évidente, $\mathcal{F}$ contient la formule $f(\diamond,\diamond)^{k_1}.a = f(\diamond,\diamond)^m.a$. Donc **Unfold 1** s'applique et le problème est transformé en :

$$\begin{array}{l}\exists m \\ \quad (n = 1 \wedge f(f(\diamond,\diamond)^{k_1}.a, f(\diamond,\diamond)^{k_2}.a) = f(a,a)) \\ \quad \vee\ (n = m+1 \wedge (f(\diamond,\diamond)^{k_1}.a = f(\diamond,\diamond)^m.a \wedge f(\diamond,\diamond)^{k_2}.a = f(\diamond,\diamond)^{k_1}.a))\end{array}$$

$f(\diamond,\diamond)^m.a$ est remplacé par $f(\diamond,\diamond)^{k_1}.a$ dans le second terme. $\diamond$

Le lemme suivant précise la forme des équations qui restent à considérer i.e. qui sont irréductibles par rapport à R et **Unfold 1**.

LEMME 11.4.2. *Soit $s = t^n.u$ une équation irréductible par rapport à* ***Unfold 1*** *telle que s n'est pas une variable.*

1. *Pour tout $q \in P_\diamond(t)$, il existe $p \in P_N(s)$, tel que $p \leq_{pref} q$.*
2. *Pour tout $q, q' \in P_\diamond(t)$, et pour tout $p, p' \in P_N(s)$ tel que $p \leq_{pref} q$ et $p' \leq_{pref} q'$ nous avons $s_{|p} = s_{|p'} = t'$*
3. *$t' \not< t^n.u$*

PREUVE. Soit $s = t^n.u$ une équation irréductible par rapport à R et **Unfold 1**.

1. Supposons qu'il existe $q \in P_\diamond(t)$, tel que pour tout $u' \in P_N(s)$, $u' \not\leq_{pref} q$. Par irréductibilité par rapport à **Unfold 1**, $\mathcal{F}r(s = t(t^m.u))$ n'est pas $\bot$. Donc, deux cas sont à distinguer.
   (a) $q \in \mathcal{P}os(s)$: alors $(s_{|q} = t^m.u) \in \mathcal{F}r(s = t(t^m.u))$, et **Unfold 1** s'applique.
   (b) Il existe un préfixe p de q, tel que $s_{|p}$ est une variable. Alors $s_{|p} = t(t^m.u)_{|p} \in \mathcal{F}r(s = t(t^m.u))$, et **Unfold 1** s'applique.
2. Supposons qu'il existe $v, v' \in P_N(s)$, tel que $s_{|v} \neq s_{|v'}$, et $q, q' \in P_\diamond(t)$ tel que $v \leq_{pref} q$ et $v' \leq_{pref} q'$. Soit $t_1 = s_{|v}$ et $t_2 = s_{|v'}$. Nous savons que $(t_1 = t(t^m.u)_{|v})$ et $(t_2 = t(t^m.u)_{|v'})$ appartient à $\mathcal{F}r(s = t(t^m.u))$. On a soit $t_1 \not< t_2$, soit $t_2 \not< t_1$. Supposons que $t_2 \not< t_1$. Par irréductibilité par rapport à **Unfold 1**, $t(t^m.u)_{|v}$ contient t_2, donc $t^m.u > t_2$. Mais $(t_2 = t(t^m.u)_{|v'}) \in \mathcal{F}r(s = t(t^m.u))$, et $t(t^m.u)_{|v'} > t^m.u > t_2$, ce qui est impossible, par définition de $\mathcal{F}r(s = t(t^m.u))$.

3. Supposons que $t' \prec t^n.u$. Nous savons que pour tout $q \in P_\diamond(t)$, il existe $p \in P_N(s)$ tel que $p <_{pref} q$, et $s_{|p} = t'$. Alors $(t' = t(t^n.u)_{|p}) \in \mathcal{F}r(s = t(t^n.u))$. Mais $t^n.u \prec t(t^n.u)_{|p}$ donc $t' \prec t(t^n.u)_{|p}$, ce qui est impossible.

C.Q.F.D.

On appelle $P'_N(s, t_2)$ l'ensemble des positions $p \in P_N(s)$ telles qu'il existe une position $q \in P_\diamond(t_2)$ telle que $p \leq_{pref} q$. Notons s le terme $s(t_1^{n_1}.u_1)$. De façon similaire, le terme $s(t)$ dénote le terme obtenu à partir de s en remplaçant chaque occurrence de $t_1^{n_1}.u_1$, par t.

Nous réduisons maintenant le cas général au cas où les longueurs des positions $Min(t_1)$ et $Min(t_2)$ sont égales. C'est l'objet de la règle ci-dessous.

Unfold 2 :

$$\begin{aligned} s(t_1^{m_1}.u_1) = t_2^{m_2}.u_2 \rightarrow & \bigvee_{1 \leq r_1 < \alpha_2} m_1 = r_1 \wedge s(t_1^{r_1}.u_1) = t_2^{m_2}.u_2 \\ & \bigvee_{1 \leq r_2 < \alpha_1} m_2 = r_2 \wedge s(t_1^{m_1}.u_1) = t_2^{r_2}.u_2 \\ & \bigvee_{0 \leq r_2 < \alpha_1, 0 \leq r_1 < \alpha_2} \exists k_1, k_2 . m_1 = \alpha_2 \times k_1 + r_1 \\ & \wedge m_2 = \alpha_1 \times k_2 + r_2 \wedge \\ & s((t_1^{\alpha_2})^{k_1}.t_1^{r_1}.u_1) = (t_2^{\alpha_1})^{k_2}.t_2^{r_2}.u_2 \end{aligned}$$

Si $Min(t_1) \neq Min(t_2)$, d est le plus grand diviseur commun de $|Min(t_1)|$ et $|Min(t_2)|$, $\alpha_1 = \frac{|Min(t_1)|}{d}$, $\alpha_2 = \frac{|Min(t_2)|}{d}$.

LEMME 11.4.3. *La règle* ***Unfold 2*** *est correcte.*

PREUVE. La preuve est similaire à celle de (COMON, 1995). C.Q.F.D.

EXEMPLE 11.4.3. Considérons le problème $f(\diamond)^n.a = f(f(\diamond))^m.a$.

On a $|Min(f(\diamond))| = 1$ et $|Min(f(f(\diamond)))| = 2$. Donc $f(\diamond)^n.a$ doit être déplié. On obtient le problème suivant.

$(n = 1 \wedge f(a) = f(f(\diamond))^m.a)$
$\vee (\exists k . n = 2 * k + 1 \wedge f(f(\diamond))^k.f(a) = f(f(\diamond))^m.a)$
$\vee (\exists k . n = 2 * k \wedge f(f(\diamond))^k.a = f(f(\diamond))^m.a)$

$\diamond$

L'objet des règles **Unfold 3**, **Clash 3** et **Replace 1, 2** est d'éliminer les équations qui contiennent certaines positions incomparables.

Unfold 3 :

$$\begin{aligned} s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2 \rightarrow & \exists m_1, m_2 . n_1 = m_1 + 1 \wedge n_2 = m_2 + 2 \wedge t = v \\ & \wedge \mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{m_2}.u_2)))\{t \leftarrow v\} \\ & \vee (n_1 = 1 \wedge s(t_1(u_1)) = t_2^{n_2}.u_2) \\ & \vee (n_2 = 1 \wedge s(t_1^{n_1}.u_1) = t_2(u_2)) \\ & \vee (n_2 = 2 \wedge s(t_1^{n_1}.u_1) = t_2(t_2(u_2))) \end{aligned}$$

Si $\mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{m_2}.u_2))) = \perp$ ou s'il existe $t = v$ (resp. $v = t$) dans $\mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{m_2}.u_2)))$, tel que $t_2^{m_2}.u_2$ ou $t_1^{m_1}.u_1$ est un sous-terme de dans $t = v$, soit $t_2^{m_2}.u_2$ ou $t_1^{m_1}.u_1$ n'apparaît pas dans $t = v$, et t est un N-terme ou une variable.

LEMME 11.4.4. ***Unfold 3*** *est correct.*

PREUVE. Soit σ une solution de $s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$. Supposons que $\sigma(n_1) > 1$ et $\sigma(n_2) > 2$. On a $\sigma(s(t_1(t_1^{m_1}.u_1))) = \sigma(t_2(t_2(t_2^{m_2}.u_2)))$, où $\sigma(m_1) = \sigma(n_1) - 1$, et $\sigma(m_2) = \sigma(n_2) - 2$. Si $\mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{n_2}.u_2)))$ est $\perp$, on obtient une contradiction. Si $(t = v) \in \mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{n_2}.u_2)))$, on a $\sigma(t) = \sigma(v)$, donc on obtient par substitutivité

$$\sigma(s(t_1(t_1^{m_1}.u_1))\{t \leftarrow v\}) = \sigma(t_2(t_2(t_2^{m_2}.u_2))\{t \leftarrow v\}).$$

C.Q.F.D.

Le lemme suivant sera souvent utilisé par la suite.

LEMME 11.4.5. *Soit v, v', t, t' quatre termes. Soit P l'ensemble des positions p telles que $v_{|p} = t$, et P' l'ensemble des positions p' telles que $v'_{|p'} = t'$. Soient q, q' 2 positions comparables dans $P \times P'$. Si $v = v'$ est satisfaisable, et si $q \leq_{pref} q'$ alors pour toute position $p, p' \in P \times P'$, ou bien p et p' ne sont pas comparables, ou bien $p \leq_{pref} p'$. De même, si $q < q'$, et si p et p' sont comparables, alors $p < p'$.*

PREUVE. La preuve est immédiate. S'il existe une solution σ de $v = v'$, et deux positions comparables $p, p' \in P \times P'$, telles que $p \leq_{pref} p'$, alors $\sigma(t')$ est un sous-terme de $\sigma(t)$. De même, s'il existe p, p' et q, q' dans $P \times P'$ tel que $p \leq_{pref} p'$ et $q' <_{pref} q$, cela signifie que $\sigma(t)$ est un sous-terme de $\sigma(t')$ et que $\sigma(t')$ est un sous-terme de $\sigma(t)$, ce qui est impossible. C.Q.F.D.

Précisons la forme des équations qui restent à considérer.

LEMME 11.4.6. *Soit $F : s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$ une équation telle que **Unfold 3** ne s'applique pas.*

1. *Pour tout $p \in P_\diamond(t_1)$, et pour tout $u \in P'_N(s, t_2)$, il existe $q \in P_\diamond(t_2)$ tel que $u.p$ et q sont comparables.*
2. *Pour tout $q \in P_\diamond(t_2)$, il existe $p \in P_\diamond(t_1)$ et $u \in P_N(s)$, tel que $u.p$ et q sont comparables.*

PREUVE. Soit $F : s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$ une équation telle que **Unfold 3** ne s'applique pas.

1. Soit $u \in P'_N(s, t_2)$, $p \in P_\diamond(t_1)$, tel que $u.p$ et q ne sont pas comparables, pour tout $q \in P_\diamond(t_2)$. Par irréductibilité par rapport à **Unfold 3**, $\mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{m_2}.u_2)))$ n'est pas $\perp$. Supposons qu'il existe un préfixe p' de $u.p$ tel que $t : t_2(t_2(t_2^{m_2}.u_2))_{|p'}$ est soit une variable, soit un N-terme. Alors, $t \neq t_2^{m_2}.u_2$ (sinon il existerait $q, q' \in P_\diamond(t_2)$, tel que $p' = q.q'$, donc $u.p > q$, ce qui est impossible, puisque $u.p$ et q ne sont pas comparables). Donc, $t \not\geq t_2^{m_2}.u_2$ (puisque t est soit une variable, soit un N-terme apparaissant dans t_2). Par irréductibilité par rapport à **Unfold 3** c'est impossible.
   Par conséquent $s(t_1(t_1^{m_1}.u_1))_{|u.p} = t_2(t_2(t_2^{m_2}.u_2))_{|u.p}$ appartient à $\mathcal{F}r(s(t_1(t_1^{m_1}.u_1)) = t_2(t_2(t_2^{m_2}.u_2)))$ Par irréductibilité par rapport à **Unfold 3** on en déduit $t_2(t_2(t_2^{m_2}.u_2))_{|u.p} \geq t_2^{m_2}.u_2$ donc il existe $q, q' \in P_\diamond(t_2)$ tel que

   $$u.p \leq q.q'.$$

   Donc $u.p \leq q$ ou $q \leq u.p$, ce qui contredit le fait que $u.p$ et q ne sont pas comparables.
2. La preuve est similaire.

C.Q.F.D.

Clash 3:

$$s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2 \rightarrow \perp$$

S'il existe $p \in P_\diamond(t_1)$, $u \in P'_N(s, t_2)$, $q \in P_\diamond(t_2)$, telles que $u.p <_{pref} q$, et si les règles **Unfold 1, 2, 3** ne s'appliquent pas.

LEMME 11.4.7. ***Clash 3** est correct.*

PREUVE. Soit $F : s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$ une équation telle que **Unfold 1, 2, 3** ne s'appliquent pas. D'après le lemme 11.4.6, il existe $q \in P_\diamond(t_1)$ et $u \in P'_N(s, t_2)$ telles que $u.q$ et $Min(t_2)$ sont comparables. Nous avons soit $u.q <_{pref} Min(t_2)$ soit $u.q \geq_{pref} Min(t_2)$. Mais, par irréductibilité par **Unfold 2**, $|Min(t_2)| = |Min(t_1)| \leq |q|$ donc $u.q \geq_{pref} Min(t_2)$.

Toute instance satisfaisable de F est de la forme $s(t_1(t)) = t_2(t')$. De plus, on a $s(t_1(t))_{|u.q} = t$ et $t_2(t')_{|Min(t_2)} = t'$ et $u.q \geq_{pref} Min(t_2)$. Par le lemme 11.4.5, nous en déduisons que si F est satisfaisable, pour tout $p \in P_\diamond(t_1)$, pour tout $u \in P'_N(s, t_2)$ et pour tout $q \in P_\diamond(t_2)$, $u.p \not<_{pref} q$. C.Q.F.D.

A ce stade, nous savons, par irréductibilité par rapport à **Clash 3** que $\forall q \in P_\diamond(t_2).q \in \mathcal{P}os(s(t_1))$. On note $v(s, t_1, q)$ le terme $s(t_1)_{|q}$.

La règle suivante est utilisée pour forcer les ensembles $P_\diamond(v(s, t_1, q))$ à être identiques pour tout $q \in P_\diamond(t_2)$. Cette propriété sera nécessaire par la suite lors de la définition de la règle de décomposition.

Replace 1 :

$$s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2 \rightarrow s(t_1(u_1)) = t_2^{n_2}.u_2 \wedge n_1 = 1 \vee \exists m_1.n_1 = m_1 + 1 \wedge (s(t_1(v)) = t_2^{n_2}.u_2) \wedge t_1^{m_1}.u_1 = v$$

S'il existe $q, q' \in P_\diamond(t_2)$ tel que $\mathcal{F}r(v(s, t_1, q)(t_1^{m_1}.u_1) = v(s, t_1, q')(t_1^{m_1}.u_1))$ contient une équation de la forme $t_1^{m_1}.u_1 = v$ (où v est un terme quelconque).

REMARQUE. Le cas où $\mathcal{F}r(v(s, t_1, q)(t_1^{m_1}.u_1) = v(s, t_1, q')(t_1^{m_1}.u_1))$ est $\perp$ sera traité par la suite avec la règle **Decompose 2**.

LEMME 11.4.8. ***Replace 1*** *est correct.*

PREUVE. Soit σ une solution de $s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$, telle que $\sigma(n_1) > 1$. Pour tout $q \in P_\diamond(t_2)$, $\sigma(s(t_1^{n_1}.u_1))_{|q} = \sigma(t_2^{n_2}.u_2)$. Donc pour tout $q, q' \in P_\diamond(t_2)$, on a

$$\sigma(s(t_1(t_1^{m_1}.u_1))_{|q}) = \sigma(s(t_1(t_1^{m_1}.u_1))_{|q'}).$$

où $\sigma(m_1) = \sigma(n_1) - 1$. Donc

$$\sigma(v(s, t_1, q)(t_1^{m_1}.u_1)) = \sigma(v(s, t_1, q')(t_1^{m_1}.u_1)).$$

Par conséquent, la correction de **Replace 1** peut facilement être prouvée en utilisant la même technique que dans le lemme 11.4.1. C.Q.F.D.

EXEMPLE 11.4.4. Soit $\mathcal{P} : g(\diamond, a)^n.x = g(\diamond, \diamond)^m.x$. Considérons les positions $q = 1$ et $q' = 2$ dans $P_\diamond(g(\diamond, \diamond))$. La frontière de $g(g(\diamond, a)^m.x, a)_{|q} = g(g(\diamond, a)^m.x, a)_{|q'}$ est $a = g(\diamond, a)^m.x$ donc $\mathcal{P}$ est transformé par **Replace 1** en

$$(n = 1 \wedge g(x, a) = g(\diamond, \diamond)^m.x) \vee (n = m + 1 \wedge a = g(\diamond, a)^m.a \wedge g(a, a) = g(\diamond, \diamond)^m.x).$$

$\diamond$

Pour toute position $q \in P'_N(s, t_2)$, nous savons par définition de $P'_N(s, t_2)$ que $q \in \mathcal{P}os(t_2)$. $w(t_2, q)$ dénote le terme $t_{2|q}$. Nous pouvons introduire la règle **Replace 2**, similaire à **Replace 1**.

Replace 2 :

$$s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2 \rightarrow s(t_1^{n_1}.u_1) = t_2(u_2) \wedge n_2 = 1 \vee \\ \exists m_2.n_2 = m_2 + 1 \wedge s(t_1^{n_1}.u_1) = t_2(v) \wedge t_2^{m_2}.u_2 = v$$

S'il existe $q, q' \in P'_N(s, t_2)$ tel que $\mathcal{F}r(w(t_2, q).t_2^{m_2} = w(t_2, q').t_2^{m_2})$ contient une équation $v = t_2^{m_2}.u_2$.

LEMME 11.4.9. ***Replace 2*** *est correct.*

PREUVE. Soit $q \in P'_N(s, t_2)$. Soit σ une solution de $s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$, telle que $\sigma(m_2) > 1$. Nous avons, par décomposition $\sigma(s(t_1^{n_1}.u_1)_{|q}) = \sigma(t_2(t_2^{m_2}.u_1)_{|q})$ où $\sigma(m_2) = \sigma(n_2) - 1$. Donc

$$\sigma(t_1^{n_1}.u_1) = \sigma(w(t_2, q)(t_2^{m_2}.u_1))$$

(où $\sigma(m_2) = \sigma(n_2) - 1$).

Pour tout $q, q' \in P'_N(s, t_2)$, on a $\sigma(w(t_2, q)(t_2^{m_2}.u_1)) = \sigma(w(t_2, q')(t_2^{m_2}.u_1))$, et la preuve est similaire à celle du lemme 11.4.8. C.Q.F.D.

Considérons la règle suivante.

Decompose 2

$$\begin{array}{l} s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2 \rightarrow \\ \quad (n_2 = 1 \wedge s(t_1^{n_1}.u_1) = t_2(u_2)) \\ \vee \\ \quad \exists m.(n_2 = m + 1) \\ \quad \wedge \Big(\bigwedge_{q,q' \in P_\diamond(t_2)} v(s, t_1, q)(a) = v(s, t_1, q')(a) \\ \quad \wedge\ v(s, t_1, Min(t_2))(a) = s'(a) \wedge t'_2(a) = s'(a) \wedge w(t_2, Min(s))(a) = t'_1(a) \\ \quad \wedge \bigwedge_{p,p' \in P'_N(s,t_2)} w(t_2, p)(a) = w(t_2, p')(a) \\ \quad \wedge\ ((n_1 = n_2) \wedge s(u_1) = u_2) \\ \qquad \vee \exists m_1.(n_1 + m_1 = n_2 \wedge s(u_1) = t_2^{m_1}.u_2) \\ \qquad \vee \exists m_2.(n_2 + m_2 = n_1 \wedge s(t_1^{m_2}.u_1) = u_2)) \end{array}$$

où **Unfold 1, 2, 3**, **Clash 3**, **Replace 1, 2** ne s'appliquent pas, s' (resp. t'_2) est obtenu en remplaçant dans s (resp. t_2) le terme à chaque position $p \in P'_N(s, t_2)$ par $\diamond$. t'_1 est obtenu en remplaçant dans t_1 chaque terme à une position $q \in P_\diamond(t_2)$ par a.

REMARQUE. a dénote *n'importe quelle constante dans la signature* Σ. En fait, cela peut être n'importe quel terme : l'irréductibilité par rapport aux règles précédentes assure que les seules équations contenant a obtenues après le processus de normalisation seront de la forme $a = a$.

Avant de prouver la correction de **Decompose 2**, voici quelques exemples qui illustrent son fonctionnement.

EXEMPLE 11.4.5. Considérons le problème $\mathcal{P} : f(\diamond)^n.x = f(\diamond)^m.y$. Ici, les ensembles $P'_N(s, t_2)$ et $P_\diamond(t_2)$ ne contiennent qu'une position. Donc les conjonctions $\bigwedge_{q,q' \in P_\diamond(t_2)} v(s, t_1, q).a = v(s, t_1, q').a$ et $\bigwedge_{p,p' \in P'_N(s,t_2)} w(t_2, p).a = w(t_2, p').a$ sont équivalentes à $\top$.

$\mathcal{P}$ est transformé en

$$
\begin{array}{l}
\quad m = 1 \wedge f(\diamond)^n.x = f(y) \\
\vee\ \exists k.m = k + 1 \\
\quad \wedge\ a = a \wedge a = a \wedge f(a) = f(a) \\
\quad \wedge((n = m) \wedge x = y) \\
\qquad \vee\ \exists m_1.(n + m_1 = m \wedge x = f(\diamond)^{m_1}.y) \\
\qquad \vee\ \exists m_2.(n = m + m_2 \wedge f(\diamond)^{m_2}.x = y)
\end{array}
$$

$\diamond$

L'exemple suivant montre que la première ligne (i.e. la condition $m = 1$) est nécessaire.

EXEMPLE 11.4.6. Soit $\mathcal{P} = f(x, g(x, f(c, \diamond))^n.d) = f(b, g(x, \diamond))^m.f(c, d)$. Ici encore, $P'_N(s, t_2)$ et $P_\diamond(t_2)$ ne contiennent qu'une seule position. $\mathcal{P}$ est irréductible par **Unfold 1, 2, 3**, **Replace 1, 2**. $\mathcal{P}$ est transformé en :

$$
\begin{array}{l}
\quad m = 1 \wedge f(x, g(x, f(c, \diamond))^n.d) = f(b, g(x, f(c, d))) \\
\vee\ \exists k.m = k + 1 \\
\quad \wedge f(b, a) = f(x, a) \wedge f(c, a) = f(x, a) \wedge g(x, a) = g(x, a) \\
\quad \wedge((n = m) \wedge f(x, d) = f(c, d)) \\
\qquad \vee\ \exists m_1.(n + m_1 = m \wedge f(x, d) = f(b, g(x, \diamond))^{m_1}(f(c, d))) \\
\qquad \vee\ \exists m_2.(n = m + m_2 \wedge f(x, g(x, f(c, \diamond)))^{m_2}.d = f(c, d))
\end{array}
$$

A cause des équations $f(b, a) = f(x, a)$ et $f(c, a) = f(x, a)$, le second disjoint est équivalent à $\bot$. Donc $\mathcal{P}$ est équivalent à $f(x, g(x, f(c, \diamond))^n.d) = f(b, g(x, f(c, d)))$, i.e. à (par **Unfold 1** et **Decompose**). $m = 1 \wedge n = 1 \wedge x = b$.

Si nous n'avions pas considéré séparément le cas $m = 1$, nous n'aurions pas obtenu cette solution ! $\diamond$

L'exemple suivant montre l'utilité des conjonctions $\bigwedge_{q,q' \in P_\diamond(t_2)} v(s, t_1, q).a = v(s, t_1, q').a$ et $\bigwedge_{p,p' \in P'_N(s,t_2)} w(t_2, p).a = w(t_2, p').a$

EXEMPLE 11.4.7. Considérons le problème $\mathcal{P} = f(x, t_1, t_1) = f(x, g(x, \diamond, \diamond), g(c, \diamond, \diamond))^m.z$, où t_1 est $g(x, f(x, \diamond, \diamond), f(d, \diamond, \diamond))^n.z$.

$\mathcal{P}$ est transformé en :

$$
\begin{array}{l}
\quad m = 1 \wedge f(x, t_1, t_1) = f(x, g(x, z, z), g(c, z, z)) \\
\vee\ \exists k.m = k + 1 \\
\quad \wedge f(x, a, a) = f(d, a, a) \\
\quad \wedge f(x, a, a) = f(x, a, a) \wedge f(x, a, a) = f(x, a, a) \wedge g(x, a, a) = g(x, a, a) \\
\quad \wedge g(x, a, a) = g(c, a, a) \\
\quad \wedge((n = m) \wedge f(x, z, z) = z) \\
\qquad \vee \exists m_1.(n + m_1 = m \\
\qquad \wedge f(x, z, z) = f(x, g(x, \diamond, \diamond), g(c, \diamond, \diamond))^{m_1}.z) \\
\qquad \vee \exists m_1.(n = m + m_2 \\
\qquad \wedge f(x, g(x, f(x, \diamond, \diamond), f(d, \diamond, \diamond))^{m_2}.z, g(x, f(x, \diamond, \diamond), f(d, \diamond, \diamond))^{m_2}.z) = z)
\end{array}
$$

Les équations $f(x, a, a) = f(d, a, a)$ et $g(x, a, a) = g(c, a, a)$ impliquent que $c = d$ donc que le problème n'a pas de solution pour $m > 1$. Sans ces équations, nous aurions trouvé des solutions pour $m > 1$. Plus précisément, l'irréductibilité par rapport à **Replace 1, 2** assure uniquement que les positions des constantes a dans les deux termes sont identiques, mais n'assure pas que ces deux termes sont unifiables. $\diamond$

LEMME 11.4.10. ***Decompose 2*** *est correct.*

PREUVE. Soit $F = s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$ une équation irréductible par rapport à **Unfold 1, 2, 3**, **Clash 3**, **Replace 1, 2**. Supposons que σ est une solution de $s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$ telle que $\sigma(n_2) > 1$ (si $\sigma(n_2) = 1$, la correction de la règle est immédiate).

Nous avons vu que, pour tout $q, q' \in P_\diamond(t_2)$, $\sigma(v(s,t_1,q)(t_1^{m_1}.u_1)) = \sigma(v(s,t_1,q')(t_1^{m_1}.u_1))$. (où $\sigma(m_1) = \sigma(n_1) - 1$). De plus, par irréductibilité par rapport à **Replace 1** $\mathcal{F}r(v(s,t_1,q)(t_1^{m_1}.u_1) = v(s,t_1,q')(t_1^{m_1}.u_1))$ ne contient pas $t_1^{m_1}.u_1$. En effet, s'il contient $t_1^{m_1}.u_1$, nous savons, par irréductibilité par rapport à **Replace 1**, que toute équation contenant $t_1^{m_1}.u_1$ doit être de la forme $t = v$, où t est un terme apparaissant dans $t_1^{n_1}.u_1$, et $v > t_1^{m_1}.u_1$. Mais si $t \prec t_1^{n_1}.u_1$, alors $t \prec t_1^{m_1}$ et $t \prec v$, ce qui est impossible, par définition de Fr. Donc, $\sigma(v(s,t_1,q)(a)) = \sigma(v(s,t_1,q)(a))$ (pour tout terme a). Soit $v = v(s,t_1,q)\sigma$.

De même, par irréductibilité par rapport à **Replace 2**, on montre que $\sigma(w(t_2,q)(a)) = \sigma(w(t_2,q')(a))$, pour tout $q \in P'_N(s,t_2)$. Soit $w = w(t_2,q)\sigma$.

Les termes $s(t_1^{n_1}.u_1)$, et $t_2^{n_2}.u_2$ doivent être de la forme donnée par la figure 11.1. On a $\sigma(s(t_1^{n_1}.u_1)) = \sigma(t_2^{n_2}.u_2)$, donc $\sigma(s(t_1^{n_1}.u_1)) = \sigma(t'_2(w(t_2,q)(t_2^{m_2}.u_2)))$ où $q \in P'_N(s)$. La frontière de $s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$ est

$$\mathcal{F}r(s' = t'_2) \cup \{t_1^{n_1}.u_1 = w(t_2,q)(t_2^{m_2}.u_2)\}.$$

(puisque par définition, pour tout $q \in P'_N(s,t_2)$, nous avons $s_{|q} = t_1^{n_1}.u_1$ et $t_2(t_2^{m_2}.u_2)_{|q} = w(t_2,q)(t_2^{m_2}.u_2)$). Donc nous avons $\sigma(s') = \sigma(t'_2)$.

De la même façon, nous avons $\sigma(t'_1) = \sigma(w(t_2,q)(a))$, pour tout $q \in P_\diamond(t_2)$.

Il reste uniquement à montrer que $\sigma(v(s,t_1,Min(t_2))(a)) = \sigma(s')$.

Soit $s_{min} = Min(P'_N(s,t_2))$ $w_{min} = Min(P_\diamond(w))$ $v_{min} = Min(P_\diamond(v))$

En premier lieu, nous prouvons que :

$$|s_{min}| = |v_{min}|$$

Nous avons (par irréductibilité par **Unfold 2**) $|Min(t_2)| = |Min(t_1)|$. Prouvons d'abord que : $|Min(t_1)| = |w_{min}.v_{min}|$

Par définition de t'_1 et $v(s,t_1,q)$, nous avons: $p \in P_\diamond(t_1)$ si et seulement s'il existe p_1 tel que $(t'_1)_{|p_1} = a$ et $p_2 \in P_\diamond(v(s,t_1,q))$, tel que $p = p_1.p_2$.

En particulier : $Min(t_1) = p_1.p_2$, où $(t'_1)_{|p_1} = a$ et $p_2 \in P_\diamond(v)$.

Nous avons: $w_{min}.p_2 \in P_\diamond(t_2)$, donc $|Min(t_1)| \leq |w_{min}.p_2|$. i.e. : $|p_1| \leq |w_{min}|$. Par conséquent $|p_1| = |w_{min}|$. De même, on montre que: $|p_2| = |v_{min}|$. Mais $|Min(t_2)| = |Min(t_1)|$, donc

$$|Min(t_1)| = |w_{min}.v_{min}|$$

Nous avons de même: $|Min(t_2)| = |s_{min}.w_{min}|$.

Par conséquent

$$|s_{min}| = |v_{min}|$$

Nous avons: $\sigma(v(s,t_1,t_1^{m_1}.u_1)) = \sigma(s(w(t_2,t_2^{m_2}.u_1)))$

Maintenant, montrons que pour tout $p \in P_\diamond(v)$, il existe $u \in P'_N(s,t_2)$, tel que p et u sont comparables. Soit $p \in P_\diamond(v)$.

Nous avons $s_{min} \in P'_N(s,t_2)$, et $w_{min}.p \in P_\diamond(t_1)$, donc par irréductibilité par rapport à **Unfold 3** (voir le lemme 11.4.6), il existe $q, q' \in P_\diamond(t_2)$, telles que $s_{min}.w_{min}.p$ et $q.q'$ sont comparables. Nous devons avoir : $q = s_{min}.w_{min}$, donc p et q' sont comparables. Mais $q' = u.q''$, où $u \in P'_N(s,t_2)$, et p et u sont comparables.

De même, on montre que pour tout $u \in P'_N(s,t_2)$, il existe $p \in P_\diamond(v)$, tel que p et u sont comparables.

Soit $p \in P_\diamond(v)$, et $u \in P'_N(s,t_2)$. Supposons p et u comparables et $p \neq u$. Deux cas doivent être distingués.

1. $p >_{pref} u$. Il existe $u' \in P'_N(s,t_2)$, tel que v_{min} et u' sont comparables. D'après le lemme 11.4.5, on en déduit : $v_{min} >_{pref} u'$, donc $|v_{min}| > |u'|$. Ceci est impossible, puisque $|u'| \geq |s_{min}|$, et $|s_{min}| = |v_{min}|$.

2. $u' >_{pref} p$. Il existe $p' \in P_\diamond(t_1)$ tel que s_{min} et p' sont comparables. Par le lemme 11.4.5, on en déduit : $s_{min} >_{pref} p'$, donc $|s_{min}| > |p|$, ce qui est impossible.

Par conséquent, pour tout $p \in P_\diamond(t_1)$, il existe $u \in P'_N(s, t_2)$, tel que $u = p$, et pour tout $u \in P'_N(s, t_2)$, il existe $p \in P_\diamond(t_1)$, tel que $u = p$.

Donc : $\mathcal{F}r(v(s, t_1, t_1^{m_1}.u_1) = s(w(t_2, t_2^{m_2}.u_1))) = \mathcal{F}r(v(s, t_1, a) = s(a)) \cup \{t_1^{m_1}.u_1 = w(t_2, t_2^{m_2}.u_1)\}$.

Donc **Decompose 2** est correct.

C.Q.F.D.

REMARQUE. Si t_1 et t_2 contiennent seulement une occurrence de $\diamond$, il est facile de voir que notre algorithme est équivalent à celui proposé par (COMON, 1995). En effet, les règles **Replace 1, 2** et **Clash 3** ne sont jamais appliquées et les règles **Unfold 1, 2, 3** présentées dans ce chapitre deviennent équivalentes aux règles **Unfold 1, 2, 3** de (COMON, 1995) suivies d'une étape de normalisation. Donc, l'algorithme de (COMON, 1995) est un cas particulier du nôtre.

11.4.3 Terminaison et complétude

Dans cette section, nous prouvons la complétude et la terminaison du système le règles R_{unif} introduit à la section précédente. Pour cela, il faut montrer que tout problème irréductible par rapport à R_{unif} est une disjonction de formes résolues et que l'application non-déterministe des règles de R_{unif} se termine pour tout problème d'unification $\mathcal{P}$.

THÉORÈME 11.4.1. *(Terminaison) Le système R_{unif} est à terminaison finie.*

PREUVE. L'idée de la preuve est la même que celle du théorème 6.1 de (COMON, 1995). Nous donnons une interprétation I des problèmes d'unification qui sont irréductibles par rapport à R, telle que la valeur de tout problème $\mathcal{P}$ diminue par application des règles dans R_{unif}.

La définition de I est similaire, mais légèrement différente de celle utilisée dans (COMON, 1995).

Nous rappelons tout d'abord la définition de la mesure *E-size* (*Exponent Size*) sur les I-termes (COMON, 1995).

- $E\text{-}size(x) = E\text{-}size(\diamond) = E\text{-}size(a) = (0, 0)$ pour toute variable x et toute constante a
- $E\text{-}size(f(s_1, \ldots, s_n)) = max\{E\text{-}size(s_i) | 1 \leq i \leq n\}$
- $E\text{-}size(s^n.t) = max\{(n_1 + 1, 0), (m_1, m_2 + 1)\}$ si $E\text{-}size(s) = (n_1, n_2)$ et $E\text{-}size(t) = (m_1, m_2)$

L'interprétation I d'une formule atomique est la suivante :

- $I(F) = 0$, si F est une formule arithmétique, ou si F est de la forme $x = t$, où x est une variable résolue (i.e. n'apparaît qu'une seule fois dans le problème).
- $I(t_1 = t_2) = (n(t_1 = t_2), \{E\text{-}size(t_1), E\text{-}size(t_2)\}, \{|\mathcal{P}os(t_1)|, |\mathcal{P}os(t_2)|\}, status)$ où :
  1. $n(F)$ est le nombre de N-termes distincts de F.
  2. $status = 1$ si **Unfold 2** est applicable sur $t_1 = t_2$, 0 sinon.

L'interprétation d'un problème d'unification est définie par :

$$I(\exists \overline{x}.F_1 \wedge \ldots \wedge F_n) = (u, \{I(F_1), \ldots, I(F_n)\})$$

où u est le nombre de variables non résolues dans la formule.

Nous allons prouver que pour tout problème d'unification $\mathcal{P}$ et pour toute application $\mathcal{P} \to \mathcal{P}'$ d'une règle de R_{unif}, alors $I(\mathcal{P}'') < I(\mathcal{P})$, où $\mathcal{P}''$ est la forme normale de $\mathcal{P}'$ par R.

Si la règle d'élimination de variables est appliquée dans la normalisation, la décroissance est immédiate (le nombre de variables non résolues décroît strictement). De façon similaire, si une règle **Clash** ou **Occur Check** est appliquée, la décroissance est immédiate. Ces cas peuvent par conséquent être éliminés. Considérons une formule atomique $F : t = t'$. Si les règles **Clash**, **variable elimination** et **Occur Check** ne sont jamais appliquées, il est facile de voir que la forme normale de F par rapport à R est : $\bigwedge_i (t_{|q} = t_{|q})$, où $t_{|q}$ ou $t'_{|q}$ est un N-terme ou une variable (par irréductibilité par rapport à **Decompose**). Par conséquent, pour chaque règle dans R_{unif}, et pour chaque formule atomique $t = t'$ dans la partie droite des règles, nous devons prouver que $t_{|q} = t'_{|q}$ est inférieur à $t = t'$.

- **Unfold 1** : 3 types d'équations doivent être considérées :
  - $t(u) = s$: il est facile de voir que $n(t(u) = s) \leq n(t^n.u = s)$. Alors, pour chaque équation $t(u)_{|q} = s_{|q}$ dans la forme normale de $t(u) = s$, on peut montrer que $E\text{-}size(s_{|q}) \leq E\text{-}size(s)$ et que $E\text{-}size(t(u)_{|q}) < E\text{-}size(t^n.u)$.
  - $\mathcal{F}r(t(t^m.u) = s\{t_1 \leftarrow t_2\})$: puisque **Variable Elimination** ne s'applique pas, chaque équation $t(t^m.u)_{|q} = s_{|q}$ dans $\mathcal{F}r(t(t^m.u) = s\{t_1 \leftarrow t_2\})$ est irréductible par rapport à R. Si t_1 est une variable, la décroissance est immédiate (remarquons que dans ce cas le nombre de variables distinctes apparaissant dans l'équation diminue strictement). Supposons que t_1 est un N-terme. Nous avons : $n(t(t^m.u) = s\{t_1 \leftarrow t_2\}) < n(t^n.u = s)$. En effet, la règle remplace le N-terme $t^n.u$ par $t^m.u$ (par irréductibilité par rapport à **Clash 2**, s ne contient pas $t^n.u$), et remplace le N-terme t_1 par t_2, où $t_1 \not< t_2$ (par définition de la frontière).
  - $t_1 = t_2$: si t_1 est une variable, la décroissance est immédiate.
    Si t_1 est un N-terme, par définition de **Unfold 1**, deux cas sont à distinguer :
    1. $t_1 = t^m.u$: il est facile de voir alors que $\mathcal{P}os(t_2) < \mathcal{P}os(s)$ ($E\text{-}size$ est constant), et $I(t_1 = t^m.u) < I(t^n.u = s)$.
    2. Si $t_2 \geq t^m.u$: par définition de **Unfold 1**, il existe un N-terme t dans s tel que t n'apparaît pas dans $t_1 = t_2$. Donc $n(t_1 = t_2) < n(t^n.u = s)$.
- **Unfold 2** : en premier lieu, nous avons à montrer que n décroît. 3 types d'équations doivent être considérées :
  1. $s(t_1^{r_1}.u_1) = t_2^{k_2}.u_2$:
  2. $s(t_1^{k_1}.u_1) = t_2^{r_2}.u_2$: la preuve est immédiate.
  3. $s((t_1^{\alpha_2})^{m_1}.u_1) = (t_2^{\alpha_1})^{m_1}.u_2$: si n augmente alors soit $t_2^{n_2}.u_2$ contient $t_1^{n_1}.u_1$ soit s contient $t_2^{n_2}.u_1$. Par irréductibilité w.r..t **Clash**, **Unfold 1**, ceci est impossible.

  Montrons ensuite que $E\text{-}size$ ou $\mathcal{P}os()$ diminue strictement. 3 types d'équations doivent être considérées :
  1. $s(t_1^{r_1}.u_1) = t_2^{k_2}.u_2$: nous avons : $E\text{-}size(s(t_1^{r_1}.u_1)) < E\text{-}size(s(t_1^{n_1}.u_1))$.
  2. $s(t_1^{k_1}.u_1) = (t_2^{r_2}.u_2)_q$. Nous avons : $E\text{-}size(s(t_1^{k_1}.u_1)) \leq E\text{-}size(s(t_1^{n_1}.u_1))$ et $E\text{-}size((t_2^{r_2}.u_2)_q) < E\text{-}size(t_2^{n_2}.u_2)$.
  3. $s((t_1^{\alpha_2})^{m_1}.t_1^{r_1}.u_1) = (t_2^{\alpha_2})^{m_2}.t_2^{r_2}.u_2$: ici $E\text{-}size$ et $\mathcal{P}os()$ n'augmentent pas et $status$ diminue strictement.
- **Unfold 3** : nous devons simplement considérer l'équation $t_1 = t_2$ (les autres équations sont similaires à celles que nous avons déjà traitées dans les règles **Unfold 1** et **Unfold 2**).
  Par définition de **Unfold 3**, nous avons soit $t_2 \not\geq t_2^{m_2}.u_2$ soit $t_1 \not\geq t_1^{m_1}.u_1$. Donc $n(t_1 = t_2) < n(s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2)$. (nous avons : $t_2^{m_2}.u_2 \not< s$ et $t_1^{m_1}.u_1 \not< t_2^{m_2}.u_2$).

- **Clash 3** : la décroissance est immédiate
- **Replace 1 :** Deux équations doivent être considérées :
  1. $F : s(t_1(v)) = t_2^{n_2}.u_2$. Il est facile de voir que :
  $$n(F) < n(s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2)$$
  En effet, $t_1^{n_1}.u_1$ n'apparaît pas dans F, et chaque N-terme de F apparaît dans $s(t_1^{n_1}.u_1) = t_2^{n_2}.u_2$.
  2. $t_1^{m_1}.u_1 = v$. Par irréductibilité par rapport à **Clash 3**, $s(t_1^{n_1}.u_1)$ ne contient pas $t_2^{n_2}.u_2$. Puisque $v \prec s(t_1^{n_1}.u_1)$, nous en déduisons que : $t_2^{n_2}.u_2$ n'apparaît pas dans $t_1^{m_1}.u_1 = v$, donc : $n(t = v) < n(F)$.
- **Replace 2 :** La preuve est similaire à la précédente.
- **Decompose 2**
  Il est très facile de voir que n n'augmente pas, et que E-*size* diminue strictement.

C.Q.F.D.

DÉFINITION 11.4.3. *Un problème d'unification $\mathcal{P}$ est dit sous forme résolue s'il est soit $\bot$, soit $\top$, soit une disjonction de formules de la forme :*

$$\exists \overline{n}.n_1 = E_1 \wedge \ldots \wedge n_k = E_k \wedge x_1 = t_1 \wedge \ldots \wedge x_n = t_n$$

où $n_1, \ldots, n_k$ sont des variables entières apparaissant seulement une fois dans la conjonction, $E_1, \ldots, E_k$ sont des expressions linéaires et $x_1, \ldots x_n$ sont des variables n'apparaissant qu'une seule fois dans la conjonction. ◇

THÉORÈME 11.4.2. *(Complétude)*
Un problème d'unification irréductible par rapport à R_{unif} est en forme résolue.

PREUVE. Il est facile de voir que si un problème $\mathcal{P}$ n'est pas en forme résolue, alors l'une des règles de R_{unif} s'applique.

C.Q.F.D.

11.4.4 Comparaison avec l'unification des R-termes

(SALZER, 1992) propose un algorithme permettant de résoudre les problèmes d'unification sur les R-termes. Les formalismes des R-termes et des I-termes avec plusieurs trous sont équivalents dans le sens où ils permettent de réprésenter les mêmes ensembles de termes.

Une des caractéristiques de notre algorithme d'unification est que – à la différence de celui de (SALZER, 1992) – il n'utilise pas de procédure "subordonnée". En effet, la règle **cycle** de (SALZER, 1992) nécessite une procédure afin de calculer pour une équation de la forme : $s(t_1^n.u_1) = t_2^{n_2}.u_2$, 4 entiers m_0, n_0, n, m tels que :

$$\mathcal{F}r(s(t_1^{n_0+k}.u_1) = t_2^{m_0+k}.u_2) = \mathcal{F}r(s(t_1^{n+k}.u_1) = t_2^{m+k}.u_2)$$

Ceci peut être très coûteux (en temps de calcul) tout particulièrement si ces entiers sont grands : on doit en effet calculer les frontières $f_{m,n}$ de ces deux termes pour toutes les valeurs de m et n et comparer chacun des $f_{m,n}$ avec toutes les frontières calculées précédemment afin de détecter un cycle (i.e. 4 entiers m_0, n_0, n, m tels que $f_{m_0,n_0} = f_{n,m}$). Cette comparaison doit naturellement être réalisée modulo les propriétés AC du connectif $\wedge$. Dans (SALZER, 1992), il est prouvé que cette procédure se termine. L'existence d'un tel cycle permet d'appliquer une règle de décomposition similaire à la règle **Decompose 2**. Notre algorithme évite l'utilisation d'une telle procédure en déterminant directement les valeurs de m_0, n_0, n, m (c'est le rôle des règles **Unfold 2**). Cette remarque met en évidence la différence principale entre l'algorithme de SALZER et le nôtre. Des implémentations efficaces et des expérimentations sont évidemment indispensables pour pouvoir juger de l'efficacité pratique des deux algorithmes.

11.5 Problèmes équationnels sur les I-termes

Dans cette section, nous étendons les règles de la section 11.4 afin de traiter des formules du premier ordre. Pour des raisons de simplicité, nous allons tout d'abord restreindre la forme des formules à un cas particulier. Les problèmes équationnels sur les termes avec exposants entiers (ou I-problèmes équationnels) sont définis comme suit.

DÉFINITION 11.5.1. *Un* I-système *est une formule de la logique du premier ordre dont toutes les sous-formules atomiques sont de la forme :* $s = t$ *ou* $s \neq t$ *(où* s *et* t *sont des* I*-termes), ou* $s < t$, $s \leq t$, $s = t$, $s \neq t$, *où* s *et* t *sont des expressions de l'arithmétique de Presburger.*

Un problème I-équationnel *est une formule de la forme :*

$$\exists \overline{x} \forall \overline{y} M$$

où M *est un* I-système. *Les variables* $\overline{z} = \mathcal{V}ar(\exists \overline{x} \forall \overline{y} M)$ *sont les* inconnues, *les* $\overline{y}$, *les* paramètres, *et les* $\overline{x}$, *les* variables auxiliaires. ◇

La notion de solution d'un problème équationnel se définit comme pour les formules classiques. Sans perte de généralité, nous pouvons nous restreindre à des formules de cette forme (voir chapitre 2). En premier lieu, nous avons à définir précisément ce que nous entendons par "forme résolue".

11.5.1 Forme résolue

La notion de forme résolue avec contraintes (COMON ET LESCANNE, 1989) peut facilement être étendue aux I-problèmes équationnels. Cependant, il faut montrer que cette définition est toujours utilisable, c'est-à-dire que toute formule de cette forme (et différente de $\bot$) a une solution.

DÉFINITION 11.5.2.

Un I*-problème équationnel* $\mathcal{P}$ *est en* forme résolue avec contraintes, *si et seulement s'il est de l'une des trois formes suivantes :*

- $\top$
- $\bot$
- $\exists \overline{w}.[\bigwedge_{j=1}^{m} x_j = s_j] \wedge [\bigwedge_{i=1}^{k} x'_i \neq s'_i] \wedge \mathcal{F}$, *où chaque* x_j *apparaît seulement une fois dans* $\mathcal{P}$, *chaque* x'_i *est syntaxiquement différent de* s'_i *et* $\mathcal{F}$ *est une formule arithmétique satisfaisable (i.e.* $\mathcal{F}$ *a au moins une solution).*

◇

THÉORÈME 11.5.1. *Tout* I*-problème équationnel en forme résolue avec contraintes et différent de* $\bot$ *a au moins une solution.*

PREUVE. Soit $\mathcal{P}$ un problème en forme résolue avec contraintes. Si $\mathcal{P} = \top$ ou $= \bot$, le théorème est trivialement valide. Supposons que $\mathcal{P}$ est de la forme :

$$\mathcal{P} = \bigwedge_{i=1}^{n} x_i = t_i \wedge \bigwedge_{i=1}^{m} y_i \neq s_j \wedge \mathcal{F}$$

où les variables x_i apparaissent au plus une fois dans $\mathcal{P}$.

$\mathcal{F}$ est satisfaisable, donc a au moins une solution σ. Soit $\mathcal{P}' = \sigma(\mathcal{P})$. $\mathcal{P}'$ est en forme résolue avec contraintes (au sens de (COMON ET LESCANNE, 1989)) et n'est pas égal à $\bot$. Par conséquent $\mathcal{P}'$ a au moins une solution ρ. $\rho\sigma$ est une solution de $\mathcal{P}$. C.Q.F.D.

11.5.2 Résolution des problèmes équationnels

Il s'agit de donner un algorithme transformant tout I-problème équationnel en une disjonction de problèmes équationnels sous forme résolue avec contraintes. Nous présentons dans cette section les règles de transformation utilisées pour simplifier le problème et nous prouvons leurs propriétés (correction, terminaison et complétude).

Règles de transformation

En premier lieu, nous utilisons le système R_{unif} pour simplifier les équations et diséquations de la forme $s = t$ apparaissant dans le problème. Une première possibilité consiste à remplacer de telles équations par leur forme normale par rapport à R_{unif}. Cependant, en procédant ainsi, on détruit la forme particulière du préfixe des problèmes équationnels (c'est-à-dire l'ordre d'imbrication des quantificateurs). En effet, les règles d'unification introduisent de nouvelles variables auxiliaires dans la formule. Celles-ci servent à "déplier" un N-terme $t^n.u$ suivant la valeur de son exposant n (voir section 11.4). Par exemple, la formule : $\forall n.f(x) = f(\diamond)^n.a$ est remplacée par : $\forall n.(n = 1 \wedge f(x) = f(a)) \vee \exists m.n = m + 1 \wedge x = f(\diamond)^m.a$, qui n'est pas un I-problème équationnel. Transférer le quantificateur existentiel $\exists m$ à la racine de la formule est incorrect puisque m et n apparaissent dans la même sous-formule atomique $n = m + 1$.

Pour éviter de modifier la forme particulière du préfixe des problèmes considérés, une autre possibilité consiste à introduire des quantificateurs universels à la place des quantificateurs existentiels. Plus précisément, la formule $f(x) = f(\diamond)^n.a$ sera remplacée par : $n \neq m+1 \vee x = f(\diamond)^m.a$ où m est un nouveau paramètre. Donc nous obtenons le problème équationnel :

$$\forall n, m.n \neq m + 1 \vee x = f(\diamond)^m.a.$$

La **règle d'unification** est par conséquent définie comme suit.

Unification Soit $s = t$ une équation non irréductible par rapport à R_{unif} et soit $\exists \overline{m}. \bigvee_{i=1}^{n}(\phi_i \wedge \psi_i)$ une forme normale de $s = t$ par rapport à R_{unif} (ϕ_i est une conjonction d'équations de la forme : $s = t$ où s, t sont des I-termes, et ψ_i est une formule arithmétique). $\overline{m}$ sont les nouvelles variables entières introduites par le processus d'unification.

$$\mathcal{P}[s = t]_p \rightarrow \exists \overline{m}.\mathcal{P}[\bigvee_{i=1}^{n}(\phi_i \wedge \psi_i)]_p$$

si $s = t$ ne contient aucun paramètre.

$$s = t \rightarrow \forall \overline{m}. \bigwedge_{i=1}^{n}(\neg\psi_i \vee \phi_i)$$

si $s = t$ contient au moins un paramètre.

LEMME 11.5.1. *La règle* ***unification*** *est correcte.*

PREUVE. La correction de la première sous-règle est immédiate. L'algorithme d'unification remplace une formule atomique $s = t$ par une formule de la forme :

$$\exists \overline{m}. \bigvee_{i=1}^{n} \phi_i \wedge \psi_i$$

où ϕ_i est soit $\bot$, soit une conjonction (éventuellement vide) d'équations de la forme $t_j = s_j$, et où t_i et s_j sont des I-termes et où ψ_i est une formule arithmétique.

De plus, nous avons : $\exists \overline{m}. \bigvee_{i=1}^{n} \psi_i \equiv \top$ et $\psi_i \wedge \psi_j \equiv \bot$ (si $i \neq j$).

Ces deux propositions sont des propriétés immédiates de l'algorithme d'unification, qui peuvent être vérifiées aisément pour chacune des règles. Intuitivement, elles expriment simplement le fait que la règle divise le "domaine" D d'une formule (c'est-à-dire le domaine des variables

entières d'une formule) en deux sous-domaines D_1 et D_2 tels que : $D_1 \cup D_2 = D$ et $D_1 \cap D_2 = \emptyset$. Supposons que $\sigma \in \mathcal{S}(\forall \overline{m}. \bigwedge_{i=1}^{n}(\neg\psi_i \vee \phi_i))$. On a $\exists \overline{m}. \bigvee_{i=1}^{n} \psi_i \equiv \top$, donc $\bigvee_{i=1}^{n} \psi_i\sigma \equiv \top$. Il existe $i \leq n$ et une substitution θ de $\overline{m}$ telle que $\psi_i\sigma\theta$ est vraie. Donc $\phi_i\sigma\theta \equiv \top$ et, par correction de l'algorithme d'unification, $(s = t)\sigma$ est valide.

Réciproquement, supposons que $s\sigma = t\sigma$. Alors, il existe $i \leq n$ et une substitution θ de $\overline{m}_i$ telle que $\phi_i\sigma\theta$ et $\psi_i\sigma\theta$ sont valides. Donc on a, pour tout $j \neq i$, $\forall \overline{m}_j.\psi_j \equiv \bot$. D'où $\sigma \in \mathcal{S}(\forall \overline{m}. \bigwedge_{i=1}^{n}(\neg\psi_i \vee \phi_i))$. C.Q.F.D.

Nous utilisons une règle analogue pour la négation.

Dis–unification

$$\mathcal{P}[s \neq t]_p \rightarrow \exists \overline{m}.\mathcal{P}[\bigvee_{i=1}^{n}(\psi_i \wedge \neg\phi_i)]_p$$

si $s = t$ ne contient pas de paramètre.

$$s \neq t \rightarrow \forall \overline{m}. \bigwedge_{i=1}^{n}(\neg\phi_i \vee \neg\psi_i)$$

si $s = t$ contient au moins un paramètre.
La correction de la règle de disunification est immédiate.

Remplacement, fusion et explosion des disjonctions

Ensuite, on utilise (comme pour les formules classiques) les règles de **Replacement**, **Merging**, et **Explosion of disjunctions** (voir (Comon et Lescanne, 1989)).

Replacement $x = t \wedge \mathcal{P} \rightarrow x = t \wedge \mathcal{P}\{x \rightarrow t\}$
Merging $x = t \wedge \mathcal{P}[x = s]_p \rightarrow \mathcal{P}[t = s]_p$

Explosion of disjunctions $\forall \overline{y} : P \wedge (P_1 \vee P_2) \rightarrow \forall \overline{y} : P \wedge P_1$
si $\mathcal{V}ar(P_1) \cap \overline{y} = \emptyset$, et $\mathcal{V}ar(P_2) \cap \overline{y} = \emptyset$.

Afin d'assurer la terminaison de l'algorithme, on impose des conditions supplémentaires sur certaines des règles.

- **Replacement** est appliquée seulement si x apparaît au moins deux fois dans le problème, t ne contient aucun paramètre, et si x n'apparaît pas dans un terme u contenant un paramètre.
- Comme nous le verrons par la suite, **Merging** est appliquée seulement après l'application d'une règle **Explosion** ou N**-Explosion**.

Elimination des paramètres

Les règles suivantes sont utilisées afin d'éliminer les paramètres des problèmes équationnels :

Elimination of parameters (EP) La règle **Elimination of parameters** élimine les quantificateurs inutiles.
$\forall \overline{y}, y' : P \rightarrow \forall \overline{y} : P$ if $y' \notin \mathcal{V}ar(P)$

Universality of parameters (UP) Les règles **Universality of parameters** élimine les paramètres x apparaissant dans des équations ou des diséquations de la forme $x \neq t$.

1 $\quad \forall \overline{y} : P \wedge y' \neq t \rightarrow \bot \qquad$ si $y' \in \overline{y}$ et $y' \notin \mathcal{V}ar(t)$

2 $\quad \forall \overline{y} : P \wedge (y' \neq t \vee R) \rightarrow \forall \overline{y} : P \wedge R\{y' \rightarrow t\}$ si $y' \in \overline{y}$ et $y' \notin \mathcal{V}ar(t)$

3 $\quad \forall \overline{y} : P \wedge z = t \rightarrow \bot$

si Σ contient au moins deux symboles,
z est différente de t,
$\mathcal{V}ar(z = t)$ contient au moins un paramètre.

4 $\forall \overline{y} : P \wedge (R \bigvee_{i=1}^{n} z_i = u_i) \rightarrow \forall \overline{y} : P \wedge R$

si tout z_i est une variable différente de u_i,
$\mathcal{V}ar(z_i = u_i)$ contient au moins un paramètre,
R ne contient aucun paramètre.

La correction de ces règles est établie dans (Comon, 1988).

Il est facile de voir que la règle d'**Explosion**, utilisée dans (Comon, 1988; Comon et Lescanne, 1989) pour éliminer les paramètres apparaissant dans une diséquation (en "faisant remonter" le paramètre à la position racine) ne termine plus dans ce cas. Considérons par exemple la formule :

$$\forall y, n.x \neq f^n(\diamond).y$$

Si nous appliquons la règle d'explosion, on obtient :

$$\exists w.\forall y, n.x \neq f^n(\diamond).y \vee x = f(w)$$

i.e.

$$\exists w.\forall y, n.\forall m.n \neq m + 1 \vee w \neq f^m(\diamond).y \vee x = f(w)$$

Nous obtenons un problème qui contient le problème initial et par conséquent la procédure ne s'arrête pas.

Pour assurer la terminaison de la procédure, il faut introduire de nouvelles conditions d'application sur la règle **Explosion**.

Explosion :

(E) $\forall \overline{y} : P \rightarrow \exists \overline{w}, \forall \overline{y} : P \wedge z = f(w_1, \ldots, w_p)$
où z est une inconnue, $\overline{w}$ sont de nouvelles variables et $f \in \Sigma$.
S'il existe dans $P, x = u$ (ou $x \neq u$) où u n'est ni une variable ni un N-terme et contient au moins un paramètre,
UP, **Merging**, **EP**, R_{unif} ne s'appliquent pas.

Ensuite, nous définissons une nouvelle règle appelée N-**Explosion** afin d'éliminer les paramètres apparaissant dans un N-terme. L'idée de cette règle est illustrée par l'exemple suivant.

Exemple 11.5.1. Soit $\mathcal{P}$ le problème :

$$\forall n.x \neq f^n(\diamond).a$$

sur la signature : $\Sigma = \{f, a\}$ Remarquons que soit $x = a$, soit il existe un entier k et un terme z tels que : $x = f^k(\diamond).z$. En prenant le maximum de tous les entiers possibles, nous obtenons un entier tel que $x = a$ ou $\exists k, z.x = f^k(\diamond).z \wedge \forall u.z \neq f(u)$.

L'idée de la règle N-**Explosion** est d'ajouter ces deux formules à $\mathcal{P}$. Nous obtenons les formules :

$$\mathcal{P}_1 \ : \exists k.\forall n, u.x \neq f^n(\diamond).a \wedge x = f^k(\diamond).z \wedge z \neq f(u)$$

$$\mathcal{P}_2 \ : \forall n.x = a \wedge x \neq f^n(\diamond).a$$

$\mathcal{P}_2$ est réduit à $x = a$ (par les règles **Merging** et **Unification**).
On utilise ensuite la règle d'explosion pour éliminer le paramètre u dans $\mathcal{P}_1$. Nous obtenons :

$$\exists k, z.\forall n.x \neq f^n(\diamond).a \wedge x = f^k(\diamond).z \wedge z = a$$

En appliquant la règle de **Merging**, ce problème peut être transformé en :

$$\exists k.\forall n.f^k(\diamond).a \neq f^n(\diamond).a \wedge x = f^k(\diamond).a$$

La diséquation $f^k(\diamond).a \neq f^n(\diamond).a$ est réduite par la procédure d'unification à $k \neq n$. Donc, le problème est transformé en :

$$\exists k.\forall n.(k \neq n \wedge x = f^k(\diamond).a)$$

En appliquant la règle **Elimination de paramètres**, on réduit la formule à $\bot$. Donc les solutions du problème initial sont $x = a$. $\diamond$

Plus généralement, le principe de la nouvelle règle que nous proposons est le suivant. Si le problème équationnel contient une équation $x \neq t^n.u$ où $t^n.u$ contient un paramètre, alors nous introduisons une nouvelle variable entière k et nous ajoutons la formule $x = t^k.z \wedge \forall u.z \neq t.u$. Remarquons que cela est possible seulement si t ne contient aucun paramètre. Si t contient un paramètre, nous remplaçons celui-ci par une nouvelle variable (similairement, nous remplaçons tout N-terme par une variable).

N-Explosion

$$\begin{array}{ll} \mathcal{P} \rightarrow \exists k, y, \overline{x}.\mathcal{P} \wedge x = \rho(t)^k.y \wedge (\forall w.y \neq \rho(t)(w)) & (\mathcal{P}_1) \\ \quad \rightarrow \forall w, \overline{x}.\mathcal{P} \wedge x \neq \rho(t)(w) & (\mathcal{P}_2) \end{array}$$

où

- $\mathcal{P}$ contient une diséquation $x \neq t^n.u$, telle que $t^n.u$ contient au moins un paramètre.
- $\rho(t)$ est un terme obtenu en remplaçant chaque paramètre et chaque N-terme dans t par de nouvelles variables $\overline{x}$.

REMARQUE. Les formules obtenues par la règle N-Explosion ne sont pas nécessairement des I-problèmes. Cependant, elles peuvent être facilement transformer en I-problèmes par transformation en forme prénexe.

LEMME 11.5.2. *[Correction de N-Explosion] Soit $\mathcal{P}_1, \mathcal{P}_2$ deux I-problèmes équationnels déduits de $\mathcal{P}$ par la règle N-**Explosion**. Alors*

$$\mathcal{P} \equiv (\mathcal{P}_1 \vee \mathcal{P}_2).$$

PREUVE. Si $\forall \overline{x}, w.x \neq \rho(t)(w)$ est vrai, alors $\mathcal{P} \equiv \mathcal{P}_2$. Supposons que $\forall \overline{x}, w.x \neq \rho(t)(w)$ n'est pas vraie. Alors, il existe $\overline{x}$ et y tel que $x = \rho(t)(y)$. Soit k le plus grand entier tel qu'il existe y tel que $x = \rho(t)^k.y$. Alors, si $x = \rho(t)^k.y$, nous avons $\forall w.y \neq \rho(t)(w)$, donc $\mathcal{P}$ est équivalent à $\mathcal{P} \wedge \exists k.(x = \rho(t)^k.y \wedge (\forall w.y \neq \rho(t)(w)))$. Par conséquent $\mathcal{P} \equiv \mathcal{P}_1$. C.Q.F.D.

REMARQUE. Toute application de la règle N-**Explosion** sera immédiatement suivie de l'application d'une règle **Merging** entre les littéraux $x = \rho(t)^k.y$ et $x \neq t^n.u$.

On note R_{simp} le système composé des règles **Unification, Dis-Unification, Replacement, Explosion of Disjunctions**. Nous définissons une règle de simplification supplémentaire.

Simplification

$$\mathcal{P} \wedge \mathcal{Q} \rightarrow \mathcal{P}$$

Si $\mathcal{P} \wedge \neg\mathcal{Q} \rightarrow_{R_{\text{simp}}} \bot$.

La correction de la règle de **Simplification** est immédiate. Cette règle sera utile par la suite dans la preuve de terminaison. Elle permet en outre d'éliminer certains sous-problèmes redondants et ainsi d'améliorer dans certains cas l'efficacité de l'algorithme de résolution.

REMARQUE. – La terminaison de R_{simp} sera prouvée par la suite.

– Evaluer la condition d'application de la règle R_{simp} peut se révéler coûteux en pratique. Cependant, nous verrons lors de la preuve de terminaison qu'elle n'a pas besoin d'être systématiquement appliquée.

Elimination des paramètres arithmétiques

Si un problème $\mathcal{P}$ est irréductible par **Unification**, **Dis–unification**, **EP**, **Merging**, **Explosion,** N**-Explosion**, alors les seuls paramètres du problème n'apparaissent dans aucune formule non-arithmétique. Il est alors possible d'utiliser les techniques d'élimination des quantificateurs classiques afin d'éliminer ces paramètres, par exemple la procédure décrite dans (PRESBURGER, 1929) ou celle (plus efficace) de (COOPER, 1972). Le processus d'élimination des paramètres n'est pas rappelé ici.

Terminaison et complétude

Soit R_{solve} le système complet. R_{solve} = {**Unification, Dis-Unification, Replacement, Explosion of Disjunctions, Elimination of Parameters, Universality of Parameters, Explosion, Simplification,** N**-Explosion**}.

THÉORÈME 11.5.2. *Le système R_{solve} est à terminaison finie.*

PREUVE. Introduisons tout d'abord une définition.

DÉFINITION 11.5.3. *Comme pour les problèmes d'unification, une variable x est dite* résolue *si et seulement si elle n'apparaît que dans une équation $x = t$, où $x \notin \mathcal{V}ar(t)$.* ◇

Nous introduisons la mesure suivante sur les problèmes I-équationnels.

DÉFINITION 11.5.4. *Soit f une formule de la forme $t_1 = t_2$ ou $t_1 \neq t_2$. $\mathcal{I}(f)$ est définie comme suit.*

– $I(f) = 0$, *si f est arithmétique.*

– $I(f) = \{|\mathcal{P}os(t_1)|, |\mathcal{P}os(t_2)|\}$, *sinon.*

◇

DÉFINITION 11.5.5. *Soit $\mathcal{P}$ un problème I-équationnel en fnc. On a $\mathcal{P} = \bigwedge_{i=1}^{n} d_i$ où d_i est une disjonction d'équations et de diséquations. Sans perte de généralité, nous supposons que le problème est irréductible par R_{unif} et $R_{disunif}$.*

Soit l'interprétation suivante sur les problèmes I-équationnels.

$$\phi(\mathcal{P}) = \{(param(d_i), \phi_1(d_i'), nvar(\mathcal{P}), \phi_1(d_i))/1 \leq i \leq n\}$$

avec

– *$param(d_i)$ est le nombre de paramètres non arithmétiques de d_i.*

– *nvar*($\mathcal{P}$) *est le nombre de variables non résolues de* $\mathcal{P}$.

– d'_i *est la disjonction des littéraux de* d_i *qui contiennent au moins un paramètre.*

– $\phi_1(\bigvee_{i=1}^{k} f_i) = \{I(f_i)/1 \leq i \leq k\}$.

ϕ_1 *est ordonné par l'extension multiensemble de l'ordre naturel sur* I. ϕ *est ordonné par l'extension lexicographique des ordres habituels.* ◇

Considérons en premier lieu le sous-ensemble contenant les règles R'_{solve} = {**Unification, Dis-Unification, Replacement, Explosion of Disjunctions, Elimination of Parameters, Universality of Parameters, Explosion, Simplification**}.

LEMME 11.5.3. *Pour toute règle* $\rho \in R'_{solve}$ *qui n'est pas une règle d'unification et de disunification et pour tout couple de problèmes* $(\mathcal{P}, \mathcal{P}')$ *tel que* $\mathcal{P} \rightarrow_\rho \mathcal{P}'$, *on a* $\phi(\mathcal{P}) > \phi(\mathcal{P}')$.

PREUVE. – **Replacement** : à cause du contrôle, *param* et $\phi_1(d'_i)$ n'augmentent pas. D'autre part, *nvar* décroît strictement.

– **Explosion de Disjunction** : $\phi_1(d'_i)$, *param*, *nvar* n'augmentent pas. De plus $\{\phi_1(d_i)/1 \leq i \leq n\}$ diminue strictement (puisque la règle supprime un des éléments d'un des d_i).

– **Universality of Parameters 2 :** *param* décroît strictement.

– **Universality of Parameters 1,3,4 :** immédiat.

– **Explosion :** à cause du contrôle, la règle est immédiatement suivie d'une application de la règle **Merging**. Une équation $x \neq t$ (où t contient au moins un paramètre) est transformée en $f(x_1, \ldots, x_n) \neq t$. Puisque t n'est pas un N-terme, cette équation est réduite par décomposition soit en $\top$, soit en $x_1 \neq t_1 \vee \ldots \vee x_n \neq t_n$. Si la règle **Universality of parameters** est appliquée, alors la décroissance est immédiate, sinon I décroît strictement.

C.Q.F.D.

Puisque ϕ est bien-fondé et puisque que les règles d'unification se terminent, on en déduit que R'_{solve} se termine (cela implique en particulier que R_{simp} se termine). Il reste simplement à considérer la règle N-**Explosion**. Pour cela, nous introduisons quelques définitions supplémentaires.

DÉFINITION 11.5.6. *L'ensemble des* N*-paramètres est le plus petit ensemble de paramètres* x *non-arithmétiques tel que* x *apparaît dans une formule atomique contenant un* N*-terme ou un* N*-paramètre.* ◇

DÉFINITION 11.5.7. *Un* N*-terme* $t^n.u$ *apparaissant dans un littéral* f *d'un problème*

$$\exists \overline{x}.\forall \overline{y}.\mathcal{P} \wedge (\mathcal{R} \vee f)$$

est dit régulier *si et seulement si pour tout terme* x, $\mathcal{P} \wedge \neg\mathcal{R} \wedge t(x) = u \rightarrow_{R_{simp}} \bot$.

Pour tout littéral f, *on note reg*(f) *le nombre de* N*-termes réguliers de* f. ◇

La définition suivante introduit une nouvelle mesure sur les problèmes I-équationnels.

DÉFINITION 11.5.8.

$$\phi'(\mathcal{P}) = \{(N\text{-}param(d_i), \phi_2(d''_i), param(d_i), \phi_1(d'_i), nvar(\mathcal{P}), \phi_1(d_i))/1 \leq i \leq n\}$$

avec

– d''_i *est la disjonction des littéraux de* d_i *contenant au moins un* N*-paramètre.*

– $\phi_2(d) = \{(n(f), reg(f))/f \in d\}$ *(*n *est défini comme dans la preuve de théorème 11.4.1).*

– *N-param(d) est le nombre de N-paramètres de d.*

◇

LEMME 11.5.4. *Les règles d'unification et de disunification n'augmentent pas n et reg.*

PREUVE. Il suffit de le vérifier pour chaque règle. C.Q.F.D.

LEMME 11.5.5. *Soit $\mathcal{P}$ et $\mathcal{P}'$ deux problèmes tels que $\mathcal{P} \rightarrow_{R'_{solve}} \mathcal{P}'$. Alors $\phi'(\mathcal{P}) > \phi'(\mathcal{P}')$.*

PREUVE. Il suffit de vérifier que *N-param* et ϕ_2 n'augmentent pas. Nous considérons uniquement la règle **Universality of Parameter** (la preuve pour les autres règles est immédiate).

Deux cas doivent être distingués. Soit y' est un N-paramètre, auquel cas *N-param* décroît strictement, soit y' n'est pas un N-paramètre d'où par définition, *N-param* et $\phi(d''_i)$ n'augmentent pas. C.Q.F.D.

LEMME 11.5.6. *Soit $\mathcal{P}, \mathcal{P}'$ deux I-problèmes tels que $\mathcal{P} \rightarrow_{R_{solve}} \mathcal{P}'$. Alors $\phi'(\mathcal{P}) > \phi'(\mathcal{P}')$.*

PREUVE. Il suffit de considérer la règle N**-Explosion**. Elle introduit de nouveaux paramètres. Mais ces paramètres ne sont pas des N-paramètres donc *N-param* reste constant. Comme pour **Explosion**, N**-Explosion** est toujours suivi de l'application de la règle **Merging**. Une disé-quation $x \neq t^n.u$ est ainsi transformée en $\rho(t)^m.y \neq t^n.u$. Ensuite, par application de la règle **Decompose 2**, on obtient :

$$\begin{array}{l}\forall k_1, k_2, k \\ (n \neq 1 \vee t(u) \neq \rho(t)^m.y) \wedge \\ (n \neq k+1 \vee ((t(a) \neq \rho(t)(a)) \vee \\ ((n \neq m) \vee (u \neq y) \\ \wedge((n \neq m + k_1) \vee y \neq t^{k_1}.u) \\ \wedge((m \neq n + k_2) \vee \rho(t)^{k_2}.y \neq u))))\end{array}$$

– On a $n(u \neq y) < n(x \neq t^n.u)$ donc ϕ_2 décroît strictement.

– Par définition de ρ, la formule $t(a) \neq \rho(t)(a)$ est réduite par **Decompose** en deux formules de la forme $x' \neq t_{|q}$. On a $n(t(a) \neq \rho(t)(a)) < n(x \neq t^n.u)$.

– $t(u) \neq \rho(t)^m(y)$: n n'augmente pas. Cette formule est réduite par **disunification** soit en $t_{|q} \neq \rho(t)_{|q}$ (dans ce cas n décroît strictement), soit en $u \neq \rho(t)^{m'}(y)$. Ou $t^n.u$ est régulier et $\rho(t)^{m'}.y \neq u$ peut être réduit à $\top$ (par **Simplification**), ou (à cause de la présence de la formule $\forall w.y \neq \rho(t)(w)$), $\rho(t)^{m'}.y$ devient régulier et *reg* décroît strictement.

Par conséquent, il ne reste plus qu'à considérer les formules $y \neq t^{k_1}.u$ et $\rho(t)^{k_2}.y \neq u$. Il est facile de voir qu'à cause de la présence des formules $\forall w.y \neq \rho(t)(w)$ et $t(a) \neq \rho(t)(a)$, $y \neq t^{k_1}.u$, est réduit à $\top$ en utilisant les règles **Explosion**, **Universality of Parameters** et **Simplification**. En effet, on a $\forall w.y \neq \rho(t)(w)$ donc $\forall w.y \neq t(w)$.

Deux cas sont à distinguer.

1. t contient au moins un N-terme. Alors $n(\rho(t)^{k_2}.y \neq u) < n(x \neq t^n.u)$.
2. t ne contient aucun N-terme. Alors $\rho(t)^{k_2}.y \neq u$ ne contient qu'un seul N-terme. De plus, soit $t^n.u$ est régulier et $\rho(t)^{k_2}.y \neq u$ peut être réduit à $\top$ par **Universality of Parameters** et **Simplification**, soit (à cause de la formule $\forall w.y \neq \rho(t)(w)$), *reg* diminue strictement.

Par le lemme 11.5.4, on en déduit que ϕ_2 diminue strictement, ce qui prouve que R_{solve} se termine. C.Q.F.D.

DÉFINITION 11.5.9.

$$\psi'(d_i) = (N\text{-}param(d_i), \{(I(s \neq t), reg(s \neq t))/s \neq t \in d'_i\}, param(d_i))$$

avec :

- d'_i *est l'ensemble des diséquations de* d_i *qui contiennent au moins un paramètre.*
- N*-param*(d) *est le nombre de* N*-paramètres apparaissant dans* d*.*
- *param*(d) *est le nombre de paramètres non-arithmétiques dans* d*.*
- *reg*$(s \neq t) = 1$ *si* $s = t$ *contient au moins un* N*-terme de la forme* $t^n.u$ *où* $\exists x.\mathcal{F} \wedge u \neq \rho(t)(x)$ [3] *ne peut être réduit à* $\top$ *par le système* R'_{solve}*,* 0 *sinon.*

◇

Nous allons montrer que ψ' décroît strictement.

- **Universality of parameter** : Soit N*-param* décroît strictement, soit *reg* et I n'augmentent pas et *param* décroît strictement.
- **Explosion** : à cause du contrôle, toute application de cette règle est suivie par une application de la règle de **Merging**. Une équation $x \neq t$ (où t contient au moins un paramètre) est transformée en une équation $f(x_1, \ldots, x_n) \neq t$. Puisque t n'est pas un N-terme, cette équation est réduite par décomposition à soit $\top$ soit $x_1 \neq t_1 \vee \ldots \vee x_n \neq t_n$. Si la règle **Universality of Parameters** est appliquée pendant le processus de normalisation, la décroissance est évidente. Sinon I décroît strictement.
- N**-Explosion** : cette règle introduit de nouveaux paramètres. Mais ces paramètres ne sont pas des N-paramètres, donc N*-param* n'augmente pas.
  Comme pour **Explosion**, la règle N**-Explosion** est toujours suivie d'une application de la règle **Merging**. Une diséquation $x \neq t^n.u$ est transformé en $\rho(t)^m.y \neq t^n.u$. Alors, en appliquant la règle **Decompose 2** sur $\rho(t)^m.y = t^n.u$, on obtient une formule de la forme (où k, k_1, k_2 sont de nouvelles variables entières) :

$$\begin{array}{l}(m \neq 1 \vee t^n.u \neq \rho(t)(y)) \wedge \\ (n \neq k+1 \vee ((t(a) \neq \rho(t)(a)) \vee \\ ((n \neq m) \vee (u \neq y) \\ \wedge((n \neq m + k_1) \vee y \neq t^{k_1}.u) \\ \wedge((m \neq n + k_2) \vee \rho(t)^{k_2}.y \neq u))))\end{array}$$

  - Nous avons $n(u \neq y) < n(x \neq t^n.u)$ donc $\phi_2(u \neq y) < \phi_2(x \neq t^n.u)$.
  - La formule $t(a) \neq \rho(t)(a)$ est réduite par **Decompose** à une formule de la forme $x' \neq t_{|q}$. Nous avons $n(t(a) \neq \rho(t)(a)) < n(x \neq t^n.u)$. Donc $\phi_2(t(a) \neq \rho(t)(a)) < \phi_2(x \neq t^n.u)$.
  - $\rho(t)(y) \neq t^n.u$ est transformée par la règle **Unification** en une formule de la forme $t_{|q} \neq x'$ (dans ce cas n décroît strictement) ou $t^n.u \neq y$. Cette dernière formule peut être réduite à $\top$ par R'_{solve}.

3. ρ est la fonction définie dans la règle N-Explosion

Donc, il reste uniquement à considérer les formules $y \neq t^{k_1}.u$ et $\rho(t)^{k_2}.y \neq u$.
En premier lieu, il est très facile de montrer que, à cause de la formule $\forall w.y \neq \rho(t)(w)$, et $t(a) \neq \rho(t)(a)$, $y \neq t^{k_1}.u$, est réduit à $\top$ en utilisant les règles **Explosion**, **Decomposition** et **Merging**. En effet, on a $\forall w.y \neq \rho(t)(w)$ donc $\forall w.y \neq t(w)$.
Nous devons alors distinguer les cas suivants :

1. u contient un N-terme. Alors $I(\rho(t)^{k_2}.y \neq u) < I(x \neq t^n.u)$.
2. u ne contient aucun N-terme et t contient au moins un N-terme. Alors $I(\rho(t)^{k_2}.y \neq u) < I(x \neq t^n.u)$.
3. u et t ne contiennent aucun N-terme. Alors $\rho(t)^{k_2}.y \neq u$ contient seulement un N-terme. De plus, soit $\rho(t)^{k_2}.y \neq u$ peut être réduit à $\bot$, soit (puisque $\forall w.y \neq \rho(t)(w)$), $reg(\rho(t)^{k_2}.y \neq u) = 0$.

Ceci prouve que R_{solve} se termine. C.Q.F.D.

THÉORÈME 11.5.3. *[complétude de R_{solve}]*
Tout I-problème équationnel $\mathcal{P}$ irréductible par rapport à R_{solve} est de la forme $\mathcal{P}' \wedge \mathcal{F}$ où $\mathcal{P}'$ est sous forme résolue avec contraintes, et $\mathcal{F}$ est une formule arithmétique.

PREUVE. Soit $\mathcal{P}$ un I-problème équationnel, irréductible par rapport à R_{solve}.
Toute formule atomique non-arithmétique apparaissant dans $\mathcal{P}$ doit être de la forme $x = t$ ou $x \neq t$ (sinon une des règles d'unification ou de disunification s'applique).
Supposons que $\mathcal{P}$ contient un paramètre. Si x apparaît dans une diséquation de la forme $y \neq t$ alors la règle **Explosion ou 2** s'applique. Si x apparaît dans une autre formule atomique non-arithmétique, l'une des règles **Universality de Parameters** s'applique. Si x apparaît seulement dans une formule arithmétique, alors x peut être éliminée par la méthode classique d'élimination des quantificateurs.
Par irréductibilité par rapport à R, et en utilisant la complétude de R, on en déduit que $\mathcal{P}$ doit être de la forme $\mathcal{P}' \wedge \mathcal{F}$, où $\mathcal{P}'$ est en forme résolue avec contraintes, et $\mathcal{F}$ est une formule arithmétique.

C.Q.F.D.

Afin d'obtenir une forme résolue avec contraintes, il suffit de vérifier si $\mathcal{F}$ possède une solution (i.e. que la fermeture existentielle de $\mathcal{F}$: $\exists \mathcal{V}ar(\mathcal{F}).\mathcal{F}$ est vraie). Cela peut être réalisé par n'importe quel algorithme de résolution des formules arithmétiques (PRESBURGER, 1929; COOPER, 1972). Nous en déduisons le théorème suivant.

THÉORÈME 11.5.4. *La théorie du premier ordre dans le langage des I-termes est décidable.*

PREUVE. C'est une conséquence immédiate des théorèmes 11.5.3 et 11.5.2 (voir (COMON, 1988)). C.Q.F.D.

Discussion

Ce résultat de décidabilité peut sembler surprenant si on considère que les théories du premier ordre sont en général indécidables dès que la validité de la formule dépend de propriétés "profondes" sur les termes. Par exemple, la théorie de la relation d'inclusion (VENKATARAMAN, 1987), de l'ordre récursif sur les chemins (TREINEN, 1992; COMON ET TREINEN, 1997), la subsomption pour les arbres rationels (DÖRRE ET ROUNDS, 1992) ont été prouvées indécidables. Ici la condition $\exists n.x = t^n.y$ impose une structure régulière sur x. Le point-clef est que les variables appartenant à t *ne peuvent pas* être instanciées avec des valeurs différentes à chaque étape. Si nous autorisons l'occurence de termes de la forme $f(\_, \diamond)^n$, où $\_$ dénote une variable anonyme qui peut être instanciée différemment à chaque étape, la théorie devient indécidable (HERMANN, 1994).

11.6 Utilité des I-termes en construction de modèles

Nous allons maintenant utiliser les résultats précédents afin d'inclure le formalisme des I-termes dans notre méthode de construction de modèles.

DÉFINITION 11.6.1. *Un sous-ensemble E de $\tau^n(\Sigma)$ est appelé un I-ensemble si et seulement si il existe une I-formule équationnelle $\mathfrak{F}_I(E)$ contenant n variables libres $x_1, \ldots, x_n$ vérifiant*

$$(x_1, \ldots, x_n)\sigma \in E - \ \sigma \in \mathcal{S}(\mathfrak{F}_I(E)).$$

Une interprétation partielle est appelée une I-eq-interprétation si et seulement si pour tout $P \in \Omega$, les ensembles $\mathcal{I}(P)^+$ et $\mathcal{I}(P)^-$ sont des I-ensembles. ◇

EXEMPLE 11.6.1. L'interprétation $\mathcal{J} = \{P(f^{2\times n}(a))\}$ est une I-eq-interprétation. On a $\mathfrak{F}_I(\mathcal{J}(P)^+) = \exists n.m = n \times 2 \wedge x_1 = f(\diamond)^m.a$. ◇

Puisque nous disposons d'un algorithme pour résoudre les problèmes équationnels sur les I-termes, les résultats du chapitre 5 ainsi que les règles de (CAFERRA ET ZABEL, 1992) peuvent être généralisés de façon immédiate aux I-c-clauses. En particulier, les résultats de correction et de complétude des règles sont préservés. En outre :

THÉORÈME 11.6.1. *Le problème de l'évaluation d'une formule $\mathcal{F}$ dans une I-eq-interprétation totale est décidable.*

L'objet des sections suivantes est d'étendre la méthode afin de construire des I-modèles. Il s'agit de définir des règles pour introduire *automatiquement* des I-termes dans l'ensemble de I-c-clauses.

11.6.1 Une nouvelle règle

Plus précisément, on cherche à simuler par une règle unique n applications de la c-résolution sur une I-c-clause autorésolvente. Par exemple, à partir de la I-c-clause $[\![P(x) \vee \neg P(f(f(x))) \ : \top]\!]$, nous voulons déduire $[\![P(x) \vee \neg P(f(f(\diamond))^n.x) \ : \top]\!]$. C'est l'objet de la règle proposée dans cette section. Elle est similaire, mais plus générale que la règle proposée par (SALZER, 1992) pour des clauses standards (voir section 11.6.2).

Le principe de la règle est le suivant. Soit $C : [\![l(\overline{t}) \vee \neg l(\overline{t'}) \vee R \ : \mathcal{X}]\!]$ une I-c-clause autorésolvente. Soit $x_1, \ldots, x_n$ les variables de C. Soit σ un renommage des variables de C. Chaque variable x_i est associée à une variable $x'_i = \sigma(x_i)$. Considérons l'équation $\sigma(\overline{t}) = \overline{t'}$. La forme normale des équations par rapport à R_{unif} est une disjonction de la forme $\bigvee_{i=1}^{m} F_i$, où les F_i sont des formules de la forme

$$F_i : \exists \overline{n}.n_1 = E_1 \wedge \ldots \wedge n_k = E_k \wedge x_1 = t_1 \wedge \ldots \wedge x_n = t_n$$

Nous dirons qu'une équation non arithmétique est *récursive* si elle est de la forme $s = t$, et s'il existe une variable x telle que $s \geq x$ et $t \geq \sigma(x)$, et s ou t n'est pas une variable. Les variables x et $\sigma(x)$ sont dites *récursives*.

F_i peut s'écrire

$$\exists \overline{n}.F'_i \wedge F''_i.$$

où F'_i contient seulement des équations récursives, et F''_i ne contient aucune équation récursive.

Dans la suite, les conditions suivantes sont supposées valides.

1. R, F''_i, $\mathcal{X}$ ne contient aucune variable récursive.
2. chaque équation récursive contient au plus deux variables récursives. Remarquons que cette condition n'est pas toujours satisfaite réalisé: on peut par exemple avoir $x = f(\sigma(x))$ et $y = g(\sigma(y), \sigma(x))$. Ici x, y, $\sigma(y)$ et $\sigma(x)$ sont récursives, et $g(y, \sigma(x))$ contient 3 variables récursives.

REMARQUE. Nous donnerons par la suite des exemples montrant l'utilité de ces deux conditions.

Ces conditions permettent d'exprimer la forme générale des résolvents des I-c-clauses C.

LEMME 11.6.1. *Soit* $C : [\![l(\overline{t}) \vee \neg l(\overline{t'}) \vee R \ : \mathcal{X}]\!]$ *une* I*-c-clause auto-résolvente satisfaisant les conditions 1-2 ci-dessus. Il existe un littéral* $l(\overline{s})$*, un problème équationnel satisfaisable* $\mathcal{Y}$ *et deux substitutions* ρ *et* θ *tels que :*

$$C \models [\![\rho(l(\overline{s})) \vee \theta(\neg l(\overline{s})) \vee R \ : \mathcal{X} \wedge \mathcal{Y}]\!]$$

où

- $\theta\rho(\overline{s}) = \rho\theta(\overline{s})$
- $\rho(\mathcal{Y}) = \theta(\mathcal{Y}) = \mathcal{Y}$
- $\rho(R) = \theta(R) = R$
- $\mathcal{Y}$ *est de la forme* $\bigwedge_{i=1}^{n} x_i = t_i$*, où* x_i *n'apparaît pas dans* t_i *et où* $\theta(x_i) = \rho(x_i) = x_i$.

PREUVE. Une équation récursive est soit de la forme $x_i = t_i$, soit de la forme $\sigma(x_j) = t_j$ (où t_i et t_j ne sont pas des variables et contiennent $\sigma(x_i)$ et x_j respectivement).

Par conséquent, F_i est de la forme

$$\exists \overline{n}.F_i'' \wedge \bigwedge_{i=1}^{k} x_i = t_i \wedge \bigwedge_{j=k+1}^{k'} \sigma(x_j) = t_j.$$

Soit σ' la substitution $\sigma' : \sigma(x) \to x$ ($\sigma' = \sigma^{-1}$). Soit ρ la substitution $\rho : \sigma(x_j) \to t_j$ ($k < j \leq k'$), et θ la substitution $\theta : x_i \to t_i$ ($1 \leq i \leq k$). En premier lieu, nous montrons que la formule $\sigma'(F_i'') \Rightarrow \sigma'\rho\sigma(\overline{t}) = \sigma'\theta(\overline{t}')$ est équivalente à $\top$. Par la condition 2, $\theta(t_i) = \rho(t_i) = t_i$ et $\theta(t_j) = \rho(t_j) = t_j$. Donc $\rho\theta(x_i) = \rho\theta(t_i)$ et $\rho\theta(\sigma(x_j)) = \rho\theta(t_j)$. Par conséquent, la formule $\rho\theta(F_i)$ est équivalente à $\rho\theta(F_i'')$ (voir la définition de F_i ci-dessus). Puisque F_i'' ne contient aucune variable récursive, $\rho\theta(F_i'') = F_i''$. Par définition de F_i : $F_i \Rightarrow \sigma(\overline{t}) = \overline{t}' \equiv \top$. Par conséquent, $\rho\theta(F_i) \Rightarrow \rho\theta\sigma(\overline{t}) = \rho\theta(\overline{t}') \equiv \top$, i.e. $F_i'' \Rightarrow \rho\theta\sigma(\overline{t}) = \rho\theta(\overline{t}') \equiv \top$. En particulier $\sigma'(F_i'') \Rightarrow \sigma'\rho\theta\sigma(\overline{t}) = \sigma'\rho\theta(\overline{t}') \equiv \top$. Mais $\theta(\sigma(\overline{t})) = \sigma(\overline{t})$ et $\rho(\theta(\overline{t}')) = \theta(\overline{t}')$. Donc $\sigma'(F_i'') \Rightarrow \sigma'\rho\sigma(\overline{t}) = \sigma'\theta(\overline{t}')$ est équivalent à $\top$. Soit $\overline{s}$ le terme $\overline{t}\{t_i \to x_i\}$. Soit $\rho' = \sigma'\rho\sigma$, et $\theta' = \sigma'\theta$. Si la formule F_i'' est vraie, $\rho(\sigma(\overline{t})) = \theta(\overline{t}')$. Pour tout $i \in [1,k]$, $\rho(\sigma(x_i)) = \sigma(x_i)$, et pour tout $j \in [k+1, k']$, toute occurrence de $\sigma(x_i)$ dans $\theta(\overline{t}')$ apparaît dans le terme t_i. Donc x_i apparaît seulement dans t_i dans $\overline{t}$, et $\overline{t} = \theta'(\overline{s})$. De même, on prouve que si $\sigma'(F_i'')$ est vraie, alors $\rho'(\overline{s}) = \overline{t}'$.

$$C' : [\![l(\overline{t}) \vee \neg l(\overline{t'}) \vee R \ : \mathcal{X} \wedge \sigma'(F_i'')]\!]$$

peut s'écrire sous la forme suivante.

$$C' : [\![\theta'(l(\overline{s})) \vee \rho'(\neg l(\overline{s})) \vee R \ : \mathcal{X} \wedge \sigma'(F_i'')]\!].$$

où $\theta'\rho'(\overline{s}) = \rho'\theta'(\overline{s})$.

Il est très facile de voir que $C \models C'$. De plus $\rho'(R) = \theta(R) = R$, $\rho'(\sigma'(F_i'') \wedge \mathcal{X}) = \theta'(\sigma'(F_i'') \wedge \mathcal{X}) = \mathcal{F}_i'' \wedge \mathcal{X}$. C.Q.F.D.

Considérons la règle :

I-Induction

$$\frac{c : [\![l(\overline{t}) \vee \neg l(\overline{t'}) \vee R \ : \mathcal{X}]\!]}{[\![l(\overline{s'}) \vee \neg l(\overline{s''}) \vee R \ : \mathcal{X} \wedge \mathcal{Y}]\!]}$$

où :

1. c satisfait les conditions 1-2 ci-dessus.
2. $[\![\rho(l(\overline{s})) \vee \theta(\neg l(\overline{s})) \vee R \ : \mathcal{X} \wedge \mathcal{Y}]\!]$ est la I-c-clause donnée par le lemme 11.6.1.
3. k, t_i, et x_i sont définis comme dans le lemme 11.6.1.
4. t_i ne contient pas de N-terme u tel que u contient une variable récursive.
5. $\overline{s'}$ (resp. $\overline{s''}$) est obtenu à partir de s en remplaçant chaque terme x_i, pour $1 \leq i \leq k$ (resp. $k < i \leq n$) par $(t_j\{x_j \leftarrow \diamond\})^n.x_j$.

REMARQUE. La condition 4 est nécessaire afin que $(t_j\{x \leftarrow \diamond\})^n.x_j$ et $(t_i\{x \leftarrow \diamond\})^n.x_i$ soient des I-termes. Sans cette condition, nous pourrions par exemple avoir (si $t_i = f(\diamond)^n.x$) :

$$x \leftarrow (f(\diamond)^n.\diamond)^m.x$$

mais ce dernier terme n'est *pas* un I-terme.

THÉORÈME 11.6.2. *La règle* ***I-Induction*** *est correcte.*

PREUVE. Nous avons

$$C \models [\![\rho'(l(\overline{s})) \vee \theta'(\neg l(\overline{s})) \vee R \ : \mathcal{X} \wedge F_i'']\!]$$

Cette I-c-clause peut s'écrire

$$[\![R \vee (\theta'(l(\overline{s})) \Rightarrow \rho'(l(\overline{s}))) \ : \mathcal{X} \wedge F_i'']\!].$$

Puisque $\theta'\rho'(\overline{s}) = \rho'\theta'(\overline{s})$, il est facile de montrer (par induction sur n) que

$$C \models [\![R \vee (\theta'^n(l(\overline{s})) \Rightarrow \rho'^n(\neg l(\overline{s}))) \ : \mathcal{X} \wedge F_i'']\!]$$

C.Q.F.D.

EXEMPLE 11.6.2. Considérons la c-clause

$$[\![R(y,z) \vee P(x,y) \vee P(f(x),z) \ : \top]\!].$$

La forme normale de $P(x',y') = P(f(x),z)$ est

$$F : x' = f(x) \wedge y' = z$$

(dans ce cas, il n'y a qu'une seule solution).

x, x' sont des variables récursives, y, y', z, z' ne sont pas récursives.

Par conséquent, on a

$$C' : [\![R(y,z) \vee P(x,y) \vee P(f(\diamond)^n.x, z') \ : y = y' \wedge z = z' \wedge y' = z]\!].$$

Par simplification, on obtient $C' : [\![R(y,y) \vee P(x,y) \vee P(f(\diamond)^n.x, y) \ : \top]\!]$

Il est facile de voir que C' est une conséquence logique de C.

$\diamond$

REMARQUE. Si $n = 0$, la I-c-clause est une tautologie. Donc, ce cas peut être éliminé. Ainsi, l'algorithme de résolution des contraintes donné dans la section 11.5 est bien adapté à notre méthode de construction de modèles puisqu'on suppose que chaque variable entière est associée à un entier $n > 0$.

Les deux exemples ci-dessous illustrent le cas où les conditions 1 et 2 ne sont pas satisfaites.

EXEMPLE 11.6.3. (Condition 1 n'est pas satisfaite) Considérons la c-clause C :

$$[\![\neg P(x) \vee P(f(x)) \vee R(x) \ : \top]\!]$$

C donne les résolvents suivants.

$$[\![\neg P(x) \vee P(f^n(x)) \vee \bigvee_{i=1}^{n} R(f^n(x)) \ : \top]\!]$$

De façon évidente, cet ensemble de c-clauses ne peut pas être exprimé par notre formalisme des I-termes. ◇

EXEMPLE 11.6.4. (Condition 2 n'est pas satisfaite) Considérons la c-clause C :

$$[\![\neg P(f(x), y) \vee P(x, g(x, y)) \ : \top]\!]$$

C donne les résolvents :

$$[\![\neg P(f(f(x)), y) \vee P(x, g(x, g(f(x), y))) \ : \top]\!]$$

et :

$$[\![\neg P(f^n(x), y) \vee P(x, g(f(x), \ldots, g(f^n(x), y))) \ : \top]\!]$$

Là encore, il est impossible d'exprimer cet ensemble par des I-termes. ◇

11.6.2 Comparaison avec les travaux existants

La règle inductive de Salzer

(SALZER, 1992) propose une règle similaire pour des clauses standards. Cette règle utilise les R-termes au lieu des I-termes. Elle a par la suite été étendue en utilisant les grammaires primales (SALZER, 1994) et l'unification cyclique (SALZER, 1993).

Les principales différences entre la règle proposée dans ce travail et celle donnée dans (SALZER, 1992) sont les suivantes.

- En premier lieu, notre règle s'applique à des I-c-clauses contraintes et non seulement à des clauses (les clauses sont des cas particuliers des c-clauses).
- En second lieu, la règle de (SALZER, 1992) s'applique seulement à des clauses de la forme $Q \leftarrow P$ où Q est une instance de P (i.e. $Q = P\lambda$, pour une certaine substitution λ). Notre règle peut traiter des clauses de la forme $Q\lambda \leftarrow Q\theta$.

EXEMPLE 11.6.5. Soit C la clause

$$P(f(x), y) \vee \neg P(x, f(y)).$$

La règle d'introduction de I-termes déduit de C la clause

$$P(f(\diamond)^n.x, y) \vee \neg P(x, f(\diamond)^n.y).$$

Cette clause *ne peut pas* être déduite par la règle proposée dans (SALZER, 1992).

◇

La règle "Cycle Unification" de Klingenbeck

Dans (Klingenbeck, 1996) est proposée une règle, appelée **Cycle Unification**, qui est très similaire à la règle I-**Induction**. Les différences entre les deux règles sont les suivantes.

- Le langage des I-termes que nous utilisons est moins restrictif que celui utilisé dans (Klingenbeck, 1996). En effet, nous n'imposons aucune restriction sur la forme des contextes ni sur le nombre de trous dans un N-terme.
- La définition de la règle I-**Induction** que nous avons proposée est d'un plus bas niveau que celle de (Klingenbeck, 1996). En effet, la définition donnée dans (Klingenbeck, 1996) suppose que la c-clause est transformée sous une forme particulière permettant un calcul aisé des contextes. Mais l'algorithme de transformation n'est pas précisé. Au contraire, notre méthode n'impose aucune contrainte sur la forme de la c-clause et permet de calculer les contextes de façon purement automatique.

11.6.3 Exemples

Exemple 11.6.6. La formule suivante est tirée de (Dreben et Goldfarb, 1979, page 205). Il s'agit d'une formule de la classe $\forall\exists\forall$-classes avec identité. Cette classe n'est pas finiment contrôlable mais elle est docile[4]. La formule considérée est satisfaisable seulement sur les univers de cardinalité infinie ou paire.

$$\begin{array}{l}\forall y \exists x \forall z \\ \quad R(x,y) \wedge Q(y,y) \wedge (R(x,z) \Rightarrow (y = z)) \\ \quad \wedge (Q(y,z) \Rightarrow P(x,z)) \wedge (P(y,z) \Rightarrow Q(x,z)) \wedge (\neg P(y,z) \vee \neg Q(y,z))\end{array}$$

L'ensemble de c-clauses correspondant S est.

$$\begin{array}{l}1\ [\![R(f(y),y)\ : \top]\!] \\ 2\ [\![Q(y,y)\ : \top]\!] \\ 3\ [\![\neg R(f(y),z)\ : y \neq z]\!] \\ 4\ [\![\neg Q(y,z) \vee P(f(y),z)\ : \top]\!] \\ 5\ [\![\neg P(y,z) \vee Q(f(y),z)\ : \top]\!] \\ 6\ [\![\neg P(y,z) \vee \neg Q(y,z)\ : \top]\!]\end{array}$$

Il est clair (il suffit de considérer les clauses 4 et 5) que chaque modèle de S contient $Q(f^{2n}(x),x)$ et $\neg Q(f^{2n+1}(x),x)$. Ces ensembles de littéraux ne peuvent être exprimés par des formules équationnelles.

Notre méthode donne :

7	$[\![\neg Q(y,z) \vee Q(f(f(y)),z)\ : \top]\!]$	(c-resolution 4,5)
8	$[\![P(y,z) \vee P(f(f(y)),z)\ : \top]\!]$	(c-resolution 4,5)
9	$[\![\neg Q(y,z) \vee Q(f(f(\diamond))^n.y,z)\ : \top]\!]$	(I-terme introduction 7)
10	$[\![Q(f(f(\diamond))^n.y,y)\ : \top]\!]$	(resolution,9,2)
11	$[\![\neg P(f(f(\diamond))^n.y,y)\ : \top]\!]$	(resolution,10,6)
12	$[\![P(f(f(f(\diamond))^n.y),y)\ : \top]\!]$	(resolution,10,4)
13	$[\![\neg Q(f(f(f(\diamond))^n.y),y)\ : \top]\!]$	(resolution,12,6)
14	$[\![\neg Q(y,z) \vee P(f(y),z)\ : \forall n.y \neq f(f(f(\diamond))^n.z)]\!]$	(dissubsumption,10,4)
15	$[\![\neg Q(y,z)\ : \forall n.y \neq f(f(f(\diamond))^n.z) \wedge y \neq z \wedge y \neq f(f(\diamond))^n.z]\!]$	(GMPL)
15	$[\![\neg P(y,z)\ : \forall n.y \neq f(f(f(\diamond))^n.z) \wedge y \neq z \wedge y \neq f(f(\diamond))^n.z]\!]$	(GMPL)

4. Une classe est dite *finiment contrôlable* si toute formule satisfaisable de cette classe admet un modèle fini (certains auteurs parlent de "propriété du modèle fini"). Elle est *docile* s'il existe une méthode effective pour décider si une formule de cette classe a un modèle fini ((Dreben et Goldfarb, 1979)).

On obtient le modèle suivant.

$$
\begin{array}{l}
[\![Q(f(f(\diamond))^n.y, y) \ : \top]\!] \\
[\![\neg P(f(f(\diamond))^n.y, y) \ : \top]\!] \\
[\![\neg Q(f(f(f(\diamond))^n.y), y) \ : \top]\!] \\
[\![P(f(f(f(\diamond)))^n.y, y) \ : \top]\!] \\
[\![\neg Q(x, y) \ : x = f(\diamond)^n.a \wedge y = f(\diamond)^m.a \wedge n < m]\!] \\
[\![\neg P(x, y) \ : x = f(\diamond)^n.a \wedge y = f(\diamond)^m.a \wedge n < m]\!]
\end{array}
$$

◇

EXEMPLE 11.6.7. Considérons l'ensemble des 4 clauses suivantes.

$$
\begin{array}{l}
1\ [\![\neg Q(x, f(x)) \ : \top]\!] \\
2\ [\![P(x, f(x)) \ : \top]\!] \\
3\ [\![\neg P(x, y) \vee Q(x, f(y)) \ : \top]\!] \\
4\ [\![\neg Q(x, y) \vee Q(x, f(y)) \ : \top]\!]
\end{array}
$$

Cet ensemble de c-clauses satisfaisable n'a pas de modèle fini (DREBEN ET GOLDFARB, 1979).

5	$[\![\neg P(x, x) \ : \top]\!]$	(c-resolution 3,1)
6	$[\![\neg Q(x, x) \ : \top]\!]$	(c-resolution 4,1)
7	$[\![\neg P(x, y) \vee Q(x, f(y)) \ : x \neq y]\!]$	(c-dissubsumption 1,3)
8	$[\![\neg P(x, y) \ : y \neq f(x)]\!]$	(GPL)
9	$[\![\neg Q(x, y) \vee Q(x, f^n(\diamond).y) \ : \top]\!]$	(I-terme introduction 4)
10	$[\![\neg Q(x, y) \vee Q(x, f^n(\diamond).y) \ : x \neq y]\!]$	(c-dissubsumption, 6, 9)
11	$[\![Q(x, f^n(\diamond).y) \ : x \neq y]\!]$	(GPL)

On obtient par conséquent le modèle suivant.

$$
\begin{array}{l}
[\![\neg Q(x, f(x)) \ : \top]\!] \\
[\![P(x, f(x)) \ : \top]\!] \\
[\![\neg P(x, x) \ : \top]\!] \\
[\![\neg Q(x, x) \ : \top]\!] \\
[\![\neg P(x, y) \ : y \neq f(x)]\!] \\
[\![Q(x, f^n(\diamond).y) \ : x \neq y]\!]
\end{array}
$$

Il est intéressant de constater qu'aucune autre méthode n'est capable de construire un modèle pour cette formule.

◇

Chapitre 12

Automates d'arbres et construction de modèles

Bien que beaucoup plus expressif que les contraintes équationnelles interprétées dans la théorie vide, le formalisme de représentation proposé au chapitre 11 souffre de certaines limitations, liées à la *complexité* de la résolution des formules du premier ordre dans la théorie des I-termes et au *pouvoir d'expression* de ces formules.

- D'une part, l'utilisation des schématisations de termes est particulièrement coûteuse, au vu de la complexité des algorithmes de résolution existants. (Comon, 1995) montre que le problème d'unification dans la théorie des I-termes est d'une complexité au moins équivalente à la résolution des équations diophantiennes.

- D'autre part, il existe des modèles très simples qui ne peuvent pas être exprimés en utilisant les I-termes. C'est essentiellement dû, d'une part, à l'unicité du contexte dans la définition inductive des I-termes, et d'autre part à l'absence de variables anonymes (c'est-à-dire pouvant être instanciées différemment à chaque étape) dans les contextes. Ces deux hypothèses sont par ailleurs nécessaires à la décidabilité de la théorie (Hermann, 1994).

  Considérons, par exemple, la formule suivante.

$$\forall x.(P(x) - \ (x = a \vee \exists y.(x = f(y) \wedge P(y)) \vee \exists y.(x = g(y) \wedge P(y)))).$$

  Le seul modèle de Herbrand de cette formule est l'ensemble $\mathcal{M}$, inductivement défini par les relations suivantes.

  - $P(a) \in \mathcal{M}$;
  - si $x \in \mathcal{M}$, $P(f(x))$ et $P(g(x))$ sont dans $\mathcal{M}$.

  Cet ensemble de termes *ne peut pas être exprimé par une formule équationnelle sur les I-termes*, car à chaque étape inductive, tout terme de $\mathcal{M}$ peut générer deux nouveaux termes distincts $P(f(x))$ et $P(g(x))$.

  Considérons maintenant la formule $P(a) \wedge \forall x.(P(x) \Rightarrow \forall y.P(f(x,y)))$. Le modèle minimal de cette formule est l'ensemble $\mathcal{M}$ défini inductivement par $a \in \mathcal{M}$, et si $t \in \mathcal{M}$ alors pour tout s, $f(t,s) \in \mathcal{M}$. Là encore, cet ensemble ne peut être représenté par des schématisations de termes car des termes y *distincts* sont introduits dans le terme à chaque étape inductive (pour plus de détails sur ce problème, consulter (Salzer, 1993; Hermann, 1994)).

Afin de remédier à ce double inconvénient, nous nous intéressons à un autre formalisme permettant d'exprimer les interprétations: *les automates d'arbres* (Gécseg et Steinby, 1984) qui

constituent un formalisme disposant de "bonnes" propriétés de décidabilité permettant de représenter de façon naturelle et efficace des modèles ne pouvant pas être exprimés par les I-termes. Les automates d'arbres ont fait l'objet de nombreuses études (Gécseg et Steinby, 1984; Comon, 1988; Bogaert et Tison, 1992; Comon et Delor, 1994). En particulier, la classe des langages représentables par des automates d'arbres standard est la classe des langages réguliers, i.e. des langages pouvant être représentés par une grammaire d'arbres régulière. Cette classe est fermée par les opérations booléennes usuelles (intersection, union, complément). D'autre part, le problème du vide (i.e. décider si une grammaire représente le langage vide) est connu pour être décidable en temps linéaire.

L'inconvénient des automates d'arbres est qu'ils ne permettent pas de représenter des termes non linéaires tels que par exemple $\{f(x,x)/x \in \tau(\Sigma)\}$ ou $\{f(x,y)/(x,y) \in \tau(\Sigma)^2, x \neq y\}$. Or, ces termes peuvent être facilement représentés par des formules équationnelles. L'idée de chercher à combiner ces deux représentations est donc très naturelle. Par ailleurs, il existe plusieurs extensions des automates d'arbres qui disposent des mêmes propriétés que les automates d'arbres (décidabilité du problème du vide et stabilité par les opérations boolénnes usuelles). Citons par exemple la classe des *automates d'arbres avec tests d'égalité entre fils* ($REQ_{\neq}$-automates).

Nous montrons dans ce chapitre que le formalisme des automates d'arbres peut être intégré de façon profitable à notre approche de la construction de modèles. Contrairement au chapitre 11, aucun résultat de décidabilité n'est établi ici. Le but, plus modeste, est simplement de fournir des techniques permettant d'utiliser ce formalisme et de le combiner avec la méthode Ramc.

Nous proposons de représenter les modèles par des *formules avec contraintes d'appartenance* (Comon et Delor, 1994) i.e. des formules contenant les prédicats "$=$" et "$\in$", interprétés sur la théorie vide. Par conséquent, les formules atomiques seront soit de la forme $t = s$ (où s, t sont des termes), soit de la forme $t \in S$, où t est un terme et S un symbole dénotant un ensemble de termes. Les ensembles apparaissant dans la formule seront définis par un automate G. Nous appellerons de telles formules *G-formules*. De façon évidente, les G-formules ont un pouvoir d'expression plus important que les formules équationnelles.

12.1 Automates d'arbres généraux

Il est possible d'étendre les automates d'arbres en autorisant des tests d'égalité ou de "diségalité" entre les sous-termes d'un terme de production. On introduit ainsi la *classe d'automates avec tests d'égalité* (Rateg) (Mongy, 1981). Malheureusement, le problème du vide est indécidable pour cette classe d'automates. Cependant il est possible de restreindre la classe d'automates autorisée afin de préserver les propriétés des grammaires régulières. Par exemple, si seuls des tests d'égalité ou de diségalité entre *fils* sont autorisés, alors le problème du vide devient décidable, et la classe de langage obtenue est stable par les transformations booléennes habituelles (union, intersection et complément) (Bogaert et Tison, 1992).

Nous donnons ci-dessous une définition très générale des automates d'arbres. Nous n'imposons aucune condition *syntaxique* sur la classe d'automates considérée, mais simplement des conditions *sémantiques* qui seront nécessaires pour utiliser ces automates pour la représentation et la construction des modèles.

Nous appellerons "automate d'arbre (général)" tout automate d'arbre avec test d'égalité ou de diségalité. Plus précisément :

Définition 12.1.1. *[Automates d'arbres généraux] Soit $\mathcal{S}_d$ un ensemble de symbole de sorte disjoint de $\mathcal{S}$ appelé* sortes dynamiques. *Un* automate d'arbres *(général) est un ensemble fini de c-clauses de la forme*

$$[\![\bigwedge_{i=1}^{n} P_i(t_i) \Rightarrow P(t) \ : \mathcal{X}]\!]$$

où $\forall i \leq n.t_i \prec t$, $P_i \in \mathcal{S}_d$, $P \in \mathcal{S}_d$. ◇

DÉFINITION 12.1.2. *[Sémantique des automates d'arbres généraux] Si G est un automate d'arbres, G est un ensemble de c-clauses de Horn, donc admet un modèle minimal de Herbrand unique (voir chapitre 14) $\mathcal{M}$. Nous noterons $\tau_G(P)$ l'ensemble des termes clos t de $\tau(\Sigma)$ tels que $\mathcal{M} \models P(t)$.* ◇

REMARQUE. Soit G un automate d'arbre.

- Si pour toute règle, $[\![\bigwedge_{i=1}^{n} P_i(t_i) \Rightarrow P(t) \ : \mathcal{X}]\!]$ de G, t est linéaire et $\mathcal{X} \equiv \top$, alors G est un automate d'arbre standard.
- Si pour toute règle, $[\![\bigwedge_{i=1}^{n} P_i(t_i) \Rightarrow P(t) \ : \mathcal{X}]\!]$ de G, t est de la forme $f(t_1, \ldots, t_n)$, où t_i est une variable et si $\mathcal{X}$ est une conjonction de diséquations ou équations entre variables de $\{t_1, \ldots, t_n\}$, alors G est un automate d'arbres avec tests d'égalité entre frères.

12.1.1 Formules équationnelles et automates d'arbres généraux

Les automates d'arbres peuvent être facilement combinés avec les formules équationnelles, par l'ajout de contraintes d'appartenance dans les formules (COMON ET DELOR, 1994). Le langage obtenu fournit un moyen naturel et efficace de représenter des nuplets de termes fermés.

DÉFINITION 12.1.3. *Une* expression de sorte s *est inductivement définie de la façon suivante.*

- $s \in \mathcal{S} \cup \mathcal{S}_d$.
- *Si $s_1, \ldots, s_n$ sont des expressions de sorte et arité$(f) = n$, alors $f(s_1, \ldots, s_n)$ est une expression de sorte.*
- *Si s_1, s_2 sont deux expressions de sorte, alors $s_1 \cup s_2$ et $s_1 \cap s_2$ sont deux expressions de sorte.*
- *Si s est une expression de sorte alors $\neg s$ est une expression de sorte.*
- $\top$ *et* $\bot$ *sont des expressions de sorte.*

De même que nous associons à chaque symbole de sorte une interprétation qui est un sous-ensemble de $\tau(\Sigma)$, nous associons également à chaque expression de sorte un sous-ensemble de $\tau(\Sigma)$ noté $\tau_G(S)$ et inductivement défini comme suit.

- $\tau_G(s) = \tau_s(\Sigma)$ *si* $s \in \mathcal{S}$.
- $\tau_G(f(s_1, \ldots, s_n)) = \{f(t_1, \ldots, t_n) / \forall i \in [1..n], t_i \in \tau_G(s_i)\}$.
- $\tau_G(s_1 \cap s_2) = \tau_G(s_1) \cap \tau_G(s_2)$.
- $\tau_G(s_1 \cup s_2) = \tau_G(s_1) \cup \tau_G(s_2)$.
- $\tau_G(\neg s) = \tau(\Sigma) \setminus \tau_G(s)$.
- $\tau_G(\top) = \tau_G(\Sigma)$.
- $\tau_G(\bot) = \emptyset$.

◇

DÉFINITION 12.1.4. *Soit G un automate d'arbres. Une G-formule $\mathcal{P}$ est une formule du premier ordre qui est soit une équation $s = t$, où s et t sont deux termes, $\bot$ (false) $\top$ (true), soit de la forme $\mathcal{P} \vee \mathcal{Q}$, $\mathcal{P} \wedge \mathcal{Q}$, $\neg\mathcal{P}$, $\exists x.\mathcal{P}$, $\forall x.\mathcal{P}$ où $\mathcal{P}$ et $\mathcal{Q}$ sont des G-formules, $x \in \mathcal{V}$, soit de la forme $t \in s$ où $t \in \tau(\Sigma, \mathcal{V})$ et s est une expression de sorte. Les variables libres d'une G-formule $\mathcal{P}$ sont appelées des* inconnues. *Les formules $\neg(t = s), \neg(t \in s)$ sont également notées $t \neq s$ et $t \notin s$.* ◇

Une G-formule étant une formule du premier ordre, les notions de *sous-formule* et de *position* dans une G-formule sont définies de façon habituelle.

Pour étendre la notion de *solution* d'une formule (voir (COMON ET LESCANNE, 1989) et chapitre 2) aux G-formules, il suffit de préciser l'interprétation de $\in$.

DÉFINITION 12.1.5. *Une substitution* σ valide *une* G*-formule* $t \in s$ *si et seulement si* $t\sigma \in \tau_G(s)$. *La notion de* solution d'une G-formule *s'étend alors de façon immédiate aux* G*-formules quelconques (par souci de concision, nous ne rappellerons pas toutes les définitions). L'ensemble des solutions d'une* G*-formule* $\mathcal{X}$ *est noté* $\mathcal{S}_G(\mathcal{X})$. ◇

Introduisons quelques notations.

DÉFINITION 12.1.6. *[Admissibilité d'une classe d'automates]*
Une classe d'automates $\mathcal{C}$ *sera dite* admissible *si et seulement si :*

(C_1) *il existe une procédure permettant de décider (pour tout* $G \in \mathcal{C}$*) si une* G*-formule* $\mathcal{X}$ *donnée est satisfaisable ;*

(C_2) *il existe une procédure permettant de décider de l'appartenance à* $\mathcal{C}$*;*

(C_3) *pour toute expression de sorte* e*, il est possible de construire un automate* $G' \in \mathcal{C}$ *contenant* G *tel que pour tout symbole de sorte* s *de* G, $\tau_G(s) = \tau_{G'}(s)$ *et tel qu'il existe un symbole de sorte* s' *vérifiant* $\tau_G(e) = \tau_{G'}(s')$.

◇

REMARQUE. La classe des automates standards (équivalents aux grammaires régulières) et la classe d'automates d'arbres avec tests entre frères (les $\mathrm{REG}_{\neq}$-automates) sont admissibles (COMON ET DELOR, 1994).

Dans la suite, nous supposerons que la classe d'automates considérée est admissible. Cette approche permet d'inclure dans notre méthode diverses procédures de résolution des contraintes, de façon modulaire, sans avoir à redonner à chaque fois de nouvelles définitions, règles et théorèmes. Nous nous contentons ici de préciser les propriétés dont nous avons besoin sur la classe d'automates sans définir explicitement cette classe.

EXEMPLE 12.1.1. $G = \{S(a), S(x) \to S(f(f(x)))\}$ est un automate d'arbres. L'ensemble $\tau_G(S)$ correspondant à S est $\{f^{2n}(a)/n \in \mathbb{N}\}$.

La formule $\mathcal{P} = x = y \vee (x = f(y) \wedge y \in S)$ est une G-formule (x et y sont des variables libres).

L'ensemble des *solutions* de $\mathcal{P}$ est l'ensemble de substitutions fermées σ telles que.

- soit $x\sigma = y\sigma$;
- soit $x\sigma = f(y\sigma)$ et $y\sigma \in \tau_G(S)$.

Par exemple $\{x \to f(f(f(a))), y \to f(f(a))\}$ est une solution de $\mathcal{P}$.

◇

12.1.2 Représentation des interprétations par des G-formules

Les G-formules peuvent être utilisées pour représenter des n-uplets de termes fermés et des interprétations de Herbrand. Rappelons brièvement les définitions correspondantes.

DÉFINITION 12.1.7. *Un ensemble* E *de* $\tau(\Sigma)^n$ *est appelé un* G*-ensemble si et seulement si il existe un automate admissible* $G = \mathfrak{G}(E)$ *et une* G*-formule* $\mathfrak{F}_G(E)$ *contenant* n *variables libres* $x_1, \ldots, x_n$ *vérifiant*

$$(t_1, \ldots, t_n) \in E - \{x_i \to t_i\} \in \mathcal{S}_G(\mathfrak{F}_G(E)).$$

*Une interprétation partielle $\mathcal{I}$ est une G-*interprétation *ssi pour tout P, $\mathcal{I}_P^+$ et $\mathcal{I}_P^-$ sont des G-ensembles. Une G-interprétation peut être représentée par la donnée d'une G-formule et d'un ensemble fini de c-clauses unitaires, dont les parties contraintes sont des G-formules.* $\diamond$

THÉORÈME 12.1.1. *Il existe un algorithme qui, étant donné une G-interprétation totale $\mathcal{I}$ et une formule logique du premier ordre $\mathcal{F}$ décide si $\mathcal{I} \models \mathcal{F}$.*

PREUVE. La preuve est une conséquence de l'admissibilité de la classe d'automates considérée et du corollaire 5.5.1. C.Q.F.D.

Il est intéressant de noter que le formalisme des G-formules subsume les techniques de représentation de modèles finis :

THÉORÈME 12.1.2. *Soit $\mathcal{I}$ une interprétation finie, telle que pour tout élément a du domaine de $\mathcal{I}$, il existe un terme t de $\tau(\Sigma)$ tel que $\mathcal{I}(t) = a$ (i.e. $\mathcal{I}$ est engendré sur Σ). Il existe un automate d'arbres G et une interprétation G-représentable $\mathcal{J}$ telle que pour toute formule $\mathcal{F}$*

$$\mathcal{I} \models \mathcal{F} - \mathcal{J} \models \mathcal{F}.$$

PREUVE. La preuve est immédiate.

Soit $\mathcal{I}$ une interprétation de domaine fini $\{a_1, \ldots, a_n\}$. Soit $\{A_1, \ldots, A_n\}$ n symboles de sorte dynamique distincts. Nous définissons l'automate G correspondant et l'interprétation $\mathcal{J}$ comme l'ensemble des clauses suivant :

$$\{\bigwedge_{j=1}^{n} A_{i_j}(x_{i_j}) \rightarrow A_i(f(x_{i_1}, \ldots, x_{i_k}))/f \in \Sigma, \mathcal{I}(f)(a_{i_1}, \ldots, a_{i_k}) = a_i\}.$$

Alors $\mathcal{J}$ est l'interprétation totale définie, pour tout prédicat P (où $a(P) = k$ et $\overline{x}_P = (x_1, x_2, \ldots, x_k)$) par :

$$\mathfrak{F}_G(\mathcal{I}_P^+) = \bigvee_{(a_{i_1}, \ldots, a_{i_k})/\mathcal{I} \models P(a_{i_1}, \ldots, a_{i_k})} \bigwedge_{j=1}^{k} x_i \in A_{i_j}.$$

On a alors $\mathcal{J} \models S$ ssi $\mathcal{I} \models S$ (c'est une conséquence triviale de la définition de $\mathcal{J}$, qui peut être démontrée par induction sur l'ensemble des formules). C.Q.F.D.

12.2 Extension de la méthode RAMC

Les résultats décrits dans la section 12.1.1 permettent d'incorporer l'utilisation d'automates d'arbres à la méthode RAMC. Plus précisément, nous considérons désormais des couples (G, S), où G est un automate d'arbres et S un ensemble de c-clauses, tels que les contraintes apparaissant dans S sont des G-formules. Un couple (G, S) est équivalent à l'ensemble des clauses fermées $C\sigma$ telles qu'il existe $[\![C \ : \mathcal{X}]\!] \in S$ et $\sigma \in \mathcal{S}_G(\mathcal{X})$. En utilisant cette équivalence, les notions usuelles de satisfaisabilité, insatisfaisabilité, implication, etc., peuvent être étendues de façon immédiate. Définissons alors les (méta-)règles suivantes.

Règles d'inférence et de disinférence

$$\frac{(G, S)}{(G, S')}$$

si $S \rightarrow_{\text{RAMC}} S'$

Règle de simplification

$$\frac{(G, S \cup \{[\![C \ : \mathcal{X}]\!]\})}{(G, S)}$$

si $\mathcal{S}_G(\mathcal{X}) \equiv \bot$

Initialement, si S est un ensemble de c-clauses sans symbole de sorte, l'ensemble de règles G est vide. Sinon, G doit bien entendu être fourni par l'utilisateur en même temps que le problème à résoudre. Puisqu'il existe une procédure de décision pour les G-formules, les résultats du chapitre 6 (correction, complétude réfutationnelle, etc.) s'étendent de façon immédiate.

Nous montrons dans cette section comment utiliser l'information déduite par la méthode RAMC pour générer *dynamiquement* de nouveaux symboles de sorte pendant la recherche. Nous présentons une nouvelle règle permettant d'introduire automatiquement de nouveaux symboles de sorte et de nouvelles règles de production dans l'automate G. Ces règles opèrent sur le couple (G, S) et non sur l'ensemble de c-clauses S. L'idée de base est similaire à celle utilisée au chapitre 11. Il s'agit de tirer parti du pouvoir d'expression des G-formules pour empêcher la divergence de la méthode.

Donnons tout d'abord une description informelle de notre approche. Une description formelle sera donnée par la section 12.2.2.

12.2.1 Présentation informelle sur un exemple

Considérons une fois de plus la formule

$$\mathcal{F} : P(a) \wedge \forall x.(P(x) - \neg P(f(x))).$$

Au chapitre 11, il a été montré que $\mathcal{F}$ admet un unique modèle de Herbrand $\mathcal{M}$.

$$\mathcal{M} = \{P(f^{2n}(a))/n \in \mathbb{N}\}$$

$\mathcal{M}$ *ne peut pas* être exprimé par un problème équationnel, mais peut être représenté par une G-formule. L'automate correspondant à cet ensemble est

$$\begin{aligned} &\rightarrow s(a) \\ s(x) &\rightarrow s(f(f(x))). \end{aligned}$$

Le modèle est alors représenté comme suit.

$$P(x) \text{ est } \mathbf{true} \text{ si et seulement si } x \in s$$

L'objet de cette section est de donner des conditions *syntaxiques* permettant de générer *automatiquement* cet automate. Pour cela, considérons la forme clausale de la formule $\mathcal{F}$.

$$\begin{aligned} c_1 &: [\![P(a) \ : \top]\!] \\ c_2 &: [\![P(x) \vee P(f(x)) \ : \top]\!] \\ c_3 &: [\![\neg P(x) \vee \neg P(f(x)) \ : \top]\!] \end{aligned}$$

En appliquant la règle de c-résolution sur les c-clauses c_2 et c_3, on obtient la c-clause

$$c_4 : [\![\neg P(x) \vee P(f(f(x))) \ : \top]\!].$$

En appliquant la règle de résolution sur $P(a)$ et c_4 (et sur les descendants de c_4) nous obtenons un ensemble infini $\mathcal{M}$ de c-clauses fermées de la forme $P(f(f(a)))$, $P(f(f(f(f(a)))))$,

$P(f^{2n}(a))\ldots$Afin d'éviter de générer explicitement ces c-clauses, il suffit de remarquer que les c-clauses c_1 et c_4 définissent un automate d'arbres (plus précisément une grammaire régulière d'arbres). Il est donc possible d'associer à P un nouveau symbole de sorte s et d'ajouter les règles :

$$\rightarrow a$$
$$s(x) \rightarrow s(f(f(x)))$$

La section suivante est consacrée à une description d'une méthode générant automatiquement un tel automate.

12.2.2 Une nouvelle règle

La définition suivante formalise cette idée.

DÉFINITION 12.2.1. *[Ensemble de contextes] Un* ensemble de contextes *est un ensemble de triplets* $\{(P_i, \overline{p}_i, s_i)/1 \leq i \leq n\}$ *tel que pour tout* $i \in [1..n]$ *:*

- P_i *est un littéral (positif ou négatif) ;*
- $\overline{p}_i$ *est un ensemble de positions de* $\mathcal{P}os(P_i)$*, incomparables 2 à 2 ;*
- s_i *est un symbole de sorte.*

◇

Informellement, un ensemble de contextes correspond à une *abstraction* (voir chapitre 13 et (PLAISTED, 1981) pour plus de détails sur les abstractions) d'un ensemble de littéraux. En remplaçant un littéral $P[t]_{\overline{p}}$ (ou $(P, \overline{p}, s) \in E$) par le littéral monadique $s(t)$, on transforme un ensemble de c-clauses en un automate d'arbres. Il suffit ensuite de vérifier si l'automate obtenu appartient à une classe admissible (c'est l'objet de la condition C_2). Les définitions suivantes formalisent cette technique.

DÉFINITION 12.2.2. *Soit* E *un ensemble de contextes et* L *un littéral.* L *est dit* E*-compatible si et seulement si* L *est de la forme* $P[t]_{\overline{p}}$ *(resp.* $\neg P[t]_{\overline{p}}$*) avec* $(P, \overline{p}, s) \in E$*. Nous notons alors* $\mathbf{abst}(E, L)$ *un littéral* $s(t)$ *(resp.* $\neg s(t)$*)* [1] ◇

EXEMPLE 12.2.1. Soit $E = \{(P(a, x), \{2\}, s)\}$.

Les littéraux

$$P(a, f(y, z)), P(a, a), \neg P(a, b)$$

sont E-compatibles. On a

$$\mathbf{abst}(E, P(a, f(y, z))) = s(f(y, z))$$
$$\mathbf{abst}(E, P(a, a)) = s(a)$$
$$\mathbf{abst}(E, \neg P(a, b)) = \neg s(b)$$

◇

Nous pouvons étendre cette notion à des ensembles de c-clauses quelconques.

DÉFINITION 12.2.3. *Soit* E *un ensemble de contextes et* S *un ensemble de c-clauses.* S *est dit* E*-compatible si et seulement si les conditions suivantes sont vérifiées.*

- *Tout littéral* L *apparaissant dans une clause de* S *est* E*-compatible. On note alors* $\mathbf{abst}(E, S)$ *l'ensemble de c-clauses obtenu en remplaçant chaque littéral* L *de* S *par un littéral* $\mathbf{abst}(E, S)$*.*

1. Plusieurs solutions sont parfois possibles, car un littéral peut correspondre à plusieurs littéraux de E. Il est alors nécessaire d'effectuer un choix arbitraire pour définir la fonction $\mathbf{abst}(E, L)$.

– $\mathbf{abst}(E,S)$ *appartient à une classe admissible d'automates d'arbres généraux.*

◇

LEMME 12.2.1. *[Compatibilité avec l'instantiation] Soit E un ensemble de contextes et S un ensemble de c-clauses E-compatible. Soit σ une substitution. Alors*

$$\mathbf{abst}(E\sigma, S\sigma) = \mathbf{abst}(E,S)\sigma$$

PREUVE. La preuve est immédiate. C.Q.F.D.

Si un ensemble de c-clauses est E-compatible, il est possible d'utiliser l'automate d'arbres $\mathbf{abst}(E,S)$ correspondant afin d'exprimer (une partie de) l'ensemble des c-clauses unitaires qui sont des conséquences logiques de S, au lieu de générer explicitement ces c-clauses par la règle de résolution. Cela permet d'empêcher dans certains cas la non-terminaison de la méthode, car l'ensemble ainsi exprimé est souvent infini.

THÉORÈME 12.2.1. *Soit E un ensemble de contextes et (G,S) un ensemble de c-clauses E-compatibles. Alors l'ensemble*

$$(G \cup \mathbf{abst}(E,S), \{[\![P[x]_{\overline{p}} \ : x \in s]\!]/(P,\overline{p},s) \in E\})$$

est une conséquence logique de S.

La preuve du théorème 12.2.1 utilise les lemmes suivants.

LEMME 12.2.2. *Soit E un ensemble de contextes et S un ensemble de c-clauses E-compatibles. S'il existe 2 c-clauses C_E et D_E de $\mathbf{abst}(E,S)$ telles que $\mathcal{R}es(C_E,D_E) = R_E$, alors il existe deux c-clauses C, D de S telles que $\mathcal{R}es(C,D) = R$, l'ensemble $S' = S \cup \{R\}$ est E-compatible et on a $\mathbf{abst}(E,S') = \mathbf{abst}(E,S) \cup \{R_E\}$.*

PREUVE. Par définition de $\mathcal{R}es$, C_E et D_E sont de la forme $[\![s_P(t) \cup R_1 \ : \mathcal{X}]\!]$ et $[\![\neg s_P(s) \cup R_2 \ : \mathcal{Y}]\!]$ et $R_E = [\![R_1 \cup R_2 \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{s} = \overline{t}]\!]$.

Par définition de $\mathbf{abst}(E,S)$, il existe deux c-clauses de S, de la forme $C : [\![P[t]_{\overline{p}} \cup R'_1 \ : \mathcal{X}]\!]$ et $D : [\![P[t]_{\overline{p}} \cup R'_2 \ : \mathcal{Y}]\!]$, telles que $\mathbf{abst}(E,R'_1) = R_1$ et $\mathbf{abst}(E,R'_2) = R_2$.

On a $\mathcal{R}es(C,D) = R : [\![R'_1 \cup R'_2 \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{t} = \overline{s}]\!]$. D'autre part on a $\mathbf{abst}(E,R) = [\![\mathbf{abst}(E,R'_1) \vee \mathbf{abst}(E,R'_2) \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{t} = \overline{s}]\!]$. D'où: $\mathbf{abst}(E,R) = R_E$ et $\mathbf{abst}(E,S') = \mathbf{abst}(E,S) \cup R_E$ C.Q.F.D.

LEMME 12.2.3. *Soit E un ensemble de contextes et S un ensemble de c-clauses E-compatible. Si $\mathbf{abst}(E,S)$ est insatisfaisable, alors S l'est également.*

PREUVE. $\mathbf{abst}(E,S)$ est un ensemble de c-clauses de Horn insatisfaisable, donc il existe une réfutation de $\mathbf{abst}(E,S)$ utilisant la règle de résolution. Nous allons montrer par induction sur la longueur n de cette réfutation (notée $\mathbf{abst}(E,S) \to S_1 \to \ldots \to S_n$) que S est insatisfaisable.

- $n = 0$. On a alors $\square \in \mathbf{abst}(E,S)$, d'où par définition de $\mathbf{abst}(E,S)$, $\square \in S$.
- $n > 0$. En appliquant le lemme 12.2.2, il existe un ensemble de c-clauses S'_1 tel que $S \to_{\mathcal{R}es} S'_1$, $\mathbf{abst}(E,S'_1) = S_1$ et S'_1 est insatisfaisable. Par hypothèse d'induction, S_1 est insatisfaisable. D'où S est insatisfaisable.

C.Q.F.D.

PREUVE. [du théorème 12.2.1] Par définition, pour tout terme de $\tau_G(s)$, $s(t)$ est une conséquence logique de $\mathbf{abst}(E,S)$ (voir définition 12.1.2). D'où, pour toute substitution $\sigma : \{x \to t\}$ telle que $t \in \tau_G(s)$, $\mathbf{abst}(E,S) \cup \{s(t)\}$ est insatisfaisable. Or on a par définition $\mathbf{abst}(E, P[t]_{\overline{p}}) = s(t)$. Par le lemme 12.2.3, on en déduit que $S \cup \{P[t]_{\overline{p}}\}$ est insatisfaisable. C.Q.F.D.

La règle G-Induction

$$\frac{(G, S \cup \{[\![C_i \vee R_i \ : \mathcal{X}_i \wedge \mathcal{Y}_i]\!]\})}{(G \cup \mathbf{abst}(E, S'\theta), S \cup \{[\![C_i \vee R_i \ : \mathcal{X}_i \wedge \mathcal{Y}_i]\!]\} \cup \{[\![R \vee P[x]_{\overline{p}} \ : x \in s \wedge \mathcal{Y}]\!]\theta / (P, \overline{p}, s) \in E\})}$$

avec :

- E est un ensemble de contextes.
- $S' = \{[\![C_i \ : \mathcal{X}_i]\!] / 1 \leq i \leq n\}$.
- Il existe θ telle que $S'\theta$ est E-compatible.
- $\mathbf{abst}(E, S'\theta)$ et $[\![R_i \ : \mathcal{Y}_i]\!]$ ne partagent aucune variable.
- $R = \bigvee_{i=1}^{n} R_i$.
- $\mathcal{Y} = \bigwedge_{i=1}^{n} \mathcal{Y}_i$.

THÉORÈME 12.2.2. *La règle G-**Induction** est correcte.*

PREUVE. Soit $(P, \overline{p}, s) \in E$. Notons C la c-clause $[\![R \vee P[x]_{\overline{p}} \ : x \in s \wedge \mathcal{Y}]\!]$. Soit un modèle $\mathcal{M}$ de S et une substitution fermée θ' des variables de $S\theta$ n'appartenant pas à $\mathbf{abst}(E, S'\theta)$. Soit $\sigma = \theta.\theta'$. Montrons que $\mathcal{M} \models C\sigma$.

Puisque $\forall i \in [1..n]$, $[\![R_i \ : \mathcal{Y}_i]\!]$ et $\mathbf{abst}(E, S)$ ne partagent aucune variable, $[\![R_i \ : \mathcal{Y}_i]\!]\theta\sigma$ est fermée. S'il existe $i \in [1..n]$, tel que $\mathcal{Y}_i\sigma = \bot$ alors de façon évidente $\sigma \notin \mathcal{S}_G(\mathcal{Y})$ (par définition de $\mathcal{Y}$), d'où $\mathcal{M} \models C\sigma$ (dans ce cas, $C\sigma$ est une tautologie).

Si $\forall i \in [1..n], \mathcal{Y}_i\sigma = \top$, alors on a $\mathcal{M} \models C_i\sigma \vee R_i\sigma$. S'il existe $i \in [1..n]$ tel que $\mathcal{M} \models R_i\sigma$, alors $\mathcal{M} \models R\sigma$, d'où $\mathcal{M} \models C\sigma$.

Sinon, on a $\forall i . \mathcal{M} \models [\![C_i \ : \mathcal{X}_i]\!]\sigma$. Or $[\![C_i \ : \mathcal{X}_i]\!]$ est E-compatible donc $[\![C_i \ : \mathcal{X}_i]\!]\sigma$ est $E\sigma$-compatible et $\mathbf{abst}(E, [\![C_i \ : \mathcal{X}_i]\!]) = \mathbf{abst}(E\sigma, [\![[\![C_i \ : \mathcal{X}_i]\!]\sigma \ :)]\!]$ (d'après le lemme 12.2.1).

Par le théorème 12.2.1, on en déduit que $P[t]_{\overline{p}}$ est une conséquence logique de $\{[\![C_i \ : \mathcal{X}_i]\!]\sigma\}$ pour tout $t \in \tau_G(s)$. D'où $\mathcal{M} \models C\sigma$. C.Q.F.D.

D'un point de vue pratique, il est clair que la règle G-**Induction** est trop générale pour pouvoir être utilisée de façon purement automatique. Elle nécessite l'intervention de l'utilisateur pour le choix de l'ensemble de contextes E. Cependant, nous pouvons imposer des conditions supplémentaires afin de restreindre l'application de la règle, et de pouvoir l'appliquer automatiquement de façon efficace. Nous pouvons, par exemple, restreindre l'application de la règle à des ensembles unitaires de contextes. Ainsi, les littéraux et les positions de l'ensemble E peuvent facilement être calculés en utilisant un algorithme d'anti-unification (PLOTKIN, 1970), mais la portée de la règle est alors plus limitée, puisque nous ne pouvons introduire qu'un seul symbole de sorte simultanément. Il est également possible de restreindre le nombre de c-clauses ou de c-littéraux de l'automate. Une autre possibilité (implicitement utilisée dans (WEIDENBACH, 1993), par exemple, voir section 12.4) est de n'autoriser que des prédicats monadiques (ou essentiellement monadiques, voir (FERMÜLLER ET AL., 1993)) dans l'ensemble de contextes E. Le calcul de E et la compilation des c-clauses en automate sont alors immédiats.

EXEMPLE 12.2.2. Considérons l'ensemble de c-clauses suivant.

$$C_1 \ [\![\neg P(x, y) \vee P(f(x), y) \ : \top]\!]$$
$$C_2 \ [\![P(a, y) \vee R(u) \ : \top]\!]$$

C_1 et C_2 sont $\{(P(x, y), \{1\}, s)\}$-compatibles.
Par la règle G-**Induction**, on déduit $[\![P(v, x) \vee R(u) \ : v \in s]\!]$
où v est une nouvelle variable et s est défini par :

$$\rightarrow s(a)$$
$$s(x) \rightarrow s(f(x))$$

$\diamond$

REMARQUE. La règle G-**Induction** est dans un certain sens similaire à la règle I-**Induction** décrite au chapitre 11 (voir aussi (SALZER, 1992; PELTIER, 1997a)), mis à part que le formalisme des automates est utilisé à la place des schématisations de termes. Ces deux règles sont *incomparables* au sens où il existe des ensembles de c-clauses qui peuvent être traitées par la règle G-**Induction** et qui ne peuvent pas être résolues par la règle I-**Induction** (et réciproquement). Cela tient évidemment aux limites du pouvoir d'expression respectif de ces deux formalismes.

Considérons par exemple l'ensemble suivant.

$$1\ [\![\neg P(x) \vee P(f(y,x)) \ : \top]\!]$$
$$2\ [\![P(a) \ : \top]\!]$$

De 1 et 2 nous pouvons inférer $P(f(y_1,a)), P(f(y_2,f(y_1,a))) \ldots$.

Cet ensemble de c-clauses *ne peut pas* être exprimé par des schématisations de termes car à chaque étape y peut être instancié par des termes différents. En revanche, notre méthode génère l'automate suivant.

$$\rightarrow S(a)$$
$$S(x) \rightarrow S(y,x)$$

Règles de transformation en automate d'arbres

Les règles que nous proposons dans cette section sont motivées par l'étude de l'exemple suivant.

EXEMPLE 12.2.3. Soit $E = \{(P(x), \{1\}, s)\}$ Soit l'ensemble S suivant.

$$[\![\neg P(x) \vee P(f(x,y)) \ : y \in s']\!]$$
$$[\![P(a) \ : \top]\!]$$

On a $\mathbf{abst}(E,S) = S$. Ici la règle G-**Induction** n'est pas applicable à cause de la condition $y \in s'$ apparaissant dans les contraintes de S. S n'est donc pas un automate d'arbres (voir définition 12.1.1). Pourtant, il est clair que nous pouvons transformer l'ensemble S en un automate d'arbres en transférant la condition $y \in s'$ dans la partie clause de $[\![\neg P(x) \vee P(f(x,y)) \ : y \in s']\!]$. Nous obtenons ainsi les c-clauses suivantes :

$$[\![\neg s'(y) \vee \neg P(x) \vee P(f(x,y)) \ : \top]\!]$$
$$[\![P(a) \ : \top]\!]$$

qui définissent un automate d'arbres. $\diamond$

Plus généralement, les règles de la figure 12.1 permettent de transformer certains ensembles de c-clauses S en un automate d'arbres équivalent, ce qui permet ensuite d'appliquer la règle G-**Induction**. Les règles visent à éliminer les contraintes supplémentaires des c-clauses afin de les transférer dans la partie clause de la règle. Certaines de ces règles sont déjà utilisées dans (COMON ET DELOR, 1994) pour simplifier les formules équationnelles avec contraintes d'appartenance.

THÉORÈME 12.2.3. *Le système $\mathcal{T}_{aut}$ est correct.*

$$\frac{(G, S \cup \{[\![C \ : x \in e \wedge \mathcal{X}]\!]\})}{(G \cup G', [\![\neg s(x) \vee C \ : \mathcal{X}]\!])}$$

Si e est une expression de sorte et si s est un nouveau symbole de sorte et G' une extension de G telle que $\tau_G(e) = \tau_{G'}(s)$ (G' et s existent d'après la condition C_3).

$$\frac{[\![C \ : \mathcal{X}[\exists x_1, \ldots, x_n.(x = f(x_1, \ldots, x_n) \wedge \bigwedge_{i=1}^{n} x_i \in s_i)]_p]\!]}{[\![C \ : \mathcal{X}[x \in f(s_1, \ldots, s_n)]_p]\!]}$$

$$\frac{[\![C \ : \mathcal{X}[x \in s \wedge x \in s']_p]\!]}{[\![C \ : \mathcal{X}[x \in s \cap s']_p]\!]}$$

$$\frac{[\![C \ : \mathcal{X}[x \in s \vee x \in s']_p]\!]}{[\![C \ : \mathcal{X}[x \in s \cup s']_p]\!]}$$

FIG. 12.1 - : *Le système $\mathcal{T}_{aut}$*

PREUVE. Immédiate. C.Q.F.D.

EXEMPLE 12.2.4. Considérons l'ensemble de c-clauses suivant.
$[\![P(a) \ : \top]\!]$
$[\![\neg P(x) \vee P(f(x)) \vee R(y) \ : y \neq a]\!]$
$[\![\neg P(x) \vee P(g(x,y)) \ : \exists z.y = f(z) \vee \exists z, z'.y = g(z,z')]\!]$

On déduit :
$[\![P(x) \vee R(y) \ : x \in S \wedge y \neq a]\!]$
où

- S est défini par les règles de production suivantes.
  - $\rightarrow S(a)$
  - $S(x) \rightarrow S(f(x))$
  - $S(x) \wedge S'(y) \rightarrow S(g(x,y))$
  - $\rightarrow S'(f(y))$
  - $\rightarrow S'(g(y,z))$

◇

Instantiation des c-clauses

Dans certains cas, il est possible que la règle G-**Induction** ne soit pas applicable sur les c-clauses considérées, mais sur des instances des c-clauses. C'est le cas dans l'exemple suivant.

$[\![P(a,b) \ : \top]\!]$
$[\![P(x,y) \rightarrow P(z, f(y)) \ : x \neq b]\!]$

Ici, nous devons appliquer la règle G-**Induction** sur la clause

$$[\![P(a,y) \rightarrow P(a, f(y)) \ : \top]\!]$$

qui est une instance de $[\![P(x,y) \to P(z,f(y)) \ : x \neq b]\!]$.

De façon similaire, considérons l'ensemble de c-clauses suivant.

$$C : [\![\neg P(x,y) \vee P(f(x,z),y) \vee R(z) \ : \top]\!]$$
$$D : [\![P(a,y) \ : \top]\!]$$

z apparaît dans le littéral $R(z)$, donc la règle G-**Induction** n'est pas applicable. Néanmoins, nous pouvons appliquer la règle sur la c-clause $[\![\neg P(x,y) \vee P(f(x,a),y) \vee R(a) \ : \top]\!]$, qui est une instance de C.

12.3 Exemples

Les exemples suivants montrent l'intérêt de la règle proposée.

EXEMPLE 12.3.1. Considérons l'exemple suivant tiré de la bibliothèque de formules TPTP, problème numéro SYN324 (SUTTNER ET SUTCLIFFE, 1995).

$$1\ [\![P(X,g(X)) \vee \neg P(X,X) \ : \top]\!]$$
$$2\ [\![\neg P(X,g(X)) \vee \neg P(X,X) \ : \top]\!]$$
$$3\ [\![P(X,g(X)) \vee P(g(X),g(X)) \ : \top]\!]$$
$$4\ [\![\neg P(X,g(X)) \vee \neg P(g(X),g(X)) \ : \top]\!]$$

Notre méthode génère les c-clauses suivantes.

$$5\ [\![\neg P(X,X) \vee \neg P(g(X),g(X)) \ : \top]\!] \quad \text{(resolution,1,4)}$$
$$6\ [\![\neg P(X,X) \vee \neg P(g(X),g(g(X))) \ : \top]\!] \quad \text{(resolution,5,1)}$$
$$7\ [\![\neg P(X,X) \vee P(g(g(X)),g(g(X))) \ : \top]\!] \quad \text{(resolution,6,3)}$$
$$8\ [\![\neg P(a,a) \ : \top]\!] \quad \text{(GPL,2)}$$
$$9\ [\![\neg P(X,g(X)) \vee \neg P(X,X) \ : x \neq a]\!] \quad \text{(dissubsumption,6,2)}$$
$$10\ [\![P(X,g(X)) \vee P(g(X),g(X)) \ : x = a]\!] \quad \text{(disresolution,7,3)}$$

A ce point, nous appliquons la règle G-**Induction** sur les c-clauses 7,10. Nous obtenons la c-clause

$$9\ [\![P(X,X) \vee \neg P(a,g(a)) \ : x \in S]\!]\ (G\text{-}\mathbf{Induction},7,10)$$

où S est défini par

$$\to S(g(a))$$
$$S(x) \to S(g(g(x)))$$

Ensuite, en appliquant la règle GMPL sur la c-clause 9, on obtient les c-clauses suivantes.

$$10\ [\![P(x,x) \ : x \in S]\!]$$
$$11\ [\![\neg P(g(x),g(x)) \ : x \in S]\!]$$
$$12\ [\![P(x,g(x)) \ : x \in S]\!]$$
$$13\ [\![\neg P(g(x),x) \ : x \in S]\!]$$

Les c-clauses $10, 11, 12, 13$ donnent un modèle de la formule initiale (les autres c-clauses peuvent être éliminées par application des règles de simplification).

Il est intéressant de mentionner que la résolution (sans stratégie de restriction), ainsi que l'hyper-résolution positive ou négative ne sont pas capables de détecter la satisfaisabilité de cette formule même si des règles de simplification telles que la subsumption sont utilisées. Ces règles génèrent en effet un ensemble infini de c-clauses. ◇

EXEMPLE 12.3.2. Les remarques 1 et 2 ci-dessous illustrent l'intérêt de l'exemple suivant.

1. La formule n'admet pas de modèle fini. En effet, l'interprétation du prédicat $R(x,y,z)$ correspond à la définition de l'égalité sur l'univers de Herbrand entre y et z à la définition de l'égalité entre y et z sur l'univers de Herbrand (si $x = f^{2n}(a)$).
2. Il n'admet pas de eq-modèle. En effet, les deux premières c-clauses impliquent $R(x,y,z) - \neg R(f(x),y,z)$.

$$
\begin{array}{l}
1\ [\![R(x,y,z) \vee \neg R(f(x),y,z)\ : \top]\!] \\
2\ [\![\neg R(x,y,z) \vee R(f(x),y,z)\ : \top]\!] \\
3\ [\![R(a,x,y) \vee \neg R(a,f(x),f(y))\ : \top]\!] \\
4\ [\![\neg R(a,a,f(x))\ : \top]\!] \\
5\ [\![R(a,x,x)\ : \top]\!]
\end{array}
$$

Notre méthode donne.

$$
\begin{array}{lll}
6\ [\![\neg R(a,x,y)\ : x \neq y]\!] & & (\text{GMPL},3) \\
7\ [\![R(u,x,y) \vee \neg R(f(f(u)),x,y)\ : \top]\!] & & (\text{résolution, } 1,2) \\
8\ [\![R(u,x,y)\ : x \neq y \wedge u \in s]\!] & & (G\textbf{-Induction}, 6,7)
\end{array}
$$

Où s est défini par les règles suivantes.

$$
\begin{array}{c}
\rightarrow s(a) \\
s(x) \rightarrow s(f(f(x)))
\end{array}
$$

De la même façon, on obtient:

$$
\begin{array}{lll}
9\ \ [\![R(v,x,x)\ : v \in s]\!] & & \\
10\ [\![\neg R(f(v),x,x)\ : v \in s]\!] & & (\text{résolution},1,9) \\
11\ [\![R(f(v),x,y)\ : v \in s \wedge x \neq y]\!] & & (\text{résolution},2,8)
\end{array}
$$

Les c-clauses $8, 9, 10, 11$ définissent un modèle de l'ensemble de c-clauses initial. ◇

12.4 Comparaison avec des travaux existants

12.4.1 Les travaux de Matzinger

Dans (MATZINGER, 1997), des grammaires régulières d'arbres sont utilisées pour représenter des modèles de Herbrand. Une méthode est proposée pour résoudre le problème d'équivalence. Les techniques utilisées sont fondées sur l'utilisation de stratégies d'ordonnancement pour la méthode de résolution et sont complétement différentes de celles utilisées dans le présent chapitre, fondées sur l'utilisation de *contraintes*. Le pouvoir d'expression du formalisme est exactement équivalent aux modèles finis, ce qui n'est pas le cas pour notre approche. La méthode de représentation proposée dans ce chapitre est donc plus générale.

12.4.2 Les travaux de Weidenbach

Dans un domaine différent de celui de la construction de modèles, (WEIDENBACH, 1993) propose une méthode permettant d'étendre la logique du premier ordre avec des sortes dynamiques. La méthode est fondée sur l'utilisation de l'unification avec sortes et sur une modification des règles de résolution et de factorisation, ainsi que de la définition des déclarations de sortes.

L'algorithme de (WEIDENBACH, 1993) pour transformer des ensembles de clauses monadiques en déclaration de sortes est comparable à la règle G-**Induction**, qui présente cependant les

avantages suivants. D'une part, elle utilise des contraintes afin d'exprimer les informations sur les sortes, ce qui évite l'utilisation de littéraux résiduels (c'est-à-dire de littéraux ajoutés à la clause obtenue par résolution pour exprimer des contraintes sur les sortes des variables). D'autre part, notre méthode n'est pas restreinte à des prédicats unaires.

Par ailleurs, nous n'utilisons pas de déclaration de sortes conditionnelles. L'interprétation des symboles de sortes que nous introduisons est fixée lors de leur introduction et ne change pas au cours de la recherche. Cela représente une différence essentielle avec (WEIDENBACH, 1993) où les symboles de sorte sont fixées au début de la recherche (c'est l'ensemble des prédicats monadiques de la signature), mais où l'*interprétation* de ces symboles est constamment modifiée.

Remarquons cependant que des déclarations conditionnelles pourraient facilement être introduites en autorisant la présence de contraintes d'appartenance dans les automates d'arbres généraux. Cela permet ensuite de généraliser la règle G-**Induction** en transférant une partie des littéraux dans les contraintes. Par exemple la clause $Man(peter) \vee Woman(peter)$ (WEIDENBACH, 1993) pourrait être transformée en déclaration de sorte : $[\![Man(peter) \ : peter \notin Woman]\!]$. La résolution des contraintes n'est alors plus décidable. Il serait malgré tout possible de formuler des règles permettant (par exemple) de transférer les parties contraintes dans le corps des c-clauses (ce qui permet de préserver la complétude réfutationnelle).

Certains de ces problèmes sont actuellement étudiés par Michaël Dierkes dans son travail de DEA (DIERKES, 1997).

Partie IV

Nouvelles approches et applications

Chapitre 13

Recherche de réfutations et de modèles par analogie

L'analogie intervient dans beaucoup d'activités intellectuelles. Elle constitue une puissante heuristique pour la découverte de nouvelles notions (par exemple en Mathématiques) et une méthode *informelle* très fréquemment utilisée pour découvrir des preuves. En démonstration formelle, elle est essentiellement évoquée comme un exemple de raisonnement fallacieux : de "la somme de deux nombres pairs est paire" on peut "déduire" par analogie "la somme de deux nombres impairs est impaire". Comme nous nous intéressons à des raisonnements formels, l'analogie sera une façon de guider la preuve (réfutation) d'une nouvelle conjecture par des preuves de théorèmes connus.

En Déduction Automatique, la mécanisation de cette démarche apparaît comme une nécessité pour améliorer les capacités des démonstrateurs. En effet, un démonstrateur automatique est souvent amené à re-démontrer plusieurs fois des formules presque identiques. En l'absence de mécanismes permettant de reconnaître et de tirer parti de telles situations, les performances du système sont alors limitées de façon importante (Bledsoe, 1977). La puissance d'un système de démonstration automatique dépend dans une large mesure de sa capacité à détecter les formules redondantes et à éviter les calculs inutiles. Mais l'analogie ne se limite pas à un simple moyen d'améliorer la puissance des démonstrateurs. La découverte d'analogies, de similarités, entre des formules ou entre leurs preuves peut avoir un grand intérêt intrinsèque (notamment en Mathématiques ou en Programmation).

Les travaux portant sur la reconnaissance et l'utilisation de l'analogie en Démonstration Automatique restent cependant très peu nombreux. La nature très informelle de la notion d'*analogie* la rend difficile à modéliser et par là même à reproduire sur ordinateur. Pour une synthèse des travaux dans ce domaine, voir (Bourely et al., 1996; Défourneaux, 1997; Hall, 1989) (on pourra également consulter (Kling, 1971; Munyer, 1981; Plaisted, 1981; Boy de la Tour et Caferra, 1987; Boy de la Tour et Kreitz, 1992; Melis, 1995; Kolbe et Walther, 1995)). En particulier, il semble extrêmement difficile de donner une définition précise de la notion d'"analogie" entre deux formules. La plupart des travaux se limitent souvent à une comparaison *syntaxique* entre les deux énoncés (en établissant par exemple un *graphe de correspondance* entre les deux formules (Kling, 1971; Munyer, 1981)). Il est cependant clair que l'analogie telle qu'elle est utilisée par l'homme porte plutôt sur des critères *sémantiques* que syntaxiques.

En outre, la totalité des travaux dans ce domaine s'intéresse à la recherche de la preuve d'un théorème. Il n'existe pas, à notre connaissance, de travaux cherchant à généraliser l'utilisation de l'analogie à la construction de contre-exemple. Pourtant, le raisonnement par analogie permettrait, dans bien des cas, de guider la recherche du contre-exemple de la même façon qu'il facilite l'obtention d'une preuve.

Nous décrirons tout d'abord une méthode de recherche simultanée de réfutation et de modèle utilisant l'analogie. Selon Peirce (Hartshorne et al.,), l'analogie peut être vue comme la

combinaison de trois modes de raisonnement : une étape d'*induction*, puis d'*abduction*, suivie d'une *déduction*. Notre approche reproduit exactement cette conception. Elle est fondée sur des techniques de *généralisation*: la formule initiale $\mathcal{F}$ (qui peut être valide *ou non valide*), dont la preuve ou le contre-exemple est connu, est transformée en un nouvel énoncé $\mathcal{F}'$ plus général (représentant en fait un ensemble de dérivations) qui est stocké dans une *base de formules*. Lors de la présentation d'une nouvelle conjecture, le système recherche dans la base de connaissances une formule plus générale que la formule proposée et reconstruit éventuellement la preuve ou le contre-exemple.

Dans ce chapitre, est tout d'abord décrite une méthode de généralisation des formules dans le cadre de la recherche simultanée de réfutation et de modèle. La méthode de généralisation que nous proposons est guidée par l'étude de la *dérivation* permettant d'obtenir la réfutation ou le modèle de la formule de départ (c'est-à-dire la séquence d'application des règles d'inférence *et de dis-inférence*). Nous proposons ensuite un algorithme de filtrage permettant d'obtenir la preuve (contre-exemple) d'une conjecture à partir d'une formule analogue de la base de connaissances. Enfin, cet algorithme est étendu pour prendre en compte le cas (important en pratique) où l'analogie échoue : la méthode génère alors des *lemmes* (ou des contre-exemples partiels) qui doivent ensuite, soit être prouvés pour reconstruire la preuve ou pour compléter le contre-exemple, soit être infirmés (ce qui permet de détecter des "mauvaises" analogies).

Contrairement à d'autres approches, nous cherchons à étendre autant que possible la notion d'analogie, sans nous limiter à une identité syntaxique entre les formules. Dans la plupart des cas, la conjecture T ne pourra directement être prouvée à partir de la formule-source S. L'analogie implique en effet une notion de *distance* entre les formules : l'algorithme doit être capable de traiter également les cas où l'insatisfaisabilité ou le modèle de la formule-cible ne peut *pas* être directement déduit(e) de celle/celui de la formule-source. Cela correspond à un processus d'abduction consistant à chercher quelles seraient les hypothèses susceptibles de "faire marcher" l'analogie et à tenter (éventuellement) de prouver (ou d'infirmer) ces hypothèses.

Les problèmes liés à la construction de la base de connaissances sortent naturellement du cadre de cette thèse (voir (Défourneaux, 1997)) [1]. L'algorithme est présenté dans le cadre de la méthode Ramc. Néanmoins, notre approche peut s'étendre très facilement à d'autres procédures (telles que par exemple la méthode Ramcet).

13.1 Formalisme de représentation

Avant de s'intéresser au processus de généralisation, il convient tout d'abord de préciser la méthode utilisée pour représenter les ensembles de dérivations. Nous proposons de représenter cet ensemble de dérivations par une formule de la logique d'ordre supérieur. Celle-ci représente l'ensemble de dérivations du premier ordre obtenu en instanciant la formule d'ordre supérieur. Il contient évidemment la dérivation initiale. Avant de décrire formellement l'algorithme de généralisation, décrivons précisément le formalisme de représentation utilisé.

Notions de logique d'ordre supérieur

Rappelons brièvement les notions de base de la logique d'ordre supérieur (nous nous limiterons ici à une présentation succincte des notions les plus élémentaires, pour plus de détails, voir, par exemple, (Huet, 1975)).

L'ensemble des types d'ordre supérieur $\mathcal{S}_{ho}$ est défini par :

- $\mathcal{S}_{ho}$ contient l'ensemble des types de base $\mathcal{S}$.
- Si $\forall i \in [1..n].t_i \in \mathcal{S}_{ho}$ et $s \in \mathcal{S}_{ho}$, alors : $t_1 \times \ldots \times t_n \rightarrow s \in \mathcal{S}_{ho}$.

1. Les résultats décrits dans ce chapitre sont le fruit d'un travail commun avec Christophe Bourely et Gilles Défourneaux.

Soit $(\mathcal{V})_{t\in\mathcal{S}_{ho}}$ une famille d'ensembles disjoints, infinis et dénombrables de variables (appelées *variables de type* t). Soit $\mathcal{V} = \bigcup_{t\in\mathcal{T}} \mathcal{V}_t$. Soit $(\Sigma)_{t\in\mathcal{S}_{ho}}$ une famille d'ensembles disjoints de constantes (de type t). On suppose que $\mathcal{V} \cap \Sigma = \emptyset$.

Les ensembles des termes $\tau_t(\Sigma, \mathcal{V})$ de type t sont les plus petits ensembles vérifiant les conditions suivantes :

- $\mathcal{V}_t \subseteq \tau_t(\Sigma, \mathcal{V})$.
- $f : s_1 \times \ldots \times s_n \to s \in \tau_{s_1\times\ldots\times s_n\to s}(\Sigma, \mathcal{V})$.
- Si $t \in \tau_{s_1\times\ldots\times s_n\to s}(\Sigma, \mathcal{V})$ et $\forall i \leq n. u_i \in \tau_{s_i}(\Sigma, \mathcal{V})$ alors $t(u_1, \ldots, u_n) \in \tau_s(\Sigma, \mathcal{V})$.
- Si $t \in \tau_s(\Sigma, \mathcal{V})$ et $\forall i \leq n. x_i \in \mathcal{V}_{s_i}$ alors $\lambda x_1, \ldots, x_n.t \in \tau_{s_1\times\ldots\times s_n\to s}(\Sigma, \mathcal{V})$.

On note $\tau(\Sigma, \mathcal{V}) = \bigcup_{t\in\mathcal{T}} \tau_t(\Sigma, \mathcal{V})$, l'ensemble des termes d'ordre supérieur. On peut étendre de façon immédiate la définition des formules et formules équationelles à l'ordre supérieur. Les notions de substitution, de solution d'une formule équationnelle (modulo la β-réduction) etc. se généralisent également de façon triviale aux termes et formules d'ordre supérieur. Par souci de concision, nous ne donnons pas ici toutes les définitions (voir par exemple (Lugiez, 1995)). De façon évidente, toute formule du premier ordre est un cas particulier de formule d'ordre supérieur.

Définition 13.1.1. *Une* formule généralisée *de $\mathcal{F}$ est une formule vérifiant les conditions suivantes :*

- *Toute variable liée de $\mathcal{F}$ est d'un des types de base.*
- *Toute variable libre X de $\mathcal{F}$ est de type $s_1 \times \ldots \times s_n \to s$, où $s_1, \ldots, s_n, s$ sont des types de base et toute occurrence de X apparaît dans un terme de la forme $X(t_1, \ldots, t_n)$.*

◇

Définition 13.1.2. *Une* c-clause généralisée *est de la forme*

$$\forall \overline{x} [\![C \ : \mathcal{X}]\!]$$

telle que $\mathcal{X}$ est une formule équationnelle généralisée, C une clause généralisée et $\overline{x}$ un n-uplet (éventuellement vide) de variables de type de base. ◇

Notation 13.1.1 *Dans la suite, et par souci de lisibilité, les variables libres d'une c-clause généralisée seront notées en majuscule ($A, B, \ldots$ si la variable est d'arité 0, $F, G, \ldots$ sinon) alors que les variables liées seront notées $x, y, \ldots$ et les symboles de fonctions $f, g, \ldots$ ou $a, b, \ldots$.*

Définition 13.1.3. *Un* ensemble contraint *est un couple $(S, \mathcal{X})$ où S est un ensemble de c-clauses généralisées et $\mathcal{X}$ une formule équationnelle généralisée. Une* séquence contrainte *est un couple $(\mathcal{S}, \mathcal{X})$ où $\mathcal{S}$ est une séquence d'ensembles de c-clauses généralisées et $\mathcal{X}$ une formule équationnelle généralisée.* ◇

Remarque. Soit $\mathcal{F}$ un ensemble de c-clauses généralisées. Pour toute substitution fermée σ de support $\mathcal{V}ar(\mathcal{F})$, $\mathcal{F}\sigma$ est un ensemble de c-clauses.

Nous étendons la notion de Ramc-dérivation aux c-clauses généralisées de la façon suivante.

Définition 13.1.4. *Une* séquence contrainte $(t, \mathcal{X})$ *est appelée une* Ramc-dérivation *si et seulement si pour toute substitution fermée σ solution de $\mathcal{X}$, $t\sigma$ est une* Ramc-dérivation *(au sens habituel).* ◇

Tous les symboles apparaissant dans la dérivation seront supposés indexés par leur première occurrence dans la dérivation.

13.2 Un ordre sur les formules

L'idée de la méthode est de généraliser l'ensemble de clauses en préservant soit l'insatisfaisabilité et la réfutation, soit la satisfaisabilité et le modèle. Mais que signifie exactement *généraliser*? Avant de présenter l'algorithme de généralisation, nous devons clarifier cette notion et en donner une définition formelle.

Nous nous plaçons dans le cadre de preuve par réfutation, donc nous parlerons d'insatisfaisabilité pour l'ensemble de c-clauses d'un théorème et de satisfaisabilité pour l'ensemble de c-clauses d'un non-théorème. Informellement, un théorème T sera dit "plus général" qu'un théorème T' si les hypothèses de T sont *moins fortes* que celles de T' ou si la conclusion de T est *plus forte* que celle de T'. En effet, il est alors intuitivement évident que le théorème T apporte "plus d'informations" que T'. Dans le cas d'ensemble de c-clauses, l'ensemble des hypothèses est l'ensemble de c-clauses lui-même et la conclusion est $\square$. Donc, T sera plus général que T' si et seulement si toutes les clauses de T appartiennent à T', c'est-à-dire si et seulement si $T \subseteq T'$. Ainsi, si S est un ensemble insatisfaisable, S sera *plus général* que tout ensemble de c-clauses S' contenant S. En effet S' n'est pas minimalement insatisfaisable : il contient des hypothèses qui sont inutiles pour la preuve. Cette notion très simple peut être complétée par les remarques suivantes.

1. D'une part, l'important ici n'est pas la définition *syntaxique* de l'ensemble de c-clauses mais plutôt sa *sémantique*, c'est-à-dire l'ensemble de clauses fermées dénotées par cet ensemble. En effet, il est possible qu'une c-clause C ne soit pas inclue dans un ensemble S', mais que toute instance fermée de C appartienne à l'ensemble des instances fermées de S'. La comparaison entre les ensembles de c-clauses doit donc être réalisée modulo la relation d'équivalence $\cong$ (voir chapitre 6).

2. D'autre part, il est également possible qu'une clause fermée de S n'appartienne pas à S' mais soit *subsumée* par une c-clause de S'.

Afin de tenir compte de ces deux remarques, nous allons introduire la relation suivante entre ensembles de c-clauses.

DÉFINITION 13.2.1. *Soit S, S' deux ensembles de c-clauses. On note $S \sqsubseteq S'$ si et seulement si pour toute instance fermée C d'une c-clause de S, il existe une instance fermée C' d'une clause de S' telle que $C' \subseteq C$.* ◇

LEMME 13.2.1. *Si $S \sqsubseteq S'$ alors S est une conséquence logique de S'. D'autre part, si d est une réfutation fermée (utilisant les règles de résolution et factorisation) de S, alors il est possible de construire automatiquement une réfutation fermée de S'.*

PREUVE.
- Soit $\mathcal{I}$ un modèle de S'. Par définition pour toute clause fermée C de S' $\mathcal{I} \models C$. Soit $D \in S$. On a $S \sqsubseteq S'$, donc il existe $C \in S'$ telle que $C \subseteq D$. $\mathcal{I} \models C$ (car $C \in S'$) donc $\mathcal{I} \models D$. D'où $\mathcal{I} \models S$.
- Soit d une réfutation fermée de S. Soit n la longueur de la dérivation d. Montrons, par induction sur n, qu'il est possible de construire une réfutation fermée de S.
  - Si $n = 0$, alors $\square \in S$, donc $\square \in S'$. D'où (S') est une réfutation de S'.
  - Si $n > 0$, alors, notons $d = (S_1, \ldots, S_n)$. S_2 est déduit de S_1 par application soit de la règle de résolution, soit de la règle de factorisation. Considérons successivement ces deux possibilités.
    1. **Résolution.** Dans ce cas, il existe deux clauses $P \vee R_1$ et $\neg P \vee R_2$ dans S_1 et $S_2 = S_1 \cup \{R_1 \vee R_2\}$. Alors, par définition de S' il existe deux clauses C_1 et C_2 dans S' subsumant respectivement $P \vee R_1$ et $\neg P \vee R_2$. Si C_1 subsume R_1, alors C_1 subsume

$R_1 \vee R_2$ donc $S_2 \sqsubseteq S'$. Puisque S_2 admet une réfutation de longueur $n-1$, par hypothèse d'induction, on peut construire une réfutation fermée de S'. La preuve est identique si C_2 subsume R_2.
Sinon, C_1 et C_2 sont respectivement de la forme (éventuellement après application d'une règle de factorisation) $P \vee R'_1$ et $\neg P \vee R'_2$, où R'_i subsume R_i $(i = 1, 2)$. En appliquant la résolution, on obtient l'ensemble $S'' = S' \cup \{R'_1 \vee R'_2\}$. On a de façon évidente $S_2 \sqsubseteq S''$. Par hypothèse d'induction, on peut construire une réfutation fermée de S'', donc de S'.

2. **Factorisation**. Il existe une clause $P \vee P \vee Q$ dans S_1 et $S_2 = S_1 \cup \{P \vee R\}$. Par définition, on a $P \vee P \vee R \leq_{dissub} P \vee R$, donc $S_2 \sqsubseteq S'$ d'où il est possible de construire une réfutation fermée de S'.

C.Q.F.D.

En revanche, dans le cas satisfaisable, T sera dit *plus général* que T' si les hypothèses de T sont plus fortes que celles de T', c'est-à-dire si T est faux dans un plus grand nombre d'interprétations que T'. Donc un ensemble satisfaisable T sera dit plus général qu'un ensemble satisfaisable T' si et seulement si $T' \sqsubseteq T$.

Ces deux notions de généralisation ne sont donc pas équivalentes. La définition suivante formalise cette idée.

DÉFINITION 13.2.2. *Soit $T_1 : (t_1, \mathcal{X}_1)$ et $T_2 : (t_2, \mathcal{X}_2)$ deux ensembles contraints. T_1 est dit plus général que T_2 (noté $T_1 \succeq_{gen} T_2$) si et seulement si pour toute solution σ_2 de $\mathcal{X}_2$ il existe une solution σ_1 de $\mathcal{X}_1$ telle que :*

- *t_1 est insatisfaisable et $t_1\sigma_1 \sqsubseteq t_2\sigma_2$*
- *t_1 est satisfaisable et $t_2\sigma_2 \sqsubseteq t_1\sigma_1$.*

◇

PROPOSITION 13.2.1. *$\succeq_{gen}$ est un pré-ordre.*

EXEMPLE 13.2.1. [Ordre de généralisation, cas insatisfaisable] Soit $t_1 : \forall x.P(x) \wedge \neg P(a)$ et $t_2 : P(a) \wedge \neg P(a)$. t_1 et t_2 sont deux ensembles de clauses insatisfaisables. L'ensemble des clauses fermées dénoté par t_1 est $\{P(t)/t \in \tau(\Sigma)\} \cup \{\neg P(a)\}$. L'ensemble des clauses fermées dénoté par t_2 est $\{P(a), \neg P(a)\}$. On a donc $t_2 \subseteq t_1$, d'où t_2 est plus général que t_1. En effet t_2 contient "plus d'information" que t_1, puisque les hypothèses de t_2 sont moins contraignantes que celles de t_1. ◇

EXEMPLE 13.2.2. [Ordre de généralisation, cas satisfaisable]
Soit $t_1 : \forall x [\![P(x) : x \neq a]\!] \wedge \neg P(a)$ et $t_2 : P(b) \wedge \neg P(a)$. t_1 et t_2 sont satisfaisables. On a $t_2 \subseteq t_1$, d'où t_2 est moins général que t_1. En effet t_1 est falsifié par plus d'interprétations que t_2 (tout modèle de t_1 est un modèle de t_2). ◇

13.3 Algorithme de généralisation

Nous présentons maintenant un algorithme permettant de transformer une RAMC-dérivation S en une RAMC-dérivation S' plus générale que S au sens du pré-ordre $\succeq_{gen}$. Cet algorithme sera spécifié sous forme de règles de transformation.

Il y a plusieurs similarités syntaxiques qui peuvent faire que deux dérivations soient analogues. La seule façon de ne pas les perdre par une abstraction trop générale est de traiter séparément chacun de ces cas. Puisque l'ordre de généralisation n'est pas le même suivant que S est satisfaisable ou insatisfaisable, l'application de certaines des règles dépend de la satisfaisabilité de l'ensemble source. Par conséquent, nous proposons un ensemble de règles de généralisation divisé en trois parties.

1. Cas général. Ces règles s'appliquent dans tous les cas.

2. Règles d'affaiblissement. Ces règles ne s'appliquent que si l'ensemble considéré est insatisfaisable. Leur objet est d'affaiblir l'ensemble de c-clauses en *enlevant des hypothèses.*

3. Règles de renforcement. Ces règles s'appliquent uniquement si l'ensemble considéré est satisfaisable. Elles visent à *ajouter des hypothèses à l'ensemble de c-clauses.*

Nous donnons pour chaque règle sa définition formelle et un exemple d'application simple. Les règles opèrent sur des séquences contraintes.

13.3.1 Règles générales

Règle d'abstraction

L'objet de la première règle est de remplacer un symbole de fonction f par une variable F de même type que f. Par exemple, la dérivation $\forall x.P(f(x)) \wedge P(f(a)) \rightarrow \square$ est transformée en $\forall x.P(F(x)) \wedge P(F(a)) \rightarrow \square$ où F est une nouvelle variable libre de même type que f. A cause de la présence de négation dans les contraintes, cela ne suffit cependant pas à garantir que le terme obtenu soit une dérivation, comme le montre l'exemple suivant.

EXEMPLE 13.3.1. Considérons la dérivation suivante.

$$\begin{array}{ll} 1\ [\![p(a) \vee r(b)\ : \top]\!] & \\ 2\ \forall x.[\![\neg p(f(x))\ : \top]\!] & \\ 3\ [\![p(a)\ : \forall x.a \neq f(x)]\!] & (\text{GPL},1,2) \\ 4\ [\![p(a)\ : \top]\!] & (\text{Résolution de contraintes } 3) \end{array}$$

Ici il est impossible de remplacer a et f par de nouvelles variables libres. En effet, la dérivation contient la règle de décomposition $a \neq f(x) \rightarrow \top$ qui suppose que les symboles a et f sont distincts. ◇

La solution retenue pour résoudre ce problème consiste à ajouter à la dérivation des contraintes assurant que les deux variables sont associées à des symboles distincts. Ainsi la dérivation ci-dessus sera transformée en :

$$(S, \forall x.A \neq F(x))$$

où S est la dérivation :

$$\begin{array}{ll} 1\ [\![P(A) \vee R(B)\ : \top]\!] & \\ 2\ \forall x.[\![\neg P(F(x))\ : \top]\!] & \\ 3\ [\![P(A)\ : \forall x.A \neq F(x)]\!] & (\text{GPL},1,2) \\ 4\ [\![P(A)\ : \top]\!] & (\text{Résolution de contraintes } 3) \end{array}$$

Plus formellement, nous introduisons la règle :

Règle 13.3.1 (Abstraction) : $\dfrac{(S, \mathcal{X})}{(S\{f \rightarrow F\}, \mathcal{X}\{f \rightarrow F\} \wedge \bigwedge_{i=1}^{n} .\forall \overline{x}.(F(\overline{x}) \neq g_i(\overline{x})))}$

où :

- S *est une* RAMC*-dérivation.*
- $\{(f, g_i)/i = 1, \ldots, n\}$ *est l'ensemble des couples* (f, g) *où* g *est une occurrence de symbole de constante telle qu'il existe dans* S *une application d'une règle de décomposition de la forme :* $f(\overline{t}) = g(\overline{s}) \rightarrow \bot$.

Introduction de variables libres

La règle suivante cherche à généraliser la dérivation en remplaçant deux occurrences distinctes de variables par de nouvelles variables distinctes. Avant de définir formellement la règle, donnons un exemple illustrant son principe.

EXEMPLE 13.3.2. Soit la dérivation:

$$\begin{array}{ll} 1\ [\![P(A) \vee P(B)\ : \top]\!] & \\ 2\ [\![\neg P(A)\ : \top]\!] & \\ 3\ [\![\neg P(B)\ : \top]\!] & \\ 4\ [\![P(B)\ : \top]\!] & \text{(Résolution 1,2)} \\ 5\ \square & \text{(Résolution 3,4)} \end{array}$$

Il est facile de voir que les deux occurrences du symbole P dans la première c-clause peuvent être remplacées par deux variables distinctes P_1, P_2, à condition de remplacer également les c-clauses $2,3$ par $[\![\neg P_1(A)\ : \top]\!]$ et $[\![\neg P_2(B)\ : \top]\!]$. En effet, aucune règle de résolution ou de décomposition n'est appliquée sur ces deux occurrences de P, donc les deux occurrences "n'ont pas besoin" d'être égales. On obtient alors la dérivation suivante, qui est de façon évidente plus générale que la dérivation initiale.

$$\begin{array}{ll} 1\ [\![P_1(A) \vee P_2(B)\ : \top]\!] & \\ 2\ [\![\neg P_1(A)\ : \top]\!] & \\ 3\ [\![\neg P_2(B)\ : \top]\!] & \\ 4\ [\![P_2(B)\ : \top]\!] & \text{(Résolution 1,2)} \\ 5\ \square & \text{(Résolution 3,4)} \end{array}$$

Informellement, cette transformation est justifiée car la validité de la RAMC-dérivation *ne nécessite pas l'égalité entre les occurrences* 1 *et* 2 *du symbole* P. La définition suivante formalise cette notion intuitive. ◇

DÉFINITION 13.3.1. *Soit deux occurrences* c_1 *et* c_2 *de symboles de* $\mathcal{V}$. *On note* $c_1 \approx c_2$ *la fermeture symétrique, transitive et réflexive de la relation suivante: il existe dans la* RAMC*-dérivation* $S_1, \ldots, S_n$ *une application d'une règle de décomposition* $c_1(\overline{t}) = c_3(\overline{s}) \rightarrow \overline{s} = \overline{t}$ *ou* $c_1(\overline{t}) \neq c_3(\overline{s}) \rightarrow \overline{s} \neq \overline{t}$, *telle que soit* $c_3 = c_2$ *soit* c_3 *est introduit par la règle d'***Explosion**. ◇

$\approx$ est une relation d'équivalence donc induit une partition de l'ensemble des occurrences de c. Pour chaque occurrence c_i du symbole c nous notons $[c_i]_{\approx}$ la classe d'équivalence de c_i.

Règle 13.3.2 (Création de nouvelles variables libres) : $\dfrac{S}{S'}$

où :

- A *est une variable libre.*
- $n_0, \ldots, n_k$ *sont des entiers tels que* $[A^{n_1}]_{\approx}, \ldots, [A^{n_k}]_{\approx}$ *est une partition de l'ensemble des occurrences de* A.
- $B_1, \ldots, B_k$ *sont des nouvelles variables distinctes du même type que* A.
- $S' = S\{\forall i, 0 < i \leq k, \forall A^j \in [A^{n_i}]_{\approx}, A^j \leftarrow B_i\}$.

Elimination de symboles

Le rôle de cette règle est de réduire le nombre de variables apparaissant dans la formule, afin de réduire la complexité de l'algorithme de filtrage. Plus précisément, un terme de la forme $F(\ldots, G(t), \ldots)$ sera remplacé par le terme $F'(\ldots, t, \ldots)$ où F' est une nouvelle variable. Evidemment, cette transformation n'est autorisée que si l'élimination de ce symbole préserve la correction de la dérivation.

EXEMPLE 13.3.3. Par exemple :

$$\forall y.[\![P(F(y)) \ : \top]\!] \wedge [\![\neg P(F(a)) \vee \neg P(F(b)) \ : \top]\!] \rightarrow_{\mathrm{RAMC}} \square$$

est remplacé par :

$$\forall y.[\![P'(y) \ : \top]\!] \wedge [\![\neg P'(a) \vee \neg P'(b) \ : \top]\!] \rightarrow_{\mathrm{RAMC}} \square$$

◇

Plus formellement :

Règle 13.3.3 (Elimination de variable libre) : $\dfrac{S}{S'}$

Si :

- *il existe* $F \in \mathcal{V}$, $i \leq$ *arité*(F) *et* $g \in \mathcal{V}$, *tel que pour tout terme de la forme* $F(t_1 \ldots t_n)$, t_i *est de la forme* $g(\overline{t})$;
- F' *est une nouvelle variable.*
- S' *est obtenu à partir de* S *en remplaçant tous les termes de la forme* $F(t_1, \ldots, t_{i-1}, g(\overline{y}), t_{i+1}, \ldots, t_n)$ *par* $F'(t_1, \ldots, t_{i-1}, \overline{t}, t_{i+1}, \ldots, t_n)$.

Règle d'élimination de paramètres

Cette règle permet de simplifier la formule-source en supprimant certains de ses sous-termes et en réduisant l'arité de certaines fonctions. Ce type de règle joue un rôle important dans les travaux de PLAISTED (voir (PLAISTED, 1981)).

EXEMPLE 13.3.4. Par exemple, la dérivation

$$\forall x.[\![P(x,x) \ : \top]\!] \wedge \forall y.[\![\neg P(y,y) \ : \top]\!] \rightarrow \square$$

sera transformée en

$$\forall x.[\![P'(x) \ : \top]\!] \wedge \forall y.[\![\neg P'(y) \ : \top]\!] \rightarrow \square$$

◇

La transformation préserve la validité de la dérivation car les sous-termes aux positions 1 et 2 dans les termes de la forme $P(\_,\_)$ sont systématiquement identiques.

Règle 13.3.4 (Elimination de paramètres) : $\dfrac{S}{S'}$

- *S'il existe* $F \in \mathcal{V}$ $i, j \leq$ *arité*(F), *tels que pour tout terme* $F(t_1, \ldots, t_n)$ *de* S, $t_i = t_j$;
- F' *est une nouvelle variable libre.*
- S' *est obtenu à partir de* S *en remplaçant chaque terme* $F(t_1, \ldots, t_n)$ *par* $F'(t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_n)$.

13.3.2 Règles d'affaiblissement pour la réfutation

Instanciation

L'objet de la règle d'**Instanciation** est de remplacer une variable liée x par une variable libre, en préservant la correction de la dérivation.

EXEMPLE 13.3.5. Considérons par exemple la dérivation :

$$\begin{array}{ll}
1\ \forall x.[\![P(x) \ : \top]\!] & \\
2\ [\![\neg P(F(A)) \vee \neg P(F(B)) \ : \top]\!] & \\
3\ [\![\neg P(F(B)) \ : \exists x.x = F(A)]\!] & \text{(Résolution 1,2)} \\
4\ [\![\square \ : \exists x, x'.(x = F(A) \wedge x' = F(B))]\!] & \text{(Résolution 3,2)} \\
5\ \square & \text{(résolution de contraintes, 4)}
\end{array}$$

Ici, la variable liée x n'est en relation qu'avec les termes $F(A)$ et $F(B)$. On peut donc, en préservant la dérivation, remplacer cette variable par un terme : $F(y)$ où y est une nouvelle variable liée. On obtient ainsi la dérivation :

$$\begin{array}{ll}
1\ \forall y.[\![P(F(y)) \ : \top]\!] & \\
2\ [\![\neg P(F(A)) \vee \neg P(F(B)) \ : \top]\!] & \\
3\ [\![\neg P(F(B)) \ : \exists y.F(y) = F(A)]\!] & \text{(Résolution 1,2)} \\
4\ [\![\square \ : \exists y, y'.(F(y) = F(A) \wedge F(y') = F(B))]\!] & \text{(Résolution 3,2)} \\
5\ [\![\square \ : \exists y, y'.(y = A \wedge y' = B)]\!] & \text{(Décomposition,4)} \\
6\ \square & \text{(résolution de contraintes, 4)}
\end{array}$$

◇

DÉFINITION 13.3.2. *Soient t et s deux termes. On note : $\rightsquigarrow$ la fermeture transitive, réflexive et symétrique de la relation définie entre deux termes s et t de la façon suivante.*

- *s et t sont deux variables liées et s est introduite par renommage de t.*
- *t est une variable liée et il existe une application de la règle de **Remplacement**, **Universalité des paramètres** ou de **Fusion** sur $t = s$ ou $t \neq s$.*

◇

La règle d'**Instanciation** est définie de la façon suivante :

Règle 13.3.5 (Instanciation) : $\dfrac{S[\forall x.\mathcal{F}]_p}{S[\forall \overline{y}.\mathcal{F}\{x \leftarrow F(\overline{y})\}]_p}$

*Si S est insatisfaisable, x est une variable liée de t telle que la règle **Universalité des paramètres** n'est jamais appliquée sur x, $f \in \mathcal{V}$, et si l'une des conditions suivantes est valide :*

- *Il existe une variable libre F telle que : pour tout s tel que $x \rightsquigarrow s$, s est soit une variable, soit de la forme : $F(\overline{u})$. $\overline{y}$ sont de nouvelles variables.*
- *Il n'existe aucun terme s n'appartenant pas à $\mathcal{V}$ tel que $x \rightsquigarrow s$, et F est une nouvelle variable libre de même type que x. $\overline{y}$ est vide.*

Elimination de variables liées

La règle suivante joue un rôle symétrique à la règle de création de symboles. Son objet est d'éliminer des variables liées dans la formule (par exemple en remplaçant le terme : $\forall x, y.P(x, y)$ par $\forall x.P(x, x)$). Afin de garantir que la dérivation est préservée par application de la règle, il est nécessaire que les deux variables considérées ne soient jamais mises en relation avec deux termes distincts.

Plus précisément, la règle d'**Elimination de variables** est la suivante.

Règle 13.3.6 (Elimination de variables liées) : $\dfrac{S[\forall x, y.\mathcal{F}]_p}{S[\forall x.\mathcal{F}\{x \leftarrow y\}]_p}$

S est insatisfaisable, et pour tout t_1 et t_2 tels que $x \rightsquigarrow t_1$ et $y \rightsquigarrow t_2$ alors $t_1 = t_2$.

EXEMPLE 13.3.6. Par exemple la dérivation :

$$\forall x, y.[\![\neg P(x, y) \ : \top]\!] \wedge \forall z.[\![P(z, z) \ : \bot]\!] \rightarrow \square$$

sera transformée en

$$\forall x.[\![\neg P(x, x) \ : \top]\!] \wedge \forall z.[\![P(z, z) \ : \bot]\!] \rightarrow \square$$

◇

13.3.3 Règles de renforcement pour la construction de modèles

Elimination de termes inutiles

La règle d'élimination de termes inutiles est l'opposée de la règle d'instanciation. Elle ne s'applique que si l'ensemble est satisfaisable. Son objet est de remplacer un terme t par une nouvelle variable liée x, de façon à obtenir un ensemble de c-clauses plus général.

Règle 13.3.7 (Elimination de termes) : $\dfrac{S[\forall \overline{y}.[\![\mathcal{F}[t]_p] \ : \mathcal{X}]\!]_q}{S[\forall x, \overline{y}.[\![\mathcal{F}[x]_p] \ : \mathcal{X}]\!]_q}$

où:

- *S est satisfaisable.*
- *t est un terme tel qu'il n'existe aucune application des règles **Décomposition, Fusion, Remplacement, Universality of Parameters** sur t et x une nouvelle variable.*

13.3.4 Correction de l'algorithme de généralisation

Nous noterons $\mathcal{R}_{gen}$ le système de règles contenant les règles de la section 13.3.

THÉORÈME 13.3.1. *Soient S un ensemble de c-clauses généralisées contraint et d une* RAMC*-dérivation. Si $S \rightarrow_{\mathcal{R}_{gen}} S'$, il est possible de construire automatiquement une* RAMC*-dérivation d' de S' telle que :*

- *Si d est une réfutation de S, alors d' est une réfutation de S'.*
- *Si d est une construction de modèle pour S, alors d' est une construction de modèle pour S'.*

De plus : $S' \succeq_{gen} S$.

PREUVE. Il suffit de considérer successivement chaque règle pour montrer que la correction de la dérivation est préservée et que la dérivation obtenue est plus générale que la dérivation initiale. Nous ne donnons pas ici la preuve détaillée. Elle peut être trouvée dans (DÉFOURNEAUX, 1997; BOURELY ET AL., 1997). C.Q.F.D.

13.4 Filtrage

La phase de *filtrage* consiste à rechercher, parmi les formules de la base de connaissances, une formule plus générale que la conjecture considérée S_T. Pour cela, il s'agit de trouver une formule contrainte $(S, \mathcal{X})$ de la base de connaissances vérifiant

$$(S, \mathcal{X}) \succeq (S_T, \top).$$

Ce problème peut être résolu en cherchant une substitution σ telle que $S\sigma \sqsubseteq S_T$ (si S est insatisfaisable) ou $S_T \sqsubseteq S\sigma$ (si S est satisfaisable). Ce problème doit naturellement être effectué modulo les propriétés habituelles des connectifs logiques $\vee, \wedge, \forall \ldots$ et modulo la relation d'équivalence $\cong$. Si le filtrage réussit, il suffit alors d'appliquer cette substitution à la RAMC-dérivation associée à S pour obtenir une RAMC-dérivation à partir de S_T, donc un *modèle* ou une *réfutation* de S_T.

La définition d'un tel algorithme est l'objet de la section courante.

DÉFINITION 13.4.1. *Un* problème de filtrage *est inductivement défini de la façon suivante : soit* $\top$, $\bot$, $\mathcal{P}_1 \vee \mathcal{P}_2$, $\mathcal{P}_1 \wedge \mathcal{P}_2$, $\exists x.\mathcal{P}$, $\forall x.\mathcal{P}$, *(où* $\mathcal{P}$ *est un problème de filtrage et* x *une variable) ou de la forme* $t = s, t \neq s$ *(où* t *et* s *sont deux termes ou deux formules) ou* $C \leq_{dissub} D$, $S \sqsubseteq S'$ *(où* S *et* S' *sont deux ensembles de c-clauses et* C, D *deux c-clauses).* ◇

Afin de donner une sémantique aux problèmes de filtrage, il suffit de donner la sémantique des formules atomiques, la sémantique des formules quelconques s'en déduisant de façon habituelle.

DÉFINITION 13.4.2. *Une substitution fermée* σ valide *un problème de filtrage de la forme*

1. $t = s$ *si et seulement si* $t\sigma = s\sigma$,
2. $t \neq s$ *si et seulement si* $t\sigma \neq s\sigma$,
3. $S \sqsubseteq S'$ *si et seulement si* $S\sigma \sqsubseteq S'\sigma$,
4. $C \leq_{dissub} D$ *si et seulement si* $C\sigma \leq_{dissub} D\sigma$.

◇

Soit $(S_S, \mathcal{X})$ une séquence contrainte, représentant une RAMC-dérivation (formule-source). Soit S_T un ensemble de c-clauses (formule cible). Afin de trouver une substitution σ telle que $S_S\sigma \subseteq S_T$ (cas insatisfaisable) ou $S_T \subseteq S_S\sigma$ (cas satisfaisable), nous allons résoudre le problème de filtrage

$$\mathcal{X} \wedge (S_S \sqsubseteq S_T) \text{ (cas insatisfaisable)}$$

ou

$$\mathcal{X} \wedge (S_T \sqsubseteq S_S) \text{ (cas satisfaisable)}.$$

Ce problème sera noté $Match((S, \mathcal{X}), S_T)$ dans la suite.

Remarquons que ce problème contient la résolution des formules équationnelles d'ordre supérieur dans une algèbre définie par la relation d'équivalence $\cong$ introduite au chapitre 6. Une étude de la résolution de formules équationnelles d'ordre supérieur peut être trouvée dans (LUGIEZ, 1995), où certains cas de décidabilité et d'indécidabilité sont précisément identifiés. La résolution des formules équationnelles d'ordre supérieur est indécidable (et non semi-décidable) en général. Ici le problème que nous étudions est très différent : nous considérons un cas très particulier de formule équationnelle d'ordre supérieur (car le problème initial est sans quantificateur, et S_T ne contient aucune variable), et nous effectuons la résolution modulo un ensemble de propriétés des connectifs logiques (commutativité, associativité, etc.). Un algorithme de filtrage d'ordre supérieur tenant compte des propriétés d'associativité et de commutativité peut être trouvé dans (CURIEN ET AL., 1996). Celui-ci n'est cependant pas suffisant pour notre travail, puisque

nous cherchons à résoudre le problème modulo la relation $\cong$, et non pas seulement modulo les propriétés AC des connectifs logiques.

Dans cette section, nous donnons simplement un système de règles correct (au sens où toute solution du problème obtenu est une solution du problème initial) permettant de trouver certaines solutions d'une formule équationnelle d'ordre 2. Nous ne cherchons pas à obtenir un algorithme efficace, ni même à donner des résultats de complétude ou de terminaison de ce système (cela sort largement du cadre de notre travail). Dans (Défourneaux et Peltier, 1997b; Défourneaux, 1997) est proposé un algorithme de filtrage partiel plus général, opérant sur les *dérivations* et non plus seulement sur les ensembles de c-clauses.

Transformation des formules

Les règles suivantes sont appliquées sur les formules apparaissant dans le problème.

Règles de transformation

$P \vee Q \quad \rightarrow Q \vee P$
$P \vee (Q_1 \vee Q_2) \rightarrow (P \vee Q_1) \vee Q_2$
$P \quad \rightarrow \neg\neg P$
$P \quad \rightarrow P \vee \bot$
$P \quad \rightarrow P \vee P$

Règles d'unification

$t = t \quad \rightarrow \top$
$t \neq t \quad \rightarrow \bot$
$f(\overline{t}) = f(\overline{s}) \rightarrow \overline{t} = \overline{s}$
$f(\overline{t}) \neq f(\overline{s}) \rightarrow \overline{t} \neq \overline{s}$
$f(\overline{t}) = g(\overline{s}) \rightarrow \bot \quad$ si $f \neq g$
$f(\overline{t}) \neq g(\overline{s}) \rightarrow \top \quad$ si $f \neq g$
$x = t \quad \rightarrow \bot \quad$ si $x \in t$
$x \neq t \quad \rightarrow \top \quad$ si $x \in t$

Remplacement

$x = t \wedge \mathcal{P} \rightarrow x = t \wedge \mathcal{P}\{x \leftarrow t\}$ si $x \notin t$
$x \neq t \vee \mathcal{P} \rightarrow x \neq t \vee \mathcal{P}\{x \leftarrow t\}$ si $x \notin t$

Explosion

$\mathcal{P} \rightarrow \exists u_1, \ldots, u_m.\mathcal{P} \wedge x = \lambda x_1, \ldots, x_n.f(u_1(x_1, \ldots, x_n), \ldots, u_m(x_1, \ldots, x_n))$
$\mathcal{P} \rightarrow \mathcal{P} \wedge x = \lambda x_1, \ldots, x_n.x_i$

où $x \in \mathcal{V}$, $f \in \Sigma$ $x : s_1 \times \ldots \times s_n \rightarrow s$, $f : s'_1 \times \ldots \times s'_m \rightarrow s$ *et pour tout* $i \leq m$, u_i *est une nouvelle variable de type* $s_1 \times \ldots \times s_n \rightarrow s'_i$.

Elimination de $\sqsubseteq$

$S_1 \cup S_2 \sqsubseteq S \quad \rightarrow S_1 \sqsubseteq S \wedge S_2 \sqsubseteq S$
$\emptyset \sqsubseteq S \quad \rightarrow \top$
$S \sqsubseteq \emptyset \quad \rightarrow S = \emptyset$
$\{\forall \overline{x}.C\} \sqsubseteq S \quad \rightarrow \forall \overline{x}.(\{C\} \sqsubseteq S)$
$\{C\} \sqsubseteq S_1 \cup S_2 \rightarrow \{C\} \sqsubseteq S_1 \vee \{C\} \sqsubseteq S_2$
$\{C\} \sqsubseteq \{D\} \quad \rightarrow D \leq_{dissub} C$

Elimination de $\leq_{dissub}$

$C \leq_{dissub} \forall y.D \quad \rightarrow \forall y.(C \leq_{dissub} D)$
$\forall x.C \leq_{dissub} D \quad \rightarrow \exists x.(C \leq_{dissub} D)$
$[\![C \ : \mathcal{X}]\!] \leq_{dissub} [\![D \ : \mathcal{Y}]\!] \rightarrow \neg\mathcal{X} \vee \mathcal{Y} \wedge D \leq_{dissub} C$
$C_1 \vee C_2 \leq_{dissub} D \quad \rightarrow (C_1 \leq_{dissub} D) \wedge (C_2 \leq_{dissub} D)$
$C \leq_{dissub} D_1 \vee D_2 \quad \rightarrow (C \leq_{dissub} D_1) \vee (C \leq_{dissub} D_2)$
$\bot \leq_{dissub} C \quad \rightarrow \top$
$P \leq_{dissub} Q \quad \rightarrow P = Q$

Nous notons $\mathcal{U}$ le système composé des règles **Unification**, **Transformation**, **Explosion**, **Remplacement**, des règles spécifiques d'élimination de $\sqsubseteq$ et $\leq_{dissub}$ et des règles habituelles de transformation des formules équationnelles (voir (Comon et Lescanne, 1989; Comon et Delor, 1994)).

Lemme 13.4.1. *[Correction syntaxique du système $\mathcal{U}$] Si $\mathcal{P} \rightarrow_{\mathcal{U}} \mathcal{P}'$ alors $\mathcal{S}(\mathcal{P}') \subseteq \mathcal{S}(\mathcal{P})$.*

Preuve. Il suffit de vérifier que chaque règle transforme un problème $\mathcal{P}$ en un problème $\mathcal{P}'$ tel que $\mathcal{S}(\mathcal{P}') \subseteq \mathcal{S}(\mathcal{P})$, ce qui est immédiat. c.q.f.d.

Définissons maintenant une forme particulière de problèmes de filtrage pour laquelle il est possible d'extraire automatiquement une solution.

Définition 13.4.3. *[forme normale] Un problème de filtrage est dit en* forme normale *si et seulement si il est de la forme $\bigwedge_{i=1}^{n} x_i = t_i$, où, pour tout $i \leq n$, x_i n'a qu'une occurrence dans la conjonction $\bigwedge_{i=1}^{n} .x_i = t_n$ et où toutes les variables apparaissant dans les t_i sont d'arité 0.* $\diamond$

Théorème 13.4.1. *Un problème de filtrage en forme normale a au moins une solution.*

Preuve. Immédiate. c.q.f.d.

Le théorème suivant assure que la reconstruction de la preuve ou du contre-exemple de la formule cible S_T est possible à partir d'une preuve/contre-exemple de la formule-source S_S et d'une solution du problème de filtrage $Match(S_S, S_T)$.

Théorème 13.4.2. *Soit $(S_S, \mathcal{X})$ un ensemble contraint, S_T un ensemble de c-clauses et σ une solution de $Match((S_S, \mathcal{X}), S_T)$.*

1. *Si $S_S\sigma$ est insatisfaisable alors S_T l'est également.*
2. *Si $S_S\sigma$ est satisfaisable alors S_T l'est également et tout modèle de $S_S\sigma$ est un modèle de S_T.*

Preuve. C'est une conséquence immédiate du lemme 13.2.1. c.q.f.d.

13.5 Le cas où l'analogie échoue

Lorsque l'analogie échoue, c'est-à-dire quand l'insatisfaisabilité ou la satisfaisabilité de l'ensemble de clauses cible *ne peut pas être* directement déduite de celle de la conjecture-source, il est possible d'utiliser la méthode proposée pour générer des informations permettant de poursuivre la recherche. Pour cela, nous allons utiliser un algorithme de *filtrage partiel*, autorisant certaines clauses ou sous-clauses de l'ensemble-source à ne correspondre à aucune c-clause de l'ensemble cible. Les c-clauses ainsi obtenues peuvent alors être considérées comme des *lemmes*. On peut compléter ensuite la dérivation de deux manières différentes.

- Soit en prouvant par la suite le (ou les) lemme(s) en utilisant un démonstrateur quelconque.
- Soit en montrant par la suite que les lemmes sont *compatibles* avec l'ensemble cible, c'est-à-dire que l'ajout de ces lemmes à l'ensemble préserve la satisfaisabilité de l'ensemble (cas satisfaisable).

La génération de lemmes peut également être utilisée pour détecter de "mauvaises" analogies, en générant des lemmes qui ne peuvent être prouvés. Afin de formaliser cette idée, il est nécessaire de donner une nouvelle sémantique aux problèmes de filtrage.

EXEMPLE 13.5.1. Considérons par exemple les ensembles $\mathcal{F}_1$: $\{P(a), R(a)\}$ et $\mathcal{F}_2$: $\{P(a), \{\neg P(a) \vee R(a)\}\}$. $\mathcal{F}_1$ et $\mathcal{F}_2$ sont syntaxiquement distincts. Cependant, on remarque facilement que la clause $\neg P(a) \vee R(a)$ peut être réduite à $R(a)$ (à cause de la clause $P(a)$) donc que $\mathcal{F}_2$ est en fait équivalent à $\mathcal{F}_1$. Dans cet exemple, le problème de filtrage $\mathcal{F}_2 \sqsubseteq \mathcal{F}_1$ *n'a pas de solution*, alors que $\mathcal{F}_1$ et $\mathcal{F}_2$ sont en fait *sémantiquement* équivalents. ◇

Nous proposons donc de modifier la définition des solutions d'un problème de filtrage afin de prendre en compte certaines propriétés sémantiques des formules.

DÉFINITION 13.5.1. *Soit $\mathcal{F}$ un problème de filtrage. Soit $\mathcal{I}$ une interprétation. Une substitution σ est une* solution sémantique *de $\mathcal{F}$ par rapport à $\mathcal{I}$ si :*

- *$\mathcal{F}$ est de la forme $t = s$ (où t, s sont deux termes) et $\mathcal{I} \models t = s$.*
- *$\mathcal{F}$ est de la forme $\phi = \psi$ (où ϕ, ψ sont deux formules) et $\mathcal{I} \models \phi - \psi$.*
- *$\mathcal{F}$ est de la forme $S_1 \sqsubseteq S_2$ et il existe un ensemble de c-clauses S_3 tel que $S_3 \sqsubseteq S_2$ et $\mathcal{I} \models (S_1 - S_2)$.*
- *$\mathcal{F}$ est de la forme $C \leq_{dissub} D$ et il existe une clause C' telle que $C' \leq_{dissub} D$ et $\mathcal{I} \models (C - C')$.*

L'ensemble des solutions sémantiques de $\mathcal{F}$ par rapport à $\mathcal{I}$ est noté $\mathcal{S}_{\mathcal{I}}(\mathcal{F})$. ◇

Notons que la condition $\mathcal{F}_1\sigma \equiv_{\mathcal{I}} \mathcal{F}_2\sigma$ est *plus faible* que la condition $\mathcal{F}_1\sigma = \mathcal{F}_2\sigma$ utilisée dans la définition 13.4.2. En effet, il existe des formules sémantiquement équivalentes mais syntaxiquement distinctes.

DÉFINITION 13.5.2. *[forme pseudo-normale] Un problème de filtrage est dit en forme pseudo-normale si et seulement s'il est de la forme $\mathcal{P} : \bigwedge_{i=1}^{n} x_i = t_i \wedge \mathcal{F} = \top$, avec :*

- *pour tout $i \leq n$, x_i n'a qu'une occurrence dans $\mathcal{P}$.*
- *$\mathcal{F}$ est une formule du premier ordre.*

◇

Nous allons maintenant ajouter de nouvelles règles, afin de tirer parti de cette nouvelle définition. Ces règles ont pour objet de transformer une formule en une formule sous forme pseudo-normale.

Nouvelles règles de transformation : génération de lemmes

$\forall y.(C = \top) \quad \rightarrow (\forall y.C) = \top$
$\exists y.(C = \top) \quad \rightarrow (\exists y.C) = \top$
$C = \bot \wedge D = \bot \rightarrow C \vee D = \bot$
$C = \bot \vee D = \bot \rightarrow C \wedge D = \bot$
$C = \top \vee D = \top \rightarrow C \vee D = \top$
$C = \top \wedge D = \top \rightarrow C \wedge D = \top$
$S = \emptyset \quad \rightarrow S = \top \quad$ (S est un ensemble de c-clause)
$\mathcal{F} = \bot \quad \rightarrow \neg\mathcal{F} = \top$

Nous notons $\mathcal{U}_s$ le système de règles ainsi obtenu.

LEMME 13.5.1. *[Correction sémantique du système $\mathcal{U}_s$] Soit $\mathcal{P} \rightarrow_{\mathcal{U}_s} \mathcal{P}'$. Pour toute interprétation $\mathcal{I}$, $\mathcal{S}_{\mathcal{I}}(\mathcal{P}') \subseteq \mathcal{S}_{\mathcal{I}}(\mathcal{P})$.*

PREUVE. Là encore, la preuve est immédiate (il suffit de vérifier la propriété pour chaque règle). C.Q.F.D.

THÉORÈME 13.5.1. *Soit $(S_S, \mathcal{X})$ un ensemble contraint et S_T un ensemble de c-clauses tel que $Match((S_S, \mathcal{X}), S_T)$ est réduit par le système $\mathcal{U}_s$ à un problème en forme pseudo-normale $\mathcal{P} \wedge \mathcal{F} = \top$.*

1. *Si S_S est insatisfaisable et si toute interprétation $\mathcal{I}$ validant S_T valide $\mathcal{F}$, alors S_T est insatisfaisable.*
2. *Si S_S est satisfaisable et si il existe un modèle $\mathcal{I}$ de S_S tel que $\mathcal{I} \models \mathcal{F}$, alors S_T est satisfaisable et $\mathcal{I} \models S_T$.*

PREUVE. 1. Soit $\mathcal{I}$ une interprétation validant S_T. On a d'après le lemme 13.5.1, $\mathcal{S}_\mathcal{I}(\mathcal{P} \wedge \mathcal{F} = \top) \subseteq \mathcal{S}_\mathcal{I}(Match((S_S, \mathcal{X}), S_T))$. $\mathcal{P}$ est en forme normale donc $\mathcal{P}$ admet une solution θ. On a $\mathcal{S}_\mathcal{I}(\mathcal{F}\theta = \top) \subseteq \mathcal{S}_\mathcal{I}(Match((S_S, \mathcal{X}), S_T)\theta)$, d'où $\mathcal{S}_\mathcal{I}(\mathcal{F} = \top) \subseteq \mathcal{S}_\mathcal{I}(Match((S_S, \mathcal{X}), S_T)\theta)$. Puisque $\mathcal{I} \models \mathcal{F}$, θ est une solution de $Match((S_S, \mathcal{X}), S_T)$ par rapport à $\mathcal{I}$. D'où S_T est insatisfaisable.

2. La preuve est similaire.

C.Q.F.D.

Dans les deux cas, la satisfaisabilité/insatisfaisabilité de S_T est déduite de façon immédiate de celle de S_S.

Montrer que $\mathcal{F}$ est valide dans tous les modèles de S_T revient alors à prouver que $\neg\mathcal{F} \wedge S_T$ est insatisfaisable, ce qui peut être prouvé en utilisant n'importe quel démonstrateur. La preuve peut également être obtenue par un *appel récursif* à la procédure de filtrage.

Si la formule-source est satisfaisable, il s'agit alors, d'après le théorème 13.5.1, de trouver une extension $\mathcal{I}$ du modèle (partiel) de $S_S\sigma$ validant $\mathcal{F}$. La différence avec le cas insatisfaisable est que l'interprétation $\mathcal{I}$ considérée *n'est pas connue* à priori avant le début de la recherche.

Dans les deux cas, la dérivation conduisant à la clause vide (resp. au modèle) à partir de l'ensemble de c-clauses cibles peut être reconstruite de façon immédiate, à partir de la dérivation source et des preuves/constructions de contre-exemple correspondant à la formule $\mathcal{F}$.

13.6 Exemples

13.6.1 Illustration de l'algorithme de filtrage

Les deux premiers exemples sont extrêmement simples. Ils ont uniquement pour objet d'illustrer le fonctionnement de l'algorithme de filtrage (cas insatisfaisable et satisfaisable). Le troisième exemple proposé est plus complexe et illustre l'intérêt de notre approche en Déduction Automatique.

Cas insatisfaisable

EXEMPLE 13.6.1. Soit S_S l'ensemble

$$\{L(a), \neg L(x) \vee L(f(x)), \neg L(f(a))\}.$$

Cet ensemble est transformé par l'algorithme de généralisation en

$$\{P, \neg P \vee Q, \neg Q\}.$$

Considérons maintenant la formule cible S_T suivante

$$\{L(a), \neg L(x) \vee L(f(x)), \neg L(f(f(a)))\}.$$

Nous allons prouver l'insatisfaisabilité de S_T par analogie avec S_S. Cela n'est pas possible directement, et passe par la génération d'un lemme intermédiaire.

L'algorithme de filtrage donne :

$$
\begin{array}{l}
S_S \sqsubseteq S_T \\
\rightarrow \{P, \neg P \vee Q, \neg Q\} \sqsubseteq \{L(a), \forall x.(\neg L(x) \vee L(f(x))), \neg L(f(f(a)))\} \\
\rightarrow L(a) \leq_{dissub} P \wedge (\forall x. \neg L(x) \vee L(f(x))) \leq_{dissub} (\neg P \vee Q) \\
\quad \wedge (\forall x.(\neg L(x) \vee L(f(x)))) \leq_{dissub} \neg Q \\
\rightarrow P = L(a) \wedge \exists x.(P = L(x) \wedge Q = L(f(x))) \wedge \exists x.(x = f(a) \wedge L(f(x)) = \bot) \\
\rightarrow P = L(a) \wedge Q = L(f(a)) \wedge \neg L(f(f(a))) = \top
\end{array}
$$

Ici, l'algorithme syntaxique (défini par les règles $\mathcal{U}$) échoue puisque l'équation $\neg L(f(f(a))) = \top$ n'a pas de solution. Cependant, le problème obtenu est en forme pseudo-normale. Il s'agit alors de prouver que la formule $\neg L(f(f(a)))$ est valide dans toute interprétation $\mathcal{I}$ validant S_T. Pour cela, nous ajoutons $L(f(f(a)))$ à l'ensemble S_T. Nous obtenons la formule S'_T

$$\{L(a), \neg L(x) \vee L(f(x)), L(f(f(a))), \neg L(f(f(a))), L(f(a))\}$$

qui est évidemment insatisfaisable. La reconstruction de la preuve est alors immédiate. $\diamond$

Cas satisfaisable

Dans le cas où l'ensemble-source est satisfaisable, le filtrage partiel produit un modèle partiel de l'ensemble de c-clauses. L'ensemble de lemmes est alors l'ensemble des c-clauses non-valides dans cette interprétation. Il s'agit alors de compléter le modèle en étendant le modèle partiel construit par analogie.

EXEMPLE 13.6.2. Soit l'ensemble de c-clauses source suivant.

$$\{P \vee Q, \neg P\}$$

La dérivation (triviale) est la suivante.

$$
\begin{array}{l}
S_S \\
\quad \rightarrow_{\mathcal{R}es} \ \{P \vee Q, \neg P, Q\} \\
\quad \rightarrow_{DSub} \{\neg P, Q\}
\end{array}
$$

Soit l'ensemble de c-clauses suivant ($\Sigma = \{f^{(1)}, a^{(0)}\}$).

$$
\begin{array}{l}
[\![P(x) \vee Q(x) \ : \top]\!] \\
[\![\neg P(a) \ : \top]\!] \\
[\![R(x) \vee \neg R(x) \ : \top]\!]
\end{array}
$$

L'algorithme de filtrage donne :

$\{\forall x.P_1(x) \vee Q_1(x), \neg P_1(a), \forall x.(R(x) \vee \neg R(x))\} \sqsubseteq \{(P \vee Q), \neg P\}$

$\rightarrow P = P_1(a) \wedge Q = Q_1(a) \wedge \forall x.(P_1(f(x)) \vee Q_1(f(x))) = \top \wedge \forall x.(R(x) \vee \neg R(x)) = \top\}$

La substitution obtenue est

$$\sigma : P \rightarrow P_1(a)$$

$$\sigma : Q \rightarrow Q_1(a).$$

Le modèle partiel est par conséquent $\{\neg P_1(a), Q_1(a)\}$. Il reste à trouver une extension de $\{\neg P_1(a), Q_1(a)\}$ validant $\forall x.(P_1(f(x)) \vee Q_1(f(x))) \wedge \forall x.(R(x) \vee \neg R(x))$. Appliquons la méthode RAMC sur l'ensemble de c-clauses :

$$[\![\neg P_1(a) \ : \top]\!]$$
$$[\![Q_1(a) \ : \top]\!]$$
$$[\![P_1(f(x)) \vee Q_1(f(x)) \ : \top]\!]$$
$$[\![R(x) \vee \neg R(x) \ : \top]\!]$$

Nous obtenons, par application des règles **dissubsomption, distautologie et GPL**, le modèle :

$$[\![\neg P_1(a) \ : \top]\!]$$
$$[\![P_1(f(x)) \ : \top]\!]$$
$$[\![Q_1(a) \ : \top]\!]$$

◇

13.6.2 Un exemple plus complexe

Nous donnons ci-dessous un exemple plus compliqué montrant l'intérêt de notre approche en Déduction Automatique.

EXEMPLE 13.6.3. Considérons les problèmes SYN310-1 et SYN312-1 de la base de problèmes TPTP (SUTTNER ET SUTCLIFFE, 1996) (tirés de (FERMÜLLER ET AL., 1993)).

Le problème-source S_S est le suivant.

$$\neg p(x2, x1, x) \vee p(x, x1, x2)$$
$$\neg p(x1, x, x2) \vee p(x, x1, x2)$$
$$\neg p(x, x1, g(x2)) \vee p(x, x1, x2)$$
$$\neg p(f(x), x1, x2) \vee p(x, x1, x2)$$
$$\neg p(a, b, c)$$
$$p(f(g(a)), f(g(b)), f(g(c)))$$

Le problème-cible est S_T:

$$p'(x, x3, x2) \vee \neg p'(x, x1, x2) \vee \neg p'(x1, x3, x2)$$
$$p'(x2, x1, x) \vee \neg p'(x, x1, x2)$$
$$p'(x1, x, x2) \vee \neg p'(x, x1, x2)$$
$$p'(x, x1, f'(x2)) \vee \neg p'(x, x1, x2)$$
$$p'(g'(x), x1, x2) \vee \neg p'(x, x1, x2)$$
$$p'(a', f'(b'), c')$$
$$p'(f'(b'), d, c')$$
$$\neg p'(g'(f'(a')), g'(f'(d)), g'(f'(c')))$$

S_S est très simple (OTTER 3.0 (MCCUNE, 1995) met 0.17 secondes pour trouver la preuve en mode "automatique" [2]). La preuve de S_T est plus difficile à obtenir (OTTER met 536.31 s pour trouver la preuve). Par conséquent, nous supposons qu'une preuve de S_S est connue et nous cherchons à construire une preuve de S_T *par analogie* avec celle de S_S. Appliquons l'algorithme de filtrage afin de résoudre le problème "$S_S \sqsubseteq S_T$?". Aucune solution ne peut être trouvée, car S_T n'est pas une instance de S_S. En revanche, nous obtenons la solution partielle suivante.

$$\{p = \lambda x_1, x_2, x_3.\neg p'(x_3, x_2, x_1), g = \lambda x.f'(x), f = \lambda x.g'(x), a = a', b = b', c = c', d = d'\}$$

2. i.e. avec le paramètre `set(auto)`.

avec les lemmes :

$$\{p'(a,b,c), \forall x_1, x_2, x_3 . p'(x_1, x_2, x_3) \vee \neg p'(x_1, x_3, x_2)\}$$

Ces deux formules sont des clauses de S_S qui ne sont pas filtrées par S_T.

D'après le théorème 13.5.1, il reste simplement à prouver ces lemmes. Le premier peut être démontré très facilement, par exemple en ajoutant la négation de $p'(a,b,c)$ à S_T et en utilisant n'importe quel démonstrateur par réfutation (OTTER met 0.37 s pour démontrer l'insatisfaisabilité de l'ensemble obtenu).

Afin de transformer le second lemme en ensemble de clauses, il convient d'utiliser la skolémisation. On obtient $\{\neg p'(a_1, a_2, a_3), p'(a_1, a_3, a_2)\}$. Ces clauses sont alors ajoutées à S_T. L'ensemble obtenu peut très facilement être réfuté par un démonstrateur (OTTER met 0.18 s).

Cet exemple suggère que le raisonnement par analogie peut augmenter de façon significative les performances d'un démonstrateur. En effet, le temps total nécessaire pour prouver les lemmes est de 0.55 s et le temps requis pour prouver le théorème sans utiliser l'analogie est de $536.31s$. Cependant, le processus de filtrage reste à implémenter, afin d'effectuer des comparaisons plus significatives. ◇

13.7 Discussion et perspectives

Nous avons présenté deux calculs pour la découverte et l'exploitation de l'analogie entre ensembles de c-clauses :

1. des règles permettant de généraliser un ensemble de c-clauses.
2. un algorithme permettant de détecter et d'exploiter les similarités entre ensembles de c-clauses. Cet algorithme génère en outre des *lemmes* permettant de reconstruire la preuve ou le contre-exemple de la formule cible à partir de celui de la formule source.

Les travaux présentés constituent les premiers pas vers la réalisation d'un système de recherche de réfutation et de modèles par analogie qui se décompose en plusieurs étapes.

1. **Généralisation de la conjecture-source.** L'algorithme de généralisation a été présenté en détail à la section 13.3. Il a fait l'objet d'une implémentation par Gilles DÉFOURNEAUX (voir (DÉFOURNEAUX, 1997)).
2. **Stockage dans la base de connaissances.** La définition de la base de connaissances (représentation des formules, algorithme de stockage, etc.) sort largement du cadre de ce travail. Nous ne l'aborderons pas ici. Ces problèmes sont traités de façon plus complète dans (DÉFOURNEAUX, 1997).
3. **Filtrage.** Lors de la présentation d'une nouvelle conjecture (conjecture cible) il faut rechercher dans la base de connaissances une formule analogue dont la preuve/contre-exemple est connu. Dans ce travail, nous avons défini un algorithme de filtrage correct. Il reste à donner une **stratégie** d'application des règles assurant la terminaison et si possible la complètude du système (dans certains cas). Cette stratégie doit également être suffisamment efficace pour traiter des problèmes de taille importante. Ces questions sont là encore traitées en détail dans (DÉFOURNEAUX, 1997). Nous n'avons pas cherché à les aborder ici, préférant étudier le cas où l'analogie échoue (par abduction et génération de *lemmes*). L'algorithme de filtrage est actuellement en cours d'implémentation.

L'analogie apparaît comme un moyen puissant de guider la recherche de preuve et de contre-exemple. Il reste cependant beaucoup de problèmes à résoudre. En premier lieu, il nous faut étudier la complexité des algorithmes que nous avons proposés et en particulier donner une version plus "procédurale" de notre algorithme de filtrage. En second lieu, les formules de filtrage

que nous avons considérées ici sont des relations entre deux formules. D'un point de vue pratique, il faut définir des algorithmes (utilisant par exemple des techniques d'indexation des termes ou des formules) permettant de comparer directement la formule cible avec l'ensemble des formules de la base de connaissances. C'est une nécessité, car une comparaison séquentielle entre la formule cible et l'ensemble de toutes les formules de la base de connaissances risque de s'avérer impraticable.

Il est également nécessaire d'étendre les définitions et les règles de généralisation que nous avons proposées aux cas de formules du premier ordre (et non d'ensembles de c-clauses). En particulier la définition de la notion de *généralité* (et donc de la relation d'analogie) ne s'applique qu'à des ensembles de c-clauses. Est-il possible d'en donner une définition qui puisse s'appliquer à des formules quelconques?

Chapitre 14

Application à la programmation en Logique

14.1 Programmes logiques sans négation

La Programmation en Logique est une des applications les plus connues et les plus utilisées de la Déduction Automatique. Elle peut être vue comme une spécialisation de la Déduction Automatique à une classe particulière de formules. Si les deux domaines ont évolués de manière indépendante, ils ont cependant conservé des rapports étroits, au point que Wos et McCune n'hésitent pas à employer le terme de "symbiose" pour qualifier leurs rapports (Wos et McCune, 1991).

Le principe de la Programmation en Logique est simple : il consiste à utiliser la logique comme langage de spécification exécutable. Bien entendu, pour que ce soit réaliste, il est nécessaire de considérer des classes de formules avec des propriétés raisonnables et des calculs efficaces. Dans sa version la plus répandue, le sous-langage choisi est celui des *clauses de Horn*, et le calcul la règle de *SLD-résolution*. Les clauses de Horn possèdent un double avantage: d'une part, les formules satisfaisables de cette classe ont la propriété d'avoir un modèle minimal unique (ce qui permet de définir une sémantique univoque) et d'autre part, la satisfaisabilité d'un ensemble de clauses de Horn peut être décidée de façon "plus simple" que dans le cas général (elle reste cependant indécidable pour les clauses du premier ordre, mais est *polynômiale* pour des ensembles de clauses de Horn propositionnelles). Un *programme* est alors défini comme un ensemble de clauses de Horn, et la sémantique associée est donnée par le modèle minimal du programme. L'exécution d'un programme correspond à la recherche des solutions d'un but b donné dans le modèle du programme, c'est-à-dire à la recherche des substitutions σ telles que $b\sigma$ est valide dans le modèle minimal associé au programme. L'objectif de ce type de langage est de rapprocher le plus possible un programme de sa *spécification*: on cherche à donner une description de la fonction calculée par le programme sans s'intéresser (en principe !) à la façon dont elle est évaluée.

Les principes théoriques de la Programmation en Logique sont brièvement rappelés ci-dessous. Pour une présentation plus détaillée, le lecteur pourra se reporter à (Emden et Kowalski, 1976; Lloyd, 1987; Apt, 1990). Nous étendons également les résultats et définitions habituels en Programmation en Logique à des programmes avec contraintes.

Définition 14.1.1. *Une* c-clause de Horn *est une c-clause dont la partie clause contient au plus un littéral positif. Une clause contrainte de Horn contenant un littéral positif est dite* définie. *Elle est de la forme :* $[\![\neg L_1 \vee \ldots \vee \neg L_n \vee L \ : \mathcal{X}]\!]$, *où les littéraux* L, L_i $(1 \leq i \leq n)$ *sont tous positifs. Elle peut aussi s'écrire sous la forme :* $[\![L \leftarrow L_1 \wedge \ldots \wedge L_n \ : \mathcal{X}]\!]$. *Le c-littéral* $[\![L \ : \mathcal{X}]\!]$ *est appelé la* tête *de la c-clause et noté* $\mathcal{H}ead([\![L \leftarrow L_1 \wedge \ldots \wedge L_n \ : \mathcal{X}]\!])$. *Un* programme logique *est un ensemble fini de c-clauses de Horn.* ◇

Remarque. Nous écrirons C à la place de $[\![C \ : \top]\!]$.

La particularité des programmes logiques est de posséder un modèle particulier appelé *minimal*, qui est le plus petit modèle du programme.

THÉORÈME 14.1.1. *Soit un programme satisfaisable $\mathcal{P}$. Soit $\mathcal{M}$ l'interprétation totale définie par les relations (pour tout littéral positif fermé L) :*
$\mathcal{M}(L) = true$ si L est une conséquence logique de $\mathcal{P}$
$\mathcal{M}(L) = false$ sinon
Alors $\mathcal{M} \models \mathcal{P}$.

$\mathcal{M}$ est appelé *modèle minimal* de $\mathcal{P}$. En effet, tout modèle $\mathcal{M}'$ de $\mathcal{P}$ est "inclu" dans $\mathcal{M}$ au sens où, pour tout littéral positif L : $\mathcal{M} \models L \Rightarrow \mathcal{M}' \models L$. Ce résultat permet d'associer à un programme logique une sémantique unique. Nous noterons $M_{\mathcal{P}}$ le modèle minimal de $\mathcal{P}$. La sémantique d'un programme logique peut être également définie en terme d'opérateur de point fixe (EMDEN ET KOWALSKI, 1976). Afin de définir formellement cette sémantique, rappelons brièvement les notions de nombre ordinal et de puissance ordinale d'un opérateur de point fixe, qui seront utiles dans la suite.

Ordinaux

DÉFINITION 14.1.2. *Un* nombre ordinal *est un ensemble α, tel que tout élément de α est une partie de α et il n'existe aucune suite infinie $a_0, a_1, \ldots$ d'éléments de α vérifiant la relation : $\forall i \in \mathbb{N}. a_{i+1} \in a_i$.* ◇

Les nombres ordinaux sont ordonnés par l'ordre d'appartenance "$\in$" sur les ensembles (noté $\leq$ dans ce cas).

On note 0 l'ordinal $\emptyset$. Pour tout ordinal α, on définit l'ordinal successeur $\alpha+1$ comme le plus petit ordinal supérieur à α (les ordinaux $\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots$sont donc notés $0, 1, 2 \ldots$). Soit E un ensemble d'ordinaux tel que, si $\alpha \in E$ alors : $\alpha + 1 \in E$ et $\forall \beta \leq \alpha. \beta \in E$. Alors E est encore un ordinal, appelé *ordinal limite*. En particulier l'ensemble des ordinaux de $\mathbb{N}$ est un ordinal noté ω et appelé le *premier ordinal transfini*.

Puissance ordinale

DÉFINITION 14.1.3. *Soit une application f, de $\mathcal{P}(E)$ dans $\mathcal{P}(E)$. Les puissances ordinales de f, (notées $f \uparrow^\alpha$) sont définies par :*

- $f \uparrow^0 = \emptyset$
- $f \uparrow^{\alpha+1} = f(f \uparrow^\alpha)$
- $f \uparrow^\alpha = \bigcup_{\beta \in \alpha} f \uparrow^\beta$ *(si α est un ordinal limite).*

◇

REMARQUE. On a : $f \uparrow^w = \bigcup_{i=0}^{\infty} f \uparrow^i$.

THÉORÈME 14.1.2. *Soit f une application de E dans E. Si f est monotone alors f admet un plus petit point fixe P et il existe un nombre ordinal α tel que $f \uparrow^\alpha = P$.*

L'hypothèse de *continuité* assure que le nombre ordinal α pour lequel l'opérateur atteint son plus petit point fixe est fini ou égal à ω.

DÉFINITION 14.1.4. *Un sous-ensemble P de $\mathcal{P}(E)$ est dit* orienté *lorsque pour chaque couple (A, B) de P^2, il existe un élément $C \in P$ tel que $A \cup B \subseteq C$.*

Une application f de $\mathcal{P}(E)$ dans $\mathcal{P}(E)$ est dite continue *si et seulement si pour toute partie orientée P de $\mathcal{P}(E)$ on a :*

$$\bigcup_{a \in P} f(a) = f(\bigcup_{a \in P} a)$$

◇

REMARQUE. Toute application continue de $\mathcal{P}(E)$ dans $\mathcal{P}(E)$ est monotone.

THÉORÈME 14.1.3. *Soit $f : \mathcal{P}(E) \to \mathcal{P}(E)$. Si f est continue alors le plus petit point fixe de f est $f \uparrow^{\omega}$ (c'est-à-dire la première puissance transfinie de f).*

Nous pouvons maintenant donner la définition de la sémantique d'un programme logique en terme d'opérateur de point fixe. Pour cela, nous introduisons l'opérateur $\mathcal{T}_{\mathcal{P}}$.

DÉFINITION 14.1.5. *Soit un programme satisfaisable $\mathcal{P}$. $\mathcal{T}_{\mathcal{P}}$ est la fonction de $\mathcal{P}(\mathcal{BH})$ dans $\mathcal{P}(\mathcal{BH})$ définie par :*

$$\mathcal{T}_P(\mathcal{I}) = \{A\sigma / [\![A \leftarrow \bigwedge_{i=1}^{k} L_i \ : \mathcal{X}]\!] \in \mathcal{P}, \sigma \in \mathcal{S}(\mathcal{X}), \forall i \leq k . L_k\sigma \in \mathcal{I}\}$$

◇

D'après (EMDEN ET KOWALSKI, 1976), $\mathcal{T}_{\mathcal{P}}$ est monotone et continue donc $\mathcal{T}_{\mathcal{P}}$ possède un plus petit point fixe $\mathcal{T}_{\mathcal{P}} \uparrow^{\omega}$.

THÉORÈME 14.1.4.

$$\mathcal{T}_{\mathcal{P}} \uparrow^{w} = M_{\mathcal{P}}^{+}$$

Il est connu que les programmes logiques permettent d'exprimer toutes les fonctions récursives (MINTS, 1990).

La théorie de la preuve en Programmation en Logique utilise habituellement la méthode de résolution et plus précisément une restriction de cette méthode : la résolution linéaire avec fonction de sélection (SLD-résolution). Nous nous contenterons ici une définition purement sémantique.

14.2 Spécialisation de RAMC à la Programmation en Logique

Dans cette section, nous présentons une spécialisation de la méthode RAMC à la Programmation en Logique et nous prouvons sa correction par rapport à la sémantique usuelle d'un programme logique. L'application de notre approche à la Programmation en Logique est une idée très naturelle. Cela permet de tirer parti des capacités de RAMC pour extraire plus d'informations que celles obtenues de façon purement déductive par les interpréteurs de programmes logiques existants. En particulier, nous montrons comment notre méthode permet de générer des faits négatifs *non fermés* qui ne peuvent en général pas être déduits par les approches existantes, fondées sur la négation par l'échec.

L'application de la méthode RAMC aux programmes logiques ne pose aucune difficulté particulière, puisque ceux-ci sont un cas particulier des ensembles de clauses contraintes. Cependant, il faut tenir compte de la sémantique particulière des programmes logiques, c'est-à-dire qu'il faut que le modèle minimal du programme soit préservé. Les règles GPL et GMPL ne peuvent donc pas être utilisées dans ce contexte puisqu'elles ne sont pas correctes (au sens où la conclusion n'est pas une conséquence logique du programme) mais seulement consistantes. Rien ne garantit en effet que le modèle minimal du programme sera préservé par application de ces règles.

Considérons, par exemple, le programme :

$even(succ(succ(x))) \leftarrow even(x).$
$even(0).$

La règle GMPL déduit la formule :

$$\forall x.even(x)$$

qui est un modèle du programme mais ne correspond évidemment pas à la signification attendue du prédicat *even* !

Une solution pour résoudre ce problème est de limiter l'application des règles GPL et GMPL à des ensembles de littéraux *négatifs*. Informellement, cette restriction assure qu'aucune information supplémentaire ne sera ajoutée au programme via un littéral positif (on ne modifiera donc pas le modèle minimal qui contient exclusivement des littéraux positifs).

Notons GPL$^-$ la règle :

$$\frac{S}{[\![\neg l(\overline{t}) \ : \mathcal{X}_{pure}]\!]}$$

où :

- $l \in \Omega$.
- $\mathcal{X}_{pure} = \bigwedge_{[\![l(\overline{s}) \leftarrow \mathcal{B} \ : \mathcal{Y}]\!] \in S} \forall \overline{y}.\overline{t} \neq \overline{s} \vee \neg \mathcal{Y}$ (où $\overline{y} = \mathcal{V}ar([\![l(\overline{s}) \leftarrow \mathcal{B} \ : \mathcal{Y}]\!])$).

De façon similaire, nous notons GMPL$^-$ la règle :

$$\frac{S}{E}$$

où E est un ensemble de littéraux *négatifs* obtenu en appliquant le système de règles suivant sur un ensemble quelconque de littéraux *négatifs* (en pratique un ensemble de c-littéraux de S) (L dénote un littéral positif).

(Restriction) $E \cup \{\neg L\} \rightarrow E \cup \{\neg L \setminus \neg L'\}$
Il existe une c-clause $C \in S$ telle que :
$\mathcal{H}ead(C) = L'$ et $DSub(E, C) \neq \top$.

(Extension) $E \rightarrow E \cup \{L'\}$
Si $C \in Res(E, S)$, $DSub(E, C) \neq \top$ et $L' \in C$
L' est négatif

(Simplification) $E \cup [\![C \ : \mathcal{X}]\!] \rightarrow E$
Si $\mathcal{X} \equiv \bot$

Nous notons RAMC$_{\text{PL1}}$ le calcul contenant les règles renommage, c-résolution, c-factorisation, c-disrésolution, c-disfactorisation, c-dissubsomption, distautologie, GPL$^-$, GMPL$^-$. Nous prouvons ci-dessous que ce nouveau calcul préserve la sémantique d'un programme logique.

THÉORÈME 14.2.1. *Soit $\mathcal{P}$ un ensemble de c-clauses de Horn et $\mathcal{P}'$ tel que $\mathcal{P} \rightarrow_{\text{RAMC}_{\text{PL1}}} \mathcal{P}'$ alors :*

1. *$\mathcal{P}'$ est un ensemble de c-clauses de Horn.*
2. *$\mathcal{P}$ est satisfaisable si et seulement si $\mathcal{P}'$ est satisfaisable.*
3. *Si $\mathcal{P}$ est satisfaisable alors : $M_{\mathcal{P}} = M_{\mathcal{P}'}$.*

PREUVE. Soit $\mathcal{P}$ un ensemble satisfaisable de c-clauses de Horn. Soit $\mathcal{P}'$ tel que $\mathcal{P} \rightarrow_{\text{RAMC}_{\text{PL1}}} \mathcal{P}'$.

1. Immédiat (il suffit de le vérifier pour chaque règle).

2. Immédiat (les règles de RAMC préservent la satisfaisabilité).
3. Il s'agit de vérifier que le modèle minimal $\mathcal{M}_{\mathcal{P}'}$ de $\mathcal{P}'$ est le même que le modèle minimal $\mathcal{M}_{\mathcal{P}}$ de $\mathcal{P}$.
   Toutes les règles de RAMC étant correctes, sauf les règles GPL et GMPL, nous pouvons donc sans perte de généralité considérer uniquement ces 2 règles. Soit une c-clause $[\![\neg L \ : \mathcal{X}]\!]$ déduite de $\mathcal{P}$ par la règle GPL$^-$. La règle GPL préserve la satisfaisabilité de l'ensemble de c-clauses donc $\mathcal{P} \cup [\![\neg L \ : \mathcal{X}]\!]$ est satisfaisable. Donc il existe un modèle $\mathcal{M}'$ de $\mathcal{P}$ tel que $[\![\neg L \ : \mathcal{X}]\!]$ est vraie dans $\mathcal{M}'$, c'est-à-dire tel que pour toute substitution $\sigma \in \mathcal{S}(\mathcal{X})$, $L\sigma$ n'appartient pas à $\mathcal{M}(P)^+$. Puisque M_P est le modèle minimal de $\mathcal{P}$, $\mathcal{M}_P^+$ ne contient aucune formule atomique n'appartenant pas à $\mathcal{M}'$. D'où $L\sigma \notin \mathcal{M}_P(L)^+$.
   Considérons maintenant la règle GMPL$^-$. Soit E un ensemble de c-littéraux déduit de $\mathcal{M}$ par application de cette règle. Alors, par définition, $E \cup \mathcal{P}$ est satisfaisable. Donc, il existe un modèle de $\mathcal{P}$ ne contenant aucune formule atomique de E. D'où le modèle minimal de $\mathcal{P}$ ne contient aucune formule atomique de E et E est valide dans $\mathcal{M}$.

C.Q.F.D.

EXEMPLE 14.2.1. Le programme suivant est censé calculer la fonction factorielle d'un entier non-négatif n. ($fact(x, y)$ signifie : "y est la factorielle de x" et $prod(x, y, z)$ signifie "$z = x \times y$").

En fait, le programme est incorrect car $n - 1$ a été remplacé par n dans le premier littéral de la deuxième clause ce qui entraîne la non-terminaison du programme.

$fact(0, 1)$.
$[\![fact(n, f) \leftarrow fact(n, m), prod(m, n, f) \ : n > 0]\!]$.
$prod(0, k, 0)$.
$[\![prod(k_1, k_2, r) \leftarrow prod(k_1', k_2, r') \ : k_1 > 0 \wedge k_1' = k_1 - 1 \wedge r = r' + k_2]\!]$.

La règle GMPL$^-$ génère la c-clause :

$$[\![\neg fact(n, m) \ : n > 0]\!]$$

Cette c-clause donne l'interprétation de $fact$. Sa signification est que le programme supposé calculer la fonction factorielle est incapable de le faire pour tout entier n strictement positif. Notons que ce fait négatif ne peut pas être déduit par la SLDNF-résolution, ni par la procédure ENF-Déduction (LUGIEZ, 1989) ni par négation constructive (CHAN, 1988) (elles ne terminent pas dans ce cas). ◇

14.3 Programmes logiques avec négation.

Dans cette section, nous étendons notre méthode aux programmes logiques avec négation, c'est-à-dire à des programmes logiques tels que le corps des c-clauses du programme peut contenir des négations de formules atomiques. L'introduction de la négation dans la Programmation en Logique est un problème important qui a fait l'objet d'un nombre considérable de travaux (voir (APT ET BOL, 1994) pour une synthèse complète de ce domaine). Autoriser la présence de négation dans le corps des c-clauses implique la définition d'une nouvelle sémantique aux programmes logiques ainsi étendus dans la mesure où ceux-ci n'ont plus nécessairement de modèle minimal unique. De nombreux auteurs ont proposé des approches différentes pour résoudre ce problème (APT ET BOL, 1994) La plupart des travaux dans ce domaine est inspirée par le concept de *formule de complétion* proposé par CLARK (CLARK, 1978). Il s'agit d'une formule du premier ordre définissant la sémantique du programme. Informellement, elle est obtenue en ajoutant au programme sa partie "seulement si". De nombreux travaux proposent de restreindre la classe des programmes logiques considérés et donnent une sémantique pour les programmes de cette classe (programmes *autorisés*, *stratifiables*, *localement stratifiables* etc.). SHEPHERDSON,

par exemple, propose de considérer la classe des programmes dits *hiérarchiques* et *autorisés* et lui donne une sémantique en utilisant la formule de complétion (SHEPHERDSON, 1984). Certains auteurs donnent aux programmes logiques avec négation une sémantique à trois valeurs de vérité (**vrai**, **faux** et **non défini**) (FITTING, 1985; GELDER ET AL., 1991).

Parmi les techniques utilisées pour l'inférence de faits négatifs, la plus connue et la plus utilisée est *la négation par l'échec* qui permet d'inférer $\neg P$ si P ne peut pas être prouvé après une exploration exhaustive de l'espace de recherche. Cette règle correspond à ce que l'on nomme habituellement l'*hypothèse du monde clos* qui consiste à supposer que le programme contient (potentiellement) toutes les informations sur le modèle, c'est-à-dire que tout ce qui ne peut pas être prouvé à partir du programme est faux (voir par exemple (SHEPHERDSON, 1985; SHEPHERDSON, 1989) pour plus de détails). Notons que l'hypothèse du monde clos, ainsi que la négation par l'échec, rend le raisonnement non monotone. La procédure de SLD-résolution enrichie par la règle de négation par l'échec est appelée la SLDNF-résolution. La négation par l'échec a cependant un double inconvénient : d'une part, elle dépend de la procédure de preuve utilisée et, d'autre part, elle ne permet pas de traiter des buts non fermés.

L'une des méthodes les plus intéressantes pour traiter le problème de la négation en Programmation Logique est la *négation constructive* proposée par (CHAN, 1988; STUCKEY, 1995) qui consiste, étant donné en but négatif $\neg b$, à rechercher les solutions de b et à nier les solutions obtenues. Cette approche est similaire à la méthode proposée par LUGIEZ, (LUGIEZ, 1989) et appelée ENF Déduction (pour **E**xtended **N**egation as **F**ailure). La ENF-Déduction utilise la formule de complétion de CLARK. Elle consiste à remplacer, dans la formule à évaluer, un littéral $P(\bar{t})$ par la formule de complétion $P^*(\bar{t})$ correspondante. La formule obtenue est alors simplifiée en utilisant les règles de résolution de contraintes de (COMON ET LESCANNE, 1989). Evidemment, l'évaluation peut ne pas se terminer, puisque la formule de complétion peut contenir des occurrences du prédicat P (éventuellement de façon indirecte).

Une alternative à la négation par l'échec est la *négation par inconsistance* (GABBAY ET SERGOT, 1986), qui introduit en Programmation en Logique une notion de la négation proche de celle utilisée de façon classique en Logique. Elle oblige le programmeur à définir *explicitement* les buts considérés comme faux et reste beaucoup moins utilisée que la négation par l'échec.

Dans cette section, nous nous proposons d'adapter les règles de RAMC aux programmes logiques avec négation et de comparer notre approche avec les méthodes existantes en Programmation Logique.

14.3.1 Programmes généraux avec contraintes

Nous donnerons en premier lieu une définition des programmes logiques avec négation également appelés *programmes généraux*. Informellement une c-clause générale est une c-clause (non nécessairement de Horn) dans laquelle on privilégie un littéral particulier appelé *tête* de la c-clause.

DÉFINITION 14.3.1. *Une* clause contrainte générale *(ou* c-clause générale*) est un triplet* $(\mathcal{H}, \mathcal{B}, \mathcal{X})$ *tel que :*

- $\mathcal{H}$ *est soit* $\bot$*, soit un littéral (la* tête *de la c-règle).*
- $\mathcal{B}$ *est une conjonction finie de littéraux* positifs ou négatifs *(le* corps *de la c-règle).*
- $\mathcal{X}$ *est une formule équationnelle* (contrainte).

Une c-clause générale est habituellement notée $[\![\mathcal{H} \leftarrow \mathcal{B} \ : \mathcal{X}]\!]$*. La c-clause* $[\![\mathcal{H} \ : \mathcal{X}]\!]$ *est appelée la* tête *de la c-clause et notée* $\mathcal{H}ead([\![\mathcal{H} \leftarrow \mathcal{B} \ : \mathcal{X}]\!])$ *(cette c-clause contient au plus un littéral).*

Une clause contrainte générale est dite définie *si la tête de la c-clause contient exactement un littéral positif. Un programme général* $\mathcal{P}$ *est un ensemble fini de c-clauses générales. L'ensemble des c-clauses non définies d'un programme* $\mathcal{P}$ *est noté* NONDEF($\mathcal{P}$). ◇

14.3.2 Programmes généraux et RAMC

Nous définissons dans cette section de nouveaux calculs sur les programmes généraux en adaptant les règles de RAMC. Nous pouvons étendre de façon immédiate les règles de RAMC agissant sur les c-clauses aux c-clauses générales. Plus précisément, nous introduisons les règles suivantes :

affaiblissement :

$$\frac{[\![\neg P(\overline{s}) \leftarrow c \ : \mathcal{X}]\!]}{[\![\leftarrow P(\overline{s}) \wedge c \ : \mathcal{X}]\!]}$$

c-résolution :

$$\frac{[\![P(\overline{t}_1) \leftarrow a \ : \mathcal{X}]\!] \qquad [\![c \leftarrow P(\overline{t}_2) \wedge b \ : \mathcal{Y}]\!]}{r : [\![c \leftarrow a \wedge b \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2]\!]}$$

c-factorisation :

$$\frac{[\![a \leftarrow P(\overline{t}_1) \wedge P(\overline{t}_2) \wedge b \ : \mathcal{X}]\!]}{r : [\![a \leftarrow P(\overline{t}_1) \wedge b \ : \mathcal{X} \wedge \overline{t}_1 = \overline{t}_2]\!]}$$

c-disrésolution 1 :

$$\frac{c_1 \ : [\![P(\overline{t}_1) \leftarrow a \ : \mathcal{X}]\!] \qquad c_2 \ : [\![c \leftarrow P(\overline{t}_2) \wedge b \ : \mathcal{Y}]\!]}{c_3 \ : [\![P(\overline{t}_1) \leftarrow a \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2]\!] \qquad c_4 \ : [\![P(\overline{t}_1) \leftarrow a \ : \mathcal{X} \wedge \forall \overline{y}.\neg \mathcal{Y} \vee \overline{t}_1 \neq \overline{t}_2]\!]}$$

où : $\overline{y} = \mathcal{V}ar(c_2)$

c-disrésolution 2 :

$$\frac{c_1 \ : [\![P(\overline{t}_1) \leftarrow a \ : \mathcal{X}]\!] \qquad c_2 \ : [\![c \leftarrow P(\overline{t}_2) \wedge b \ : \mathcal{Y}]\!]}{c_3 \ : [\![c \leftarrow P(\overline{t}_2) \wedge b \ : \mathcal{Y} \wedge \mathcal{X} \wedge \overline{t}_1 = \overline{t}_2]\!] \qquad c_4 \ : [\![c \leftarrow P(\overline{t}_2) \wedge b \ : \mathcal{Y} \wedge \forall \overline{y}.\neg \mathcal{X} \vee \overline{t}_1 \neq \overline{t}_2]\!]}$$

où : $\overline{y} = \mathcal{V}ar(c_2)$

c-disfactorisation :

$$\frac{[\![a \leftarrow P(\overline{t}_1) \wedge P(\overline{t}_2) \wedge b \ : \mathcal{X}]\!]}{[\![a \leftarrow P(\overline{t}_1) \wedge P(\overline{t}_2) \wedge b \ : \mathcal{X} \wedge \overline{t}_1 \neq \overline{t}_2]\!]}$$

c-dissubsomption 1 :

$$\frac{c_1 \ : [\![a(\overline{s}) \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{s_i}) \ : \mathcal{X}]\!] \qquad c_2 \ : [\![a(\overline{t}) \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{t_i}) \wedge b \ : \mathcal{Y}]\!]}{c_3 \ : [\![a(\overline{t}) \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{s_i}) \wedge b \ : \mathcal{Y} \wedge \forall \overline{x}[\neg \mathcal{X} \vee \bigvee_{i=1}^{n} \overline{s}_i \neq \overline{t}_i \vee \overline{s} \neq \overline{t}]]\!]}$$

avec $\overline{x} = \mathcal{V}ar(c_1)$

c-dissubsomption 2 :

$$\frac{c_1 \ : [\![\bot \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{s_i}) \ : \mathcal{X}]\!] \qquad c_2 \ : [\![a \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{t_i}) \wedge b \ : \mathcal{Y}]\!]}{c_3 \ : [\![a \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{t_i}) \wedge b \ : \mathcal{Y} \wedge \forall \overline{x}[\neg \mathcal{X} \vee \bigvee_{i=1}^{n} \overline{s}_i \neq \overline{t}_i]]\!]}$$

avec $\overline{x} = \mathcal{V}ar(c_1)$

c-dissubsomption 3 :

$$\frac{c_1 \ : [\![b(\overline{t}) \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{s_i}) \ : \mathcal{X}]\!] \qquad c_2 \ : [\![a \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{t_i}) \wedge \neg b(\overline{s}) \wedge c \ : \mathcal{Y}]\!]}{c_3 \ : [\![a \leftarrow \bigwedge_{i=1}^{n} l_i(\overline{s_i}) \wedge \neg b(\overline{s}) \wedge c \ : \mathcal{Y} \wedge \forall \overline{x} [\neg \mathcal{X} \vee \bigvee_{i=1}^{n} \overline{s}_i \neq \overline{t}_i \vee \overline{s} \neq \overline{t}]]\!]}$$

avec $\overline{x} = \mathcal{V}ar(c_1)$

c-distautologie 1 :

$$\frac{c_1 \ : [\![a \leftarrow P(\overline{t}_1) \wedge \neg P(\overline{t}_2) \wedge a \ : \mathcal{X}]\!]}{c_2 \ : [\![a \leftarrow P(\overline{t}_1) \wedge \neg P(\overline{t}_2) \wedge a \ : \mathcal{X} \wedge \overline{t}_1 \neq \overline{t}_2]\!]}$$

c-distautologie 2 :

$$\frac{c_1 \ : [\![P(\overline{t}_1) \leftarrow P(\overline{t}_2) \wedge a \ : \mathcal{X}]\!]}{c_2 \ : [\![P(\overline{t}_1) \leftarrow P(\overline{t}_2) \wedge a \ : \mathcal{X} \wedge \overline{t}_1 \neq \overline{t}_2]\!]}$$

Nous noterons RAMC$_{\mathrm{SC}}$ (pour **S**émantique de **C**lark), RAMC$_{\mathrm{SC3}}$ et RAMC$_{\mathrm{MBF}}$ (pour **M**odèle **B**ien **F**ondé) les calculs respectivement composés des règles :

- RAMC$_{\mathrm{SC}}$: affaiblissement, c-résolution, c-factorisation, c-disrésolution 1, c-disrésolution 2, c-disfactorisation, c-dissubsomption 1, 2, 3, c-distautologie 1, GPL$^-$.
- RAMC$_{\mathrm{SC3}}$: affaiblissement, c-résolution, c-factorisation, c-disrésolution 1, c-disrésolution 2, c-disfactorisation, c-dissubsomption 1, c-dissubsomption 3 unitaire, c-distautologie 1, GPL$^-$.
- RAMC$_{\mathrm{MBF}}$: affaiblissement, c-résolution, c-factorisation, c-disrésolution 1, c-disrésolution 2, c-disfactorisation, c-dissubsomption 1, c-dissubsomption 3 unitaire, c-distautologie 1,2, GMPL$^-$.

DÉFINITION 14.3.2. *Une* sémantique S *pour un programme* $\mathcal{P}$ *est une fonction associant à* $\mathcal{P}$ *un ensemble de modèles partiels* $\mathcal{M}_{\mathcal{P}}(S)$. *Soit une règle de calcul* ρ *sur les programmes logiques.* ρ *est dite* adéquate par rapport à S *(ou* S*-adéquate) si et seulement si pour tout* $\mathcal{P} \rightarrow_\rho \mathcal{P}'$, $\mathcal{M}_{\mathcal{P}}(S) = \mathcal{M}_{\mathcal{P}'}(S)$. *De même, un calcul* $\mathcal{R}$ *sera dit* S*-adéquat si et seulement si toute règle* ρ *de* $\mathcal{R}$ *est* S*-adéquate. Deux programmes* $\mathcal{P}$ *et* $\mathcal{P}'$ *sont dits* S-équivalents *si et seulement si :* $\mathcal{M}_S(\mathcal{P}) = \mathcal{M}_S(\mathcal{P}')$. ◇

Dans les sections suivantes, nous prouvons l'adéquation de ces calculs respectivement par rapport à trois sémantiques particulières des programmes généraux: la formule de complétion de CLARK (avec 2 ou 3 valeurs de vérité) et la sémantique des modèles bien fondés.

14.3.3 Adéquation par rapport à la formule de complétion de Clark

Interprétations à deux valeurs de vérité

DÉFINITION 14.3.3. *Soit* $\mathcal{P}$ *un programme logique et* p *un symbole de prédicat. On note* $p^*(\overline{x})$ *la formule :*

$$\bigvee_{[\![p(\overline{t}) \leftarrow \mathcal{B} \ : \mathcal{X}]\!] \in \mathcal{P}} \exists \overline{y}. \overline{x} = \overline{t} \wedge \mathcal{X} \wedge \mathcal{B} \ (\text{où } \overline{y} = \mathcal{V}ar([\![p(\overline{t}) \leftarrow \mathcal{B} \ : \mathcal{X}]\!]))$$

On note $\mathcal{P}^*$ la formule :

$$\bigwedge_{p \in \Omega} \forall \overline{x}.p(\overline{x}) \leftarrow p^*(\overline{x}) \wedge \textsc{NonDef}(\mathcal{P})$$

$\diamond$

Cette formule permet de donner une sémantique aux programmes généraux. Plus précisément, nous considérerons qu'une interprétation totale de Herbrand $\mathcal{I}$ sera un modèle d'un programme général $\mathcal{P}$ si et seulement si $\mathcal{I} \models \mathcal{P}^*$. Le principal problème de cette définition réside dans l'existence et l'unicité d'un modèle de $\mathcal{P}^*$: d'une part, ce modèle n'est pas nécessairement unique, et d'autre part, $\mathcal{P}^*$ n'est pas nécessairement satisfaisable (même si $\mathcal{P}$ ne contient que des c-clauses définies).

La sémantique de Clark sera notée SC dans la suite.

Remarque. Il est clair que la sémantique de Clark dépend uniquement de l'ensemble $\mathcal{S}(\mathcal{P})$ et non du programme $\mathcal{P}$ lui même.

Interprétations à trois valeurs de vérité

Dans (Fitting, 1985), Fitting propose de considérer des interprétations à trois valeurs de vérité au lieu de deux dans la sémantique de la formule de complétion. Le but de ce travail est de prendre en compte le cas où un littéral L donné diverge, ce qui est réalisé par l'ajout d'une troisième valeur de vérité. Avec cette approche, on associe à chaque programme un modèle unique, mais ce modèle est *partiel* et non plus *total*. La sémantique peut être également définie à l'aide d'un opérateur de point fixe : l'opérateur de conséquence immédiate (Apt et Bol, 1994).

Définition 14.3.4. *Soit $\mathcal{P}$ un programme logique et $\mathcal{I}$ une interprétation partielle. L'*opérateur de conséquence immédiate $\mathcal{T}3_{\mathcal{P}}$ *est défini comme suit :*

$$\mathcal{T}3_{\mathcal{P}}(\mathcal{I}) = (\mathcal{T}, \mathcal{F})$$

où :

- $\mathcal{T} = \{\mathcal{H}\sigma / [\![\mathcal{H} \leftarrow \mathcal{B} \ : \mathcal{X}]\!], \sigma \in \mathcal{S}(\mathcal{X}), \mathcal{I} \models \mathcal{B}\sigma\}$.
- $\mathcal{F} = \{b / \forall [\![\mathcal{H} \leftarrow \mathcal{B} \ : \mathcal{X}]\!] \in \mathcal{P}, \sigma \in \mathcal{S}(\mathcal{X}), \mathcal{I} \models \neg\mathcal{B}\sigma \vee b \neq \mathcal{H}\sigma\}$.

$\diamond$

$\mathcal{T}3_{\mathcal{P}}$ est monotone donc a un plus petit point fixe $\mathcal{T}3_{\mathcal{P}} \uparrow \alpha$. Si $\mathcal{T}3_{\mathcal{P}} \uparrow \alpha$ est consistant (i.e. ne contient pas $\bot$ et ne contient pas deux littéraux complémentaires), $\mathcal{T}3_{\mathcal{P}} \uparrow^{\alpha}$ est un modèle (à trois valeurs de vérité) de $\mathcal{P}^*$ (Apt et Bol, 1994). $\mathcal{T}3_{\mathcal{P}}$ n'est pas, en général, continu, donc le point fixe pourra ne pas être égal à la première puissance transfinie de $\mathcal{T}3_{\mathcal{P}}$.

Si $\mathcal{T}3_{\mathcal{P}} \uparrow^{\omega}$ est consistant, $\mathcal{P}$ est dit SC_3-satisfaisable, et

$$\mathcal{M}_{\mathcal{P}}(SC_3) = \{\mathcal{T}3_{\mathcal{P}} \uparrow^{\omega}\}.$$

Adéquation

Théorème 14.3.1. *Le calcul* $\textsc{Ramc}_{SC3}$ *est adéquat par rapport à* SC_3.

Preuve. Il est facile de voir que les règles **c-factorisation, c-disrésolution, c-disfactorisation, c-dissubsomption 1, c-distautologie 1** ne modifient pas $\mathcal{T}3_{\mathcal{P}}$. Il suffit donc de considérer les règles Gpl$^-$, c-résolution, affaiblissement et dissubsomption 3 unitaire.

- **GPL$^-$**. Soit $\mathcal{P}$ un programme, $[\![\neg L \ : \mathcal{X}]\!]$ un c-littéral déduit de $\mathcal{P}$ par Gpl$^-$. Soit σ une solution de $\mathcal{X}$. Par définition, pour toute tête $[\![\mathcal{H} \ : \mathcal{Y}]\!]$ d'une clause de $\mathcal{P}$ et pour tout $\theta \in \mathcal{S}(\mathcal{Y})$, on a $\mathcal{H}\theta \neq L\sigma$. Par conséquent, $\neg L\sigma$ appartient à $\mathcal{T}3_{\mathcal{P}}(\emptyset)$.

- **C-résolution.** Soit $\mathcal{P}$ un programme et $r \ : [\![c \leftarrow a \wedge b \ : \mathcal{X} \wedge \mathcal{Y} \wedge \overline{t}_1 = \overline{t}_2]\!]$ un résolvent de $[\![P(\overline{t}_1) \leftarrow a \ : \mathcal{X}]\!]$ et $[\![c \leftarrow P(\overline{t}_2) \wedge b \ : \mathcal{Y}]\!]$. On note $\mathcal{P}'$ le programme $\mathcal{P} \cup \{r\}$. Soit $\mathcal{I}$ une interprétation. Il est facile de vérifier que $\mathcal{T}3_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{T}3_{\mathcal{P}'}(\mathcal{I})$ et que $\mathcal{T}3_{\mathcal{P}'}(\mathcal{I}) \subseteq \mathcal{T}3^2_{\mathcal{P}}(\mathcal{I})$. Donc les premières puissances transfinies de $\mathcal{T}3_{\mathcal{P}}$ et $\mathcal{T}3_{\mathcal{P}'}$ sont identiques.
- **Affaiblissement.** Soit $\mathcal{P}$ un programme et $[\![\leftarrow P(\overline{s}) \wedge c \ : \mathcal{X}]\!]$ une c-clause déduite d'une c-clause $[\![\neg P(\overline{s}) \leftarrow c \ : \mathcal{X}]\!]$ de $\mathcal{P}$ par affaiblissement. Alors si $\mathcal{P}$ est satisfaisable, $P(\overline{s}) \wedge c$ n'est pas valide dans le modèle de $\mathcal{P}$, donc la règle ne modifie pas les puissances successives de $\mathcal{T}3_{\mathcal{P}}$. Si $\mathcal{P}$ est insatisfaisable, alors $\mathcal{P} \cup \{[\![\leftarrow P(\overline{s}) \wedge c \ : \mathcal{X}]\!]\}$ l'est également.
- **Dissubsomption 3 unitaire.** Soit $\mathcal{P}$ un programme et $\mathcal{P}'$ un programme déduit de $\mathcal{P}$ par **dissubsomption 3 unitaire** entre $c_1 \ : [\![b(\overline{t}) \leftarrow \ : \mathcal{X}]\!]$ et $c_2 \ : [\![a \leftarrow \neg b(\overline{s}) \wedge c \ : \mathcal{Y}]\!]$. Soit $\sigma \in \mathcal{S}(\mathcal{Y} \wedge \mathcal{X} \wedge \overline{s} = \overline{t})$. Il est clair que si $b(\overline{t})\sigma \in \mathcal{I}$ et si $\neg b(\overline{t})\sigma \notin \mathcal{I}$, alors $\mathcal{T}3_{\mathcal{P}}(\mathcal{I}) = \mathcal{T}3_{\mathcal{P}'}(\mathcal{I})$. De plus, on a $\mathcal{T}3_{\mathcal{P}}(\emptyset) = \mathcal{T}3_{\mathcal{P}'}(\emptyset)$ et $b(\overline{t})\sigma \in \mathcal{T}3_{\mathcal{P}}(\emptyset)$. Par conséquent, par induction sur n soit $\mathcal{P}$ et $\mathcal{P}'$ sont insatisfaisables, soit $\mathcal{T}3^n_{\mathcal{P}}(\emptyset) = \mathcal{T}3^n_{\mathcal{P}'}(\emptyset)$.

C.Q.F.D.

THÉORÈME 14.3.2. *Le calcul* RAMC$_{\text{SC}}$ *est adéquat par rapport à SC.*

PREUVE. Toute règle SC$_3$-adéquate est SC-adéquate, donc il suffit de considérer les règles **dissubsomption 2,3**.

- Montrons que la règle de **c-dissubsomption 3** est SC-adéquate. Soit $\mathcal{P}'$ déduit de $\mathcal{P}$ par application de cette règle (voir la définition de la règle pour les notations). Soit p un symbole de prédicat et $\mathcal{I}$ une interprétation. On a : $\mathcal{S}(\mathcal{P}') \subseteq \mathcal{S}(\mathcal{P})$ donc $p_{\mathcal{P}'}(\overline{x})\sigma \models p_{\mathcal{P}}(\overline{x})\sigma$. De plus, si $\mathcal{I} \models \mathcal{P}^*$ ou $\mathcal{I} \models \mathcal{P}'^*$, alors pour toute substitution $\sigma : \mathcal{I} \models c_1\sigma$ donc $\mathcal{I} \not\models (\bigwedge_{i=1}^{n} l_i(\overline{t_i}) \wedge \neg b(\overline{s}))\sigma$, d'où $\mathcal{I} \models (p_{\mathcal{P}}(\overline{x})\sigma - \ p_{\mathcal{P}'}(\overline{x})\sigma)$.
- **c-dissubsomption 2** : la preuve est similaire.

C.Q.F.D.

14.3.4 Adéquation par rapport à la sémantique des modèles bien fondés

Intéressons nous maintenant à une nouvelle sémantique des programmes généraux, appelée *sémantique des modèles bien fondés.*

La sémantique des modèles bien fondés (que nous désignerons pour simplifier MBF dans la suite de ce chapitre) est définie comme point fixe d'un certain opérateur monotone $\mathcal{W}_P$. Le modèle partiel associé au programme $\mathcal{P}$ n'est pas nécessairement récursivement énumérable.

Définissons tout d'abord la notion d'*ensemble non fondé*[1].

DÉFINITION 14.3.5. *Un ensemble de littéraux positifs fermés U est appelé* ensemble non fondé *d'un programme général $\mathcal{P}$ par rapport à une interprétation partielle $\mathcal{I}$ si et seulement si pour tout atome $p(\overline{t}) \in U$, pour toute c-règle $C \ : [\![p(\overline{s}) \leftarrow \mathcal{B} \ : \mathcal{X}]\!]$ et pour toute solution σ de $\mathcal{X}$, telle que $\overline{t}\sigma = \overline{s}\sigma$ au moins une des conditions suivantes est vérifiée :*

1. *Il existe un littéral de $\mathcal{B}\sigma$ falsifié par $\mathcal{I}$.*
2. *Il existe un littéral* positif *de $\mathcal{B}\sigma$ appartenant à U.*

1. nous traduisons ainsi l'anglais *unfounded set.*

Le plus grand ensemble non fondé de P par rapport à une interprétation partielle $\mathcal{I}$ (notée $\mathcal{U}_P(\mathcal{I})$) est l'union de tous les ensembles non fondés de P par rapport à $\mathcal{I}$ (il est facile de voir que $\mathcal{U}_P(\mathcal{I})$ est encore un ensemble non fondé de P par rapport à $\mathcal{I}$).

◇

DÉFINITION 14.3.6. *L'opérateur $\mathcal{W}_P$ de $\mathcal{IP}$ dans $\mathcal{IP}$ est défini par*

$$\mathcal{W}_P(\mathcal{I}) = \mathcal{T}_P(\mathcal{I}) \cup \neg\mathcal{U}_P(\mathcal{I})$$

où $\mathcal{T}_P$ est l'opérateur défini à la section 14.1. ◇

THÉORÈME 14.3.3. *$\mathcal{W}_P$ est monotone.*

D'après 14.1.2, $\mathcal{W}_P$ admet un plus petit point fixe $\mathcal{W}_P \uparrow^\alpha$, α étant un ordinal, *non nécessairement égal à ω* (car $\mathcal{W}_P$ n'est pas continu).

DÉFINITION 14.3.7. *Soit un programme $\mathcal{P}$. Soit $\mathcal{W}_P \uparrow^\alpha$ le plus petit point fixe de $\mathcal{W}_P$. Si $\mathcal{W}_P \uparrow^\alpha$ est consistant, alors $\mathcal{P}$ est dit MBF-satisfaisable et le modèle bien fondé (en abrégé MBF) de $\mathcal{P}$ est le plus petit point fixe de l'opérateur $\mathcal{W}_P$.* ◇

THÉORÈME 14.3.4. *RAMC$_{\text{MBF}}$ est adéquat par rapport à la MBF.*

PREUVE. Toute règle SC_3-adéquate est MBF-adéquate, donc il suffit de considérer les règles GMPL$^-$ et **distautologie 2**.

- Soit $\mathcal{M}$ le plus petit point fixe de $\mathcal{W}_\mathcal{P}$. Il suffit de montrer que pour tout ensemble E déduit de P par la règle GMPL$^-$, $E \subseteq \mathcal{M}$.
  Pour cela, il suffit de prouver que $\neg E$ est un ensemble non fondé de P par rapport à $\mathcal{M}$ (en fait par rapport à une interprétation quelconque).
  En effet, supposons qu'il existe une c-clause $[\![\mathcal{H} \leftarrow \mathcal{B} \ : \mathcal{X}]\!] \in \mathcal{P}$ et une substitution $\sigma \in \mathcal{S}(\mathcal{X})$ telles que $\neg\mathcal{H}\sigma \in E$, alors par stabilité de E par $R'_\mathcal{P}$ il doit exister un littéral (positif) $L \in E$ tel qu'il existe un littéral b appartenant à $\mathcal{B}\sigma$ telle que $L \succ b$. D'où E est un ensemble non fondé de $\mathcal{P}$ par rapport à $\emptyset$.
- La règle de **distautologie 2** peut être simulée par l'application de la règle de **dissubsomption 1** entre c_1 et la c-clause $C \ : P(\overline{t}_1) \leftarrow P(\overline{t}_1)$. Il suffit donc de prouver que pour tout programme $\mathcal{P}$. $\mathcal{P}$ et $\mathcal{P}' = \mathcal{P} \cup \{C\}$ sont MBF-équivalents. Or on a de façon évidente : $\mathcal{T}_\mathcal{P} = \mathcal{T}_{\mathcal{P}'}$ et $\mathcal{U}_\mathcal{P} = \mathcal{U}_{\mathcal{P}'}$. D'où $\mathcal{P}$ et $\mathcal{P}$' sont MBF-équivalents.

C.Q.F.D.

REMARQUE. Il est clair que la règle dissubsomption 3 *n'est pas adéquate* par rapport à la sémantique MBF. De même la règle dissubsomption 3 *n'est pas adéquate* par rapport à la sémantique SC_3.

14.3.5 Complétude

Nous montrons ci-dessous que la procédure contenant les règles

$$\{\text{GPL}^-, \text{c-résolution, c-dissubsomption 3 unitaire}\}$$

est complète par rapport à $\mathcal{M}_\mathcal{P}(SC_3)$.

THÉORÈME 14.3.5. *Soit un programme $\mathcal{P}$. Soit b tel que $\neg b \in \mathcal{T}3_\mathcal{P} \uparrow^n$. Alors il existe une RAMC$_{\text{PL1}}$-dérivation contenant n applications des règles c-résolution ou GPL$^-$ et générant une c-clause C telle que $C \leq_{dissub} \neg b$.*

PREUVE. La preuve est par induction sur n.

- $n = 0$. Considérons deux cas, suivant le signe de b.
  1. Si b est négatif, alors il existe une c-clause unitaire C telle que $b \cap \neg C \neq \emptyset$, i.e. telle que $C \leq_{dissub} \neg b$.
  2. Si b est positif, alors pour toute tête de c-clause H on doit avoir $\neg\neg b \cap H = \emptyset$. Donc la règle GPL$^-$ génère la c-clause $\neg b$.
- $n + 1$. Là encore, nous distinguons deux cas.
  1. Si b est négatif, alors il existe une c-clause $[\![\mathcal{H} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n \ : \mathcal{X}]\!]$ et une substitution σ telle que $b = \mathcal{H}\sigma$ et telle que tout littéral $\neg\mathcal{B}_i\sigma$ $(1 \leq i \leq n)$ appartient à $\mathcal{T}3 \uparrow^n$. D'où, par hypothèse d'induction, il existe une dérivation de $C_1, \ldots, C_n$, où pour tout $i \in [1..n]$, $C_i \succ \mathcal{B}_i\sigma$. En appliquant la règle de c-résolution entre $[\![\mathcal{H} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n \ : \mathcal{X}]\!]$ et $C_1, \ldots, C_n$, on obtient une c-clause C de la forme $[\![\mathcal{H} \ : \mathcal{X} \wedge \mathcal{Y}]\!]$ telle que σ soit solution de $\mathcal{X} \wedge \mathcal{Y}$ et $\mathcal{H}\sigma = b$, i.e. $[\![\mathcal{H} \ : \mathcal{X} \wedge \mathcal{Y}]\!] \leq_{dissub} b$.
  2. Si b est positif, alors pour toute clause $[\![\mathcal{H} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n \ : \mathcal{X}]\!]$, telle qu'il existe $\sigma \in \mathcal{S}(\mathcal{X})$ telle que $\mathcal{H}\sigma = b$, $\neg\mathcal{B}_i\sigma \in \mathcal{T}3 \uparrow^n$ $(1 \leq i \leq n)$. Par hypothèse d'induction, il existe une dérivation générant des c-clauses $C_1, \ldots, C_n$ telles que $C_i \succ \mathcal{B}_i\sigma$ $(1 \leq i \leq n)$. En appliquant la règle de c-dissubsomption 3 unitaire sur $[\![\mathcal{H} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n \ : \mathcal{X}]\!]$ et $C_1, \ldots, C_n$, on transforme la c-clause $[\![\mathcal{H} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n \ : \mathcal{X}]\!]$ en $[\![\mathcal{H} \leftarrow \mathcal{B}_1 \wedge \ldots \wedge \mathcal{B}_n \ : \mathcal{X} \wedge \mathcal{Y}]\!]$, telle que $\sigma \notin \mathcal{S}(\mathcal{Y})$. En appliquant la règle GPL$^-$ on génère ensuite le littéral $\neg b$.

C.Q.F.D.

14.4 Evaluation de buts complexes

Nous nous intéressons dans cette section à l'évaluation de buts complexes, incluant les connectifs logiques $\neg, \vee$ et les quantificateurs universels ou existentiels. Une façon de procéder pour évaluer une formule du premier ordre consiste à traduire la formule du premier ordre considérée en un ensemble de c-clauses générales associé à un but b et à évaluer b (cela est toujours possible à cause de la présence de négations dans le corps des règles, qui permettent d'exprimer des imbrications de quantificateurs quelconques). Nous proposons ici une méthode différente utilisant les formules ϕ^- et ϕ^+ définies au chapitre 5. L'un des avantages de cette méthode est que l'évaluation de la formule est indépendante du calcul du modèle. Elle ne nécessite pas de transformation de la formule, ne dépend pas de la sémantique utilisée, ni de la procédure de preuve et pourra donc être utilisée sur toutes les classes de programmes et avec tous les calculs précédemment introduits. L'idée est d'évaluer la formule considérée dans le modèle partiel du programme. Ce modèle partiel est généré en utilisant n'importe quel calcul adéquat par rapport à la sémantique considérée.

Plus précisément, on peut distinguer deux phases.

- Utiliser une procédure quelconque pour énumérer le modèle partiel du programme (ou une partie de celui-ci s'il n'est pas récursivement énumérable).
- Evaluer *durant la recherche du modèle* la formule-but dans le modèle partiel. Les résultats du chapitre 5 fournissent un algorithme incomplet pour trouver certaines des solutions d'une formule donnée dans un modèle partiel.

Plus précisément, nous définissons une procédure **Extended_Interpreter** paramètrée par un calcul sur les programmes logiques (figure 14.1).

THÉORÈME 14.4.1. *Si $\mathcal{C}$ est adéquat par rapport à une sémantique S alors Sol est un ensemble de solutions de $\mathcal{F}$ dans le(s) modèle(s) correspondant(s) à la sémantique S de P.*

Procedure Extended_Interpreter

INPUT :
Une formule $\mathcal{F}$ du premier ordre.
Un programme $\mathcal{P}$
Un calcul $\mathcal{C}$ sur les programmes.
OUTPUT :
Un ensemble de substitutions σ
begin
$Sol \leftarrow \emptyset$
$\mathcal{M} \leftarrow \emptyset$
while $(\phi^+_{\mathcal{M}}(\mathcal{F}) \vee \phi^-_{\mathcal{M}}(\mathcal{F})) \neq \top$
$\mathcal{P} \leftarrow \mathcal{C}(\mathcal{P})$
$\mathcal{M} \leftarrow \mathcal{M} \cup Unit(\mathcal{P})$
$Sol \leftarrow Sol \cup \phi^+_{\mathcal{M}}(\mathcal{F})$
$\mathcal{F} = \text{Normalize}_{\mathcal{M}}(\mathcal{F})$
return(Sol)
end

FIG. 14.1 - : *La procédure* ***Extended_Interpreter***

PREUVE. La preuve est immédiate. C.Q.F.D.

THÉORÈME 14.4.2. *Soit $\mathcal{I}$ une interprétation partielle et $\mathcal{F}$ une formule. Alors pour toute substitution σ telle que $\mathcal{I}$ est un modèle à trois valeurs de vérité de $\mathcal{F}$, on a $\sigma \in \mathcal{S}(\phi^+_{\mathcal{I}}(\mathcal{F}))$.*

PREUVE. La preuve est immédiate, par induction structurelle sur l'ensemble de formules. C.Q.F.D.

EXEMPLE 14.4.1. Soit $\mathcal{P}$ le programme logique suivant (c-clauses de Horn) :
$append([a|x], y, [a|z]) \leftarrow append(x, y, z)$.
$append(nil, y, y)$.

Considérons la formule

$$\forall y, z. \neg append([z], y, x)$$

La procédure **Extended_Interpreter** utilisant le calcul RAMC_{PL1} donne :

$3\ \neg[\![append([a|x], y, z) \ : \forall z'. z \neq [a|z']]\!]$ (GMPL$^-$)
$4\ [\![append([a], y, [a|y]) \ : \top]\!]$ (résolution, 1, 2)

A ce stade le modèle partiel généré par le programme (noté $\mathcal{M}$) est le suivant.
$append(x, y, z)$ if $(x = [] \wedge y = z) \vee (\exists a. x = [a] \wedge z = [a|y])$

$\neg append(x, y, z)$ if $\exists a, x'. (x = [a|x'] \wedge \forall z'. z \neq [a|z'])$

La formule $\phi^+_{\mathcal{M}}(\mathcal{F})$ correspondante est :

$$\forall y, z. \exists x', a.$$
$$[z] = [a|x'] \wedge \forall z'. x \neq [a|z']$$

La procédure de résolution de contraintes (voir (Comon et Lescanne, 1989), chapitre 2 et annexe A) transforme cette formule en

$$\phi^+_{\mathcal{M}}(\forall y, z.\neg append([z], y, x)) \equiv (x = [])$$

La procédure **Extended_Interpreter** génère la solution $x = []$. Il s'agit maintenant de vérifier que cette substitution est la seule solution possible. Pour cela, on calcule la formule :

$\phi^-_{\mathcal{M}}(\forall y, z.\neg append([z], y, x))$
$\equiv (\exists y, z.(\exists y_1.[z] = [] \wedge y = y_1 \wedge x = y_1)$
$\quad \vee (\exists a_1, y_1.[z] = [a_1] \wedge y = y_1 \wedge x = [a_1|y_1]))$
i.e. (par transformation de contraintes) :

$$\phi^-_{\mathcal{M}}(\forall y, z.\neg append([z], y, x)) \equiv (\exists a_1, y_1.x = [a_1|y_1])$$

On calcule ensuite la disjonction :

$$\phi^-_{\mathcal{M}}(\mathcal{F}) \vee \phi^+_{\mathcal{M}}(\mathcal{F}) \equiv (x = [] \vee \exists a_1, y_1.x = [a_1|y_1])$$

On obtient (par résolution des contraintes) :

$$\phi^-_{\mathcal{M}}(\mathcal{F}) \vee \phi^+_{\mathcal{M}}(\mathcal{F}) \equiv \top$$

Ceci montre que $\mathcal{M}$ et $\mathcal{F}$ vérifient les conditions d'arrêt de la procédure **Extended_Interpreter**. Par conséquent, la procédure s'arrête et retourne :

$$\{x = []\}.$$

$\diamond$

Exemple 14.4.2. Le deuxième exemple que nous proposons est tiré de (Lugiez, 1989). Comme l'indique (Lugiez, 1989), ni la SLDNF-résolution, ni la plupart des autres approches ne peuvent traiter cet exemple. La ENF-déduction proposée par D. Lugiez est capable d'énumérer toutes les solutions du programme. Cependant elle ne se termine pas, car il existe un nombre infini de solutions. Nous montrons ci-dessous que notre méthode permet de déduire *directement* le fait négatif : $\forall x \neq 0.\neg p(x)$, au lieu d'énumérer l'ensemble : $p(s(0)), p(s(s(0))), \ldots$

Soit $\Sigma = \{0, s\}$ et le programme $\mathcal{P}$ défini par les c-clauses :
$r(s(s(x))) \leftarrow r(x)$
$p(x) \leftarrow q(x, y)$
$p(x) \leftarrow r(x)$
$q(0, 0)$

Soit le but : $\neg p(x)$. Le problème est de trouver les solutions de ce but dans le modèle du programme (ici le modèle minimal, puisqu'il s'agit d'un programme logique classique), i.e. de trouver les termes x tels que $p(x)$ est faux dans le modèle de $\mathcal{P}$.

Notre méthode donne :
1 $[\![r(s(s(x))) \vee \neg r(x) \ : \top]\!]$
2 $[\![p(x) \vee \neg q(x, y) \ : \top]\!]$
3 $[\![p(x) \vee \neg r(x) \ : \top]\!]$
4 $[\![q(0, 0) \ : \top]\!]$
5 $[\![\neg q(x, y) \ : x \neq 0 \vee y \neq 0]\!]$ (Gmpl$^-$)
6 $[\![p(0) \vee \neg q(0, 0) \ : \top]\!]$ (dissubsomption,5,2)
7 $[\![p(0) \vee \neg q(0, 0) \ : \bot]\!]$ (disrésolution,4,6)
8 $[\![p(0) \ : \top]\!]$ (résolution,4,6)
9 $[\![\neg r(x) \ : \top]\!]$ (Gmpl$^-$)
10 $[\![\neg p(x) \ : x \neq 0]\!]$ (Gmpl$^-$)

La c-clause 10 montre que $p(x)$ est faux pour tout x différent de 0. La c-clause 8 montre que $p(0)$ est vrai. Par conséquent, ces deux c-clauses donnent l'interprétation du prédicat p et fournissent les solutions de la formule $\neg p(x)$.

Plus précisément, on a :

$$\phi^{+}_{\mathcal{M}}(\mathcal{F}) = x \neq 0$$

et :

$$\phi^{-}_{\mathcal{M}}(\mathcal{F}) = x = 0$$

REMARQUE. Notre méthode, au contraire de la procédure de ENF-déduction proposée par (LUGIEZ, 1989) n'énumère pas toutes les solutions mais s'arrête et retourne directement l'ensemble des solutions de la formule $\neg p(x)$.

◇

14.5 Application: détection d'erreurs dans des programmes

La manière la plus courante pour vérifier qu'un programme logique $\mathcal{P}$ satisfait une spécification formelle (exprimée par une formule du premier ordre $\mathcal{F}$) est de prouver la *terminaison*, la *correction* et la *complétude* du programme (voir par exemple (HOGGER, 1984)). Les notions de correction et de complétude ont ici le même sens qu'en Logique ou en Déduction Automatique : un programme est dit *correct* si tous les faits déductibles du programme sont valides dans la spécification (i.e. si : $\mathcal{P} \models \mathcal{F}$) et *complet* si tous les faits valides dans la spécification peuvent être déduits du programme (i.e. si $\mathcal{F} \models \mathcal{P}$). Selon HOGGER (HOGGER, 1984), ces critères de vérification ont été pour la première fois publiés dans (CLARK ET TARNLUND, 1977) (voir également (FLENER, 1995)).

Une question aussi importante (et qui a très peu été étudiée) est : comment vérifier qu'un programme *ne satisfait pas* ses spécifications ? Si l'on veut rester dans ce cadre il s'agit de montrer qu'il est impossible de prouver que le programme satisfait ses spécifications. Mais la manière la plus convaincante et la plus simple de détecter des raisonnements fallacieux est de trouver un contre-exemple montrant que le raisonnement n'est pas correct. Cette capacité est particulièrement utile dans le cadre de la programmation puisqu'elle permet non seulement de détecter qu'un programme n'est pas correct mais peut également montrer *pourquoi* il est incorrect et fournir des indices sur la façon de *corriger* le programme. Dans cette section, nous montrons comment utiliser la procédure **Extended_Interpreter** pour détecter des erreurs dans les programmes logiques.

Détection d'erreurs dans des programmes formellement spécifiés

Considérons un programme logique $\mathcal{P}$ associé à une *spécification* $\mathcal{F}$. Autrement dit, $\mathcal{P}$ est supposé implémenter $\mathcal{F}$. Cette spécification est pour l'instant exprimée par une formule de la logique du premier ordre (la méthode n'est cependant pas limitée en principe à ce type de spécification). L'objectif est de vérifier que le programme $\mathcal{P}$ satisfait la spécification $\mathcal{F}$, et si cela n'est pas le cas, de donner un *contre-exemple* montrant que $\mathcal{P}$ ne valide pas la spécification. Pour cela, nous utilisons la procédure **Extended_Interpreter** définie à la section 14.4 sur le programme $\mathcal{P}$ et la formule $\neg\mathcal{F}$. Si on obtient une solution σ de $\neg\mathcal{F}$, alors le programme $\mathcal{P}$ ne valide pas la spécification et σ constitue un contre-exemple. Si la procédure s'arrête et retourne $\emptyset$ (c'est-à-dire si on obtient un modèle partiel $\mathcal{M}$ tel que $\phi^{-}_{\mathcal{M}}(\mathcal{F}) = \top$) alors on a : $\mathcal{M} \models \mathcal{F}$ donc $\mathcal{P}$ vérifie la spécification $\mathcal{F}$.

Nous donnons ci-dessous quelques exemples montrant l'utilité de la méthode dans la détection d'erreurs dans des programmes incorrects.

EXEMPLE 14.5.1. Nous notons S le programme suivant. Le littéral $positive(l)$ est supposé signifier : "l est une liste d'entiers positifs". En fait, il n'est pas valide si l contient 0.

$positive([x|l]) \leftarrow positive(l), x > 0$
$positive([])$

La spécification du premier ordre $\mathcal{F}$ de *positive* est la suivante.

$$\forall l.(positive(l) - (\forall x.(member(x,l) \Rightarrow x \geq 0)))$$

où *member* est défini comme suit :
$member(x,[x|l])$.
$member(x,[y|l]) \leftarrow member(x,l), x \neq y.$

Le problème est de tester si le modèle minimal de S satisfait la formule du premier ordre $\mathcal{F}$. RAMC$_{\text{PL1}}$ donne :

1 $[\![positive([x|l]) \vee \neg positive(l) \ : x > 0]\!]$
2 $[\![positive([]) \ : \top]\!]$
3 $[\![member(x,[x|l]) \ : \top]\!]$
4 $[\![member(x,[y|l]) \vee \neg member(x,l) \ : x \neq y]\!]$
5 $[\![positive([x]) \ : x > 0]\!]$ (résolution,2,1)
6 $[\![positive([x|l]) \vee \neg positive(l) \ : x > 0 \wedge l \neq []]\!]$ (disrésolution,2,1)
7 $[\![\neg positive([x]) \ : x \leq 0]\!]$ (GMPL$^-$)
8 $[\![\neg member(x,[]) \ : \top]\!]$ (GMPL$^-$)
9 $[\![member(x,[y|l]) \vee \neg member(x,l) \ : x \neq y \wedge l \neq []]\!]$ (GMPL$^-$)
10 $[\![\neg member(x,[y]) \ : y \neq x]\!]$

Le modèle partiel défini par les c-clauses $5, 7, 3, 10$ suffit à prouver que le programme n'est pas correct par rapport à sa spécification. En effet, la formule $\phi^-_{\mathcal{M}}(\mathcal{F})$ est égale à

$$\begin{array}{l}\exists l. \\ \quad (\exists y.l = [y] \wedge y > 0) \wedge \forall x.l = [x|l'] \wedge x \geq 0 \\ \quad \vee (\exists x'.l = [x'] \wedge x' \leq 0 \wedge (\forall x.\exists z.(l = [z] \wedge z \neq x) \vee x \geq 0))\end{array}$$

Cette formule est évidemment valide ($l = [0]$ satisfait la seconde condition) donc peut être transformée par l'algorithme de résolution de contraintes en $\top$. Cela prouve que $\mathcal{F}$ est fausse dans le modèle du programme donc que S n'est pas correct. La solution obtenue : $\{l = [0], \neg positive[l]\}$ est un exemple montrant que le programme ne correspond pas à la spécification. $\diamond$

Tri par insertion.

Le programme suivant (tiré de (STERLING ET SHAPIRO, 1986), page 55) est censé réaliser un tri par insertion. En fait, les variables Y_s et Z_s dans le dernier littéral de la première règle ont été interchangées, ce qui rend le programme incorrect.
$sort([X|X_s],Y_s) \leftarrow sort(X_s,Z_s), insert(X,Y_s,Z_s).$
$sort([],[]).$
$insert(X,[],[X]).$
$insert(X,[Y|Y_s],[Y|Z_s]) \leftarrow X > Y, insert(X,Y_s,Z_s).$
$insert(X,[Y|Y_s],[X,Y|Y_s) \leftarrow X \leq Y.$

La spécification de *sort* est :

$$sort(X,X') \Rightarrow (sorted(X') \wedge (\forall x.member(x,X) - member(x,X')))$$

avec :

$$sorted(X) \leftarrow (X = [] \vee \exists Y.X = [Y] \vee \exists Y, Y', L.(X = [Y, Y'|L] \wedge sorted([Y'L]) \wedge Y \leq Y')).$$

La procédure **Extended_Interpreter** génère les c-clauses :
1 $[\![sort([X|Xs], Ys) \vee \neg sort(Xs, Zs) \vee \neg insert(X, Ys, Zs) \ : \top]\!]$
2 $[\![sort([], []) \ : \top]\!]$
3 $[\![insert(X, nil, [X]) \ : \top]\!]$
4 $[\![insert(X, [Y|Ys], [Y|Zs]) \vee \neg insert(X, Ys, Zs) \ : X > Y]\!]$
5 $[\![insert(X, [Y|Ys], [X, Y|Ys]) \ : X \leq Y]\!]$
6 $[\![sort([X], Ys]) \vee \neg insert(X, Ys, []) \ : \top]\!]$
(résolution,2,1)
7 $[\![sort([X|Xs], Ys) \vee \neg sort(Xs, Zs) \vee \neg insert(X, Ys, Zs) \ : (Xs \neq []) \vee (Zs \neq [])]\!]$
(disrésolution,2,1)
8 $[\![\neg insert(X, Ys, []) \ : \top]\!]$
(GMPL$^-$)
9 $[\![\neg sort(Xs, Zs) \ : (Xs \neq []) \vee (Zs \neq [])]\!]$
(GMPL$^-$)

Les c-clauses 9 et 2 donnent l'interprétation de *sort* selon la définition du programme. Par exemple la c-clause 9 montre que $sort(X, Z)$ est *faux* si X ou si Z n'est pas une liste vide et montre en particulier qu'il n'est pas possible de trier une liste non vide ! (cela contredit de façon immédiate la spécification de *sort*).

14.6 Perspectives

Dans ce chapitre, nous n'avons pas cherché à effectuer une étude exhaustive de l'application de RAMC à la Programmation en Logique, mais simplement à mettre en évidence l'utilité et la généralité de notre approche de la construction de modèle. Notons que les propriétés des calculs proposés ne dépendent pas de la stratégie utilisée. Il est donc possible d'utiliser n'importe quelle stratégie pour guider l'évaluation d'un but donné (par exemple des stratégies en chaînage "avant" ou "arrière" peuvent être utilisées). La différence principale entre la méthode proposée ici et la négation constructive (STUCKEY, 1995) est qu'elle n'est pas restreinte à une sémantique particulière. Comme nous l'avons vu, elle peut être adaptée facilement à plusieurs types de sémantiques (en choisissant une stratégie adaptée). Pour des programmes standards ou bien fondés, la méthode est plus générale que la négation constructive (voir exemple 14.2.1).

Les résultats présentés dans ce chapitre ouvrent plusieurs voies de recherche et d'application. Il semble important de poursuivre l'étude que nous avons commencée dans ce chapitre en étendant les résultats obtenus à d'autres types de sémantiques (par exemple la sémantique des modèles stables (GELFOND ET LIFSCHITZ, 1988)). Le problème est de déterminer les règles adéquates pour une sémantique donnée et de comparer la méthode obtenue avec les procédures de preuve utilisées en Programmation en Logique. Des extensions à d'autres classes de programmes logiques peuvent également être considérées (programmes disjonctifs, du premier ordre etc.).

Enfin l'application de la méthode à la vérification de programmes mérite d'être approfondie en fournissant par exemple une méthode pour identifier la partie du programme responsable de l'erreur et éventuellement pour corriger *automatiquement* un programme incorrect. D'autre part, les règles pourraient également être utilisées pour la transformation et la synthèse de programme.

L'étude de ces problèmes exige des efforts à plus long terme et sort donc du cadre de la thèse. L'étude de l'application de la méthode à des programmes *stratifiés* peut être trouvée dans (DIERKES, 1996).

Partie V

RAMC$_{\text{ATINF}}$: un système de recherche simultanée de réfutations et de modèles

Chapitre 15

RAMC$_{\text{ATINF}}$: description générale

Ce chapitre décrit brièvement le système RAMC$_{\text{ATINF}}$, sans s'attarder sur les aspects techniques et les détails de l'implémentation. Le but du logiciel est de fournir un prototype permettant d'expérimenter et de valider certaines des idées introduites dans ce mémoire. Il s'agit d'un **démonstrateur/constructeur de modèles interactif**, qui constitue une partie du système ATINF[1] développé au sein du projet ATINF depuis 1985 (CAFERRA ET AL., 1991; CAFERRA ET HERMENT, 1993; CAFERRA ET HERMENT, 1995).

15.1 Fonctionnalités

L'objet du logiciel est de fournir à un utilisateur un large éventail d'outils destinés à l'assister dans la recherche simultanée de preuves et de contre-exemples pour une formule logique. Ses principales fonctionnalités sont les suivantes.

- **Résolveur de contraintes**. La procédure de résolution des contraintes équationnelles est la pierre angulaire de notre approche de la représentation, vérification et construction de modèles de Herbrand. RAMC$_{\text{ATINF}}$ intégre donc un résolveur de formules équationnelles du premier ordre dans l'algèbre des termes (voir chapitre 2). L'accent est mis sur l'efficacité, ainsi que sur la facilité d'utilisation et d'extension, ce qui devrait permettre d'intégrer de nouvelles méthodes de résolution avec un minimum d'efforts.

- **Construction de eq-modèles.** Le logiciel implémente les deux méthodes de construction de modèles RAMC$_{\mathcal{I}}$ et EQMC (voir chapitres 6 et 8). Les règles de décomposition et de coupure sont également utilisées. La méthode RAMCET-2 décrite au chapitre 9 (méthode des tableaux sémantiques étendus par les problèmes équationnels) a été partiellement implémentée : nous avons seulement implémenté la règle d'extraction du modèle R_{mod} définie au chapitre 9. Remarquons que si aucune variable libre n'est introduite dans le tableau, la règle R_{in} peut être vue comme un cas particulier de la règle de **coupure**, dans lequel la formule $\mathcal{Y}$ sur laquelle la règle est appliquée est de la forme $x = t$, où t est un terme fermé. D'autre part, puisque les formules sont des ensembles de c-clauses, la règle $R_{\vee}$ est équivalente à la règle de décomposition et la règle de simplification est simulée par les règles résolution unitaire, disrésolution unitaire et dissubsumption unitaire. Par conséquent, la méthode implémentée peut être vue comme une *combinaison* des méthodes RAMC$_{\mathcal{I}}$ et RAMCET-2$_{\mathcal{I}_s}$, restreintes à des ensembles de c-clauses.

- **Construction de modèles finis.** Le logiciel contient également le constructeur de modèles finis *FMC* (voir chapitre 3). Cette composante du logiciel est celle où le souci d'efficacité a été le plus présent.

1. ATINF signifie **At**elier d'**Inf**érence.

La réalisation du logiciel a été guidée par les objectifs suivants : **modifiabilité**, **facilité d'intégration**, **interactivité** et, autant que possible, **efficacité**.

- **Interactivité.** Il s'agit d'un point important dans la conception du logiciel, en particulier dans le cadre du projet ATINF (CAFERRA ET HERMENT, 1993; CAFERRA ET HERMENT, 1995; CAFERRA ET AL., 1991), dont le but est la conception d'outils d'inférence "orientés-utilisateur".
- **Modifiabilité.** Le logiciel doit être capable d'évoluer avec les améliorations et extensions successives de la méthode (extension des contraintes acceptées, nouvelles stratégies, nouvelles règles, etc.).
- **Facilité d'intégration.** Le logiciel doit pouvoir être connecté facilement à d'autres outils existants (tels que $\text{DISC}_{\text{ATINF}}$ (BOURELY ET PELTIER, 1996) ou $\text{GLEF}_{\text{ATINF}}$). Là encore, cet aspect est de la plus haute importance dans le cadre du projet ATINF.

Bien que l'efficacité ne soit pas notre souci premier, nous avons cherché à définir un système permettant de traiter des problèmes non-triviaux en un temps de calcul raisonnable, afin de faciliter les expérimentations.

Codage

Le logiciel est écrit en langage C et C++ et comprend environ 50000 lignes de code.

15.2 Utilisation

La communication entre l'utilisateur et le logiciel est réalisée grâce à un interpréteur de commandes. Les commandes ont pour but de définir les objets (énoncés), et d'exécuter des méthodes sur ces objets. L'exécution de ces méthodes est contrôlée par un certain nombre de *paramètres*, qui permettent de choisir la stratégie à utiliser, les critères d'arrêt, etc. Les commandes peuvent être saisies en mode texte, où graphiquement, à l'aide de la souris. L'interface graphique a été réalisée à l'aide de l'interface graphique générique d'ATINF, nommée INGRAT[2] (voir (DESBRUN, 1993) pour plus de précisions au sujet de l'interface INGRAT). Nous donnons ci-dessous la liste de quelques unes des commandes disponibles ainsi que leur signification.

15.2.1 Exemples de commandes

load(*file*)	Lit le fichier *file*
quit	Fin du programme
stats	Affichage des statistiques mémoire, temps, règles utilisées...
options	Affiche les valeurs courantes des paramètres
reset	Ré-initialisation
param = *val*	Affecte la valeur *val* au paramètre *param*
param	Affiche la valeur du paramètre *param*
list(*liste*)	Indique le début d'une liste d'objets (liste de clauses ou de formules)
end	Indique la fin de la liste courante
profile	Début d'une déclaration de symboles
formed	Appel de l'éditeur graphique de formules
edit	Appel de l'éditeur de texte

2. **IN**terface **GR**aphique d'**AT**INF.

15.2.2 Saisie des objets

Le logiciel est capable de traiter des formules de la logique des prédicats du premier ordre, ainsi que des formules de la logique relationnelle (ORŁOWSKA, 1991) (par traduction en logique classique).

Décrivons brièvement les conventions syntaxiques utilisées. Les symboles logiques $\wedge, \vee, \neg$ sont respectivement notés `& |, -`. Les variables quantifiées $\forall x_1, \ldots, x_n.P, \exists x_1, \ldots, x_n.P$ s'écrivent `all x1,...,xn.P` et `exists x1,...,xn.P`. $\neq$ est noté `!=`.

Un objet peut être :

- une *clause contrainte*

  ```
  [[ P(x) | Q(x,f(y)) : x != y | all z. x != g(z) ]].
  ```

- une *formule du premier ordre*, i.e. l'une des formes suivantes.

  ```
  P & Q                    P | Q
  (P)                      -P
  all x1,...,xn.P        exists x1,...,xn.P
  P(t1,...,tn)             P -> Q
  P <-> Q
  ```

  où `t1,...tn` dénotent des termes, `x1,...xn` dénotent des variables, et `P, Q` dénotent des formules.

- Une *formule relationnelle*, qui est inductivement construite soit comme une formule du premier ordre, soit de la forme `x E y` où `x` et `y` sont deux termes et `E` une *expression relationnelle* inductivement définie de la façon suivante : `P + Q, P - Q, P;Q, P / Q, (P), R` où `P, Q` sont des expressions relationnelles et `R` est une constante relationnelle i.e. un symbole de prédicat d'arité 2. Nous ne rappellerons pas ici la sémantique de la logique relationnelle (voir par exemple (ORŁOWSKA, 1991) pour une introduction à la logique relationnelle). Notons que la logique relationnelle a le même pouvoir d'expression que la logique des prédicats du premier ordre.

15.2.3 Exemples de méthodes

solve(*obj*$_1$[, *obj*$_2$]))	Exécute la méthode SOLVE sur les objets *obj*$_1$ et *obj*$_2$
ramc(*obj*)	Appelle la méthode RAMC sur l'objet *obj*
eqmc(*obj*)	Appelle la méthode EQMC sur l'objet *obj*
fmb(*obj*)	Exécute la méthode FMB sur l'objet *obj*

Nous donnons ci-dessous une brève description de chaque méthode. Les algorithmes utilisés ont déjà été spécifiés aux chapitres 3, 6, 8, 9, aussi décrirons-nous uniquement la stratégie utilisée et les principaux choix d'implémentation. La liste des principaux paramètres disponibles et leur signification est également donnée.

Traduction et manipulation des formules

La traduction entre formules du premier ordre et ensembles de c-clauses est réalisée par l'algorithme de mise sous forme clausale du transformateur de formules TMS (**T**ransformateur **M**ulti-**S**ortes) réalisé par T. BOY DE LA TOUR. Il s'agit d'un transformateur de formules par renommage implémentant des résultats de minimalité de la forme clausale (BOY DE LA TOUR, 1992). Un algorithme de traduction très simple de la logique relationnelle en logique du premier ordre a également été implémenté (ORŁOWSKA, 1991).

Résolution d'une formule équationnelle

`solve(obj)`

Calcule une forme résolue avec contraintes équivalente à la formule équationnelle `obj`.

`solve(obj`$_1$`,obj`$_2$`)`

Evalue la formule `obj`$_1$ dans l'eq-interprétation `obj`$_2$. Calcule et affiche la forme résolue avec contraintes correspondant à la formule $\phi^{+}_{\texttt{obj}_1}(\texttt{obj}_2)$.

La méthode utilisée pour la résolution des contraintes est celle de (Comon et Lescanne, 1989), mais les formules ne sont pas systématiquement mises sous forme normale conjonctive après chaque application d'une règle: comme dans l'algorithme décrit dans (Comon et Delor, 1994), la transformation en forme normale est réalisée, si cela s'avère nécessaire, pendant le processus de résolution, par l'application de règles de distributivité. Les règles d'unification (décomposition, test d'occurrence, incompatibilité) et d'universalité des paramètre sont appliquée avec une priorité maximale. Puis les règles de fusion et de remplacement sont appliquées et enfin les règles de distributivité, d'explosion et d'explosion des disjonctions.

RAMC

`ramc(`*obj*`)`

Applique la méthode Ramc sur l'objet *obj*.

On distingue trois niveaux de règles, associés à trois niveaux de priorité décroissante ($R_1 > R_2 > R_3$).

(R_1) **C-dissubsomption, c-distautologie, c-factorisation, c-disfactorisation et résolution de contraintes.** Comme nous l'avons vu précédemment (voir chapitre 6) le système $Nramc$ défini par les règles du groupe R_1 est à terminaison finie.

(R_2) **C-résolution, c-disrésolution et GMPL.** Ces règles ne sont pas à terminaison finie.

(R_3) **Décomposition, coupure, génération du modèle (R_{mod}).** Ces règles ne sont pas nécessaires pour la complétude réfutationnelle, mais augmentent la classe de modèles constructibles (voir chapitres 8 et 9). Notons que la règle de décomposition est particulière, car elle introduit un facteur de branchement dans l'algorithme de recherche. Elle remplace donc un problème donné par deux problèmes distincts, qui doivent être résolus indépendamment. La règle de coupure n'est pour l'instant applicable qu'en mode interactif.

Notons que si aucune stratégie n'est utilisée pour restreindre l'application des règles de R_2, les règles de R_3 peuvent ne jamais être appliquées.

La règle GMPL

Il est clair que l'application non-déterministe de la règle Gmpl est très coûteuse en pratique. Par conséquent, le système limite le nombre de c-littéraux candidats considérés simultanément. Si aucun ensemble non vide ne peut être généré, la taille de l'ensemble est incrémentée. D'autre part, la règle d'**extension** n'est appliquée que si cela est nécessaire, c'est-à-dire si l'application de la règle **Réduction** conduit à un ensemble vide. Cela permet de limiter l'augmentation de la taille de l'ensemble de c-littéraux candidats durant le processus de calcul.

Divers degrés d'interactivité

L'utilisateur dispose de plusieurs possibilités pour contrôler le processus de recherche. En premier lieu, il peut choisir la stratégie utilisée pour guider l'exploration de l'espace de recherche en fixant la valeur de certains paramètres, qui permettent de guider le choix des règles et des c-clauses à considérer en priorité (exploration en largeur d'abord, en profondeur d'abord, ou choix des c-clauses de moindre poids), de fixer des limites à la recherche (temps de calcul, nombre ou taille des c-clauses générées...), etc. Il s'agit d'un mode *semi-automatique*, où l'utilisateur fixe uniquement la stratégie utilisée.

En second lieu, l'utilisateur a également la possibilité de guider pas à pas l'application de certaines règles. En mode *interactif*, l'ordinateur calcule et affiche les règles possibles, et laisse à l'utilisateur le choix de contrôler le processus de recherche.

Le système permet d'établir plusieurs *niveaux d'interactivité*, ce qui offre à l'utilisateur la possibilité de contrôler certaines règles-clefs de façon précise, tout en laissant à l'ordinateur le soin de contrôler totalement d'autres parties du processus. L'utilisateur peut à tout moment changer la stratégie utilisée, suggérer des lemmes, ou proposer un modèle partiel.

Quelques paramètres

Paramètre	Type	Signification
rules	liste de termes	liste des règles actives de RAMC
for_sub	booléen	dissubsomption "forward"
back_sub	booléen	dissubsomption "backward"
unit_sub	booléen	dissubsomption unitaire ("forward" uniquement)
dis_taut	booléen	distautologie
gmpl	booléen	Règle GMPL
binary_res	booléen	résolution binaire
dis_binary_res	booléen	disrésolution binaire
factor	booléen	factorisation
dis_factor	booléen	disfactorisation

EQMC

`eqmc(obj)`

Applique la procédure EQMC sur l'objet `obj`. La procédure *Deduce* utilisée peut être soit la méthode RAMC, soit la méthode Normalize$_{\mathcal{I}}(\mathcal{F})$ (voir chapitre 8). A la différence de RAMC, EQMC est directement applicable sur des formules quelconques (non nécessairement en forme c-clausale), donc ne nécessite pas l'utilisation de la transformation en forme clausale. Là encore, plusieurs niveaux d'interactivité sont disponibles. En mode automatique, le système applique les règles dans l'ordre suivant : nettoyage, déduction, décomposition. Il utilise certaines des heuristiques proposées au chapitre 8 (notamment pour le choix des littéraux sur lesquels appliquer la règle de décomposition).

FMC

L'algorithme utilisé est précisément décrit au chapitre 3. Nous nous contenterons ici de donner la liste de quelques-uns des paramètres disponibles.

`order`	entier	Ordre sur les cellules
`refutation_precedence`	entier	Choix des réfutations
`detect_symmetry`	booléen	Utilise la règle de **détection des symétries**
`ref`	booléen	Utilise la fonction ϕ_{ref}
`cref`	booléen	Utilise la fonction ϕ'_{cref}

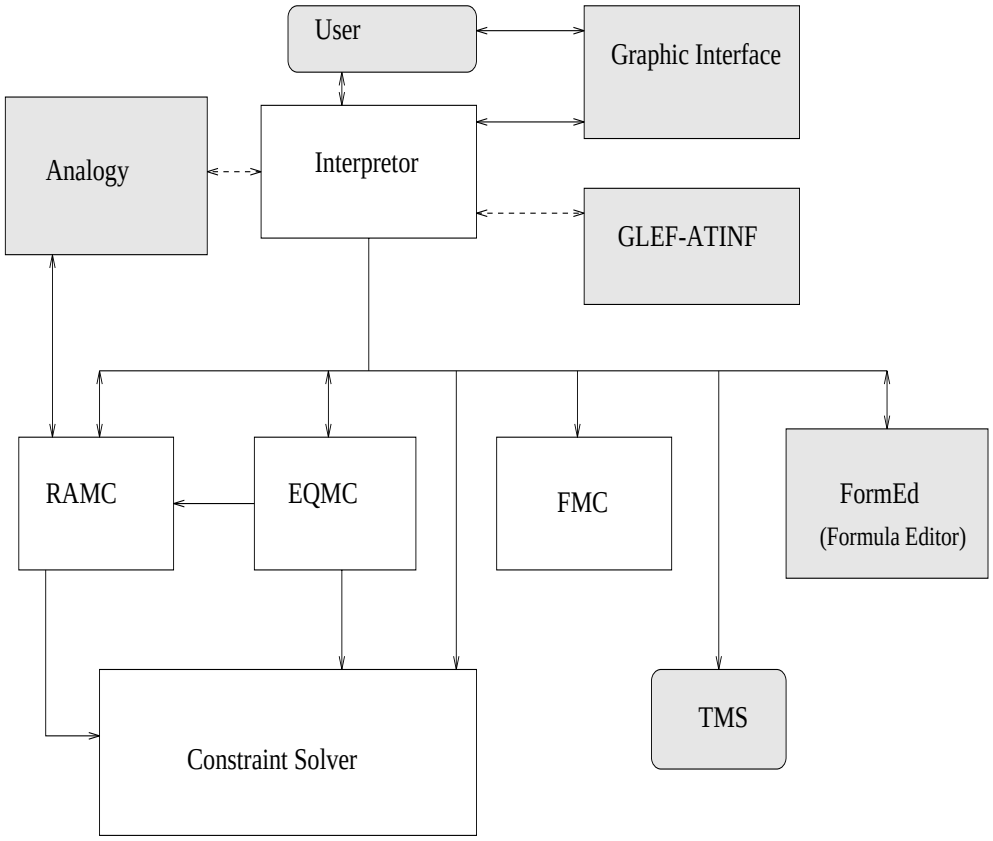


FIG. 15.1 - : *$RAMC_{ATINF}$: schéma d'organisation général.*

La taille du domaine de chaque sorte doit être spécifiée par l'utilisateur (commande `set_size`).

La figure 15.1 résume l'organisation des différentes parties du logiciel $\text{RAMC}_{\text{ATINF}}$. Les parties grisées sont extérieures au logiciel (elles indiquent l'intégration de RAMC dans ATINF). Les lignes en pointillés indiquent des connections prévues mais non encore réalisées. La figure 15.2 indique la place de $\text{RAMC}_{\text{ATINF}}$ au sein du système ATINF.

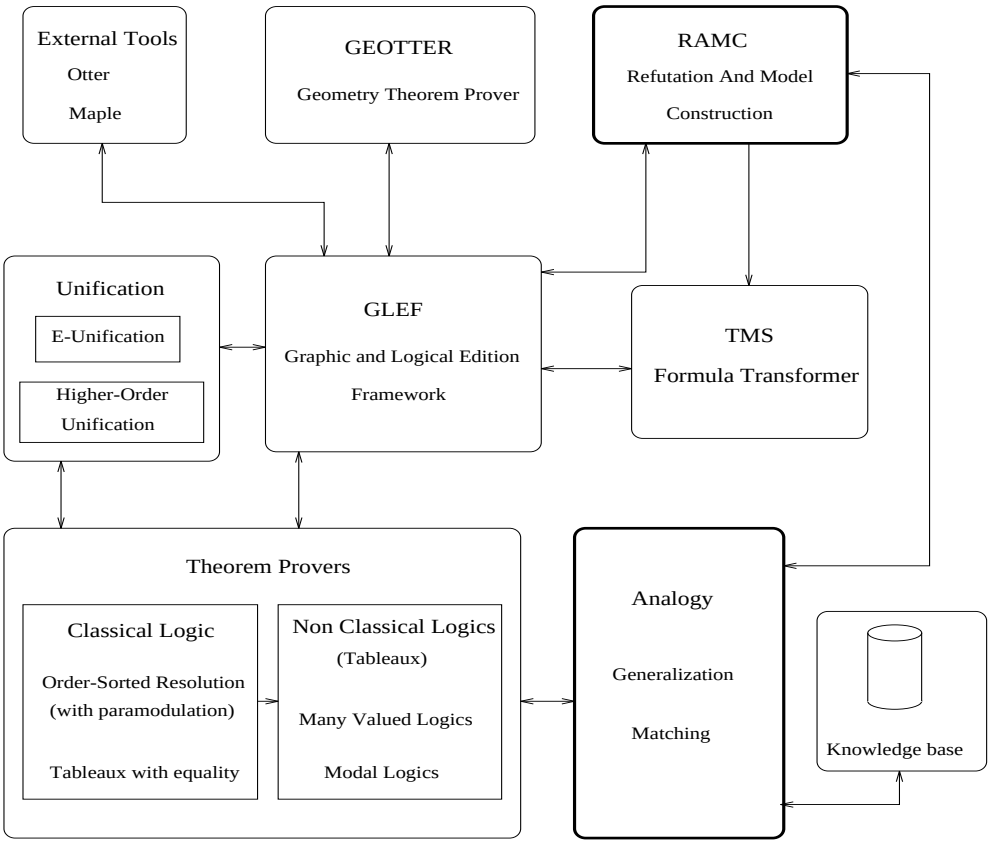


FIG. 15.2 - : *ATINF: état actuel.*

Chapitre 16

Exemples et expérimentations

"L'homme n'a de connaissance des choses naturelles que par les moyens de la correspondance avec ce qui tombe sous les sens"

René Descartes

Nous donnons ci-dessous des exemples de session de travail et des résultats obtenus avec le logiciel Ramc.

16.1 Exemples d'utilisation du logiciel

16.1.1 Résolveur de contraintes

Exemple

```
profile.   % declaration de profils (optionnel)
a: -> top.
f: top -> top.
g: top, top -> top.
end.

signature = { a,f,g }. % signature

list(prob).   % probleme a resoudre
(f(A) =f(B) | (all Z. ((Z = a) | (A != g(Z,Z))))).
end.

solve(prob).

% Forme resolue

prob = (exists T,U,S ((-(T = U) & (A = g(U,T)))
| (A = g(U,a)) | (A = a) | (A = f(S)) | (A = B)))

Run Time: 0.01
```

Quelques "benchmarks"

Nous donnons quelques exemples de formules équationnelles simples, qui nous ont été communiquées par Hubert Comon, (voir figure 16.1) ainsi que les formes résolues et le temps de calcul obtenu avec $\text{Ramc}_{\text{Atinf}}$(figure 16.2).

Problème	Définition
1	$0 = 0 \vee (x = y \wedge x \neq z)$
2	$0 \neq 0 \vee (x = y \wedge x \neq z)$
3	$\forall y.s(x) \neq s(y) \vee x = y$
4	$f(x_1, x_2) = f(x_2, x_1)$ $\vee g(x_1, x_2) = g(x_2, x_1) \vee x_1 \neq s(x_3)$
5	$f(y, y) = f(x(z), z) \vee f(x, y) = f(y, s(x)) \vee z \neq s(y)$
6	$x = y \wedge (z \neq s(x) \vee x = s(z))$
7	$\forall y.x \neq f(f(y, s(y)), s(s(y)))$
8	$x = y \wedge x = z$
9	$x = s(s(z)) \wedge x = s(y) \wedge y = s(s(x)) \wedge z = s(s(y))$
10	$\forall y.(\exists x, z.(x = s(z) \vee x \neq z) \wedge x \neq s(s(z)))$
11	$\forall y.(y = s(x) \wedge z = s(x)) \vee (\exists z.z \neq s(x) \wedge z \neq x)$
12	$\forall y.((x \neq s(s(y)) \vee x \neq s(0)) \wedge z = s(x))$ $\vee (\exists z.z \neq s(x) \wedge z \neq x)$
13	$\forall y_1, y_2, y_3, y_4.$ $(y_1 = x_1 \vee f(x_1, x_4, x_4) = f(x_1, x_2, s(x_3)))$ $\wedge (f(y_1, y_2, s(y_2)) \neq f(s(y_2), x_1, x_2))$ $\wedge x_1 \neq f(y_1, y_2, y_3)$
14	$\forall y.s(x) = s(z) \wedge (x = z \vee x \neq s(y))$
15	$\forall y_1, y_2, y_3, y_4.(y_2 \neq f(y_1, y_1) \vee y_3 \neq f(y_2, y_2)$ $\vee y_4 \neq f(y_3, y_3) \vee x \neq f(y_1, y_1)$
16	$(\exists x.\forall y.\exists z.(x \neq 0 \wedge x \neq f(y, z)))$
17	$\forall y.x \neq y \vee x \neq s(y)$

FIG. 16.1 - : *Exemples de formules équationnelles*

16.1.2 Construction de modèles

Dans la mesure où la construction de modèles n'a pas fait l'objet de beaucoup de travaux (voir chapitre 1.1), on ne trouve que peu d'exemples de formules satisfaisables dans la littérature. Les exemples présentés dans cette section sont tirés d'articles présentant d'autres approches en construction de modèles de Herbrand.

```
% Inference and dis-inference rules

rules = { binary_res, dis_binary_res, factor, dis_factor,
dis_taut,for_sub, back_sub, gmpl }.

% Automatic mode

clear(interact).

clear(prolog_style_variables).

list(sos).
P(x) | Q(g(x,x)).
-Q(y) | R(x,y).
-R(a,a) | -R(f(b),a).
R(f(x),y).
P(x) | -P(f(x)).
-P(a).
end_of_list.

% RAMC procedure
```

Problème	Forme résolue	Temps de calcul (en ms)
1	$\top$	0.65
2	$y \neq z \wedge x = y$	0.85
3	$\top$	0.68
4	$x_1 \neq s(x_3) \vee x_2 \neq s(x_3)$	1.17
5	$z \neq s(y)$	0.93
6	$x = y \wedge x = z$	0.91
7	$(\exists u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}$ $(x = f(u_1, s(f(u_2, u_3))))$ $\vee(x = f(u_1, s(0)))$ $\vee(x = f(u_1, 0))$ $\vee(x = f(u_1, f(u_4, u_5)))$ $\vee(u_6 \neq s(s(u_7)) \wedge (x = f(f(u_7, u_8), u_6)))$ $\vee(u_6 \neq s(s(u_7)) \wedge (x = f(0, u_6)))$ $\vee(u_6 \neq s(s(u_7)) \wedge (x = f(s(u_9), u_6)))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(u_1, s(f(u_2, u_3)))))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(u_1, s(0))))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(u_1, 0)))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(u_1, f(u_4, u_5))))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(f(u_7, u_8), u_6)))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(0, u_6)))$ $\vee(u_8 \neq s(u_7) \wedge (x = f(s(u_9), u_6)))$ $\vee(x = f(0, u_6))$ $\vee(x = f(s(u_9), u_6))$ $\vee(x = s(u_{10}))$ $\vee(x = 0))$	16.69
8	$z = y \wedge x = z$	0.65
9	$\bot$	0.95
10	$\top$	1.36
11	$\exists z.(z \neq x \wedge z \neq s(x))$	1.52
12	$\exists u.((u \neq x) \wedge (u \neq s(x)) \vee (z = s(x)))$	1.75
13	$\exists u.((x = 0) \wedge (x_4 = x_2)$ $\wedge(x_2 = s(x_3))) \wedge (x_1 = s(u) \wedge u \neq x_3)$ $\wedge(x_4 = x_2) \wedge (x_2 = s(x_3))$	8.32
14	$x = z$	0.97
15	$\exists u_1, u_2.(u_1 \neq u_2 \wedge x = f(u_1, u_2))$ $\vee x = 0$	2.18
16	$\exists x.x \neq 0$	1.14
17	$\top$	0.71

FIG. 16.2 - : *Exemples de formules équationnelles (formes résolues)*

```
ramc(sos).

process.

------------> process sos:
** KEPT: 1 [] [[ -P(a) : TRUE ]].
** KEPT: 2 [] [[ P(x) | -P(f(x)) : TRUE ]].
** KEPT: 3 [] [[ R(f(x),y) : TRUE ]].
** KEPT: 4 [] [[ -R(a,a) | -R(f(b),a) : TRUE ]].
** KEPT: 5 [] [[ -Q(x) | R(y,x) : TRUE ]].
** KEPT: 6 [] [[ P(x) | Q(g(x,x)) : TRUE ]].

** Clause #5 is deleted
** KEPT: 7 [dissubsumption,3,5] [[ -Q(x) | R(y,x) :
exists v,z. (y = g(v,z)) | (y = b) | (y = a) ]].

------------> done processing input.
--- Set Of Support ---
1 [] [[ -P(a) : TRUE ]].
2 [] [[ P(x) | -P(f(x)) : TRUE ]].
3 [] [[ R(f(x),y) : TRUE ]].
4 [] [[ -R(a,a) | -R(f(b),a) : TRUE ]].
6 [] [[ P(x) | Q(g(x,x)) : TRUE ]].
7 [dissubsumption,3,5] [[ -Q(x) | R(y,x) :
exists v,z. (y = g(v,z)) | (y = b) | (y = a) ]].
end_of_list.

given clause: #1: (wt=2)
** KEPT: 8 [gpl] [[ R(y,x) : exists v,z . (y = g(v,z))
| (y = b) | ((x != a) & ((y = g(v,z)) | (y = b) | (y = a))) ]].
** Clause #7 is deleted
** KEPT: 9 [dissubsumption,8,7] [[ -Q(a) | R(a,a) : TRUE ]].

given clause: #1: (wt=2)
** KEPT: 10 [gpl] [[ -Q(a) : TRUE ]].
** Clause #9 is deleted

given clause: #10: (wt=2)
** KEPT: 11 [gpl] [[ Q(g(x,x)) : TRUE ]].
** Clause #6 is deleted

given clause: #1: (wt=2)
** KEPT: 12 [gpl] [[ -R(a,a) : TRUE ]].
** Clause #4 is deleted

given clause: #10: (wt=2)
** KEPT: 13 [gpl] [[ -P(f(x)) : TRUE ]].
** Clause #2 is deleted

Model:

R(y,x) is true if: (exists u,v,w. (((y = a) & -(x = a))
| ((y = b) & -(x = a)) | ((y = g(u,v)) & -(x = a))
| (y = b) | (y = g(u,v)) | (y = f(w))))))

Q(x) is true if: (exists w. (x = g(w,w)))

R(y,x) is false if: ((y = a) & (x = a))
```

```
P(x) is false if: (exists w. ((x = f(w)) | (x = a)))

Q(x) is false if: (x = a)
Run Time:     0.22
```

16.2 Quelques résultats expérimentaux

Nous donnons ci-dessous les résultats obtenus avec RAMC$_{\text{ATINF}}$ sur quelques exemples de formules satisfaisables publiées dans la littérature sur la construction de modèles de Herbrand. Afin de faciliter les comparaisons, la procédure de simplification *Deduce* (voir chapitre 8) utilisée par EQMC est la procédure Normalize$_{\mathcal{I}}(\mathcal{F})$ (voir chapitre 15).

Problem	Description	RAMC	EQMC	OTTER
	Ex 4.1 (CAFERRA ET ZABEL, 1992)	0.17	0.1	-
SYN303-1	(BOURELY ET AL., 1994)	0.32	0.08	-
SYN304-1	(BOURELY ET AL., 1994)	0.06	0.1	0.04
SYN306-1	(FERMÜLLER ET AL., 1993)	0.17	16.87	-
SYN307-1	(FERMÜLLER ET AL., 1993)	0.22	0.08	-
SYN308-1	(FERMÜLLER ET AL., 1993)	0.05	0.1	0.05
SYN309-1	(FERMÜLLER ET AL., 1993)	0.11	0.08	0.04
SYN317-1	(CHURCH, 1956)	0.01	0.02	0.06
SYN320-1	(CHURCH, 1956)	0.02	0.04	0.09
SYN322-1	(CHURCH, 1956)	0.02	0.02	0.12
SYN329-1	(CHURCH, 1956)	0.01	0.28	0.01
SYN337-1	(CHURCH, 1956)	0.01	0.62	0.01
SYN342-1	(CHURCH, 1956)	0.01	0.03	0.01
SYN344-1	(CHURCH, 1956)	0.14	0.38	0.16
PUZ001-1	Dreadbury Mansion	1.1	10.78	0.9
K1	(KLINGENBECK, 1996), ex. 36	7.04	1.41	-
K2	(KLINGENBECK, 1996), ex. 37	0.17	0.66	-
K1+K2	(KLINGENBECK, 1996)	17.39	6.48	-
K3	(KLINGENBECK, 1996), ex. 27	0.1	0.18	-
P1	Exemple 7.3.4	-	0.08	-
P2	Exemple 7.3.5	-	0.06	-
P3	Exemple 8.5.1	-	0.93	-

Les résultats obtenus avec le démonstateur OTTER (MCCUNE, 1995), qui est l'un des démonstrateurs les plus puissants actuellement disponibles dans le domaine public, sont donnés à titre de comparaison (la résolution binaire et la factorisation sont utilisées). Pour certains de ces problèmes, OTTER est capable de détecter la satisfaisabilité de l'ensemble de clauses. Cependant, il ne construit pas explicitement un modèle de la formule (OTTER est un démonstrateur, non un système de construction de modèles).

Les problèmes K_1 et K_2 appartiennent à la classe RRSD introduit dans (KLINGENBECK, 1996), pages 119-120. Il est prouvé que l'hyperrésolution ne se termine pas sur le problème $K2$ et que la résolution A-ordonnée ne se termine pas sur K_1. Toute formule satisfaisable de RRSD admet un eq-modèle, donc la méthode RAMCET est une procédure de décision pour la classe RRSD. Les problèmes SYN317-344 sont tirés de (CHURCH, 1956). Notons que ces problèmes sont de difficultés très inégales : certains (par exemple les problèmes SYN317, SYN320...) sont triviaux, d'autres sont plus complexes, et n'appartiennent souvent pas à la classe $\mathfrak{C}_{\text{eq-model}}$, donc sont hors de portée de notre implémentation (c'est le cas, par exemple, des problèmes

SYN330, SYN335, etc.). Les problèmes P_1, P_2 et P_3 sont des problèmes que nous avons introduits pour clairement illustrer les limites de RAMC (exemples 7.3.4, 7.3.5, et 8.5.1 respectivement). Notons que les problèmes P_1, P_2 ne peuvent pas être résolus par la méthode RAMC puisque ils ne possèdent aucun eq-modèle générable. L'utilisation de la règle R_{mod} est nécessaire pour obtenir le modèle. Le problème P_3 est une formule du premier ordre. Sa transformation en forme clausale n'a pas de eq-modèle, donc la méthode RAMC ne peut construire un modèle pour P_3. Les expérimentations tendent à montrer que la méthode RAMC s'avère plus efficace en pratique que la méthode EQMC (cette dernière étant néanmoins plus générale, du point de vue de la construction de modèle). Néanmoins, cela n'est pas toujours vrai, car l'efficacité de EQMC dépend essentiellement de la profondeur minimale du eq-modèle (voir chapitre 8) ce qui n'est pas le cas de RAMC (voir par exemple les résultats obtenus sur les problèmes K_1 et $K_1 + K_2$).

16.3 Solution interactive d'un problème difficile

Dans cette section, nous étudions la formule suivante (notée $\mathcal{F}$ dans le reste de ce chapitre), tirée de (GOLDFARB, 1984). Il s'agit d'une formule satisfaisable, ne possédant pas de modèles finis. $\mathcal{F}$ appartient à la classe de GÖDEL avec égalité. Elle a été imaginée par GOLDFARB pour prouver l'indécidabilité de cette classe et réfuter ainsi une conjecture émise par GÖDEL, qui pensait pouvoir étendre sa démonstration de la décidabilité de la classe de GÖDEL au cas avec égalité.

$$\mathcal{F} : \forall x.\forall y.\exists z_0.\mathcal{H}$$

où $\mathcal{H}$ est la conjonction des formules suivantes.

(1) $Z(x) \wedge Z(y) \rightarrow x = y$
(2) $Z(z_0) \wedge \neg S(z_0, x) \wedge \bigwedge_{\delta=1,2} P_\delta(x, z_0) \wedge P_\delta(x, y) \rightarrow y = z_0$
(3) $\exists z.S(z, x)$
(4) $\neg Z(x) \wedge x \neq y \rightarrow \exists z.(S(x, z) \wedge \neg S(y, z))$
(5) $\exists z.[N(x, z) \wedge (Q(x, y) \rightarrow Q(z, y)) \wedge (R_1(x, y) \rightarrow R_1(z, y)) \wedge (R_2(x, y) \rightarrow R_2(z, y))]$
(6) $N(x, y) \rightarrow \exists z.(P_2(x, z) \wedge P_2(y, z))$
(7) $N(x, y) \rightarrow \exists w, u(P_1(x, w) \wedge S(u, w) \wedge P_1(y, u))$
(8) $S(x, y) \rightarrow \exists z.(Q(z, x) \wedge P_2(z, y) \wedge P_1(z, z_0))$
(9) $Q(x, y) \rightarrow \exists z.(P_1(x, z) \wedge (S(y, z) \rightarrow P_2(x, z))$
(10) $\bigwedge_{\delta=1,2}[P_\delta(x, y) \wedge \neg Z(y) \rightarrow \exists z, w.(R_\delta(z, x) \wedge P_2(z, x) \wedge P_1(z, z_0) \wedge S(y, w))]$
(11) $\bigwedge_{\delta=1,2}[R_\delta(x, y) \rightarrow \exists z, w.(P_1(x, z) \wedge S(w, z) \wedge (P_\delta(y, w) \rightarrow P_2(x, z)))]$

Nous allons construire un modèle de la formule $\mathcal{F}$. Le processus de construction de modèle sera décomposé en deux parties.

1. Une partie *interactive* consistant en une simplification de la formule initiale. Elle permet de réduire de façon importante l'espace de recherche et fixe le *domaine* de l'interprétation.

2. Une partie purement automatique utilisant la méthode EQMC présentée au chapitre 8.

Simplification de la formule

En premier lieu, le quantificateur $\exists z_0$ est éliminé en utilisant la skolémisation. Toutes les occurrences de z_0 sont remplacées par le terme $f(x, y)$ où f est un nouveau symbole de fonction. On obtient ainsi la formule $\forall x, y.\mathcal{H}'$, où $\mathcal{H}'$ est la conjonction des formules suivantes.

(1) $Z(x) \wedge Z(y) \rightarrow x = y$
(2) $Z(f(x,y)) \wedge \neg S(f(x,y),x) \wedge \bigwedge_{\delta=1,2} P_\delta(x, f(x,y)) \wedge P_\delta(x,y) \rightarrow y = f(x,y)$
(3) $\exists z.S(z,x)$
(4) $\neg Z(x) \wedge x \neq y \rightarrow \exists z.(S(x,z) \wedge \neg S(y,z))$
(5) $\exists z.[N(x,z) \wedge (Q(x,y) \rightarrow Q(z,y)) \wedge (R_1(x,y) \rightarrow R_1(z,y)) \wedge (R_2(x,y) \rightarrow R_2(z,y))]$
(6) $N(x,y) \rightarrow \exists z.(P_2(x,z) \wedge P_2(y,z))$
(7) $N(x,y) \rightarrow \exists w,u(P_1(x,w) \wedge S(u,w) \wedge P_1(y,u))$
(8) $S(x,y) \rightarrow \exists z.(Q(z,x) \wedge P_2(z,y) \wedge P_1(z,f(x,y)))$
(9) $Q(x,y) \rightarrow \exists z.(P_1(x,z) \wedge (S(y,z) \rightarrow P_2(x,z))$
(10) $\bigwedge_{\delta=1,2}[P_\delta(x,y) \wedge \neg Z(y) \rightarrow \exists z,w.(R_\delta(z,x) \wedge P_2(z,x) \wedge P_1(z,f(x,y)) \wedge S(y,w))]$
(11) $\bigwedge_{\delta=1,2}[R_\delta(x,y) \rightarrow \exists z,w.(P_1(x,z) \wedge S(w,z) \wedge (P_\delta(y,w) \rightarrow P_2(x,z)))]$

En particulier on a :

(1) $\neg Z(x) \vee \neg Z(y) \vee x = y$
(2) $Z(f(x,y))$

On en déduit (par résolution) la clause :

$$f(x,y) = f(x',y').$$

f est donc constante. Il est donc possible de remplacer les termes $f(x,y)$ par un nouveau terme d'arité 0 noté 0 [1] On obtient ainsi l'ensemble de formules suivant.

(1) $Z(x) \wedge Z(y) \rightarrow x = y$
(2) $Z(0) \wedge \neg S(0,x) \wedge \bigwedge_{\delta=1,2} P_\delta(x,0) \wedge P_\delta(x,y) \rightarrow y = 0$
(3) $\exists z.S(z,x)$
(4) $\neg Z(x) \wedge x \neq y \rightarrow \exists z.(S(x,z) \wedge \neg S(y,z))$
(5) $\exists z.[N(x,z) \wedge (Q(x,y) \rightarrow Q(z,y)) \wedge (R_1(x,y) \rightarrow R_1(z,y)) \wedge (R_2(x,y) \rightarrow R_2(z,y))]$
(6) $N(x,y) \rightarrow \exists z.(P_2(x,z) \wedge P_2(y,z))$
(7) $N(x,y) \rightarrow \exists w,u(P_1(x,w) \wedge S(u,w) \wedge P_1(y,u))$
(8) $S(x,y) \rightarrow \exists z.(Q(z,x) \wedge P_2(z,y) \wedge P_1(z,0))$
(9) $Q(x,y) \rightarrow \exists z.(P_1(x,z) \wedge (S(y,z) \rightarrow P_2(x,z))$
(10) $\bigwedge_{\delta=1,2}[P_\delta(x,y) \wedge \neg Z(y) \rightarrow \exists z,w.(R_\delta(z,x) \wedge P_2(z,x) \wedge P_1(z,0) \wedge S(y,w))]$
(11) $\bigwedge_{\delta=1,2}[R_\delta(x,y) \rightarrow \exists z,w.(P_1(x,z) \wedge S(w,z) \wedge (P_\delta(y,w) \rightarrow P_2(x,z)))]$

Nous utilisons également la skolémisation pour éliminer le quantificateur $\exists z.$ dans la clause (3). On introduit un symbole de fonction s d'arité 1 et on remplace $\exists z.S(z,x)$ par $S(s(x),x)$. Ensuite les quantificateurs sont transférés aux profondeurs maximales dans la formule.

On obtient finalement :

1. Cela correspond à la règle d'introduction de nouvelles fonctions utilisée par le démonstrateur OTTER (MCCUNE, 1990).

(1) $\forall x, y.(Z(x) \wedge Z(y) \rightarrow x = y))$
(2) $Z(0) \wedge \neg\forall x.S(0, x) \wedge \bigwedge_{\delta=1,2} \forall x, y.(P_\delta(x, 0) \wedge P_\delta(x, y) \rightarrow y = 0)$
(3) $\forall x, y.S(f(x), x))$
(4) $\forall x, y.(\neg Z(x) \wedge x \neq y \rightarrow \exists z.(S(x, z) \wedge \neg S(y, z)))$
(5) $\forall x, y.(\exists z.[N(x, z) \wedge (Q(x, y) \rightarrow Q(z, y)) \wedge (R_1(x, y) \rightarrow R_1(z, y)) \wedge (R_2(x, y) \rightarrow R_2(z, y))])$
(6) $\forall x, y.(N(x, y) \rightarrow \exists z.(P_2(x, z) \wedge P_2(y, z)))$
(7) $\forall x, y.(N(x, y) \rightarrow \exists w, u(P_1(x, w) \wedge S(u, w) \wedge P_1(y, u)))$
(8) $\forall x, y.(S(x, y) \rightarrow \exists z.(Q(z, x) \wedge P_2(z, y) \wedge P_1(z, 0)))$
(9) $\forall x, y.(Q(x, y) \rightarrow \exists z.(P_1(x, z) \wedge (S(y, z) \rightarrow P_2(x, z)))$
(10) $\bigwedge_{\delta=1,2}[\forall x, y.(P_\delta(x, y) \wedge \neg Z(y) \rightarrow \exists z, w.(R_\delta(z, x) \wedge P_2(z, x) \wedge P_1(z, 0) \wedge S(y, w)))]$
(11) $\bigwedge_{\delta=1,2}[\forall x, y.(R_\delta(x, y) \rightarrow \exists z, w.(P_1(x, z) \wedge S(w, z) \wedge (P_\delta(y, w) \rightarrow P_2(x, z))))]$

Cette formule est notée $\mathcal{F}_{simp}$ dans la suite.

Introduction de sortes

Nous utilisons ensuite un algorithme d'inférence de types afin de calculer un ensemble de sortes $\mathcal{S}$ et une fonction *profil* telle que la formule $\mathcal{F}_{simp}$ est bien typée. Cette opération préserve la satisfaisabilité et a pour but de réduire l'espace de recherche.

L'algorithme d'inférence de types produit les profils suivants.

$$\mathcal{S} = \{S_1, S_2\}$$

$$\begin{array}{lll}
s: & S_1 & \rightarrow S_1 \\
0: & & \rightarrow S_1 \\
S: & S_1, S_1 & \rightarrow Boolean \\
Z: & S_1 & \rightarrow Boolean \\
R_1: & S_2, S_2 & \rightarrow Boolean \\
R_2: & S_2, S_2 & \rightarrow Boolean \\
Q: & S_2, S_1 & \rightarrow Boolean \\
N: & S_2, S_2 & \rightarrow Boolean \\
P_1: & S_2, S_1 & \rightarrow Boolean \\
P_2: & S_2, S_1 & \rightarrow Boolean
\end{array}$$

Utilisation de la méthode EQMC

La méthode EQMC peut alors être utilisée sur la formule $\mathcal{F}'_{simp}$. Il faut pour cela spécifier le *domaine* de l'interprétation, c'est-à-dire l'ensemble des symboles fonctionnels de la signature. Celui-ci contient évidemment les symboles de $\mathcal{F}'_{simp}$:

$$\{0, s\}.$$

D'autre part, nous ajoutons également une fonction g de profil

$$g : S_1, S_1 \rightarrow S_2.$$

S_1 est donc isomorphe à $\mathbb{N}$ et S_2 à $\mathbb{N}^2$. Ce choix est guidé par la connaissance à priori de ce que devrait être le modèle de $\mathcal{F}'_{simp}$. C'est la *seule partie* qui nécessite une intervention réelle de l'utilisateur (les autres pouvant être aisément automatisées). La méthode EQMC est ensuite utilisée pour calculer automatiquement l'interprétation des symboles de prédicats de la formule. On obtient le modèle suivant.

$Z(0)$ is true
$S(s(A), A)$ is true
$P1(g(A,B), A)$ is true
$P2(g(A,B), B)$ is true
$R1(g(A,B), g(s(B), C))$ is true
$R2(g(A,B), g(C, s(B)))$ is true
$N(g(A,B), g(s(A), B))$ is true
$Q(g(A,B), s(B))$ is true
$Z(s(A))$ is false
$S(A,B)$ is false if: $A \neq s(B)$
$P1(g(A,B), C)$ is false if: $C \neq A$
$P2(g(A,B), C)$ is false if: $C \neq B$
$R1(g(A,B), g(C,D))$ is false if: $C \neq s(B)$
$R2(g(A,B), g(C,D))$ is false if: $D \neq s(B)$
$N(g(A,B), g(C,D))$ is false if: $C \neq s(A) \vee (B \neq D)$
$Q(g(A,B), C)$ is false if: $C \neq s(B)$

L'aide de l'utilisateur est essentielle ici pour préciser le *domaine* de l'interprétation, alors que l'interprétation des symboles de prédicats peut être construite de façon purement automatique.

Le traitement de ce type de formules logiques montre que notre système peut apporter une aide effective à un utilisateur (logicien ou mathématicien) dans le traitement de formules complexes qui ne peuvent être résolues par d'autres approches (voir chapitre 17).

Chapitre 17

Bilan et perspectives

"La bêtise consiste à vouloir conclure"
Gustave Flaubert.

Dans ce mémoire, nous nous sommes intéressés au problème de la recherche automatique ou interactive de modèles en Déduction Automatique et à l'étude de certaines applications des techniques introduites. La grande diversité des travaux présentés est orientée vers un seul but bien précis : **définir un système automatique ou interactif de recherche simultanée de preuves et de contre-exemples**. Notre contribution peut être résumée de la façon suivante.

1. Nous avons proposé et étudié plusieurs méthodes pour la construction de modèles, finis ou infinis.

   - **Une nouvelle méthode de construction de modèles finis**, nommée *FMC*, a été définie. Il s'agit d'une approche modulaire, fondée sur une énumération des interprétations, et utilisant des techniques efficaces pour réduire l'espace de recherche (notamment les symétries).

   - Nous nous sommes également intéressés à des méthodes de **construction de modèles de Herbrand**. Notre approche est fondée sur l'utilisation de *contraintes* qui permettent de représenter et de manipuler efficacement les interprétations. Les méthodes Ramc et Ramcet définies dans (Caferra et Zabel, 1992; Caferra et Zabel, 1993) ont été étudiées et étendues de façon importante, par la définition de nouvelles règles et stratégies. Une nouvelle méthode de construction de modèles, appelée Eqmc, et combinant la méthode Ramc avec des techniques d'énumération des interprétations a également été proposée. En outre, nous avons identifié certaines classes de formules logiques pour lesquelles les méthodes ci-dessus sont des procédures de décision.

   La figure 17.1 compare les méthodes proposées aux différentes méthodes existantes en construction de modèles et résume leur pouvoir respectif. Les méthodes sont comparées *exclusivement* sur la *classe de modèles qu'elles sont capables de construire* et sans tenir compte des aspects d'efficacité (évidemment très importants en pratique, mais qui restent largement à approfondir).

   Les méthodes Ramc, Eqmc et Ramcet-$2_{\mathcal{I}_s}$ constituent trois approches différentes pour la construction de eq-modèles. Ramc utilise des règles d'inférence et de disinférence et cherche à construire un modèle en générant des conséquences *et des non-conséquences* de la formule initiale. La limite principale de cette approche est — en l'absence de la règle de coupure sur les contraintes — que la classe de modèles constructibles (modèles générables) est intrinsèquement limitée par la syntaxe de la formule initiale (voir chapitres 10 et 8). Au contraire, Eqmc et Ramcet-$2_{\mathcal{I}_s}$ sont des méthodes plus systématiques, dont les propriétés ne dépendent pas de la forme syntaxique de la formule, mais plutôt de ses propriétés *sémantiques* (c'est-à-dire de

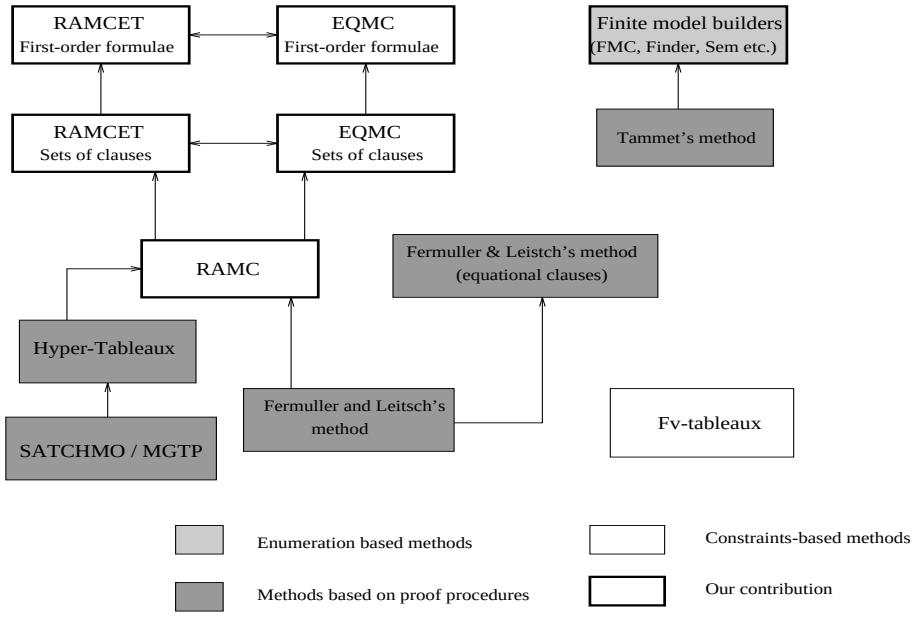


FIG. 17.1 - : *Pouvoirs comparés des méthodes existantes pour la construction de modèles*

l'appartenance à la classe de formules $\mathfrak{C}_{\text{eq-model}}$). Les différences entre ces deux méthodes sont les suivantes.

- EQMC est fondée sur l'utilisation de techniques d'énumération des eq-interprétations tandis que RAMCET-$2_{\mathcal{I}_s}$ utilise des techniques d'*induction*, consistant à généraliser les modèles partiels générés pendant la construction du tableau.
- RAMCET est complet pour la réfutation ce qui n'est pas le cas de EQMC.
- Du point de vue de l'efficacité, EQMC est particulièrement dépendante de la taille de l'ensemble de représentation choisi. En effet, si l'ensemble de représentation est fixé et contient n littéraux, il est clair que le nombre des branches générées par décomposition est alors (dans le pire des cas) 2^n. Par conséquent, la méthode risque de se révéler impraticable si n est trop grand. Au contraire, l'efficacité de RAMCET-$2_{\mathcal{I}_s}$ ne dépend pas de la taille de l'ensemble de représentation. Ainsi, la méthode RAMCET-$2_{\mathcal{I}_s}$ semble plus efficace en pratique lorsque la profondeur du eq-modèle est importante.

2. Nous avons étudié dans un deuxième temps le problème de la *représentation des interprétations*. Après avoir montré les limites des techniques de représentation existantes, nous avons proposé de nouveaux formalismes (fondés sur les techniques de schématisation de termes et d'automates d'arbres) permettant d'étendre la classe d'interprétations représentables. Nous avons notamment prouvé la décidabilité de la théorie du premier ordre dans les termes avec exposants entiers, ce qui permet l'utilisation de ce type de termes pour la représentation et la construction des modèles. La figure 17.2 représente les différentes classes de représentations existantes, ordonnées suivant la classe de modèles qu'elles permettent de représenter.

3. Enfin, nous nous sommes intéressés à certaines applications de la méthode : la recherche de **preuves ou de contre-exemples par analogie** (en collaboration avec Gilles Défourneaux) et la **Programmation Logique**. Une méthode de recherche de preuves ou de contre-exemples par analogie fondée sur des techniques de généralisation et de filtrage d'ordre supérieur a été proposée.

Le prototype d'un système implémentant certaines des idées proposées a été réalisé. Il fait partie intégrante du système ATINF.

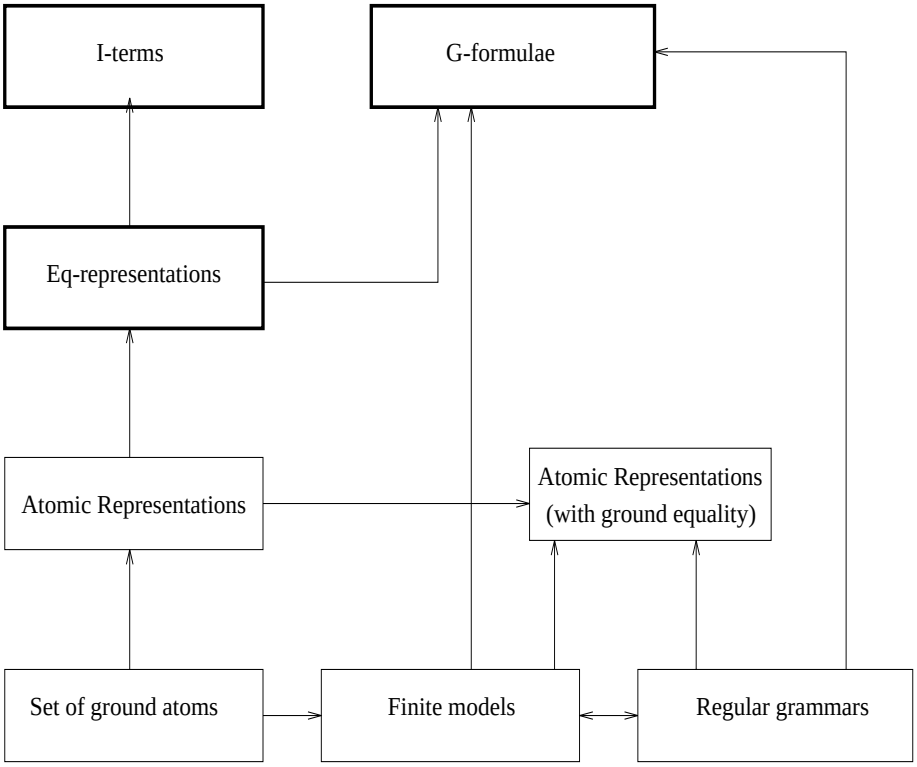


FIG. 17.2 - : *Pouvoirs comparés des formalismes de représentation de modèles*

L'approche de la construction de modèles utilisé au sein du projet ATINF a comme effet de bord de constituer une application des travaux de recherche sur les contraintes, ce qui peut éventuellement guider les recherches théoriques dans ce domaine et fournir un cadre pour tester la complexité pratique des méthodes de résolution.

Cette étude est cependant loin d'être achevée. De nombreux problèmes restent à résoudre. Ils ont été énumérés au fur et à mesure de l'exposé, dans la conclusion de chaque chapitre, aussi nous ne mentionnerons ici que les problèmes qui paraissent les plus intéressants et les plus importants.

1. Tout d'abord, il convient d'étudier de façon plus complète et détaillée les méthodes que nous avons proposées, notamment de comparer leurs performances et leur complexité théorique. Cela passe par la poursuite de l'étude théorique (stratégies, etc.) de ces approches, mais aussi par un travail important d'implémentation et d'expérimentation exhaustive. En particulier, l'implémentation d'un système plus efficace de résolution des contraintes est une étape préliminaire qui nous paraît incontournable. Il paraît également important de pouvoir définir et intégrer des techniques efficaces pour l'implémentation des règles d'inférence et de disinférence de notre méthode (par exemple, des algorithmes similaires à ceux décrits dans (GOTTLOB ET LEITSCH, 1985) pour la subsomption pourraient être adaptés et utilisés pour la règle de dissubsomption). Une implémentation plus efficace et complète des nouvelles méthodes EQMC et RAMCET-$2_{\mathcal{I}_s}$ que nous avons proposées et une étude plus poussée de ces approches semblent en particulier intéressantes et prometteuse. Un tel travail, qui exige des efforts de plus longue haleine, ne peut pas être envisagé dans la période de rédaction d'une thèse.

2. Il paraît également important de chercher à *combiner* les méthodes existantes notamment les méthodes de construction de modèles finis (par exemple *FMC*) avec des méthodes déductives (telles que RAMC, RAMCET-$2_{\mathcal{I}_s}$, ou l'hyperrésolution). Cela permettrait de tirer parti des avantages de chaque approche. L'utilisation de clauses contraintes apparaît comme un cadre privilégié pour l'intégration et l'unification des méthodes existantes pour la construction de modèles.

3. En outre, il apparaît nécessaire de trouver d'autres formalismes de représentation des interprétations. Des schématisations plus générales que les I-termes peuvent être utilisées, telles que les grammaires primales (HERMANN, 1994). Cependant, la décidabilité de la théorie du premier ordre des grammaires primales est pour l'instant un problème ouvert. D'autres techniques (fondées par exemple sur l'utilisation de contraintes d'ordonnancement) peuvent également être utilisées. Cela pourrait notamment permettre d'intégrer dans notre approche l'utilisation de stratégies d'ordonnancement pour la méthode de résolution, utilisée de façon intensive pour la décision de certaines classes de formules (TAMMET, 1991; FERMÜLLER ET AL., 1993).

4. Enfin une étude plus précise et plus poussée de l'utilisation des symétries en construction de modèles finis apparaît comme une nécessité pour l'amélioration des performances de l'algorithme de construction de modèles finis. En particulier, est-il possible d'énumérer de façon efficace les orbites du groupe de symétrie (c'est-à-dire les classes d'équivalence de la relation de symétrie) au lieu d'énumérer l'ensemble des interprétations?

5. La construction de modèles pour des formules telles que celle de (GOLDFARB, 1984) est un premier exemple montrant que les résultats de cette thèse et les logiciels qui en sont l'implémentation permettent d'envisager des systèmes d'*aide effective* en recherche et enseignement en Logique ou en Mathématique, dans des tâches pour lesquelles seule une assistance très limitée peut être obtenue actuellement (voir par exemple (SLANEY, 1993)). En particulier, nous espérons pouvoir étudier par la suite certaines applications de nos travaux dans le cadre de l'aide à l'enseignement de la géométrie, grâce à une collaboration avec le projet CABRI (plus précisément le projet CABRI-EUCLIDE).

La construction de modèles s'affirme de plus en plus comme un nouveau sous-domaine à part entière au sein de la Déduction Automatique. La difficulté du sujet — il a fallut attendre presque 40 ans pour qu'il soit enfin reconnu comme tel — et la diversité des approches possibles et surtout des applications en font un domaine extrêmement riche qui s'annonce particulièrement prometteur. Nous espérons pouvoir approfondir certains de ces problèmes par la suite, notamment dans le cadre d'une collaboration européenne (éventuellement plus large) entre les diverses équipes travaillant sur la construction de modèles.

Table des matières

Table des figures

Index

Bibliographie

Akers, S. (1978). Binary decision diagrams. *IEEE Trans. Computer*, C-27:509–516.

Amaniss, A. (1996). *Méthodes de schématisation pour la démonstration automatique.* PhD thesis, Université Henri Poincaré. Nancy 1.

Amaniss, A., Hermann, M., et Lugiez, D. (1993). Etude comparative des méthodes de schématisation de séquences infinies de termes du premier ordre. Research Report 93-R-114, Centre de Recherche en Informatique de Nancy.

Apt, K. et Bol, R. (1994). Logic programming and negation: a survey. *Journal of Logic Programming*, 19,20:9–71.

Apt, K. R. (1990). Logic Programming. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, chapter 10, pages 493–574. Elsevier.

Baumgartner, P., Furbach, U., et Niemela, I. (1996). Hyper-tableaux. In *Logics in AI, JELIA'96.* Springer.

Benhamou, B. et Sais, L. (1994). Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1):89–102.

Bledsoe, W. W. (1977). Non-resolution theorem proving. *Artificial Intelligence*, 9:1–35.

Bogaert, B. et Tison, S. (1992). Equality and disequality constraints on direct subterms in tree automata. In *Symposium on Theoretical Aspects of Computer Science*, pages 436–449. LNCS 577.

Börger, E., Grädel, E., et Gurevich, Y. (1997). *The Classical Decision Problem.* Springer, Perspective in Mathematical Logic.

Bourely, C. (1992). Construction de modèle et problèmes équationnels. Rapport de DEA.

Bourely, C. (1998). *RAMCET. Théorie et expérimentation.* PhD thesis, INPG, Grenoble. A paraître.

Bourely, C., Caferra, R., et Peltier, N. (1994). A method for building models automatically. Experiments with an extension of Otter. In *Proceedings of CADE-12*, pages 72–86. Springer. LNAI 814.

Bourely, C., Défourneaux, G., et Peltier, N. (1996). Building proofs or counterexamples by analogy in a resolution framework. In *Proceedings of JELIA 96, LNAI 1126*, pages 34–49. Springer, LNAI.

Bourely, C., Défourneaux, G., et Peltier, N. (1997). Semantic generalizations for proving and disproving conjectures by analogy. *Journal of automated Reasoning*, (Special issue of JELIA's best papers). To appear.

Bourely, C. et Peltier, N. (1996). $\text{Disc}_{\text{ATINF}}$: a system for implementing strategies and calculi. In *Proceeding of DISCO'96.* Springer.

Boy de la Tour, T. (1992). An optimality result for clause form translation. *Journal of Symbolic Computation*, 14:283–301.

Boy de la Tour, T. (1996). Ground resolution with group computations on semantic symmetries. In *Proc. of CADE-13, LNAI 1104*, pages 478–492. Springer.

Boy de la Tour, T. et Caferra, R. (1987). Proof analogy in interactive theorem proving: A method to express and use it via second order pattern matching. In *Proceedings of AAAI 87*, pages 95–99. Morgan Kaufmann.

Boy de la Tour, T. et Demri, S. (1995). On the complexity of extending ground resolution with symmetry rules. In *Proceeding of IJCAI'95*, pages 289–295. Morgan Kaufmann.

Boy de la Tour, T. et Kreitz, C. (1992). Building proofs by analogy *via* the Curry-Howard isomorphism. In *Proceedings of LPAR 92*, pages 202–213. Springer.

Bry, F. et Yahya, A. (1996). Minimal model generation with positive unit hyper-resolution tableaux. In *Proceeding of Tableaux'96*, LNAI 1071, pages 143–159. Springer.

Bryant, R. (1992). Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 23(3).

Caferra, R. (1986). Note sur la logique. Polycopié ENSIMAG.

Caferra, R. et Herment, M. (1993). $\text{GLEF}_{\text{ATINF}}$: A graphic framework for combining provers and editing proofs. In *Proceeding of DISCO'93*, pages 229–240. Springer.

Caferra, R. et Herment, M. (1995). A generic graphic framework for combining inference tools and editing proofs and formulae. *Journal of Symbolic Computation*, 19(2):217–243.

Caferra, R., Herment, M., et Zabel, N. (1991). User-oriented theorem proving with the ATINF graphic proof editor. In *Fundamentals of Artificial Intelligence Research*, pages 2–10. Springer, LNCS 535.

Caferra, R. et Peltier, N. (1995a). Extending semantic resolution via automated model building: applications. In *Proceeding of IJCAI'95*, pages 328–334. Morgan Kaufman.

Caferra, R. et Peltier, N. (1995b). Model building and interactive theory discovery. In *Proceeding of Tableaux'95*, LNAI 918, pages 154—168. Springer.

Caferra, R. et Peltier, N. (1996a). Decision procedures using model building techniques. In *Proceeding of Computer Science Logic, CSL'95*, pages 130–144. Springer, LNCS 1092.

Caferra, R. et Peltier, N. (1996b). A significant extension of logic programming by adapting model buildings rules. In *Proc. of Extensions of Logic Programming 96*, pages 51–65. Springer, LNAI 1050.

Caferra, R. et Peltier, N. (1997a). Combining inference and disinference rules with enumeration for model building. Workshop on model-based reasoning. IJCAI'97.

Caferra, R. et Peltier, N. (1997b). A new technique for verifying and correcting logic programs. To appear in the Journal of Automated Reasoning.

Caferra, R. et Zabel, N. (1990). Extending resolution for model construction. In *Logics in AI, JELIA'90*, pages 153–169. Springer, LNAI 478.

Caferra, R. et Zabel, N. (1992). A method for simultaneous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13:613–641.

Caferra, R. et Zabel, N. (1993). Building models by using tableaux extended by equational problems. *Journal of Logic and Computation*, 3:3–25.

Caicedo, X. (1978). A Formal System for the Non-Theorems of the Propositional Calculus. *Notre Dame Journal of Formal Logic*, 19(1):147–151.

Chan, D. (1988). Constructive negation based on the completed database. In *Proc of the Fifth International Conference in Logic Programming*, pages 111—125. The MIT Press.

Chen, H. et Hsiang, J. (1991). Logic programming with recurrence domains. In *Automata, Languages and Programming (ICALP'91)*, pages 20–34. Springer, LNCS 510.

Chen, H., Hsiang, J., et Kong, H. (1990). On finite representations of infinite sequences of terms. In *Conditional and Typed Rewriting Systems, 2nd International Workshop*, pages 100–114. Springer, LNCS 516.

Chu, H. et Plaisted, D. A. (1997). CLIN-S: A semantically guided first-order theorem prover. *Journal of Automated Reasoning*, 18:183–188.

Church, A. (1956). *Introduction to Mathematical Logic I*. Princeton University Press, Princeton, USA.

Clark, K. L. (1978). Negation as failure. In Gallaire, H. et Minker, J., editors, *Logic and Databases*, pages 293—322. Plenum Press.

Clark, K. L. et Tarnlund, S. (1977). A first order theory of data and programs. In *Proc. of IFIP-77*, pages 939–944. North-Holland.

Comon, H. (1988). *Unification et Disunification. Théorie et Applications*. PhD thesis, INPG, Grenoble.

Comon, H. (1991). Disunification: a survey. In Jean-Louis Lassez, Gordon Plotkin, editors. Computational logic: Essays in Honor of Alan Robinson. MIT Press.

Comon, H. (1992). On unification of terms with integer exponents. Technical report, LRI, Orsay, France.

Comon, H. (1993). Complete axiomatizations of some quotient term algebras. *Theoretical Computer Science*, 118(2):167–191.

Comon, H. (1995). On unification of terms with integer exponents. *Mathematical System Theory*, 28:67–88.

Comon, H. et Delor, C. (1994). Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216.

Comon, H. et Lescanne, P. (1989). Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–475.

Comon, H. et Treinen, R. (1997). The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science*, 176(1–2):67–87.

Cooper, D. (1972). Theorem proving in arithmetic without multiplication. In Meltzer, B. et Michie, D., editors, *Machine Intelligence 7*, chapter 5, pages 91–99. Edinburgh University Press.

Curien, R., Qian, Z., et Hui-Shi (1996). Efficient second-order matching. In Ganzinger, H., editor, *Proceedings of 7th Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 317–331. Springer.

Davis, M. (1983). The prehistory and early history of automated deduction. In Siekmann, J. et Wrightson, G. E., editors, *Automation of Reasoning. Classical Papers on Computational Logic 1957-1966, Vol. 1*, pages 1–48. Springer-Verlag.

Davis, M. et Putnam, H. (1960). A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215.

Défourneaux, G. (1997). *Un Calcul pour la Découverte et l'Utilisation d'Analogies en Déduction Automatique*. PhD thesis, Institut National Polytechnique de Grenoble.

Défourneaux, G. et Peltier, N. (1997a). Analogy and abduction in automated reasoning. In *Proceedings of IJCAI'97*.

Défourneaux, G. et Peltier, N. (1997b). Partial matching for analogy discovery in proofs and counter-examples. In *Proceedings of CADE 14*. Springer Verlag.

Desbrun, M. (1993). Manuel d'utilisateur d'ingrat. Rapport de stage, ENSIMAG.

Dierkes, M. (1996). Rapport de stage, ENSIMAG 2ème année.

Dierkes, M. (1997). Résolution de contraintes avec sortes pour la construction de modèles. Rapport de DEA, ENSIMAG.

Dörre, J. et Rounds, W. (1992). On subsumption and semiunification in feature algebras. *Journal of Symbolic Computation*, 13(4):441–461.

Dreben, B. et Goldfarb, W. D. (1979). *The Decision Problem, Solvable Classes of Quantificational Formulas*. Addison-Wesley.

Egly, U. et Rath, T. (1996). On the practical value of different definitional translations to normal form. In *Proc. of CADE-13, LNAI 1104*, pages 403–417. Springer.

Emden, M. V. et Kowalski, R. (1976). The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, 23:733–742.

Fagin, R. (1974). Generalized first order spectra and polynomial-time recognizable sets. In *Complexity of Computation, SIAM-AMS Proceedings*, pages 43–73.

Fagin, R. (1993). Finite model theory. a personal perspective. *Theorical Computer Science*, 116:3–31.

Fermüller, C. et Leitsch, A. (1996). Decision procedures and model building in equational clause logic. To appear.

Fermüller, C. et Leitsch, A. (1996). Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–203.

Fermüller, C., Leitsch, A., Tammet, T., et Zamov, N. (1993). *Resolution Methods for the Decision Problem*. LNAI 679. Springer.

Fitting, M. (1985). A Kripke-Kleene semantics for general logic programs. *Journal of Logic Programming*, 2:295–312.

Fitting, M. (1990). *First-Order Logic and Automated Theorem Proving*. Springer.

Flener, P. (1995). *Logic Program Synthesis from Incomplete Information*. Kluwer Academic Publishers.

Floyd, R. (1960). Nondeterministic algorithms. *Journal of the ACM*, 4(14):636–644.

Fujita, M. et Hasegawa (1991). A model generation theorem prover in KL1 using a ramified stack algorithm. In *Proceedings of 8th International Conference Symp. Logic Programming*, pages 1070—1080.

Fujita, M., Slaney, J., et Bennett, F. (1993). Automatic generation of some results in finite algebra. In *Proceedings of IJCAI'93*. Morgan Kaufmann.

Gabbay, D. et Sergot, M. (1986). Negation as inconsistency. *Journal of Logic Programming*, 3:1–35.

Gallier, J. H. (1986). *Logic for Computer Science*. Harper & Row.

Gécseg, F. et Steinby, M. (1984). *Tree automata*. Akadémiai Kiadó, Budapest, Hungary.

Gelder, A., Ross, K., et Schlipf, J. (1991). Well Founded Semantics for Logic Programs. *Journal of the Association for Computing Machinery*, 38(3):621–650.

Gelernter, H., Hansen, J., et Loveland, D. (1983). Empirical explorations of the geometry theorem-proving machine. In Siekmann, J. et Wrightson, G., editors, *Automation of Reasoning, vol. 1*, pages 140–150. Springer. Originally published in 1960.

Gelfond, M. et Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of 5th International Conference and Symposium on Logic Programming*, pages 1070–1080, Seattle. MIT Press.

Gilmore, P. (1960). A proof method for quantification theory: its justification and realization. *IBM J. Res. Dev.*, pages 28–35.

Goldfarb, W. D. (1984). The unsolvability of the Gödel class with identity. *Journal of Symbolic Logic*, 49(4):1237–1252.

Gottlob, G. et Leitsch, A. (1985). On the efficiency of subsumption algorithms. *Journal of the ACM*, 32(2):280—295.

Gurevich, Y. (1982). Toward logic tailored for computational complexity. In *Computation and Proof Theory*, pages 175–316. Lecture Notes in Mathematics 1104.

Hall, R. (1989). Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, pages 39–120.

Hartshorne, Weiss, et Burks. *Collected Papers of C.S. Peirce (1930–1958)*. Harward U. Press.

Hermann, M. (1992). On the relation between primitive recursion, schematization, and divergence. In *Proceeding 3rd Conference on Algebraic and Logic Programming*, pages 115–127. Springer, LNCS 632.

Hermann, M. (1994). Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes. Habilitation, Université de Nancy I, and CRIN-CNRS Inria-Lorraine, Nancy, France.

Hogger, C. J. (1984). *Introduction to Logic Programming*. Academic Press.

Hsiang, J. et Rusinowitch, M. (1986). A new method for establishing refutational completeness in theorem proving. In *Proceeding of CADE'86*, pages 141–152.

Huet, G. (1975). A unification algorithm for typed λ-calculus. *Theorical Computer Science*, 1:27–57.

Immerman, N. (1983). Languages which capture complexity classes. In *Proc. of 15th ACM Symposium on Theory of Computing*, pages 147–152.

Jouannaud, J. et Kirchner, C. (1991). Solving equations in abstract algebras: a rule based survey of unification. In Lassez, J.-L. et Plotkin, G., editors, *Essays in Honor of Alan Robinson*, pages 91–99. The MIT-Press.

Kling, R. (1971). A paradigm for reasoning by analogy. *Artificial Intelligence*, 2.

Klingenbeck, S. (1996). *Counter Examples in Semantic Tableaux*. PhD thesis, University of Karlsruhe.

Kolbe, T. et Walther, C. (1995). Second-order matching modulo evaluation – A technique for reusing proofs. In Mellish, C. S., editor, *Proceedings of IJCAI 95*, pages 190–195. IJCAI, Morgan Kaufmann.

Kunen, K. (1987). Answer sets and negation as failure. In Lassez, J. L., editor, *Proceeding of the Fourth International Conference on Logic Programming*, pages 219–227. The MIT Press.

Lalement, R. (1990). *Logique, réduction, résolution*. Etudes et recherches en Informatique. Masson.

Lee, C. (1959). Representation of switching circuits by binary decision programs. *Bell System Tech. J.*, 38:985–999.

Lee, S.-J. et Plaisted, D. (1992). Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9:25–42.

Lee, S.-J. et Plaisted, D. (1994a). Model finding in semantically guided instance-based theorem-proving. *Fundamenta Informaticae*, 21:221–235.

Lee, S.-J. et Plaisted, D. (1994b). Problem solving by searching for models with a theorem prover. *Artificial Intelligence*, 69:205–233.

Leitsch, A. (1993). Deciding clause classes by semantic clash resolution. *Fundamenta Informaticae*, 18:163–182.

Leitsch, A. (1997). *The resolution calculus*. Springer. Texts in Theoretical Computer Science.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Springer, second edition.

Loveland, D. W. (1984). Automated theorem proving: a quarter century review. In Bledsoe, W. et Loveland, D., editors, *Automated Theorem Proving: After 25 Years*, pages 1–45. American Mathematical Society.

LUGIEZ, D. (1989). A deduction procedure for first order programs. In Levi, F. et Martelli, M., editors, *Proceedings of the sixth International Conference on Logic Programming*, pages 585–599. The MIT Press.

LUGIEZ, D. (1995). Positive and negative results for higher-order disunification. *Journal of Symbolic Computation*.

LUKASIEWICZ, J. (1972). *La Syllogistique d'Aristote*. Philosophies pour l'âge de la science. Armand Colin.

MAHER, M. (1988). Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 248—357. IEEE Computer Society.

MAL'CEV, A. (1971). *The Metamathematics of Algebraic Systems: Collected Papers 1936–1967*, chapter Axiomatizable classes of locally free algebra of various type, pages 262–281. Benjamin Franklin Wells editor, North Holland. chapter 23.

MANTHEY, R. ET BRY, F. (1988). SATCHMO: A theorem prover implemented in Prolog. In *Proc. of CADE-9*, pages 415–434. Springer, LNCS 310.

MATZINGER, R. (1997). Computational representations of Herbrand models using grammars. In *Computer Science Logic, CSL'96*.

MCCUNE, W. (1993). Single axioms for groups and abelian groups with various operations. *Journal of Automated Reasoning*, 10:1–13.

MCCUNE, W. (1995). *Otter 3.0 Reference Manual and Guide*. Argonne National Laboratory. Revision A.

MCCUNE, W. ET HENSCHEN, L. (1985). Experiments with semantic paramodulation. *Journal of Automated Reasoning*, 1:231–261.

MCCUNE, W. W. (1990). OTTER *2.0 Users Guide*. Argonne National Laboratory.

MELIS, E. (1995). A model of analogy-driven proof-plan construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 182–189, Montreal.

MINTS, G. E. (1990). Several formal systems of the logic programming. *Computers and Artificial Intelligence*, 9:19–41.

MONGY, J. (1981). *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Villeneuve d'Ascq, France.

MUNYER, J. (1981). *Analogy as a mean of discovery in problem-solving and learning*. PhD thesis, Univ. Calif. Santa Cruz.

OPHELDERS, W. ET DE SWART, H. (1993). Tableaux versus resolution. a comparison. *Fundamenta Informaticae*, 18:109–127.

ORLOWSKA, E. (1991). Relational proof systems for some AI logics. In *International Workshop on Fundamentals of Artificial Intelligence Proceedings*, pages 33–47.

PARAMASIVAM, M. ET PLAISTED, D. (1997). Automated deduction techniques for classification in description logic systems. *Journal of Automated Reasoning*. to appear.

PELTIER, N. (1997a). Increasing the capabilities of model building by constraint solving with terms with integer exponents. *Journal of Symbolic Computation*, 24:59–101.

PELTIER, N. (1997b). A new method for automated finite model building exploiting failures and symmetries. To appear in the Journal of Logic and Computation.

PELTIER, N. (1997c). Simplifying formulae in tableaux. Pruning the search space and building models. In *Proceeding of Tableaux'97*, pages 313–327. Springer. LNAI 1227.

Peltier, N. (1997d). Tree automata and automated model building. To appear in Fundamenta Informaticae.

Peterson, G. (1988). Non-Obviousness-One More Time. *Association for Automated Reasoning Newsletter*, 10:4–5.

Plaisted, D. (1981). Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108.

Plaisted, D. et Greenbaum, S. (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304.

Plotkin, G. (1970). A note on inductive generalization. *Machine Intelligence*, 5:153–163.

Presburger, M. (1929). Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchen die addition als einzige operation hervortritt. In *Comptes Rendus du I congrés de Mathématiciens des Pays Slaves*, pages 92–101.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, 12:23–41.

Salzer, G. (1992). The unification of infinite sets of terms and its applications. In *Logic Programming and Automated Reasoning (LPAR'92)*, pages 409–429. Springer, LNAI 624.

Salzer, G. (1993). On the relationship between cycle unification and the unification of infinite sets of terms. In Snyder, W., editor, *UNIF'93*, Boston, (MA,USA).

Salzer, G. (1994). Primal grammar and unification modulo a binary clause. In *Proc. of CADE-12*, pages 72–86. Springer. LNAI 814.

Sandford, D. (1980). *Using Sophisticated Models in Resolution Theorem Proving*. LNCS 90. Springer-Verlag.

Shepherdson, J. (1984). Negation as failure: a comparison of Clark's completed database and Reiter's closed world assumption. *Journal of Logic Programming*, 1:51–79.

Shepherdson, J. (1985). Negation as failure. *Journal of Logic Programming*, 2:185–202.

Shepherdson, J. (1989). A sound and complete semantics for a version of negation as failure. *Theoretical Computer Science*, 65:343–371.

Slagle, J. R. (1967). Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697.

Slaney, J. (1992). Finder (finite domain enumerator): Notes and guides. Technical report, Australian National University Automated Reasoning Project, Canberra.

Slaney, J. (1993). scott: a model-guided theorem prover. In *Proceedings IJCAI-93*, volume 1, pages 109–114. Morgan Kaufmann.

Slupecki, J., Bryll, G., et Wybraniec-Skardowska, U. (1971). Theory of rejected propositions I. *Studia Logica*, 29:75–115.

Smullyan, R. M. (1968). *First-Order Logic*. Springer.

Sterling, L. et Shapiro, E. (1986). *The Art of Prolog*. The MIT Press.

Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(4):333–355.

Stuckey, P. (1995). Negation and constraint logic programming. *Information and Computation*, 118:12–33.

Suttner, C. et Sutcliffe, G. (1996). The TPTP problem library. Technical report, TU München / James Cook University. V-1.2.1.

SUTTNER, C. B. ET SUTCLIFFE, G. (1995). The TPTP problem library. Technical report, TU München / James Cook University. V-1.2.0.

TAMMET, T. (1991). Using resolution for deciding solvable classes and building finite models. In *Baltic Computer Science*, pages 33–64. Springer, LNCS 502.

TIOMKIN, M. (1988). Proving unprovability. In *Logic In Computer Science'88*, pages 22–26.

TREINEN, R. (1992). A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14(5):437–457.

VARDI, M. (1982). The complexity of relational query languages. In *Proc. of 14th ACM Symposium on Theory of Computing*, pages 137–146.

VENKATARAMAN, K. (1987). Decidability of the purely existential fragment of the theory of term algebra. *Journal of the ACM*, 34(2):492–510.

WEIDENBACH, C. (1993). Extending the resolution method with sorts. In *Proceedings of IJCAI-93*, volume 1, pages 60–65, Morgan Kaufmann.

WINKER, S. (1982). Generation and verification of finite models and counter-examples using an automated theorem prover answering two open questions. *Journal of the ACM*, 29(2):273–284.

WOS, L. (1993a). Automated reasoning. *Notices of the American Mathematical Society*, 40(1):15–26.

WOS, L. (1993b). The kernel strategy and its use for the study of combinatory logic. *Journal of Automated Reasoning*, 10:287–343.

WOS, L. ET MCCUNE, W. (1991). Automated theorem proving and logic programming: a natural symbiosis. *Journal of Logic Programming*, 11(1):1–53.

WOS, L., WINKER, S., MCCUNE, W., OVERBEEK, R., LUSK, E., STEVENS, R., ET BUTLER, R. (1990). Automated reasoning contributes to mathematics and logic. In *Proc. of CADE-10*, pages 485–499. Springer. LNAI 449.

ZHANG, J. (1993). Search for models of equational theories. In *Proceedings of ICYCS-93*, pages 60–63.

ZHANG, J. (1994). Problems on the generation of finite models. In *Proc. of CADE-12*, pages 753–757. Springer. LNAI 814.

ZHANG, J. ET ZHANG, H. (1995). SEM: a system for enumerating models. In *Proc. IJCAI-95*, volume 1, pages 298–303. Morgan Kaufmann.

Annexe A

Vers une résolution plus efficace des contraintes

La résolution des contraintes équationnelles joue dans notre approche un rôle primordial, analogue à celui de l'unification dans la méthode de résolution. Par conséquent, il est important en pratique de disposer de méthodes de résolution de contraintes aussi efficaces que possible. Des expérimentations avec l'algorithme de résolution défini dans (Comon et Lescanne, 1989) montrent les limites de cet algorithme. Elles sont dues principalement à deux facteurs. En premier lieu, la règle-clef d'**explosion** introduit un facteur de branchement important qui entraîne une augmentation considérable de la taille des formules et oblige l'algorithme à traiter beaucoup d'information redondante, car les n problèmes obtenus par l'application de la règle **explosion** sont très similaires. En second lieu, les transformations systématiques en formes normales conjonctives (nécessaires pour éliminer les quantificateurs universels) ou disjonctives (pour éliminer les quantificateurs existentiels) sont très coûteuses. Elles entraînent en effet une augmentation exponentielle de la taille de la formule.

Ces transformations ne peuvent pas être évitées dans le cas général, même s'il existe beaucoup de problèmes pour lesquels ces transformations ne sont pas nécessaires et peuvent donc être évitées. Ainsi, dans (Comon et Delor, 1994) est proposée une méthode de résolution plus efficace évitant l'application systématique de ces transformations. Informellement, les transformations en fnc et fnd sont remplacées par l'application de règles de distributivité (du type $\phi \wedge (\psi_1 \vee \psi_2) \rightarrow (\phi \wedge \psi_1) \vee (\phi \wedge \psi_2)$) qui permettent de *retarder* ces transformations.

Dans ce chapitre, nous cherchons à réduire la complexité de la méthode de résolution des contraintes en améliorant ces deux aspects. Nous proposons un nouvel algorithme permettant de résoudre les contraintes équationnelles dans la théorie vide. Comme l'algorithme de (Comon et Delor, 1994) et à la différence de celui présenté dans (Comon et Lescanne, 1989) notre méthode n'est pas restreinte à des problèmes en forme prénexe. Il permet de traiter directement n'importe quel type de formule du premier ordre. L'algorithme proposé est défini par de nouvelles règles de résolution et par une nouvelle stratégie pour l'application des règles existantes (Comon et Lescanne, 1989; Comon et Delor, 1994). La différence principale avec la méthode de (Comon et Delor, 1994) est que le langage utilisé pour représenter les solutions des formules (c'est-à-dire les "formes normales" obtenues) est plus puissant et plus naturel : nous proposons une nouvelle définition de "formes normales" qui autorise l'occurrence de disjonctions et de quantificateurs universels dans les formules. Les ensembles de solutions sont exprimés d'une façon à la fois plus naturelle et plus concise que par les "définitions avec contraintes" utilisées dans (Comon et Lescanne, 1989; Comon et Delor, 1994) (voir chapitre 2). En outre, le nombre de transformations opérées sur la formule est réduit. Les avantages de notre algorithme peuvent être résumés comme suit.

– En premier lieu, il évite l'application systématique des règles de distributivité. Ce point est important puisque ces règles augmentent de façon exponentielle la taille de la formule. Le but

de notre méthode est de retarder et si possible d'éviter l'application de ces règles. Le principe est de ne les appliquer que si elles sont réellement nécessaires.

- En second lieu, l'algorithme réduit le facteur de branchement introduit par la règle d'**Explosion**. Cette règle transforme un problème donné en un ensemble de problèmes $\mathcal{P}_1, \ldots, \mathcal{P}_n$, où n est le nombre de symboles fonctionnels de la signature. L'application de cette règle entraîne une augmentation rapide de la taille de la formule, tout particulièrement si elle contient beaucoup de quantificateurs imbriqués et si la signature contient beaucoup de symboles. L'algorithme que nous proposons utilise une nouvelle règle dite d'**Explosion Binaire**. Celle-ci est *indépendante de la signature*. Elle remplace un problème donné par deux problèmes, ce qui réduit à la fois la taille de la formule et la complexité de l'algorithme de résolution.

- A l'opposé des approches existantes, notre algorithme s'efforce de réduire la portée des quantificateurs *avant* d'appliquer les règles d'élimination. Comme nous le verrons par la suite (voir section A.3), la taille des formules obtenues est alors réduite dans certains cas. Notre algorithme évite donc la transformation en forme normale prénexe (utilisée par (Comon et Lescanne, 1989)), ainsi que les règles de normalisation telles que la règle $\mathcal{S}_3$ utilisée dans (Comon et Delor, 1994), qui conduisent à une *augmentation* de la portée des quantificateurs et par là même à une augmentation de la taille des formules.

- Le langage utilisé pour représenter les ensembles de solutions est *plus expressif* que les "définitions avec contraintes" utilisées dans (Comon et Lescanne, 1989; Comon et Delor, 1994), au sens où les formes normales ainsi obtenues sont plus simples, et souvent plus faciles à comprendre pour l'utilisateur que les *formes résolues avec contraintes* de (Comon et Lescanne, 1989). Contrairement à (Comon et Lescanne, 1989; Comon et Delor, 1994), nous autorisons la présence de disjonctions et de quantificateurs universels dans les formes normales. Cela permet d'augmenter le pouvoir d'expression du langage des formes normales, c'est-à-dire que les solutions peuvent être exprimées de manière plus concise. Il s'avère que cette forme normale est aussi plus naturelle.

- Notre algorithme peut être utilisé lorsque la signature est *infinie* ou *inconnue*. Dans ce dernier cas, la résolution des contraintes n'est pas toujours possible, puisque aucun axiome de fermeture du domaine (*domain closure axiom*) ne peut être utilisé. La capacité de traiter des signatures inconnues est utile dans le cadre de la construction *incrémentale* de modèles (voir chapitre 7). En effet, dans ce cas, l'ensemble des symboles fonctionnels n'est pas nécessairement connu à priori.

Remarque. Dans ce chapitre, toutes les formules seront supposées ne contenir aucune occurrence du symbole $\forall$. Les sous-formules $\forall x.\mathcal{F}$ sont systématiquement remplaçées par $\neg\exists x.\neg\mathcal{F}$. Cette transformation n'est pas nécessaire (et ne devrait pas être explicitement réalisée dans une implémentation) : elle a uniquement pour objet de simplifier l'écriture des règles. Pour des raisons de lisibilité, nous écrirons quelquefois $\forall x.\mathcal{F}$ à la place de $\neg\exists x.\neg\mathcal{F}$, particulièrement si $\mathcal{F}$ est atomique.

A.1 Algorithme de résolution

Suivant une approche classique, l'algorithme de résolution des formules équationnelles sera décrit par un ensemble de règles de réécriture associé à une stratégie. Les règles transforment toute formule équationnelle en une formule sous une forme particulière appelée "forme normale", dont les solutions sont obtenues de manière immédiate.

Rappelons en premier lieu quelques règles de transformation classiques sur les formules équationnelles : des règles de transformation booléennes et les règles de **Decomposition**, **Clash**,

Occur-check, **Replacement** et **Merging**. Introduisons ensuite une nouvelle règle appelée **Generalized Replacement** dont l'objet sera précisé plus loin. Nous donnons une nouvelle notion de *forme normale* et nous définissons de nouvelles règles d'élimination des quantificateurs. Un contrôle particulier guidant l'application des règles et permettant d'assurer la terminaison du système sera proposé par la suite. La complétude du système (par rapport à la notion de forme normale précédemment introduite) est également établie.

A.1.1 Premières règles de transformation

Dans la suite de ce chapitre, les connectifs $\vee$ et $\wedge$ (resp. le prédicat $=$) sont implicitement considérés comme associatifs et commutatifs (resp. commutatif) : nous ne faisons aucune différence par exemple entre $\mathcal{F} \vee \mathcal{G}$ et $\mathcal{G} \vee \mathcal{F}$. D'autre part, l'ordre dans lequel apparaissent deux quantificateurs existentiels n'a aucune importance pour la sémantique de la formule : i.e. $\exists x.\exists y.\mathcal{F}$ est équivalent à $\exists y.\exists x.\mathcal{F}$.

Nous introduisons un ensemble de règles opérant sur les formules équationnelles. Ces règles sont correctes, c'est-à-dire que la formule obtenue est équivalente à la formule de départ. Nous ne spécifions *pas* ici le contrôle sur l'application des règles.

Règles booléennes

Les règles introduites dans cette section ne dépendent que des propriétés booléennes des symboles logiques $\neg, \vee, \wedge, \top, \bot, \exists$ (elles sont donc correctes dans toute théorie). Elles ont pour objet de simplifier la formule et de transférer les occurrences de quantificateur le plus profondément possible dans la formule.

Nous distinguerons ici deux types de règles.

Règles de simplification. Soit le système suivant (noté **S** dans la suite).

$\mathbf{S}_1$	$\top \vee \phi$	$\rightarrow \top$
$\mathbf{S}_2$	$\bot \vee \phi$	$\rightarrow \phi$
$\mathbf{S}_3$	$\top \wedge \phi$	$\rightarrow \phi$
$\mathbf{S}_4$	$\bot \wedge \phi$	$\rightarrow \bot$
$\mathbf{S}_5$	$\exists x.(\phi \vee \psi)$	$\rightarrow \exists x.\phi \vee \exists x.\psi$
$\mathbf{S}_6$	$\exists x.(\phi \wedge \psi)$	$\rightarrow \exists x.\phi \wedge \psi$ if $x \notin \mathcal{V}ar(\psi)$
$\mathbf{S}_7$	$\neg(\phi \vee \psi)$	$\rightarrow \neg\phi \wedge \neg\psi$
$\mathbf{S}_8$	$\neg(\phi \wedge \psi)$	$\rightarrow \neg\phi \vee \neg\psi$
$\mathbf{S}_9$	$\neg\neg\phi$	$\rightarrow \phi$
$\mathbf{S}_{10}$	$\neg\top$	$\rightarrow \bot$
$\mathbf{S}_{11}$	$\neg\bot$	$\rightarrow \top$

Il est clair que le système **S** est à terminaison finie.

Explosion of Disjunctions (ED). La règle suivante augmente la taille de la formule, mais est nécessaire afin d'assurer la complétude du système.

(ED) $(\phi_1 \vee \phi_2) \wedge \psi \rightarrow (\phi_1 \wedge \psi) \vee (\phi_2 \wedge \psi)$

Règles d'unification

Les règles suivantes permettent de simplifier les formules équationnelles de la forme $t = s$. Il s'agit ici des règles d'unification habituelles.

(Decompose)	$f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$	$\rightarrow \bigwedge_{i=1}^{n} t_i = s_i$
(Clash)	$f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)$	$\rightarrow \bot$ (si $f \not\equiv g$)
(Occur Check)	$x = t$	$\rightarrow \bot$ (si $x \prec t$)

REMARQUE. Des règles similaires pour les négations ne sont évidemment pas nécessaires. En effet, elles seront simulées par les règles d'unification et les règles de transformation booléennes.

Par exemple la formule $f(x,y) \neq f(x',y')$, i.e. $\neg(f(x,y) = f(x',y'))$ sera transformée en $\neg(x = x' \wedge y = y')$ par la règle **Decompose**, donc en $x \neq x' \vee y \neq y'$.

Les règles d'unification sont correctes et à terminaison finie (voir par exemple (JOUANNAUD ET KIRCHNER, 1991)).

Pour toute formule atomique $s = t$, nous notons $\mathcal{N}(s = t)$ une forme normale de $s = t$ par rapport aux règles d'unification.

Règles de remplacement et de fusion

Les règles **Replacement** et **Merging** sont définies comme suit.

(Replacement) $x = t \wedge \mathcal{P} \rightarrow x = t \wedge \mathcal{P}\{x \rightarrow t\}$

(Merging 1) $x = t \wedge \mathcal{P}[x = s]_p \rightarrow x = t \wedge \mathcal{P}[t = s]_p$

(Merging 2) $x \neq t \vee \mathcal{P}[x = s]_p \rightarrow x \neq t \vee \mathcal{P}[t = s]_p$

Ici encore, la correction de ces règles est immédiate.

Nous introduisons également une nouvelle règle appelée **Generalized Replacement**. L'utilité de cette règle est illustrée par l'exemple suivant.

EXEMPLE A.1.1. Soit la formule

$$(\exists x.y = f(x)) \wedge y \neq a.$$

Le problème est que la règle **Replacement** n'est pas applicable ici, à cause du quantificateur existentiel. Cependant, la règle **Replacement** est applicable sur la forme *prénexe* de $\mathcal{F} : \exists x.(y = f(x) \wedge y \neq a)$. ◇

Une solution évidente à ce problème serait de transférer les quantificateurs existentiels à la position minimale dans $\mathcal{F}$ en appliquant la règle

$$\exists u.\phi \wedge \psi \rightarrow \exists u(\phi \wedge \psi),$$

afin de pouvoir appliquer la règle **Replacement**.

Cette technique est utilisée dans (COMON ET DELOR, 1994). Elle correspond à une transformation *restreinte* en forme normale prénexe. Néanmoins, cette transformation a deux inconvénients : elle empêche l'application de certaines règles de simplification (voir exemple A.3.3) et augmente la taille de la formule obtenue après application de la règle **Explosion** (voir exemple A.3.4). C'est pourquoi nous proposons ici d'utiliser une nouvelle règle appelée **Generalized Replacement**, permettant de surmonter les limites de la règle **Replacement** sans transformer systématiquement la formule sous forme prénexe (restreinte).

(Generalized Replacement) $P[x = t]_p \rightarrow \exists \overline{u}.(x = t \wedge P'\{x \rightarrow t\})$

*Si $P[\bot]_p = \bot$ (modulo **S**) et s'il n'existe aucune position $q < p$ telle que $P_{|q}$ est une négation. $\overline{u}$ sont les variables de t qui sont liées dans P et P' est obtenu à partir de P en supprimant tous les quantificateurs $\exists u$ où $u \in \overline{u}$.*

EXEMPLE A.1.2.

$$\exists x.(y = f(x)) \wedge y \neq a$$
$$\rightarrow \exists x.(y = f(x) \wedge f(x) \neq a)$$

◇

LEMME A.1.1. *La règle **Generalized replacement** est correcte.*

PREUVE. Puisqu'il n'existe aucune position $q < p$ telle que $P_{|q}$ est une négation, P est équivalent à $\exists \overline{u}.P'$ (il suffit de transférer tous les quantificateurs universels $\exists u$ où $u \in \overline{u}$ en tête de la formule en appliquant les règles $\exists x.\exists y.\phi \rightarrow \exists y.\exists x.\phi$, $\exists x.\phi \vee \phi \rightarrow \exists x.(\phi \vee \psi)$ et $\exists x.\phi \wedge \phi \rightarrow \exists x.(\phi \wedge \psi)$), qui sont évidemment correctes. Si $\exists \overline{u}.x = t$ est fausse, alors on a $P = P[\bot]_p$, d'où $P = \bot$ (modulo $\mathbf{S}$). Donc $\exists \overline{u}.x = t$ est vraie et P' est équivalent à $P' \wedge \exists \overline{u}.x = t$ donc à $\exists \overline{u}.(P'\{x \rightarrow t\} \wedge x = t)$. C.Q.F.D.

EXEMPLE A.1.3. Soit $\mathcal{F}$ la formule suivante.

$$\exists z.(x \neq f(u,a) \vee x = f(g(u),z))$$

$\mathcal{F}$ est transformée comme suit.

$$\begin{array}{rl} \mathcal{F} & \\ \rightarrow_{Merging} & \exists z.(x \neq f(u,a) \vee f(u,a) = f(g(u),z)) \\ \rightarrow_{Decompose} & \exists z.(x \neq f(u,a) \vee (u = g(u) \wedge a = z)) \\ \rightarrow_{Occur\ Check} & \exists z.(x \neq f(u,a) \vee (\bot \wedge a = z)) \\ \rightarrow_{S_4} & \exists z.(x \neq f(u,a) \vee \bot) \\ \rightarrow_{S_2} & \exists z.x \neq f(u,a) \\ \rightarrow_{S_6} & x \neq f(u,a) \end{array}$$

◇

A.1.2 Langage de représentation des solutions

Avant de définir les règles permettant d'éliminer les quantificateurs, nous devons préciser la forme des formules que nous voulons obtenir, en définissant formellement la notion de *forme normale*. Une forme normale est une classe particulière de formules utilisée pour représenter les solutions de la formule initiale. Un tel langage doit satisfaire les propriétés suivantes, définies dans (COMON, 1991).

- Toute forme normale syntaxiquement distincte de $\bot$ a au moins une solution.
- Il existe un algorithme "efficace" permettant d'énumérer les solutions de la formule.
- Le langage doit avoir un pouvoir d'expression suffisant pour représenter les solutions de façon *concise* et *naturelle*. C'est important, tout d'abord pour des raisons d'efficacité (la taille des formules doit être aussi réduite que possible pour permettre une représentation aisée en machine) mais également pour faciliter l'interaction avec l'utilisateur. En construction de modèle, les formes normales sont utilisées pour représenter les modèles de Herbrand. Cette représentation doit être facilement compréhensible pour un utilisateur.

Dans cette section, nous introduisons une nouvelle classe de formules qui vérifient ces propriétés. Dans la section suivante, nous présenterons un algorithme permettant de transformer toute formule équationnelle du premier ordre en forme normale. Quelques définitions préliminaires sont nécessaires. Introduisons tout d'abord la notion de *problème de complément linéaire* (pcl) et prouvons certaines propriétés de ces formules. Les pcl seront utilisés par la suite pour définir les formes normales.

DÉFINITION A.1.1.

Un problème de complément linéaire *(ou* pcl*)* sur x *est une formule équationnelle de la forme*

$$\bigwedge_{i=1}^{n} \forall \overline{u}_i x \neq t_i$$

où t_i *contient exactement une occurrence de chaque variable de* $\overline{u}_i$, $\mathcal{V}ar(t_i) = \overline{u}_i$. *Pour tout pcl* $\mathcal{F}$, *on note* $E(\mathcal{F})$ *l'ensemble* $\{t_1, \ldots, t_n\}$. ◇

EXEMPLE A.1.4. Soit $\Sigma = \{a, f, g\}$.

Les formules

$$\mathcal{F}_1 : \forall x.y \neq f(x) \wedge y \neq a$$

et

$$\mathcal{F}_2 : \forall x_1.y \neq f(x_1) \wedge \forall x_1, x_2.y \neq g(x_1, x_2) \wedge y \neq a$$

sont des pcl sur y. Les solutions de $\mathcal{F}_1$ sont les substitutions de la forme $\{y \to g(s_1, s_2)\}$ avec $s_1, s_2 \in \tau(\Sigma)$. $\mathcal{F}_2$ *n'a pas de solution* sur la signature car les termes de $\tau(\Sigma)$ sont soit de la forme $f(x_1)$ (pour un certain x_1) soit de la forme $g(x_1, x_2)$ (pour certains x_1, x_2) ou a. ◇

Nous allons définir un algorithme permettant de décider si un pcl donné possède des solutions.

DÉFINITION A.1.2. *Soit* E *un ensemble fini de termes linéaires de sorte* s. *On note* $Inst(E)$ *l'ensemble des instances fermées des termes de* E. ◇

DÉFINITION A.1.3. *Un ensemble de termes linéaires* E *est dite* normalisé *si il n'existe pas de couple* $(s, t) \in E^2$ *tels que* s *est une instance de* t. ◇

REMARQUE. Il est très facile de transformer un ensemble E en un ensemble E' normalisé tel que $Inst(E) = Inst(E')$: il suffit d'enlever de E touts les termes de t qui sont des instances (propres) de termes de E.

DÉFINITION A.1.4. *Soit* E *un ensemble fini de termes linéaires normalisés de sorte* s. *L'ensemble* $\mathcal{T}_E$ *est le plus petit ensemble satisfaisant les conditions suivantes.*

- *Il existe une variable* x *de sorte* s *telle que* $x \in \mathcal{T}_E$.
- *Si* $t \in \mathcal{T}_E$, x *est une variable de sorte* s *dans* t, *et* $f : s_1 \times \ldots \times s_n \to s \in \Sigma$, *alors* $t\{x \to f(x_1, \ldots, x_n)\} \in \mathcal{T}_E$,*avec*
  - *les* x_i $(1 \leq i \leq n)$ *sont des nouvelles variables distinctes de sorte* s_i ;
  - *il existe un nombre fini de symboles de fonction de profil* $\overline{s} \to s$;
  - *il existe un terme* $t' \in E$ *et une substitution* θ *telle que* $t'\theta = t\theta$ *et* $x\theta \notin \mathcal{X}$.

◇

REMARQUE. $\mathcal{T}_E$ est fini, puisque la profondeur des termes dans $\mathcal{T}_E$ ne peut pas être supérieure à celle des termes dans E.

DÉFINITION A.1.5. *Un terme* t *est dit* minimal *dans un ensemble* E, *s'il n'existe aucune instance de* t *dans* E. ◇

LEMME A.1.2. *Soit* E *un ensemble fini de termes linéaires normalisés. Si* $\mathcal{T}_E$ *contient un terme* t *minimal tel que* $t \in \mathcal{T}_E$, x *est une variable de sorte* s *à une position* p *dans* t *et* $f : s_1 \times \ldots \times s_n \to s \in \Sigma$, *où il existe un terme* $t' \in E$ *et une substitution* θ *tels que* $t'\theta = t\theta$ *et* $x\theta \notin \mathcal{X}$.

Alors le pcl $\bigwedge_{t' \in E} \forall \mathcal{V}ar(t').y \neq t'$ *possède une infinité de solutions.*

PREUVE. Soit t un tel terme et soit $\overline{x}$ l'ensemble des variables de t satisfaisant les conditions du lemme. Pour tout $x \in \overline{x}$, il existe un symbole de fonction f_x n'apparaissant pas dans E. Soit σ une substitution fermée associant à chaque variable $x \in \overline{x}$ de t un terme de la forme $f_x(\overline{t})$. Supposons que $t\sigma \in \mathit{Inst}(E)$. Alors il existe $s \in E$ et une substitution θ avec $s\theta = t\sigma$. D'où t et s sont unifiables. De plus, puisque f_x n'apparaît pas dans un terme de E, pour toute position p telle que $t_{|p} \in \overline{x}$, il existe un préfixe q de p tel que $s_{|p} \in \mathcal{V}$. D'où t est une instance de s. Par définition de t, il existe un terme minimal $t' \in E$ tel que t et t' soient unifiables. Puisque t est une instance de s, s et t' sont unifiables. Par construction de T_E soit t' est une instance de s, ce qui est impossible car E est normalisé, soit s est une instance de t', ce qui est impossible puisque par définition de t', il existe une position $q.p \in \mathcal{P}os(t')$ où $q \in \mathcal{P}os(t)$ et $p \neq \Lambda$. Donc $t\sigma \notin \mathit{Inst}(E)$ d'où $\{y \rightarrow t\sigma\} \in \mathcal{S}(\bigwedge_{t \in E} \forall \mathcal{V}ar(t).y \neq t)$. Cela prouve que ce pcl admet une infinité de solutions, car il existe une infinité de substitutions possibles. C.Q.F.D.

Par conséquent, on suppose que les conditions du lemme A.1.2 ci-dessus ne sont pas satisfaites. On définit alors l'ensemble $\overline{E}$ comme l'ensemble des termes $t \in \mathcal{T}_E$ tel qu'il n'existe aucun terme $s \in E$ unifiable avec t.

THÉORÈME A.1.1. *Soit E un ensemble de termes linéaires normalisé de sorte s.*

$$\mathit{Inst}(\overline{E}) = \tau_s(\Sigma) \setminus \mathit{Inst}(E).$$

PREUVE.
- Tous les termes de $\mathit{Inst}(\overline{E})$ sont de la forme $t\sigma$ où $t \in \overline{E}$ et σ est une substitution fermée. Supposons que $t\sigma \in \mathit{Inst}(E)$. Alors il existe un terme s et une substitution θ tels que $s\theta = t\sigma$. D'où il existe deux termes $s \in E$ et $t \in \overline{E}$ tels que s et t sont unifiables, ce qui est impossible par définition de $\overline{E}$.
- Tous les termes de $\tau_s(\Sigma)$ sont de la forme $t\sigma$ avec $t \in \mathcal{T}_E$. Soit t_m un terme de $\mathcal{T}(E)$ tel qu'il existe une substitution θ vérifiant $t_m\theta = t\sigma$ et tel qu'il n'existe aucune instance de t_m possèdant cette propriété. Si $t\sigma \notin \mathit{Inst}(\overline{E})$ on a $t_m \notin \overline{E}$. Par conséquent, il existe $s \in E$ tel que s et t_m sont unifiables. Pour toute position $p \in s$, le terme à la position p dans t_m n'est *pas* une variable (par définition de t_m). Donc puisque t_m est linéaire, il existe une substitution θ telle que $t_m = s\theta$. D'où $t\sigma \in \mathit{Inst}(E)$.

C.Q.F.D.

COROLLAIRE A.1.1. *Soit E un ensemble fini de termes linéaires normalisés.*

$$\bigwedge_{t \in E, \overline{u} = \mathcal{V}ar(t)} \forall \overline{u}.x \neq t \; - \bigvee_{t \in \overline{E}, \overline{u} = \mathcal{V}ar(t)} \exists \overline{u}.x = t$$

Le corollaire A.1.1 peut être utilisé pour transformer un problème de complément linéaire en une contrainte purement existentielle et positive. Cependant, il n'est pas nécessaire d'effectuer explicitement cette opération. En effet, il suffit de vérifier que le pcl possède un nombre infini de solutions comme nous le verrons par la suite. Pour cela, il suffit d'énumérer l'ensemble $\overline{E}$ et de s'arrêter dès que l'on obtient un terme contenant une variable de sorte infinitaire.

EXEMPLE A.1.5. Σ, $\mathcal{F}_1$ et $\mathcal{F}_2$ sont définis comme dans l'exemple A.1.4.

$E(\mathcal{F}_1)$ est $\{a, f(x_1)\}$. On a $\mathcal{T}(E(\mathcal{F}_1)) = \{x, a, f(x_1), g(x_1, x_2)\}$. Donc $\overline{E(\mathcal{F}_1)} = \{g(x_1, x_2)\}$. D'après le corollaire A.1.1 on a

$$\mathcal{F}_1 \equiv \exists x_1, x_2.y = g(x_1, x_2).$$

D'autre part, on a $\overline{E(\mathcal{F}_2)} = \emptyset$ d'où $\mathcal{F}_2 \equiv \bot$. ◇

La définition A.1.6 introduit les formes normales. Contrairement à (COMON ET LESCANNE, 1989) nous autorisons l'occurrence de quantificateurs universels et de disjonctions dans les formes

normales. Cependant, tous les quantificateurs universels apparaissant dans la formule doivent apparaître dans des pcl.

DÉFINITION A.1.6. *Une formule $\exists\overline{x}.\mathcal{F}$ est dite* en forme normale *si et seulement si elle est syntaxiquement égale à $\top, \bot$ ou si sa forme normale disjonctive est une disjonction de formules de la forme suivante.*

$$\bigwedge_{i=1}^{n} x_i = t_i \wedge \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{m_i} (y_i \neq s_{ij} \wedge \mathcal{C}_i) \ (\star).$$

où :

- *$\mathcal{C}_i$ $(1 \leq i \leq k)$ est un pcl sur y_i avec un nombre infini de solutions ;*
- *x_i $(1 \leq i \leq n)$ n'apparaît qu'une seule fois dans la conjonction ;*
- *$y_i \not\equiv y_j$ (si $i \not\equiv j$, $1 \leq i \leq n$).*

◇

EXEMPLE A.1.6. Soit $\Sigma = \{a, f, g\}$. La formule

$$\forall y.(x_1 \neq f(y)) \wedge x_2 \neq a \wedge x_3 = g(x_4, x_4)$$

est en forme normale. La disjonction de "formes résolues avec contraintes" (voir (COMON ET LESCANNE, 1989) ou chapitre 2) équivalent à cette formule est

$(x_1 = a \wedge x_2 \neq a \wedge x_3 = g(x_4, x_4)) \vee (x_2 = g(u, v) \wedge x_2 \neq a \wedge x_3 = g(x_4, x_4))$. ◇

Le théorème A.1.2 prouve que cette définition est correcte, c'est-à-dire que toute forme normale distincte de $\bot$ possède au moins une solution. *La preuve de ce théorème fournit également un algorithme permettant d'énumérer les solutions d'une formule en forme normale.*

Introduisons tout d'abord une définition.

DÉFINITION A.1.7. *Soit $\mathcal{F}$ une formule et p une position dans $\mathcal{F}$. Le* niveau *de p dans $\mathcal{F}$ (noté $\mathcal{L}evel(p)$) est le nombre de positions $q < p$ telles que $\mathcal{F}_{|q}$ est de la forme $\neg\mathcal{H}$. La polarité de p dans $\mathcal{F}$ (notée polarity(p)) est 1 si $\mathcal{L}evel(p)$ est impair, 0 sinon. La polarité d'une sous-formule $\mathcal{G}$ apparaissant à la position p dans $\mathcal{F}$ est la polarité de p dans $\mathcal{F}$. De même, le niveau (resp. polarité) d'une variable x dans une formule $\mathcal{F}$ est le niveau (resp. polarité) de la sous-formule la plus grande (au sens de $\prec$) de $\mathcal{F}$ contenant x comme variable libre.* ◇

THÉORÈME A.1.2. *Toute formule $\mathcal{F}$ en forme normale et différente de $\bot$ a une solution. D'autre part si $\mathcal{F} \not\equiv \bot$ et si $\mathcal{F}$ ne contient que des formules atomiques de polarité 1 contenant au moins une variable de $\overline{x}$, alors $\exists\overline{x}.\mathcal{F}$ est valide.*

PREUVE. 1. Soit $\mathcal{F}$ une formule en forme normale. Si $\mathcal{F}$ est $\bot$ ou $\top$, la preuve est immédiate. Soit $\bigvee_{i=1}^{n} \mathcal{G}_i$ la forme normale disjonctive de $\mathcal{F}$. Considérons, par exemple, la formule $\mathcal{G}_1$. $\mathcal{G}_1$ est de la forme $(\star)$. Soit θ une substitution des variables de $\mathcal{G}_1$ n'appartenant pas à $\overline{x}$ ni à $x_1, \ldots, x_n, y_1, \ldots, y_k$. Montrons par induction sur k que $\mathcal{G}_1\theta$ possède au moins une solution.

- La preuve est immédiate si $k = 0$ (il suffit de considérer la substitution $\sigma : \{x_i \to t_i\theta\}$).
- Puisque $\mathcal{C}_i$ a une infinité de solutions, il existe un terme t syntaxiquement différent de $s_{i1}\theta, \ldots, s_{i,m_1}\theta$ et tel que la formule $\mathcal{C}_i\theta\{x_1 \to t\}$ est valide.
  Soit $\mathcal{G}$ la formule obtenue en remplaçant dans $\mathcal{G}_1\theta\{x_1 \to t\}$, $\mathcal{C}_i$ par $\top$ et chaque équation $t \neq s_{ij}\theta$ par $\neg\mathcal{F}r(t = s_{ij}\theta)$. Il est clair que $\mathcal{G}_1\{x_1 \to t\}$ est équivalent à $\mathcal{G}$. Soit $\mathcal{G}'$ la forme normale disjonctive de $\mathcal{G}$. Puisqu'il n'existe aucun terme s_{ij} tel que $s_{ij} \equiv t$ pour toute formule $t = s_{ij}$ soit $\mathcal{N}(t = s_{ij})\theta$ est $\top$, soit il contient une équation de la forme $x = s$. Par conséquent, il existe un disjoint $\mathcal{H}$ dans $\mathcal{G}'$ ne contenant aucune équation de la forme $s \neq s$. Puisque $\theta\{x \to t\}$ est fermée, s ne contient pas x. D'où $\mathcal{H}$ est de la forme $(\star)$ donc admet au moins une solution σ (par hypothèse d'induction). Alors $\sigma\{x_1 \to t\}$ est une solution de $\mathcal{G}_1\theta$.

2. Soit θ une substitution des variables non existentielles de $\mathcal{F}$. Soit $\mathcal{G}$ la formule obtenue en remplaçant chaque équation $\neg(s = t)$ de $\mathcal{F}\theta$ par $\neg\mathcal{N}(s = t)$. Puisque toute équation dans $\mathcal{G}$ contient au moins une variable existentielle, les pcls apparaissant dans $\mathcal{F}$ ne contiennent aucune variable libre dans $\mathcal{F}$. De plus pour toute formule atomique f de $\mathcal{F}$, soit $f\theta$ est équivalent à $\top$, soit (puisque f contient au moins une variable existentielle) la forme normale de f contient une équation de la forme $x = t$, où x est existentielle et t est un terme fermé. Soit $\mathcal{G}'$ la formule $\bigwedge_{f\in\mathcal{G}} \neg l(f) \wedge \bigwedge_{i=1}^{k} \mathcal{C}_i$. $\mathcal{G}'$ est de la forme $(\star)$, donc a au moins une solution. De plus, elle ne contient que des variables existentielles. D'où $\exists\overline{x}.\mathcal{G}'$ est valide, et $\mathcal{F}$ est valide.

C.Q.F.D.

REMARQUE. Les ensembles de termes fermés représentés par les pcl pourraient également être représentés par des *contraintes d'appartenance* (voir (COMON ET DELOR, 1994)), au lieu d'utiliser des quantificateurs universels. Par exemple, la formule $\forall y.x \neq g(y) \wedge \forall y.x \neq f(y)$ est équivalente à la formule $x \in \neg(g(\top_s) \vee f(\top_s))$, où $\top_s$ dénote l'ensemble des termes clos. Voir (COMON ET DELOR, 1994) pour plus de détails à propos des contraintes d'appartenance.

A.1.3 Elimination des quantificateurs

Les deux règles suivantes permettent d'éliminer certains des quantificateurs apparaissant dans la formule. Avant de définir formellement ces règles, nous avons besoin d'une définition.

DÉFINITION A.1.8. *Soit $\exists x.\mathcal{F}$ une formule. Soit E l'ensemble des formules atomiques de $\mathcal{F}$ qui contiennent x. La variable x est dite* presque résolue *si et seulement si*

- *Toute formule de E est de polarité 1 dans $\mathcal{F}$.*
- *Toute variable de E est libre dans $\mathcal{F}$.*
- *La formule $\exists x.\bigwedge_{f\in E} \neg f$ est en forme normale.*

◇

(EQ$_1$) $\mathcal{F} \wedge \mathcal{G} \quad \rightarrow \bigvee_{t\in\overline{E(\mathcal{F})}} x = t \wedge \mathcal{G}\{x \rightarrow t\}$
Si $\mathcal{F}$ est un pcl, $Inst(\overline{E(\mathcal{F})})$ est fini.
(EQ$_2$) $\exists x.\mathcal{F} \quad \rightarrow \mathcal{F}'$
Si x est presque résolue ;
$\mathcal{F}'$ est obtenue à partir de $\mathcal{F}$ en remplaçant toute formule atomique contenant x par $\bot$.
(EQ$_3$) $\exists x.x = t \rightarrow \top$

EXEMPLE A.1.7. (**Elimination of Quantifiers**) Soit $\Sigma = \{a, b, f\}$.

$$\begin{aligned}
&\exists x.\forall u.x \neq f(x) \wedge y = f(x)\\
&\rightarrow_{\mathbf{EQ}_1} \exists x.(x = a \wedge y = f(a)) \vee (x = b \wedge y = f(b))\\
&\rightarrow_{\mathbf{S}_5} (\exists x.x = a) \wedge y = f(a) \vee (\exists x.x = b) \wedge y = f(b)\\
&\rightarrow_{\mathbf{EQ}_3} y = f(a) \vee y = f(b)
\end{aligned}$$

◇

Soit $\mathcal{A}$ et $\mathcal{B}$ deux formules ne contenant pas la variable x.

$$\exists x.(x \neq a \vee \mathcal{A}) \wedge (x \neq b \vee \mathcal{B})$$
$$\rightarrow_{\mathbf{EQ}_2} (\top \vee \mathcal{A}) \wedge (\top \vee \mathcal{B})$$
$$\rightarrow_{\mathbf{S}} \top$$

L'utilisation de la règle $\mathbf{EQ}_2$ permet ici d'éviter de dupliquer les formules $\mathcal{A}$ et $\mathcal{B}$ lors de l'élimination de x et de ne pas augmenter pas la taille de la formule.

LEMME A.1.3. *Les règles (EQ_1), (EQ_2) et (EQ_3) sont correctes.*

PREUVE. 1. C'est une conséquence triviale du corollaire A.1.1.

2. Puisque x apparaît seulement dans des formules atomiques de polarité 1 on a $\mathcal{S}(\exists x.\mathcal{F}) \subseteq \mathcal{S}(\mathcal{F}')$. Il suffit par conséquent de montrer que $\mathcal{S}(\mathcal{F}') \subseteq \mathcal{S}(\exists x.\mathcal{F})$. Soit σ une solution de $\mathcal{F}'$. Soit E l'ensemble des sous-formules atomiques de $\mathcal{F}$ qui contiennent x. D'après le théorème A.1.2, $\exists x. \bigwedge_{f \in E} \neg f$ est valide. Donc, il existe un terme t tel que $\sigma' = \sigma \cup \{x \rightarrow t\}$ est une solution de $\exists x. \bigwedge_{f \in E} \neg f$. Par définition, pour toute formule f de E $f\sigma'$ est équivalent à $\bot$. Donc $\mathcal{F}\{x \rightarrow t\}\sigma'$ est équivalente à $\mathcal{F}'$, donc σ' est une solution de $\mathcal{F}$. Par définition, cela implique que σ est une solution de $\exists x.\mathcal{F}$. Donc $\mathcal{S}(\mathcal{F}') \subseteq \mathcal{S}(\exists x.\mathcal{F})$.
D'où $\mathcal{S}(\exists x.\mathcal{F}) = \mathcal{S}(\mathcal{F}')$.

3. Trivial.

C.Q.F.D.

Une nouvelle règle d'Explosion

Le but de la règle d'**Explosion** est d'éliminer les quantificateurs ne pouvant être éliminés par les règles précédentes. C'est le cas lorsque la formule contient une équation de la forme $x = t$, où t contient une variable "dépendant" de x. Nous définissons une nouvelle règle (appelée **Explosion Binaire**), dont l'objet est d'éviter le facteur de branchement important des règles d'**Explosion** définies dans (COMON ET LESCANNE, 1989; COMON ET DELOR, 1994). Avant de spécifier formellement cette règle, nous allons expliquer informellement le principe de la règle **Explosion** et de l'extension que nous proposons.

EXEMPLE A.1.8. Soit $\Sigma = \{a, b, f, g\}$. Soit $\mathcal{F}$ la formule $\neg\exists x.(y = f(x) \wedge x \neq z)$. Afin de trouver les solutions de $\mathcal{F}$, il faut éliminer $\exists x$. Pour cela, il est nécessaire de distinguer plusieurs cas, suivant la valeur de la variable y. y appartient à $\tau(\Sigma)$, donc par définition, y est de l'une des formes suivantes : $a, b, g(u, v), f(u)$. Nous allons donc transformer (en utilisant une règle d'**Explosion**) la formule $\mathcal{F}$ en une disjonction de quatre formules obtenues par l'analyse des valeurs possibles pour y.

$\mathcal{F}_1$ $\neg\exists x.(y = f(x) \wedge x \neq z) \wedge y = a$
$\mathcal{F}_2$ $\neg\exists x.(y = f(x) \wedge x \neq z) \wedge y = b$
$\mathcal{F}_3$ $\exists u, v.\neg\exists x.(y = f(x) \wedge x \neq z) \wedge y = g(u, v)$
$\mathcal{F}_4$ $\exists u.(\neg\exists x.(y = f(x) \wedge x \neq z) \wedge y = f(u))$

Les trois premières formules sont réduites respectivement à $y = a, y = b$ et $\exists u, v.y = g(u, v)$ par application des règles **Replacement** et **Clash**.

La quatrième formule est réduite à $\exists u.(\neg\exists x.f(u) = f(x) \wedge x \neq z) \wedge y = f(u)$ (par la règle **Replacement**) donc à $\exists u.(\neg\exists x.u = x \wedge x \neq z) \wedge y = f(u)$. Nous pouvons alors utiliser les règles **Replacement** et **Elimination of Quantifiers 1** pour obtenir la formule $\exists u.(u \neq z \wedge y = f(u))$.

Cependant, en considérant avec plus d'attention la formule $\mathcal{F}$, il est clair que les problèmes $\mathcal{F}_1$, $\mathcal{F}_2$ et $\mathcal{F}_3$ sont très similaires. Les règles appliquées pour résoudre ces formules sont identiques. Par conséquent, il n'est pas nécessaire de distinguer explicitement les cas $y = a, y = b$ et $y = g(u, v)$. En effet, les problèmes correspondants seront traités exactement de la même façon. Afin d'éviter la duplication du problème, nous allons fusionner ces trois cas en un seul, en ajoutant la formule $(\neg\exists u.y = f(u))$ (i.e. $\forall u.y \neq f(u)$). Cette formule peut être considérée comme une abréviation pour la disjonction $y = a \vee y = b \vee \exists u, v.y = g(u, v)$. Par conséquent, nous n'éliminons pas tous les quantificateurs universels de la formule. Nous n'avons pas besoin d'éliminer explicitement les nouveaux quantificateurs universels introduits dans la formule car ceux-ci sont des pcl. ◇

D'une façon plus formelle, la règle **Explosion Binaire** est définie comme suit.

Explosion Binaire

$P[y = t]_p \rightarrow \neg\exists\overline{u}.y = \rho(t) \wedge P[\bot]_p \vee \exists\overline{u}.y = \rho(t) \wedge P[\rho(t) = t]_p$

Si $\rho(t)$ est un terme obtenu en remplaçant chaque occurrence d'une variable de t par de nouvelles variables et $\overline{u} = \mathcal{V}ar(\rho(t))$.

LEMME A.1.4. *La règle **Explosion Binaire** est correcte.*

PREUVE. Soit σ une substitution de $\mathcal{V}ar(\mathcal{P})$. Deux cas sont à distinguer. Soit $y\sigma$ est de la forme $\rho(t)\theta$ pour une certaine substitution θ, ou $y\sigma$ n'est pas de la forme $\rho(t)\theta$. Si $y\sigma \equiv \rho(t)\theta$, alors P est équivalent à $\exists\overline{u}.y = \rho(t) \wedge P$ i.e. à $\exists\overline{u}.P[\rho(t) = t]_p$ (par (EQ_1)). Si $t \not\equiv \rho(t)\theta$ (pour tout θ) alors P est équivalent à $\forall\overline{u}.y \neq \rho(t) \wedge P$. En outre, on a $y\theta = t - \bot$. C.Q.F.D.

A.2 Propriétés de l'algorithme de résolution de contraintes

Nous prouvons les principales propriétés de notre algorithme : complétude (par rapport à la notion de forme normale introduite à la section A.1.2) et terminaison. En premier lieu, nous introduisons une stratégie particulière d'application des règles des sections A.1.1 et A.1.3. Cette stratégie sera utilisée dans la preuve de terminaison.

A.2.1 Une stratégie d'application des règles

Pour cela, nous allons introduire quelques notations supplémentaires.

Soit $\mathcal{H}$ la formule à résoudre. Si x est une variable de $\mathcal{H}$, nous notons $pos(x)$ la position minimale p telle que x est libre dans $\mathcal{H}_{|p}$. Nous introduisons un ordre partiel sur les variables, défini comme suit. $x < y$ (resp. $x \leq y$) si et seulement si $pos(x) < pos(y)$ (resp. $pos(x) \leq pos(y)$). Une formule quantifiée $\exists x.\mathcal{G}$ est dite *résolue* dans $\mathcal{H}$ s'il n'existe pas de variable y telle que $y < x$ ou si $\exists x.\mathcal{G}$ apparaît dans un pcl de $\mathcal{H}$. Une équation $x = t$ est dite *dangereuse* si elle n'apparaît pas dans un pcl et si t contient une variable y telle que $y > x$ et $polarity(y) \neq polarity(x)$ et si $\mathcal{L}evel(x = t) - \mathcal{L}evel(x)$ est impair.

Une variable x est dite *résolue* dans une formule $\mathcal{G}$ telle que x est libre dans $\mathcal{G}$ si et seulement si x apparaît une seule fois dans $\mathcal{G}$ et si $\mathcal{G}$ est une conjonction de formules contenant en particulier une équation de la forme $x = t$. x est *résolue* si elle est résolue dans $\mathcal{H}_{|pos(x)}$.

Une variable x est dite *maximale* dans une sous-formule $\mathcal{G}$ apparaissant à la position p dans $\mathcal{F}$ seulement si $p = pos(x)$. Une variable x est dite *spéciale* dans une formule $\mathcal{G}$ si x est libre dans $\mathcal{G}$ et soit x est finitaire, soit x apparaît dans une formule de polarité 0 ou dans un pcl de la forme $\forall\overline{u}.x \neq t$ (où $\overline{u} \neq \emptyset$).

$\mathcal{N}^*(\mathcal{G})$ est la formule obtenue en remplaçant dans $\mathcal{G}$ toutes les formules atomiques $s = t$ de $\mathcal{G}$ par $\mathcal{N}(s = t)$. Pour tout terme t, $size(t)$ est la longueur de la position maximale de t.

On impose les restrictions suivantes sur les règles de transformation (voir la définition des règles pour les notations).

– **Merging :** s et t ne sont pas des variables, $size(t) \leq size(s)$.

- **Replacement :** x est maximal dans $\mathcal{F}$ et x n'apparaît dans aucune formule de niveau strictement supérieur à $\mathcal{L}evel(x)$.
- **ED :** ϕ_1 ou (ϕ_2) contient au moins une variable spéciale apparaissant dans ϕ.
- **EQ$_2$:** x n'apparaît dans aucune formule de niveau strictement supérieur à $\mathcal{L}evel(x)$.
- **Explosion Binaire :** x est du même niveau que P et $x = t$ est dangereuse.

Nous notons $\mathcal{R}_{solve}$ la procédure contenant les règles de $\mathcal{R}$ appliquées suivant la stratégie précédente.

A.2.2 Justification informelle

Avant de prouver la terminaison et la complétude du système $\mathcal{R}_{solve}$, nous allons donner une justification informelle de ces conditions.

Les conditions d'application sur la règle **Merging** assurent que le nombre de symboles dans les équations diminue strictement (après application de la règle de décomposition). Les conditions d'application sur la règle **Replacement** assurent que la variable x est résolue après application de la règle, ce qui a pour effet de diminuer le nombre de variables non résolues (ce ne serait pas le cas si x n'était pas maximale). D'autre part, elles assurent également que les sous-formules de niveau supérieur à x restent inchangées. Enfin, la condition sur la règle **Explosion of Disjunctions** n'est *pas* nécessaire pour garantir la terminaison. Elle a simplement pour objet d'éviter des applications inutiles de la règle. Informellement, la règle **Explosion of Disjunctions** n'est utile (pour la complétude du système) *que si elle permet l'application d'une règle de résolution.*

Exemple A.2.1. Considérons par exemple la formule $\mathcal{P}$

$$(x = a \vee x = b) \wedge (y = a \vee y = b).$$

La règle **Explosion of Disjunction** (non restreinte) transforme la formule en sa forme normale disjonctive

$$(x = a \wedge y = a) \vee (x = a \wedge y = b) \vee (x = b \wedge y = a) \vee (x = b \wedge y = b).$$

Cependant, cette transformation est inutile ici car les deux conjoints de $\mathcal{P}$ *ne partagent aucune variable.* ◇

Exemple A.2.2. Considérons la formule

$$(x \neq a \vee y \neq b) \wedge (x \neq b \vee z \neq b)$$

La règle **Explosion of Disjunction** non-restreinte transforme la formule en

$$(x \neq a \wedge x \neq b) \vee (x \neq a \wedge z \neq b) \vee (y \neq b \wedge x \neq b) \vee (y \neq b \wedge x \neq b).$$

Encore une fois, cette transformation est inutile puisqu'aucune règle n'est applicable sur la formule obtenue. ◇

A.2.3 Terminaison et complétude

Théorème A.2.1. *[terminaison] Le système $\mathcal{R}_{solve}$ est à terminaison finie.*

Preuve. Le principe de la preuve est classique. Nous donnons une interprétation $\mathcal{I}$ des formules équationnelles telle que la valeur de $\mathcal{I}$ décroît strictement dès qu'une règle de $\mathcal{R}_{solve}$ est appliquée[1]

Soit $\mathcal{F}$ une formule. On note $\psi(\mathcal{F})$ le multi-ensemble de formules défini comme suit.

- $\psi(\mathcal{F} \wedge \mathcal{G}) = \{x \wedge y / x \in \psi(\mathcal{F}), y \in \psi(\mathcal{G})\}$.

1. En fait, il est possible que l'application de certaines règles augmente la valeur de $\mathcal{I}$, mais nous montrons que les applications suivantes font décroître la valeur de $\mathcal{I}$ à un niveau inférieur au niveau original.

- $\psi(\mathcal{F} \vee \mathcal{G}) = \psi(\mathcal{F}) \wedge \psi(\mathcal{G})$.
- $\psi(s = t) = \{s = t\}$.
- $\psi(s \neq t) = \{s = t\}$.
- $\psi(\exists x.\mathcal{F}) = \psi(\mathcal{F})$.
- $\psi(\neg\exists x.\mathcal{F}) = \{\exists x.\mathcal{F}\}$.
- $\psi(\neg\neg\mathcal{F}) = \psi(\mathcal{F})$
- $\psi(\neg(\mathcal{F}_1 \vee \mathcal{F}_2)) = \psi(\neg\mathcal{F}_1 \wedge \neg\mathcal{F}_2)$
- $\psi(\neg(\mathcal{F}_1 \wedge \mathcal{F}_2)) = \psi(\neg\mathcal{F}_1 \vee \neg\mathcal{F}_2)$

Pour toute conjonction de formules y, on note $atomic(y)$ le multi-ensemble des équations de y et $complex(y)$ le multi-ensemble des formules non-atomiques de y.

On note $\phi(\mathcal{G})$ le multi-ensemble :

$$\phi(\mathcal{G}) = (\{\phi'(x)/x \in \psi(\mathcal{G})\}).$$

avec

- $\phi'(\mathcal{F}) = (\{\phi(\mathcal{G})/\mathcal{G} \in complex(\mathcal{F})\}, nvar(\mathcal{F}), \{max(size(t), size(s))/t = s \in atomic(\mathcal{F})\})$;
- $nvar(\mathcal{F})$ est le nombre de variables x libres non résolues de $\mathcal{F}$ telles que $\mathcal{L}evel(x) = \mathcal{L}evel(\mathcal{F})$.

Soit I l'interprétation des formules équationnelles définie comme suit.

$$I(\mathcal{F}) = (\phi(\mathcal{F}), \mathcal{F})$$

$\mathcal{F}$ est ordonnée en utilisant la relation de réductibilité par les règles de transformation booléennes (le système $\mathbf{S} \cup \{\mathbf{ED}\}$). ϕ est ordonnée par les extensions lexicographiques et multi-ensembles de l'ordre habituel sur les entiers (cf. chapitre 2).

Nous allons prouver que $I(\mathcal{F})$ décroît strictement.

Soit $\mathcal{F}$ une formule. Soit $\rho \in \mathcal{R}_{solve}$. Soit $\mathcal{F}'$ la formule obtenue en appliquant la règle ρ sur $\mathcal{F}$. Alors, par définition, il existe une position p telle que $\mathcal{F}' = \mathcal{F}[\mathcal{G}']_p$, où $\mathcal{G}'$ est obtenue en appliquant la règle ρ à la position Λ sur $\mathcal{G} = \mathcal{F}_{|p}$.

$$\mathcal{F}[\mathcal{G}]_p \rightarrow_\rho \mathcal{F}[\mathcal{G}']_p$$

Nous prouvons maintenant que $\mathcal{I}$ décroît strictement, en considérant tous les cas possibles pour la règle ρ. Voir section A.1 pour les notations.

- **Règles de transformation booléennes.** Il est très facile de vérifier que ϕ n'augmente pas. De plus, $\mathcal{F}$ décroît strictement (par définition).
- **Decomposition**, **Clash**, et **Occur Check**. *nvar* n'augmente pas, et *size* décroît strictement.
- **Merging rule.** Ces règles transforment une équation $x = t$ en $t = s$. De façon évidente, cela n'augmente pas *nvar*. *size* décroît strictement après application de la règle **Decomposition** ou **Clash** (puisque s et t ne sont pas des variables et $size(s) < size(t)$).
- **Replacement** et **Generalized Replacement**. x devient résolue donc *nvar* décroît strictement. A cause du contrôle, les sous-formules de niveau supérieur restent inchangées.
- **Elimination of Quantifiers.**

1. (EQ_1). Le nombre de variables décroît strictement.
2. (EQ_2). Immédiat.
3. (EQ_3). Immédiat.

– **Explosion Binaire.** Cette règle remplace une équation de la forme $y = t$ par des équations entre variables (après application des règles d'unification). Par conséquent, la taille de la sous-formule diminue strictement. En outre, *nvar* n'augmente pas, puisque les variables ajoutées à la sous-formule sont de niveau égal à $\mathcal{L}evel(x)$, donc strictement supérieur au niveau de $x = t$ (puisque $x = t$ est dangeureuse).

C.Q.F.D.

THÉORÈME A.2.2. *[complétude] Soit $\mathcal{F}$ une formule équationnelle irréductible par le système $\mathcal{R}_{solve}$. Alors $\mathcal{F}$ est en forme normale.*

PREUVE. Supposons que $\mathcal{P}$ contient un quantificateur universel $\exists x$ non résolu. Sans perte de généralité, nous supposons que $\exists x$ est le quantificateur non résolu apparaissant au niveau le plus bas dans la formule $\mathcal{F}$, c'est-à-dire un quantificateur apparaissant dans une formule $\exists x.\mathcal{F}$, où $\mathcal{F}$ ne contient que des quantificateurs résolus. Par irréductibilité par rapport à **S**, $\mathcal{F}$ est de la forme $\bigwedge_{i=1}^{k} .\mathcal{P}_i$, où x apparaît dans $\mathcal{P}_i$ $(\forall i \leq k)$. Puisque $\exists x$ est le quantificateur non résolu apparaissant au niveau le plus bas, si $\mathcal{P}_i$ est de la forme $\exists u.\mathcal{R}$ alors u est résolu. Si $\mathcal{P}_i$ est de la forme $\neg\exists u.\mathcal{R}_i$, alors $\mathcal{P}_i$ doit être un pcl (sinon, on aurait $x < u$). Par irréductibilité par rapport à $\mathbf{EQ}_1$ $Inst(\overline{E(\mathcal{P}_i)})$ est infini. Puisque $x \in \mathcal{P}_i$, $\mathcal{P}_i$ est un pcl sur x.

Supposons que $\mathcal{P}_i$ est une équation $t = s$. Par irréductibilité par rapport à la règle **Unification**, t ou s doit être une variable. Supposons que $t \in \mathcal{V}$. Si $t = x$ alors, soit la règle **Generalized Replacement** s'applique, soit la règle **S** et la règle EQ_3 s'appliquent. Donc $t \neq x$. Par irréductibilité par rapport à la règle **Explosion Binaire**, $t = s$ ne doit pas être dangereuse. Donc t est une variable de même niveau que x. D'où soit $\mathcal{L}evel(x) = 0$, soit les règles **Generalized Replacement** et **S** s'appliquent et éliminent t.

Si $\mathcal{P}_i$ est une disjonction, alors par irréductibilité par rapport à la règle **Explosion of Disjunctions**, x ne doit pas être spéciale, d'où la règle $\mathbf{EQ}_2$ s'applique. Par conséquent $\mathcal{F}$ est de la forme $(\star)$. D'autre part, si $\mathcal{L}evel(x) > 0$, toute formule atomique de $\mathcal{F}$ est négative et contient x. D'où $\mathbf{EQ}_2$ s'applique.

Donc $\mathcal{P}$ ne contient aucun quantificateur non résolu. Soit $\exists\overline{x}.\mathcal{P}'$ la forme prénexe de $\mathcal{P}$. Nous montrons par induction structurelle sur l'ensemble des formules que $\exists\overline{x}.\mathcal{P}'$ est en forme normale.

– Si $\mathcal{P}'$ est de la forme $\bigwedge_{i=1}^{n} \mathcal{R}_i$ où $\mathcal{R}_i$ $(1 \leq i \leq n)$ est une formule atomique ou un pcl (avec éventuellement $n = 0$), alors par irréductibilité par la règle **Replacement**, chaque variable x telle que $\mathcal{R}_i$ est de la forme $x = t_i$ apparaît une fois et une seule dans $\mathcal{P}$. D'autre part, par irréductibilité par la règle $\mathbf{EQ}_1$ tout pcl de $\mathcal{P}$ a un nombre infini de solutions.

– Si $\mathcal{P}'$ est de la forme $\mathcal{P}_1 \wedge \mathcal{P}_2$, où $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) n'est pas une conjonction, alors par irréductibilité par la règle **Explosion of Disjunction**, $\mathcal{P}_1$ et $\mathcal{P}_2$ ne partagent aucune variable spéciale. Soit $\mathcal{P}'_1 \equiv \bigvee_{i=1}^{n_1} .f_i$ et $\mathcal{P}'_2 \equiv \bigvee_{i=1}^{n_2} .g_i$ les formes normales disjonctives respectives de $\mathcal{P}_1$ et $\mathcal{P}_2$. Par hypothèse d'induction, $\mathcal{P}'_1$ et $\mathcal{P}'_2$ sont en forme normale. Soit $\mathcal{P}'' = \bigvee_{(i,j)\in[1..n_1]\times[1..n_2]} f_i \wedge g_i$ la forme normale disjonctive de $\mathcal{P}'$. f_i et g_i sont en forme normale. Puisque $\mathcal{P}_1$ et $\mathcal{P}_2$ ne partagent aucune variable spéciale, $f_i \wedge g_i$ est de la forme $(\star)$.

– Si $\mathcal{P}'$ est de la forme $\mathcal{P}_1 \vee \mathcal{P}_2$, la preuve est immédiate : la forme normale disjonctive de $\mathcal{P}'$ est simplement l'union des formes normales disjonctives de $\mathcal{P}_1$ et $\mathcal{P}_2$.

C.Q.F.D.

A.3 Exemples et comparaisons avec les approches existantes

Illustrons le fonctionnement de notre algorithme de résolution par quelques exemples[2] et mettons en évidence les avantages de notre approche par rapport aux méthodes existantes (Comon et Lescanne, 1989; Comon et Delor, 1994).

Le premier exemple montre l'intérêt d'éviter (comme dans (Comon et Delor, 1994), ou dans l'algorithme présenté dans ce chapitre) la transformation systématique en forme normale conjonctive.

Exemple A.3.1. Soit $\Sigma = \{a, g\}$. Soit $\mathcal{F}$ la formule

$$\exists x.\forall y.(x = a \wedge x \neq g(y)) \vee y = z.$$

Notre méthode donne :

$$\begin{array}{ll} \mathcal{F} & \\ \rightarrow_{\text{Merging}} & \exists x.\forall y.(x = a \wedge a \neq g(y)) \vee x = z \\ \rightarrow_{\text{Clash}} & \exists x.\forall y.(x = a \wedge \bot) \vee x = z \\ \rightarrow_{\text{S}} & \exists x.\forall y.(\bot \vee x = z) \\ \rightarrow_{\text{S}} & \forall y.x = z \\ \rightarrow_{\text{S}} & x = z \end{array}$$

Si on applique le système de règles de (Comon et Lescanne, 1989), on obtient, par transformation en forme normale conjonctive, le problème

$$\exists x.\forall y.(x = a \vee x = z) \wedge (x \neq g(y) \vee x = z).$$

Dans ce cas, la règle **Explosion** est nécessaire pour éliminer le quantificateur universel y apparaissant dans la sous-formule $x \neq g(y)$. Par conséquent, nous obtenons deux problèmes différents :

$$\exists x.\forall y.(x = a \vee x = z) \wedge (x \neq g(y) \vee x = z) \wedge x = a$$

et :

$$\exists x, x_1.\forall y.(x = a \vee x = z) \wedge (x \neq g(y) \vee x = z) \wedge x = g(x_1).$$

Ces deux problèmes sont réduits, par **Merging** et **Decomposition**, en :

$$\mathcal{F}_1 : \exists x.\forall y.((x = a \vee x = z) \wedge (x = z) \wedge (x = a))$$

et

$$\mathcal{F}_2 : \exists x, x_1.\forall y.((x = a \vee x = z) \wedge (y \neq x_1 \vee x = z) \wedge x = g(x_1)).$$

$\mathcal{F}_2$ est transformée en $\exists x, x_1.\forall y.((x = a \vee x = z) \wedge (x = z) \wedge x = g(x_1))$. Ainsi, nous obtenons finalement la "définition avec contraintes" suivante :

$$x = a \wedge z = a$$

ou

$$\exists x_1.x = g(x_1) \wedge z = g(x_1).$$

2. Par souci de clarté, nous utilisons le quantificateur $\forall$ dans les exemples à la place de $\neg\exists\neg$.

Cet ensemble de solutions est naturellement équivalent à celui obtenu avec notre méthode. Cependant, la forme normale obtenue et le processus de transformation sont plus simples avec la méthode que nous proposons. L'application de la règle **Explosion** est évitée, ce qui réduit la complexité du processus de résolution. En effet, la transformation en forme normale conjonctive (ou disjonctive) peut dans certains cas *empêcher l'application des règles de simplification* (ici la règle de **Merging** entre $x = a$ et $x \neq g(y)$) (voir (COMON, 1988) pour plus de détails sur ce problème). ◇

L'exemple suivant illustre l'utilisation des règles **Explosion Binaire** et **Elimination of quantifiers 1,2**.

EXEMPLE A.3.2. Soit $\Sigma = \{a, b, f\}$. Soit $\mathcal{F}$ la formule

$$\forall y.x \neq f(y, y) \wedge x \neq b.$$

Notre méthode donne :

$$\begin{aligned}
&\mathcal{F}\\
\rightarrow_{\text{Explosion Binaire}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge \forall y.f(x_1, x_2) \neq f(y, y))\\
&\vee(\forall x_1, x_2.x \neq f(x_1, x_2))) \wedge x \neq b\\
\rightarrow_{\text{Decompose}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge \forall y.(x_1 \neq y \vee x_2 \neq y))\\
&\vee(\forall x_1, x_2.x \neq f(x_1, x_2))) \wedge x \neq b\\
\rightarrow_{\text{Replace}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge \forall y.(x_1 \neq y \vee x_1 \neq x_2))\\
&\vee(\forall x_1, x_2.x \neq f(x_1, x_2))) \wedge x \neq b\\
\rightarrow_{\text{S}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge (\forall y.(x_1 \neq y) \vee x_1 \neq x_2))\\
&\vee(\forall x_1, x_2.x \neq f(x_1, x_2))) \wedge x \neq b\\
\rightarrow_{\text{EQ}_2}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge x_1 \neq x_2)\\
&\vee(\forall x_1, x_2.x \neq f(x_1, x_2))) \wedge x \neq b\\
\rightarrow_{\text{ED}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge x_1 \neq x_2) \wedge x \neq b)\\
&\vee((\forall x_1, x_2.x \neq f(x_1, x_2)) \wedge x \neq b)\\
\rightarrow_{\text{Replace}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge x_1 \neq x_2 \wedge f(x_1, x_2) \neq b))\\
&\vee((\forall x_1, x_2.x \neq f(x_1, x_2)) \wedge x \neq b)\\
\rightarrow_{\text{Decompose}}\ &(\exists x_1, x_2.(x = f(x_1, x_2) \wedge x_1 \neq x_2))\\
&\vee((\forall x_1, x_2.x \neq f(x_1, x_2)) \wedge x \neq b)\\
\rightarrow_{\text{EQ}_1}\ &\exists x_1, x_2.(x = f(x_1, x_2) \wedge x_1 \neq x_2) \vee (x = a)
\end{aligned}$$

REMARQUE. La formule $(\forall x_1, x_2.x \neq f(x_1, x_2)) \wedge x \neq b$ est un pcl avec un nombre fini de solutions, ce qui justifie l'application de la règle **Elimination of Quantifier 1**.

◇

EXEMPLE A.3.3. Soit $\mathcal{F}$ la formule

$$\exists x.(\mathcal{P} \wedge (x = a \vee \mathcal{Q}))$$

où $\mathcal{P}$ et $\mathcal{Q}$ sont deux formules équationnelles et x n'apparaît pas dans $\mathcal{P}$.

Notre méthode donne :

$$
\begin{aligned}
&\mathcal{F}\\
&\rightarrow_{S_6} \ \mathcal{P} \wedge (\exists x.(x = a \vee \mathcal{Q}))\\
&\rightarrow_{S_5} \ \mathcal{P} \wedge (\exists x.x = a) \vee (\exists x.\mathcal{Q})\\
&\rightarrow_{EQ_2} \ \mathcal{P} \wedge (\top \vee \exists x.\mathcal{Q})\\
&\rightarrow_{S_1} \ \mathcal{P} \wedge \top\\
&\rightarrow_{S_3} \ \mathcal{P}
\end{aligned}
$$

En utilisant l'algorithme de (Comon et Delor, 1994) ou (Comon et Lescanne, 1989), nous devons résoudre les sous-formules $\mathcal{P}$ et $\mathcal{Q}$, puis appliquer des règles de distributivité afin de transformer $\mathcal{F}$ en forme normale conjonctive, avant d'éliminer la formule x. En fait, il est inutile ici de résoudre la formule $\mathcal{Q}$ et de transformer la formule sous forme normale conjonctive. Notre méthode permet d'éviter ces deux transformations, ce qui peut être important si $\mathcal{Q}$ est de grande taille. On résoud les sous-formules apparaissant dans le problème *indépendemment* des autres sous-formules *avant* de combiner les solutions obtenues ce qui permet de réduire dans certains cas la complexité de l'algorithme de résolution.

◇

Exemple A.3.4. Soit $\mathcal{F}$ une formule.

$$\exists x.(\forall y.x \neq f(y,y) \wedge \mathcal{Q}).$$

Supposons que x n'apparaît pas dans $\mathcal{Q}$.

Nous devons appliquer la règle **Explosion Binaire** afin d'éliminer la variable x. Notre méthode applique en premier lieu la règle S_6 pour réduire la portée du quantificateur existentiel :

$$(\exists x.\forall y.x \neq f(y,y)) \wedge \mathcal{Q}.$$

Ensuite, la règle **Explosion Binaire** est appliquée.

$$(\exists x.(\forall y_1, y_2.x \neq f(x_1,x_2) \vee \exists y_1, y_2.x \neq f(y_1,y_2) \wedge \forall y.f(y_1,y_2) = f(x,x))) \wedge \mathcal{Q}$$

Après simplification, cette formule est transformée en :

$$(\exists x.(\forall y_1, y_2.x \neq f(x_1,x_2)) \vee \exists x.\exists y_1, y_2.y_1 \neq y_2 \wedge x = f(y_1,y_2)) \wedge \mathcal{Q}$$

Ici, on obtient, en appliquant la règle **Elimination of Quantifiers**, la formule : $\top \wedge \mathcal{Q}$ i.e. $\mathcal{Q}$.

Notre algorithme ne duplique pas la formule $\mathcal{Q}$ lors du processus de résolution.

En revanche, si la règle S_6 n'est pas appliquée pour réduire la portée du quantificateur $\exists x$, la formule $\mathcal{Q}$ peut être inutilement dupliquée.

◇

Annexe B

Correction interactive de programmes impératifs: exemple

Nous avons appliqué (chapitre 14) notre méthode à la détection d'erreur dans des programmes logiques. Il n'est pas difficile de voir qu'elle peut s'appliquer aussi, en principe, à des programmes impératifs. Nous donnons ci-dessous un exemple d'application de la méthode RAMC en vérification de programme impératif. Le programme suivant (en PASCAL) est supposé détecter des cycles de longueur 2 dans un graphe de n noeuds. En fait, pour chaque noeud N, il ne détecte qu'un seul cycle contenant N, et les autres cycles possibles sont perdus.

```
begin
  p := 1;
  while p ≤ n do
    q := 1;
    cyc :=False;
    while (q ≤ n and not cyc) do
      if (arc[p,q] and arc[q,p]) then
        cyc :=True;
        cycle[p,q] :=True;
      (*)
      q := q + 1;
    p := p + 1;
end
```

Pour plus de clarté, nous introduisons les formules suivantes $I(p)$ et $I'(p,q)$. $cycle[p,q]$ est correct si et seulement si $(arc[p,q] \wedge arc[q,p]) - cycle[p,q]$. $I(p)$ signifie que $cycle[p',q]$ est correct si $p' < p$, et *false* is $p' > p$. $I'(p,q)$ signifie que $cycle(p,q')$ est correct si $q' < q$ *false* sinon.

$$I(p) \quad \forall q', p' \in [1;n]^2 \; (1 \leq p' < p) \Rightarrow (cycle(p',q') - (arc(p',q') \wedge arc(q',p')))$$
$$(p < p' \leq n) \Rightarrow (\neg cycle(p',q'))$$

$$I'(p,q) \quad \forall q' \in [1;n] \quad (1 \leq q' < q) \Rightarrow (cycle(p',q') - (arc(p',q') \wedge arc(q',p')))$$
$$(q \leq q' \leq n) \Rightarrow (\neg cycle(p,q'))$$

Après génération des pré et post-conditions, on obtient :

```
begin
{I(1) ∧ I'(1,1)}
p := 1;
{I(p) ∧ I'(p,1)}
while p ≤ n do
  {I(p) ∧ I'(p,1) ∧ p ≤ n}
```

$\{I(p) \wedge I'(p,1)\}$
$q := 1;$
cyc :=**False**;
while ($q \leq n$ **and not** cyc) **do**
$\{I(p) \wedge I'(p,q) \wedge q \leq n \wedge \neg cyc\}$
if ($arc[p,q]$ **and** $arc[q,p]$) **then**
cyc :=**True**;
$cycle[p,q]$:=**True**;
(*)
$\{I(p) \wedge I'(p,q+1)\}$
$q := q+1;$
$\{I(p) \wedge I'(p,q)\}$
$\{I(p) \wedge I'(p,q) \wedge \neg(q \leq n \wedge \neg cyc)\}$
$\{I(p+1) \wedge I'(p+1,1)\}$
$p := p+1;$
$\{I(p) \wedge I'(p,1)\}$
$\{I(p) \wedge I'(p,1) \wedge p > n\}$
end

A partir des pré-conditions et post-conditions du programme, on obtient les formules suivantes.

$$I(p) \wedge I'(p,q) \wedge \neg(q \leq n \wedge \neg cyc) \wedge \neg(I(p+1) \wedge I'(p+1,1))$$

Après transformation en forme clausale et simplification :

1 $[\![cyc \ : (q \leq n)]\!]$
2 $[\![-cycle(x,y) \vee arc(x,y) \ : (1 \leq x < p) \wedge (1 \leq y \leq n)]\!]$
3 $[\![-cycle(x,y) \vee arc(y,x) \ : (1 \leq x < p) \wedge (1 \leq y \leq n)]\!]$
4 $[\![-arc(x,y) \vee -arc(y,x) \vee cycle(x,y) \ : (1 \leq x < p) \wedge (1 \leq y \leq n)]\!]$
5 $[\![-cycle(x,y) \ : (p < x \leq n) \wedge (1 \leq y \leq n)]\!]$
6 $[\![-cycle(p,x) \vee arc(x,y) \ : (1 \leq x < q)]\!]$
7 $[\![-cycle(p,x) \vee arc(y,x) \ : (1 \leq x < q)]\!]$
8 $[\![cycle(p,x) \vee -arc(y,x) \vee -arc(x,y) \ : (1 \leq x < q)]\!]$
9 $[\![-cycle(p,x) \ : (q \leq x \leq n)]\!]$
10 $[\![-cycle(i,j) \vee -arc(i,j) \vee -arc(j,i) \vee cycle(p+1,j) \ : (i \leq p+1)]\!]$
11 $[\![cycle(i,j) \vee arc(i,j) \vee cycle(p+1,j) \ : \top]\!]$
12 $[\![cycle(i,j) \vee arc(j,i) \vee cycle(p+1,j) \ : \top]\!]$
13 $[\![cycle(i,j) \vee cycle(p+1,j) \ : (i \geq p+1)]\!]$
14 $[\![\Box \ : 1 > i \vee i > n]\!]$
15 $[\![\Box \ : 1 > j \vee j > n]\!]$
16 $[\![-cyc \vee cycle(p,q-1) \ : \top]\!]$
17 $[\![cyc \vee -cycle(p,q-1) \ : \top]\!]$

REMARQUE. Ici i,j,n,p,q sont des *constantes* entières (non interprétées) et x,y sont des *variables entières*. Voir (CAFERRA ET PELTIER, 1997b) pour plus de détails sur le traitement des constantes entières lors de la résolution des contraintes.

La méthode RAMC donne :

19 (resolution,5,13) $[\![cycle(i,j) \ : (i \geq p+1)]\!]$
20 (dis-resolution,5,13) $[\![cycle(i,j) \vee cycle(p+1,j) \ : \bot]\!]$
21 (resolution,5,19) $[\![\Box \ : (i \geq p+1)]\!]$
22 (dis-resolution,5,19) $[\![cycle(i,j) \ : \bot]\!]$
23 (resolution,5,10) $[\![-cycle(i,j) \vee -arc(i,j) \vee -arc(j,i) \ : (1 \leq p+1)]\!]$
24 (dis-resolution,5,10) $[\![-cycle(i,j) \vee -arc(i,j) \vee -arc(j,i) \vee -cycle(p+1,j) \ : \bot]\!]$
25 (dis-resolution,4,23) $[\![-cycle(i,j) \vee -arc(i,j) \vee -arc(j,i) \ : (i \leq p+1) \wedge (i \geq p)]\!]$
26 (dis-resolution,4,23) $[\![-cycle(i,j) \vee -arc(i,j) \vee -arc(j,i) \ : (i < p)]\!]$
27 (resolution,4,10) $[\![-arc(i,j) \vee -arc(j,i) \ : (i < p)]\!]$
28 (resolution,5,11) $[\![cycle(i,j) \vee arc(i,j) \ : \top]\!]$
29 (dis-resolution,5,11) $[\![cycle(i,j) \vee arc(i,j) \vee cycle(p+1,j) \ : \bot]\!]$
30 (resolution,2,28) $[\![arc(i,j) \ : (i < p)]\!]$
31 (resolution,5,12) $[\![cycle(i,j) \vee arc(j,i) \ : \top]\!]$
32 (dis-resolution,5,12) $[\![cycle(i,j) \vee arc(j,i) \vee cycle(p+1,j) \ : \bot]\!]$
33 (resolution,3,31) $[\![arc(j,i) \ : (i < p)]\!]$
34 (resolution,30,27) $[\![-arc(j,i) \ : (i < p)]\!]$
35 (resolution,33,24) $[\![\Box \ : (i < p)]\!]$
36 (simplify,21,35) $[\![\Box \ : i \neq p]\!]$
37 (resolution,8,25) $[\![-arc(p,j) \vee -arc(j,p) \ : (1 \leq j < q)]\!]$
38 (resolution,6,38) $[\![arc(p,j) \ : (1 \leq j < q)]\!]$
39 (resolution,7,31) $[\![arc(j,p) \ : (1 \leq j < q)]\!]$
40 (resolution,37,38) $[\![-arc(j,p) \ : (1 \leq j < q)]\!]$
41 (resolution,39,40) $[\![\Box \ : (1 \leq j < q)]\!]$
42 (simplify,15,41) $[\![\Box \ : j < q]\!]$
43 (resolution,9,28) $[\![arc(p,j) \ : \top]\!]$
44 (dis-resolution,9,28) $[\![cycle(i,j) \vee arc(i,j) \ : \bot]\!]$
45 (resolution,9,31) $[\![arc(j,p) \ : \top]\!]$
46 (dis-resolution,9,28) $[\![cycle(i,j) \vee arc(j,i) \ : \bot]\!]$
47 (simplify,35,15,1) $[\![cyc \ : \top]\!]$
48 (resolution,47,16) $[\![cycle(p,q-1) \ : \top]\!]$
49 (resolution,48,6) $[\![arc(p,q-1) \ : \top]\!]$
50 (resolution,48,7) $[\![arc(q-1,p) \ : \top]\!]$

On obtient le modèle suivant.

41 (resolution,39,40) $[\![\Box \ : (1 \leq j < q)]\!]$
45 (resolution,9,31) $[\![arc(j,p) \ : \top]\!]$
47 (simplify,35,15,1) $[\![cyc \ : \top]\!]$
48 (resolution,47,16) $[\![cycle(p,q-1) \ : \top]\!]$
49 (resolution,48,6) $[\![arc(p,q-1) \ : \top]\!]$
50 (resolution,48,7) $[\![arc(q-1,p) \ : \top]\!]$
36 (simplify,21,35) $[\![\Box \ : i \neq p]\!]$

Ce modèle donne un indice qui indique pourquoi le programme n'est pas correct et aide ainsi à corriger le programme initial. Ici cyc := **False** doit remplacer (*). En fait, le booléen cyc est inutile puisque il sera toujours *false*, quel que soit le graphe donné en entrée.

Cet exemple illustre la possibilité d'appliquer notre méthode pour la détection d'erreur dans des programmes impératifs. Notons que la génération des pré et post-conditions, ainsi que le choix de la formule sur laquelle appliquer la méthode, ont été effectués manuellement.