Numéro d'ordre : 2308

Contraintes ensemblistes définies et co-définies : extensions et applications

THÈSE

présentée et soutenue publiquement le 17 juillet 1998

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Jean-Marc Talbot

Composition du jury

Président :	Max Dauchet	Université de Lille 1
Rapporteurs :	Bruno Courcelle	Université de Bordeaux I
	Nevin Heintze	Computing Sciences Research Center - Bell Labs
	Andreas Podelski	Max-Planck-Institut für Informatik - Saarbrücken
Examinateurs :	Gérard Ferrand	Université d'Orléans
	Damian Niwiński	Université de Varsovie
	Philippe Devienne	Université de Lille 1
	Sophie Tison	Université de Lille 1

Université des Sciences et Technologies de Lille
UFR IEEA - Bât. M3 - 59655 Villeneuve d'Asq Cedex
Tél. : 03-20-43-47-24 Fax : 03-20-43-65-66

Mis en page avec la classe thloria.

Remerciements

La page de remerciements est bien souvent la dernière écrite d'une thèse (celle-ci n'échappant pas à la règle), mais également la première (et souvent la seule) lue. L'intéressé y cherchera son nom et le curieux celui des autres. Aussi, rappellerai-je que cette page a été précédée de quelques cent cinquante autres. La rédaction d'une thèse est un moment agréable, car elle concrétise trois années de travail avec ses moments de découragement, d'effervescence et de stress ; c'est aussi un moment tourmenté où le temps ne s'écoule plus en jours et en semaines, mais en caractères et en pages ne s'enchainant jamais assez vite.

Lorsque cette période infernale s'achève, un sentiment de satisfaction vous emplit, mais il est très vite remplacé par celui du doute, car si le mémoire est à la hauteur du travail qu'il a nécessité, il n'est pas à celle de la qualité espérée. Merci donc à Bruno Courcelle, Nevin Heintze et Andreas Podelski d'avoir accepté d'être les rapporteurs de ce travail et d'avoir pris le temps de lire ce mémoire avec toutes les imperfections qu'il comporte.

Je remercie Max Dauchet de m'avoir fait l'amitié, malgré un emploi du temps très chargé, de présider le jury de soutenance. Les discussions que nous avons eues devant une tasse de café font pour moi partie des bons moments de cette thèse.

Merci également à Damian Niwiński d'avoir bien voulu participer à ce jury et de l'intérêt qu'il a porté à mon travail.

Je remercie Gérard Ferrand d'avoir accepté de participer au jury. Je tiens également à remercier à travers lui tous les membres du LIFO, sans qui cette thèse ne serait sans doute pas ce qu'elle est.

Je remercie tout particulièrement Sophie Tison et Philippe Devienne pour la confiance qu'ils m'ont témoignée tout au long de cette thèse, de leur soutien de chaque instant et de la disponibilité dont ils ont fait preuve. Ils ont toujours su conjuguer, lors de nos très nombreuses réunions de travail, « informatique » et « bonne humeur » ; Ils sont le parfait exemple que qualités humaines et scientifiques s'accordent pleinement.

Merci à ma famille et à mes amis de toujours, Emmanuelle, Christelle, Hervé et Eric, du soutien qu'ils ont su m'apporter durant l' « épreuve » de la rédaction.

Merci à Cyrille, compagnon de labeur. Les moments de stress et de cafard ont été bien moins pénibles grâce à lui.

Merci à mes collègues du bureau 318 et des équipes LAC et MÉTHÉOL, tout particulièrement à Bruno, Sébastien, Jean-Stéphane et Jean-Christophe, relecteurs de ce mémoire.

Et enfin, merci à tous les membres du LIFL qui ont fait que la vie de tous les jours me fut agréable.

Table des matières

Table des figures

Introduction

Ce travail de thèse s'inscrit dans une collaboration entre l'équipe METHEOL (Outils et méthodes pour la programmation en logique) et l'équipe LAC (Langages, automates et contraintes) au sein du LIFL (Laboratoire d'Informatique Fondamentale de Lille). Il vise à définir des classes de contraintes ensemblistes *a priori* moins générales que celles qui ont déjà fait l'objet d'une étude au sein de l'équipe LAC [72], mais plus adaptées à l'analyse de programmes logiques, une des motivations de l'équipe METHEOL [44].

Dans ce travail, nous étendons deux classes de contraintes ensemblistes, appelées respectivement contraintes définies [39] et contraintes co-définies [18] en des classes que nous nommons respectivement contraintes ensemblistes définies et co-définies généralisées. Nous présentons des algorithmes, basés sur les automates d'arbres, permettant de tester la satisfiabilité d'un système de contraintes définies et co-définies généralisées et qui, de plus, dans le cas de systèmes satisfiables, construisent respectivement la plus petite et la plus grande solution de ses systèmes. Ce sont ces solutions particulières, qui permettent d'utiliser les contraintes ensemblistes en analyse de programmes logiques.

Cette introduction se compose de trois parties. La première présente, de manière générale, les contraintes ensemblistes. La seconde décrit l'une des principales applications de ce formalisme, l'analyse ensembliste. La troisième partie expose à partir de programmes logiques de forme très simple, les idées intuitives qui sont à la base des algorithmes présentés dans cette thèse.

Les contraintes ensemblistes

Les contraintes ensemblistes forment une thématique de recherche riche de bien des points de vue. Elle regroupe à la fois des aspects théoriques et pratiques, abordés non seulement par de nombreux travaux, mais également par de nombreuses équipes de recherche.

Une contrainte ensembliste est une inclusion $Sexp \subseteq Sexp'$, ou une non-inclusion $Sexp \not\subseteq Sexp'$, où $Sexp$ et $Sexp'$ sont appelées expressions ensemblistes. Voici un exemple d'ensemble (ou de système) de contraintes ensemblistes :

$$\begin{array}{ll} 0 \cup s(N) \subseteq N & X \subseteq L \\ N \subseteq 0 \cup s(N) & X \not\subseteq \bot \\ nil \cup cons(N, L) = L & \end{array}$$

Comme dans l'exemple ci-dessus, les expressions ensemblistes sont formées de variables $L, N, \ldots$, appelées variables ensemblistes, de symboles de fonction tels que $0, nil, s, cons$, d'opérateurs ensemblistes tels que l'union, l'intersection, le complémentaire ou le vide $(\cup, \cap, \complement, \bot)$, ainsi que d'opérateurs de projection de la forme $cons_2^{-1}$.

Le sens donné aux opérateurs ensemblistes $\cup, \cap, \complement, \perp$ est l'interprétation standard de la théorie des ensembles ; les symboles de fonction sont interprétés de manière ensembliste, c-à-d que 0 est vu comme le singleton $\{0\}$ et pour un symbole f admettant deux arguments, l'expression $f(\{a,b\}, \{c,d\})$ est interprété comme l'ensemble de termes $\{f(a,c), f(a,d), f(b,c), f(b,d)\}$. Un opérateur de projection f_1^{-1}, quant-à-lui, appliqué à un ensemble de termes comme par exemple,$\{a, f(b,c), f(d,a), g(e)\}$, produit l'ensemble des premiers sous-termes des termes de cet ensemble dont le symbole de tête est un f, c-à-d ici, $\{b,d\}$.

Une interprétation ensembliste associe à chaque variable ensembliste un ensemble de termes clos et s'étend aux expressions ensemblistes comme ce qui vient d'être présenté. Une interprétation ensembliste $\mathcal{I}$ est solution d'une contrainte $Sexp \subseteq Sexp'$ (respectivement d'une contrainte $Sexp \not\subseteq Sexp'$) si l'ensemble des termes clos $\mathcal{I}(Sexp)$ est inclus (respectivement n'est pas inclus) dans $\mathcal{I}(Sexp')$.

Ainsi, la première contrainte de l'exemple $0 \cup s(N) \subseteq N$ spécifie que 0 doit être un élément de N et que pour tout terme t de N, $s(t)$ doit appartenir à N. Ainsi, une interprétation associant à la variable N, l'ensemble

$$\{0, nil, s(0), s(nil), s(s(0)), s(s(nil)), s(s(s(0))), s(s(s(nil))), \dots\}$$

est une solution de cette contrainte. Cependant, si l'on souhaite que la solution de la première contrainte soit également solution de la seconde $N \subseteq 0 \cup s(N)$, alors celle interprétation ne convient plus, car l'élément *nil* ne saurait être élément d'un ensemble de la forme $0 \cup s(N)$ et ce, quelque soit, la valeur de N. Ainsi, seule une interprétation associant à la variable N l'ensemble des entiers naturels (formels) $\{0, s(0), s(s(0)), s(s(s(0))), \dots\}$ peut être solution des deux premières contraintes.

Cette double inclusion $Sexp \subseteq Sexp'$ et $Sexp' \subseteq Sexp$ peut également s'écrire comme dans la troisième contrainte comme $Sexp = Sexp'$. Une solution pour les trois premières contraintes du système doit donc associer à la variable L, l'ensemble de toutes les listes d'entiers naturels.

Les deux dernières contraintes spécifient le fait que X doit être un sous-ensemble de L, et donc un ensemble de listes d'entiers naturels $X \subseteq L$, et que ce sous-ensemble ne peut être vide $X \not\subseteq \perp$. On peut donc remarquer qu'il n'y a pas en général pour un ensemble de contraintes ensemblistes, unicité de la solution, puisqu'ici une solution de l'ensemble de toutes les contraintes peut associer à X l'ensemble $\{nil\}$, tandis qu'une autre peut lui associer l'ensemble $\{cons(0, nil), cons(s(0), nil), cons(s(s(0)), nil)\}$.

D'un point de vue théorique, l'étude des contraintes ensemblistes et principalement de la satisfiabilité des systèmes de contraintes a amené l'introduction de nouveaux outils ou concepts comme les automates d'ensembles d'arbres [72], le problème d'accessibilité non-linéaire des systèmes d'équations diophantiennes [67], les espaces rationnels [43], mais également l'utilisation de résultats de logique plus anciens [1, 46] comme dans [7, 14, 15].

L'analyse ensembliste

D'un point de vue pratique, les contraintes ensemblistes sont à la base d'une nouvelle technique d'analyse de programmes, appelée analyse ensembliste [37]. Cette technique peut se résumer en deux étapes : la première est la proposition d'un mécanisme d'extraction de contraintes ensemblistes d'un programme et la preuve de correction de ce mécanisme vis-à-vis de la sémantique du programme. La seconde étape est la proposition d'un algorithme de résolution pour

cette classe sous la forme du calcul d'une solution particulière ou d'un test de satisfiabilité.

L'idée sur laquelle se base cette approche est la perte de dépendance pouvant exister entre les valeurs prises par les variables au cours de l'exécution d'un programme, permettant de calculer une sur-approximation de la sémantique d'un programme.

Dans le cas d'un programme impératif, la sémantique de celui-ci peut s'exprimer comme une fonction transformant un environnement, qui décrit les liaisons entre les variables et leur valeur, en un autre environnement.

La perte de dépendance entre les variables se traduit au cours de l'exécution et pour être plus précis, à certains points de cette exécution (appelés points de contrôle) par le fait de ne plus considérer l'ensemble des environnements possibles à ces points, mais un environnement ensembliste associant à chaque variable l'ensemble des valeurs que celle-ci peut prendre.

Détaillons ceci sur un exemple de programme impératif (donné dans [37]) :

```
  L := cons(a,cons(b,nil))
  X := c
1 while (L <> nil) do
 2 X := car(L)
 3 L := cdr(L)
4 enddo
5
```

Les indices se trouvant à gauche du programme sont des points de contrôle, qui marquent les positions où l'analyse est effectuée.

Au premier point de contrôle du programme, les variables L et X contiennent respectivement les valeurs $cons(a, cons(b, nil))$ et c, ce qui se traduit par les deux contraintes

$$c \subseteq X_1$$
$$cons(a, cons(b, nil)) \subseteq L_1$$

Au second point, la valeur de X est soit celle du point 1, lors de l'exécution de la première itération de la boucle, soit du point 4 pour un itération ultérieure. Il en va de même pour la variable L, qui ne peut cependant pas être la liste vide, du fait du test de la boucle.

$$X_1 \cup X_4 \subseteq X_2$$
$$(L_1 \cup L_4) \cap \complement nil \subseteq L_2$$

Au troisième point, la valeur de L est inchangée, tandis que celle de X est celle de la tête de la liste L au point 2, ce qui se traduit par,

$$cons_1^{-1}(L_2) \subseteq X_3$$
$$L_2 \subseteq L_3$$

Similairement, au point 4, on a

$$X_3 \subseteq X_4$$
$$cons_2^{-1}(L_3) \subseteq L_4$$

Et enfin, au point 5, la valeur de X est soit celle de X_1 (dans le cas où aucun exécution de la boucle ne s'est produite), soit celle de X_4. La valeur de L est soit une valeur prise pour L_1, soit pour L_4, mais qui du fait de la condition de la boucle est nécessairement la valeur *nil*,

$$X_1 \cup X_4 \subseteq X_5$$
$$(L_1 \cup L_4) \cap nil \subseteq L_5$$

La plus petite solution de l'ensemble de ces contraintes nous donne une approximation prise par les variables en chacun des points de contrôle :

$$\begin{array}{ll} X_1 = \{c\} & L_1 = \{cons(a, cons(b, nil))\} \\ X_2 = \{a, b, c\} & L_2 = \{cons(a, cons(b, nil)), cons(b, nil)\} \\ X_3 = \{a, b\} & L_3 = \{cons(a, cons(b, nil)), cons(b, nil)\} \\ X_4 = \{a, b\} & L_4 = \{cons(b, nil), nil\} \\ X_5 = \{a, b, c\} & L_5 = nil \end{array}$$

Le résultat produit par cette technique est correct dans le sens, où si une variable X peut prendre, à un certain point de l'exécution du programme, une valeur v, alors cette valeur est contenue dans l'approximation ensembliste de X.

Notre approche

Nous allons, pour donner une idée plus intuitive de notre travail, non pas considérer les contraintes ensemblistes, mais des programmes logiques n'ayant que des symboles de prédicats unaires et une forme particulière.

Notons tout de suite qu'un programme P dont tous les symboles de prédicat $\{p_1, p_2, p_3, \dots\}$ sont unaires peut être vu comme définissant des ensembles $\{E_1, E_2, E_3, \dots\}$ en supposant qu'un terme clos τ appartient à E_i si et seulement si $p_i(\tau)$ appartient au plus petit modèle de Herbrand du programme P. Il est bien évident que cette restriction à des symboles de prédicats unaires n'est pas une limitation du pouvoir d'expression d'un programme logique. Nous allons supposer de plus que les clauses du programmes sont de l'une des formes suivantes :

$$p(a)$$
$$p(f(x_1, \dots, x_m)) \Leftarrow p_1(x_1), \dots, p_m(x_m)$$
$$p(x) \Leftarrow p_1(t_1), \dots, p_l(t_l)$$

où a est une constante, $t_1, \dots, t_l$ sont des termes quelconques et $x, x_1, \dots, x_m$ sont des variables, telles que les $x_1, \dots, x_m$ sont deux-à-deux distinctes.

De tels programmes sont dits quasi-automates. Ce nom vient du fait que les deux premiers types de clause peuvent être vus comme des règles de transition d'un automate d'arbres

$$a \rightarrow p$$
$$f(p_1, \dots, p_m) \rightarrow p$$

où les symboles de prédicat sont les états de cet automate. Le dernier type de clause justifie le terme « quasi ».

Il se trouve que pour tout symbole de prédicat p du programme P, l'ensemble des termes clos τ tel que $p(\tau)$ appartient au plus petit modèle (de Herbrand) de P est un langage régulier, *i.e.*

reconnaissable par un automate d'arbres. Nous allons en fait présenter comment un tel automate peut être construit.

Pour un symbole de prédicat p, on peut considérer la fonction caractéristique de ce prédicat, ξ_p définie comme une application de l'ensemble des termes clos dans $\{0,1\}$ par $\xi_p(\tau) = 1$ si et seulement si $p(\tau)$ appartient au plus petit modèle du programme logique. Si on considère maintenant non plus un seul symbole de prédicat, mais tous ces symboles sous la forme du n-uplet $(p_1, \ldots, p_n)$, alors l'application $\xi_{(p_1,\ldots,p_n)}$ de l'ensemble des termes clos vers $\{0,1\}^n$, les n-uplets de 0 et de 1, définit une fonction caractéristique pour tous les symboles de prédicats en supposant que pour tout i, $\tau \in p_i(\tau)$ si et seulement si la $i^{ème}$ composante de $\xi_{(p_1,\ldots,p_n)}(\tau)$ est égale à 1. Ainsi, le plus petit modèle de Herbrand d'un programme logique quelconque peut être décrit à l'aide d'une telle fonction caractéristique.

En général, il se peut que pour un ensemble quelconque défini par une fonction caractéristique ξ, on ait pour deux termes clos τ, τ', $\xi(\tau) = \xi(\tau')$ et que pourtant, pour un symbole s, $\xi(s(\tau)) \neq \xi(s(\tau'))$, c-à-d que que τ et τ' appartiennent aux mêmes ensembles, sans que cela soit le cas pour $s(\tau)$ et $s(\tau')$.

Cependant, pour les programmes logiques auxquels nous nous intéressons, nous avons la propriété suivante : pour tout symbole de fonction f, si pour tout i, on a l'égalité $\xi(\tau_i) = \xi(\tau'_i)$ alors $\xi(f(\tau_1, \ldots, \tau_m)) = \xi(f(\tau'_1, \ldots, \tau'_m))$. Ceci signifie que la valeur de la fonction caractéristique pour un terme clos est totalement définie par les valeurs de celle-ci pour les sous-termes immédiats de ce terme.

Ceci nous autorise à écrire par exemple, une égalité de la forme $f((1,0,1),(0,0,1)) = (1,0,0)$ traduisant le fait que si τ_1 appartient à l'ensemble défini par les prédicats p_1 et p_3 sans appartenir à celui défini par p_2 et que τ_2 appartient à celui défini par p_3 sans appartenir ni à p_1, ni à p_2, alors $f(\tau_1, \tau_2)$ appartient à l'ensemble défini par p_1 sans appartenir ni à p_2, ni à p_3.

En tenant compte de ceci, on peut affirmer que la fonction caractéristique peut être totalement décrite en précisant pour chaque symbole de fonction f d'arité m une application de $\overbrace{\{0,1\}^n \times \ldots \times \{0,1\}^n}^{m}$ dans $\{0,1\}^n$. Cette définition correspond en fait à la fois, à un automate d'arbres complet et déterministe dont les états sont les n-uplets de 0 et de 1, c-à-d à une algèbre dont le support est lui aussi l'ensemble des n-uplets de 0 et de 1.

Il est à noter que l'algèbre qui est ainsi définie s'étend dans une interprétation « naturelle » correspondant au sens que nous avons donné aux n-uplets de 0 et de 1. Cette interprétation associe à chaque symbole p_i l'ensemble des n-uplets de 0 et de 1, ayant un 1 sur leur $i^{ème}$ composante, autrement dit pour un n-uplet v, $p_i(v)$ est vrai si v a un 1 sur sa $i^{ème}$ composante.

Il faut également remarquer que certains n-uplets de 0 et de 1 ont une interprétation qui est vide, dans le sens où pour la fonction caractéristique ξ définie par l'algèbre de n-uplets de 0 et de 1, il n'existe pas de termes clos τ telle que $\xi(\tau)$ soit égal à l'un de ses états. Ceci peut se résumer par le fait qu'aucun terme clos ne correspond à ces n-uplets.

Le but de l'algorithme est alors la construction d'un automate complet et déterministe dont les états sont les n-uplets de 0 et de 1 « non-vides », et donc, d'une algèbre (ayant pour support ces mêmes n-uplets) impliquant une unique structure comme nous l'avons précédemment définie, devant être modèle du programme logique.

Cette construction peut se voir comme un algorithme de point fixe transformant un automate en un autre automate « simulant » l'opérateur de conséquence logique défini par le programme.

Par exemple, pour le programme suivant,

$$
\begin{aligned}
&p_1(a)\\
&p_2(b)\\
&p_1(f(x)) \Leftarrow p_1(x)\\
&p_2(y) \Leftarrow p_1(y), p_1(f(z)), p_2(z)
\end{aligned}
$$

Le fait qu'il n'y ait que deux symboles de prédicats dans le programme entraîne que les états de l'automate sont des couples de 0 et de 1, la première composante correspondant au prédicat p_1 et la seconde au prédicat p_2.

L'automate initial du calcul correspond à un ensemble vide d'atomes, il associe donc à chaque terme clos le couple $(0,0)$, *i.e.*

$$
\begin{aligned}
&a \rightarrow (0,0)\\
&b \rightarrow (0,0)\\
&f((0,0)) \rightarrow (0,0)\\
&f((0,1)) \rightarrow (0,0)\\
&f((1,0)) \rightarrow (0,0)\\
&f((1,1)) \rightarrow (0,0)
\end{aligned}
$$

Le seul état « non-vide » de l'automate est $(0,0)$, ce qui fait que la structure qu'il définit n'est pas un modèle du programme, puisque en particulier à a est associé $(0,0)$, qu'on a $p_1(a)$ et que pour cette structure, $p_1((0,0))$ est faux.

La première clause stipule que a doit appartenir à l'ensemble défini par p_1. On peut tenir compte de ce fait en modifiant la première règle de l'automate (voir l'automate de gauche ci-dessous),

$$
\begin{aligned}
&a \rightarrow (\mathbf{1},0)\\
&b \rightarrow (0,0)\\
&f((0,0)) \rightarrow (0,0)\\
&f((0,1)) \rightarrow (0,0)\\
&f((1,0)) \rightarrow (0,0)\\
&f((1,1)) \rightarrow (0,0)
\end{aligned}
\qquad\qquad
\begin{aligned}
&a \rightarrow (1,0)\\
&b \rightarrow (0,\mathbf{1})\\
&f((0,0)) \rightarrow (0,0)\\
&f((0,1)) \rightarrow (0,0)\\
&f((1,0)) \rightarrow (0,0)\\
&f((1,1)) \rightarrow (0,0)
\end{aligned}
$$

En considérant la deuxième clause, affirmant que b doit appartenir au langage défini par p_2, on transforme l'automate précédent et on obtient celui de droite ci-dessus.

On peut noter que ces deux transformations font maintenant que pour le terme a, $p_1((1,0))$ est vrai, et pour le terme b, $p_2((0,1))$ est vrai. Cependant, pour le terme $f(a)$, sa valeur dans l'automate est $(0,0)$ et $p_1((0,0))$ est faux. Ainsi la partie « non-vide » de l'automate ne représente toujours pas un modèle du programme.

La troisième clause $p_1(f(x)) \Leftarrow p_1(x)$ affirme que si un terme clos τ appartient à l'ensemble

défini par le prédicat p_1, alors il en va de même pour $f(\tau)$. Ce qui nous donne l'automate,

$$\begin{aligned} a &\rightarrow (1,0) \\ b &\rightarrow (0,1) \\ f((0,0)) &\rightarrow (0,0) \\ f((0,1)) &\rightarrow (0,0) \\ f((1,0)) &\rightarrow (\mathbf{1},0) \\ f((1,1)) &\rightarrow (\mathbf{1},0) \end{aligned}$$

puisque si τ appartient à l'ensemble défini par p_1, alors pour la fonction caractéristique ξ définissant le plus petit modèle du programme, $\xi(\tau)$ a un 1 sur la première composante.

A cet instant du calcul, l'automate définit l'ensemble d'atomes clos suivant

$$\{p_2(b)\} \cup \{p_1(f^n(a)) \mid n \in \mathbb{N}\}$$

La quatrième clause $p_2(y) \Leftarrow p_1(y), p_1(f(z)), p_2(z)$ énonce le fait que pour tout terme clos τ, tel que τ appartient à l'ensemble défini par p_1 doit appartenir à l'ensemble p_2 en supposant qu'il existe un terme τ' tel que τ' appartienne à l'ensemble p_2 et que $f(\tau')$ appartienne à l'ensemble p_1. Pour que cela soit le cas, il est nécessaire que $\xi(\tau')$ ait un 1 sur sa seconde composante et que $\xi(f(\tau')) = f(\xi(\tau'))$ est un 1 sur sa première composante. Ceci est possible au regard de l'automate pour l'état $(1,1)$; cependant, il faut remarquer que cet état correspond à un n-uplet vide, signifiant qu'à ce moment du calcul, aucun terme n'est à la fois dans les ensembles définis par p_1 et par p_2. Ainsi, l'automate ne subit aucune modification.

Les clauses du programme sont alors ré-inspectées une à une comme nous venons de le faire dans le cadre d'itérations successives, jusqu'à ce qu'aucune transformation n'ait été opérée sur l'automate, obtenant ainsi un point fixe de l'algorithme. On peut voir en réexaminant les règles que cela est le cas sur notre exemple.

Lorsque ce point fixe est atteint, on obtient alors une description finie (sous la forme d'une algèbre finie, *i.e.* d'un automate d'arbre déterministe et complet) de la fonction caractéristique du plus petit modèle du programme logique. De plus, en ne considérant dans cette algèbre que les n-uplets de 0 et de 1 non-vides, alors la structure définie par cette algèbre est le plus petit modèle du programme logique défini sur cette algèbre.

Plan de la thèse

Cette thèse se découpe en cinq chapitres, suivant en grande partie la progression temporelle de notre travail :

Le premier chapitre de cette thèse **Outils mathématiques et logiques** regroupe les notions théoriques souhaitables pour aborder ce document. Il introduit de manière concise la logique mathématique, les arbres et les automates, ainsi que la programmation logique, fixant les définitions et les notations qui seront utilisées tout au long de cette thèse.

Le second chapitre **Contraintes ensemblistes : un état de l'art** présente le sujet principal de cette thèse, à savoir les contraintes ensemblistes. Ce second chapitre se scinde en deux sections : la première aborde de manière chronologique les différents travaux menés sur le problème de satisfiabilité, qui ont conduit du problème originel résolu par Heintze et Jaffar pour une classe

de contraintes ensemblistes limitée, appelée *contraintes définies* [39] à la solution du problème générale (laissé comme un problème ouvert dans [39]) proposé par Charatonik et Podelski dans [15]. La seconde section détaille quant-à-elle les sous-classes de contraintes ensemblistes, les plus proches de nos travaux, à savoir les contraintes définies [39], les contraintes co-définies [18] et l'introduction dans la première de ces deux classes d'un opérateur de description intensionnelle d'ensembles [28, 40].

Le troisième chapitre **Contraintes définies : une extension** propose une extension des contraintes ensemblistes définies, nommée contraintes définies généralisées, ainsi qu'un algorithme répondant au problème de satisfiabilité d'un système de telles contraintes, lorsqu'elles sont interprétées sur les ensembles d'arbres finis. Cet algorithme nous permet de montrer que notre extension n'entraîne pas de surcoût du point de vue de la complexité pour le problème de satisfiabilité, puisque celui-ci est comme par les contraintes définies *EXPTIME*-complet. Nous montrons cependant que notre extension est stricte dans le sens où on peut exhiber un système de contraintes définies généralisées tel qu'aucun système défini ne lui est équivalent (*i.e.*, possède les mêmes solutions).

Les contraintes définies généralisées posent les bases d'une généralisation pour la classe des contraintes co-définies, par le lien de dualité par complémentation existant entre ces deux classes. Basé sur l'algorithme qui a été proposé pour la classe définie généralisée , ce lien permet de définir un algorithme pour tester la satisfiabilité d'un système de ces contraintes co-définies généralisées (lorsqu'elles sont interprétées sur les ensembles d'arbres finis).

L'introduction de la classe des contraintes ensemblistes co-définies [17, 18] a été motivée par des considérations d'analyse statique de programmes, qui ont fait que le domaine d'interprétation « naturel » de cette classe sont les ensembles d'arbres finis et infinis. L'objet du quatrième chapitre, intitulé **Contraintes co-définies : une extension** est de considérer l'extension proposée dans le chapitre précédent, mais en s'intéressant à l'interprétation des contraintes co-définies généralisées sur les ensembles d'arbres finis et infinis. Nous proposerons pour cette classe un algorithme répondant au problème de satisfiabilité (qui montre qu'un nouvelle fois, cette extension ne modifie pas la complexité du problème de satisfiabilité), et étudierons deux restrictions de cette extension.

Enfin, le dernier chapitre **Application à l'analyse de programmes logiques** aborde un des domaines principaux d'application des contraintes ensemblistes en synthétisant divers travaux et en donnant quelques pistes pour une amélioration de ceux-ci.

Chapitre 1

Outils mathématiques et logiques

1.1 Généralités

1.1.1 Ensembles et relations

Soit E, un ensemble. On note $\wp(E)$, l'ensemble des sous-ensembles de E. Une relation binaire sur E est un sous-ensemble de $E \times E$, *i.e.* un ensemble de couples (e_1, e_2) avec $e_1, e_2 \in E$. Par extension, une relation n-aire sur E est un sous-ensemble de E^n, dont les éléments sont des n-uplets $(e_1, \ldots, e_n)$ (avec pour tout i, $e_i \in E$). Une relation unaire sur E n'est qu'un sous-ensemble de E.

Une relation binaire $\mathcal{R}$ (définie sur $E \times E$) est dite :

- *réflexive* si pour tout x de E, $x\mathcal{R}x$.
- *symétrique* si pour tout x, y de E, si $x\mathcal{R}y$ alors $y\mathcal{R}x$.
- *anti-symétrique* si pour tout x, y de E, si $x\mathcal{R}y$ et $y\mathcal{R}x$ alors $x = y$.
- *transitive* si pour tout x, y, z de E, si $x\mathcal{R}y$ et $y\mathcal{R}z$ alors $x\mathcal{R}z$.

Une relation réflexive et transitive sur $E \times E$ est appelée *pré-ordre*. Un *ordre partiel* $\sqsubseteq$ est une relation réflexive, anti-symétrique et transitive sur $E \times E$. On dit alors que E est *partiellement ordonné* par $\sqsubseteq$, ce qui est noté $(E, \sqsubseteq)$. Si de plus, cette relation vérifie $\forall x, y \in E$, on a soit $x \sqsubseteq y$, soit $y \sqsubseteq x$, cette relation d'ordre est dite *totale* et que E est *totalement ordonné* par $\sqsubseteq$.

Soit $(E, \sqsubseteq)$, un ensemble partiellement ordonné. Un élément x de E est dit *minimal* (respectivement *maximal*) si pour tout élément y de E tel que $y \sqsubseteq x$ (resp. $x \sqsubseteq y$) alors $x = y$; x est *le plus petit* (resp. *le plus grand*) élément de E, si pour tout y de E, $x \sqsubseteq y$ (resp. pour tout y de E, $y \sqsubseteq x$).

Soit S, un sous-ensemble de E. Une *borne supérieure* (resp. *borne inférieure*) de S (dans E) est un élément M (resp. m) de E tel que pour tout élément s de S, on a $s \sqsubseteq M$ (resp. $m \sqsubseteq s$). L'ensemble S admet une plus petite borne supérieure (resp. une plus grande borne inférieure), notée $\sqcup S$ (resp. $\sqcap S$), si pour toute borne supérieure M (resp. borne inférieure m) de S, $\sqcup S \sqsubseteq M$ (resp. $m \sqsubseteq \sqcap S$).

1.1.2 Treillis et points fixes

Soit $(E, \sqsubseteq)$, un ensemble partiellement ordonné. Si toute paire d'éléments de E admet pour $\sqsubseteq$ une plus petite borne supérieure (resp. une plus grande borne inférieure), alors on dit que $(E, \sqsubseteq, \sqcup)$ est un *semi-treillis supérieur* (resp. $(E, \sqsubseteq, \sqcap)$ est un *semi-treillis inférieur*). $(E, \sqsubseteq, \sqcup, \sqcap)$

est un *treillis* si $(E, \sqsubseteq, \sqcup)$ est un semi-treillis supérieur et $(E, \sqsubseteq, \sqcap)$ est un semi-treillis inférieur. De plus, si pour tout sous-ensemble S de E, S admet une plus petite borne supérieure (resp. une plus grande borne inférieure), alors on dit que $(E, \sqsubseteq, \sqcup, \top, \bot)$ est un *semi-treillis supérieur complet* (resp. $(E, \sqsubseteq, \sqcap, \top, \bot)$ est un *semi-treillis inférieur complet*), où $\top = \sqcup E = \sqcap \varnothing$ et $\bot = \sqcap E = \sqcup \varnothing$. $(E, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ est un *treillis complet* si $(E, \sqsubseteq, \sqcup, \top, \bot)$ est un semi-treillis supérieur complet et $(E, \sqsubseteq, \sqcap, \top, \bot)$ est un semi-treillis inférieur complet.

Soit $(S, \sqsubseteq)$ un ensemble partiellement ordonné et Φ une fonction totale de S dans S. On dit que Φ est *monotone* si $x \sqsubseteq y$ implique $\Phi(x) \sqsubseteq \Phi(y)$.

On dit que x est un *pré-point fixe* (resp. un *post-point fixe*) de Φ si $x \sqsubseteq \Phi(x)$ (resp. $\Phi(x) \sqsubseteq x$). Un point fixe de Φ est un élément x de S qui est à la fois post- et pré-point fixe de Φ, *i.e.* tel que $\Phi(x) = x$. On note PF_Φ, l'ensemble des point fixes de Φ.

On peut alors énoncer le théorème de Tarski :

Théorème 1 [70] Soit $(E, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$, un treillis complet et Φ, une fonction totale monotone de E dans E, l'ensemble des points fixes de Φ sur ce treillis forme un treillis $(PF_\Phi, \sqsubseteq, \sqcup, \sqcap, pppf_\Phi, pgpf_\Phi)$.

Les éléments extrema de ce treillis $pppf_\Phi$ et $pgpf_\Phi$ sont appelés respectivement *plus petit point fixe* et *plus grand point fixe* de Φ. De plus, si on définit les fonctions itérées $\Phi \uparrow$ et $\Phi \downarrow$ comme suit

$$\begin{cases} \Phi \uparrow 0 = \bot \\ \Phi \uparrow \alpha = \Phi(\Phi \uparrow (\alpha - 1)) & \textit{si } \alpha \textit{ est un ordinal successeur} \\ \Phi \uparrow \alpha = \sqcup_{\alpha' < \alpha} \Phi \uparrow \alpha' & \textit{si } \alpha \textit{ est un ordinal limite} \end{cases}$$

$$\begin{cases} \Phi \downarrow 0 = \top \\ \Phi \downarrow \alpha = \Phi(\Phi \downarrow (\alpha - 1)) & \textit{si } \alpha \textit{ est un ordinal successeur} \\ \Phi \downarrow \alpha = \sqcap_{\alpha' < \alpha} \Phi \downarrow \alpha' & \textit{si } \alpha \textit{ est un ordinal limite} \end{cases}$$

alors il existe deux plus petits ordinaux μ et ν tels que $pppf_\Phi = \Phi \uparrow \mu$ et $pgpf_\Phi = \Phi \downarrow \nu$.

Une fonction totale Φ définie sur un treillis $(E, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ est dite *continue* si pour toute suite croissante (au sens de $\sqsubseteq$) $(e_n)_{n \geq 0}$ d'éléments de E, on a

$$\Phi(\bigsqcup_{n \geq 0} e_n) = \bigsqcup_{n \geq 0} \Phi(e_n)$$

Théorème 2 Si Φ est monotone et continue sur le treillis $(E, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$, alors pour ω, le plus petit ordinal limite transfini,

$$pppf_\Phi = \Phi \uparrow \omega$$

La condition de continuité ne suffit cependant pas pour assurer que le plus grand point fixe de Φ soit atteint pour $\Phi \downarrow \omega$.

1.2 Eléments de logique

1.2.1 Termes et algèbres

Signature et termes

Une *signature fonctionnelle* (ou simplement, *signature*) est un couple (Σ, ρ), où Σ est une ensemble fini de symboles de fonctions et ρ est une application, nommée *arité*, de Σ dans $\mathbb{N}$, l'ensemble des entiers naturels. Les symboles d'arité 0 sont les *constantes* et ceux d'arité $1, 2, \ldots, n$

sont dits respectivement *unaire*, *binaire*,..., *n-aire*. La plus grande valeur prise par ρ sur Σ, l'arité maximale, est notée *amax*. On note Σ_i, l'ensemble des symboles de fonctions de Σ d'arité i. Ainsi, $\Sigma = \bigcup_{i=0}^{amax} \Sigma_i$.

Dans la suite de ce document, nous ne distinguerons plus la signature fonctionnelle de l'ensemble Σ, en supposant que ρ désignera toujours l'application arité correspondant à Σ.

Soit $\mathcal{X}$, un ensemble dénombrable de *variables*, dites *du premier ordre*. L'ensemble des *termes* construits sur la signature Σ et sur les variables de $\mathcal{X}$, noté $TERM(\Sigma, \mathcal{X})$, est défini comme le plus petit ensemble vérifiant que :

- les variables sont des termes, *i.e.* $\mathcal{X} \subseteq TERM(\Sigma, \mathcal{X})$,
- les constantes de Σ sont des termes, *i.e.* $\Sigma_0 \subseteq TERM(\Sigma, \mathcal{X})$,
- Si f est un symbole d'arité m et $t_1, \ldots, t_m$ sont des termes, *i.e.* $t_1, \ldots, t_m \in TERM(\Sigma, \mathcal{X})$, alors $f(t_1, \ldots, t_m) \in TERM(\Sigma, \mathcal{X})$.

Pour un terme t, $Var_{\mathcal{X}}(t)$ désigne l'ensemble des variables apparaissant dans ce terme. Un terme ne comportant pas de variable est dit *clos*. $TERM(\Sigma)$ dénote l'ensemble des termes clos sur Σ.

Algèbres

Une Σ-*algèbre* A est définie par un ensemble D_{A} (appellé *support* de l'algèbre A) et pour chaque symbole de fonction f de Σ d'arité m, par une fonction totale f^{A} de $(D_{\mathsf{A}})^m$ dans D_{A}. En particulier, à chaque constante de Σ est associé un élement de D_{A}. On désigne alors cette Σ-algèbre A par $\langle D_{\mathsf{A}}, \{f^{\mathsf{A}}\}_{f \in \Sigma} \rangle$. Par convention, si aucune autre précision n'est apportée, pour une Σ-algèbre A, D_{A} dénote le support de celle-ci et pour tout symbole de fonction f de Σ, la fonction associée dans A est f^{A}.

Une Σ-algèbre $\mathsf{A}' = \langle D_{\mathsf{A}'}, \{f^{\mathsf{A}'}\}_{f \in \Sigma} \rangle$ est une *sous-algèbre* d'une Σ-algèbre $\mathsf{A} = \langle D_{\mathsf{A}}, \{f^{\mathsf{A}}\}_{f \in \Sigma} \rangle$ si $D_{\mathsf{A}'} \subseteq D_{\mathsf{A}}$ et pour tout symbole f de Σ, d'arité m, $f^{\mathsf{A}'}(d_1^{A'}, \ldots, d_m^{A'}) = f^{\mathsf{A}}(d_1^{A}, \ldots, d_m^{A})$ pour tout $d_1^{A'}, \ldots, d_m^{A'} \in D_{\mathsf{A}'}$.

Soient A et B, deux Σ-algèbres. Un Σ-morphisme Λ est une application de D_{A} dans D_{B} telle que pour tout f^A et tout $d_1^{\mathsf{A}}, \ldots, d_m^{\mathsf{A}}$ de D_{A}, $\Lambda(f^A(d_1^{\mathsf{A}}, \ldots, d_m^{\mathsf{A}})) = f^B(\Lambda(d_1^{\mathsf{A}}), \ldots, \Lambda(d_m^{\mathsf{A}}))$.

Parmi l'ensemble des Σ-algèbres, nous en distinguons deux particulières : la Σ-algèbre (ou *Univers*) *de Herbrand* , notée T_Σ et la Σ-algèbre *termale*, notée $\mathsf{T}_{\Sigma,\mathcal{X}}$. Ces algèbres T_Σ et $\mathsf{T}_{\Sigma,\mathcal{X}}$ auront comme support respectivement $TERM(\Sigma)$ et $TERM(\Sigma, \mathcal{X})$. Dans ces deux algèbres, la fonction associée à un symbole de fonction f est canonique, dans le sens où pour $t_1, \ldots, t_m$ appartenant à $TERM(\Sigma, \mathcal{X})$ (respectivement à $TERM(\Sigma)$), $f^{\mathsf{T}_{\Sigma,\mathcal{X}}}(t_1, \ldots, t_m)$ (resp. $f^{\mathsf{T}_\Sigma}(t_1, \ldots, t_m)$) est égal à $f(t_1, \ldots, t_m)$. De ce fait, on peut remarquer que T_Σ est une sous-algèbre de $\mathsf{T}_{\Sigma,\mathcal{X}}$.

Une *valuation* définie sur une algèbre A est une application d'un ensemble de variables de premier ordre $V_{\mathcal{X}}$ ($V_{\mathcal{X}} \subseteq \mathcal{X}$) vers D_{A}, le support de A. Lorsque l'ensemble $V_{\mathcal{X}}$ ($V_{\mathcal{X}} = \{x_1, \ldots, x_l\}$) est fini, nous nous autorisons à écrire une valuation sous une forme extensionnelle $[x_1 \mapsto d_1^{\mathsf{A}}, \ldots, x_l \mapsto d_l^{\mathsf{A}}]$.

Soit t, un terme de $TERM(\Sigma, \mathcal{X})$. L'*interprétation* d'un terme t dans une algèbre A sous une valuation ν définie pour $Var_{\mathcal{X}}(t)$, est notée $\lfloor t \rfloor_\nu^{\mathsf{A}}$ et est définie inductivement comme suit :

- $\lfloor x \rfloor_\nu^{\mathsf{A}} = \nu(x)$ pour $x \in \mathcal{X}$,
- $\lfloor a \rfloor_\nu^{\mathsf{A}} = a^{\mathsf{A}}$ si a est une constante de Σ,
- $\lfloor f(t_1, \ldots, t_m) \rfloor_\nu^{\mathsf{A}} = f^{\mathsf{A}}(\lfloor t_1 \rfloor_\nu^{\mathsf{A}}, \ldots, \lfloor t_m \rfloor_\nu^{\mathsf{A}})$.

1.2.2 Formules et structures

Formules

Une *signature du premier ordre* est définie comme un couple $(\Sigma, \mathcal{P})$, où Σ est une signature fonctionnelle et $\mathcal{P}$ est un ensemble fini de symboles de *prédicats*, muni d'une application $\rho_{\mathcal{P}}$ de $\mathcal{P}$ dans $\mathbb{N}$, désignant l'arité des symboles de prédicats.

Un *atome* (ou *formule atomique*) est un objet syntaxique de la forme $p(t_1, \ldots, t_m)$, où $t_1, \ldots, t_m$ sont des termes et p est un symbole de prédicat d'arité m, *i.e.* $p \in \mathcal{P}$ et $\rho_{\mathcal{P}}(p) = m$. Si les termes $t_1, \ldots, t_m$ sont des termes clos, on dit alors que l'atome $p(t_1, \ldots, t_m)$ est *clos*. On note $\mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})}$ (resp. $\mathbb{A}_{(\Sigma,\mathcal{P})}$), l'ensemble des atomes (resp. des atomes clos) définis sur une signature du premier ordre $(\Sigma, \mathcal{P})$ et un ensemble de variables $\mathcal{X}$.

L'ensemble des *formules du premier ordre*, construites sur un ensemble de symboles de fonction Σ, un ensemble de symboles de prédicats $\mathcal{P}$ et un ensemble de variables de premier ordre $\mathcal{X}$ est noté $\mathbb{L}_{(\Sigma,\mathcal{P},\mathcal{X})}$ et est défini comme le plus petit ensemble vérifiant :

- les atomes sont des formules, *i.e.* $\mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})} \subseteq \mathbb{L}_{(\Sigma,\mathcal{P},\mathcal{X})}$.
- si φ est une formule, alors $\neg\varphi$ (négation) est une formule.
- si φ et φ' sont deux formules, alors $\varphi \wedge \varphi'$ (conjonction), $\varphi \vee \varphi'$ (disjonction), $\varphi \Rightarrow \varphi'$ (implication) et $\varphi \Leftrightarrow \varphi'$ (équivalence) sont des formules.
- si φ est une formule et x une variable du premier ordre, alors $\exists x\varphi$ (quantification existentielle) et $\forall x\varphi$ (quantification universelle) sont des formules.

Pour une formule de la forme $\exists x\varphi$ ou $\forall x\varphi$, la formule φ est appelé *portée* de la quantification de la variable x. Une variable y n'apparaissant pas dans la quantification d'une formule ou hors de la portée d'un quantificateur est dite *libre*. $Var_{\mathcal{X}}(\varphi)$ désigne l'ensemble des variables libres (du premier ordre) de la formule φ. Une formule ne comportant pas de variable libre est dite *close*.

Structures

La sémantique d'une formule de $\mathbb{L}_{(\Sigma,\mathcal{P},\mathcal{X})}$ est liée à la notion de structure, étendant la notion d'algèbre en prenant en compte les symboles de prédicats.

Une $(\Sigma, \mathcal{P})$-*structure* R est un couple $\langle\langle \mathsf{A}, \{p^{\mathsf{R}}\}_{p\in\mathcal{P}} \rangle\rangle$ tel que A est une Σ-algèbre et pour chaque symbole p de $\mathcal{P}$, p^{R} est une relation sur $(D_{\mathsf{A}})^{\rho_{\mathcal{P}}(p)}$, où D_{A} est le support de l'algèbre A.

On appelle *valeur de vérité* les éléments de $\{\mathsf{vrai}, \mathsf{faux}\}$, ensemble noté $\mathbb{B}$. L'*interprétation d'une formule* de $\mathbb{L}_{(\Sigma,\mathcal{P},\mathcal{X})}$dans une $(\Sigma, \mathcal{P})$-structure $\mathsf{R}=\langle\langle \mathsf{A}, \{p^{\mathsf{R}}\}_{p\in\mathcal{P}} \rangle\rangle$ sous une valuation ν définie de $Var_{\mathcal{X}}(\varphi)$ dans D_A est une valeur de vérité (notée $\lfloor\varphi\rfloor^{\mathsf{R}}_{\nu}$) affectée comme suit : $\lfloor\varphi\rfloor^{\mathsf{R}}_{\nu}= \mathsf{vrai}$ si et seulement si

- φ est un atome $p(t_1, \ldots, t_m)$ et $(\lfloor t_1\rfloor^{\mathsf{A}}_{\nu}, \ldots, \lfloor t_m\rfloor^{\mathsf{A}}_{\nu}) \in p^{\mathsf{R}}$ [1].
- φ est une négation $\neg\varphi'$ et $\lfloor\varphi'\rfloor^{\mathsf{R}}_{\nu} = \mathsf{faux}$
- φ est une conjonction $\varphi_1 \wedge \varphi_2$ et $\lfloor\varphi_1\rfloor^{\mathsf{R}}_{\nu}=\lfloor\varphi_2\rfloor^{\mathsf{R}}_{\nu} = \mathsf{vrai}$.
- φ est une disjonction $\varphi_1 \vee \varphi_2$ et $\lfloor\varphi_1\rfloor^{\mathsf{R}}_{\nu}= \mathsf{vrai}$ ou $\lfloor\varphi_2\rfloor^{\mathsf{R}}_{\nu}= \mathsf{vrai}$.
- φ est une implication $\varphi_1 \Rightarrow \varphi_2$ et $\lfloor\varphi_1\rfloor^{\mathsf{R}}_{\nu}= \mathsf{faux}$ ou $\lfloor\varphi_2\rfloor^{\mathsf{R}}_{\nu}= \mathsf{vrai}$.

[1] Par abus, nous noterons parfois $\lfloor t\rfloor^{\mathsf{R}}_{\nu}$ au lieu de $\lfloor t\rfloor^{\mathsf{A}}_{\nu}$ en considérant la structure R définie sur l'algèbre A. De même, nous parlerons du domaine d'une structure pour désigner, en fait, le domaine de l'algèbre sur laquelle cette structure est définie.

- φ est une équivalence $\varphi_1 \Leftrightarrow \varphi_2$ et $\lfloor\varphi_1\rfloor^{\mathsf{R}}_{\nu}=\lfloor\varphi_2\rfloor^{\mathsf{R}}_{\nu}$. [2]
- φ est une quantification existentielle $\exists x\varphi'$ et il existe une valuation ν' telle pour toute variable y différente de x, $\nu'(y) = \nu(y)$ et $\lfloor\varphi'\rfloor^{\mathsf{R}}_{\nu'} = \mathsf{vrai}$
- φ est une quantification universelle $\forall x\varphi'$ et pour toute valuation ν' telle pour toute variable y différente de x, $\nu'(y) = \nu(y)$ et $\lfloor\varphi'\rfloor^{\mathsf{R}}_{\nu'}= \mathsf{vrai}$.

Pour une formule φ de $\mathbb{L}_{(\Sigma,\mathcal{P},\mathcal{X})}$, nous écrirons $(\mathsf{R},\nu) \models \varphi$ si $\lfloor\varphi\rfloor^{\mathsf{R}}_{\nu} = \mathsf{vrai}$. On dit qu'une $(\Sigma,\mathcal{P})$-structure R est modèle de φ (notée $\mathsf{R} \models \varphi$) si et seulement si pour toute valuation ν, $(\mathsf{R},\nu) \models \varphi$. On dit qu'une formule φ est satisfiable s'il existe une structure R et une valuation ν telles que $(\mathsf{R},\nu) \models \varphi$. On dit qu'une formule est valide si pour toute structure R, R est un modèle de φ. Une formule est dite *inconsistante* si elle n'a pas de modèle.

On dit qu'une formule du premier ordre φ est *conséquence logique* d'une formule ψ (qu'on note $\psi \models \varphi$) si tout modèle R de ψ est un modèle de φ. Si $\psi \models \varphi$ et $\varphi \models \psi$, alors φ et ψ seront dites équivalentes et on notera $\varphi \equiv \psi$.

Une formule φ est dite conséquence logique d'une formule ψ *dans une algèbre* A si pour toute structure R construite au dessus de cette algèbre A, R est un modèle de ψ implique que R est un modèle de φ. On notera alors $\psi \models_{\mathsf{A}} \varphi$. On peut alors dire que deux formules sont équivalentes dans une algèbre A (noté $\varphi \equiv_{\mathsf{A}} \psi$) si $\psi \models_{\mathsf{A}} \varphi$ et $\varphi \models_{\mathsf{A}} \psi$

Conventions

On écrira $\varphi(x_1,\ldots,x_m)$ afin de souligner le fait que les variables $x_1,\ldots,x_m$ sont les variables libres de φ. $\exists\varphi$ et $\forall\varphi$ sont appelés respectivement *clôture existentielle* et *clôture universelle* de la formule φ. Si $Var_{\mathcal{X}}(\varphi)= \{x_1,\ldots,x_m\}$, alors ces notations désignent $\exists x_1 \ldots \exists x_m\varphi$ et $\forall x_1 \ldots \forall x_m\varphi$. On notera $\exists_{-x}$ (resp. $\forall_{-x}$) la formule où toutes les variables libres de φ à l'exception de la variable x sont quantifiées existentiellement (resp. universellement).

Nous dirons qu'une formule du premier ordre φ est positive si elle ne comporte pas pas de symboles de négation ($\neg$), ni d'implication ($\Rightarrow$) et ni d'équivalence ($\Leftrightarrow$).

Une formule φ est dite en forme *prénexe* si elle s'écrit comme $Q\varphi'$, où φ' est une formule ne comportant pas de quantificateurs et Q est une suite de quantifications $\exists x$, $\forall x$. Pour toute formule φ, il existe (au moins) une formule φ_p en forme prénexe qui lui est équivalente. On peut alors distinguer les formules selon la forme de la quantification de leur(s) forme(s) prénexe(s). Pour une forme donnée, l'ensemble des formules qui mis en forme prénexe ont cette forme de quantification sont appelées *fragments*. Par exemple, la formule $\varphi = \forall x\,(f(x,x) \vee (\forall y \forall z\, g(x,z) \wedge g(y,z)) \Rightarrow g(x,x)$ a pour prénexe $\forall x\, \exists y \exists z (f(x,x) \vee (g(x,z) \wedge g(y,z)) \Rightarrow g(x,x)$, on dit alors que φ appartient au fragment $\forall^*\exists^*$ (ou, pour être encore plus précis $\forall\exists^2$) de la logique du premier ordre.

Nous utiliserons dans ce document, une autre notation, équivalente à celle présentée précédemment, en ce qui concerne les interprétations des atomes. Pour une algèbre A et pour un symbole de prédicat p d'arité m, un A-atome est un objet de la forme $p(d_1^{\mathsf{A}},\ldots,d_m^{\mathsf{A}})$, où $d_1^{\mathsf{A}},\ldots,d_m^{\mathsf{A}}$ sont des éléments du support de A. On peut donc assimiler l'interprétation d'un symbole de prédicat p dans une structure R (définie à partir d'une algèbre A) à un ensemble de A-atomes labellés par p. Ainsi, pour une algèbre A fixée, nous confondrons une structure (définie sur cette algèbre) et un ensemble de A-atomes.

En particulier, pour l'algèbre de Herbrand et l'algèbre termale, l'ensemble des A-atomes est respectivement l'ensemble des atomes et l'ensemble des atomes clos (ou *base de Herbrand*).

[2] On peut remarquer que pour toutes formules φ_1 et φ_2, $\varphi_1 \Leftrightarrow \varphi_2$ et $(\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$ ont les mêmes valeurs de vérité pour toute structure et toute valuation.

Formules monadiques

On peut noter que dans le cas où un symbole de prédicat est unaire (ou *monadique*) alors l'interprétation d'un atome dans une structure définie à l'aide d'une algèbre A est un sous-ensemble de D_{A}. Dans le cas où l'ensemble $\mathcal{P}$ ne contient que des symboles de prédicats monadiques, nous nous autoriserons à écrire un atome sous la forme $t \in X$, où t est un terme et X, une *variable du second ordre* assimilable à un symbole de prédicat monadique. Dans ce cas, une formule écrite à l'aide de tels atomes sera dite *monadique* . Si $\mathcal{V}$ est un ensemble de variables de second ordre, alors $\mathbb{L}_{(\Sigma,\mathcal{V},\mathcal{X})}$ désignera l'ensemble des formules monadiques écrites à l'aide d'une signature (fonctionnelle) Σ, de $\mathcal{V}$ et d'un ensemble de variable du premier ordre $\mathcal{X}$.

La logique de premier ordre monadique ne permet pas de quantifier ces variables de second ordre. On peut simplement noter que le problème de satisfiabilité d'une formule revient à rechercher un modèle et donc à les quantifier implicitement existentiellement, tandis que le problème de validité revient à tester la validité dans tous les modèles et donc à les quantifier implicitement universellement. La quantification explicite de telles variables n'est possible qu'en logique (monadique) du second ordre.

Une formule de la *logique monadique du second ordre* est une formule définie par la grammaire suivante :

$$\phi ::= t \in X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \phi \Rightarrow \phi \mid \exists x\phi \mid \forall x\phi \mid \forall X\phi \mid \exists X\phi$$

où x et X sont respectivement des variables du premier et du second ordre et t est un terme (du premier ordre). $\mathbb{L}^2_{(\Sigma,\mathcal{V},\mathcal{X})}$ désignera l'ensemble des formules monadiques du second ordre construite sur une signature Σ, un ensemble de variables du premier ordre $\mathcal{X}$ et du second ordre $\mathcal{V}$.

La sémantique de telles formules dans une algèbre A est définie comme pour les formules monadiques du premier ordre, en sachant qu'aux variables du second ordre sont associés des sous-ensembles de D_{A} et que le symbole $\in$ est interprété comme l'appartenance. Ainsi, $\forall X\phi$ signifie que pour toute assignation d'un sous-ensemble de D_A à X, la valeur de vérité de ϕ doit être vrai.

Contraintes

Lorsqu'on s'intéresse à des formules du premier ordre interprétées sur une structure fixée, on parle alors de contraintes. Soient $(\Sigma, \mathcal{P})$, une signature du premier ordre et $\mathcal{X}$, un ensemble de variables du premier ordre. Les définitions suivantes sont extraites de [41].

Définition 1 Un *domaine de contraintes* est la donnée d'un couple $(\mathcal{L}, \mathsf{R})$, où R est le domaine sémantique, c'est-à-dire une $(\Sigma, \mathcal{P})$-structure, et $\mathcal{L}$ est le domaine syntaxique, c'est-à-dire est un ensemble de formules construites sur Σ, $\mathcal{P}$ et $\mathcal{X}$ ($\mathcal{L} \subseteq \mathbb{L}_{(\Sigma,\mathcal{P},\mathcal{X})}$) appelées *contraintes* et vérifiant :

- $\mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})} \subseteq \mathcal{L}$, les éléments de $\mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})}$ étant appelés *contraintes basiques*,
- $\mathcal{L}$ est clos par renommage de variables, quantification existentielle et conjonction.

Les notions introduites concernant la logique et définies précédemment restent valides sur le plan des contraintes, en tenant compte du fait que la structure d'interprétation y est fixée.

Soit φ une contrainte. Une valuation ν (définie de $Var_{\mathcal{X}}(\varphi)$ vers le domaine de R[3] telle que $(\mathsf{R}, \nu) \models \varphi$ est appelée *solution* de la contrainte φ. Une contrainte φ est dite *satisfiable* si elle possède une solution. Une contrainte ψ *implique* une contrainte φ (noté $\psi \models_{\mathsf{R}} \varphi$) si toute solution de ψ est une solution de φ. Deux contraintes ψ et φ sont équivalentes si elles possèdent les mêmes solutions, *i.e.* $\psi \models_{\mathsf{R}} \varphi$ et $\varphi \models_{\mathsf{R}} \psi$.

[3]Pour être plus précis, vers le domaine de l'algèbre sur laquelle est définie la structure R.

1.3 Arbres et automates

1.3.1 Arbres

Nous rappelons que Σ est une signature munie d'une fonction d'arité ρ, que *amax* est l'arité maximale des symboles de fonction de Σ et que Σ_i désigne l'ensemble des symboles de fonctions d'arité i de Σ.

Définitions

Soit $\mathbb{N}_+$, l'ensemble des entiers strictement positifs. Soit $(\mathbb{N}_+)^*$, le monoïde libre engendré sur $\mathbb{N}_+$ par concaténation $\cdot$ et par le mot vide ϵ. Les éléments de $(\mathbb{N}_+)^*$ sont appelés *mot* de $(\mathbb{N}_+)^*$.

Soient u, v deux mots de $(\mathbb{N}_+)^*$. On dit que v est un *préfixe* de u si il existe un mot w de $(\mathbb{N}_+)^*$ tel que $u = v \cdot w$. On note cette relation « est préfixe de» par $\trianglelefteq$. Il est immédiat de voir que $\trianglelefteq$ est une relation d'ordre (partielle) sur $(\mathbb{N}_+)^*$. On note $\triangleleft$, la relation de préfixe stricte sur $(\mathbb{N}_+)^*$, *i.e.* $u \triangleleft v$ si $u \trianglelefteq v$ et $u \neq v$.

Un sous-ensemble M de $(\mathbb{N}_+)^*$ est dit :

- *clos par préfixe* si pour tout élément u de M, si $v \trianglelefteq u =$, alors $v \in M$.
- *clos par aînesse* si $u \cdot i \in D$ (avec $i \in \mathbb{N}_+$), alors pour tout $j \leq i$, $u \cdot j \in D$.

On appelle *domaine d'arbre* tout sous-ensemble non vide de $(\mathbb{N}_+)^*$ clos par préfixe et par aînesse. Un L-arbre τ (ou arbre étiqueté sur L) est une application d'un domaine d'arbre (noté $dom(\tau)$) dans L. Un L-arbre τ tel que $dom(\tau)$ est un ensemble fini sera dit *fini*. L'ensemble des arbres (respectivement des arbres finis) étiquetés par un ensemble L est noté $\mathbb{TR}^{\omega}_L$ (respectivement $\mathbb{TR}_L$).

Pour un L-arbre τ, les éléments de $dom(\tau)$ sont appelés *positions* (ou *nœuds*). La position ϵ est appelée *racine* de l'arbre. Une position d telle que $d \cdot 1 \notin dom(\tau)$ est une *feuille*. Pour un L-arbre τ, le *sous-arbre* de τ à la position d (avec $d \in dom(\tau)$) est un L-arbre, noté $\tau_{|d}$ et défini par $dom(\tau_{|d}) = \{u \mid d \cdot u \in dom(\tau)\}$ et pour tout $u \in dom(\tau_{|d})$, on a $\tau_{|d}(u) = \tau(d \cdot u)$. Pour une position d d'un arbre τ et pour $i \in \mathbb{N}_+$ tel que $d \cdot i \in dom(\tau)$, on dit que l'arbre $\tau_{|d \cdot i}$ est un *fils* du nœud d.

Soient τ,τ', deux L-arbres. La *greffe* de l'arbre τ' dans l'arbre τ à la position d ($d \in dom(\tau)$) est un L-arbre, noté $\tau[d \leftarrow \tau']$ et défini comme suit :

$$dom(\tau[d \leftarrow \tau']) = \{d \cdot u \mid u \in dom(\tau')\} \cup \{d' \mid d' \in dom(\tau) \text{ et } d \not\trianglelefteq d'\}$$

$$\tau[d \leftarrow \tau'](d') = \begin{cases} \tau'(u) & \text{si } d' \in \{d \cdot u \mid u \in dom(\tau')\} \\ \tau(d') & \text{si } d' \in \{d' \mid d' \in dom(\tau) \text{ et } d \not\trianglelefteq d'\} \end{cases}$$

La greffe d'un arbre τ' dans τ à la position d n'est que le remplacement du sous-arbre $\tau_{|d}$ dans τ par l'arbre τ'.

Un chemin π dans un L-arbre τ est un sous-ensemble de $dom(\tau)$ totalement ordonné par $\trianglelefteq$ et clos par préfixe. Un chemin sera dit infini si π est un ensemble infini, il est dit fini dans le cas contraire. Pour un L-arbre τ, $\Pi^{\omega}(\tau), \Pi(\tau), \Pi^{\infty}(\tau)$ désignent respectivement l'ensemble des chemins, des chemins finis et des chemins infinis dans τ. On note $\tau|\pi$, la restriction de (l'application) τ au chemin π.

Soit τ, un L-arbre. On définit $Inf(\tau)$ (resp. $Inf(\tau|\pi)$) l'ensemble des étiquettes rencontrées infiniment souvent dans l'arbre τ (resp. infiniment souvent le long du chemin π de l'arbre τ), *i.e.* :

$$Inf(\tau) = \{l \in L \mid \exists^{\infty} d \in dom(\tau), \tau(d) = l\} \quad Inf(\tau|\pi) = \{l \in L \mid \exists^{\infty} d \in \pi, \tau(d) = l\}$$

Nous appellerons Σ-arbre, un arbre τ étiqueté sur Σ respectant les arités des fonctions de Σ, *i.e.* dont le domaine vérifie : pour tout $d \in dom(\tau)$, $\rho(\tau(d)) = n$ et $n > 0$ ssi $d \cdot n \in dom(\tau)$ et $d \cdot (n+1) \notin dom(\tau)$ [4]. On peut alors remarquer que la définition d'un Σ-arbre implique que si d est une feuille, alors cette position est étiquetée par une constante.

Comme précédemment, $\mathbb{TR}^{\omega}_{\Sigma}$ et $\mathbb{TR}_{\Sigma}$ désignent respectivement l'ensemble des Σ-arbres et l'ensemble des Σ-arbres infinis.

Dorénavant, lorsque le mot « arbre » sera utilisé sans plus de précision, il fera référence à « Σ-arbre ».

Algèbres d'arbres

Considérons les ensembles $\mathbb{TR}^{\omega}_{\Sigma}$ et $\mathbb{TR}_{\Sigma}$ et associons à chaque symbole de fonction f de Σ, une application f^{Tr} de $(\mathbb{TR}^{\omega}_{\Sigma})^{\rho(f)}$ dans $\mathbb{TR}^{\omega}_{\Sigma}$ telle que : $f^{\mathsf{Tr}}(\tau_1, \ldots, \tau_{\rho(f)}) = \tau$ avec τ vérifiant $dom(\tau) = \{\epsilon\} \cup \{i \cdot d_i \mid d_i \in dom(\tau_i) \text{ et } 1 \leq i \leq \rho(f)\}$ et $\tau_{|i} = \tau_i$ pour $1 \leq i \leq \rho(f)$. On a alors

Proposition 1 $\mathsf{Tr}^{\omega}_{\Sigma} = \langle \mathbb{TR}^{\omega}_{\Sigma}, \{f^{\mathsf{Tr}}\}_{f \in \Sigma} \rangle$ et $\mathsf{Tr}_{\Sigma} = \langle \mathbb{TR}_{\Sigma}, \{f^{\mathsf{Tr}}\}_{f \in \Sigma} \rangle$ sont deux Σ-algèbres.

$\mathsf{Tr}^{\omega}_{\Sigma}$ est appelée *algèbre des arbres* et Tr_{Σ} *algèbre des arbres finis*. On peut remarquer que Tr_{Σ} est une sous-algèbre $\mathsf{Tr}^{\omega}_{\Sigma}$.

Souvent dans la suite de ce document, par abus de notation, $f(\tau_1, \ldots, \tau_m)$ sera mis pour l'arbre $f^{\mathsf{Tr}}(\tau_1, \ldots, \tau_m)$.

De plus, on peut remarquer qu'il existe deux morphismes surjectifs Λ_t et Λ_τ respectivement de T_{Σ} dans Tr_{Σ} et de Tr_{Σ} dans T_{Σ} vérifiant de surcroît que $\Lambda_t \circ \Lambda_\tau$ est l'identité. Pour cette raison, nous confondrons « terme clos » et « Σ-arbre fini ».

Pour un arbre fini (ou un terme clos) τ, la *hauteur* de celui-ci est notée $|\tau|$ et est définie par :

$$|\tau| = \begin{cases} 0 & \text{si } \tau \text{ est réduit à une feuille} \\ 1 + max\{|\tau_i| \mid 1 \leq i \leq m\} & \text{si } \tau \text{ est de la forme } f(\tau_1, \ldots, \tau_m) \end{cases}$$

1.3.2 Automates d'arbres finis

Dans cette section, les arbres manipulés étant finis, nous emploierons la syntaxe des termes clos pour les décrire en nous accordant toutefois le droit d'utiliser les notions introduites pour les arbres.

Nous ne donnerons ici des automates que les notions et propriétés utiles à notre propos. Le lecteur désireux d'en savoir plus sur les automates d'arbres finis pourra se reporter à [31] et [21].

Il existe, concernant les automates d'arbres finis, deux types d'automates : les *automates ascendants* et les *automates descendants* d'arbres finis. Cette distinction tient à la façon dont l'automate « calcule » à partir d'un terme clos τ : des feuilles vers la racine pour un automate

[4] Cette définition est équivalente à celle de [22].

ascendant et de la racine vers les feuilles pour un automate descendant. Cette distinction est principalement importante lorsqu'on aborde la notion de déterminisme (voir page 18).

Automates ascendants d'arbres finis

Définition 2 Un *automate ascendant d'arbres finis* est défini par un quadruplet $(\Sigma, \mathcal{Q}, F, \Delta)$. Σ est une signature, $\mathcal{Q}$ est un ensemble fini d'états, F (avec $F \subseteq \mathcal{Q}$) est un ensemble d'états dit *finals* et Δ est une relation sur $\bigcup_{i=0}^{amax} \Sigma_i \times \mathcal{Q}^i \times \mathcal{Q}$, dont les éléments, appelés *règles de transition*, sont notés :

$$f(q_1, \ldots, q_m) \to q \quad \text{avec } q, q_1, \ldots, q_m \in \mathcal{Q} \text{ et } f \text{ un symbole d'arité } m$$

Soit $TERM(\Sigma \cup \mathcal{Q})$, l'ensemble des termes construits à l'aide de la signature Σ et de l'ensemble $\mathcal{Q}$, où les états sont vus comme des symboles de constantes (*i.e.* en posant pour tout q de $\mathcal{Q}$, $\rho(q) = 0$). On remarque que $TERM(\Sigma) \subseteq TERM(\Sigma \cup \mathcal{Q})$.

Une transition élémentaire dans un automate $\mathcal{A}$ est une relation sur $TERM(\Sigma \cup \mathcal{Q}) \times TERM(\Sigma \cup \mathcal{Q})$, notée $\to_{\mathcal{A}}$, est définie par $\tau \to_{\mathcal{A}} \tau'$ si il existe une position d dans $dom(\tau)$ tel que $\tau_{|d} = l$, $\tau' = \tau[d \leftarrow r]$, et $l \to r \in \Delta$.

Soit $\to_{\mathcal{A}}^*$, la clôture transitive de $\to_{\mathcal{A}}$. Un arbre τ ($\tau \in \mathbb{T}_{\mathbb{R}_\Sigma}$) est *accepté* par l'automate $\mathcal{A}$ si et seulement si il existe un état final q_f de $\mathcal{A}$ ($q_f \in F$) tel que $\tau \to_{\mathcal{A}}^* q_f$. Le *langage reconnu* par un automate $\mathcal{A}$, est noté $L(\mathcal{A})$ et est défini par $L(\mathcal{A}) = \{\tau \mid \tau \in TERM(\Sigma), q_f \in F \text{ et } \tau \to_{\mathcal{A}}^* q_f\}$. Un ensemble d'arbres (finis) L est dit *reconnaissable* ou *régulier* s'il existe un automate d'arbres finis $\mathcal{A}$ tel que $L = L(\mathcal{A})$.

Un état q est dit *accessible* (dans un automate $\mathcal{A}$) si il existe un arbre τ ($\tau \in \mathbb{T}_{\mathbb{R}_\Sigma}$) tel que $\tau \to_{\mathcal{A}}^* q$. L'ensemble des états accessibles dans un automate $\mathcal{A}$ est noté $Reachable(\mathcal{A})$.

Nous allons énoncer une suite de propriétés liées aux automates ascendants d'arbres. Ces propriétés sont données sans preuve.

Propriétés Soit $\mathcal{A}$, un automate ascendant d'arbres finis et τ, un arbre fini :

- Le test du vide (« $L(\mathcal{A}) = \varnothing$? ») est décidable et est linéaire par rapport à la taille de l'automate, notée $|\mathcal{A}|$ [5].
- Le test d'appartenance (« $\tau \in L(\mathcal{A})$? ») est décidable et est :
  - polynômial dans la taille de l'arbre si $\mathcal{A}$ est non-déterministe.
  - linéaire dans la taille de l'arbre si $\mathcal{A}$ est déterministe.
- Le problème d'accessibilité d'un état dans un automate est décidable et est linéaire par rapport à la taille de l'automate.

On dit qu'un automate ascendant d'arbres finis est *déterministe* si Δ ne contient pas deux règles de transition ayant le même membre gauche. Un automate est dit *complet* si pour tout symbole f de Σ et pour tout $q_1, \ldots, q_{\rho(f)} \in \mathcal{Q}$, il existe une règle $f(q_1, \ldots, q_{\rho(f)}) \to q$ dans Δ.

Dans le cas d'un automate ascendant $\mathcal{A}$ déterministe et complet, on peut remarquer que la restriction de la relation $\to_{\mathcal{A}}^*$ (définie sur $TERM(\Sigma \cup \mathcal{Q}) \times TERM(\Sigma \cup \mathcal{Q})$) à l'ensemble $TERM(\Sigma \cup \mathcal{Q}) \times \mathcal{Q}$ définit, du fait du déterminisme, une fonction de $TERM(\Sigma \cup \mathcal{Q})$ dans $\mathcal{Q}$. De plus, de part la complétude de l'automate, cette fonction est totale sur $TERM(\Sigma \cup \mathcal{Q})$. Dans la

[5] La taille $|\mathcal{A}|$ d'un automate d'arbres finis $\mathcal{A}$ est classiquement définie dans [21] : $|\mathcal{A}| = |\mathcal{Q}| + \Sigma_{f(q_1,\ldots,q_m)\to q \in \Delta}(\rho(f) + 2)$, $|\mathcal{Q}|$ est le cardinal de $\mathcal{Q}$. Nous verrons cependant que pour les automates que nous utiliserons dans les chapitres 3, 4, nous considérerons un majorant de la taille ainsi définie.

suite de ce document, pour un automate ascendant déterministe et complet $\mathcal{A}$, on notera cette fonction $run_{\mathcal{A}}$. Ainsi un arbre τ est reconnu par un tel automate $\mathcal{A}$ si $run_{\mathcal{A}}(\tau) \in F$.

Comme pour les cas des automates de mots, le déterminisme et/ou la complétude ne modifie pas le pouvoir d'expression : si L est un ensemble régulier d'arbres finis, alors il existe un automate ascendant déterministe et complet $\mathcal{A}$ tel que $L(\mathcal{A}) = L$.

Automates descendants d'arbres finis

La définition des automates descendants d'arbres finis est très simple : elle consiste simplement à « inverser » les membres droits et gauches des règles de transition dans la définition d'un automate ascendant.

Définition 3 Un *automate descendant d'arbres finis* est défini par un quadruplet $(\Sigma, \mathcal{Q}, I, \Delta)$. Σ est une signature, $\mathcal{Q}$ est un ensemble fini d'états, I (avec $I \subseteq \mathcal{Q}$) est un ensemble d'états dits *initiaux* et Δ est une relation sur $\mathcal{Q} \times \bigcup_{i=0}^{amax} \Sigma_i \times \mathcal{Q}^i$, dont les éléments, appellés *règles de transition*, sont notés :

$$q \to f(q_1, \ldots, q_m) \quad \text{avec } q, q_1, \ldots, q_m \in \mathcal{Q} \text{ et } f \text{ un symbole d'arité } m$$

Considérons l'ensemble des termes $TERM(\Sigma \cup \Sigma_{\mathcal{Q}})$, construits à l'aide des symboles de fonctions de Σ et d'un ensemble de symboles de fonctions unaires $\Sigma_{\mathcal{Q}}$, tel que pour tout état q de l'automate, il existe un symbole de fonction unaire q dans cet ensemble.

Une transition élémentaire dans un automate descendant $\mathcal{A}$ est définie par une relation de $TERM(\Sigma \cup \Sigma_{\mathcal{Q}})$ dans lui-même (notée $\to_{\mathcal{A}}$), telle que $\tau \to_{\mathcal{A}} \tau'$ s'il existe d une position de τ telle que $\tau_{|d}$ est de la forme $q(f(\tau_1, \ldots, \tau_m))$ (avec $q \in \Sigma_{\mathcal{Q}}$) et une règle $q \to f(q_1, \ldots, q_m)$ dans l'automate $\mathcal{A}$ telle que $\tau' = \tau[d \leftarrow f(q_1(\tau_1), \ldots, q_m(\tau_m))]$.

Soit $\to^*_{\mathcal{A}}$, la cloture transitive de $\to_{\mathcal{A}}$. Un arbre τ est reconnu par un automate descendant d'arbres finis s'il existe un état initial q_i ($q_i \in I$) tel que $q_i(\tau) \to^*_{\mathcal{A}} \tau$. Le langage reconnu par un tel automate est noté $L(\mathcal{A})$ et défini par $L(\mathcal{A}) = \{\tau \in \mathbb{T}_{\mathbb{R}_\Sigma} \mid q_i(\tau) \to^*_{\mathcal{A}} \tau\ ,\ q_i \in I\}$. Les langages reconnaissables par un automate descendant d'arbres finis sont exactement les langages réguliers.

Un automate descendant est dit *déterministe* (resp. *complet*) si pour tout état q et tout symbole de fonction f ($f \in \Sigma_m$), il existe au plus (resp. au moins) un m-uplet $(q_1, \ldots, q_m)$ tel que $q \to f(q_1, \ldots, q_m)$ soit une règle de transition de $\mathcal{A}$.

La principale différence entre les automates ascendants et descendants d'arbres finis tient dans le fait suivant : l'ensemble des langages reconnaissables à l'aide d'un automate descendant déterministe n'est qu'un sous-ensemble strict des langages réguliers. Ainsi, pour le langage $L = \{f(a,b), f(b,a)\}$, il n'existe pas d'automate descendant déterministe $\mathcal{A}$ tel que $L(\mathcal{A}) = L$.

1.3.3 Automates d'arbres infinis

La notion d'automates d'arbres finis est très intuitive. Dans cette section, nous allons introduire la notion d'*automates d'arbres infinis*, qui étend la notion d'automates d'arbres finis.

Les définitions que nous allons donner ne sont pas exactement celles que l'on peut trouver dans la littérature, notamment dans [71]. Les arbres y étant généralement présentés comme des

arbres binaires[6], *i.e.* dont le domaine d'arbres est $\{0,1\}^*$, étiqueté sur un alphabet A quelconque, ceci simplifie d'une certaine façon la présentation des automates s'y rattachant.

Cependant, il nous semble plus simple de conserver explicitement, pour la notion de Σ-arbre, des arbres dont les nœuds peuvent avoir un nombre quelconque de fils (dépendant bien évidemment de l'arité des symboles étiquetant ces nœuds). Ces définitions se retrouvent dans [54].

La définition d'automates d'arbres infinis que nous proposons tout d'abord est des plus générales, ne fixant pas de condition d'acceptance. Cette définition sera ensuite instanciée pour deux conditions particulières, à savoir la condition d'acceptance de Büchi [59] et la condition d'acceptance de Rabin [58].

Définition 4 Un *automate d'arbres infinis* $\mathcal{A}$ est un quintuplet $(\Sigma, Q, I, \Delta, \Omega)$. Σ est une signature, Q est un ensemble fini d'états, I (avec $I \subseteq Q$) est un ensemble d'états dit *initiaux*, Ω est une *condition d'acceptance* et Δ est une relation sur $\bigcup_{i=0}^{amax} \Sigma_i \times Q^i \times Q$, dont les éléments appelés *règles de transition* sont notés :

$$f(q_1, \ldots, q_m) \rightarrow q \quad \textit{avec } q, q_1, \ldots, q_m \in Q \textit{ et } f \textit{ un symbole d'arité } m$$

Un calcul[7] r pour un arbre τ ($\tau \in \mathbb{TR}^{\omega}_{\Sigma}$) dans un automate d'arbres infinis $\mathcal{A}$ est une application de $dom(\tau)$ dans Q, *i.e.* par définition un Q-arbre ($r \in \mathbb{TR}^{\omega}_{Q}$), telle que pour toute position d de $dom(\tau)$, si $\tau(d) = f$ et l'arité de ce symbole f est m, alors

$$f(r(d \cdot 1), \ldots, r(d \cdot m)) \rightarrow r(d) \quad \in \Delta$$

Un calcul r est dit *réussi* si r vérifie la condition Ω.

Le langage reconnu par un automate d'arbres infinis $\mathcal{A}$, noté $L(\mathcal{A})$ est l'ensemble des arbres τ de $\mathbb{TR}^{\omega}_{\Sigma}$ pour lesquels il existe un calcul réussi r dans $\mathcal{A}$ tel que $r(\epsilon) \in I$.

Un état q est dit accessible dans un automate d'arbres infinis $\mathcal{A}$ s'il existe un arbre τ, un calcul réussi r pour τ dans $\mathcal{A}$ et une position d appartenant à $dom(t)$ tel que $r(d) = q$.

Nous allons présenter maintenant deux sous-classes des automates d'arbres infinis en fixant pour chacune d'elles la condition d'acceptance Ω.

Définition 5 Un automate d'arbres infinis est dit *de Büchi* si la condition d'acceptance Ω s'exprime à l'aide d'un ensemble d'états finals F et de la condition suivante :

r est un calcul réussi si pour tout chemin infini π de r, $Inf(r|\pi) \cap F \neq \varnothing$

Cette condition peut se résumer par : un calcul r est réussi sous la condition d'acceptance de Büchi si le long de tout chemin infini de r, l'ensemble des états apparaissant infiniment souvent traverse F.

Définition 6 Un automate d'arbres infinis est dit *de Rabin* si la condition d'acceptance Ω s'exprime à l'aide d'une collection de couples d'ensembles états $\{(L_1, U_1), \ldots, (L_n, U_n)\}$ et de la condition suivante :

r est un calcul réussi si pour tout chemin infini π de r, il existe un indice $i \in \{1, \ldots, n\}$ tel que $Inf(r|\pi) \cap L_i = \varnothing$ et $Inf(r|\pi) \cap U_i \neq \varnothing$

Le test du vide pour un langage reconnu par un automate d'arbres infinis est respectivement (par rapport à la taille de l'automate) [71] :

[6] Cette restriction vient du fait que tout arbre peut être transformé en un arbre binaire.

[7] Cette notion de calcul ne contredit en rien celles définies pour les automates ascendant et descendant d'arbres finis, qui peuvent être vues comme des mécanismes opérationnels définissant ce calcul.

- dans *PTIME* pour un automate de Büchi,
- *NP*-complet pour un automate de Rabin.

Un automate d'arbres infinis est dit *sans condition d'acceptance* si Ω est une tautologie. C'est-à-dire si on considère un automate de Büchi, $F = Q$ et pour un automate de Rabin, on a par exemple, pour tout i, $(L_i, U_i) = (\varnothing, Q)$ (voir [54]).

1.4 Programmation logique

La logique est avant tout un outil de formalisation mathématique, mais elle permet également de définir un paradigme de programmation, appelé *programmation logique*, dont la syntaxe est celle d'un fragment de la logique du premier ordre et dont les sémantiques sont nombreuses et, fort heureusement, équivalentes. Nous allons tout d'abord considérer les aspects syntaxiques de ce paradigme en définissant ce fragment de la logique du premier ordre, puis nous allons considérer trois sémantiques différentes de la programmation logique.

1.4.1 Syntaxe

Supposons donnée $(\Sigma, \mathcal{P})$ une signature du premier ordre et $\mathcal{X}$, un ensemble dénombrable de variables du premier ordre.

On appelle *littéral* un atome A ($A \in \mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})}$) ou la négation d'un atome $\neg A$. Un atome (resp. la négation d'un atome) est appelé *littéral positif* (resp. *négatif*). Une *clause* est une disjonction de littéraux close universellement, *i.e.* de la forme $\forall(A_1 \vee \ldots \vee A_k \vee \neg B_1 \ldots \vee \neg B_l)$. La clause ne comportant aucun littéral est la *clause vide* et est notée $\square$. Sa valeur de vérité est faux.

Une clause est dite *de Horn* si elle comporte au plus un littéral positif ($k \leq 1$), c'est-à-dire si elle est de l'une des trois formes suivantes :

$$\begin{array}{l} \forall A \\ \forall(A \vee \neg B_1 \ldots \vee \neg B_l) \\ \forall(\neg B_1 \ldots \vee \neg B_l) \end{array}$$

Une clause de Horn de la première forme est appelée *clause unitaire* ou *fait*. Une clause de la seconde forme est appelée *règle*. Une *clause définie* est soit une clause unitaire, soit un fait. Pour ce qui est de la dernière forme, une telle clause est appelée *clause négative* ou *but*.

Comme il est de coutume, nous adopterons les conventions de notation suivantes, omettant notamment les quantifications :

$$\begin{array}{lll} A & & \textit{(Fait)} \\ A & \Leftarrow B_1, \ldots, B_l & \textit{(Règle)} \\ & \Leftarrow B_1, \ldots, B_l & \textit{(But)} \end{array}$$

Pour une clause définie $A \Leftarrow B_1, \ldots, B_l$ (ou dans le cas d'un fait, A), A sera appelé la *tête* de la clause et $B_1, \ldots, B_l$ le *corps* de la clause.

Un *programme logique défini* est un ensemble fini de clauses définies. D'un point de vue sémantique, cet ensemble est interprété comme la conjonction des clauses y apparaissant.

Exemple 1 : *Soit P_1 le programme logique*

$$\begin{array}{ll} p(a,a) & p(f(x),y) \Leftarrow p(x,x), q(y) \\ p(b,c) & r(x) \Leftarrow q(x) \\ q(a) & r(x) \Leftarrow p(f(x),y) \end{array}$$

Nous avons précédemment parlé de la notion de valuation définie pour un ensemble de variables sur un domaine d'algèbre. Lorsque le domaine de définition de cette valuation est fini et que l'algèbre considérée est $\mathsf{T}_{\Sigma,\mathcal{X}}$ ou T_Σ, plutôt que de valuation, on parle alors respectivement de *substitution* et de *substitution close*.

Si $\sigma = [x_1 \mapsto t_1, \ldots, x_m \mapsto t_m]$ est une substitution, alors $\sigma(t)$ est le terme égal à $\lfloor t \rfloor_\sigma^{\mathsf{T}_{\Sigma,\mathcal{X}}}$. Plus simplement, chaque occurrence de x_i dans t a été remplacée par le terme $\sigma(x_i)$. $\sigma(t)$ est appelée *instance* de t et *instance close* de t si $\sigma(t)$ un terme clos.

Cette notion de substitution est étendue aux atomes et aux clauses : soit A un atome égal à $p(t_1, \ldots, t_n)$, alors $\sigma(A)$ désigne l'atome (ou de manière équivalente, le $\mathsf{T}_{\Sigma,\mathcal{X}}$-atome) $p(\sigma(t_1), \ldots, \sigma(t_n))$. Soit c une clause de la forme $\forall(A_1 \vee \ldots \vee A_k \vee \neg B_1 \ldots \vee \neg B_l)$ et σ, une substitution, l'instance de la clause c par σ, notée $\sigma(c)$, est égale à $\forall(\sigma(A_1) \vee \ldots \vee \sigma(A_k) \vee \neg\sigma(B_1) \ldots \vee \neg\sigma(B_l))$. Dans le cas où cette formule n'a plus de variable, on dit alors que $\sigma(c)$ est une instance close et on notera $INST_{\mathsf{T}_\Sigma}(P)$, l'ensemble des instances closes des clauses du programme P.

On peut noter que, d'un point de vue logique, on a toujours pour toute clause c et toute substitution σ, $c \models \sigma(c)$.

Lorsqu'on considère l'algèbre de Herbrand ou l'algèbre termale, l'instance d'un atome ou d'une clause reste de par la nature particulière de ces deux algèbres des « objets syntaxiques ». Il n'en va pas de même lorsqu'on considère des algèbres quelconques. Nous avons déjà mentionné le fait que qu'un modèle dans une structure R construit au dessus d'une algèbre A pouvait être vu comme un ensemble de A-atomes de la forme $p(d_1^{\mathsf{A}}, \ldots, d_m^{\mathsf{A}})$ où p est un symbole de prédicat d'arité m.

Ainsi, si on considère un atome $p(t_1, \ldots, t_m)$ de $\mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})}$, une algèbre A et une valuation ν_{A} pour l'algèbre A définie sur les variables de $p(t_1, \ldots, t_m)$, alors le A-atome $p(\lfloor t_1 \rfloor_{\nu_{\mathsf{A}}}^{\mathsf{A}}, \ldots, \lfloor t_m \rfloor_{\nu_{\mathsf{A}}}^{\mathsf{A}})$ est appelé A-instance de $p(t_1, \ldots, t_m)$ (pour la valuation ν_{A}). Ce concept est étendu aux clauses en disant que $H^{\mathsf{A}} \Leftarrow B_1^{\mathsf{A}}, \ldots B_l^{\mathsf{A}}$ est une A-instance d'une clause $H \Leftarrow B_1, \ldots, B_l$ si pour une même valuation ν_{A} (définie sur les variables du premier ordre de la clause), H^{A} est une A-instance de H et pour tout i, B_i^{A} est une A-instance de B_i. On notera $INST_{\mathsf{A}}(P)$, l'ensemble des A-instances des clauses du programme P.

1.4.2 Sémantiques

Nous nous intéresserons dans ce travail principalement à la sémantique de Herbrand d'un programme logique, comme dans [45]. Cependant, suivant l'exemple de [23], nous soulignerons que les résultats s'appliquent à un cadre plus général.

Considérer principalement les sémantiques de Herbrand n'est pas vraiment une restriction du point de vue de la logique si l'on se réfère au théorème de Herbrand, qui établit que :

Théorème 3 Soit S un ensemble de clauses ; S admet un modèle si et seulement si il admet un modèle de Herbrand, soit de manière équivalente : S est inconsistante si et seulement si S n'a pas de modèle de Herbrand.

Comme nous l'avons déjà mentionné lorsque l'algèbre A est fixée, alors une structure (basée sur cette algèbre) peut être vue comme un ensemble de A-atomes. Ainsi pour l'algèbre des termes clos (*i.e.* l'algèbre de Herbrand), une structure sera vue comme un ensemble d'atomes clos I ($I \subseteq \mathbb{A}_{(\Sigma,\mathcal{P})}$).

Sémantique des modèles

Un programme logique étant avant tout une formule logique, il est évident que la logique, et plus précisément la notion de modèles, nous offre le premier cadre pour définir la sémantique d'un tel programme.

Un modèle de Herbrand d'un programme logique défini P est un ensemble d'atomes clos M tel que $M \models P$. Soit $\mathcal{M}_{P,\mathsf{T}_\Sigma}$, l'ensemble des modèles de Herbrand d'un programme P. Cet ensemble est naturellement ordonné par la relation d'inclusion. De plus, l'ensemble des modèles de Herbrand est clos par intersection : soit M_P, un ensemble de modèles de Herbrand d'un programme logique P. On définit $\cap M_P$ comme l'ensemble des atomes clos vérifiant $A \in \cap M_P$ si et seulement si pour tout $M \in M_P$, $A \in M$. On a alors le fait que $\cap M_P$ est un modèle de Herbrand de P. Ainsi, pour tout programme logique défini P, il existe un plus petit modèle au sens de l'inclusion[8]. Ce plus petit modèle est noté $DEN_{\mathsf{T}_\Sigma}(P)$ et est égal à $\cap\mathcal{M}_{P,\mathsf{T}_\Sigma}$. Ceci peut se résumer par le fait que $(\mathcal{M}_{P,\mathsf{T}_\Sigma}, \subseteq, \cap, DEN_{\mathsf{T}_\Sigma}(P))$ est un semi-treillis inférieur complet.

Si on considère le programme P_1 donné dans l'exemple 1, son plus petit modèle de Herbrand est $\{p(a,a),\ p(b,c),\ p(f(a),a),\ q(a),\ r(a)\}$.

Ce plus petit modèle est lié à la notion de conséquence logique pour les atomes clos par :

Théorème 4 [45] Soit $A \in \mathbb{A}_{(\Sigma,\mathcal{P})}$, $A \in DEN_{\mathsf{T}_\Sigma}(P)$ ssi $P \models A$, *i.e.* $DEN_{\mathsf{T}_\Sigma}(P)$ est donc égal à l'ensemble des conséquences logiques atomiques closes du programme P.

Ce résultat s'étend en particulier pour une algèbre quelconque A (en particulier pour l'algèbre termale). Un modèle M_A d'un programme logique P pour une algèbre fixée A est un ensemble de A-atomes, que l'on peut confondre avec une unique structure $\mathsf{R}_{M_\mathsf{A}}$ (définie sur l'algèbre A) tel que pour toute clause[9] c du programme P, on a $\mathsf{R}_{M_\mathsf{A}} \models c$. On en déduit que l'ensemble des A-modèles d'un programme logique est un ensemble « clos par intersection » et donc, qu'il existe un plus petit A-modèle, noté $DEN_\mathsf{A}(P)$.

D'autres sémantiques des programmes logiques peuvent être rencontrées dans la littérature. Nous allons brièvement en donner trois autres ici, à savoir la sémantique des preuves, la sémantique des points fixes et la sémantique opérationnelle.

Sémantique des preuves

Cette seconde sémantique se base sur la notion de preuves et plus précisément sur la notion d'*arbres de preuve*.

Un *arbre de preuve* pt est un arbre fini étiqueté sur l'ensemble des atomes clos (*i.e.* un élément de $\mathbb{TR}_{\mathbb{A}_{(\Sigma,\mathcal{P})}}$) vérifiant : pour toute position d, si H est l'atome étiquetant cette position dans pt et $d_1, \ldots, d_m$ les fils de d et $B_1, \ldots, B_m$, les atomes étiquetant ceux-ci, alors $H \Leftarrow B_1, \ldots, B_m$ est une instance close d'une clause de P [10]. Si la racine de pt est étiquetée par A, on dit alors que pt est un arbre de preuve de A.

Soit $PT_{\mathsf{T}_\Sigma}(P)$, l'ensemble des arbres de preuve pour un programme P et $RPT_{\mathsf{T}_\Sigma}(P)$, l'ensemble des atomes étiquetant la racine des arbres de $PT_{\mathsf{T}_\Sigma}(P)$.

[8] On peut remarquer que la base de Herbrand est toujours modèle d'un programme logique défini et que par conséquent c'est le plus grand modèle de Herbrand de tout programme logique défini.

[9] Nous rappelons qu'une clause, étant universellement quantifiée, ne comporte pas de variable libre du premier ordre.

[10] Ceci implique notamment que les feuilles d'un arbre de preuve sont étiquetées par des instances closes de faits.

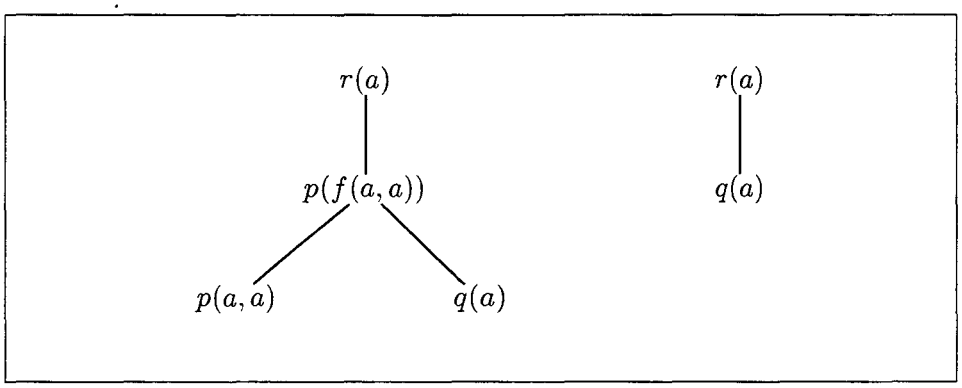

FIG. 1.1: Exemple d'arbres de preuves pour $r(a)$ et le programme P_1

L'équivalence des sémantiques des modèles et des preuves est donnée par :

Théorème 5 [23] $RPT_{\mathsf{T}_\Sigma}(P) = DEN_{\mathsf{T}_\Sigma}(P)$

En revenant au programme P_1 de l'exemple 1, la figure 1.1 présente deux arbres de preuve différents pour l'atome $r(a)$.

Ce résultat peut se formuler pour une algèbre quelconque A de la manière suivante : Un arbre de preuve pour un programme P et une algèbre A est un arbre fini dont les nœuds sont étiquetés par des A-atomes et tel qu'à toute position d, si celle-ci est étiquetée par un A-atome H^{A} et si les fils $d_1, \ldots, d_m$ de cette position sont étiquetés par des A-atomes $B_1^{\mathsf{A}}, \ldots, B_m^{\mathsf{A}}$, alors $H^{\mathsf{A}} \Leftarrow B_1^{\mathsf{A}}, \ldots, B_m^{\mathsf{A}}$ est une A-instance d'une clause de P.

L'ensemble des racines d'arbres de preuves pour un programme P et une algèbre A est exactement $DEN_{\mathsf{A}}(P)$, le plus petit A-modèle du programme P.

Nous donnons maintenant une autre sémantique des programmes logiques en terme de points fixes.

Sémantique des points fixes

Soit $INST_{\mathsf{T}_\Sigma}(P)$, l'ensemble des instances closes des clauses d'un programme logique défini P. On peut remarquer que $(\wp(\mathbb{A}_{(\Sigma,\mathcal{P})}), \subseteq, \cup, \cap, \varnothing, \mathbb{A}_{(\Sigma,\mathcal{P})})$ est un treillis complet.

Définition 7 Soit T_{P,T_Σ}, l'*opérateur de conséquence logique immédiate* (sur l'univers de Herbrand) défini de $\wp(\mathbb{A}_{(\Sigma,\mathcal{P})})$ vers $\wp(\mathbb{A}_{(\Sigma,\mathcal{P})})$ comme suit :

$$T_{P,\mathsf{T}_\Sigma}(I) = \left\{ H \;\middle|\; \begin{array}{l} H \Leftarrow B_1, \ldots, B_l \in INST_{\mathsf{T}_\Sigma}(P) \\ B_1, \ldots, B_l \in I \end{array} \right\}$$

T_{P,T_Σ} est un opérateur monotone et continu sur $(\mathbb{A}_{(\Sigma,\mathcal{P})}, \subseteq, \cup, \cap, \varnothing, \mathbb{A}_{(\Sigma,\mathcal{P})})$. Il admet donc un plus petit point fixe $pppf_{T_{P,\mathsf{T}_\Sigma}}$ et par le théorème de Tarski, ce point fixe est égal à $T_{P,\mathsf{T}_\Sigma} \uparrow \omega$.

L'équivalence entre la sémantique des modèles et la sémantique des points fixes a été établie et se formule par :

Théorème 6 [75] $pppf_{T_{P,\mathsf{T}_\Sigma}} = DEN_{\mathsf{T}_\Sigma}(P)$

On rappelle que $pppf_{T_{P,\mathsf{T}_\Sigma}} = T_{P,\mathsf{T}_\Sigma} \uparrow \omega = \bigcup_{n\in\mathbb{N}} T_{P,\mathsf{T}_\Sigma} \uparrow n$. En fait, il existe une propriété plus forte des modèles de Herbrand d'un programme logique défini qui s'énonce comme suit : Soit M, un ensemble d'atomes clos,

Théorème 7 M est un modèle de Herbrand de P ssi M est un post-point fixe de $T_{P,\mathsf{T}_\Sigma}(M)$, *i.e.* $T_{P,\mathsf{T}_\Sigma}(M) \subseteq M$.

Exemple 2 : *Pour le programme P_1 de l'exemple 1, on a $T_{P,\mathsf{T}_\Sigma} \uparrow 1 = \{p(a,a), p(b,c), q(a)\}$. L'unique instance close des clauses du programme dont les atomes du corps appartiennent à $T_{P,\mathsf{T}_\Sigma} \uparrow 1$ est $p(f(a),a) \Leftarrow p(a,a), q(a)$, et donc,*

$$T_{P,\mathsf{T}_\Sigma}(T_{P,\mathsf{T}_\Sigma} \uparrow 1) = T_{P,\mathsf{T}_\Sigma} \uparrow 2 = \{\{p(a,a),\ p(b,c),\ q(a),\ p(f(a),a)\}$$

De plus, $T_{P,\mathsf{T}_\Sigma}(T_{P,\mathsf{T}_\Sigma} \uparrow 2) = T_{P,\mathsf{T}_\Sigma} \uparrow 2$, donc le plus petit modèle de Herbrand du programme est $\{\{p(a,a),\ p(b,c),\ q(a),\ p(f(a),a),\ r(a)\}$.

Comme il a été fait pour les autres sémantiques, ceci peut être étendu pour une algèbre A quelconque, en définissant l'opérateur $T_{P,\mathsf{A}}$, défini sur l'ensemble de tous les A-atomes par :

$$T_{P,\mathsf{A}}(I) = \left\{ H \;\middle|\; \begin{array}{l} H \Leftarrow B_1,\ldots,B_l \in INST_{\mathsf{A}}(P) \\ B_1,\ldots,B_l \in I \end{array} \right\}$$

Cet opérateur monotone et continue vérifie que $pppf_{T_{P,\mathsf{A}}} = T_{P,\mathsf{A}} \uparrow \omega = DEN_{\mathsf{A}}(P)$.

Nous allons terminer cette section en présentant une sémantique opérationnelle des programmes logiques définis, la SLD-résolution.

Sémantique opérationnelle

Une des sémantiques opérationnelles de la programmation logique est la *SLD-résolution* qui est basée sur le *principe de résolution* défini par Robinson dans [60].

Avant de décrire cette sémantique, nous allons introduire quelques définitions.

On appelle *substitution de renommage*, une application définie d'un sous-ensemble fini de $\mathcal{X}$ (l'ensemble des variables du premier ordre) vers $\mathcal{X}$ telle que cette application soit injective. On appelle *variante* d'une clause c, une instance $\sigma(c)$ de c telle que σ soit une substitution de renommage.

Soient A_1, A_2 deux atomes de $\mathbb{A}_{(\Sigma,\mathcal{P},\mathcal{X})}$. On dit que A_1 et A_2 sont *unifiables* s'il existe une substitution σ idempotente[11] telle que $\sigma(A_1) = \sigma(A_2)$. Dans ce cas, on dit que σ est un unificateur de A_1 et A_2. On dit que σ est l'unificateur le plus général (ou *mgu*) de A_1 et A_2 si et seulement si pour tout unificateur σ' de A_1 et A_2, il existe une substitution σ'' telle que $\sigma = \sigma' \circ \sigma''$.

La SLD-résolution est décrite par une relation $\vdash_{SLD}$ définie sur l'ensemble des buts en fonction d'un programme logique défini P et d'une règle de sélection S[12] par :

Définition 8 Soient $G = \Leftarrow A_1,\ldots,A_m$ et G' deux buts. On notera $G \vdash_{SLD} G'$ si :

- la règle de sélection S sélectionne l'atome A_i,
- θ est le *mgu* de A_i et de H, où $H \Leftarrow B_1,\ldots,B_l$ est une variante d'une clause c de P ne partageant pas de variables avec G,
- $G' = \Leftarrow \theta(A_1),\ldots,\theta(A_{i-1}),\theta(B_1),\ldots,\theta(B_l),\theta(A_{i+1}),\ldots,\theta(A_m)$.

On dit dans ce cas que G' est la résolvante de G et c par θ.

[11] Une substitution σ est dite *idempotente* si $\sigma \circ \sigma = \sigma$.

[12] Cette règle de sélection S peut être définie comme une fonction de l'ensemble des buts vers les atomes et pour but donné $\Leftarrow A_1,\ldots,A_m$ retourne l'un des A_i

L'itération de cette relation permet de définir la notion de SLD-dérivation :

Définition 9

On appelle *SLD-dérivation* une séquence finie $G_0, G_1, \ldots, G_n$ de résolvantes avec pour *mgu* la séquence $\theta_1, \ldots, \theta_n$ (*i.e.* pour toute i, $G_i \vdash_{SLD} G_{i+1}$ avec pour *mgu* θ_i).

On dit qu'une SLD-dérivation est une *SLD-dérivation succès* si $G_n = \square$.

On dit qu'une SLD-dérivation est une *SLD-dérivation échec* si $G_n \neq \square$ et il n'existe pas de G' tel que $G_n \vdash_{SLD} G'$.

Une SLD-dérivation succès ayant pour un but G_0 est appelée une réfutation de G_0 et la substitution $\theta_1 \circ \ldots \circ \theta_n$ restreinte aux variables de G_0 est appelée *substitution réponse* pour le but G_0. Ceci vient du fait que si on note σ cette restriction, alors $P \cup \sigma(G_0)$ est inconsistant.

La *OLD-résolution* est une instance de la SLD-résolution pour laquelle la règle de sélection est « choisir l'atome le plus à gauche », *i.e.* pour tout but $\Leftarrow A_1, \ldots, A_m$, l'atome choisi est A_1.

On peut noter que pour une étape de résolution le choix de la clause est indéterminé (correspondant à $\vdash_{SLD}$), plusieurs (variantes de) clauses pouvant convenir (c-à-d dont la tête peut s'unifier avec l'atome choisi). Pour modéliser ceci, on utilise la notion de SLD-arbre.

Définition 10 Un *SLD-arbre* pour un but G_0 est un arbre fini dont les nœuds sont étiquetés par des buts, la racine étant étiquetée par G_0. Cet arbre vérifie de plus que pour toute position d, si d est étiqueté par un but G et si $\{c_1, \ldots, c_l\}$ est exactement l'ensemble des clauses de P pouvant produire une résolvante sur ce but, alors le nœud d possède l fils, chacun d'entre eux étant étiqueté par un G_i correspondant à la résolvante de G et (d'une variante) de c_i par la substitution θ_i [13].

On peut noter que, par définition, une branche d'un SLD-arbre, est une SLD-dérivation ; Ainsi, on appelle *branche succès* (resp. *branche échec*) une branche d'un SLD-arbre si elle correspond à une SLD-dérivation succès (resp. à une SLD-dérivation échec).

La correction de cette sémantique opérationnelle vis-à-vis de la sémantique des modèles de Herbrand peut être énoncée comme suit :

Théorème 8 Soit A un atome clos tel que $A \in DEN_{\mathsf{T}_\Sigma}(P)$. Pour le programme P, il existe une SLD-dérivation succès pour le but $\Leftarrow A$, ou de manière équivalente il existe un SLD-arbre de racine $\Leftarrow A$ ayant une branche succès.

La notion de succès pour un atome clos amène à définir la notion d'échec.

Définition 11 On dit qu'un atome clos A est un échec fini pour le programme P s'il existe un SLD-arbre de racine $\Leftarrow A$ tel que toutes les branches de cet arbre sont des branches échecs.

Cette notion opérationnelle d'échec fini est liée à l'opérateur de conséquence logique immédiate T_{P,T_Σ} par le théorème suivant :

Théorème 9 Un atome clos A est un échec fini pour le programme P si et seulement si $A \notin T_{P,\mathsf{T}_\Sigma} \downarrow \omega$.

Nous rappelons que $T_{P,\mathsf{T}_\Sigma} \downarrow \omega$ n'est en général pas un point fixe de l'opérateur T_{P,T_Σ} comme cela est montré dans l'exemple 3

[13] On peut remarquer que ceci sous-entend un ordre sur les clauses du programme. Mais on peut de manière équivalente supposer que cette arbre est défini modulo une permutation des $\{c_1, \ldots, c_l\}$.

Exemple 3 : *Soit P_2 le programme logique défini,*

$$p(f(x)) \Leftarrow p(x)$$
$$q(a) \Leftarrow p(x)$$

on a

$$T_{P_2,\mathsf{T}_\Sigma} \downarrow 0 = \mathbb{A}_{(\Sigma,\mathcal{P})}$$
$$T_{P_2,\mathsf{T}_\Sigma} \downarrow 1 = \{q(a), p(f(a)), p(f(f(a))), , p(f(f(f(a)))), \dots\}$$
$$T_{P_2,\mathsf{T}_\Sigma} \downarrow 2 = \{q(a), p(f(f(a))), , p(f(f(f(a)))), \dots\}$$
$$\dots$$
$$T_{P_2,\mathsf{T}_\Sigma} \downarrow \omega = \{q(a)\}$$
$$T_{P_2,\mathsf{T}_\Sigma} \downarrow (\omega + 1) = pgpf_{T_{P_2,\mathsf{T}_\Sigma}} = \varnothing$$

On peut remarquer que pour tout atome clos de la forme $p(f^n(a))$ (f^n désignant la composition de n fois le symbole f) avec $n \geq 0$, le but $\Leftarrow p(f^n(a))$ est un échec fini puisque le seul SLD-arbre constructible a pour feuille $\Leftarrow p(a)$ (qui n'est pas la clause vide) ; il en va de même pour les atomes de la forme $q(f^m(a))$ avec $m \geq 1$, car dans ce cas le SLD-arbre se résume à une racine $\Leftarrow q(f^m(a))$.

Pour l'atome $q(a)$, pour toute toute SLD-dérivation constructible $G_0, G_1, \dots, G_n$, on peut trouver un atome G_{n+1} (unique à un renommage de variables près) tel que $G_n \vdash_{SLD} G_{n+1}$ et $G_0, G_1, \dots, G_n, G_{n+1}$ n'est ni une SLD-dérivation succès, ni SLD-dérivation échec.

Chapitre 2

Contraintes ensemblistes : un état de l'art

2.1 Définition

Nous allons tout d'abord présenter la base « sémantique» des contraintes ensemblistes que sont les algèbres d'ensembles d'arbres. Nous présenterons ensuite la syntaxe de ces contraintes et enfin, leur sémantique.

2.1.1 Algèbres d'ensembles d'arbres

Les définitions suivantes sont extraites de [54]. Nous rappelons que Σ est une signature fonctionnelle, c'est-à-dire un ensemble fini de symboles de fonctions muni d'une fonction d'arité ρ.

Soient $\wp(\mathbb{TR}^{\omega}_{\Sigma})$ et $\wp(\mathbb{TR}_{\Sigma})$ respectivement l'ensemble des ensembles d'arbres et l'ensemble des ensembles d'arbres finis. A chaque constante a de Σ, on associe le singleton $\{a\}$. A chaque symbole f d'arité m supérieure ou égale à 1, on associe une fonction $f^{\mathsf{PTr}^{\omega}_{\Sigma}}$ totale de $(\wp(\mathbb{TR}^{\omega}_{\Sigma}))^m$ dans $\wp(\mathbb{TR}^{\omega}_{\Sigma})$ définie par : pour tout $E_1, \ldots, E_m \in \wp(\mathbb{TR}^{\omega}_{\Sigma})$,

$$f^{\mathsf{PTr}^{\omega}_{\Sigma}}(E_1, \ldots, E_m) = \{f(\tau_1, \ldots, \tau_m) \mid \tau_1 \in E_1, \ldots, \tau_m \in E_m\}\ ^{14}$$

On considérera $f^{\mathsf{PTr}_{\Sigma}}$, la restriction de $f^{\mathsf{PTr}^{\omega}_{\Sigma}}$ à l'ensemble $\wp(\mathbb{TR}_{\Sigma})$.

Les Σ-algèbres $\langle \wp(\mathbb{TR}^{\omega}_{\Sigma}), \{f^{\mathsf{PTr}^{\omega}_{\Sigma}}\}_{f\in\Sigma}\rangle$ et $\langle \wp(\mathbb{TR}_{\Sigma}), \{f^{\mathsf{PTr}_{\Sigma}}\}_{f\in\Sigma}\rangle$ sont appelées respectivement Σ-algèbre d'ensembles d'arbres, notée $\mathsf{PTr}^{\omega}_{\Sigma}$ et Σ-algèbre d'ensembles d'arbres finis, notée PTr_{Σ}.

2.1.2 Contraintes ensemblistes

Les *contraintes ensemblistes* sont apparues (du moins sous ce nom) pour la première fois dans [39]. Elles y sont mentionnées en fait plus précisément comme « Herbrand Set Constraints»[15]. Une contrainte ensembliste (basique) est une formule atomique (du premier ordre) interprétée à l'aide d'une sémantique (en l'occurrence, une structure) fixée. Cette structure est basée sur une

[14] La notation $f(\tau_1, \ldots, \tau_m)$ est un abus et est mise pour $f^{\mathsf{Tr}^{\omega}_{\Sigma}}(\tau_1, \ldots, \tau_m)$, $f^{\mathsf{Tr}^{\omega}_{\Sigma}}$ étant la fonction associée au symbole f dans l'algèbre des arbres $\mathsf{Tr}^{\omega}_{\Sigma}$.

[15] En français « Contraintes Ensemblistes de Herbrand».

$$Sexp ::= X \mid f(Sexp, \ldots, Sexp) \mid Sexp \cup Sexp \mid Sexp \cap Sexp \mid \complement Sexp \mid f_i^{-1}(Sexp) \mid \top \mid \bot$$

$$\psi ::= Sexp \subseteq Sexp \mid Sexp \not\subseteq Sexp \mid Sexp = Sexp$$

avec $X \in \mathcal{V}$, $f \in \Sigma$ et $f_i^{-1} \in \Sigma_{-1}$.

FIG. 2.1: Syntaxe abstraite des contraintes ensemblistes

algèbre d'ensembles d'arbres finis (d'où le terme « Herbrand ») étendue pour prendre en compte des opérations et des relations ensemblistes.

Nous rappelons dans cette section, les définitions tant syntaxique que sémantique se trouvant dans [39], et reprises dans la plupart des travaux postérieurs.

Considérons Σ, une signature fonctionnelle et $\mathcal{V}$, un ensemble de variables dites *ensemblistes*. Soient Σ_B, Σ_{-1}, Σ_n trois signatures fonctionnelles avec $\Sigma_B = \{\cup, \cap, \complement\}$, dont les symboles de fonctions sont binaires pour les deux premiers et unaire pour le dernier, et appelés union, intersection et complémentaire. Σ_{-1} est l'ensemble des symboles de fonctions unaires f_i^{-1} pour $f \in \Sigma$ et $1 \leq i \leq \rho(f)$, appelés symboles de projections. Finalement, Σ_n ne contient que deux symboles nullaires $\top$ et $\bot$. Nous scinderons l'ensemble Σ_B en deux parties $\Sigma_{B+} = \{\cup, \cap\}$ et $\Sigma_{B-} = \{\complement\}$. Considérons également, les symboles de prédicats binaires $\subseteq$, $\not\subseteq$ et $=$, appelés respectivement inclusion, non-inclusion et égalité.

Une *expression ensembliste* $Sexp$ est un terme construit sur Σ, Σ_B, Σ_n, Σ_{-1} et $\mathcal{V}$, *i.e.* un élément de $TERM(\Sigma \cup \Sigma_B \cup \Sigma_n \cup \Sigma_{-1}, \mathcal{V})$. Par commodité, les symboles $\cup$ et $\cap$ seront utilisés en écriture infixée ; par exemple, au lieu de $\cup(f(X,Y), f(a,Z))$, nous écrirons $f(X,Y) \cup f(a,Z)$.

Une *contrainte ensembliste* ψ est un atome $Sexp_1 \subseteq Sexp_2$, $Sexp_1 \not\subseteq Sexp_2$ ou $Sexp_1 = Sexp_2$, où $Sexp_1$ et $Sexp_2$ sont des expressions ensemblistes.

Une définition sous forme de syntaxe abstraite des expressions et contraintes ensemblistes est donnée figure 2.1.

La sémantique des expressions ensemblistes est basée sur la Σ-algèbre d'ensembles des arbres finis (construits sur Σ) PTr_Σ, étendue en une algèbre, de même support (*i.e.* $\wp(\mathbb{TR}_\Sigma)$) notée SC, fixant une sémantique pour les symboles de fonctions de Σ_B, Σ_{-1} et Σ_n comme suit : Soient $E_1, E_2 \in \wp(\mathbb{TR}_\Sigma)$,

$$\bot^{\mathsf{SC}} = \varnothing \qquad E_1 \cup^{\mathsf{SC}} E_2 = E_1 \cup E_2$$
$$\top^{\mathsf{SC}} = \mathbb{TR}_\Sigma \qquad E_1 \cap^{\mathsf{SC}} E_2 = E_1 \cap E_2$$
$$\complement^{\mathsf{SC}} E_1 = \mathbb{TR}_\Sigma \setminus E_1 \qquad (f_i^{-1})^{\mathsf{SC}}(E_1) = \{\tau_i \mid f(\tau_1, \ldots, \tau_m) \in E_1\}$$

Une *interprétation ensembliste* $\mathcal{I}$ est une valuation de $\mathcal{V}$ dans $\wp(\mathbb{TR}_\Sigma)$. Afin d'alléger les notations, pour une expression ensembliste $Sexp$, nous écrirons $\mathcal{I}(Sexp)$ au lieu de $\lfloor Sexp \rfloor_{\mathcal{I}}^{\mathsf{SC}}$ en supposant que l'interprétation $\mathcal{I}$ a été étendue à l'ensemble des expressions ensemblistes selon la sémantique définie par SC.

La structure $\mathsf{SC}_\subseteq$ est la structure basée sur l'algèbre SC et associe aux symboles $\subseteq$, $\not\subseteq$ et

$=$, les relations $\subseteq^{\mathsf{SC}_\subseteq}$, $\not\subseteq^{\mathsf{SC}_\subseteq}$ et $=^{\mathsf{SC}_\subseteq}$, désignant respectivement l'inclusion, la non-inclusion et l'égalité dans $\wp(\mathbb{T}_{\mathbb{R}_\Sigma})$. Lorsqu'il n'y aura aucune ambiguïté possible, nous écrirons simplement $\subseteq$, $\not\subseteq$ et $=$ au lieu de $\subseteq^{\mathsf{SC}_\subseteq}$, $\not\subseteq^{\mathsf{SC}_\subseteq}$ et $=^{\mathsf{SC}_\subseteq}$.

Une contrainte ensembliste ψ est dite satisfiable s'il existe une interprétation ensembliste $\mathcal{I}$ telle que $(\mathsf{SC}_\subseteq, \mathcal{I}) \models \psi$, c'est-à-dire si ψ est de la forme $Sexp_1 \subseteq Sexp_2$ (resp. $Sexp_1 \not\subseteq Sexp_2$, $Sexp_1 = Sexp_2$), alors $\mathcal{I}(Sexp_1) \subseteq \mathcal{I}(Sexp_2)$ (resp. $\mathcal{I}(Sexp_1) \not\subseteq \mathcal{I}(Sexp_2)$, $\mathcal{I}(Sexp_1) = \mathcal{I}(Sexp_2)$). On dira alors que $\mathcal{I}$ est une *solution* de la contrainte ψ. On notera $SOL(\psi)$, l'ensemble des solutions d'une contrainte ψ.

On peut définir sur l'ensemble des interprétations ensemblistes (définies sur un même ensemble $\mathcal{V}$ de variables) une relation d'ordre très naturelle. Cette relation, notée $\sqsubseteq$, est définie pour deux interprétations $\mathcal{I}$, $\mathcal{I}'$ par $\mathcal{I} \sqsubseteq \mathcal{I}'$ si pour toute variable X de $\mathcal{V}$, $\mathcal{I}(X)$ est un sous-ensemble de $\mathcal{I}'(X)$. Sur ce même ensemble, muni de cette ordre $\sqsubseteq$, les opérateurs de borne supérieure $\sqcup$ et borne inférieure $\sqcap$ correspondent à l'intersection et à l'union, *i.e.* pour un ensemble S d'interprétations ensemblistes définies pour un ensemble $\mathcal{V}$, $\sqcap S$ et $\sqcup S$ vérifient respectivement pour tout $X \in \mathcal{V}$, $\sqcap S(X) = \cap_{\mathcal{I} \in S} \mathcal{I}(X)$ et $\sqcup S(X) = \cup_{\mathcal{I} \in S} \mathcal{I}(X)$. Ainsi, si $Int_\mathcal{V}$ désigne l'ensemble des interprétations ensemblistes (définies sur un ensemble de variables $\mathcal{V}$), $\mathcal{I}_\bot$ est l'interprétation ensembliste associant à chaque variable de $\mathcal{V}$, l'ensemble vide et $\mathcal{I}_\top$ est l'interprétation associant à chaque variable l'ensemble de tous les arbres, alors $(Int_\mathcal{V}, \sqcap, \sqcup, \mathcal{I}_\bot, \mathcal{I}_\top)$ forme un treillis complet.

Exemple 4 : *Voici deux exemples de contraintes ensemblistes :*

$$\psi_1 \equiv a \subseteq f(X, Y) \qquad \psi_2 \equiv N \subseteq 0 \cup s(N)$$

Il est bien évident que la contrainte ψ_1 est insatisfiable : le singleton $\{a\}$ ne peut être inclus dans un ensemble dont tous les termes ont f pour symbole de tête. La contrainte ψ_2 possède plusieurs solutions (en fait, une infinité) : ainsi, l'interprétation associant à la variable N, l'ensemble vide ($\varnothing$) est une solution (puisque le vide est inclus dans tous les ensembles). On peut également noter que cette solution est nécessairement la plus petite au sens de $\sqsubseteq$. Associer à N le singleton $\{0\}$ nous donne une autre solution puisque $\{0\}$ est bien inclus dans $\{0, s(0)\}$. Enfin, si $\mathcal{I}(N)$ est égal à l'ensemble des termes clos construits uniquement à l'aide de s et 0, i.e. $\{0, s(0), s(s(0)), s(s(s(0))), \dots\}$, alors $\mathcal{I}$ est aussi une solution. De plus, cette solution est la plus grande au sens de $\sqsubseteq$.

Les contraintes ensemblistes étant définies comme des atomes de la logique du premier ordre, on peut alors considérer, comme langage de domaine de contraintes, l'ensembles des formules du premier ordre écrites avec ces atomes.

2.2 Théorie du premier ordre des contraintes ensemblistes

Considérons les formules du premier ordre construites à l'aide des contraintes ensemblistes basiques,

Définition 12 La *théorie (du premier ordre) des contraintes ensemblistes* est syntaxiquement définie comme l'ensemble des formules du premier ordre dont les atomes sont les contraintes ensemblistes basiques. Cette théorie est notée $\mathbb{L}_{SC}$. La sémantique de cette théorie est fixée par $\mathsf{SC}_\subseteq$.

Nous allons voir dans cette section que le problème de la validité d'une formule dans la théorie du premier ordre des contraintes ensemblistes est indécidable. Prouver ceci est en fait très simple,

puisque qu'on peut encoder la *théorie monadique du second ordre de l'arbre (fini)* dans la théorie du premier ordre des contraintes ensemblistes.

La théorie monadique du second ordre de l'arbre (fini) est syntaxiquement définie par les formules de $\mathbb{L}^2_{(\Sigma,\mathcal{V},\mathcal{X})}$ et sémantiquement interprétée sur une structure $\langle\langle \mathsf{T}_\Sigma, \mathcal{I}\rangle\rangle$, $\mathcal{I}$ associant à chaque variable libre du second ordre (*i.e.* appartenant à $\mathcal{V}$) un sous-ensemble de $TERM(\Sigma)$. Nous rappelons que la syntaxe des formules de $\mathbb{L}^2_{(\Sigma,\mathcal{V},\mathcal{X})}$ est la suivante :

$$\varphi ::= t \in X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \neg\varphi \mid \exists x\varphi \mid \forall x\varphi \mid \exists X\varphi \mid \forall X\varphi$$

où x et X sont des variables respectivement du premier et du second ordre et t est un terme de $TERM(\Sigma, \mathcal{X})$.

L'idée principale de l'encodage de cette théorie dans la théorie du premier ordre des contraintes ensemblistes basiques est d'identifier un terme clos t avec le singleton $\{t\}$ et de remplacer le prédicat d'appartenance $\in$ par une inclusion $\subseteq$.

Si l'on se donne une formule monadique du second ordre φ, on peut construire récursivement sur la structure de cette formule, une formule φ_{SC} de $\mathbb{L}_{SC}$ dont les variables ensemblistes sont les variables du premier et du second ordre de φ. Cette construction est donnée par :

$$\varphi_{SC} = \begin{cases} t \subseteq X & si\ \ \varphi = t \in X \\ \varphi^1_{SC} \wedge \varphi^2_{SC} & si\ \ \varphi = \varphi^1 \wedge \varphi^2 \\ \varphi^1_{SC} \vee \varphi^2_{SC} & si\ \ \varphi = \varphi^1 \vee \varphi^2 \\ \exists X\, \varphi'_{SC} & si\ \ \varphi = \exists X\varphi' \\ \forall X\, \varphi'_{SC} & si\ \ \varphi = \forall X\varphi' \\ \exists x\, (Singleton(x) \wedge \varphi'_{SC}) & si\ \ \varphi = \exists x\varphi' \\ \forall x\, (Singleton(x) \Rightarrow \varphi'_{SC}) & si\ \ \varphi = \forall x\varphi' \end{cases}$$

Singleton(x) étant une formule qui est vraie pour une valuation ensembliste $\mathcal{I}$ ssi $\mathcal{I}(x)$ est un singleton. Cette formule peut donc s'écrire comme

$$Singleton(x) \equiv \neg(x \subseteq a \wedge x \subseteq b) \wedge \forall y(\neg(y \subseteq a \wedge y \subseteq b) \wedge y \subseteq x) \Rightarrow y = x)$$

Les expression de la forme $\neg(z \subseteq a \wedge z \subseteq b)$ assurant simplement que la valeur de la variable x n'est pas l'ensemble vide.

Pour une formule monadique du second ordre φ et la formule de la théorie du premier ordre de la théorie des contraintes ensemblistes φ_{SC} construite à partir de φ selon le mécanisme précédemment décrit, considérons la formule Φ suivante :

$$\Phi = \bigwedge_{x \in Var_{\mathcal{X}}(\varphi)} (Singleton(x) \Rightarrow \varphi_{SC})$$

Il est évident de voir que la formule φ est valide pour l'algèbre T_Σ si et seulement si Φ est valide dans $\mathsf{SC}_\subseteq$. Or, il est connu que le problème de validité d'une formule dans la théorie monadique du second ordre de l'arbre est indécidable [68]. On en déduit donc l'indécidabilité de la théorie du premier ordre des contraintes ensemblistes.

On peut noter que le résultat précédent amenait à écrire une formule dont les contraintes ne comportaient aucun opérateur ensembliste. Un résultat d'indécidabilité plus précis pour cette

théorie a été donné [65] : ce résultat concerne le fragment $\exists^*\forall^*$ de la théorie du premier ordre des contraintes ensemblistes, où les contraintes ne contiennent que l'opérateur ensembliste d'union $\cup$.

Nous allons donner ici une nouvelle preuve qui démontre que cette indécidabilité est toujours présente (pour un fragment équivalent) même si l'on ne considère que des contraintes ne comportant aucun opérateur ensembliste. De telles contraintes sont appelées dans [39] *contraintes ensemblistes atomiques.*

Cette preuve d'indécidabilité se fait par réduction du problème de *correspondance de Post (PCP)* vers celui de la validité d'une formule dans la théorie des contraintes ensemblistes atomiques. Nous rappelons que le problème de correspondance de Post s'énonce comme suit :

PROBLÈME DE CORRESPONDANCE DE POST
Données : Soient A, un alphabet (fini) et $U = \{u_1, \ldots, u_m\}$, $V = \{v_1, \ldots, v_m\}$, deux ensembles de mots (ne contenant pas le mot vide ϵ) construits sur A.
Question : Existe-t-il une suite d'indices $[i_1, \ldots, i_n]$ telle que :

$$u_{i_1} \cdot \ldots \cdot u_{i_n} = v_{i_1} \cdot \ldots \cdot v_{i_n}$$

Théorème 10 [57] Le Problème de Correspondance de Post est indécidable.

Pour une instance de ce problème, nous allons construire une formule φ_{PCP} de $\mathbb{L}_{SC}$ valide sur $\mathsf{SC}_{\subseteq}$ si et seulement si l'instance correspondante du problème de Post a une solution.

Comme il est de coutume, à chaque lettre de l'alphabet A, on fait correspondre dans Σ un symbole a d'arité 1 et on suppose présent dans Σ une constante ϵ servant de « marqueur de fin » de mot. Ainsi, le mot *abac* est codé par le terme $a(b(a(c(\epsilon))))$. Nous confondrons désormais les termes clos construits avec les symboles unaires de Σ (correspondant aux symboles de A) et la constante ϵ avec les mots de A^*.

On suppose également que Σ contient un symbole de fonction binaire *post*. Tous ces symboles définissent complètement Σ.

Considérons tout d'abord la formule monadique du premier ordre φ universellement quantifiée de $\mathbb{L}_{(\Sigma,\mathcal{X},\{I\})}$, *i.e.* construite sur Σ et $\mathcal{X}$ et ne comportant qu'une variable du second ordre I :

$$\forall \bigwedge_{1 \leq i \leq m} (post(u_i(\epsilon), v_i(\epsilon)) \in I \ \wedge \ (post(x_i, y_i) \in I \Rightarrow post(u_i(x_i), v_i(y_i)) \in I))$$

Les expressions $u_i(\epsilon)$ et $u_i(x_i)$ ne sont que des abréviations : par exemple, si u_i est le mot *abac*, alors les deux termes correspondants sont $a(b(a(c(\epsilon))))$ et $a(b(a(c(x_i))))$.

On peut remarquer que cette formule φ est en fait une conjonction de clauses définies écrites sous une forme monadique. Ainsi, cette formule peut être vue comme un programme logique, possédant un plus petit modèle de Herbrand $DEN_{\mathsf{T}_\Sigma}(\varphi)$ (voir section 1.4.2.0).

Proposition 2 Il existe un terme clos τ tel que $\varphi \models post(\tau, \tau) \in I$ (ou de manière équivalente $post(\tau, \tau)$ appartient à $DEN_{\mathsf{T}_\Sigma}(\varphi)$, le plus petit modèle de Herbrand de φ) si et seulement si le mot représenté par le terme τ est solution de l'instance *(PCP)* correspondante.
Preuve :

Montrons tout d'abord que le plus petit modèle de Herbrand de φ pour la variable de second ordre I est égal à l'ensemble des termes clos de la forme $post(\tau_u, \tau_v)$, où τ_u, τ_v sont construits par concaténation respectivement sur U et V à l'aide de la même suite d'indices.
En définissant S_n comme l'ensemble des termes clos de la forme $post(\tau_u, \tau_v)$, où τ_u, τ_v sont construits par concaténation respectivement sur U et V à l'aide de la même suite d'indices de longueur égale à n, la précédente phrase peut être reformulée comme :

$$\bigcup_{n\in\mathbb{N}} S_n = \bigcup_{n\in\mathbb{N}} T_{\varphi,\mathsf{T}_\Sigma} \uparrow n$$

Commençons par montrer par induction sur n que $\forall n > 0$, $S_n \subseteq T_{\varphi,\mathsf{T}_\Sigma} \uparrow n$:

- *pour $n = 1$: si $post(\tau_u, \tau_v)$ appartient à S_1, alors il existe u_i, v_i deux mots de respectivement U et V tel que $post(\tau_u, \tau_v) = post(u_i(\epsilon), v_i(\epsilon))$. Du fait de φ, $post(u_i(\epsilon), v_i(\epsilon)) \in T_{\varphi,\mathsf{T}_\Sigma} \uparrow 1$.*
- *pour $n > 1$: si $post(\tau_u, \tau_v)$ appartient à S_n, alors il existe u_i, v_i deux mots de respectivement U et V et deux mots τ'_u, τ'_v tels que $post(\tau'_u, \tau'_v) \in S_{n-1}$ et $post(\tau_u, \tau_v) = post(u_i(\tau'_u), v_i(\tau'_v))$. Par hypothèse d'induction, $post(\tau'_u, \tau'_v) \in T_{\varphi,\mathsf{T}_\Sigma} \uparrow (n-1)$. Or par l'instance de la $i^{ième}$ règle, $post(\tau'_u, \tau'_v) \Rightarrow post(u_i(\tau'_u), v_i(\tau'_v))$, on a :*

$$post(\tau_u, \tau_v) = post(u_i(\tau'_u), v_i(\tau'_v)) \in T_{\varphi,\mathsf{T}_\Sigma}(T_{\varphi,\mathsf{T}_\Sigma} \uparrow (n-1)) = T_{\varphi,\mathsf{T}_\Sigma} \uparrow n$$

Puisque $S_0 = T_{\varphi,\mathsf{T}_\Sigma} \uparrow 0 = \varnothing$,

$$\bigcup_{n\in\mathbb{N}} S_n = \bigcup_{n\in\mathbb{N}_+} S_n \subseteq \bigcup_{n\in\mathbb{N}_+} T_{\varphi,\mathsf{T}_\Sigma} \uparrow n = \bigcup_{n\in\mathbb{N}} T_{\varphi,\mathsf{T}_\Sigma} \uparrow n$$

Montrons maintenant que pour tout $n > 0$, $T_{\varphi,\mathsf{T}_\Sigma} \uparrow n \subseteq \bigcup_{1\leq i\leq n} S_i$. Par induction sur n,

- *pour $n = 1$: $T_{\varphi,\mathsf{T}_\Sigma} \uparrow 1$ est l'ensemble des faits du programme φ, qui correspondent exactement aux termes clos $post(\tau_u, \tau_v)$, où τ_u, τ_v sont construits par concaténation respectivement sur U et V à l'aide de la même suite d'indices de longueur 1. Donc, $T_{\varphi,\mathsf{T}_\Sigma} \uparrow 1 \subseteq S_1$.*
- *pour $n > 1$: si $post(\tau_u, \tau_v)$ appartient à $T_{\varphi,\mathsf{T}_\Sigma} \uparrow n$, alors il existe une instance de clause de φ telle que la tête de celle-ci est de la forme $post(\tau_u, \tau_v) \in I$ et que les termes clos du corps appartiennent à $T_{\varphi,\mathsf{T}_\Sigma} \uparrow (n-1)$. Selon le type de la clause :*
  - *Si la clause est un fait, alors du fait du cas de base, $post(\tau_u, \tau_v) \in S_1 \subseteq \bigcup_{1\leq i\leq n} S_i$.*
  - *Si la clause est une règle $post(\tau'_u, \tau'_v) \in I \Rightarrow post(u_i(\tau'_u), v_i(\tau'_v)) \in I$ avec $post(\tau_u, \tau_v) = post(u_i(\tau'_u), v_i(\tau'_v))$. Par hypothèse de récurrence, le terme clos $post(\tau'_u, \tau'_v)$ appartient à $\bigcup_{1\leq i\leq(n-1)} S_i$, i.e. que τ'_u et τ'_v sont deux mots construits par concaténation respectivement sur U et V à l'aide de la même suite d'indices de longueur inférieure ou égale à $n-1$. Donc, $post(\tau_u, \tau_v) \in \bigcup_{1\leq i\leq n} S_i$.*

On en conclut que :

$$\bigcup_{n\in\mathbb{N}} T_{\varphi,\mathsf{T}_\Sigma} \uparrow n \subseteq \bigcup_{n\in\mathbb{N}} \bigcup_{1\leq i\leq n} S_i = \bigcup_{n\in\mathbb{N}} S_n$$

On peut déduire de cette double inclusion que si $post(\tau, \tau)$ appartient à $DEN_{\mathsf{T}_\Sigma}(\varphi)$, alors τ peut être construit à la fois sur U et sur V à l'aide de la même suite d'indices et donc par définition, est solution de l'instance considérée du problème de Post.

■

Considérons maintenant la formule φ_{SC} (dont toutes les variables, à l'exception de I, sont universellement quantifiées) de $\mathbb{L}_{SC}$ suivante :

$$\forall_{-I}(\bigwedge_{1\leq i\leq m} post(u_i(\epsilon),v_i(\epsilon)) \subseteq I \wedge \bigwedge_{1\leq i\leq m} post(X_i,Y_i) \subseteq I \Rightarrow post(u_i(X_i),v_i(Y_i)) \subseteq I)$$

Soit $\mathcal{I}$, une application de $\{I\}$ vers $\wp(TERM(\Sigma))$. Cette application est par définition une interprétation ensembliste pour la variable ensembliste I. Mais nous avons vu qu'une telle application peut aussi être considérée comme définissant la sémantique de la variable du second ordre I dans une structure construite au dessus de l'algèbre de Herbrand.

On a alors que tout modèle de Herbrand de la formule φ est solution de la formule de la théorie du premier ordre des contraintes ensemblistes atomiques φ_{SC}, c-à-d

Proposition 3 $\langle\langle \mathsf{T}_\Sigma, \mathcal{I}\rangle\rangle \models \varphi$ ssi $(\mathsf{SC}_\subseteq, \mathcal{I}) \models \varphi_{SC}$

Preuve :

Montrons tout d'abord que $\langle\langle \mathsf{T}_\Sigma, \mathcal{I}\rangle\rangle \models \varphi$ *implique* $(\mathsf{SC}_\subseteq, \mathcal{I}) \models \varphi_{SC}$ *: soit* $M = \mathcal{I}(I)$*, un modèle de Herbrand de* φ*. La formule* φ *étant vue comme un programme logique, on a* $DEN_{\mathsf{T}_\Sigma}(\varphi) \subseteq M$*. Montrons que* M *est une solution de* φ_{SC}*, selon la forme des clauses de* φ*, i.e. chacun des éléments de la conjonction composant* φ*,*

- *Chacun des faits* $post(u_i(\epsilon), v_i(\epsilon))$ *appartient à* $T_{\varphi,\mathsf{T}_\Sigma} \uparrow 1 = T_{\varphi,\mathsf{T}_\Sigma}(\varnothing)$*. Or puisque* $T_{\varphi,\mathsf{T}_\Sigma}$ *est monotone, on a* $T_{\varphi,\mathsf{T}_\Sigma}(\varnothing) \subseteq T_{\varphi,\mathsf{T}_\Sigma}(M)$ *et comme* M *est un modèle de Herbrand de* φ*,* $T_{\varphi,\mathsf{T}_\Sigma}(\varnothing) \subseteq M$*.*
- *pour les clauses de la forme* $\forall X_i, Y_i\, post(X_i, Y_i) \subseteq I \Rightarrow post(u_i(X_i), v_i(Y_i)) \subseteq I$ *: pour toute interprétation (ensembliste)* $\mathcal{I}'$ *telle que* $\mathcal{I}'(I) = M$*, si* $post(\mathcal{I}'(X_i), \mathcal{I}'(Y_i)) \subseteq M$*, alors* $T_{\varphi,\mathsf{T}_\Sigma}$ *étant monotone, on a*

$$T_{P_\varphi,\mathsf{T}_\Sigma}(post(\mathcal{I}'(X_i), \mathcal{I}'(Y_i))) \subseteq T_{P_\varphi,\mathsf{T}_\Sigma}(M)$$

Or M *étant un modèle de Herbrand du programme,* $T_{\varphi,\mathsf{T}_\Sigma}(M) \subseteq M$*, il est donc suffisant de montrer que*

$$post(\mathcal{I}'(u_i(X_i)), \mathcal{I}'(v_i(Y_i))) \subseteq T_{\varphi,\mathsf{T}_\Sigma}(post(\mathcal{I}'(X_i), \mathcal{I}'(Y_i)))$$

Par définition, $post(t'_1, t'_2) \in post(\mathcal{I}'(u_i(X_i)), \mathcal{I}'(v_i(Y_i)))$ *si et seulement si* t'_1 *et* t'_2 *sont respectivement de la forme* $u_i(t_2)$ *et* $v_i(t_2)$ *avec* $t_1 \in \mathcal{I}'(X_i)$ *et* $t_2 \in \mathcal{I}'(Y_i)$ *et donc,* $post(t_1, t_2) \in post(\mathcal{I}'(X_i), \mathcal{I}'(Y_i))$*. Or du fait de l'instance (close) de la clause considérée* $post(t_1, t_2) \subseteq I \Rightarrow post(u_i(t_1), v_i(t_2)) \subseteq I$*, on a nécessairement que*

$$post(u_i(t_1), v_i(t_2)) = post(t'_1, t'_2) \in T_{\varphi,\mathsf{T}_\Sigma}(post(\mathcal{I}'(X_i), \mathcal{I}'(Y_i)))$$

Montrons maintenant que $(\mathsf{SC}_\subseteq, \mathcal{I}) \models \varphi_{SC}$ *implique* $\langle\langle \mathsf{T}_\Sigma, \mathcal{I}\rangle\rangle \models \varphi$ *: considérons une solution* $\mathcal{I}$ *de la contrainte* φ_{SC} *associant à la variable ensembliste* I *un ensemble de termes* M*. Montrons que* M *est un modèle de Herbrand de* φ*. Pour cela, il suffit de prouver que* $T_{\varphi,\mathsf{T}_\Sigma}(M) \subseteq M$*, i.e. pour toute instance close d'une clause de* φ *de la forme* $H \Leftarrow B_1, \ldots, B_m$*, si* $B_1, \ldots, B_m \in M$*, alors* $H \in M$*. Selon la forme des clauses,*

- *pour les faits : de part la formule* φ_{SC}*,* $(\mathsf{SC}_\subseteq, \mathcal{I}) \models post(u_i(\epsilon), v_i(\epsilon)) \subseteq I$ *et donc,* $post(u_i(\epsilon), v_i(\epsilon)) \in M$*.*

- *pour les règles : soient* t_1, t_2 *tels que* $post(t_1, t_2) \in M$. *Ceci implique que* $(\mathsf{SC}_{\subseteq}, \mathcal{I}) \models post(t_1, t_2) \subseteq I$. *Or, on a aussi*

$$(\mathsf{SC}_{\subseteq}, \mathcal{I}) \models \forall X_i \forall Y_i \, post(X_i, Y_i) \subseteq I \Rightarrow post(u_i(X_i), v_i(Y_i)) \subseteq I$$

et en particulier pour les singletons $\{t_1\}, \{t_2\}$. *On peut en déduire que* $(\mathsf{SC}_{\subseteq}, \mathcal{I}) \models post(u_i(t_1), v_i(t_2)) \subseteq I$, *et donc* $post(u_i(t_1), v_i(t_2)) \in \mathcal{I}(I) = M$.

■

On peut déduire de cela que parmi les solutions pour la structure $\mathsf{SC}_{\subseteq}$ de la formule φ_{SC}, il en existe une plus petite (au sens de l'inclusion), *i.e.* qui est contenue dans toutes les autres solutions.

Du fait des propositions 2 et 3, l'instance du Problème de Post considérée admet une solution si dans toutes les solutions de φ_{SC} et donc en particulier dans la plus petite, on peut trouver un ensemble non vide S tel que $post(S, S) \subseteq I$. En effet, pour un terme τ appartenant à S, $post(\tau, \tau) \subseteq I$ est alors conséquence logique de φ_{SC} pour $\mathsf{SC}_{\subseteq}$.

Soit φ_{PCP}, la formule de $\mathbb{L}_{SC}$ suivante :

$$\forall I \, \varphi_{\mathsf{SC}} \Rightarrow [\exists S \neg(S \subseteq \epsilon \wedge S \subseteq u_1(\epsilon)) \ \wedge \ post(S, S) \subseteq I]$$

L'expression $\neg(S \subseteq \epsilon \wedge S \subseteq u_1(\epsilon))$ assure que l'interprétation de la variable ensembliste S est non vide dans toutes les modèles de φ_{PCP}, puisque seul l'ensemble vide peut à la fois être inclus dans les singletons $\{\epsilon\}$ et $\{u_1(\epsilon)\}$.

Proposition 4 φ_{PCP} est valide sur $\mathsf{SC}_{\subseteq}$ si et seulement si l'instance correspondante du problème de Post a une solution.

Preuve :

Immédiat, par les propositions 2 et 3 et la remarque ci-dessus.

■

On en déduit donc

Théorème 11 Le fragment $\forall\exists^*$ de la théorie du premier ordre des contraintes ensemblistes atomiques est indécidable.

Preuve :

Par simple mise en forme prénexe de φ_{PCP}.

■

Notons tout de suite que, lorsque l'on parle de contraintes ne comportant pas de variable libre, la structure d'interprétation de ces contraintes étant fixée (ici, à $\mathsf{SC}_{\subseteq}$), les notions de validité et de satisfiabilité de formules coïncident. Ainsi, en considérant la contraposée de (la forme prénexée de) la formule φ_{PCP}, on en déduit que le fragment $\exists\forall^*$ de la théorie du premier ordre des contraintes ensemblistes atomiques est indécidable ; ce résultat améliore donc bien celui de [65].

Ce résultat montre également la différence existant entre les relations d'inclusion et égalitaire ensemblistes. En effet, de nombreux travaux ont eu pour but de donner une axiomatisation complète de *la théorie égalitaire de l'arbre* [48, 49] qui permet de décider de la validité d'une formule du premier ordre écrite sur une signature fonctionnelle Σ (finie ou infinie) et utilisant comme unique symbole de prédicat l'égalité $=$. Ces formules sont interprétées sur l'algèbre des termes clos T_Σ, étendue dans la structure où $=$ est interprété comme l'identité. Comme il est

noté dans [50], cette axiomatisation reste valide si on considère comme structure d'interprétation l'algèbre des ensembles non-vides de termes clos, étendue en interprétant une nouvelle fois le prédicat $=$ comme l'identité, prouvant la décidabilité de la théorie égalitaire des ensembles non-vides d'arbres. On peut remarquer que notre approche permet également de considérer cette structure des ensembles d'arbres non-vides, et donc comme corollaire de notre résultat, on peut affirmer que les fragments $\forall\exists^*$ et $\exists\forall^*$ de la théorie de l'inclusion sur les ensembles non-vides d'arbres sont indécidables.

On peut également noter qu'en utilisant le fait que $X \subseteq Y \equiv_{\mathsf{SC}_{\subseteq}} (X \cap Y = X)$ et $X \subseteq Y \equiv_{\mathsf{SC}_{\subseteq}} (X \cup Y = Y)$, on peut déduire du résultat présenté dans cette section que le fragment $\exists\forall^*$ de la théorie ensembliste égalitaire avec intersection (resp. avec union) est indécidable et ce pour une interprétation sur les ensembles de termes clos ou sur les ensembles non-vides de termes clos.

Dans cette section, nous avons montré que le problème de validité d'une formule de la théorie du premier ordre des contraintes ensemblistes était indécidable. Cependant, il est toujours possible de se poser cette question (de validité ou de satisfiabilité) dans un cadre plus restreint.

Par exemple, en supposant que la signature fonctionnelle Σ ne comporte que des symboles d'arité inférieure ou égale à 1, le problème de validité ou de satisfiabilité d'une formule de la théorie du premier ordre des contraintes ensemblistes peut être encoder dans le problème correspondant de la *théorie monadique du second ordre à k successeurs* démontrée décidable par Rabin [58].

Une autre possibilité de restriction de cette théorie du premier ordre des contraintes ensemblistes est d'imposer des conditions syntaxiques sur les formules manipulées. Dans ce cas également, certains fragments peuvent se révéler décidables.

2.3 Systèmes de contraintes ensemblistes

Un système de contraintes ensemblistes Ψ est un ensemble de contraintes ensemblistes, sémantiquement interprété comme la conjonction de ces contraintes. Un système de contraintes ensemblistes peut donc être vu comme une formule de $\mathbb{L}_{SC}$ ne comportant que des conjonctions ($\wedge$) et sans quantification. Ainsi, un système Ψ est satisfiable s'il existe une valuation $\mathcal{I}$ qui soit solution de chacune des contraintes ψ appartenant à Ψ. On notera $SOL(\Psi)$ l'ensemble des solutions d'un système de contraintes ensemblistes Ψ.

Comme nous l'avons dit précédemment, les définitions tant syntaxique que sémantique concernant les contraintes ensemblistes sont apparues dans [39]. Cependant, cet article restreignait la définition d'une contrainte ensembliste en considérant pour symbole de prédicat uniquement l'inclusion et en limitant la forme des expressions ensemblistes pouvant apparaître en partie droite et gauche d'une inclusion. Cette classe de contraintes ensemblistes a été appelée classe des *contraintes définies (DSC)*. Nous reviendrons plus précisément sur le travail présenté dans cet article plus avant dans ce chapitre. Citons simplement un résultat important concernant cette classe : tout système de contraintes ensemblistes définies satisfiable admet une plus petite solution au sens de $\sqsubseteq$.

A la suite de ce travail, de nombreuses équipes ont entrepris de résoudre le problème de décidabilité de la satisfiabilité des contraintes ensemblistes pour la classe de contraintes la plus large possible, c'est-à-dire avec le moins de restrictions possibles sur les symboles de prédicats utilisés et la forme des expressions ensemblistes, ainsi que de caractériser la complexité de ce problème pour ces différentes classes.

$$Sexp ::= X \mid f(Sexp, \ldots, Sexp) \mid Sexp \cup Sexp \mid Sexp \cap Sexp \mid \complement Sexp \mid \top \mid \bot$$

$$\psi_{PSC} ::= Sexp \subseteq Sexp$$

$$\text{avec } X \in \mathcal{V} \text{ et } f \in \Sigma.$$

FIG. 2.2: Syntaxe abstraite des contraintes ensemblistes positives *(PSC)*

2.3.1 Historique

Notre présentation de ces divers travaux suivra le plus possible l'ordre chronologique, qui correspond en fait également à la « taille» des classes considérées. Nous n'exhiberons pas en détail les techniques mises en œuvre dans ces différents travaux, mais n'en donnerons simplement qu'un aperçu, tout en exposant les principaux résultats.

Contraintes ensemblistes positives

En 1992, Aiken et Wimmers ont introduit une nouvelle classe de contraintes appelée *classes des contraintes positives (PSC)* [5]. Cette classe est syntaxiquement définie comme une inclusion entre deux expressions ensemblistes construites sans symbole de projection, *i.e.* des termes de $TERM(\Sigma \cup \Sigma_B \cup \Sigma_n, \mathcal{V})$. La définition de cette classe sous la forme de syntaxe abstraite est donnée à la figure 2.2.

Bien que cette classe soit disjointe syntaxiquement de la classe de contraintes définies, il a été montré qu'en fait à tout système de contraintes ensemblistes définies, on peut associer un système de contraintes ensemblistes positives[16] tel que les deux systèmes sont équivalents sémantiquement *i.e.* qu'ils possèdent les mêmes solutions montrant ainsi que l'expressivité (en terme d'ensemble des solutions) de la classe *(PSC)* est aussi grande que celle de la classe des contraintes définies (voir [16]). En réalité, la classe *(PSC)* est strictement plus expressive que la classe des contraintes définies puisque contrairement à cette dernière, la satisfiabilité d'un système de contraintes *(PSC)* n'implique pas l'existence d'une plus petite solution. Ainsi, pour le système[17]

$$X \cup Y = a \cup b$$
$$X \cap Y = \bot$$

il n'existe que deux solutions $\mathcal{I}$ et $\mathcal{I}'$, définies par $\mathcal{I}(X) = \mathcal{I}'(Y) = \{a\}$ et $\mathcal{I}(Y) = \mathcal{I}'(X) = \{b\}$. Ces deux solutions sont incomparables au sens de $\sqsubseteq$. En ce sens, on peut alors dire que la classe *(PSC)* est strictement plus expressive que la classe *(DSC)*.

Le problème abordé dans [5] est celui de la satisfiabilité d'un système de contraintes de la classe *(PSC)*. L'algorithme proposé applique une série de transformations syntaxiques sur le système, transformant un système en un (ou plusieurs[18]) autre(s) système(s) sémantiquement

[16]En fait, une sous-classe stricte des contraintes positives est suffisante (voir section 2.4.2).

[17]L'égalité $A = B$ est mise ici pour les deux inclusions $A \subseteq B$ et $B \subseteq A$.

[18]Un système peut être transformé en un ensemble de systèmes exprimant une forme de disjonction sur des hypothèses de vacuité de l'interprétation de certaines variables.

équivalent(s). Il résulte de cet algorithme, un ensemble de systèmes tel que chacun d'eux, soit contient une contrainte trivialement insatisfiable, soit est dans une forme dite résolue. Un tel système en forme résolue est un ensemble d'équations de la forme

$$\begin{cases} X_1 = Sexp_1 \\ X_2 = Sexp_2 \\ \vdots \quad\quad \vdots \\ X_n = Sexp_n \end{cases}$$

Si chacun de ces systèmes contient une contrainte trivialement insatisfiable, alors le système initial était insatisfiable. Dans le cas contraire, pour chacuns des systèmes ne contenant pas de telles contraintes insatisfiables, les variables ensemblistes n'apparaissant pas dans les membres gauches des équations sont dites libres, dans le sens où leur interprétation n'est pas fixée par le système de contraintes. Ainsi, une valuation quelconque de ces variables s'étend en une interprétation unique des variables $\{X_1, \ldots, X_n\}$, solution du système initial. D'un point de vue de la complexité du problème de satisfiabilité de la classe *(PSC)*, Aiken et Wimmers ont démontré que le problème de satisfiabilité était dans *NEXPTIME* et *EXPTIME*-dur (en encodant le problème d'inclusion de deux langages réguliers d'arbres exprimés au moyen d'automates d'arbres).

Dans le même temps, Gilleron, Tommasi et Tison [32] ont étudié le même problème pour cette même classe de contraintes *(PSC)*. L'algorithme de décision proposé dans [32] est basé sur la définition d'une nouvelle classe d'automates, les *automates d'ensembles d'arbres* et de la notion de calcul s'y rattachant. Nous allons sommairement décrire cette approche. On peut définir une interprétation ensembliste par une valuation représentant la fonction caractéristique, *i.e.* par une valuation associant à chaque arbre fini τ et chaque variable ensembliste X une valeur de vérité, vrai si pour cette interprétation τ est dans X et faux sinon. Considérons maintenant l'ensemble des sous-expressions ensemblistes apparaissant dans un système de contraintes ensemblistes : pour le système donné en exemple précédemment, $\{X, Y, X \cup Y, X \cap Y, a, b, a \cup b\}$. La valuation définie précédemment peut être étendue à cet ensemble de sous-expressions de manière cohérente avec les opérateurs ensemblistes (par exemple, si pour la valuation φ, $\varphi(X) = \varphi(Y) =$ vrai, alors on doit avoir $\varphi(X \cap Y) =$ vrai). L'autre forme de cohérence à vérifier est la cohérence vis-à-vis des symboles de fonctions : ceci est réalisé en considérant pour $f(E_1, \ldots, E_m)$, une (sous-)expression du système, les règles de la forme $f(\varphi, \ldots, \varphi) \rightarrow \varphi$ telle que $\varphi(f(E_1, \ldots, E_m))=$ vrai si et seulement si pour tout i, $\varphi(E_i) =$ vrai. Cela définit donc un automate, dont les états sont les valuations booléennes cohérentes avec les opérateurs ensemblistes et dont les règles assurent la cohérence pour les symboles de fonctions. Un calcul dans un tel automate est une application de l'ensemble des arbres vers les états de l'automate compatible avec les règles de celui-ci ; un calcul représente donc une fonction caractéristique. On peut alors dire qu'un tel automate reconnaît des n-uplets d'ensembles d'arbres. L'interprétation ne doit bien-sûr pas être quelconque puisqu'elle doit satisfaire la contrainte ensembliste. C'est pourquoi est ajouté à l'automate un ensemble d'ensembles d'états dits acceptants. Ces ensembles sont des ensembles d'états (de valuations) φ vérifiant pour toute contrainte $A \subseteq B$ du système $\varphi(A) \Rightarrow \varphi(B)$. Un calcul pour lequel l'image de l'ensemble des arbres finis est un ensemble acceptant, est dit réussi. Les auteurs ont démontré que chaque calcul réussi définit une solution du système de contraintes ensemblistes. Ainsi, le problème de satisfiabilité d'un système de contraintes ensemblistes positives se ramène au problème du vide pour un automate d'ensembles d'arbres, qui est montré décidable et *NEXPTIME*-complet. Ils ont également démontré que tout système satisfiable possédait des solutions maximale et minimale régulières, c-à-d associant à chaque variable ensembliste un ensemble régulier d'arbres.

Cette même année, Bachmair, Ganzinger et Waldmann [7] ont donné une caractérisation

$$\forall x,\ \neg p_{\perp}(x)$$
$$\forall x,\ p_{\top}(x)$$
$$\forall x,\ p_{E\cup F}(x) \Leftrightarrow p_E(x) \vee p_F(x)$$
$$\forall x,\ p_{E\cap F}(x) \Leftrightarrow p_E(x) \wedge p_F(x)$$
$$\forall x,\ p_{\complement E}(x) \Leftrightarrow \neg p_E(x)$$
$$\forall x_1, \ldots, x_n,\ p_{f(E_1,\ldots,E_n)}(f(x_1,\ldots,x_n)) \Leftrightarrow p_{E_1}(x_1) \wedge \ldots \wedge p_{E_n}(x_n)$$
$$\forall x_1, \ldots, x_n,\ \neg p_{f(E_1,\ldots,E_m)}(g(x_1,\ldots,x_n)) \qquad \textit{pour tout } g \textit{ de } \Sigma \textit{ différent de } f$$

FIG. 2.3: Axiomes sous forme de clauses de Horn codant les opérateurs ensemblistes

logique au problème de satisfiabilité d'un système de contraintes ensemblistes positives en démontrant que ce problème pouvait se ramener à un problème de satisfiabilité d'un ensemble de clauses. L'idée est d'associer à toute sous-expression ensembliste E apparaissant dans un système de contraintes ensemblistes Ψ un prédicat unaire p_E, pour lequel p_E est vrai si « x est dans E ». Ainsi, la contrainte $E \subseteq E'$ est simplement encodée comme la clause $\forall x,\ p_E(x) \Rightarrow p_{E'}(x)$. Ils définissent également un ensemble d'axiomes (sous la forme d'un ensemble de clauses de Horn) codant la nature ensembliste du problème. Par exemple, pour une expression ensembliste $E \cup F$ du système Ψ, $\forall x,\ p_{E\cup F}(x) \Leftrightarrow p_E(x) \vee p_F(x)$ code le fait que pour une solution, si un terme est dans (l'interprétation de) $E \cup F$, alors il est soit dans E, soit dans F, ou encore pour une expression $f(E_1,\ldots,E_m)$ de Ψ, $\forall x_1,\ldots,x_n,\ \neg p_{f(E_1,\ldots,E_m)}(g(x_1,\ldots,x_n))$ code le fait qu'un terme dont le symbole de tête est g ne peut être dans un ensemble défini par $f(E_1,\ldots,E_m)$. La liste complète de ces axiomes est donnée à la figure 2.3.

Exemple 5 : *Pour une signature comprenant un symbole f binaire et une constante a, le système $\Psi = \{a \subseteq Y\ ,\ f(X,Y) \subseteq X \cup Y\}$ sera codé comme suit :*

par les clauses représentant les contraintes,

$$\forall x,\ p_a(x) \Rightarrow p_Y(x)$$
$$\forall x,\ p_{f(X,Y)}(x) \Rightarrow p_{X\cup Y}(x)$$

par les axiomes,

$$p_a(a)$$
$$\forall x, y,\ \neg p_a(f(x,y))$$
$$\neg p_{f(X,Y)}(a)$$
$$\forall x, y,\ p_{f(X,Y)}(f(x,y)) \Leftrightarrow p_X(x) \wedge p_Y(y)$$
$$\forall x,\ p_{X\cup Y}(x) \Leftrightarrow p_X(x) \vee p_Y(x)$$

Utilisant le fait que si un ensemble de clauses possède un modèle, alors il possède un modèle de Herbrand, il est alors évident que cet ensemble de clauses auquel on ajoute les axiomes précédemment décrits n'est satisfiable que si le système de contraintes ensemblistes correspondant admet une solution.

Nombre de symboles [a]			complexité du
d'arité 0	d'arité 1	d'arité 2+	problème de satisfiabilité
0	0+	0+	trivial
1+	0	0	*NP*-complet
1+	1	0	*PSPACE*-complet
1+	2+	0	*EXPTIME*-complet
1+	0+	1+	*NEXPTIME*-complet

[a] $n+$ est mis pour « n ou plus »

FIG. 2.4: Complexité du problème de satisfiabilité pour la classe *(PSC)* selon la signature

Ils ont alors remarqué que la forme de ces clauses n'était pas quelconque et que chacune d'elles était équivalente à la skolémisation d'une formule monadique du premier ordre sans symbole de fonction. Par exemple, $\forall x, y,\ p_{f(X,Y)}(f(x,y)) \Leftrightarrow p_X(x) \wedge p_Y(y)$ est la skolémisation de $\forall x, y\, \exists f,\ p_{f(X,Y)}(f) \Leftrightarrow p_X(x) \wedge p_Y(y)$. Utilisant un résultat de Löwenheim [46] et d'Ackermann [1], affirmant que le problème de la satisfiabilité de la théorie monadique du premier ordre sans symbole de fonction était décidable et *NEXPTIME*-complet, ils donnèrent une nouvelle preuve de la décidabilité du problème de satisfiabilité des contraintes positives à l'aide d'un algorithme *NEXPTIME*. De plus, en démontrant que toute instance du problème de la satisfiabilité des clauses d'un fragment précis de la théorie monadique du premier ordre sans symbole de fonction, connu comme *NEXPTIME*-complet, pouvait être réduit à la satisfiabilité d'un système de contraintes ensemblistes positifs, ils en déduisirent que le problème de satisfiabilité d'un système de la classe *(PSC)* était *NEXPTIME*-complet. Ce travail comprend de plus deux extensions de cette classe *(PSC)*, à savoir une autorisation limitée de symboles de projection et de symboles dits *de diagonalisation*.

Un troisième travail a été mené en parallèle sur cette classe des contraintes *(PSC)* par Aiken, Kozen, Vardi et Wimmers [2]. Il s'appuie sur le travail précédent de deux des auteurs [5] sur cette même classe, mais propose une approche voisine de [32]. Le système de contraintes est tout d'abord mis dans une forme normale, puis un hypergraphe est construit à partir de celle-ci. Les auteurs montrent que l'existence d'une solution est conditionnée par l'existence d'un sous-hypergraphe induit clos. Ce sous-hypergraphe peut en fait être vu comme un calcul dans un automate d'ensembles d'arbres. Les auteurs ont de plus fait une étude exhaustive de la complexité de la classe *(PSC)* selon l'arité des symboles de fonctions de la signature. Ces résultats sont résumés dans à la figure 2.4.

Contraintes ensemblistes positives et négatives

L'extension sur laquelle les efforts se sont alors portés est la classe des contraintes ensemblistes positives et négatives *(PNSC)*. Elle étend la classe *(PSC)* en considérant en plus de la relation d'inclusion, la relation de non-inclusion. La classe *(PNSC)* est donc constituée de contraintes d'inclusions et de non-inclusions entre des termes de $TERM(\Sigma \cup \Sigma_B \cup \Sigma_n, \mathcal{V})$. La définition de cette classe sous la forme de syntaxe abstraite est donnée à la figure 2.5.

Pour traiter le problème de satisfiabilité de la classe *(PNSC)*, Gilleron, Tommasi et Tison [33] ont étendu la définition des ensembles d'états acceptants pour prendre en compte les relations de non-inclusions. Par exemple, si on ajoute au système de contraintes positifs Ψ, considéré à

$$Sexp ::= X \mid f(Sexp, \ldots, Sexp) \mid Sexp \cup Sexp \mid Sexp \cap Sexp \mid \complement Sexp \mid \top \mid \bot$$

$$\psi_{PNSC} ::= Sexp \subseteq Sexp \mid Sexp \not\subseteq Sexp$$

$$\text{avec } X \in \mathcal{V} \text{ et } f \in \Sigma.$$

FIG. 2.5: Syntaxe abstraite des contraintes ensemblistes positives et négatives *(PNSC)*

l'exemple 5, la contrainte $X \not\subseteq \bot$, qui énonce simplement que pour toute solution, l'interprétation de X est non vide. Ainsi, il est nécessaire de pouvoir exprimer sur un calcul réussi r qu'un état φ, vérifiant $\varphi(X) = \mathsf{vrai}$ et $\varphi(\bot) = \mathsf{faux}$, est rencontré au cours de celui-ci, c-à-d qu'il existe un certain arbre τ tel que $r(\tau) = \varphi$ impliquant que τ est élément de l'interprétation de X pour la solution correspondant à ce calcul. Les auteurs ont démontré qu'ajouter de telles conditions d'acceptance ne modifiait pas la complexité du problème de l'existence d'un calcul réussi et que, comme pour la classe *(PSC)*, les calculs réussis de l'automate peuvent être mis en correspondance avec les solutions du système. La satisfiabilité d'un système de la classe *(PNSC)* est réduite au problème du vide dans un tel automate, donnant une borne supérieure *NEXPTIME*. La question de l'expressivité a été également abordée : les auteurs ont prouvé en utilisant des arguments topologiques sur les ensembles de solutions que la classe des contraintes *(PNSC)* était strictement plus expressive que la classe *(PSC)* [72]. Cela signifie simplement qu'on peut exhiber un système de contraintes ensemblistes positives et négatives tel qu'aucun système de la classe *(PSC)* n'admette le même ensemble de solutions. Les auteurs ont également montré que pour cette classe l'existence d'une solution impliquait l'existence d'une solution régulière, *i.e.* pour laquelle à chaque variable ensembliste était associé un ensemble régulier d'arbres.

Dans des travaux menés en parallèle par Aiken, Kozen et Wimmers [3, 4], il a été proposé une autre preuve de la stricte inclusion au sens de l'expressivité de la classe *(PSC)* dans *(PNSC)*, ainsi qu'un algorithme pour le problème de satisfiabilité de cette dernière classe. Cet algorithme est basé sur l'approche des hypergraphes (proposée dans [2]) : le problème de satisfiabilité d'un système de contraintes ensemblistes a été montré équivalent au test d'une propriété particulière dans l'hypergraphe construit à partir du système de contraintes. Les auteurs démontrent alors que ce problème de test peut lui aussi se réduire dans un autre problème, appelé *problème d'accessibilité non-linéaire*, défini pour un système (calculé à partir de l'hypergraphe) d'inéquations diophantiennes. Stefansson a, par ailleurs, établi que le problème d'accessibilité non-linéaire était un problème *NP*-complet [67] et donc, que l'algorithme proposé dans [3] était dans *NEXPTIME*.

Dans la continuité des idées développées par Bachmair, Ganzinger et Waldmann, Charatonik et Pacholski [14] ont proposé une extension permettant le test de satisfiabilité d'un système de la classe *(PNSC)*. En utilisant un codage similaire à [7], un système de contraintes ensemblistes est codé sous la forme d'une formule monadique du premier ordre sans symbole de fonction, en tenant compte du fait qu'une contrainte de la forme $E \not\subseteq E'$ peut être vue comme une formule $\exists x,\ p_E(x) \wedge \neg p_{E'}(x)$. Ce test de satisfiabilité se base sur le fait que toute formule monadique satisfiable de ce type admet un modèle fini dont la taille du support est inférieure ou égale à 2^N, où N est le nombre de prédicats monadiques de la formule. A partir d'un choix non-déterministe d'un

tel modèle fini (s'il n'en existe pas, la contrainte est tout simplement insatisfiable), l'algorithme essaye de reconstruire un modèle de Herbrand de la formule. Si, pour aucun des choix possibles de modèles finis, un modèle de Herbrand est reconstructible, alors la contrainte est insatisfiable. Les auteurs ont démontré que la reconstruction était possible en un temps *EXPTIME* ce qui, couplé au choix du modèle fini, donnait une complexité *NEXPTIME* (et donc, *NEXPTIME*-complète) au problème. Tout comme dans [7], Charatonik et Pacholski ont proposé une extension (de même complexité pour le problème de satisfiabilité) de cette classe *(PNSC)* étendue par des opérateurs de diagonalisation et une utilisation limitée des opérateurs de projections.

Signalons également que Gilleron, Tommasi et Tison ont donné dans [34] des résultats de décidabilité (autre que celui de la satisfiabilité) concernant les classes *(PSC)* et *(PNSC)*; par exemple, la décidabilité de l'équivalence de deux systèmes de la classe *(PNSC)* ou la décidabilité de l'existence d'une solution finie (*i.e.* associant un ensemble fini à chaque variable ensembliste) pour un système de la classe *(PSC)*.

Contraintes ensemblistes positives et négatives avec projection

Finalement, le problème général de satisfiabilité des contraintes ensemblistes, posé dans [39], a été résolu par Charatonik et Pacholski dans [15]. Cette classe est la plus générale, puisqu'elle est définie syntaxiquement comme des contraintes d'inclusions et de non-inclusions entre deux expressions ensemblistes contenant à la fois des variables ensemblistes, des symboles de fonctions, les connecteurs booléens usuels (union, intersection et complémentation) et des symboles de projections (voir figure 2.1). La méthode proposée dans cet article revient à celle de [14], puisqu'une nouvelle fois la théorie monadique du premier ordre (sans symbole de fonction) y est utilisée, en ajoutant un axiome pour les expressions de projections, de type $f_i^{-1}(E)$:

$$\forall x_i,\, p_{f_i^{-1}(E)}(x_i) \Leftrightarrow \exists_{-x_i} p_E(f(x_1, \ldots, x_m))$$

La technique est alors similaire à [14], passant par la tentative de reconstruction d'un modèle de Herbrand à partir d'un modèle fini de l'ensemble des formules monadiques sans symbole de fonction représentant le système de contraintes.

Par ailleurs, Seynhaeve [63] a démontré en utilisant des résultats topologiques sur l'espace des solutions des contraintes de la classe *(PNSC)* [72] que l'ajout des projections à la classe *(PNSC)*, augmente strictement l'expressivité de cette dernière.

Dans la section suivante, nous allons brièvement parler d'autres travaux concernant les contraintes ensemblistes inscrits dans la lignée de ceux présentés dans cette section. Ces travaux concernent principalement, la prise en compte d'un nouvel opérateur ensembliste, appelé *opérateur de diagonalisation*, les aspects logiques et topologiques des (ensembles des solutions des) contraintes ensemblistes, ainsi que d'interprétations non-standards (*i.e.* « non-Herbrand ») des contraintes ensemblistes.

2.3.2 Diagonalisation et autres travaux

Nous allons tout d'abord faire mention d'un autre opérateur ensembliste, l'*opérateur de diagonalisation*. Cet opérateur n'est pas présent dans le travail originel de Heintze et Jaffar [39], mais se retrouve dans certains travaux que nous avons précédemment cités.

Un opérateur de diagonalisation permet de définir des termes avec une relation (limitée) d'égalité ou de diségalité entre les sous-termes de ceux-ci [19].

Un opérateur de diagonalisation (voir [7]) Δ^f_Φ est défini pour un symbole de fonction f de Σ (supposé d'arité supérieure ou égale à 1) et par une formule Φ. Cette formule sans quantification s'écrit à l'aide de disjonctions et conjonctions d'atomes de la forme $x_i = x_j$ et $x_i \neq x_j$, où x_i, x_j sont des variables du premier ordre avec $1 \leq i, j \leq \rho(f)$:

La sémantique d'un tel opérateur est définie pour un ensemble de termes clos E par :

$$\Delta^f_\Phi(E) = \{f(\tau_1, \ldots, \tau_m) \mid f(\tau_1, \ldots, \tau_m) \in E \wedge (\mathsf{T}^{=}_{\Sigma}, [x_1 \mapsto \tau_1, \ldots, x_m \mapsto \tau_m]) \models \Phi\}$$

où $\mathsf{T}^{=}_{\Sigma}$ est la structure définie sur l'algèbre des termes avec pour seul prédicat $=$, interprété comme l'identité sur cette algèbre. Dans le formalisme de la logique monadique de [7], sa définition est donnée par les deux formules suivantes :

$$p_{\Delta^f_\Phi(E)}(f(x_1, \ldots, x_m)) \Leftrightarrow p_E(f(x_1, \ldots, x_m)) \wedge \Phi$$

$$\neg p_{\Delta^f_\Phi(E)}(g(x_1, \ldots, x_l)) \quad \textit{pour } f \neq g$$

Cette définition sous la forme d'une formule monadique avec égalité implique que, comme cela est noté dans [14], l'utilisation de tels opérateurs ne peut se faire que de manière limitée. En effet, la définition telle que donnée précédemment des formules Φ peut entraîner le fait que la première des deux formules ne se trouve plus être une clause, seule garantie dans l'approche prise par Bachmair, Ganzinger et Waldmann que l'existence d'un modèle implique l'existence d'un modèle de Herbrand. Ces derniers ont proposé dans [8] une procédure de complétion pour la classe des clauses monadiques avec égalité donnant une procédure de décision pour cette extension (limitée) de la classe *(PSC)* avec des opérateurs de diagonalisation.

Toujours pour cette même classe des contraintes *(PSC)*, Seynhaeve [64] a étendu les automates d'ensembles d'arbres en autorisant le test d'egalité entre termes-frères (comme cela avait été fait pour les automates d'arbres dans [10]). L'application immédiate de ce travail est l'ajout à la classe *(PSC)* des opérateurs de diagonalisation « positifs », *i.e.* ne comportant pas d'atomes négatifs $x_i \neq x_j$.

Nous rappelons que Charatonik et Pacholski ont prouvé qu'étendre la classe *(PNSC)* avec ces opérateurs de diagonalisation ne changeait pas la complexité du problème de satisfiabilité (*NEXPTIME*-complétude) [14] ; ils affirment en outre que les techniques présentés dans l'article [15] combinées avec celles développées dans [14], permettent d'étendre le résultat de décidabilité de la classe la plus générale en présence d'opérateurs de diagonalisation.

Dans [42], Kozen propose une axiomatisation équationelle définie sur une signature Σ et les opérateurs ensemblistes de Σ_B et Σ_n, qui se révèle être complète pour ce qu'il définit comme *les algèbres d'ensembles de termes.* Parmi ces algèbres, il identifie *les algèbres d'ensembles de termes à interprétation ensembliste*, dans lesquels les connecteurs de Σ_B et Σ_n ont leurs interprétations ensemblistes usuelles et on y retrouve également l'interprétation des expressions ensemblistes (donnée page 28), appelée *interprétation standard.* Une étude des modèles de cette axiomatisation lui permet de proposer une unification des différentes approches proposées dans [32], [7] et [2].

Ces travaux ont également servi de base pour une étude topologique très précise de l'ensemble des solutions des contraintes ensemblistes de la classe *(PNSC)* [43] : Kozen y introduit les *espaces rationnels* et y étudie certaines de leurs propriétés. Ils montrent que les ensembles de solutions

[19] Des tels opérateurs peuvent se rapprocher de la sémantique ensembliste proposée dans [73, 74].

d'une contrainte, de calculs dans un automate d'ensembles d'arbres forment de tels espaces. Ceci lui permet de démontrer, comme des corollaires de son étude, différents résultats : les résultats topologiques de l'ensembles des solutions de [72], ou dans ce même travail, les propriétés de clôture des automates d'ensembles d'arbres, mais également les propriétés des hypergraphes utilisées dans [2] et [4]. Il a également démontré que d'un point de vue topologique, l'ensemble des solutions d'une contrainte ensembliste était complètement déterminé par ces solutions régulières.

Dans la continuité des deux travaux précédents, notamment basé sur l'axiomatisation de [42], Cheng et Kozen propose dans [20] un calcul des séquents (dans l'esprit de celui proposé par Gentzen pour la logique du premier ordre), qui se révèle être un système de déduction réfutationnellement complet sur l'interprétation ensembliste standard, mais incomplet de manière générale (certaines conséquences logiques du système, valides dans l'interprétation ensembliste standard, ne sont pas déductibles). Il est cependant complet si l'on considère l'ensemble des algèbres d'ensembles de termes à interprétation ensembliste.

Citons de plus quelques travaux qui sortent du cadre de « Herbrand » des contraintes ensemblistes. En effet, l'algèbre des ensembles d'arbres finis est définie à partir de l'algèbre des arbres finis (ou de manière équivalente, de l'algèbre des termes clos), qui est souvent appelée *algèbre libre*. Charatonik a proposé une procédure de décision pour la classe *(PNSC)*, lorsque l'algèbre des ensembles d'arbres est définie à partir de l'algèbre des termes clos quotientée par une relation de congruence. Cette relation est finiment décrite par une théorie équationnelle E, c-à-d un ensemble d'équations $s = t$, où $s, t \in TERM(\Sigma, \mathcal{X})$. Le problème revient à considérer, non plus des ensembles de termes clos, mais des ensembles de E-classes de congruence. Considérer un tel problème change bien sur la nature de la satisfiabilité d'un système de contraintes.

Exemple 6 : *Soient $\Sigma = \{a, f\}$ avec a une constante et f un symbole d'arité 1. Considérons la théorie équationelle $E = \{f(x) = x\}$ et la contrainte $\mathcal{SC} = \{a \subseteq f(X)\}$. Il est bien évident qu'interprétée dans $\mathsf{SC}_{\subseteq}$, cette contrainte est insatisfiable. Cependant si on considère le problème modulo la théorie équationnelle E, le système $\mathcal{SC}$, admet alors une (en fait, une infinité de) solution(s) ; en effet, toute interprétation ensembliste associant à X un ensemble non-vide est solution, puisque tous les termes clos sont E-congrus.*

Charatonik a démontré dans [12] que le problème de satisfiabilité de la classe *(PSC)* modulo une théorie équationnelle E quelconque était indécidable. Il a démontré ce résultat en particulier lorsque E code l'associativité d'un symbole ($E = \{f(x, f(y, z)) = f(f(x, y), z)\}$). L'idée de cette preuve est très simple, puisque considérer un tel symbole de fonction binaire et associatif permet très simplement l'encodage de langages algébriques de mots ; le problème du vide de l'intersection de deux de ces langages, connu pour être indécidable, peut être réduit à un problème de satisfiabilité d'un système de contraintes de la classe *(PSC)* modulo associativité. Il aborde également le cas où E exprime le fait que f est l'opérateur d'un monoïde commutatif d'élément neutre e ($E = \{f(x, e) = x,\ f(x, f(y, z)) = f(f(x, y), z),\ f(x, y) = f(y, x)\}$) ; dans ce cas, c'est le problème indécidable de l'accessibilité d'un état dans une machine à deux compteurs qui est réduit vers un problème de satisfiabilité d'un système de contraintes ensemblistes modulo ces axiomes.

Cependant, Charatonik a également démontré deux résultats de décidabilité. Le problème de satisfiabilité de la classe *(PSC)* modulo une théorie équationnelle « shallow » linéaire[20] et non-

[20]Une équation $s = t$ est dite « shallow » linéaire si s et t le sont. Un terme t est dit « *shallow* » si ses variables apparaissent à des positions « fils » de la racine. Il est dit *linéaire* si chaque variable n'apparaît qu'à une seule position dans t.

erasing[21] était *NEXPTIME*-complet ; il a aussi démontré que, pour les contraintes de la classe *(PNSC)*, le problème de satisfiabilité était décidable dans le cas où E est un ensemble d'équations « shallow » et linéaires. L'algorithme proposé pour ce dernier problème combine l'approche logique de [14] par le choix non-déterministe d'un modèle fini et l'approche des automates d'ensembles d'arbres proposée dans [32] ayant pour états les éléments du modèle fini et avec, de plus, une condition de réussite du calcul « quotientée » par la théorie équationnelle E, la complexité de ce problème restant ouverte.

Finalement, Givan, Kozen, McAllester and Witty ont étudié dans [35] les contraintes ensemblistes dite *de Tarski* (« Tarskian Set Constraints »). Par opposition aux contraintes ensemblistes de Herbrand, le domaine d'interprétation de ces contraintes est une algèbre d'ensembles définie à partir d'une algèbre arbitraire. Ceci change profondément la nature du problème, puisque par exemple le système de contraintes

$$a \subseteq X$$
$$X \subseteq a$$
$$X \subseteq b$$

est un système de Herbrand insatisfiable, mais est un système de Tarski satisfiable. En effet, en considérant l'algèbre d'ensembles construite à partir de l'algèbre minimale ayant pour domaine $\{e\}$, l'interprétation de a et b dans cette algèbre d'ensembles est le singleton $\{e\}$ et donc, l'interprétation ensembliste associant $\{e\}$ à X est (l'unique) solution du système.

Concernant les opérateurs ensemblistes, les auteurs considèrent ceux définis dans [39] à l'exception des symboles de projection, mais considèrent en plus des symboles de relation et un opérateur de récursion μ. Pour une algèbre d'ensembles $\mathcal{M}$ de domaine $\mathcal{M}_d$ et un symbole de relation R, l'expression $R(Sexp_1, \ldots, Sexp_m)$ est interprétée comme un sous-ensemble de $(\mathcal{M}_d)^m$ et l'expression $\mu X \cdot Sexp$, où X est une variable ensembliste est interprétée comme la plus petit solution (ou de manière équivalente, comme l'intersection de toutes les solutions) sur $\mathcal{M}$ de l'égalité ensembliste $X = Sexp$ pour une interprétation ensembliste fixée des variables de $Sexp$ différentes de X[22].

2.3.3 En bref...

Nous avons présenté dans cette section comment à partir de la définition des contraintes ensemblistes proposée par Heintze et Jaffar dans [39] et l'introduction de la classe (restreinte) des contraintes définies, les travaux de différents auteurs ont considéré successivement des classes de plus en plus grandes, aboutissant à la résolution du problème de satisfiabilité de la classe la plus générale. Nous avons brièvement cité d'autres travaux concernant des extensions de cette classe, soit par ajout d'opérateurs, soit par considération de sémantiques plus riches.

Ces travaux ont eu pour but d'étendre le plus possible l'expressivité des contraintes ensemblistes. Une autre démarche très naturelle, surtout dans l'optique d'une utilisation pratique de ces contraintes, est la démarche inverse, *i.e.* l'appauvrissement de la syntaxe en terme d'opérateurs [23] (éventuellement de manière différente entre les membres droit et gauche des inclusions

[21] Une équation est dite non-erasing si les ensembles des variables apparaissant dans s et t sont égaux.

[22] Pour que cette plus petite solution soit effectivement définie, une condition entre les occurrences de la variable X et les opérateurs de complémentation est nécessaire.

[23] Nous avons déjà cité un autre type d'appauvrissement, celui des restrictions sur l'arité des symboles de fonctions de la signature Σ (voir le tableau à la figure 2.4).

ou des non-inclusions) dans l'espoir d'un gain de complexité pour le problème de satisfiabilité ou d'implication. C'est dans cette vision que s'inscrit notre travail.

Nous allons dans la section suivante revenir plus précisément sur la classe des contraintes définies, et introduire la classe des contraintes co-définies. Nous allons également présenter des travaux, s'inscrivant plus dans le cadre de la programmation logique, mais qui peuvent être vus comme une extension de la classe des contraintes définies. L'ensemble de ces travaux ont été à l'initiative des nôtres.

2.4 Contraintes définies et co-définies

Comme nous l'avons mentionné précédemment, de nombreux travaux ont eu pour but la définition de classes de contraintes de complexité plus faible pour une application de celles-ci, principalement à l'analyse de programme. Ces définitions passent par une utilisation limitée des opérateurs ensemblistes et éventuellement avec une différenciation des opérateurs pouvant apparaître en partie droite ou gauche des symboles d'inclusion et de non-inclusion. En ce sens la classe des contraintes ensemblistes définies introduites par Heintze et Jaffar correspond bien à une restriction au sens où nous l'entendons. C'est la classe que nous allons détailler maintenant, car elle est l'une des plus proches de nos travaux.

On peut noter qu'il est connu que pour les contraintes ensemblistes définies comme des inclusions entre deux expressions ensemblistes atomiques, *i.e.* ne comportant pas d'opérateur ensembliste, le problème de satisfiabilité de tels systèmes a une complexité (en temps) de $O(n^3)$, où n est la taille du système de contraintes. Ce résultat est notamment suggéré par [38, 39]. Un algorithme de cette complexité est proposé dans la version complète de [50], comme extension d'un algorithme également en $O(n^3)$, mais testant la satisfiabilité de systèmes de contraintes atomiques interprétés sur les ensembles non-vides d'arbres[24].

2.4.1 Contraintes ensemblistes définies

La classe des contraintes ensemblistes définies *(DSC)* a été la première classe de contraintes pour laquelle le problème de satisfiabilité a été prouvé décidable [39]. Elle possède la particularité que tout système de contraintes définies satisfiable possède une plus petite solution au sens de $\sqsubseteq$, justifiant son nom de « définie ».

Une contrainte de cette classe est définie syntaxiquement comme une inclusion de la forme $Sexp_A \subseteq Sexp_B$ avec $Sexp_A$, une expression ensembliste ne comportant pas de symboles de complémentation, *i.e.* un terme de $TERM(\Sigma \cup \Sigma_{B^+} \cup \Sigma_n \cup \Sigma_{-1}, \mathcal{V})$ et $Sexp_B$ est une expression ensembliste atomique, *i.e.* un terme de $TERM(\Sigma, \mathcal{V})$. La définition d'une contrainte définie est donnée sous forme de syntaxe abstraite à la figure 2.6.

Exemple 7 : *Soit Ψ le système de contraintes ensemblistes définies suivant :*

$$\begin{array}{ll} a \subseteq X & W \subseteq f(a, U) \\ a \cup f(X, Y) \subseteq Y & U \subseteq a \\ f(Y, f_1^{-1}(Y)) \cap Y \subseteq W & \end{array}$$

[24] Les auteurs de cet article appellent de telles contraintes, les *contraintes INES*, pour « Inclusions over Non-Empty Sets of trees ».

$$
\begin{array}{lll}
Sexp_A & ::= & X \mid f(Sexp_A, \ldots, Sexp_A) \mid Sexp_A \cup Sexp_A \mid Sexp_A \cap Sexp_A \mid f_i^{-1}(Sexp_A) \mid \top \mid \bot \\
Sexp_B & ::= & X \mid f(Sexp_B, \ldots, Sexp_B) \\
\\
\psi_{DSC} & ::= & Sexp_A \subseteq Sexp_B \\
\\
\multicolumn{3}{c}{\text{avec } X \in \mathcal{V},\ f \in \Sigma \text{ et } f_i^{-1} \in \Sigma_{-1}.}
\end{array}
$$

FIG. 2.6: Syntaxe abstraite des contraintes ensemblistes définies *(DSC)*

Nous allons détailler l'algorithme proposé par Heintze et Jaffar pour résoudre le problème de satisfiabilité d'un système de contraintes ensemblistes définies.

La présentation de cet algorithme est faite essentiellement à l'aide d'un ensemble de transformations sur une notion étendue de grammaires d'arbres. Cependant, de par la similitude de ces grammaires avec les contraintes ensemblistes, nous utiliserons plutôt ces dernières pour décrire cet algorithme[25]. L'algorithme produit dans le cas où le système de contraintes est satisfiable, un système en forme explicite, *i.e.* où toutes les contraintes sont de la forme $Sexp \subseteq X$, où $Sexp$ est une expression ensembliste atomique. Un tel système en forme explicite peut être vu comme une grammaire d'arbres dont les règles de production correspondantes sont de la forme $X \Rightarrow Sexp$, cette grammaire reconnaissant la plus petite solution du système de contraintes.

L'algorithme prend en entrée un système de contraintes ensemblistes définies et peut alors se résumer en quatre étapes : la première est une phase dite d'aplatissement consistant à transformer le système selon la règle

$$\frac{\mathcal{SC} \cup \{Sexp \subseteq f(Sexp_1, \ldots, Sexp_m)\}}{\mathcal{SC} \cup \{f_1^{-1}(Sexp) \subseteq Sexp_1, \ldots, f_m^{-1}(Sexp) \subseteq Sexp_m\}}$$

appliquée récursivement sur les contraintes.

Le système Ψ' à l'issue de cette étape ne comprend plus que deux types de contraintes de la forme $X \subseteq a$ ou $Sexp \subseteq X$, où a est une constante, X une variable ensembliste et $Sexp$, une expression ensembliste de $TERM(\Sigma \cup \Sigma_{B+} \cup \Sigma_n \cup \Sigma_{-1}, \mathcal{V})$. Il est à noter que cette transformation ne préserve pas l'ensemble des solutions du système de contraintes. Néanmoins, on a $\Psi \models_{SC_\subseteq} \Psi'$.

Exemple 7 : (Suite I) *Pour le système Ψ donné à l'exemple 7, le système aplati correspondant Ψ' est*

$$
\begin{array}{ll}
a \subseteq X & f_1^{-1}(W) \subseteq a \\
a \cup f(X,Y) \subseteq Y & f_2^{-1}(W) \subseteq U \\
f(Y, f_1^{-1}(Y)) \cap Y \subseteq W & U \subseteq a
\end{array}
$$

La seconde phase prend en entrée l'ensemble des contraintes de Ψ' de la forme $Sexp \subseteq X$, où X est une variable ensembliste, et produit un système de contraintes sémantiquement équivalent à ce système dont les contraintes sont de la forme $Sexp' \subseteq X$, où $Sexp'$ ne contient plus de symboles d'union et les symboles de projection et d'intersection n'apparaissent qu'à la racine

[25] Cette approche a d'ailleurs été prise dans [37].

de $Sexp'$. Cette transformation utilise le fait que $(X \supseteq A \cup B) \equiv_{\mathsf{SC}_\subseteq} (X \supseteq A) \wedge (X \supseteq B)$ et qu'une (sous-)expression ensembliste peut être remplacée par une nouvelle variable pour laquelle la contrainte correspondante est ajoutée.

Exemple 7 : (Suite II) *Pour le système Ψ' calculé par la première phase, on obtient par ces transformations, le système Ψ_{in} suivant :*

$$\begin{array}{ll} a \subseteq X & f_2^{-1}(W) \subseteq U \\ a \subseteq Y & f(X,Y) \subseteq Y \\ f(Y,Z) \cap Y \subseteq W & f_1^{-1}(Y) \subseteq Z \end{array}$$

La troisième phase est le cœur de l'algorithme : elle consiste principalement à appliquer trois règles d'inférence jusqu'à obtention d'une saturation de l'ensemble des contraintes Ψ_{in} :

- La première de ces règles est très simple et correspond à une règle d'instanciation ; elle ajoute à l'ensemble des contraintes une contrainte $Sexp' \subseteq X$ s'il existe dans le système deux contraintes $Sexp \subseteq X$ et $f(Sexp_1, \ldots, Sexp_m) \subseteq Y$ telles que Y apparaît dans $Sexp$ et $Sexp'$ est issue du remplacement de Y dans $Sexp$ par $f(Sexp_1, \ldots, Sexp_m)$. Cet ajout ne se fait que si la variable remplacée Y n'apparaît pas sous un symbole de fonction de Σ. Cette condition est bien-sûr essentielle pour la terminaison.
- La seconde vise à éliminer les symboles de projections : si dans le système, il se trouve une contrainte $Sexp \subseteq X$, telle que l'expression $Sexp$ possède une sous-expression de la forme $f_i^{-1}(f(Sexp_1, \ldots, Sexp_m))$, alors est ajoutée au système la contrainte $Sexp' \subseteq X$, où $Sexp'$ est l'expression $Sexp$ dans laquelle $f_i^{-1}(f(Sexp_1, \ldots, Sexp_m))$ a été remplacée par $Sexp_i$. Cet ajout de règle est cependant conditionné par un test de non-vacuité de l'expression $f(Sexp_1, \ldots, Sexp_m)$. Ce test en cas de réponse positive assure que dans toutes les solutions du système, l'interprétation de cette expression est différente de l'ensemble vide. Ce test de non-vacuité est très simple et peut se faire de manière incrémentale en affirmant qu'une expression atomique sans variable est non-vide, et que si $S_1, \ldots, S_m$ sont non vides, alors pour tout f de Σ, $f(S_1, \ldots, S_m)$ est non vide. De plus, si $Sexp \subseteq X$ est une contrainte du système, alors $Sexp$ est non vide implique que X est non vide.
- La troisième règle a pour but d'éliminer les symboles d'intersection : si le système contient une contrainte $Sexp \subseteq X$ et que l'expression $Sexp$ contient une sous-expression de la forme $f(Sexp_1, \ldots, Sexp_m) \cap f(Sexp'_1, \ldots, Sexp'_m)$, alors on ajoute au système la contrainte $Sexp' \subseteq X$, où $Sexp'$ est égale à l'expression $Sexp$ dans laquelle l'expression $f(Z_1, \ldots, Z_m)$ (où $Z_1, \ldots, Z_m$ sont de nouvelles variables n'apparaissant pas dans le système initial) a remplacé la sous-expression $f(Sexp_1, \ldots, Sexp_m) \cap f(Sexp'_1, \ldots, Sexp'_m)$. On y ajoute de plus les contraintes $Sexp_i \cap Sexp'_i \subseteq Z_i$, pour tout i compris entre 1 et m.

  Il est à noter qu'afin d'avoir un contrôle sur la terminaison lors de l'application de cette règle, le nom des nouvelles variables ajoutées au système n'est pas quelconque. En fait, pour la règle précédemment décrite, la variable Z_i sera en fait la variable $Z_{\{Sexp_i, Sexp'_i\}}$, qui intuitivement a pour « valeur » $Sexp_i \cap Sexp'_i$. Ce mécanisme de nommage permet d'identifier $Z_{\{s_i,s'_i\}} \cap Z_{\{s'_i,s''_i\}}$ à la variable $Z_{\{s_i,s'_i,s''_i\}}$. On peut aisément vérifier que les expressions indexant les variables ne sont en fait que des (sous-)expressions ensemblistes atomiques apparaissant explicitement dans le système initial de cette phase de saturation. Il est noté en remarque dans [37] que ce mécanisme est en fait très proche de la « subset construction », utilisée pour transformer un automate ascendant d'arbres finis non-déterministe en un automate déterministe[26].

[26] Cette notion se retrouve également dans l'algorithme de satisfiabilité des contraintes avec intersection [16]

On peut noter que ces trois transformations préservent et ce, tout au long, du calcul, la forme des expressions $Sexp'$ pour lesquelles les positions des intersections et des symboles de projections ne sont jamais imbriquées dans les expressions ensemblistes.

Exemple 7 : (Suite III) *Pour le système Ψ_{in} de l'exemple précédent, la première étape (phase d'instanciation) produit le système suivant :*

$$\begin{array}{ll} a \subseteq X & f_2^{-1}(W) \subseteq U \\ a \subseteq Y & f(X,Y) \subseteq Y \\ f(Y,Z) \cap Y \subseteq W & f_1^{-1}(Y) \subseteq Z \\ f(Y,Z) \cap a \subseteq W & f_1^{-1}(a) \subseteq Z \\ f(Y,Z) \cap f(X,Y) \subseteq W & f_1^{-1}(f(X,Y)) \subseteq Z \end{array}$$

La seconde étape d'élimination des symboles de projection, compte tenu de la non-vacuité de $f(X,Y)$ (puisque du fait de $a \subseteq X$ et $a \subseteq Y$, X et Y ont des interprétations non-vide dans toute solution du système), produit le système suivant :

$$\begin{array}{ll} a \subseteq X & f_2^{-1}(W) \subseteq U \\ a \subseteq Y & f(X,Y) \subseteq Y \\ f(Y,Z) \cap Y \subseteq W & f_1^{-1}(Y) \subseteq Z \\ f(Y,Z) \cap a \subseteq W & f_1^{-1}(a) \subseteq Z \\ f(Y,Z) \cap f(X,Y) \subseteq W & f_1^{-1}(f(X,Y)) \subseteq Z \\ X \subseteq Z & \end{array}$$

La troisième étape éliminant les intersections donne le système suivant :

$$\begin{array}{ll} a \subseteq X & f_2^{-1}(W) \subseteq U \\ a \subseteq Y & f(X,Y) \subseteq Y \\ f(Y,Z) \cap Y \subseteq W & f_1^{-1}(Y) \subseteq Z \\ f(Y,Z) \cap a \subseteq W & f_1^{-1}(a) \subseteq Z \\ f(Y,Z) \cap f(X,Y) \subseteq W & f_1^{-1}(f(X,Y)) \subseteq Z \\ X \subseteq Z & f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \subseteq W \\ X \cap Y \subseteq Z_{\{X,Y\}} & Y \cap Z \subseteq Z_{\{Y,Z\}} \end{array}$$

On itère maintenant cette étape de saturation, en réappliquant la phase d'instanciation :

$$\begin{array}{lll} a \subseteq X & f_2^{-1}(W) \subseteq U & a \subseteq Z \\ a \subseteq Y & f(X,Y) \subseteq Y & f_2^{-1}(f(Z_{\{X,Y\}}, Z_{\{Y,Z\}})) \subseteq U \\ f(Y,Z) \cap Y \subseteq W & f_1^{-1}(Y) \subseteq Z & a \cap a \subseteq Z_{\{X,Y\}} \\ f(Y,Z) \cap a \subseteq W & f_1^{-1}(a) \subseteq Z & a \cap f(X,Y) \subseteq Z_{\{X,Y\}} \\ f(Y,Z) \cap f(X,Y) \subseteq W & f_1^{-1}(f(X,Y)) \subseteq Z & a \cap a \subseteq Z_{\{Y,Z\}} \\ X \subseteq Z & f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \subseteq W & f(X,Y) \cap a \subseteq Z_{\{Y,Z\}} \\ X \cap Y \subseteq Z_{\{X,Y\}} & Y \cap Z \subseteq Z_{\{Y,Z\}} & \end{array}$$

(voir la section suivante), mais également dans la forme des états des automates utilisés dans le chapitre 3, qui peuvent être vus comme fonctions caractéristiques de ces ensembles.

La phase d'élimination des symboles de projection ne modifie alors pas le système car le test de non-vacuité de $f(Z_{\{X,Y\}}, Z_{\{Y,Z\}})$ *échoue. On applique alors la troisième phase délimination des intersections :*

$a \subseteq X$	$f_2^{-1}(W) \subseteq U$	$a \subseteq Z$
$a \subseteq Y$	$f(X,Y) \subseteq Y$	$f_2^{-1}(f(Z_{\{X,Y\}}, Z_{\{Y,Z\}})) \subseteq U$
$f(Y,Z) \cap Y \subseteq W$	$f_1^{-1}(Y) \subseteq Z$	$a \cap a \subseteq Z_{\{X,Y\}}$
$f(Y,Z) \cap a \subseteq W$	$f_1^{-1}(a) \subseteq Z$	$a \cap f(X,Y) \subseteq Z_{\{X,Y\}}$
$f(Y,Z) \cap f(X,Y) \subseteq W$	$f_1^{-1}(f(X,Y)) \subseteq Z$	$a \cap a \subseteq Z_{\{Y,Z\}}$
$X \subseteq Z$	$f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \subseteq W$	$f(X,Y) \cap a \subseteq Z_{\{Y,Z\}}$
$X \cap Y \subseteq Z_{\{X,Y\}}$	$Y \cap Z \subseteq Z_{\{Y,Z\}}$	
$a \subseteq Z_{\{X,Y\}}$	$a \subseteq Z_{\{Y,Z\}}$	

La phase d'instanciation ne modifie alors pas le système de contraintes. En revanche, la phase d'élimination des symboles de projection opère une modification : le test de non-vacuité de $f(Z_{\{X,Y\}}, Z_{\{Y,Z\}})$ *réussit (du fait des contraintes* $a \subseteq Z_{\{X,Y\}}$ *et* $a \subseteq Z_{\{Y,Z\}}$*)*

$a \subseteq X$	$f_2^{-1}(W) \subseteq U$	$a \subseteq Z$
$a \subseteq Y$	$f(X,Y) \subseteq Y$	$f_2^{-1}(f(Z_{\{X,Y\}}, Z_{\{Y,Z\}})) \subseteq U$
$f(Y,Z) \cap Y \subseteq W$	$f_1^{-1}(Y) \subseteq Z$	$a \cap a \subseteq Z_{\{X,Y\}}$
$f(Y,Z) \cap a \subseteq W$	$f_1^{-1}(a) \subseteq Z$	$a \cap f(X,Y) \subseteq Z_{\{X,Y\}}$
$f(Y,Z) \cap f(X,Y) \subseteq W$	$f_1^{-1}(f(X,Y)) \subseteq Z$	$a \cap a \subseteq Z_{\{Y,Z\}}$
$X \subseteq Z$	$f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \subseteq W$	$f(X,Y) \cap a \subseteq Z_{\{Y,Z\}}$
$X \cap Y \subseteq Z_{\{X,Y\}}$	$Y \cap Z \subseteq Z_{\{Y,Z\}}$	$Z_{\{Y,Z\}} \subseteq U$
$a \subseteq Z_{\{X,Y\}}$	$a \subseteq Z_{\{Y,Z\}}$	

Une ultime itération (par application de la règle d'instanciation) donne pour résultat de cette phase le système :

$a \subseteq X$	$f_2^{-1}(W) \subseteq U$	$a \subseteq Z$
$a \subseteq Y$	$f(X,Y) \subseteq Y$	$f_2^{-1}(f(Z_{\{X,Y\}}, Z_{\{Y,Z\}})) \subseteq U$
$f(Y,Z) \cap Y \subseteq W$	$f_1^{-1}(Y) \subseteq Z$	$a \cap a \subseteq Z_{\{X,Y\}}$
$f(Y,Z) \cap a \subseteq W$	$f_1^{-1}(a) \subseteq Z$	$a \cap f(X,Y) \subseteq Z_{\{X,Y\}}$
$f(Y,Z) \cap f(X,Y) \subseteq W$	$f_1^{-1}(f(X,Y)) \subseteq Z$	$a \cap a \subseteq Z_{\{Y,Z\}}$
$X \subseteq Z$	$f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \subseteq W$	$f(X,Y) \cap a \subseteq Z_{\{Y,Z\}}$
$X \cap Y \subseteq Z_{\{X,Y\}}$	$Y \cap Z \subseteq Z_{\{Y,Z\}}$	$Z_{\{Y,Z\}} \subseteq U$
$a \subseteq Z_{\{X,Y\}}$	$a \subseteq Z_{\{Y,Z\}}$	$a \subseteq U$

La dernière phase consiste à ne conserver du système saturé de l'étape précédente, que les contraintes de la forme $X \supseteq sexp$, où $sexp$ est une expression ensembliste atomique. Il est évident que ce système peut être vu comme une grammaire d'arbres. Il est alors possible de vérifier si cette grammaire est ou non solution du système initial $\mathcal{SC}$. Si c'est le cas, alors c'est la plus petite solution de $\mathcal{SC}$; dans le cas contraire, $\mathcal{SC}$ est insatisfiable.

Exemple 7 : (Suite IV) *Pour le système saturé précédent, on obtient le système suivant qui, vu comme une grammaire. De plus, on peut vérifier que cette grammaire est bien solution du système original* Ψ *de l'exemple 7*

$a \subseteq X$	$a \subseteq Z$	$a \subseteq Y$
$f(X,Y) \subseteq Y$	$f(Z_{\{X,Y\}}, Z_{\{Y,Z\}}) \subseteq W$	$a \subseteq Z_{\{X,Y\}}$
$a \subseteq Z_{\{Y,Z\}}$	$a \subseteq U$	

$$Sexp ::= X \mid f(Sexp, \ldots, Sexp) \mid Sexp_A \cap Sexp_A$$
$$\psi_{SCI} ::= Sexp \subseteq Sexp \mid Sexp \not\subseteq Sexp$$
$$\text{avec } X \in \mathcal{V} \text{ et } f \in \Sigma.$$

FIG. 2.7: Syntaxe abstraite des contraintes ensemblistes avec intersection *(SCI)*

On peut résumer cet algorithme, ou pour être plus précis la partie centrale de celui-ci, par une saturation visant à ajouter au système des conséquences de celui-ci sous la forme de contraintes atomiques $Sexp \subseteq X$. Cette saturation est « complète » dans le sens où une fois terminée, seules les contraintes atomiques sont utiles pour le test de satisfiabilité et la représentation de la plus petite solution si solution, il y a.

Dans ce même article, il est de plus démontré que l'ensemble des solutions d'un système de contraintes ensemblistes définies Ψ est clos par intersection (ou borne inférieure, $\sqcap$), *i.e.* $(SOL(\Psi), \sqcap, min(\Psi))$ est un semi-treillis inférieur complet, où $min(\Psi)$ est la plus petite solution du système $\mathcal{SC}$. Cependant dans ce travail, il n'est pas fait mention de la complexité de l'algorithme proposé, ni de celle du problème. Ce problème de complexité a été résolu ultérieurement par Charatonik et Podelski en introduisant une nouvelle classe de contraintes, la classe des contraintes ensemblistes avec intersection.

2.4.2 Contraintes ensemblistes avec intersection

Nous allons maintenant considérer une autre classe de contraintes ensemblistes, à savoir *les contraintes ensemblistes avec intersection*. Même si *a priori* syntaxiquement incomparables avec les contraintes définies, elles ont pourtant un étroit rapport avec ces dernières.

La classe des contraintes avec intersection a été introduite par Charatonik et Podelski dans [16]. La syntaxe de cette classe est définie par des inclusions et des non-inclusions entre des expressions ensemblistes comportant des symboles de fonctions d'une signature Σ et comme unique opérateur ensembliste, l'intersection. Une contrainte avec intersection définie comme une inclusion est dite *positive*. La définition d'une contrainte ensembliste avec intersection est donnée sous forme de syntaxe abstraite à la figure 2.7.

Charatonik et Podelski démontrent dans cet article, l'équivalence entre la classe des contraintes ensemblistes avec intersection positives et la classe des contraintes définies. Pour cela, ils démontrent qu'à tout système de contraintes avec intersection positif, on peut associer un système de contraintes définies. Il suffit pour cela d'éliminer les symboles d'intersection en membre droit des inclusions. Ceci est fait simplement en remarquant que $Sexp \subseteq C[Sexp_1 \cap Sexp_2] \equiv_{\mathsf{SC}_\subseteq} Sexp \subseteq C[Sexp_1] \wedge Sexp \subseteq C[Sexp_2]$ [27]. Inversement, à tout système de contraintes définies, on peut associer un système de contraintes avec intersection positif ; tout d'abord, il est évident que l'ensemble vide et celui de tous les termes clos peuvent être définis à l'aide de

[27] La notation $C[.]$ désigne un *contexte*, c'est-à-dire une expression avec une position particulière marquée par . ; $C[Sexp]$ désigne l'expression construite à partir du contexte $C[.]$, dans laquelle le . a été remplacé par l'expression $Sexp$.

(systèmes de) contraintes avec intersection. Ainsi, avec $X_\bot \subseteq a \wedge X_\bot \subseteq b$ (avec $a \neq b$) et $\bigwedge_{f\in\Sigma} f(X_\top,\ldots,X_\top) \subseteq X_\top$, les variables ensemblistes $X_\bot$ et $X_\top$ ont pour toute solution du système, leur interprétation respective égale à celle de $\bot$ et $\top$. Il suffit alors d'éliminer les symboles d'unions et de projections en partie gauche des symboles d'inclusions. Ceci est réalisé en remarquant que $C[Sexp_1 \cup Sexp_2] \subseteq Sexp \equiv_{\mathsf{SC}_\subseteq} C[Sexp_1] \subseteq Sexp \wedge C[Sexp_2] \subseteq Sexp$ et $f_i^{-1}(Sexp_1) \subseteq Sexp_2 \equiv_{\mathsf{SC}_\subseteq} Sexp_1 \cap f(X_\top, X_\top, \ldots, X_\top) \subseteq f(X_\top,\ldots,Sexp_2,\ldots,X_\top)$, tel que $Sexp_2$ est le $i^{ème}$ argument de f dans l'expression ensembliste précédente.

Exemple 8 : *Ainsi pour le système de contraintes définies de l'exemple 7 (dans lequel l'expression $f_1^{-1}(Y)$ a été extraire au moyen de la nouvelle variable Z),*

$$\begin{array}{ll} a \subseteq X & W \subseteq f(a,U) \\ a \cup f(X,Y) \subseteq Y & U \subseteq a \\ f(Y,Z) \cap Y \subseteq W & f_1^{-1}(Y) \subseteq Z \end{array}$$

on obtient un système de contraintes avec intersection en remplaçant :

- *la contrainte $f_1^{-1}(Y) \subseteq Z$ par les contraintes (en présupposant que la signature ne comprend que deux symboles de fonctions, la constante a et le symbole binaire f)*

$$\begin{array}{c} Y \cap f(X_\top, X_\top) \subseteq f(Z, X_\top) \\ a \subseteq X_\top \\ f(X_\top, X_\top) \subseteq X_\top \end{array}$$

- *la contrainte $a \cup f(X,Y) \subseteq Y$ par les contraintes $a \subseteq Y$ et $f(X,Y) \subseteq Y$.*

Nous allons détailler l'algorithme proposé dans [16] pour tester la satisfiabilité d'un système de contraintes ensemblistes avec intersection positives.

Les contraintes considérées sont supposées dans une forme aplatie, *i.e.* construites à partir de la grammaire suivante (pour le non-terminal φ) :

$$\begin{array}{l} \tau ::= X \mid f(X_1,\ldots,X_m) \\ \theta ::= \tau \mid \tau \cap \theta \\ \varphi ::= \theta_1 \subseteq \theta_2 \end{array}$$

Une transformation très simple de tout système de contraintes ensemblistes avec intersection permet d'obtenir une telle forme sans perte de généralité. Pour le système de contraintes ensemblistes avec intersections positives de l'exemple 8, par le remplacement de la contrainte $W \subseteq f(a,U)$ par les contraintes $W \subseteq f(V,U)$ et $V \subseteq a$, on obtient un système en forme aplatie.

Ce système est alors transformé dans un autre système (non équivalent) pour lequel on retrouve un nommage de variables similaire à celui proposé par Heinze et Jaffar, décrit précédemment à la page 45. Chaque expression décrite par θ peut être définie comme l'ensemble des expressions atomiques, sous-termes (directs) d'un symbole d'intersection. Par exemple, on peut associer à $f(X,Y) \cap (Y \cap Z)$ l'ensemble $\{f(X,Y), Y, Z\}$. Le système est transformé en un système dit de représentation dans lequel chaque membre (droit et gauche) d'une contrainte du système est remplacé par une nouvelle variable indicée par l'ensemble des expressions atomiques correspondant à ce membre. Ces variables sont appelées *variables d'intersection.*

Exemple 9 : *Mettons côte-à-côte le système (à droite) et sa représentation (à gauche) :*

$a \subseteq X$	$V_{\{a\}} \subseteq V_{\{X\}}$
$a \subseteq Y$	$V_{\{a\}} \subseteq V_{\{Y\}}$
$f(X,Y) \subseteq Y$	$V_{\{f(X,Y)\}} \subseteq V_{\{Y\}}$
$Y \cap f(X_\top, X_\top) \subseteq f(Z, X_\top)$	$V_{\{Y, f(X_\top,X_\top)\}} \subseteq V_{\{f(Z,X_\top)\}}$
$a \subseteq X_\top$	$V_{\{a\}} \subseteq V_{\{X_\top\}}$
$f(X_\top, X_\top) \subseteq X_\top$	$V_{\{f(X_\top,X_\top)\}} \subseteq V_{\{X_\top\}}$
$f(Y,Z) \cap Y \subseteq W$	$V_{\{f(Y,Z),Y\}} \subseteq V_{\{W\}}$
$W \subseteq f(V,U)$	$V_{\{W\}} \subseteq V_{\{f(U,V)\}}$
$V \subseteq a$	$V_{\{V\}} \subseteq V_{\{a\}}$
$U \subseteq a$	$V_{\{U\}} \subseteq V_{\{a\}}$

Comme nous l'avons dit précédemment la représentation et le système original ne sont pas équivalents. Cependant, Charatonik et Podelski montrent qu'une forme d'équivalence peut être établie par la notion d'interprétation $\cap$-compatible. Une interprétation (définie sur les variables ensemblistes et les variables d'intersection) est dite $\cap$-compatible si pour toute variable d'intersection V_S, $\mathcal{I}(V_S) = \cap_{\tau \in S} \mathcal{I}(\tau)$. On peut alors dire que $\mathcal{I}$ est une solution $\cap$-compatible de la représentation si et seulement si elle est solution du système originel. Il y a donc équivalence entre l'existence d'une solution $\cap$-compatible du système et l'existence d'une solution $\cap$-compatible de la représentation de celui-ci. On retrouve ici l'intuition donnée dans [39] puisque les sémantiques (dans une interprétation donnée) de la variable d'intersection $V_{\{\tau_1,\tau_2\}}$ et de l'expression $\tau_1 \cap \tau_2$ coïncident.

Pour résoudre le problème de satisfiabilité, les auteurs proposent un système d'axiomes, qui est utilisé pour saturer la représentation d'un système. Cette saturation vise à tester l'existence d'une solution $\cap$-compatible de cette représentation.

Le premier axiome stipule simplement que la relation d'inclusion $\subseteq$ est réflexive et transitive. Le second axiome code d'une certaine façon l'intersection en précisant le « sens » des ensembles indiçant les variables d'intersection : $V_S \subseteq V_\tau$ pour $\tau \in S$. Les troisième et quatrième axiomes, quant-à-eux, assurent la cohérence des indices des variables par rapport à la composition fonctionnelle : $f(V_{X_1}, \ldots, V_{X_m}) \subseteq V_{\{f(X_1,\ldots,X_m)\}}$ et $V_{\{f(X_1,\ldots,X_m)\}} \subseteq f(V_{X_1}, \ldots, V_{X_m})$.

Le cinquième axiome définit la combinaison de deux contraintes dont les membres gauches partagent le même symbole de fonction de tête : $f(V_{S_1}, \ldots, V_{S_m}) \subseteq V_S \wedge f(V_{S'_1}, \ldots, V_{S'_m}) \subseteq V_{S'} \Rightarrow f(V_{S_1 \cup S'_1}, \ldots, V_{S_m \cup S'_m}) \subseteq V_{S \cup S'}$. Nous avons vu que dans [39] le test de vacuité d'expressions ensemblistes (dans toute solution) était crucial et comment il pouvait être réalisé. On le retrouve ici codé sous la forme de deux axiomes définissant un prédicat unaire *nonempty* : $nonempty(\tau) \wedge \tau \subseteq \tau' \Rightarrow nonempty(\tau')$ et $nonempty(V^1) \wedge \ldots \wedge nonempty(V^m) \Rightarrow nonempty(f(V^1, \ldots, V^m))$ [28].

Ce prédicat est utilisé dans les deux derniers axiomes : le premier permet la décomposition fonctionnelle de l'inclusion $nonempty(f(V^1, \ldots, V^m)) \wedge f(V^1, \ldots, V^m) \subseteq f(U^1, \ldots, U^m) \Rightarrow V^i \subseteq U^i$, pour tout i [29]. Le dernier axiome permet lui de détecter la non-satisfiabilité du système, produisant une inconsistance sous la forme de $\square$: $nonempty(\tau) \wedge \tau \subseteq f(V^1, \ldots, V^m) \wedge \tau \subseteq g(U^1, \ldots, U^l) \Rightarrow \square$.

Les auteurs démontrent la correction de ces axiomes en prouvant la validité de chacun d'eux

[28] Cet axiome implique en particulier $nonempty(a)$ pour toute constante a de Σ.

[29] On peut noter ici l'importance du test de non-vacuité, puisque $f(\varnothing, a) \subseteq f(a,b)$ est vrai dans $SC_\subseteq$, ce qui n'est pas le cas de la contrainte $a \subseteq b$ issue de la décomposition fonctionnelle de cette contrainte.

pour toute valution $\cap$-compatible, impliquant que toute interprétation $\cap$-compatible solution de la représentation est également solution du système saturé. La complétude vient simplement de la preuve que si $\square$ est obtenu dans le système saturé, alors la représentation, et donc le système, n'ont pas de solution ($\cap$-compatible).

De plus, si la contrainte est satisfiable, en ne conservant de l'ensemble des contraintes, résultat de la saturation par les axiomes, les contraintes de la forme $f(V^1, \ldots, V^m) \subseteq V$ et vérifiant de plus $nonempty(f(V^1, \ldots, V^m))$, tout comme dans [39], on obtient une représentation (proche d'une grammaire ou d'un automate d'arbres) de la plus petite solution du système de contraintes ensemblistes avec intersection initial.

Sur le plan de la complexité, cet algorithme est montré dans *EXPTIME*. De plus, en encodant le problème du vide d'un langage issu de l'intersection de n automates d'arbres, ce problème est démontré *EXPTIME*-dur [62]. Ceci permet de conclure que le problème de satisfiabilité d'un système de contraintes ensemblistes avec intersection positives (ou de manière équivalente, d'un système de contraintes ensemblistes définies) est *EXPTIME*-complet.

Cet article [16] comporte d'autres aspects que nous allons simplement survoler. Ces extensions nécessitent cependant de supposer que la signature fonctionnelle Σ est un ensemble infini. Sous cette hypothèse, Charatonik et Podelski démontrent que l'implication de systèmes de contraintes ensemblistes avec intersection positives est un problème *EXPTIME*-complet. Ils montrent également que cette implication, si elle est interprétée sur les ensembles non vides d'arbres finis, possède la *propriété d'indépendance*, *i.e.* pour un système φ et des contraintes $sc_1, \ldots, sc_m$, $\varphi \models sc_1 \vee \ldots \vee sc_m$ si et seulement si il existe un i tel que $\varphi \models sc_i$. On peut déduire immédiatement de ceci un algorithme de satisfiabilité pour les contraintes ensemblistes avec intersection positives (avec inclusion) et négatives (avec non-inclusion) interprétées sur les ensembles non-vides d'arbres finis. Finalement, ils étendent ce résultat à $\mathsf{SC}_{\subseteq}$ et montrent que ce problème est également *EXPTIME*-complet.

Dans l'esprit de la classe des contraintes ensemblistes définies, Charatonik et Podelski ont proposé une nouvelle classe, appelée contraintes ensemblistes co-définies. Ce titre de « co-définie » ne se justifie pas complètement par la syntaxe de la classe, mais plutôt par des propriétés duales du point de vue des solutions et par la complexité identique du problème de satisfiabilité.

2.4.3 Contraintes ensemblistes co-définies

Motivés par des considérations d'analyse de programmes [17], Charatonik et Podelski ont introduit dans [17, 18] la classe des *contraintes ensemblistes co-définies*. Pour l'analyse de programmes, le domaine principal d'interprétation de ces contraintes est l'ensemble des ensembles d'arbres finis ou infinis.

Comme mentionné dans [18], le terme « co-défini » ne signifie pas que ces contraintes sont duales (vis-à-vis de l'inclusion, *i.e.* inverser membre droit et membre gauche d'une inclusion) des contraintes définies. Une contrainte co-définie est syntaxiquement définie comme une inclusion $Sexp_A \subseteq Sexp_B$, où $Sexp_B$ est une expression ensembliste ne comportant pas de symbole de complémentation, *i.e.* un terme de $TERM(\Sigma \cup \Sigma_{B^+} \cup \Sigma_n \cup \Sigma_{-1}, \mathcal{V})$ et $Sexp_A$ est une expression construite avec des variables ensemblistes, des symboles de fonctions unaires et nullaires et des connecteurs d'unions, *i.e.* un terme de $TERM(\Sigma_0 \cup \Sigma_1 \cup \{\cup\}, \mathcal{V})$. La définition d'une contrainte ensembliste co-définie est donnée sous forme de syntaxe abstraite à la figure 2.8.

$$
\begin{array}{rcl}
Sexp_A & ::= & X \mid h(Sexp_A) \mid a \mid Sexp_A \cup Sexp_A \\
Sexp_B & ::= & X \mid f(Sexp_B, \ldots, Sexp_B) \mid Sexp_B \cup Sexp_B \mid Sexp_B \cap Sexp_B \mid f_i^{-1}(Sexp_B) \mid \top \mid \bot \\
\\
\psi_{CDSC} & ::= & Sexp_A \subseteq Sexp_B \\
\\
\multicolumn{3}{c}{\text{avec } X \in \mathcal{V},\ f \in \Sigma,\ a \in \Sigma_0,\ h \in \Sigma_1 \text{ et } f_i^{-1} \in \Sigma_{-1}.}
\end{array}
$$

FIG. 2.8: Syntaxe abstraite des contraintes ensemblistes co-définies *(CDSC)*

Exemple 10 : *Voici un exemple de système de contraintes ensemblistes co-définies :*

$$
\begin{array}{c}
a \subseteq X \\
X \subseteq a \cup f(X, Y) \\
X \subseteq a \cup f(Y, X) \\
Y \subseteq f_1^{-1}(X)
\end{array}
$$

Comme nous l'avons dit précédemment, les auteurs étaient principalement intéressés par une sémantique définie sur les ensembles d'arbres finis et infinis. Cependant, l'algorithme proposé dans [18] s'adapte très simplement sur $\mathsf{SC}_{\subseteq}$. Nous soulignerons le moment venu cette différence dans la présentation que nous allons faire de cet algorithme.

On constate donc que la non-dualité vient du fait que seuls des symboles de fonctions d'arité au plus 1 sont autorisés en membres gauches des inclusions pour les contraintes co-définies. Selon les auteurs, ce terme de « co-définies » vient plutôt du fait que symétriquement aux contraintes ensemblistes définies, tout système de contraintes co-définies satisfiable admet une plus grande solution. Comme noté dans [18], ce ne serait pas le cas si des symboles binaires (ou plus) étaient autorisés en membre gauche d'inclusion. Par exemple, $f(X, Y) \subseteq f(a, a) \cup f(b, b)$ admet deux solutions maximales, mais pas de plus grande solution.

Nous allons maintenant détailler l'algorithme proposé dans [18] permettant de résoudre le problème de satisfiabilité d'un système de contraintes ensemblistes co-définies et dans le cas d'une réponse positive de donner une représentation sous la forme d'un automate d'arbre de la plus grande solution.

Les auteurs (comme dans le cas des contraintes avec intersection) considèrent une syntaxe réduite pour les contraintes définie par la grammaire suivante :

$$
\begin{array}{rcl}
\tau & ::= & X \mid f(X_1, \ldots, X_m) \mid \tau_1 \cup \tau_2 \mid \varnothing \\
\varphi & ::= & a \subseteq X \mid X \subseteq \tau \mid X \subseteq f_k^{-1}(Y)
\end{array}
$$

On remarque que cette syntaxe réduite permet d'expliciter dans les parties droites des inclusions l'ensemble vide $\varnothing$, ce qui sera utile pour l'algorithme de test de satisfiabilité d'un système.

Un système de contraintes ensemblistes co-définies en syntaxe réduite est un ensemble (une conjonction) équivalent de telles contraintes. Il est très simple de transformer un système quelconque de contraintes co-définies en un tel système en remarquant que en ce qui concerne les membres gauches des inclusions $C[Sexp_1 \cup Sexp_2] \subseteq Sexp \equiv_{\mathsf{SC}_{\subseteq}} (C[Sexp_1] \subseteq Sexp \wedge C[Sexp_2] \subseteq$

Sexp) et que pour tout symbole de fonction h unaire, $h(Sexp) \subseteq Sexp' \equiv_{\mathsf{SC}_\subseteq} Sexp \subseteq h^{-1}(Sexp')$ et en ce concerne les membes droits, que $Sexp \subseteq C[Sexp_1 \cap Sexp_2] \equiv_{\mathsf{SC}_\subseteq} (Sexp \subseteq C[Sexp_1] \wedge Sexp \subseteq C[Sexp_2])$ et que le remplacement des occurences de $\top$ par une nouvelle variable $X_\top$ définie par la contrainte ajoutée $X_\top \subseteq \bigcup_{f\in\Sigma} f(X_\top, \ldots, X_\top)$ ne modifie pas les solutions du système. Remarquons également que $\bot$ est noté $\varnothing$ dans la syntaxe réduite. On peut noter que le système donné dans l'exemple 10 est déjà en forme réduite.

Avant de présenter l'algorithme, nous allons aborder deux notions importantes introduites par Charatonik et Podelski pour définir celui-ci et donner une représentation sous la forme d'un automate d'arbres (infinis) de la plus grande solution dans le cas où le système de contraintes ensemblistes co-définies considéré est satisfiable.

Tout d'abord mentionnons le fait que pour des contraintes $a \subseteq X$ et $X \subseteq \tau$, a et τ sont appelés respectivement borne inférieure et supérieure de la variable X.

Supposons qu'un système contienne pour une certaine variable X plusieurs contraintes définissant des bornes supérieures pour cette variable X ; par exemple, $\{X \subseteq \tau_1, \ldots, X \subseteq \tau_m\}$, on peut alors considérer la contrainte (qui n'est pas dans une forme réduite) $X \subseteq \tau_1 \cap \ldots \cap \tau_m$ équivalente à ce sous-système. Cette contrainte peut se voir alors comme la plus petite borne supérieure de la variable X. Le membre droit de cette contrainte se présente donc comme une intersection d'unions de termes de la forme $f(X_1, \ldots, X_m)$ ou $\varnothing$. Charatonik et Podelski appellent cette forme *forme normale conjonctive*. Pour le système donné en exemple 10, la plus petite borne supérieure de X en forme normale conjonctive est $(a \cup f(X,Y)) \cap (a \cap f(Y,X))$.

En distribuant les symboles d'union par rapport aux symboles d'intersection dans une forme normale conjonctive, on obtient une expression définie comme une union d'intersections de termes de la forme $f(X_1, \ldots, X_m)$ ou $\varnothing$. Cette expression est appelée *forme normale disjonctive*. En notant que $a \cap f(X,Y) = \varnothing$ et que $\varnothing$ est l'élément neutre de l'union et l'élément absorbant de l'intersection, on obtient pour l'exemple une borne supérieure de la variable X sous forme normale disjonctive exprimée comme la contrainte $X \subseteq a \cup (f(X,Y) \cap f(Y,X))$. Il peut arriver deux cas dégénérés de cette définition : dans le cas où $X \subseteq \varnothing$ est une contrainte appartenant au système, alors la borne supérieure de la variable implique que l'interprétation de X doit être l'ensemble vide dans toutes les solutions. Il peut se faire également qu'une variable Y n'ait pas de borne supérieure.

Les auteurs exposent alors comment d'un système de contraintes co-définies $\mathcal{SC}$ en ne considérant que les contraintes de la forme $X \subseteq lub$ (où cette contrainte est la seule ayant la variable X en membre gauche et *lub* désigne la borne supérieure de cette variable en forme normale disjonctive) on peut extraire un système, dit de « contraintes-automate», noté $\Psi(\mathcal{SC})$. Cette construction est assez immédiate et se base une nouvelle fois sur l'idée de « subset construction » : les variables considérées sont alors les ensembles de variables ensemblistes présentes dans le système original. Comme précédemment la variable $\{X,Y\}$ doit être comprise comme l'expression $X \cap Y$. Il est important de remarquer qu'en forme normale disjonctive, les intersections peuvent « franchir » les symboles de fonctions, *i.e.* l'expression $f(X,Y) \cap f(Y,X)$ est équivalente à $f(X \cap Y, Y \cap X)$. En utilisant cette remarque, le système de « contraintes-automate» extrait du système donné dans l'exemple 10 est :

$$\begin{aligned} \{X\} &\subseteq a \cup f(\{X,Y\},\{X,Y\}) \\ \{Y\} &\subseteq \top \\ \{X,Y\} &\subseteq a \cup f(\{X,Y\},\{X,Y\}) \end{aligned}$$

la contrainte $\{Y\} \subseteq \top$ vient du fait que la variable Y n'a pas de borne supérieure (sous la forme d'une union d'intersections d'expressions $f(X_1, \ldots, X_m)$ ou $\varnothing$) dans le système de contraintes. La

dernière contrainte s'obtient par intersection des deux précédentes, en sachant que $\top$ représente intuitivement l'ensemble de tous les arbres et qu'il est donc élément neutre de $\cap$.

L'algorithme en lui-même est alors décrit par un ensemble d'axiomes saturant le système initial : les deux premiers axiomes sont très simples. Ils affirment que l'inclusion (entre variables) est transitive ($X \subseteq Y \wedge Y \subseteq Z \Rightarrow X \subseteq Z$) et qu'on peut propager les inclusions concernant les expressions ensemblistes de la forme τ ($X \subseteq \tau_1 \cup Y \wedge Y \subseteq \tau_2 \Rightarrow X \subseteq \tau_1 \cup \tau_2$).

Les deux derniers axiomes visent eux à tester la présence d'une insatisfiabilité dans le système soit du fait d'un conflit sur la vacuité d'une variable $a \subseteq X \wedge X \subseteq \varnothing$, soit d'un fait de l'incompatibilité entre la borne inférieure et supérieure d'une variable $a \subseteq X \wedge X \subseteq \cup_i f_i(X_1^i, \ldots, X_m^i)$ et a est différent de tous les f_i. Ces deux axiomes ajoutent explicitement l'inconsistance $\square$ au système.

Les deux autres axiomes sont quant-à-eux les axiomes centraux de l'algorithme. Ils permettent d'ajouter des conséquences en prenant en compte les opérateurs de projection. Ceux-ci se basent sur la construction du système de « contraintes-automate » $\Psi(\mathcal{SC})$ à partir du système courant $\mathcal{SC}$.

Considérons une contrainte de $\mathcal{SC}$ de la forme $V \subseteq f_k^{-1}(W)$ et la borne supérieure de la variable $\{W\}$ dans $\Psi(\mathcal{SC})$. Si cette borne supérieure est égale à l'ensemble vide $\varnothing$ (*i.e.* la contrainte $\{W\} \subseteq \varnothing$ apparaît explicitement dans $\Psi(\mathcal{SC})$), alors on peut ajouter à $\mathcal{SC}$, la contrainte $V \subseteq \varnothing$. Si telle n'est pas le cas, alors il existe dans $\Psi(\mathcal{SC})$ une unique contrainte $\{W\} \subseteq \tau_1 \cup \ldots \cup \tau_l$. Parmi les τ_i, seuls ceux de la forme $f(X_1, \ldots, X_m)$, c-à-d ayant f pour symbole de tête, les X_j étant des variables « ensemble », peuvent prétendre « participer » au résultat de l'application de l'opérateur de projection f_k^{-1} à W. Pour l'une de ces expressions $f(X_1, \ldots, X_m)$, un test de vacuité des variables[30] est effectué. Si aucune de celles-ci n'est vide (pour aucun X_j, $X_j \subseteq \varnothing$ n'appartient à $\Psi(\mathcal{SC})$), alors X_k est l'un des « éléments » de la projection. Pour résumer est ajoutée au système $\mathcal{SC}$ une contrainte ayant V pour membre gauche en forme normale conjonctive (ou, de manière équivalente un ensemble de contraintes, chacun des membres droits correspondant à un des éléments de l'intersection) dont la forme normale disjonctive est l'union des variables « ensemble » X_k apparaissant dans des expressions non-vides $f(X_1, \ldots, X_m)$ de la borne supérieure de $\{W\}$ dans $\Psi(\mathcal{SC})$. Si tous les τ_i correspondent à des interprétations égales à l'ensemble vide, alors on ajoute au système $V \subseteq \varnothing$ (l'ensemble vide étant élément neutre de l'union).

En revenant à l'exemple 10, pour la contrainte $Y \subseteq f_1^{-1}(X)$, en considérant le système de « contraintes-automate » donné précédemment, on ajoute au système les contraintes $Y \subseteq X$ et $Y \subseteq Y$ correspondant à la forme normale disjonctive $\{Y\} \subseteq \{X, Y\}$.

Ainsi, pour l'exemple 10, on obtient comme système de « contraintes-automate » final

$$\begin{aligned} \{X\} &\subseteq a \cup f(\{X,Y\},\{X,Y\}) \\ \{Y\} &\subseteq a \cup f(\{X,Y\},\{X,Y\}) \\ \{X,Y\} &\subseteq a \cup f(\{X,Y\},\{X,Y\}) \end{aligned}$$

De cette contrainte, on peut très facilement extraire un automate d'arbres finis ou infinis (ou, plus précisément une famille d'automates) sans condition d'acceptance dont les états sont les variables « ensemble » et les règles de transition sont définies comme suit : pour une contrainte

[30] Le test de vacuité est la principale différence entre une interprétation sur les ensembles d'arbres finis ou sur les ensembles d'arbres finis et infinis. Lorsque seuls les ensembles finis sont considérés, devrait être ajouté au système un axiome spécifiant une sorte de test d'occurrence. Par exemple, la contrainte $X \subseteq f(X, Y)$ n'admet pas de solution pour l'interprétation sur les ensembles d'arbres finis, au contraire du cas infini.

$V \subseteq \tau_1 \cup \ldots \cup \tau_l$, on considère les règles de transition $\{V \to \tau_1, \ldots, V \to \tau_l\}$. Et donc, pour l'exemple :

$$\begin{aligned}
\{X\} &\to a \\
\{X\} &\to f(\{X,Y\},\{X,Y\}) \\
\{Y\} &\to a \\
\{Y\} &\to f(\{X,Y\},\{X,Y\}) \\
\{X,Y\} &\to a \\
\{X,Y\} &\to f(\{X,Y\},\{X,Y\})
\end{aligned}$$

L'interprétation d'une variable V du système initial dans la plus grande solution de celui-ci est alors égale au langage reconnu par l'automate ayant ces règles de transitions et $\{V\}$ pour état initial.

Le système saturé (et donc, final) contient $\square$ si et seulement le système initial était insatisfiable. Si ce n'est pas le cas, alors l'automate extrait de ce système reconnaît exactement la plus grande solution du système de contraintes co-définies donné en entrée. On peut dire qu'en fait cette méthode par saturation (comme celles présentées précédemment) a pour but d'ajouter des conséquences du système (dans la théorie $\mathsf{SC}_{\subseteq}$) visant à rendre redondantes pour le test de satisfiabilité les contraintes impliquant des symboles de projections.

Les auteurs démontrent en utilisant le problème du vide du langage de l'intersection de n automates d'arbres [62] que le problème de satisfiabilité d'un système de contraintes ensemblistes co-définies est *EXPTIME*-dur et de plus, du fait de la complexité de l'algorithme proposé, que ce problème est *EXPTIME*-complet.

Il apparaît que cet algorithme mélange d'une certaine façon des aspects syntaxiques, comme la saturation du système par l'ensemble des axiomes, et des aspects plus sémantiques comme la construction et l'utilisation des automates d'arbres, représentants explicites d'une valuation ensembliste.

Podelski et Charatonik insistent dans [18] sur le fait que bien que la complexité du test de satisfiabilité des contraintes définies et co-définies soit la même et qu'il y ait présence de propriétés respectives de plus petite/plus grande solution, les classes des contraintes définies et co-définies ne sont pas duales tant sur le plan syntaxique, que sur la méthode de résolution. Nous verrons cependant que ce jugement peut être nuancé sur la dualité à la fois de l'algorithme de résolution ou de la syntaxe dans le cas d'une extension de ces deux classes.

Dans la section suivante, nous allons sortir quelque peu du cadre que nous nous étions fixé et ce pour plusieurs raisons. Tout d'abord, le cadre des travaux qui y sont abordés n'est plus à proprement parler celui des contraintes ensemblistes (nous verrons toutefois qu'en réalité, la distinction n'apparaît que dans la présentation du problème) et peut d'une certaine façon sembler moins riche. D'un autre point de vue, ces travaux introduisent une notion de description intensionnelle d'ensembles qu'on ne trouve pas dans la vision précédemment présentée des contraintes ensemblistes, qui abordait principalement ce qui est communément appelé les *opérateurs booléens*, à savoir les opérateurs d'union, d'intersection et de complémentation (même si les projections sont considérées dans certains travaux). Cependant, il s'avère que cette extension est à l'origine de notre travail et des extensions que nous proposerons pour les classes des contraintes ensemblistes définies et co-définies, mais également à l'origine de l'article « fondateur » de Heintze et Jaffar au travers d'un précédent article des mêmes auteurs [40].

2.4.4 Opérateurs intensionnels

Nous allons exposer dans cette section un nouveau type d'opérateurs qui définit des ensembles de manière *intensionnelle*. Nous présenterons ces opérateurs dans le cadre où ils sont apparus en ce qui concerne les contraintes ensemblistes à savoir la programmation logique (pour être plus précis, le typage de programmes logiques) [31]. Cette présentation a l'avantage de se baser sur un concept en général bien connu, donnant sans doute une idée plus intuitive du propos.

Une description intensionnelle d'un ensemble peut être définie comme une expression de la forme $\{x \mid P(x)\}$, où P est une propriété devant être vérifiée par les éléments de l'ensemble ainsi défini. Par exemple, l'ensemble des entiers naturels pairs peut être décrit par $\{n \mid n \in \mathbb{N} \wedge (\exists n'\, n' \in \mathbb{N} \wedge n = 2 * n')\}$. La signification d'une telle écriture est bien-sûr très informelle si on ne fixe pas de sémantique pour la propriété P. Cette définition peut être formalisée d'un point de vue logique. On considérera ici que la propriété P est décrite par une formule du premier ordre.

Les deux travaux présentés dans cette section seront ceux de Heintze et Jaffar [37, 40], ainsi que ceux de Früwirth, Shapiro, Vardi et Yardeni [28], qui ont reformulé le premier de ceux-ci dans le cadre de la programmation logique. Pour des raisons de clarté, nous commencerons par exposer ce dernier et nous terminerons par les travaux de Heintze et Jaffar plus proches des contraintes ensemblistes.

Ensembles intensionnels et programmes logiques

Nous allons aborder le principe de « description intensionnelle» d'ensembles de termes clos (ou, de manière équivalente d'arbres finis) sous la forme de programmes logiques limités.

Considérons des programmes logiques dont les symboles de prédicats sont tous unaires et dont les clauses sont de la forme $p(t) \Leftarrow p_1(t_1), \ldots, p_m(t_m)$, où t est un terme linéaire (*i.e.* chaque variable qu'il contient n'apparaît qu'une seule fois dans celui-ci) et $t_1, \ldots, t_m$ sont des termes quelconques. De tels programmes logiques sont appelés *uniformes* dans [28] et sont essentiellement les *programmes quasi-automates* définis dans [24], en notant que sans perte de généralité par ajout éventuel de nouveaux symboles de prédicats, les clauses de ces programmes peuvent être mises sous l'une des trois formes suivantes :

$$p(f(x_1, \ldots, x_m)) \Leftarrow p_1(x_1), \ldots, p_m(x_m)$$
$$p(x) \Leftarrow p_1(t_1), \ldots, p_m(t_m)$$
$$p(c) \Leftarrow p_1(t_1), \ldots, p_m(t_m)$$

où les variables (du premier ordre) $x_1, \ldots, x_m$ de tête de la première clause sont deux à deux distinctes , $t_1, \ldots, t_m$ sont des termes quelconques, x est une variable du premier ordre et c un symbole de constante.

Nous avons déjà mentionné le fait que lorsque l'on considère des formules comprenant uniquement des prédicats monadiques ces symboles de prédicats peuvent être confondus avec des variables du second ordre, en tenant compte du fait qu'un modèle d'un tel programme logique peut être vu comme une valuation ensembliste. Ainsi, on peut réécrire un tel programme logique sous la forme d'une contrainte ensembliste intégrant un opérateur de description intensionnelle d'ensembles. On suppose qu'au prédicat p correspond la variable ensembliste X et qu'à chacun

[31] Nous présenterons plus avant dans ce document les liens existants entre les contraintes ensemblistes et l'analyse de programmes logiques.

des p_i correspond la variable X_i. Nous avons déjà rencontré le premier type de clause ; on peut lui associer la contrainte $f(X_1, \ldots, X_m) \subseteq X$.

Considérons maintenant le second type de clause et une interprétation (de Herbrand) $\mathcal{I}$. Cette dernière peut être vue comme une valuation ensembliste associant à chaque variable ensembliste (correspondant à un symbole de prédicat) un ensemble de termes clos. Soit $\{x \mid \exists_{-x} t_1 \in X_1 \wedge \ldots \wedge t_l \in X_l\}$, une description intensionnelle d'ensembles d'arbres finis définie comme une conjonction monadique du premier ordre dont x est la seule variable (éventuellement) libre. La sémantique de cette description est définie comme étant l'ensemble des arbres finis τ tels que $(\mathcal{I}, [x \mapsto \tau]) \models \exists_{-x} t_1 \in X_1 \wedge \ldots \wedge t_l \in X_l$. D'après cette sémantique et le lien entre symbole de prédicat et variable ensembliste, on peut affirmer qu'une clause du type $p(x) \Leftarrow p_1(t_1), \ldots, p_m(t_m)$ peut s'écrire sous la forme $\{x \mid \exists_{-x} t_1 \in X_1 \wedge \ldots \wedge t_l \in X_l\} \subseteq X$.

La dernière clause peut être vue comme un cas dégénéré du second type. Son codage comme une contrainte est un peu plus particulier et peut se faire comme suit : si on considère un symbole binaire f (présent ou ajouté à Σ)

$$f_1^{-1}(f(c, \{x \mid \exists_{-x} t_1 \in X_1 \wedge \ldots \wedge t_l \in X_l\})) \subseteq X$$

Ceci traduit bien le fait que c appartient à l'interprétation de X pour une solution, uniquement si l'expression $\{x \mid \exists_{-x} t_1 \in X_1 \wedge \ldots \wedge t_l \in X_l\}$ ne s'évalue pas à l'ensemble vide dans cette solution, *i.e.* que dans le modèle du programme correspondant à cette solution, il existe une instance close du corps de cette clause dont les atomes appartiennent à ce modèle. On peut noter que le symbole de projection $f_1^{-1}(...)$ peut lui-même être remplacé par l'expression $\{x_1 \mid \exists x_2\, f(x_1, x_2) \in ...\}$ qui lui est équivalente.

Il est bien évident que du fait que tout programme logique défini admette un modèle de Herbrand, les systèmes de telles contraintes sont toujours satisfiables. L'un des problèmes abordés [32] dans [28] est celui de donner une représentation (sous la forme d'un automate d'arbres finis) du plus petit modèle d'un programme uniforme (*i.e.* de la plus petite solution du système de contraintes qui lui est associé).

L'idée est de transformer un programme logique uniforme en utilisant le fait que son seul modèle d'intérêt est le plus petit. Ceci signifiant que pour ces transformations seul ce plus petit modèle est préservé. Nous allons donner l'idée de la première des transformations qui est appliquée à partir d'un exemple :

Exemple 11 : *Considérons le programme uniforme suivant :*

$p(x) \Leftarrow q(f(g(x), x))$	$s(g(x)) \Leftarrow s(x)$
$q(f(x, y)) \Leftarrow r(x), q(y)$	$r(a)$
$q(f(x, y)) \Leftarrow r(x), s(y)$	$q(a)$
$r(g(x)) \Leftarrow r(x)$	$s(a)$
$r(f(x, y)) \Leftarrow r(x), r(y)$	

Nous rappelons que les atomes présents dans le plus petit modèle d'un programme sont exactement les racines des arbres de preuve construits avec les instances closes des clauses de celui-ci. Par exemple, pour l'atome $p(a)$, on peut construire les deux arbres de preuve de la figure 2.9. Pour les atomes $p(g(a))$ et $p(f(a, a))$, on peut construire les arbres de preuve suivants donnés en figure 2.10.

En fait, pour tout terme clos τ, un arbre de preuve pour $p(\tau)$ aura, selon que le terme τ ait un f ou g pour symbole de tête, la forme décrite à la figure 2.11.

[32] L'autre problème traité dans cet article est un test d'appartenance, c-à-d tester pour un arbre τ et pour un symbole de prédicat p le fait que $p(\tau)$ soit ou non conséquence logique du programme.

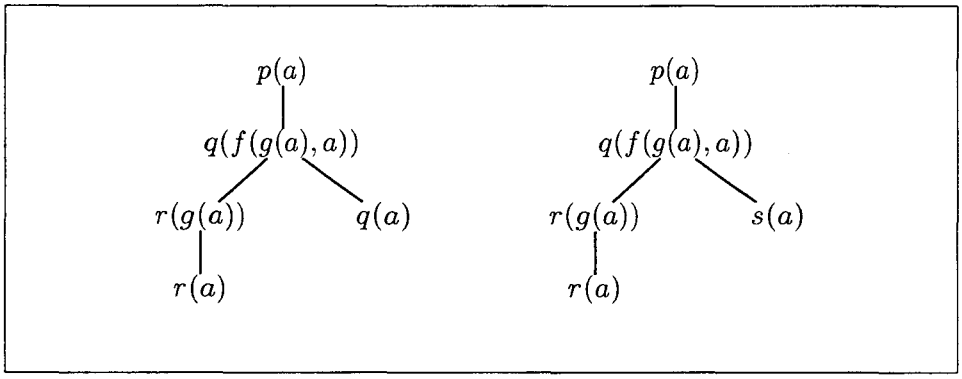

FIG. 2.9: Exemple de deux arbres de preuves pour $p(a)$ et le programme de l'exemple 11

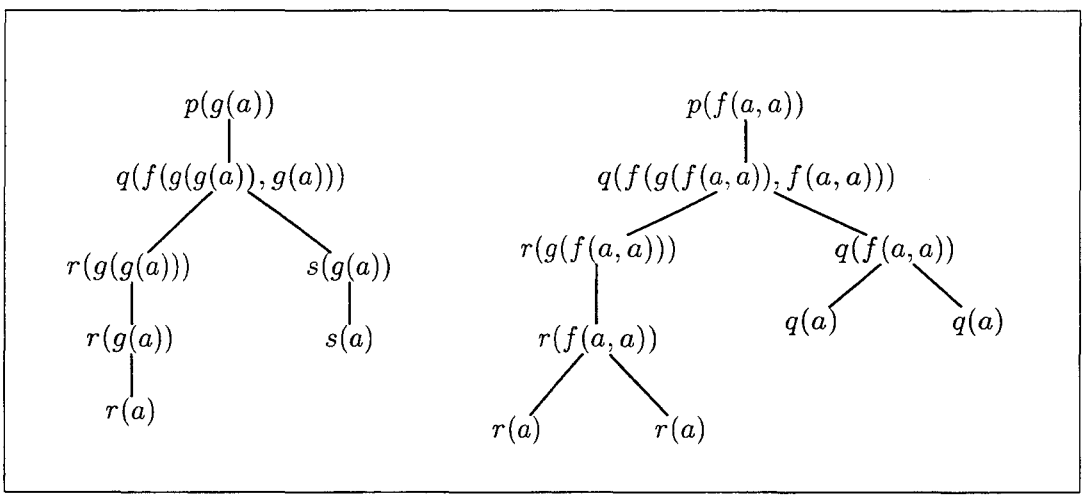

FIG. 2.10: Arbres de preuves pour $p(g(a))$ et $p(f(a,a))$ pour le programme de l'exemple 11

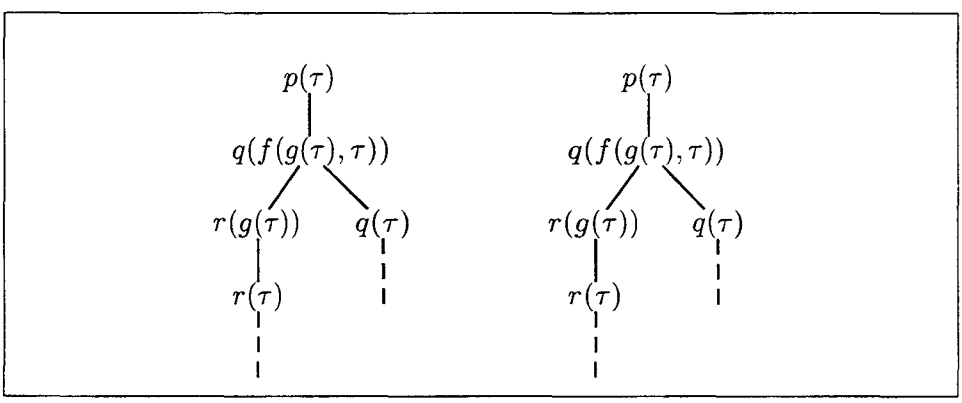

FIG. 2.11: Forme des arbres de preuves pour le programme de l'exemple 11 et $p(\tau)$ pour un terme τ de symbole de tête f (à droite) ou g (à gauche)

On peut remarquer sur la figure 2.11 que le terme τ de la racine (pour le prédicat p) se retrouve à des nœuds plus bas dans l'arbre pour les prédicats r et, s ou q selon la forme de τ. On peut d'une certaine façon dire que cette partie de l'arbre est en fait inutile, puisque le terme τ est conservé intact. Si on s'intéresse au rapport existant entre les termes de tête et ceux du corps dans les instances de clauses considérées, on voit que lorsque le terme de tête devient un sous-terme dans le corps, cette croissance doit être compensée puisque globalement les feuilles de l'arbre doivent être de taille 1. *On peut donc dans l'exemple éviter cette croissance locale en remplaçant la clause $p(x) \Leftarrow q(f(x,x)), r(f(x,y))$ par les deux clauses $p(x) \Leftarrow r(x), q(x)$ et $p(x) \Leftarrow r(x), s(x)$ éliminant l'accroissement des termes selon que la décroissance ultérieure se fasse avec la première ou la seconde clause ayant pour tête $q(f(x,y))$.*

Cette idée se base en fait sur celle de transformation d'un automate d'arbres finis « 2-way » en un automate « 1-way ». Le programme produit par une telle transformation est un programme logique dont les clauses sont de la forme suivante :

$$
\begin{array}{l}
p(f(x_1,\ldots,x_m)) \Leftarrow p_1(x_1),\ldots,p_m(x_m) \\
p(x) \Leftarrow p_1(x),\ldots,p_m(x) \\
p(x) \Leftarrow p_1(t_1),\ldots,p_m(t_m) \quad \textit{avec } x \textit{ n'apparaissant dans aucun des } t_i \\
p(c) \Leftarrow p_1(t_1),\ldots,p_m(t_m)
\end{array}
$$

Un tel programme est appelé *programme régulier-unaire*. En utilisant une idée similaire les clauses des deux derniers types de clauses peuvent être éliminées et ce programme logique est alors appelé *régulier-unaire réduit*. Les auteurs font alors remarquer qu'un programme régulier-unaire réduit se trouve être essentiellement un *automate d'arbres finis alternant* [11] que l'on peut transformer en un automate d'arbres finis, *i.e.* dans un programme logique n'ayant que des clauses de la forme $p(f(X_1,\ldots,X_m)) \Leftarrow p_1(X_1),\ldots,p_l(X_l)$.

Toutes ces transformations sont prouvées correctes dans le sens où elles préservent le plus petit modèle du programme et les auteurs démontrent que la taille du programme (*i.e.* de l'automate) produit est exponentielle par rapport à la taille du programme initial.

En reprenant le point de vue des contraintes ensemblistes, on peut affirmer que cette méthode permet le calcul (*i.e.* la représentation sous la forme d'un automate d'arbres finis de la plus petite solution d'un système de contraintes ensemblistes de la forme $Sexp \subseteq X$, où l'expression $Sexp$ est générable par la grammaire suivante :

$$
\begin{array}{rl}
Sexp ::= & X \mid f(Sexp,\ldots,Sexp) \mid Sexp \cup Sexp \mid Sexp \cap Sexp \mid \\
& f_k^{-1}(Sexp) \mid \{x \mid \exists_{-x}\, t_1 \in Sexp \wedge \ldots \wedge t_l \in Sexp\}
\end{array}
$$

Mentionnons le fait que dans [27], précédent le travail de Früwirth, Shapiro, Vardi et Yardeni [28] que nous venons de décrire, Filé a introduit une classe d'automates d'arbres, appelée *« Pattern Replacing Automata »* (« Automates de Remplacement de Motifs ») et montre l'équivalence entre ces automates et les programmes logiques définis : à tout programme logique défini, on peut associer un tel automate tel que le langage reconnu par celui-ci correspond au plus petit modèle de Herbrand du programme logique. Il étudie comme cas particulier les programmes logiques ayant uniquement des symboles de prédicats unaires et tel que les têtes de clauses de celui-ci sont de la forme $p(t)$, où t est soit une constante, soit un terme $f(x_1,\ldots,x_m)$ tel que les variables $x_1,\ldots,x_m$ sont deux-à-deux distinctes[33]. Il montre alors que les automates de remplacement de

[33] Il est immédiat de voir que tout programme logique uniforme de [28] peut être transformé un tel programme.

motifs correspondant aux programmes de ce type (appelés automates de remplacement de motifs faiblement monadiques) reconnaissent un langage régulier d'arbres.

Ces travaux de Früwirth, Shapiro, Vardi et Yardeni avaient principalement pour but de clarifier et de simplifier un résultat proposé par Heintze et Jaffar dans [40]. Heintze a lui-même proposé dans [37] une approche voisine de l'algorithme de résolution des contraintes définies précédemment présentée (voir section 2.4.1). Nous allons sommairement exposer cette algorithme dans la sous-section suivante.

Expressions ensemblistes quantifiées

Motivé par l'analyse de programmes logiques (voir chapitre 5), Heintze et Jaffar ont développé dans [40] un formalisme permettant de calculer une approximation de la sémantique d'un programme logique. C'est sur ce formalisme, appelé « formules ensemblistes » qu'ils ont ensuite proposé la définition des contraintes ensemblistes de [39]. Le travail de Früwirth, Shapiro, Vardi et Yardeni [28] que nous venons de présenter avaient essentiellement pour but de reformuler les « formules ensemblistes » dans le cadre de la programmation logique. Heintze dans [37] complète l'approche de [39] et l'intègre complètement dans les contraintes ensemblistes sous la forme d'« expressions ensemblistes quantifiées », qui sont en fait (dans le sens informel que nous avons donné) des descriptions intensionnelles d'ensembles $\{x \mid P(x)\}$.

On peut dire (en restreignant quelque peu la définition qui en est donné dans [37]) que la propriété P dans les expressions ensemblistes quantifiées est une formule du premier ordre s'écrivant comme une conjonction d'atomes implicitement existentiellement quantifiée. Les atomes de ces formules sont de la forme $t \in Sexp$ et $t\dagger Sexp$; le premier atome s'interprète comme un test d'appartenance, *i.e.* $t \in Sexp$ est vrai pour une substitution close σ et une interprétation ensembliste $\mathcal{I}$ si $\sigma(t) \in \mathcal{I}(Sexp)$. Le second $t\dagger Sexp$ est vrai pour une substitution close σ et une interprétation ensembliste $\mathcal{I}$ si il existe un terme clos τ appartenant à $\mathcal{I}(Sexp)$ différent de $\sigma(t)$.

Un terme clos τ appartient pour une interprétation ensembliste $\mathcal{I}$ à l'interprétation de l'expression $\{x \mid Atome_1 \wedge \ldots \wedge Atome_l\}$, où les $Atome_i$ sont des formules atomiques, s'il existe une substitution close σ définie pour les variables (du premier ordre) apparaissant dans les atomes et pour x telle que $\sigma(x) = \tau$ et pour tout i, $Atome_i$ est vrai pour σ et $\mathcal{I}$.

Ce second type d'atome, sortant quelque peu de l'approche que nous suivront, ne sera pas considéré dans la présentation qui suit. Nous supposerons donc que les expressions d'appartenance sont de la forme $\{x \mid \bigwedge_i t_i \in Sexp_i\}$, où les t_i sont des termes définis par Σ et un ensemble de variables du premier ordre $\mathcal{X}$ et les $Sexp_i$ sont des expressions ensemblistes.

Les contraintes que nous allons considérer ici sont en fait une extension de la classe des contraintes ensemblistes définies donnée à la section 2.4.1. Ainsi, une contrainte de cette classe peut être définie comme une inclusion de la forme $Sexp_A \subseteq Sexp_B$, où $Sexp_A$ et $Sexp_B$ sont générables par la grammaire suivante :

$$\begin{aligned} Sexp_A ::= & X \mid f(Sexp_A, \ldots, Sexp_A) \mid \top \mid \bot \mid Sexp_A \cup Sexp_A \mid Sexp_A \cap Sexp_A \mid \\ & f_k^{-1}(Sexp_A) \mid \{x \mid \bigwedge_i t_i \in Sexp_A\} \\ Sexp_B ::= & f(Sexp_B, \ldots, Sexp_B) \end{aligned}$$

Les expressions ensemblistes quantifiées sont donc simplement considérées comme une classe de nouveaux opérateurs ensemblistes pouvant apparaître en membre gauche du symbole d'inclusion.

Comme il est fait dans [37], ces expressions ensemblistes quantifiées peuvent en réalité s'écrire sous une forme restreinte ; cette forme impose simplement que les atomes de ces expressions sont soit de la forme $x \in Sexp$ (où $Sexp$ est une expression ensembliste atomique, constituée uniquement de symboles de fonctions et de variables ensemblistes) ou de la forme $t \in X$, où X est une variable ensembliste. Le système manipulé est donc toujours supposé sous cette forme.

Nous allons décrire uniquement l'ensemble des règles de transformation traitant les expressions ensemblistes quantifiées ; ces règles s'ajoutent en fait aux règles décrites pour les contraintes définies dans la section 2.4.1, étendant ainsi l'algorithme qui y est présenté pour tenir compte de cette extension de la définition des contraintes.

La première de ces règles est une règle d'instanciation des variables ensemblistes présentes dans une expression ensembliste quantifiée : si le système contient une contrainte de la forme $\{x \mid t \in Y \wedge conj\} \subseteq X$ et une autre de la forme $Sexp \subseteq Y$, où $Sexp$ est une expression atomique, alors on peut remplacer la première contrainte par $\{x \mid t \in Sexp \wedge conj\} \subseteq X$.

La seconde vise à décomposer les tests d'appartenance présents dans les expressions ensemblistes quantifiées : une contrainte de la forme $\{x \mid f(t_1, \ldots, t_m) \in f(Sexp_1, \ldots, Sexp_m) \wedge conj\} \subseteq X$ est remplacée par $\{x \mid t_1 \in Sexp_1 \wedge \ldots t_m \in Sexp_m \wedge conj\} \subseteq X$.

La troisième règle élimine les contraintes contenant une expression ensembliste quantifiée inconsistante, c-à-d dont la conjonction d'atomes ne peut être satisfaite : une contrainte de la forme $\{x \mid t \in Sexp \wedge conj\} \subseteq X$ est éliminée du système de contrainte si un test de non-vacuité échoue pour l'expression $Sexp$ ou si t est de la forme $f(t_1, \ldots, t_m)$ et $Sexp$ est de la forme $g(Sexp_1, \ldots, Sexp_l)$ avec $g \neq f$.

Enfin, la dernière règle de transformation a pour but de partiellement transformer une contrainte contenant une expression ensembliste quantifiée en une contrainte codant l'intersection de deux expressions ensembliste : si le système contient une contrainte de la forme $\{x \mid y \in Sexp \wedge y \in Sexp \wedge conj\} \subseteq X$ (x et y pouvant être la même variable), alors cette contrainte est remplacée par les deux contraintes $\{x \mid y \in Z_{\{Sexp, Sexp'\}} \wedge conj\} \subseteq X$ et $Sexp \cap Sexp' \subseteq Z_{\{Sexp, Sexp'\}}$.

Ces règles de transformation ajoutées à l'algorithme décrit dans la section 2.4.1 en constitue une extension permettant de tester la satisfiabilité d'un système de cette classe étendue et si ce dernier se révèle satisfiable donne une représentation de la plus petite solution de celui-ci.

Chapitre 3

Les contraintes définies : une extension

Nous allons dans ce chapitre présenter une extension de la classe des contraintes définies (interprétée sur les ensembles d'arbres finis, ou de manière équivalente, de termes clos), principalement par l'ajout d'un opérateur de description intensionnelle d'ensembles. Après avoir défini syntaxiquement et sémantiquement cette classe, nous proposerons pour celle-ci un algorithme répondant au problème de satisfiabilité. Nous verrons également que cette extension préserve les caractéristiques principales de la classes des contraintes ensemblistes définies. Pour conclure ce chapitre, nous mettrons en lumière les liens de cette classe avec la classe de contraintes co-définies et nous aborderons le problème de l'expressivité de cette extension.

Nous allons supposer dans ce chapitre, ainsi que dans le suivant, que la signature considérée Σ est finie.

3.1 Les contraintes définies généralisées

Nous donnons tout d'abord la syntaxe et la sémantique de l'extension proposée. Cette classe de contraintes est appelée *classe des contraintes définies généralisées* [34].

3.1.1 Syntaxe et sémantique

L'extension de la classe de contraintes ensemblistes définies que nous considérons consiste en l'ajout d'un opérateur intensionnel, appelé expression d'appartenance. Ces expressions sont décrites par des formules positives du premier ordre dont le seul symbole de prédicat est le prédicat d'appartenance « $\in$ ».

La syntaxe d'une contrainte définie généralisée ψ_{GDSC} est donnée à la figure 3.1 sous forme abstraite. La formule Φ est appelée *formule du premier ordre monadique positive* [35] *étendue.* On peut remarquer que la forme de ces formules est celle des formules monadiques positives du premier ordre, à l'exception du fait que les formules atomiques sont des tests d'appartenance non pas à une variable du second ordre, mais à une expression ensembliste. Ceci justifie l'appellation *étendue.* La notation $\Phi(x)$ marque le fait que x est l'unique variable libre de Φ. L'expression ensembliste $\{x \mid \Phi(x)\}$ est appelée *expression d'appartenance.*

[34] Ces travaux sont publiés dans [69], et sont une extension d'une partie de ceux présentés dans [25].

[35] Nous rappelons qu'une formule du premier ordre est dite *positive* si elle ne comporte pas de connecteurs de négation ($\neg$), d'implication ($\Rightarrow$) et d'équivalence ($\Leftrightarrow$).

[36] Cette notion de constante complémentée est apparue dans [37], mais pour une motivation différente de la nôtre (voir section 3.3.2).

$$
\begin{aligned}
Sexp_A &::= X \mid f(Sexp_A, \ldots, Sexp_A) \mid Sexp_A \cup Sexp_A \mid Sexp_A \cap Sexp_A \mid \top \mid \bot \mid \\
&\quad f_i^{-1}(Sexp_A) \mid \{x \mid \Phi(x)\} \\
\Phi &::= t \in Sexp_A \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists y \Phi \mid \forall y \Phi \\
Sexp_B &::= X \mid f(Sexp_B, \ldots, Sexp_B) \mid a^{\complement} \\
\psi_{GDSC} &::= Sexp_A \subseteq Sexp_B
\end{aligned}
$$

avec $X \in \mathcal{V}$, $f \in \Sigma$, $t \in TERM(\Sigma, \mathcal{X})$ et $a^{\complement}$ est un symbole nullaire, appelé *constante complémentée* et introduit pour chaque constante a de la signature [36].

FIG. 3.1: Syntaxe abstraite des contraintes ensemblistes définies généralisées

Exemple 12 : *Voici un exemple de système de contraintes définies généralisées construit sur l'ensemble de variables ensemblistes* $\{X_1, X_2, X_3\}$ *et la signature* $\Sigma = \{a/0, b/0, c/0, f/2, g/2\}$,

$$
\begin{gathered}
f(a,b) \cup f(b,\top) \subseteq X_1 \\
g(b,b) \cup g(b,X_2) \subseteq X_2 \\
\{x \mid \exists z \forall y\, f(x,y) \in X_1 \wedge g(z,x) \in X_2\} \subseteq X_3
\end{gathered}
$$

Pour fixer la sémantique de telles contraintes ensemblistes, nous rappelons qu'une interprétation ensembliste $\mathcal{I}$ est une application d'un ensemble de variables ensemblistes vers $\wp(TERM(\Sigma))$, l'ensemble des ensembles des termes clos. Cette interprétation est étendue aux expressions ensemblistes de la manière suivante :

- $\mathcal{I}(f(Sexp_1, \ldots, Sexp_m)) = \{f(\tau_1, \ldots, \tau_m) \mid \tau_1 \in \mathcal{I}(Sexp_1)$ *et* $\ldots$ *et* $\tau_m \in \mathcal{I}(Sexp_m)\}$ [37]
- $\mathcal{I}(Sexp_1 \cup Sexp_2) = \mathcal{I}(Sexp_1) \cup \mathcal{I}(Sexp_2)$
- $\mathcal{I}(Sexp_1 \cap Sexp_2) = \mathcal{I}(Sexp_1) \cap \mathcal{I}(Sexp_2)$
- $\mathcal{I}(\top) = TERM(\Sigma)$
- $\mathcal{I}(\bot) = \varnothing$
- $\mathcal{I}(f_k^{-1}(Sexp)) = \{\tau_k \mid f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(Sexp)\}$

Afin de prendre en compte les nouveaux opérateurs ensemblistes que nous avons introduits, s'ajoutent aux axiomes précédents les deux axiomes suivants :

- $\mathcal{I}(a^{\complement}) = TERM(\Sigma) \setminus \{a\}$
- $\mathcal{I}(\{x|\Phi(x)\}) = \{\tau \mid (\langle\!\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\!\rangle, [x \mapsto \tau]) \models \Phi(x)\}$ où $[x \mapsto \tau]$ est une valuation du singleton $\{x\}$ dans $TERM(\Sigma)$.

La notation $(\langle\!\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\!\rangle, [x \mapsto \tau]) \models \Phi(x)$ est en réalité un abus de celle donnée dans la section 1.2.2.0. Pour être plus correct, il conviendrait de définir l'interprétation d'une formule atomique

[37] Cette sémantique est en fait fixée par l'algèbre des ensembles d'arbres finis PTr_Σ.

$t \in Sexp$ sous une valuation ν (pour les variables du premier ordre de t) par la valeur de vérité de $\lfloor t \rfloor_\nu^{\mathcal{I}\Sigma} \in \mathcal{I}(Sexp)$, où $\mathcal{I}$ est une interprétation des variables du second ordre (donc, une interprétation ensembliste) étendue aux expressions ensemblistes par les axiomes précédents ; cette interprétation des formules atomiques s'étend aux formules monadiques positives étendues du premier ordre comme ce qui est présenté à la page 12.

Dans l'algorithme que nous proposerons pour répondre au problème de satisfiabilité d'un système de contraintes ensemblistes définies généralisées, nous allons imposer une restriction sur la forme syntaxique de ce système. Cette restriction n'entraîne cependant aucune perte de généralité dans le sens où tout système peut être effectivement transformé dans un système à syntaxe restreinte de telle manière que ces deux systèmes ont le même ensemble de solutions, du moins en ce qui concerne les variables apparaissant dans le premier de ceux-ci.

Cette restriction syntaxique peut être obtenue à partir d'un système de contraintes définies généralisées en lui appliquant les règles d'inférence du système $\mathcal{S}$ données à la figure 3.2.

Ce système $\mathcal{S}$ se compose de deux sous-systèmes $\mathcal{S}_1$ et $\mathcal{S}_2$. On applique tout d'abord sur le système de contraintes $\mathcal{SC}$ donné en entrée les règles du sous-système $\mathcal{S}_1$ (jusqu'à obtenir un système sur lequel plus aucune règle de $\mathcal{S}_1$ n'est applicable) : en introduisant de nouvelles variables, le sous-système $\mathcal{S}_1$ « aplatit » le système de contraintes de telle façon que les opérateurs ensemblistes (ainsi que les atomes des expressions d'appartenance) du système ne s'appliquent plus qu'à des variables ensemblistes. De plus, dans le système final, pour chaque contrainte, si l'un des membres de l'inclusion n'est pas une variable ensembliste, alors l'autre membre de la contrainte en est nécessairement une.

Il est aisé de voir que le sous-système $\mathcal{S}_1$ transforme un système de contraintes en un système équivalent (tout du moins en ce qui concerne les variables ensemblistes originellement dans le système de contraintes). De plus, les contraintes du système de contraintes produit sont alors de l'une des formes suivantes :

$$\begin{array}{ccccc} f(X_1,\ldots,X_m) \subseteq X & \{x \mid \varphi(x)\} \subseteq X & Y \subseteq X & X \cup Y \subseteq Z & \top \subseteq X \\ X \subseteq f(X_1,\ldots,X_m) & X \subseteq a^{\complement} & f_k^{-1}(Y) \subseteq X & X \cap Y \subseteq Z & \bot \subseteq X \end{array}$$

où φ est une formule monadique du premier ordre ayant x pour unique variable libre, *i.e.* φ ne comprend que des atomes de la forme $t \in X$ avec X, une variable ensembliste.

Sur le système ainsi obtenu à l'issu de la normalisation par le sous-système $\mathcal{S}_1$, on applique les règles d'inférence du sous-système $\mathcal{S}_2$ jusqu'à ce qu'aucune de ses règles ne soit plus applicable. Le système final obtenu est dit en *forme aplatie.*

Il est évident que le sous-système $\mathcal{S}_2$ transforme un système de contraintes en un autre système qui lui est équivalent. Ainsi, le système en forme aplatie est équivalent au système de contraintes ensemblistes initial sur lequel on a appliqué les sous-systèmes $\mathcal{S}_1$ et $\mathcal{S}_2$.

De par le forme des contraintes issu de la normalisation par $\mathcal{S}_1$, il est alors immédiat de voir qu'un système de contraintes ensemblistes définies généralisées en forme aplatie ne contient que des contraintes de la forme :

$$\begin{array}{cc} f(X_1,\ldots,X_m) \subseteq X & \{x \mid \varphi(x)\} \subseteq X \\ X \subseteq f(X_1,\ldots,X_m) & X \subseteq a^{\complement} \end{array}$$

Il est à noter que la mise en forme aplatie d'un système (au travers de la normalisation par le sous-système $\mathcal{S}_1$, puis $\mathcal{S}_2$) ne fait qu'augmenter polynômialement (par rapport à la taille du système initial) la taille de celui-ci.

$$\frac{\mathcal{SC} \cup \{Sexp \subseteq Sexp'\}}{\mathcal{SC} \cup \{Sexp \subseteq W \ , \ W \subseteq Sexp'\}}$$

$$\frac{\mathcal{SC} \cup \{f(Sexp_1, \ldots, Sexp_m) \subseteq X\}}{\mathcal{SC} \cup \{f(W_1, \ldots, W_m) \subseteq X \ , \ Sexp_1 \subseteq W_1 \ , \ \ldots \ , \ Sexp_m \subseteq W_m\}}$$

$$\frac{\mathcal{SC} \cup \{X \subseteq f(Sexp_1, \ldots, Sexp_m)\}}{\mathcal{SC} \cup \{X \subseteq f(W_1, \ldots, W_m) \ , \ W_1 \subseteq Sexp_1 \ , \ \ldots \ , \ W_m \subseteq Sexp_m\}}$$

$$\frac{\mathcal{SC} \cup \{ \{x \mid \varphi[t \in Sexp]\} \subseteq X\}}{\mathcal{SC} \cup \{ \{x \mid \varphi[t \in W]\} \subseteq X \ , \ Sexp \subseteq W\}} \qquad \frac{\mathcal{SC} \cup \{ f_i^{-1}(Sexp) \subseteq X \ \}}{\mathcal{SC} \cup \{ f_i^{-1}(W) \subseteq X \ , \ Sexp \subseteq W \ \}}$$

$$\frac{\mathcal{SC} \cup \{Sexp \cup Sexp' \subseteq X\}}{\mathcal{SC} \cup \{Sexp \subseteq X \ , \ Sexp' \subseteq X\}} \qquad \frac{\mathcal{SC} \cup \{Sexp \cap Sexp' \subseteq X\}}{\mathcal{SC} \cup \{W_1 \cap W_2 \subseteq X \ , \ Sexp \subseteq W_1 \ , \ Sexp' \subseteq W_2\}}$$

où :

- $W, W_1, \ldots, W_m$ sont des variables nouvelles (et ce à chaque application d'une règle d'inférence), *i.e.* n'apparaissant pas dans le système de contraintes sur lequel la règle est appliquée,
- $Sexp, Sexp_1, \ldots, Sexp_m$ ne sont pas des expressions ensemblistes égales à des variables de second ordre.

Le sous-système d'inférence $\mathcal{S}_1$

$$\frac{\mathcal{SC} \cup \{X \cup Y \subseteq Z\}}{\mathcal{SC} \cup \{X \subseteq Z \, , \, Y \subseteq Z\}} \qquad \frac{\mathcal{SC} \cup \{X \cap Y \subseteq Z\}}{\mathcal{SC} \cup \{\{x \mid x \in X \wedge x \in Y\} \subseteq Z\}}$$

$$\frac{\mathcal{SC} \cup \{\bot \subseteq X\}}{\mathcal{SC}} \qquad \frac{\mathcal{SC} \cup \{\top \subseteq X\}}{\mathcal{SC} \cup \bigcup_{f \in \Sigma} \{f(X, \ldots, X) \subseteq X\}}$$

$$\frac{\mathcal{SC} \cup \{Y \subseteq X\}}{\mathcal{SC} \cup \{ \{y \mid y \in Y\} \subseteq X\}} \qquad \frac{\mathcal{SC} \cup \{f_k^{-1}(Y) \subseteq X\}}{\mathcal{SC} \cup \{ \{x_k \mid \exists_{-x_k} f(x_1, \ldots, x_m) \in Y\} \subseteq X\}\}}$$

Le sous-système d'inférence $\mathcal{S}_2$

FIG. 3.2: Le système d'inférence $\mathcal{S}$ de mise en forme aplatie

Exemple 13 : *Voici le système donné dans l'exemple 12 mis en forme aplatie,*

$$\begin{array}{c} a \subseteq X_4 \\ b \subseteq X_5 \\ f(X_4, X_5) \subseteq X_1 \\ f(X_5, X_7) \subseteq X_1 \end{array} \qquad \begin{array}{c} g(X_5, X_5) \subseteq X_2 \\ g(X_5, X_2) \subseteq X_2 \\ \{x \mid \exists z \forall y\, f(x,y) \in X_1 \wedge g(z,x) \in X_2\} \subseteq X_3 \\ \{x \mid \exists y,\, y \in X_4 \vee x \in X_7\} \subseteq X_7 \end{array}$$

La variable X_7 est mise ici pour l'opérateur $\top$. Les expressions d'appartenance permettent ici d'éviter d'utiliser une description extensionnelle de l'ensemble des termes clos. En effet, la contrainte $a \subseteq X_4$ assure que pour toute solution du système, l'interprétation de la variable X_4 est non-vide. Ainsi, pour toute solution du système, l'interprétation de X_7 est l'ensemble de tous les termes clos, puisque pour tout terme clos τ et toute solution $\mathcal{I}$, $(\langle\!\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\!\rangle, [x \mapsto \tau]) \models \exists y,\, y \in X_4 \vee x \in X_7$.

3.1.2 Outils et notions auxiliaires

Dans cette section, nous allons introduire quelques notions particulières de logique qui viendront compléter celles données dans la section 1.2. Puis, nous définirons une légère extension à la définition des automates ascendants d'arbres finis donnée à la section 1.3.2.

Notions auxiliaires de logique

Nous allons introduire des compléments de logique, ayant pour objets principaux les formules monadiques positives du premier ordre et des variantes de celles-ci.

Nous allons commencer par prouver une propriété intéressante des formules monadiques positives du premier ordre. Considérons A et B, deux Σ-algèbres et Λ un morphisme surjectif de A dans B. Soient $\mathcal{I}_\mathsf{A}$ et $\mathcal{I}_\mathsf{B}$, deux applications de l'ensemble des variables du second ordre $\mathcal{V}$ vers $\wp(D_\mathsf{A})$ et $\wp(D_\mathsf{B})$, les ensembles des ensembles d'éléments du support respectivement de A et de B.

On a alors :

Proposition 5 Les deux énoncés suivants sont équivalents :

- pour tout élément d^A du support de l'algèbre A, $d^\mathsf{A} \in \mathcal{I}_\mathsf{A}(X_i) \Longleftrightarrow \Lambda(d^\mathsf{A}) \in \mathcal{I}_\mathsf{B}(X_i)$
- pour toute formule monadique positive du premier ordre φ et toute valuation ν_A de $Var_\mathcal{X}(\varphi)$ dans D_A,

$$(\langle\!\langle \mathsf{A}, \mathcal{I}_\mathsf{A} \rangle\!\rangle, \nu_\mathsf{A}) \models \varphi \Longleftrightarrow (\langle\!\langle \mathsf{B}, \mathcal{I}_\mathsf{B} \rangle\!\rangle, \Lambda(\nu_\mathsf{A})) \models \varphi$$

Preuve :

Il est évident que le second énoncé implique le premier, puisque celui-ci en est un cas particulier pour une formule atomique de la forme $x \in X_i$. L'implication réciproque est immédiate et se prouve par induction sur la forme de la formule φ.

■

Nous avons vu que les formules du premier ordre monadiques positives (étendues) sont les formules permettant de définir l'extension proposée de la classe des contraintes définies. Cependant, nous allons voir que cette classe se révélera insuffisante pour l'algorithme permettant de tester

la satisfiabilité d'un système de cette classe. C'est pourquoi nous allons introduire un nouveau type de formule et nous verrons comment ce type peut être lié du point de vue de ses modèles aux formules monadiques positives.

Définition 13 Les formules monadiques $\top$-bornées sont les formules monadiques du premier ordre définies par la grammaire suivante :

$$\chi ::= t \in X \mid \chi_1 \wedge \chi_2 \mid \chi_1 \vee \chi_2 \mid \exists x(x \in \top \wedge \chi) \mid \forall x(x \in \top \Rightarrow \chi)$$

où $\top$ est une variable du second ordre particulière.

On peut associer à chaque formule monadique positive φ une unique formule monadique $\top$-bornée,

Définition 14 Soit φ un formule monadique positive du premier ordre, on notera $\tilde{\varphi}$ la formule monadique $\top$-bornée (associée à φ) définie récursivement comme suit :

$$\tilde{\varphi} = \begin{cases} t \in X & si \ \varphi = t \in X \\ \tilde{\varphi_1} \wedge \tilde{\varphi_2} & si \ \varphi = \varphi_1 \wedge \varphi_2 \\ \tilde{\varphi_1} \vee \tilde{\varphi_2} & si \ \varphi = \varphi_1 \vee \varphi_2 \\ \exists x\,(x \in \top \wedge \tilde{\varphi'}) & si \ \varphi = \exists x \varphi' \\ \forall x\,(x \in \top \Rightarrow \tilde{\varphi'}) & si \ \varphi = \forall x \varphi' \end{cases}$$

On supposera dorénavant que pour φ une formule monadique positive, $\tilde{\varphi}$ désignera la formule monadique $\top$-bornée qui lui est associée selon la définition 14. Nous allons voir maintenant le lien sémantique existant entre une formule monadique positive et la formule monadique $\top$-bornée qui lui est associée.

Toute structure R, modèle d'une formule monadique positive φ peut être étendue en une structure $\tilde{\mathsf{R}}$, fixant la sémantique de la variable du second ordre particulière $\top$. Considérons une structure $\langle\langle \mathsf{A}, \mathcal{I} \rangle\rangle$ définie pour un ensemble $V_{\mathcal{V}}$ de variables du second ordre et définissons une extension $\langle\langle \mathsf{A}, \tilde{\mathcal{I}} \rangle\rangle$ de celui-ci à l'ensemble $V_{\mathcal{V}} \cup \{\top\}$ en posant $\forall X \in V_{\mathcal{V}}$, $\tilde{\mathcal{I}}(X) = \mathcal{I}(X)$ et $\tilde{\mathcal{I}}(\top) = D_{\mathsf{A}}$, le support de l'algèbre A. On a alors

Proposition 6 Pour toute valuation ν, $(\langle\langle \mathsf{A}, \mathcal{I} \rangle\rangle, \nu) \models \varphi$ *ssi* $(\langle\langle \mathsf{A}, \tilde{\mathcal{I}} \rangle\rangle, \nu) \models \tilde{\varphi}$

Considérons maintenant deux algèbres A et B, telles que B est une sous-algèbre de A. Soit $\mathcal{I}$, une application associant à chaque variable du second ordre de $\mathcal{V}$ un élément de $\wp(D_{\mathsf{B}})$, *i.e.* un sous-ensemble du support de l'algèbre B et à la variable particulière $\top$ l'ensemble D_{B}, support de B. On a alors

Proposition 7 Pour toute formule monadique $\top$-bornée $\tilde{\varphi}$ et toute valuation ν de l'ensemble $Var_{\mathcal{X}}(\tilde{\varphi})$ dans D_{B}, $(\langle\langle \mathsf{B}, \mathcal{I} \rangle\rangle, \nu) \models \tilde{\varphi}$ si et seulement si $(\langle\langle \mathsf{A}, \mathcal{I} \rangle\rangle, \nu) \models \tilde{\varphi}$.

Automates ascendants d'arbres finis à rang

Nous allons étendre la définition proposée dans la section 1.3.2 en ajoutant aux automates la notion de rang.

Définition 15 Un *automate ascendant d'arbres finis de rang n* est défini par un quadruplet $(\Sigma, \mathcal{Q}, F, \Delta)$. Σ est une signature, $\mathcal{Q}$ est un ensemble fini d'états, $F = (F_1, \ldots, F_n)$ est un n-uplet d'ensembles d'états dit finals ($F_i \subseteq \mathcal{Q}$) et Δ est une relation définie par un ensemble de règles de transition.

La notion de calcul pour un arbre dans un tel automate, ainsi que celle de l'accessibilité sont les mêmes que pour celles d'un automate « classique » (voir page 17).

Pour un automate $\mathcal{A}$ ascendant d'arbres finis de rang n, on dira que l'arbre τ est reconnu pour la $i^{ème}$ composante du langage $L(\mathcal{A})$ si $\tau \rightarrow_{\mathcal{A}}^{*} q$ et $q \in F_i$. Le langage reconnu par un automate ascendant d'arbres finis de rang n est un n-uplet d'ensembles d'arbres finis $L(\mathcal{A}) = (L_1, \ldots, L_n)$ tel que l'arbre τ appartient à L_i si et seulement si τ est reconnu par la $i^{ème}$ composante du langage $L(\mathcal{A})$. Un tel automate de rang n peut donc être vu comme n automates ascendants d'arbres finis ayant les mêmes états, le même ensemble de règles de transition et ne variant que sur l'ensemble des états finals, *i.e.* le $i^{ème}$ automate a pour ensemble d'états finals F_i, la $i^{ème}$ composante du n-uplets d'états finals de l'automate de rang n.

Nous allons en fait nous intéresser uniquement aux automates ascendants d'arbres finis à rang déterministes et complets, *i.e.* tels que pour tout membre gauche de règle de transition constructible, il existe une unique règle ayant ce membre gauche dans l'automate. La définition du calcul étant la même que pour un automate classique, on a donc le fait que dans un automate à rang déterministe et complet $\mathcal{A}$, pour tout arbre fini τ, il existe un unique état q tel que $\tau \rightarrow_{\mathcal{A}}^{*} q$ et cet état est noté $run_{\mathcal{A}}(\tau)$.

Automates et logique

Dans cette section, nous allons expliciter les liens existants entre logique et automates à rang déterministes et complets.

Un automate ascendant d'arbres finis déterministe et complet $\mathcal{A} = (\Sigma, \mathcal{Q}, \Delta, \mathcal{F})$ définit deux algèbres $\mathsf{A}_{\mathcal{A}}$ et $\mathsf{A}_{rch(\mathcal{A})}$ comme suit :

- $\mathsf{A}_{\mathcal{A}}=\langle \mathcal{Q}, \{f^{\mathcal{A}}\}_{f\in\Sigma}\rangle$ où $f^{\mathcal{A}}(q_1, \ldots, q_m) = q$ ssi $f(q_1, \ldots, q_m) \rightarrow q \in \Delta$
- $\mathsf{A}_{rch(\mathcal{A})}=\langle Reachable(\mathcal{A}), \{f^{rch(\mathcal{A})}\}_{f\in\Sigma}\rangle$ où $f^{rch(\mathcal{A})}(q_1, \ldots, q_m) = q$ ssi $f(q_1, \ldots, q_m) \rightarrow q \in \Delta$

On peut noter que $\mathsf{A}_{rch(\mathcal{A})}$ est une sous-algèbre de $\mathsf{A}_{\mathcal{A}}$, pour laquelle le support a été restreint aux états accessibles de l'automate $Reachable(\mathcal{A})$.

Nous avons déjà cité le fait que pour $\mathcal{A}$, un automate à rang déterministe et complet, $run_{\mathcal{A}}$ définissait une application de $TERM(\Sigma)$ dans $\mathcal{Q}$ (et pour être plus précis, dans $Reachable(\mathcal{A})$). D'un point de vue algébrique, on peut affirmer que $run_{\mathcal{A}}$ est un morphisme de l'algèbre T_{Σ} vers les algèbres $\mathsf{A}_{\mathcal{A}}$ et $\mathsf{A}_{rch(\mathcal{A})}$. Ce résultat est en fait immédiatement déductible des définitions données précédemment :

$$run_{\mathcal{A}}((f(\tau_1, \ldots, \tau_m))) = f^{\mathcal{A}}(run_{\mathcal{A}}(\tau_1), \ldots, run_{\mathcal{A}}(\tau_m)) = f^{rch(\mathcal{A})}(run_{\mathcal{A}}(\tau_1), \ldots, run_{\mathcal{A}}(\tau_m))$$

Ces égalités proviennent de la définition d'un calcul et de celles de $f^{\mathcal{A}}$ et $f^{rch(\mathcal{A})}$. On peut de plus affirmer de par la définition de la notion d'accessibilité que $run_{\mathcal{A}}$ est un morphisme surjectif de T_{Σ} vers $\mathsf{A}_{rch(\mathcal{A})}$. La dénotation d'un terme clos τ dans ces deux algèbres est donnée par $run_{\mathcal{A}}(\tau)$, *i.e.* $\lfloor\tau\rfloor_{\mathsf{A}_{\mathcal{A}}} = \lfloor\tau\rfloor_{\mathsf{A}_{rch(\mathcal{A})}} = run_{\mathcal{A}}(\tau)$.

Les algèbres $\mathsf{A}_{\mathcal{A}}$ et $\mathsf{A}_{rch(\mathcal{A})}$ peuvent être étendues pour un ensemble $\{V_1, \ldots, V_n\}$ de variables du second ordre respectivement en deux $(\Sigma, \{V_1, \ldots, V_n\})$-structures $\mathsf{R}_{\mathcal{A}}=\langle\langle \mathsf{A}_{\mathcal{A}}, \mathcal{I}\rangle\rangle$ et $\mathsf{R}_{rch(\mathcal{A})}=\langle\langle \mathsf{A}_{rch(\mathcal{A})}, \mathcal{I}^r\rangle\rangle$, où $\mathcal{I}(V_i) = F_i$ et $\mathcal{I}^r(V_i) = F_i \cap Reachable(\mathcal{A})$.

3.2 Une solution au problème de satisfiabilité

Nous allons dans cette section proposer un algorithme permettant de tester la satisfiabilité d'un système de contraintes ensemblistes définies généralisées (en forme aplatie). Les données principales manipulées par notre algorithme seront des automates ascendants d'arbres finis à rang à la fois déterministes et complets. L'algorithme sera défini à partir d'un ensemble de règles d'inférence, qui en fonction des contraintes du système, transforme un automate d'arbres finis en un autre automate ou produit la valeur spéciale $\Box$. Nous prouverons la correction de cet algorithme et finalement, donnerons sa complexité.

3.2.1 L'algorithme

Cette section se décompose en deux sous-sections : la première définit en fonction des variables ensemblistes d'un système de contraintes en forme aplatie la classe des automates d'arbres que nous allons considérer. La seconde présentera un système de règles d'inférence, qui définira notre algorithme.

Préliminaires

Nous allons tout d'abord présenter les automates d'arbres qui vont être utilisés par notre algorithme. La forme de ces automates est simplement définie en fonction des variables ensemblistes apparaissant dans le système de contraintes (en forme aplatie).

Considérons un système de contraintes ensemblistes définies généralisées $\mathcal{SC}$, supposé en forme aplatie [38]. Soit $\{X_1, \ldots, X_n\}$, l'ensemble des variables ensemblistes apparaissant dans $\mathcal{SC}$. Nous allons considérer l'ensemble des automates (ascendants d'arbres finis) déterministes et complets de rang n de la forme $\mathcal{A} = (\Sigma, Q, F, \Delta)$ avec Σ est la signature (finie) sur laquelle est définie $\mathcal{SC}$, $Q = \{0,1\}^n$, *i.e.* l'ensemble des n-uplets de valeurs booléennes $\{0,1\}$ et $F = (F_1, \ldots, F_n)$, où F_i est l'ensemble des états ayant un 1 sur leur $i^{ième}$ composante. On notera $TA_{\mathcal{SC}}$, l'ensemble de ces automates.

L'idée est d'associer une composante du n-uplet d'états finals à chacune des variables ensemblistes de $\mathcal{SC}$, ce qui de par la définition de ce n-uplet revient à associer une composante d'un état à chaque variable ensembliste. Un tel automate $\mathcal{A}$, reconnaissant un unique langage $(L_1, \ldots, L_n)$, définit donc de manière unique une interprétation ensembliste $\mathcal{I}_{\mathcal{A}}$ des variables de $\{X_1, \ldots, X_n\}$ en posant pour tout $i \in \{1, \ldots, n\}$, $\mathcal{I}_{\mathcal{A}}(X_i) = L_i$, ce qui équivaut à dire que pour tout terme clos τ, $\tau \in \mathcal{I}_{\mathcal{A}}(X_i)$ si et seulement si $run_{\mathcal{A}}(\tau) \in F_i$.

Avant de présenter l'algorithme, nous allons terminer cette section en introduisant quelques notations.

Nous avons vu dans la section 3.1.2.0 qu'un automate à rang déterministe et complet permettait la définition de deux structures $\mathsf{R}_{\mathcal{A}}$ et $\mathsf{R}_{rch(\mathcal{A})}$; de par la forme des automates que nous considérons, ces deux structures seront vues comme des $(\Sigma, \{X_1, \ldots, X_n\})$-structures. De plus, en présence de formules monadiques $\top$-bornées, ces deux structures sont étendues en des $(\Sigma, \{X_1, \ldots, X_n, \top\})$-structures respectivement $\mathsf{R}^{\top}_{\mathcal{A}}$ et $\mathsf{R}^{\top}_{rch(\mathcal{A})}$ en posant que, pour chacune d'elles, l'interprétation de la variable particulière $\top$ est égale à $Reachable(\mathcal{A})$.

Il est à noter que puisque pour les automates considérés, les ensembles Σ, Q et F sont fixés, un automate sera totalement défini par Δ, l'ensemble de ses règles de transition. C'est pourquoi nous

[38] Ceci peut toujours être le cas du fait de l'« algorithme » proposé page 67. Nous verrons que les mesures de complexité données dans la section 3.2.2.0 n'en sont pas affectées.

ne distinguerons plus un automate de l'ensemble de ses règles de transitions. Nous utiliserons la notation $switch_1(q,i)$ pour désigner l'état q' tel que q' est égal à q à l'exception (éventuelle) de la $i^{ième}$ composante qui est positionnée à 1 dans l'état q', c'est-à-dire plus formellement $\forall j,\ q' \in F_j \Leftrightarrow (q \in F_j \vee i = j)$. On notera Δ_0, l'« automate vide», dont toutes les règles ont pour membre droit le n-uplet $\{0\}^n$.

Le système d'inférence

L'algorithme est très simple : il est basé sur un système de règles d'inférence $\mathcal{R}$, transformant selon les contraintes un automate de la classe $TA_{\mathcal{SC}}$ en un autre automate de celle-ci et consiste, partant de l'« automate vide» Δ_0, à appliquer les règles de $\mathcal{R}$ jusqu'à l'obtention d'un automate final (stable par application d'une règle quelconque de $\mathcal{R}$) ou d'une valeur spéciale $\Box$. Une règle d'inférence de $\mathcal{R}$ ne fait que modifier le membre droit d'une règle de transition d'un automate en positionnant une des composantes de cet état à 1 (à l'aide de $switch_1$) et ceci en fonction d'une des contraintes du système $\mathcal{SC}$. L'idée sous-jacente à ces transformations (et donc, aux règles d'inférence) est de construire (itérativement) pour un système de contraintes satisfiable un automate Δ_f vérifiant que (i) si pour un terme clos τ, on a $run_{\Delta_f}(\tau) \in F_i$, alors pour toute solution $\mathcal{I}$ du système $\mathcal{SC}$, on a $\tau \in \mathcal{I}(X_i)$, la construction échouant si le système n'est pas satisfiable (correction partielle), (ii) le langage reconnu par Δ_f est solution du système $\mathcal{SC}$ (complétude).

Le système d'inférence $\mathcal{R}$ est donné par les cinq règles de la figure 3.3. Informellement, ce système peut être décrit comme suit. Le sens de la règle *(Compose)* est le suivant : si pour toute solution $\mathcal{I}$, chacun des τ_k vérifie $\tau_k \in \mathcal{I}(X_{i_k})$, alors du fait de la contrainte, le terme $f(\tau_1, \ldots, \tau_m)$ doit appartenir à $\mathcal{I}(X_i)$. Pour la règle *(Clash1)*, dans toute solution $\mathcal{I}$, la constante a appartient à $\mathcal{I}(X_i)$; la contrainte $X_i \subseteq a^{\complement}$ implique donc le fait que le système n'a pas de solution. La règle *(Clash2)* signifie qu'il existe un terme clos de la forme $f(\tau_1, \ldots, \tau_k)$ dans $\mathcal{I}(X_i)$ pour toute solution $\mathcal{I}$; or, la contrainte $X_i \subseteq g(X_{i_1}, \ldots, X_{i_l})$ impose que pour une solution, les termes appartenant à l'interprétation de la variable X_i aient un g comme symbole de tête (avec $f \neq g$). On en conclut que le système $\mathcal{SC}$ est nécessairement insatisfiable. Pour la règle *(Decomp)*, si un terme de la forme $g(\tau_1, \ldots, \tau_l)$ appartient à $\mathcal{I}(X_o)$ pour une solution $\mathcal{I}$, alors du fait de la contrainte, il est nécessaire que pour chacun des k, on ait $\tau_k \in \mathcal{I}(X_k)$. Finalement, la dernière règle *(Member)* signifie que si pour un terme clos τ et pour une solution $\mathcal{I}$, la condition $(\langle\!\langle \mathsf{T}_\Sigma, \tilde{\mathcal{I}} \rangle\!\rangle, [\nu \mapsto \tau]) \models \tilde{\varphi}(x)$ (qui est équivalente à $(\langle\!\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\!\rangle, [\nu \mapsto \tau]) \models \varphi(x)$) est vérifiée, alors du fait de la contrainte $\tau \in \mathcal{I}(X_i)$.

On définit la relation $\vdash_{\mathcal{R}}$ sur l'ensemble $TA_{\mathcal{SC}} \cup \{\Box\}$ en posant $e \vdash_{\mathcal{R}} e'$ si e' est obtenu à partir e par application d'une règle d'inférence de $\mathcal{R}$ ou si $e = e'$ [39]. Notre algorithme calcule un élément final e_f (soit $\Box$, soit un automate de $TA_{\mathcal{SC}}$) tel que $\Delta_0 \vdash^*_{\mathcal{R}} e_f$ ($\vdash^*_{\mathcal{R}}$ dénotant la clôture transitive de la relation $\vdash_{\mathcal{R}}$) et pour tout e de $TA_{\mathcal{SC}} \cup \{\Box\}$, si $e_f \vdash_{\mathcal{R}} e$, alors $e = e_f$, *i.e.* que e_f est un point fixe de la relation $\vdash_{\mathcal{R}}$.

On peut remarquer qu'aucune stratégie d'application des règles d'inférence n'ayant été définie, il convient d'en choisir une pour garantir l'effectivité de l'algorithme. En fait, une simple condition d'équité de la stratégie est suffisante. Ceci n'est cependant pas important du point de vue de la correction de l'algorithme ; nous y reviendrons lorsque nous aborderons le problème de la complexité de celui-ci.

[39] La relation $\vdash_{\mathcal{R}}$ est réflexive.

- (*Compose*)

$$\frac{\Delta \cup \{f(q_1, \ldots, q_m) \to q\}}{\Delta \cup \{f(q_1, \ldots, q_m) \to switch_1(q, i)\}} \text{ si } \begin{cases} f(X_{i_1}, \ldots, X_{i_m}) \subseteq X_i \in \mathcal{SC} \\ \forall k,\ q_k \in F_{i_k} \end{cases}$$

- (*Clash1*)

$$\frac{\Delta \cup \{a \to q\}}{\Box} \text{ si } \begin{cases} X_i \subseteq a^{\mathcal{C}} \in \mathcal{SC} \\ q \in F_i \end{cases}$$

- (*Clash2*)

$$\frac{\Delta \cup \{f(q_1, \ldots, q_m) \to q\}}{\Box} \text{ si } \begin{cases} X_i \subseteq g(X_{i_1}, \ldots, X_{i_l}) \in \mathcal{SC} \\ f \neq g,\ q \in F_i \\ \{q_1, \ldots, q_m\} \subseteq Reachable(\Delta \cup \{f(q_1, \ldots, q_m) \to q\}) \end{cases}$$

- (*Decomp*)

$$\frac{\Delta \cup \{lhs \to q\}}{\Delta \cup \{lhs \to switch_1(q, i)\}} \text{ si } \begin{cases} X_o \subseteq g(X_1, \ldots, X_l) \in \mathcal{SC} \\ \exists\ g(q'_1, \ldots, q'_l) \to q' \in \mathcal{S}\ t.q. \\ \quad \begin{cases} q' \in F_o,\ q = q'_i \\ \{q'_1, \ldots, q'_l\} \backslash \{q'_i\} \subseteq Reachable(\Delta \cup \{lhs \to q\}) \end{cases} \end{cases}$$

- (*Member*)

$$\frac{\Delta \cup \{lhs \to q\}}{\Delta \cup \{lhs \to switch_1(q, i)\}} \text{ si } \begin{cases} \{x \mid \varphi(x)\} \subseteq X_i \in \mathcal{SC} \\ (\mathsf{R}^{\top}_{\Delta \cup \{lhs \to q\}}, [x \mapsto q]) \models \tilde{\varphi}(x)\ ^a \end{cases}$$

FIG. 3.3: Le système d'inférence $\mathcal{R}$ du test de satisfiabilité

[a] Nous rappelons que $\tilde{\varphi}(x)$ est la formule monadique $\top$-bornée correspondante à la formule monadique positive φ apparaissant dans la contrainte.

3.2.2 Correction - complexité

Dans cette section, nous allons démontrer que notre algorithme est partiellement correct en montrant que lorsque celui-ci a pour résultat un automate d'arbres finis, alors l'interprétation qu'il définit est plus petite (au sens de $\sqsubseteq$) que toute solution du système $\mathcal{SC}$ et lorsque le résultat est la valeur spéciale $\Box$, alors le système est effectivement insatisfiable. La complétude sera vérifiée en montrant que si l'algorithme produit un automate, alors l'interprétation fixée par celui-ci est solution du système de contraintes. Nous terminerons cette section en donnant la complexité de l'algorithme proposé (en fixant une stratégie très simple d'application des règles d'inférence).

Correction partielle

Pour prouver la correction de notre algorithme, nous allons énoncer trois propriétés pour un certain automate de $TA_{\mathcal{SC}}$ et une certaine interprétation ensembliste. Nous montrerons que les deux premières propriétés sont équivalentes et impliquent la troisième. Nous verrons ensuite que ces propriétés sont préservées par la relation $\vdash_{\mathcal{R}}$ pour les interprétations correspondant à des solutions du système de contraintes. Ceci nous permettra de conclure aisément à la correction partielle de notre algorithme.

La première propriété pose une condition de correction partielle des composantes positionnées à 1 dans l'état en membre droit de règle vis-à-vis d'une interprétation quelconque en fonction de la correction des états se trouvant en membre gauche vis-à-vis de cette même interprétation.

Propriété 1 $\forall f(q_1, \ldots, q_m) \to q \in \Delta, \forall \tau_1, \ldots, \tau_m \in TERM(\Sigma)$,

$$\Big(\bigwedge_{1 \leq k \leq m} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \Big)$$
$$\Longrightarrow$$
$$(\forall j,\ q \in F_j \Longrightarrow f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_j))$$

La seconde propriété étend cette propriété aux formules monadiques $\top$-bornées. Ainsi, pour une certaine interprétation $\mathcal{I}$,

Propriété 2 $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l$,

$$\Big(\bigwedge_{1 \leq k \leq l} \forall j,\ q_k \in F_j \Rightarrow \tau_k \in \mathcal{I}(X_j) \Big)$$
$$\Longrightarrow$$
$$\left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}(y_1, \ldots, y_l) \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}(y_1, \ldots, y_l) \end{array} \right)$$

Finalement, la troisième propriété est un cas particulier de la seconde pour des formules atomiques construites à l'aide d'un terme clos,

Propriété 3 $\forall \tau \in TERM(\Sigma),\ run_{\Delta}(\tau) \in F_i \Longrightarrow \tau \in \mathcal{I}(X_i)$

On peut alors montrer que :

Lemme 1 Pour une interprétation $\mathcal{I}$ et un automate d'arbres Δ de $TA_{\mathcal{SC}}$,

- Si la Propriété 2 est vraie pour $\mathcal{I}$ dans Δ, alors la Propriété 3 est vraie pour $\mathcal{I}$ dans Δ.
- Les Propriétés 1 et 2 sont équivalentes.

Preuve :

Le premier point est évident, puisque pour une formule atomique quelconque $\tau \in X_i$ (sans variable libre du premier ordre, puisque τ est un terme clos), $\mathsf{R}_\Delta^\top \models \tau \in X_i$ est équivalent à $run_\Delta(\tau) \in F_i$ et que $\langle\langle \mathsf{T}_\Sigma, \tilde{\mathcal{I}} \rangle\rangle \models \tau \in X_i$ est équivalent à $\tau \in \mathcal{I}(X_i)$. Ainsi, cette implication est valide.

Pour le second point, il est évident de voir que si la Propriété 2 est vraie pour une interprétation $\mathcal{I}$, alors ceci implique que la Propriété 1 est vraie elle aussi pour cette même interprétation. La Propriété 1 est en fait un cas particulier de la Propriété 2 pour les formules atomiques de la forme $f(y_1, \ldots, y_m) \in X_j$ (avec $j \in \{1, \ldots, n\}$).
Prouvons maintenant par induction sur la structure des formules monadiques $\top$-bornées $\tilde{\varphi}$ que pour une interprétation $\mathcal{I}$ fixée, la Propriété 1 implique la Propriété 2.

- *si $\tilde{\varphi}$ est une formule atomique, i.e. de la forme $t \in X_i$, alors par induction sur la structure du terme t :*
  - *si t est une variable (du premier ordre), alors l'implication est trivialement vraie.*
  - *si t est une constante ($t = a$), alors l'implication est déduite immédiatement du fait que la Propriété 1 est vraie pour la règle de transition $a \to q$.*
  - *si t est un terme « composé », i.e. de la forme $t = f(t_1, \ldots, t_m)$, alors par hypothèse d'induction, la Propriété 2 est vraie pour tous les t_i et pour toute variable du second ordre X_e.*
    Ainsi, pour tout $\tau_1, \ldots, \tau_l$ et tout $q_1, \ldots, q_l$, tels que la proposition suivante est vraie

$$\bigwedge_{1 \leq k \leq l} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j)$$

on peut en déduire que

$$\bigwedge_{1 \leq i \leq m} \bigwedge_{1 \leq e \leq n} \left(\begin{array}{c} (\mathsf{R}_\Delta^\top, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models t_i \in X_e \\ \Longrightarrow \\ (\langle\langle \mathsf{T}_\Sigma, \tilde{\mathcal{I}} \rangle\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models t_i \in X_e \end{array} \right)$$

En posant $\nu = [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]$ et $\sigma = [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]$, ceci est par définition équivalent à affirmer que :

$$\bigwedge_{1 \leq i \leq m} \bigwedge_{1 \leq e \leq n} \left(\begin{array}{c} \lfloor t_i \rfloor_\nu^{\mathsf{R}_\Delta^\top} \in F_e \\ \Longrightarrow \\ \sigma(t_i) \in \mathcal{I}(X_e) \end{array} \right)$$

où $\sigma(t_i)$ désigne l'application de la substitution close σ au terme t_i.

Posons $q'_i = \lfloor t_i \rfloor_\nu^{\mathsf{R}_\Delta^\top}$ et $\tau'_i = \sigma(t_i)$. En utilisant ces notations, la proposition précédente peut être reformulée de la façon suivante :

$$\bigwedge_{1 \leq k \leq m} \forall e,\ q'_k \in F_e \Longrightarrow \tau'_k \in \mathcal{I}(X_e)$$

Par hypothèse, la Propriété 1 étant vraie dans Δ, et ce en particulier pour les q'_i et les τ'_i, on peut donc en déduire que :

$$(\forall e,\ \lfloor f(y_1, \ldots, y_m) \rfloor_{[y_1 \mapsto q_1, \ldots, y_m \mapsto q_m]}^{\mathsf{R}_\Delta^\top} \in F_e \Longrightarrow f(\tau'_1, \ldots, \tau'_m) \in \mathcal{I}(X_e))$$

Ceci est équivalent à dire, par définition, pour tout X_e, que

$$\left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models f(t_1, \ldots, t_m) \in X_e \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models f(t_1, \ldots, t_m) \in X_e \end{array} \right)$$

qui est bien l'expression de la partie impliquée de la Propriété 2, pour une formule $\tilde{\varphi}$ de la forme $f(t_1, \ldots, t_m) \in X_e$.

- *si $\tilde{\varphi}$ est de la forme $\tilde{\varphi}_1 \wedge \tilde{\varphi}_2$ ou $\tilde{\varphi}_1 \vee \tilde{\varphi}_2$, alors par hypothèse d'induction, la Propriété 2 est vraie pour $\tilde{\varphi}_1$ et $\tilde{\varphi}_2$. Considérons ces deux hypothèses écrites pour l'ensemble des variables du premier ordre libres dans $\tilde{\varphi}_1$ et $\tilde{\varphi}_2$, i.e. $Var_{\mathcal{X}}(\tilde{\varphi}_1) \cup Var_{\mathcal{X}}(\tilde{\varphi}_2)$; en posant*

$$\Psi(q_1, \ldots, q_l, \tau_1, \ldots, \tau_l) \equiv \bigwedge_{1 \leq k \leq l} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j)$$

on a (pour l'indice $e \in \{1, 2\}$) : $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l,$

$$\Psi(q_1, \ldots, q_l, \tau_1, \ldots, \tau_l) \Longrightarrow \left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}_e \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}_e \end{array} \right)$$

ceci étant équivalent à $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l,$

$$\Psi(q_1, \ldots, q_l, \tau_1, \ldots, \tau_l) \Longrightarrow \bigwedge_{1 \leq e \leq 2} \left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}_e \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}_e \end{array} \right)$$

En utilisant les équivalences logiques habituelles, on peut déduire de la proposition précédente les deux énoncés montrant que l'implication des propriétés est vraie pour ces deux formes de formules, à savoir : $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l$

$$\Psi(q_1, \ldots, q_l, \tau_1, \ldots, \tau_l) \Longrightarrow \left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}_1 \wedge \tilde{\varphi}_2 \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}_1 \wedge \tilde{\varphi}_2 \end{array} \right)$$

et $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l$

$$\Psi(q_1, \ldots, q_l, \tau_1, \ldots, \tau_l) \Longrightarrow \left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}_1 \vee \tilde{\varphi}_2 \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}_1 \vee \tilde{\varphi}_2 \end{array} \right)$$

- *si $\tilde{\varphi}$ est de la forme $\exists x\,(x \in \top \wedge \tilde{\varphi}')$, alors il est suffisant de monter que : pour tout état accessible q de l'automate Δ, il existe un terme clos τ tel que* $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l,$

$$\begin{array}{c} \left(\bigwedge_{1 \leq k \leq l} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \right) \\ \Longrightarrow \\ \left(\begin{array}{c} (\mathsf{R}^{\top}_{\mathcal{S}}, [x \mapsto q, y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}'(x, y_1, \ldots, y_l) \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [x \mapsto \tau, y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}'(x, y_1, \ldots, y_l) \end{array} \right) \end{array}$$

Soit τ un terme clos tel que $run_\Delta(\tau)=\lfloor\tau\rfloor_{R^\top_\Delta}=q$ (trouver un tel terme est possible puisque, par hypothèse, q est accessible dans Δ).

Or, puisque la Propriété 2 est valide pour $\tilde{\varphi}'$ in Δ par hypothèse d'induction et que le choix pour le terme τ implique que $\forall j,\ q \in F_j \Longrightarrow \tau \in \mathcal{I}(X_j)$ (car, du fait du cas de base, la Propriété 2 est vraie dans Δ pour les formules atomiques (closes)), on peut donc en déduire que l'énoncé précédent est valide.

- *si $\tilde{\varphi}$ est de la forme $\forall x\,(x \in \top \Rightarrow \tilde{\varphi}')$, alors le preuve est duale du cas précédent. Il suffit donc de montrer que : pour tout terme clos τ, il existe un état accessible q dans Δ tel que $\forall q_1, \ldots, q_l, \forall \tau_1, \ldots, \tau_l$,*

$$\left(\bigwedge_{1\leq k\leq l} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \right)$$
$$\Longrightarrow$$
$$\left(\begin{array}{c} (\mathsf{R}^\top_\Delta, [x \mapsto q, y_1 \mapsto q_1, \ldots, y_l \mapsto q_l]) \models \tilde{\varphi}'(x, y_1, \ldots, y_l) \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_\Sigma, \tilde{\mathcal{I}} \rangle\!\rangle, [x \mapsto \tau, y_1 \mapsto \tau_1, \ldots, y_l \mapsto \tau_l]) \models \tilde{\varphi}'(x, y_1, \ldots, y_l) \end{array} \right)$$

Là encore, la Propriété 2 est vraie pour $\tilde{\varphi}'$ dans Δ. Posons $q = \lfloor\tau\rfloor_{R^\top_\Delta}$ (ce qui entraîne par définition que q est accessible dans Δ). De plus, puisque la Propriété 2 est vraie pour toute formule atomique close (du fait du cas de base) dans Δ, on a $\forall j,\ q \in F_j \Longrightarrow \tau \in \mathcal{I}(X_j)$. Ces deux faits nous permettent alors de conclure à la validité de l'énoncé précédent.

■

Nous allons montrer que la Propriété 1, et donc, les Propriétés 2 et 3, sont préservées par la relation $\vdash_\mathcal{R}$ lorsque les interprétations considérées sont des solutions du système de contraintes ensemblistes.

Lemme 2 Si la Propriété 1 est vraie pour un automate d'arbres Δ et pour une solution $\mathcal{I}$ et que $\Delta \vdash_\mathcal{R} \Delta'$ (avec $\Delta' \neq \Box$), alors la Propriété 1 est vraie pour Δ' et $\mathcal{I}$.

Preuve :

Le cas où $\Delta = \Delta'$ est trivial ; nous supposons donc maintenant que $\Delta \neq \Delta'$.

Considérons une solution $\mathcal{I}$ du système de contraintes. Il est évident que la propriété 1 demeure vraie pour les règles qui n'ont pas été modifiées par l'application de la règle d'inférence correspondant à $\Delta \vdash_\mathcal{R} \Delta'$.

Soit $f(q_1, \ldots, q_m) \to q$, la règle de transition de Δ qui s'est vue transformée dans Δ' en $f(q_1, \ldots, q_m) \to switch_1(q, i)$. Puisque par hypothèse, la Propriété 1 est vraie pour Δ et $\mathcal{I}$, il est alors suffisant de prouver que $\forall \tau_1, \ldots, \tau_m \in TERM(\Sigma)$,

$$\left(\bigwedge_{1\leq k\leq m} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \right) \Longrightarrow f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_i)$$

est vraie dans l'automate calculé Δ'.

Ainsi, selon la règle d'inférence de $\mathcal{R}$ appliquée pour le pas $\Delta \vdash_\mathcal{R} \Delta'$:

- *Pour la règle (Compose) : par les conditions de la règle d'inférence, $\forall k,\ q_k \in F_{i_k}$. Donc, pour tous $\tau_1, \ldots, \tau_m$ tels que $\tau_k \in \mathcal{I}(X_{i_k})$, $f(\tau_1, \ldots, \tau_m)$ doit appartenir à $\mathcal{I}(X_i)$. Ceci est vrai car $\mathcal{I}$ est une solution et que, par définition, le fait que pour tout k, $\tau_k \in \mathcal{I}(X_{i_k})$ implique que $f(\tau_1, \ldots, \tau_m) \in f(\mathcal{I}(X_{i_1}), \ldots, \mathcal{I}(X_{i_m}))$.*

- *Pour la règle (Decomp) : Considérons le terme* $t = g(x'_1, \ldots, f(x_1, \ldots, x_m), \ldots, x'_l)$ *et la formule atomique* $t \in X_o$. *Puisque la Propriété 1 et donc, par le lemme 1, la Propriété 2 sont vraies pour* $\mathcal{S}$ *et* $\mathcal{I}$, *on a*
$\forall q'_1, \ldots, q'_l,\ \forall \tau'_1, \ldots, \tau'_l,\ \forall \tau_1, \ldots, \tau_m,$

$$\bigwedge_{1 \leq k \leq m} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \ \wedge \bigwedge_{\substack{1 \leq k' \leq l \\ k' \neq i}} \forall j,\ q'_{k'} \in F_j \Longrightarrow \tau'_{k'} \in \mathcal{I}(X_j)$$

$$\Longrightarrow$$

$$\left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, (\nu'_q \circ \nu_q)) \models t \in X_o \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, (\nu'_\tau \circ \nu_\tau)) \models t \in X_o \end{array} \right)$$

où $\begin{cases} \nu'_q = [x'_1 \mapsto q'_1, \ldots, x'_l \mapsto q'_l] & \nu_q = [x_1 \mapsto q_1, \ldots, x_m \mapsto q_m] \\ \nu'_\tau = [x'_1 \mapsto \tau'_1, \ldots, \tau'_l \mapsto \tau'_l] & \nu_\tau = [x_1 \mapsto \tau_1, \ldots, \tau_m \mapsto \tau_m] \end{cases}$
Cette proposition est vraie en particulier si on pose que pour tout k', $q'_{k'} = run_{\Delta}(\tau'_{k'})$. *Dans ce cas, en utilisant le fait que la Propriété 3 est vraie pour* Δ *et pour* $\mathcal{I}$ *(puisque, du fait du lemme 1, impliquée par la Propriété 1), la proposition suivante est valide*

$$\bigwedge_{\substack{1 \leq k' \leq l \\ k' \neq i}} \forall j,\ q'_{k'} \in F_j \Longrightarrow \tau'_{k'} \in \mathcal{I}(X_j)$$

De plus, par les conditions de la règle d'inférence, on a $(\mathsf{R}^{\top}_{\Delta}, (\nu'_q \circ \nu_q)) \models t \in X_o$ *Ainsi, on peut en déduire que*

$$\Big(\bigwedge_{1 \leq k \leq m} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \Big) \Longrightarrow (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle,\ (\nu'_\tau \circ \nu_\tau) \models t \in X_o)$$

ce qui, par définition, est équivalent à

$$\Big(\bigwedge_{1 \leq k \leq m} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \Big) \Longrightarrow (g(\tau'_1, \ldots, f(\tau_1, \ldots, \tau_m), \ldots, \tau'_l) \in \mathcal{I}(X_o))$$

Or, puisque $\mathcal{I}$ *est une solution, la proposition* $g(\tau'_1, \ldots, f(\tau_1, \ldots, \tau_m), \ldots, \tau'_l) \in \mathcal{I}(X_o)$ *implique* $f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_i)$.

- *Pour la règle (Member) : puisque, par hypothèse, la Propriété 1 est vraie pour* Δ *et* $\mathcal{I}$, *on a pour la règle* $f(q_1, \ldots, q_m) \to q$ *de* Δ *considérée par la règle (Member)*
$\forall \tau_1, \ldots, \tau_m,$

$$\Big(\bigwedge_{1 \leq k \leq m} \forall j,\ q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j) \Big) \Longrightarrow (\forall j,\ q \in F_j \Longrightarrow f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_j))$$

De plus, la Propriété 1 impliquant la Propriété 2 pour tout automate, on a en particulier pour Δ *et* $\tilde{\varphi}$,

$$(\forall j,\ q \in F_j \Longrightarrow f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_j))$$

$$\Longrightarrow$$

$$\left(\begin{array}{c} (\mathsf{R}^{\top}_{\Delta}, [x \mapsto q]) \models \tilde{\varphi}(x) \\ \Longrightarrow \\ (\langle\!\langle \mathsf{T}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [x \mapsto f(\tau_1, \ldots, \tau_m)]) \models \tilde{\varphi}(x) \end{array} \right)$$

Or, du fait de la proposition 6, $(\langle\langle \mathsf{T}_\Sigma, \tilde{\mathcal{I}}\rangle\rangle, [x \mapsto f(\tau_1, \ldots, \tau_m)]) \models \tilde{\varphi}(x)$ *est équivalent à* $(\langle\langle \mathsf{T}_\Sigma, \mathcal{I}\rangle\rangle, [x \mapsto f(\tau_1, \ldots, \tau_m)]) \models \varphi(x)$*, qui est, par définition, lui-même équivalent à* $f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(\{x \mid \varphi(x)\})$*. Puisque* $\mathcal{I}$ *est une solution du système, cette dernière proposition implique* $f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_i)$*.*
De plus, de par les conditions de la règle d'inférence, la proposition $(\mathsf{R}_\Delta^\top, [x \mapsto q]) \models \tilde{\varphi}(x)$ *est valide, ce qui nous permet de conclure à*

$$\Big(\bigwedge_{1 \le k \le m} \forall j,\, q_k \in F_j \Longrightarrow \tau_k \in \mathcal{I}(X_j)\Big) \Longrightarrow f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_i)$$

■

Il est alors immédiat de conclure à la correction partielle de l'algorithme présenté.

Théorème 12 Soit e_f la valeur calculée par l'algorithme pour un système de contraintes ensemblistes $\mathcal{SC}$.

- si e_f est un automate Δ_f ($e_f \neq \Box$), alors pour toute solution $\mathcal{I}$ et pour tout terme clos τ, $run_{\Delta_f}(\tau) \in F_i$ implique $\tau \in \mathcal{I}(X_i)$.
- si $e_f = \Box$, alors $\mathcal{SC}$ est insatisfiable.

Preuve :

Il est évident de voir que la Propriété 1 (et, de fait du lemme 1, les Propriétés 2 et 3) sont valides pour Δ_0*, et ce, pour toute solution. Ainsi, par le lemme 2, ces propriétés sont vraies pour tout automate et toute solution du système au cours du calcul.*

- *Le premier point est immédiat, puisque, du fait de l'affirmation précédente, la Propriété 3 est vraie dans* Δ_f *pour toute solution.*
- *Pour le second point : soit* Δ_e *un automate* ($\Delta_e \neq \Box$) *tel que* $\Delta_0 \vdash_{\mathcal{R}}^* \Delta_e$ *et* $\Delta_e \vdash_{\mathcal{R}} \Box$*. La propriété 3 est donc valide pour* Δ_e *et pour toute solution* $\mathcal{I}$*. Selon la règle d'inférence appliquée pour* $\Delta_e \vdash_{\mathcal{R}} \Box$ *:*
  - *Pour la règle (Clash1) : puisque la Propriété 3 est vraie pour* Δ_e*, on peut affirmer que* $q \in F_i$ *implique que* $a \in \mathcal{I}(X_i)$ *pour toute solution* $\mathcal{I}$*.*
  - *Pour la règle (Clash2) : par définition de la notion d'accessibilité, il existe des termes clos* $\tau_1, \ldots, \tau_m$ *tels que* $run_{\Delta_e}(\tau_k) = q_k$ *et donc,* $run_{\Delta_e}(f(\tau_1, \ldots, \tau_m)) = q \in F_i$ *(avec* $f \neq g$*). Puisque la Propriété 3 est vraie pour* Δ_e*, ceci implique que pour toute solution* $\mathcal{I}$*,* $f(\tau_1, \ldots, \tau_m) \in \mathcal{I}(X_i)$*.*

Dans chacun des deux cas, on peut donc affirmer que le système de contraintes n'a pas de solution.

■

La correction partielle peut se résumer en disant que si la valeur finale calculée est $\Box$, alors le système $\mathcal{SC}$ est insatisfiable, et dans le cas contraire, c-à-d si cette valeur est un automate, alors l'interprétation qu'il définit est plus petite (au sens de $\sqsubseteq$) que toute solution du système.

Complétude

Soit e_f, un point fixe calculé de $\vdash_{\mathcal{R}}$ par notre algorithme. Supposons que e_f soit en fait un automate d'arbres Δ_f (*i.e.* $e_f \neq \Box$). Nous allons montrer que l'interprétation $\mathcal{I}_{\Delta_f}$ définie par cet automate est une solution du système de contraintes $\mathcal{SC}$.

Théorème 13 $\mathcal{I}_{\Delta_f}$, l'interprétation définie par Δ_f [40], est une solution de $\mathcal{SC}$.

[40] Nous rappelons que si Δ_f reconnait le langage $(L_1, \ldots, L_n)$, alors $\mathcal{I}_{\Delta_f}$ est définie par $\forall X_i$, $\mathcal{I}_{\Delta_f}(X_i) = L_i$.

Preuve :

Par définition, $\tau \in \mathcal{I}_{\Delta_f}(X_i)$ ssi $run_{\Delta_f}(\tau) \in F_i$. Selon les différents types de contraintes apparaissant dans un système en forme aplatie :

- *Pour une contrainte de la forme $f(X_{i_1}, \ldots, X_{i_m}) \subseteq X_i$:*
  Considérons $\tau_1, \ldots, \tau_m \in TERM(\Sigma)$, tels que $\tau_k \in \mathcal{I}_{\Delta_f}(X_{i_k})$. Puisque par hypothèse, Δ_f est un point fixe de $\vdash_{\mathcal{R}}$, du fait de la règle (Compose), $run_{\Delta_f}(f(\tau_1, \ldots, \tau_m)) \in F_i$. Ainsi, $f(\tau_1, \ldots, \tau_m) \in \mathcal{I}_{\Delta_f}(X_i)$.
- *Pour une contrainte de la forme $X_o \subseteq g(X_1, \ldots, X_l)$:*
  Considérons un terme clos τ appartenant à $\mathcal{I}_{\Delta_f}(X_o)$, i.e. vérifiant $run_{\Delta_f}(\tau) \in F_o$. Selon la forme de ce terme :
  Si $\tau = f(\tau_1, \ldots, \tau_m)$ et $f \neq g$, alors la règle (Clash2) aurait du être appliquée à Δ_f. Ceci est impossible puisque Δ_f est un point fixe différent de $\square$.
  Si $\tau = g(\tau_1, \ldots, \tau_l)$, par définition il existe une règle de transition $g(q'_1, \ldots, q'_l) \to q'$ dans Δ_f telle que $q' \in F_o$ et pour tout k, q_{i_k} est un état accessible dans cet automate. Puisque Δ_f est un point fixe de $\vdash_{\mathcal{R}}$, pour tout i, $q'_i \in F_i$. Et donc, par définition, $\tau_i \in \mathcal{I}_{\Delta_f}(X_i)$; ainsi, $g(\tau_1, \ldots, \tau_l) \in \mathcal{I}_{\Delta_f}(g(X_1, \ldots, X_l))$.
- *Pour une contrainte de la forme $X_i \subseteq a^{\complement}$:*
  Il est suffisant de montrer que $a \notin \mathcal{I}_{\Delta_f}(X_i)$. Si tel n'était pas le cas, alors par définition $run_{\Delta_f}(a) \in F_i$. Ainsi, pour la règle de transition $a \to q$ de Δ_f, q appartiendrait à F_i. Ceci est impossible puisque Δ_f est un point fixe pour $\vdash_{\mathcal{R}}$ (et, en particulier pour la règle (Clash1)) supposé différent de $\square$.
- *Pour une contrainte de la forme $\{x \mid \varphi(x)\} \subseteq X_i$:*
  Soit τ un terme clos tel que $\tau \in \mathcal{I}_{\Delta_f}(\{x \mid \varphi(x)\})$. Par définition, ceci est équivalent à $\langle\langle \mathsf{T}_\Sigma, \mathcal{I}_{\Delta_f} \rangle\rangle, [x \mapsto \tau] \models \varphi(x)$. En utilisant la proposition 5 et le contenu de la section 3.1.2.0, on peut déduire que $(\mathsf{R}_{rch(\Delta_f)}, [x \mapsto run_{\Delta_f}(\tau)]) \models \varphi(x)$
  Donc, par la proposition 6, $(\mathsf{R}^{\top}_{rch(\Delta_f)}, [x \mapsto run_{\Delta_f}(\tau)]) \models \tilde{\varphi}(x)$
  Ainsi, par la proposition 7 et le contenu de la section 3.1.2.0, $(\mathsf{R}^{\top}_{\Delta_f}, [x \mapsto run_{\Delta_f}(\tau)]) \models \tilde{\varphi}(x)$
  Finalement, puisque Δ_f est un point fixe de $\vdash_{\mathcal{R}}$, par la règle (Member), $run_{\Delta_f}(\tau) \in F_i$ ce qui implique que $\tau \in \mathcal{I}_{\Delta_f}(X_i)$.

■

On peut déduire des preuves de correction partielle et de complétude que notre algorithme calcule la valeur $\square$ si et seulement si le système de contraintes ensemblistes $\mathcal{SC}$ considéré est insatisfiable et que si la valeur finale calculée est un automate d'arbres, alors l'interprétation ensembliste définie par celui-ci est la plus petite solution du système $\mathcal{SC}$. Par conséquent, sous l'hypothèse d'une stratégie « terminante», ceci permet de répondre au problème de confluence de notre algorithme.

Nous allons dans la section suivante aborder le problème de la complexité de l'algorithme présenté.

Complexité

Nous rappelons que le problème de satisfiabilité d'un système de contraintes ensemblistes définies est un problème *EXPTIME*-complet [16]. Ceci nous donne une borne inférieure à la complexité de notre problème.

Proposition 8 Le problème de satisfiabilité d'un système de contraintes ensemblistes définies généralisées est *EXPTIME*-dur.

Preuve :

Trivial, puisque tout système de contraintes définies est par définition un système défini généralisé.

■

En utilisant l'algorithme précédemment décrit (avec une stratégie particulière d'application des règles d'inférence), on a

Proposition 9 Le problème de satisfiabilité d'un système de contraintes ensemblistes définies généralisées est un problème *EXPTIME*-complet.

Preuve :

Notons tout d'abord que tout système de contraintes définies généralisées peut être mis en forme aplatie à l'aide d'un algorithme polynômial en temps (et donc, en espace). Il est donc suffisant de démontrer que l'algorithme proposé (répondant au problème de satisfiabilité d'un système en forme aplatie) est EXPTIME[41].

Pour ce faire, nous allons tout d'abord spécifier une stratégie pour l'application des règles d'inférence de $\mathcal{R}$.

Cette stratégie consiste en l'itération d'un procédure de type « cherche-et-switch », qui s'achève lorsque deux itérations successives n'ont conduit à aucune modification de l'automate ou lorsque la valeur $\square$ a été calculée. Cette procédure revient à chercher une contrainte et une règle de transition telles qu'une règle d'inférence de $\mathcal{R}$ puisse être déclenchée avec ce couple entraînant soit la modification (explicite) de la règle de transition (transformation d'un 0 en un 1), soit le calcul de $\square$. Ceci assure qu'à chaque exécution de cette procédure si l'automate auquel elle s'applique n'est pas un point fixe de $\vdash_{\mathcal{R}}$, alors (au moins) un état d'un membre droit de règle est modifié.

Soit $N_{\mathcal{V}}$ (resp. $N_{\mathcal{SC}}$) le nombre de variables ensemblistes (resp. de contraintes ensemblistes) apparaissant dans le système de contraintes donné en entrée.

Soit S_φ la longueur de la plus grande formule monadique positive apparaissant dans les expressions ensemblistes des contraintes de $\mathcal{SC}$ et S_Q la longueur de la plus grande quantification de ces formules.

Soient N_f et amax respectivement le nombre de symboles de fonctions et l'arité maximale de la signature Σ.

Dans les automates que nous considérons, il y a exactement $2^{N_{\mathcal{V}}}$ états et au plus $2^{N_{\mathcal{V}}\,\text{amax}}\,N_f$ règles de transition. Nous notons $|\mathcal{S}|$ une borne supérieure de la taille d'un automate donnée par $2^{N_{\mathcal{V}}\,\text{amax}}\,N_f\,(\text{amax}+1)\,N_{\mathcal{V}}$. On peut supposer que tester la finalité d'un état et que l'opération switch$_1$ peuvent être faites en temps constant. L'accessibilité d'un état peut être testée en un temps linéaire par rapport à $|\mathcal{S}|$.

Puisqu'à chaque itération, au moins un état en membre droit de règle voit l'une des ses composantes modifiée de 0 vers 1, on peut donc borner le nombre total d'itérations par $2^{N_{\mathcal{V}}\,\text{amax}}\,N_f\,N_{\mathcal{V}}$. De plus, du fait du système d'inférence $\mathcal{R}$, il se trouve que pour chaque couple (contrainte, règle de transition) au plus deux règles d'inférence peuvent (éventuellement) être appliquées[42] *; ainsi, au plus deux tests de déclenchabilité (pour la ou les règles d'inférence*

[41] Pour un problème de taille n, $EXPTIME(n) = \bigcup_k DTIME(2^{n^k})$.

[42] Une telle situation peut se produire pour un couple dont la contrainte est de la forme $X_i \subseteq g(X_{i_1}, \ldots, X_{i_l})$ entraînant le fait que le test de déclenchabilité doit être réalisé pour la règle *(Decomp)* et la règle *(Clash2)*.

correspondant au couple) doit être effectué pour un tel couple. Donc, à chaque exécution de la procédure « cherche-et-switch », il est réalisé au plus $(2^{amax\,N_\mathcal{V}}\,2\,N_f\,N_{SC})$ *tests de déclenchabilité.*

Le coût de ce test de déclenchabilité dépend de la règle d'inférence : il peut être effectué en $O(amax)$ *pour la règle (Compose) et en* $O(1)$ *pour la règle (Clash1). Pour la règle (Clash2), ce test consiste principalement à tester l'accessibilité d'au plus amax états ; ceci peut être fait en* $O(amax\,|\mathcal{S}|)$*. Pour la règle (Decomp), une règle de transition particulière ayant g pour symbole de tête doit être trouvée ; pour vérifier qu'une règle ayant g pour symbole de tête convient, il est nécessaire pour l'une d'entre elles, de tester parmi les (au plus) amax états apparaissant en membre gauche, la présence de l'état q et l'accessibilité des (au plus)* $(amax - 1)$ *autres états de ce même membre gauche. Ainsi, le test pour cette règle d'inférence peut être fait en au plus* $O(2^{N_\mathcal{V}\,amax} * ((amax - 1)\,|\mathcal{S}| + amax))$*. Le test de déclenchabilité pour la règle (Member) peut être décomposé en deux parties : la première calcule l'ensemble des états accessibles dans l'automate en* $O(|\mathcal{S}|)$ *et la seconde tente de prouver la validité de la formule* $\tilde{\varphi}$*, ce qui peut être réalisé en* $O(2^{N_\mathcal{V}\,S_Q}.S_\varphi)$*.*

Notons T, la taille du système de contraintes ensemblistes donné en entrée. Globalement, le nombre de tests de déclenchabilité peut être majoré par $k_1\,T^2\,2^{k_2\,T}$*, où* k_1, k_2 *sont des constantes dépendantes de la signature* Σ*. Le coût de chacun des tests de déclenchabilité (selon la règle d'inférence) peut être borné par* $O(k_3\,T\,2^{k_4\,T^2})$*, où* k_3, k_4 *sont des constantes dépendantes de la signature. Ainsi, la complexité globale de l'algorithme proposé est en* $O(k\,T^3\,2^{k'\,(T^2+T)})$*, où* k, k' *sont des constantes dépendantes de la signature.*

■

Remarques :

Les automates de la classe TA_{SC} Le point crucial de la méthode que nous avons présentée ici est la classe des automates TA_{SC} et ceci sur deux points.

Le premier est le lien existant entre les variables ensemblistes d'un système et la définition des états d'un automate de cette classe, ainsi que celle du n-uplets d'états finals. Cet aspect se retrouve dans les travaux qui ont précédé celui-ci sur les classes de contraintes définies, (positive) à intersection, et la classe *(PNSC)*. En effet, un état de l'automate peut être vu comme la fonction caractéristique d'un sous-ensemble des variables ensemblistes d'un système $\mathcal{SC}$. On retrouve là d'une certaine manière les variables créées par l'algorithme de [39] et plus particulièrement l'indice associé à ses variables. Il en va de même des variables d'intersection de [16] pour lesquelles nous avons déja fait le rapprochement avec celles du travail précédemment cité. Plus proche encore sont les états des automates d'ensembles d'arbres définis dans [32], puisqu'ils sont tout comme les nôtres définis comme une valuation booléenne. La principale distinction qui peut être faite serait que dans tous ces travaux, les « états » sont en fait un n-uplet de booléens dont les composantes sont associées aux expressions et non aux variables du système. C'est la forme aplatie, ainsi que l'automate qui contient les notions de composition de termes, qui font que seules les variables sont considérées ici.

Le second point est la nature déterministe et complète des automates qui met en lumière l'aspect algèbrique de le solution proposé. En effet, tout comme dans [14], l'algorithme peut tout simplement être vu comme la recherche d'une algèbre finie (ayant pour domaine les états accessibles de l'automate) définissant une unique structure (le n-uplets d'états finals étant fixé) ; cette recherche s'effectue parmi toutes les algèbres possibles (*i.e.* celles ayant pour domaine les parties de l'ensemble des variables ensemblistes, pouvant être vues comme des ensembles de

prédicats monadiques) et non pas de manière non-déterminste, mais par un calcul itératif.

Treillis des solutions Nous avons vu précédemment qu'un système de contraintes ensemblistes définies généralisées satisfiable avait une plus petite solution (qui est celle calculée par notre algorithme sous la forme d'un automate d'arbres). Cependant, nous avons déja mentionné le fait que cela était aussi le cas pour un système de contraintes définies et pouvait être vu comme un conséquence du fait que l'ensemble des solutions d'un tel système de contraintes était clos par intersection $\sqcap$. Cette propriété peut se résumer par le fait que l'ensemble des solutions forme un semi-treillis inférieur complet. Nous allons voir que la généralisation que nous proposons préserve cette propriété. Pour tout ensemble (non vide) de solutions $S_{\mathcal{I}}$ d'un système de contraintes définies généralisées, $\sqcap S_{\mathcal{I}}$ [43] est aussi une solution, ce qui se résume en disant que l'ensemble des solutions est « clos par intersection ».

Proposition 10 Pour tout système satisfiable $\mathcal{SC}$ de contraintes ensemblistes définies généralisées, ayant $\mathcal{I}_m$ pour plus petite solution, $(SOL(\mathcal{SC}), \sqsubseteq, \sqcap, \mathcal{I}_m)$ est un semi-treillis inférieur complet.

Preuve :

Pour prouver cela, il suffit de montrer que pour tout ensemble de solutions S, $\sqcap S$ est une solution. Ceci peut être fait en inspectant chacun des types de contraintes (que nous supposerons sans perte de généralité en forme aplatie). Pour les contraintes concernant des compositions fonctionnelles, à savoir $f(X_1, \ldots, X_m) \subseteq X$ et $X \subseteq f(X_1, \ldots, X_m)$, la preuve est similaire à celle faite dans [39] pour le cas des contraintes définies. Nous allons simplement rappeler le cas $f(X_1, \ldots, X_m) \subseteq X$, l'autre cas étant symétrique.
Pour tout terme τ,

$$
\begin{aligned}
&\tau \in \sqcap S(f(X_1, \ldots, X_m)) \\
&\Leftrightarrow \tau \in f(\sqcap S(X_1), \ldots, \sqcap S(X_m)) \quad \textit{(car } \sqcap S \textit{ est une interprétation ensembliste)} \\
&\Leftrightarrow \forall \mathcal{I} \in S,\ \tau \in f(\mathcal{I}(X_1), \ldots \mathcal{I}(X_m)) \quad \textit{(par définition de } \sqcap S\textit{)} \\
&\Leftrightarrow \forall \mathcal{I} \in S,\ \tau \in \mathcal{I}(f(X_1, \ldots, X_m)) \quad \textit{(car } \mathcal{I} \textit{ est une interprétation ensembliste)} \\
&\Rightarrow \forall \mathcal{I} \in S,\ \tau \in \mathcal{I}(X) \quad \textit{(car } \mathcal{I} \textit{ est une solution)} \\
&\Leftrightarrow \tau \in \sqcap S(X) \quad \textit{(par définition de } \sqcap S\textit{)}
\end{aligned}
$$

La preuve pour $X \subseteq a^{\complement}$ est trivial. Détaillons le cas pour $\{x \mid \varphi(x)\} \subseteq X$
Pour tout terme τ,

$$
\begin{aligned}
&\tau \in \sqcap S(\{x \mid \varphi(x)\}) \\
&\Leftrightarrow (\sqcap S, [x \mapsto \tau]) \models \varphi(x) \quad \textit{(car } \sqcap S \textit{ est une interprétation ensembliste)} \\
&\Rightarrow \forall \mathcal{I} \in S,\ (\mathcal{I}, [x \mapsto \tau]) \models \varphi(x) \quad (*) \\
&\Leftrightarrow \forall \mathcal{I} \in S,\ \tau \in \mathcal{I}(\{x \mid \varphi(x)\}) \quad \textit{(car } \mathcal{I} \textit{ est une interprétation ensembliste)} \\
&\Rightarrow \forall \mathcal{I} \in S,\ \tau \in \mathcal{I}(X) \quad \textit{(car } \mathcal{I} \textit{ est une solution)} \\
&\Rightarrow \tau \in \sqcap S(X) \quad \textit{(par définition de } \sqcap S\textit{)}
\end{aligned}
$$

La preuve de l'implication (*) se fait par induction sur la structure de la formule φ.

■

[43] Nous rappelons que l'interprétation $\sqcap S_{\mathcal{I}}$ est définie par : pour toute variable ensembliste X, $\tau \in \sqcap S_{\mathcal{I}}(X)$ si et seulement si pour toute solution $\mathcal{I}$ de $S_{\mathcal{I}}$, $\tau \in \mathcal{I}(X)$.

Nous avons vu dans cette section que deux des propriétés de base de la classe des contraintes définies étaient préservées par notre extension, à savoir la complexité du problème de satisfiabilité et le fait que les solutions d'un système satisfiable forment un semi-treillis inférieur complet.

Nous allons dans la section suivante détailler le fonctionnement de notre algorithme sur un exemple.

3.2.3 Exemple

Reprenons le système de contraintes en forme aplatie donné dans l'exemple 13.

$$\begin{array}{c} a \subseteq X_4 \\ b \subseteq X_5 \\ f(X_4, X_5) \subseteq X_1 \\ f(X_5, X_6) \subseteq X_1 \end{array} \qquad \begin{array}{c} g(X_5, X_5) \subseteq X_2 \\ g(X_5, X_2) \subseteq X_2 \\ \{x \mid \exists z \forall y\, f(x, y) \in X_1 \wedge g(z, x) \in X_2\} \subseteq X_3 \\ \{x \mid \exists y,\ y \in X_4 \vee x \in X_6\} \subseteq X_6 \end{array}$$

Les états de l'automate que nous allons utiliser seront de la forme $[X_1, X_2, X_3, X_4, X_5, X_6]$. Par exemple, pour la règle $a \to [1, 0, 0, 0, 1, 1]$, le membre droit signifie que pour toute solution, a appartient à l'interprétation de X_1, X_5 et X_6.

Afin de pouvoir représenter les automates de manière plus compacte, les états en membre gauche de règle pourront avoir des composantes non-définies « _ » ; Ainsi, l'état $[1, 0, 0, \_, 1, 1]$ sera une méta-représentation des états $[1, 0, 0, 0, 1, 1]$ et $[1, 0, 0, 1, 1, 1]$. Pour une règle de transition, la méta-règle $f([1, 0, 0, \_, 1, 1], [0, 0, 0, 0, 0, \_]) \to [0, 0, 1, 0, 0, 0]$ représentera les règles

$$\begin{array}{l} f([1, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0]) \to [0, 0, 1, 0, 0, 0] \\ f([1, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1]) \to [0, 0, 1, 0, 0, 0] \\ f([1, 0, 0, 1, 1, 1], [0, 0, 0, 0, 0, 0]) \to [0, 0, 1, 0, 0, 0] \\ f([1, 0, 0, 1, 1, 1], [0, 0, 0, 0, 0, 1]) \to [0, 0, 1, 0, 0, 0] \end{array}$$

L'automate initial Δ_0 est donc donné par :

$$\begin{array}{ll} a \to [0, 0, 0, 0, 0, 0] & f([\_, \_, \_, \_, \_, \_], [\_, \_, \_, \_, \_, \_]) \to [0, 0, 0, 0, 0, 0] \\ b \to [0, 0, 0, 0, 0, 0] & g([\_, \_, \_, \_, \_, \_], [\_, \_, \_, \_, \_, \_]) \to [0, 0, 0, 0, 0, 0] \\ c \to [0, 0, 0, 0, 0, 0] & \end{array}$$

La première contrainte $a \subseteq X_4$ modifie par la règle d'inférence *(Compose)* la règle ayant a pour membre gauche en la remplaçant par : $a \to [0, 0, 0, \mathbf{1}, 0, 0]$.

De le même façon, de la contrainte suivante il résulte la règle de transition :

$$b \to [0, 0, 0, 0, \mathbf{1}, 0]$$

La troisième contrainte $f(X_4, X_5) \subseteq X_1$ signifie intuitivement que si pour une solution, un arbre τ_1 (resp. τ_2) appartient à l'interprétation de X_4 (resp. de X_5), alors l'arbre $f(\tau_1, \tau_2)$ appartient à l'interprétation de X_1. Par l'application de la règle *(Compose)*, cela se traduit dans (la méta-représentation de) l'automate par le remplacement de la méta-règle

$$f([\_, \_, \_, \_, \_, \_], [\_, \_, \_, \_, \_, \_]) \to [0, 0, 0, 0, 0, 0]$$

par les règles

$$f([\_,\_,\_,1,\_,\_],[\_,\_,\_,\_,1,\_]) \to [\mathbf{1},0,0,0,0,0]$$
$$f([\_,\_,\_,0,\_,\_],[\_,\_,\_,\_,1,\_]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,\_,\_],[\_,\_,\_,\_,0,\_]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,0,\_,\_],[\_,\_,\_,\_,0,\_]) \to [0,0,0,0,0,0]$$

Il faut remarquer que cette transformation, qui semble ici n'être qu'un pas de calcul, peut être vue comme l'application de la règle d'inférence *(Compose)* sur chacune des règles de transition méta-réprésentées dans l'automate ici décrit.

En considérant les contraintes suivantes de la même forme, on obtient alors l'automate suivant :

$$a \to [0,0,0,1,0,0]$$
$$b \to [0,0,0,0,1,0]$$
$$c \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,1,1]) \to [1,0,0,0,0,0]$$
$$f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,1,1]) \to [\mathbf{1},0,0,0,0,0]$$
$$f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,0,1]) \to [\mathbf{1},0,0,0,0,0]$$
$$f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,0,1]) \to [\mathbf{1},0,0,0,0,0]$$
$$f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,1,0]) \to [1,0,0,0,0,0]$$
$$f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,1,0]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,1,1]) \to [1,0,0,0,0,0]$$
$$f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,1,1]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,0,1]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,0,1]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,1,0]) \to [1,0,0,0,0,0]$$
$$f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,1,0]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,0]$$
$$f([\_,\_,\_,0,0,,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,0]$$
$$g([\_,\_,\_,\_,1,\_],[\_,1,\_,\_,1,\_]) \to [0,\mathbf{1},0,0,0,0]$$
$$g([\_,\_,\_,\_,1,\_],[\_,1,\_,\_,0,\_]) \to [0,\mathbf{1},0,0,0,0]$$
$$g([\_,\_,\_,\_,1,\_],[\_,0,\_,\_,1,\_]) \to [0,\mathbf{1},0,0,0,0]$$
$$g([\_,\_,\_,\_,1,\_],[\_,0,\_,\_,0,\_]) \to [0,0,0,0,0,0]$$
$$g([\_,\_,\_,\_,0,\_],[\_,1,\_,\_,1,\_]) \to [0,0,0,0,0,0]$$
$$g([\_,\_,\_,\_,0,\_],[\_,1,\_,\_,0,\_]) \to [0,0,0,0,0,0]$$
$$g([\_,\_,\_,\_,0,\_],[\_,0,\_,\_,1,\_]) \to [0,0,0,0,0,0]$$
$$g([\_,\_,\_,\_,0,\_],[\_,0,\_,\_,0,\_]) \to [0,0,0,0,0,0]$$

Examinons maintenant la contrainte $\{x \mid \exists z \forall y\, f(x,y) \in X_1 \wedge g(z,x) \in X_2\} \subseteq X_3$. On va donc tenter d'appliquer la règle *(Member)*. Les états accessibles dans l'automate à ce niveau du calcul sont exactement les états présents en membre droit de règle. On peut alors remarquer que quelque soit la règle de transition considérée, son membre gauche ne peut être utilisé comme valuation de la variable x de telle façon que la formule de l'expression d'appartenance ait pour modèle la structure $\mathsf{R}_\Delta^\top$ défini par l'automate. Ceci vient en particulier du fait que quelque soit le membre gauche considéré q (pour x), il n'existe pas d'état accessible q' tel que pour la règle de l'automate $f(q,q') \to q''$, q'' soit final pour F_1 (correspondant à la variable X_1). Par exemple, pour x valué à $[0,0,0,0,1,0]$, pour l'état accessible $[0,0,0,0,0,0]$, la règle de transition (méta-représentée dans l'automate) est

$$f([0,0,0,0,1,0],[0,0,0,0,0,0]) \to [0,0,0,0,0,0]$$

Cet état comme tous les autres ne peut convenir comme valuation pour x. Ainsi, après cette étape, l'automate demeure inchangé.

La contrainte suivante est $\{x \mid \exists y,\, y \in X_4 \vee x \in X_6\} \subseteq X_6$. L'étape précédente n'ayant pas modifié l'automate, tous les états présents en membre droit de règles sont toujours tous accessibles. De par la formule présente dans l'expression d'appartenance de cette contrainte, le pas de calcul est très simple : pour tout membre gauche valuant x, on peut trouver un état accessible appartenant à F_4 (en considérant le membre gauche de la règle $a \to q$ de l'automate).

Ainsi, par application de la règle *(Member)* à tous les membres gauches de règle, on obtient l'automate suivant :

$$
\begin{array}{ll}
a \to [0,0,0,1,0,1] & \\
b \to [0,0,0,0,1,1] & f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,1,0]) \to [1,0,0,0,0,1] \\
c \to [0,0,0,0,0,1] & f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,1,0]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,1,1]) \to [1,0,0,0,0,1] & f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,1,1]) \to [1,0,0,0,0,1] & f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,0,1]) \to [1,0,0,0,0,1] & g([\_,\_,\_,\_,1,\_],[\_,1,\_,\_,1,\_]) \to [0,1,0,0,0,1] \\
f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,0,1]) \to [1,0,0,0,0,1] & g([\_,\_,\_,\_,1,\_],[\_,1,\_,\_,0,\_]) \to [0,1,0,0,0,1] \\
f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,1,0]) \to [1,0,0,0,0,1] & g([\_,\_,\_,\_,1,\_],[\_,0,\_,\_,1,\_]) \to [0,1,0,0,0,1] \\
f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,1,0]) \to [0,0,0,0,0,1] & g([\_,\_,\_,\_,1,\_],[\_,0,\_,\_,0,\_]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,1,1,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,1] & g([\_,\_,\_,\_,0,\_],[\_,1,\_,\_,1,\_]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,0,0]) \to [0,0,0,0,0,1] & g([\_,\_,\_,\_,0,\_],[\_,1,\_,\_,0,\_]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,1,1]) \to [1,0,0,0,0,1] & g([\_,\_,\_,\_,0,\_],[\_,0,\_,\_,1,\_]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,1,1]) \to [0,0,0,0,0,1] & g([\_,\_,\_,\_,0,\_],[\_,0,\_,\_,0,\_]) \to [0,0,0,0,0,1] \\
f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,0,1]) \to [0,0,0,0,0,1] & \\
f([\_,\_,\_,0,0,\_],[\_,\_,\_,\_,0,1]) \to [0,0,0,0,0,1] &
\end{array}
$$

L'algorithme réitère alors le calcul en ré-inspectant toutes les contraintes. Pour toutes les premières contraintes de la forme $f(X_1,\ldots,X_m) \subseteq X$, l'application de la règle *(Compose)* laisse inchangé l'automate.

Réexaminons alors la contrainte $\{x \mid \exists z \forall y\, f(x,y) \in X_1 \wedge g(z,x) \in X_2\} \subseteq X_3$ avec la règle d'inférence correspondante à savoir *(Member)*. Pour la règle ayant a pour membre gauche, la valuation de la variable x avec le membre droit $[0,0,0,1,0,0]$ entraîne une nouvelle fois, le fait que la condition de la règle d'inférence n'est pas remplie. Par exemple, pour l'état accessible $[0,0,0,0,0,1]$, la règle $f([\_,\_,\_,1,0,\_],[\_,\_,\_,\_,0,\_,1]) \to [0,0,0,0,0,1]$ de l'automate a un membre droit non final pour F_1 impliquant le fait que le premier élément de la conjonction de l'expression d'appartenance n'est pas satisfait.

Inspectons maintenant la règle ayant b pour membre gauche et $[0,0,0,0,1,1]$ pour membre droit : remarquons tout de suite que tous les états en membre droit de règle (et donc, en particulier tous les états accessibles) sont finals pour F_6. Ainsi, par les règles de transition

$$
\begin{array}{l}
f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,1,1]) \to [1,0,0,0,0,1] \\
f([\_,\_,\_,0,1,\_],[\_,\_,\_,\_,0,1]) \to [1,0,0,0,0,1]
\end{array}
$$

on peut en conclure que pour une valuation associant $[0,0,0,0,1,1]$ à x et un état accessible quelconque à y, le premier élément de la conjonction est satisfait. De plus, de par la règle $g([\_,\_,\_,\_,1,\_],[\_,0,\_,\_,1,\_]) \to [0,1,0,0,0,1]$, on peut effectivement trouver un état accessible pour valuer z de telle manière que la seconde partie de la conjonction soit satisfaite ; il suffit de considérer $[0,0,0,0,1,1]$ lui-même. Ainsi, la règle *(Member)* est applicable pour cette valuation de x produisant la règle

$$b \to [0,0,1,0,1,1]$$

En continuant d'inspecter les autres règles, on s'aperçoit que c'est pour cette seule règle de transition que la règle d'inférence *(Member)* était applicable.

L'examen de la dernière contrainte ne modifie pas l'automate. De plus en réitérant une nouvelle fois l'inspection des contraintes, l'automate reste inchangé ce qui permet de conclure à

la fin de l'algorithme.

La plus petite solution du système de contraintes ensemblistes considéré peut être représentée sous la forme de la grammaire d'arbres suivante (les non-terminaux correspondant aux variables ensemblistes du système) :

$$\begin{aligned} X_1 &\Rightarrow f(a,b) \mid f(b,X_6) \\ X_2 &\Rightarrow g(b,b) \mid g(b,X_2) \\ X_3 &\Rightarrow b \\ X_4 &\Rightarrow b \\ X_5 &\Rightarrow b \\ X_6 &\Rightarrow a \mid b \mid c \mid f(X_6,X_6) \mid g(X_6,X_6) \end{aligned}$$

Nous allons dans la section suivante détailler des propriétés propres à l'extension que nous avons introduite. Le premier point abordera l'expressivité de cette extension et le second le lien entre la classe des contraintes définies généralisées et la classe des contraintes co-définies.

3.3 Propriétés de l'extension

Nous allons dans cette section exhiber des propriétés de la classe des contraintes ensemblistes définies généralisées qui sont propres à cette classe. Nous montrerons tout d'abord que l'ajout des expressions d'appartenance à la classe des contraintes définies augmente strictement le pouvoir d'expressivité de celle-ci en terme d'ensemble des solutions. Puis, nous verrons que cette classe permet de faire le lien entre contraintes définies et co-définies.

3.3.1 Expressivité

Dans cette section, nous allons nous intéresser à l'expressivité de la classe des contraintes définies généralisées vis-à-vis de la classe des contraintes définies et de la classe *(PNSC)* en termes d'ensembles des solutions. Nous allons dans un premier temps, restreindre l'extension proposée à ce que nous appellerons *contraintes définies avec expressions d'appartenance simples* et comparer l'expressivité de cette classe avec les classes définies et *(PNSC)*. Dans un second temps, nous étudierons l'expressivité de cette restriction par rapport à celle de la classe des contraintes définies généralisées.

Les contraintes définies avec expressions d'appartenance simples et leur expressivité

Les contraintes définies avec expressions d'appartenance simples [44] sont un cas particulier de contraintes définies généralisées pour lesquelles les formules du premier ordre monadiques étendues des expressions d'appartenance $\{x \mid \phi(x)\}$ sont limitées à la grammaire suivante :

$$\phi ::= t \in Sexp_A \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x\, \phi$$

où $Sexp_A$ est définie comme une expression pouvant apparaître en membre gauche d'une inclusion dans une contrainte définie généralisée (avec de telles expressions d'appartenance).

[44] Cette classe est apparue sous le nom des « Contraintes Définies avec Expressions d'Appartenance » dans [25]. Elle correspond à l'une des classes utilisées en analyse de programmes logiques (voir chapitre 5.

Comme pour les contraintes définies généralisées, il est possible de transformer un système de telles contraintes en un système aplati en préservant l'ensemble des solutions sur les variables du système initial. Un tel système aplati aura des contraintes de la forme :

$$f(X_1, \ldots, X_m) \subseteq X$$
$$\{x \mid \phi(x)\} \subseteq X$$
$$X \subseteq f(X_1, \ldots, X_m)$$
$$X \subseteq a^{\complement}$$

où $\phi(x)$ est une formule monadique positive ne comportant pas de quantificateur universel et ayant x pour unique variable libre du premier ordre.

Nous supposerons désormais que les systèmes de contraintes définies avec expressions d'appartenance simples considérés seront en forme aplatie. De plus, sans perte de généralité, nous supposerons que ϕ est en forme normale disjonctive prénexe, *i.e.* de la forme $\exists(\bigvee_i \bigwedge_j t_{ij} \in X_{ij})$.

Nous allons montrer ici que la classe des contraintes définies avec expressions d'appartenance simples et la classe *(PNSC)* ne sont pas comparables du point de vue de l'expressivité entre terme d'ensembles de solutions. L'expressivité de la classe des contraintes définies étant strictement plus faible que celle de la classe *(PNSC)*, nous pourrons en déduire que la classe des contraintes définies avec expressions d'appartenance simples est strictement plus expressive que la classe des contraintes définies en terme d'ensembles de solution.

Tout d'abord, compte tenu du fait que tout système satisfiable de contraintes définies avec expressions d'appartenance simples possède une plus petite solution et qu'il n'en est pas de même pour les systèmes de la classe *(PNSC)*, on en déduit que la classe des contraintes définies avec expressions d'appartenance simples ne peut être aussi expressive que la classe des contraintes *(PNSC)*.

Nous allons prouver qu'il en va de même pour la réciproque : pour ce faire, nous allons exhiber un système de contraintes définies avec expressions d'appartenance simples et montrer qu'aucun système de la classe *(PNSC)* ne peut avoir le même ensemble de solutions.

Considérons E, la contrainte définie (avec expressions d'appartenance simples) suivante

$$\{x \mid f(x,x) \in X\} \subseteq Y$$

et la solution $\mathcal{I}$ (définie pour les variables X et Y) par

$$\mathcal{I}(Y) = \varnothing \text{ et } \mathcal{I}(X) = \{f(g^n(a), g^p(a)) \mid n \neq p\}$$

où $g^n(a)$ est défini récursivement par $g^0(a) = a$ et $g^{i+1}(a) = g(g^i(a))$.

Soit $\mathcal{SC}$, un système de la classe *(PNSC)* ayant pour variables ensemblistes $\{X, Y\} \cup V$; soit $\bar{\mathcal{I}}$, une interprétation définie sur cet ensemble de variables et telle que $\bar{\mathcal{I}}(X) = \mathcal{I}(X)$ et $\bar{\mathcal{I}}(Y) = \mathcal{I}(Y)$ (impliquant que $\bar{\mathcal{I}}$ est solution de E). Nous allons montrer qu'on peut construire une interprétation $\bar{\mathcal{I}}'$ telle que si $\bar{\mathcal{I}}$ est solution de $\mathcal{SC}$, alors $\bar{\mathcal{I}}'$ est également solution de $\mathcal{SC}$, mais $\bar{\mathcal{I}}'$ n'étant pas solution de E.

Le système $\mathcal{SC}$ se décompose en deux sous-systèmes $\mathcal{SC}^+$ et $\mathcal{SC}^-$ tels que $\mathcal{SC}^+$ (resp. $\mathcal{SC}^-$) ne contient que des contraintes positives, *i.e.* d'inclusion (resp. négatives).

Il est très simple de montrer que pour toute solution $\mathcal{S}$ de $\mathcal{SC}^-$, il existe un entier N tel si on considère $\mathcal{S}'$ une interprétation telle que pour toute variable Z de $\{X, Y\} \cup V$,

$$\tau \in \mathcal{S}'(Z) \Delta \mathcal{S}(Z) \Longrightarrow |\tau| \geq N$$ [45]

[45] Pour deux ensembles A et B, $A \Delta B$ est la différence symétrique de A et B, *i.e.* $A \Delta B = (A \cap \complement B) \cup (\complement A \cap B)$.

alors $\mathcal{S}'$ est une solution de $\mathcal{SC}^-$.

En effet, pour une solution $\mathcal{S}$ de $\mathcal{SC}^-$, pour chacune des contraintes $Sexp_i \not\subseteq Sexp'_i$, il existe un plus petit terme clos τ_i appartenant à $\mathcal{S}(Sexp_i)$ et n'appartenant pas à $\mathcal{S}(Sexp'_i)$. Il suffit donc de choisir un entier N strictement supérieur à $|\tau_i|$, pour chacun des i. Pour une interprétation $\mathcal{S}'$ telle que $\mathcal{S}$ et $\mathcal{S}'$ ne se différencient que sur des termes clos de hauteur supérieure ou égale à N, pour chacune des contraintes, $\tau_i \in \mathcal{S}'(Sexp_i)$ et $\tau_i \notin \mathcal{S}'(Sexp'_i)$. Ainsi, $\mathcal{S}'$ est solution de $\mathcal{SC}^-$.

Puisque l'ensemble des variables $\{X, Y\} \cup V$ est fini, on peut trouver n_0 et p_0, deux entiers avec $n_0 > p_0 \geq N$ tel que $\forall Z \in \{X, Y\} \cup V, g^{n_0}(a) \in \bar{\mathcal{I}}(Z) \Leftrightarrow g^{p_0}(a) \in \bar{\mathcal{I}}(Z)$

Notons $\tilde{\tau}$, l'arbre qui pour un arbre τ correspond au remplacement de tous les sous-termes de τ de la forme $f(g^{n_0}(a), g^{n_0}(a))$ par le terme $f(g^{n_0}(a), g^{p_0}(a))$. Posons $\bar{\mathcal{I}}'$ l'interprétation définie par : pour tout terme clos τ,

$$\forall Z \in \{X, Y\} \cup V, \tau \in \bar{\mathcal{I}}'(Z) \Longleftrightarrow \tilde{\tau} \in \bar{\mathcal{I}}(Z)$$

On peut remarquer que $\bar{\mathcal{I}}'(Y) = \emptyset$ et que $f(g^{n_0}(a), g^{p_0}(a))$ appartenant à $\bar{\mathcal{I}}(X)$, par définition, $f(g^{n_0}(a), g^{n_0}(a)) \in \bar{\mathcal{I}}'(X)$. Ceci entraîne que $\bar{\mathcal{I}}'$ n'est pas solution de E.

Notons également que du fait de la remarque concernant les solutions de $\mathcal{SC}^-$ et le choix particulier de n_0 et p_0, l'interprétation $\bar{\mathcal{I}}'$ est solution de $\mathcal{SC}^-$. Nous allons maintenant démontrer que $\bar{\mathcal{I}}'$ est nécessairement solution de $\mathcal{SC}^+$ (et donc, de $\mathcal{SC}$).

Nous allons supposer que le système $\mathcal{SC}^+$ est dans une forme aplatie, *i.e.* que les expressions ensemblistes apparaissant en membre droit ou gauche des symboles d'inclusion sont de l'une des formes suivantes [46] :

$\top$	Z	$Z_1 \cup Z_2$
$\bot$	$h(Z_1, \ldots, Z_m)$	$Z_1 \cap Z_2$

On peut alors étendre la propriété définissant $\bar{\mathcal{I}}'$ aux expressions en forme aplatie :

Proposition 11 Pour toute expression $Sexp$ en forme aplatie :

$$\tau \in \bar{\mathcal{I}}'(Sexp) \Longleftrightarrow \tilde{\tau} \in \bar{\mathcal{I}}(Sexp)$$

Preuve :

Selon la forme de Sexp :

- *pour* $Sexp = \top$ *ou* $sexp = \bot$ *: évident.*
- *pour* $Sexp = Z$ *: immédiat par définition de* $\bar{\mathcal{I}}'$.
- *pour* $Sexp = Z_1 \cup Z_2$ *(resp.* $Sexp = Z_1 \cap Z_2$*) : par définition d'une interprétation ensembliste et de* $\bar{\mathcal{I}}'$*, on a les équivalences suivantes :*

$$\begin{aligned} & \tau \in \bar{\mathcal{I}}'(Z_1 \cup Z_2) \quad (\textit{resp. } \bar{\mathcal{I}}'(Z_1 \cap Z_2)) \\ \Leftrightarrow\ & \tau \in \bar{\mathcal{I}}'(Z_1) \textit{ ou (resp. et) } \tau \in \bar{\mathcal{I}}'(Z_1) \\ \Leftrightarrow\ & \tilde{\tau} \in \bar{\mathcal{I}}(Z_1) \textit{ ou (resp. et) } \tilde{\tau} \in \bar{\mathcal{I}}(Z_1) \\ \Leftrightarrow\ & \tilde{\tau} \in \bar{\mathcal{I}}(Z_1 \cup Z_2) \quad (\textit{resp. } \bar{\mathcal{I}}'(Z_1 \cap Z_2)) \end{aligned}$$

Cela signifie que les termes clos appartenant à $\mathcal{S}'(Z)$ et n'appartenant pas à $\mathcal{S}(Z)$ ou inversement ont tous une hauteur supérieure à N.

[46] Tout système de la classe *(PNSC)* peut être mis sous forme par ajout de variables auxiliaires et en notant que $Sexp_1 = \complement Sexp_2$ est équivalent à $\{Sexp_1 \cup Sexp_2 = \top, Sexp_1 \cap Sexp_2 = \bot\}$.

- *pour* $Sexp = h(Z_1, \ldots, Z_m)$, *selon le symbole* h :
  - *si* $h \neq f$ *ou* $h = f$, *mais pour un terme* τ *différent de* $f(g^{n_0}(a), g^{n_0}(a))$, *alors* $\tau \in \bar{\mathcal{I}}'(h(Z_1, \ldots, Z_m))$ *est équivalent à dire qu'il existe* $\tau_1, \ldots, \tau_m$ *tels que* $\tau = h(\tau_1, \ldots, \tau_m)$ *et pour tout* i, $\tau_i \in \bar{\mathcal{I}}'(Z_i)$. *Par définition de* $\bar{\mathcal{I}}'$, *cette dernière proposition est équivalente à dire que pour tout* i, $\tilde{\tau_i} \in \bar{\mathcal{I}}(Z_i)$. *Or puisque,* $\tau = h(\tau_1, \ldots, \tau_m)$, *on a* $\tilde{\tau} = h(\tilde{\tau_1}, \ldots, \tilde{\tau_m})$. *Ainsi, Il est équivalent de dire* $\tilde{\tau} \in \bar{\mathcal{I}}(h(Z_1, \ldots, Z_m))$.
  - *si* $h = f$ *et pour* $\tau = f(g^{n_0}(a), g^{n_0}(a))$,

$$
\begin{aligned}
& f(g^{n_0}(a), g^{n_0}(a)) \in \bar{\mathcal{I}}'(f(Z_1, Z_2)) \\
\Leftrightarrow\ & g^{n_0}(a) \in \bar{\mathcal{I}}'(Z_1) \text{ et } g^{n_0}(a) \in \bar{\mathcal{I}}'(Z_2) \quad \textit{(par définition d'une interprétation)} \\
\Leftrightarrow\ & g^{n_0}(a) \in \bar{\mathcal{I}}(Z_1) \text{ et } g^{n_0}(a) \in \bar{\mathcal{I}}(Z_2) \quad \textit{(par définition de } \bar{\mathcal{I}}') \\
\Leftrightarrow\ & g^{n_0}(a) \in \bar{\mathcal{I}}(Z_1) \text{ et } g^{p_0}(a) \in \bar{\mathcal{I}}(Z_2) \quad \textit{(par hypothèse sur } \bar{\mathcal{I}}) \\
\Leftrightarrow\ & f(g^{n_0}(a), g^{p_0}(a)) \in \bar{\mathcal{I}}(f(Z_1, Z_2) \quad \textit{(par définition d'une interprétation)}
\end{aligned}
$$

■

On en déduit alors

Proposition 12 Soit $Sexp_1 \subseteq Sexp_2$, une contrainte de $\mathcal{SC}^+$ (supposé en forme aplatie), si $\bar{\mathcal{I}}$ satisfait cette contrainte, alors il en va de même pour $\bar{\mathcal{I}}'$.

Preuve :

Pour tout terme clos τ, *si* $\tau \in \bar{\mathcal{I}}'(Sexp_1)$ *alors par définition de* $\bar{\mathcal{I}}'$, $\tilde{\tau} \in \bar{\mathcal{I}}(Sexp_1)$. *Puisque* $\bar{\mathcal{I}}$ *satisfait la contrainte,* $\tilde{\tau} \in \bar{\mathcal{I}}(Sexp_2)$. *Et donc, par la proposition 11,* $\tau \in \bar{\mathcal{I}}'(Sexp_2)$.

■

On en conclut donc que pour la contrainte définie avec expressions d'appartenance simple E, il ne peut exister de système de contraintes de la classe *(PNSC)* ayant exactement le même ensemble de solutions (restreintes aux variables ensemblistes de E).

Des expressions d'appartenance simples à la généralisation

Dans cette section, nous allons montrer en utilisant un argument topologique que la classe des contraintes définies généralisées est strictement plus expressive que la classe des contraintes définies avec expressions d'appartenance simples en terme d'ensemble des solutions.

Soit d_E, la distance[47] définie entre deux ensembles de termes clos A et B, par

$$
d_E(A, B) = \begin{cases} 0 & \textit{si } A = B \\ 2^{-inf\{|t| \,|\, t \in A \Delta B\}} & \textit{sinon} \end{cases}
$$

Cette distance est étendue en une distance d_S sur les interprétations ensemblistes définies sur un même ensemble V de variables ensemblistes, par

$$
d_S(\mathcal{I}, \mathcal{I}') = max\{d_E(\mathcal{I}(X), \mathcal{I}'(X)) \,|\, X \in V\}
$$

Nous allons montrer que l'ensemble des solutions d'un système de contraintes définies avec expressions d'appartenance simples est un fermé de la topologie $(S_V(\mathcal{I}), d_S)$, où $S_V(\mathcal{I})$ est l'ensemble des interprétations ensemblistes définies sur l'ensemble V.

[47] On peut vérifier qu'il s'agit effectivement d'une distance en montrant que pour tout A, $d_E(A) \geq 0$, que $d_E(A, B) = 0 \Leftrightarrow A = B$, que d_E est symétrique ($d_E(A, B) = d_E(B, A)$) et que d_E vérifie l'inégalité triangulaire $d_E(A, B) \leq d_E(A, C) + d_E(C, B)$.

Proposition 13 Soit $(\mathcal{S}_n)_{n\in\mathbb{N}}$, une suite de solutions d'un système $\mathcal{SC}$ de contraintes définies avec expressions d'appartenance simples (supposé en forme aplatie) convergente au sens de d_S vers une interprétation $\mathcal{S}$,

$$\mathcal{S} \text{ est une solution de } \mathcal{SC}$$

Preuve :

Supposons que $\mathcal{S}$ ne soit pas solution de $\mathcal{SC}$; il existe donc une contrainte $Sexp_1 \subseteq Sexp_2$ de $\mathcal{SC}$ et un terme clos τ tels que $\tau \in \mathcal{S}(Sexp_1)$ et $\tau \notin \mathcal{S}(Sexp_2)$. Selon les types de contraintes :

- *pour une contrainte de la forme $X \subseteq a^{\complement}$: Si $\tau \in \mathcal{S}(X)$ et $\tau \notin \mathcal{S}(a^{\complement})$, alors $\tau = a$. Puisque $(\mathcal{S}_n)_{n\in\mathbb{N}}$ est une suite de solutions, $\forall n$, $a \notin \mathcal{S}_n(X)$. Ainsi, pour tout n, $d_S(\mathcal{S}_n, S) \geq 1$.*

Pour les autres types de contrainte, nous allons appliquer le même raisonnement ; s'il existe un terme clos τ tel que $\tau \in \mathcal{S}(Sexp_1)$ et $\tau \notin \mathcal{S}(Sexp_2)$, alors il existe un entier n_0 tel que $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(Sexp_2)$. Or puisque $(\mathcal{S}_n)_{n\in\mathbb{N}}$ est une suite de solutions de $\mathcal{SC}$, on aurait $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(Sexp_1)$.

- *pour une contrainte de la forme $f(X_1, \ldots, X_m) \subseteq X$: soit τ un terme clos tel que $\tau \notin \mathcal{S}(X)$; la suite considérée étant convergente vers $\mathcal{S}$, posons n_0 le plus petit entier tel que $\forall n \geq n_0$, $d_S(\mathcal{S}_n, \mathcal{S}) < 2^{-|\tau|}$. Ceci implique donc que $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(X)$. Puisque $(\mathcal{S}_n)_{n\in\mathbb{N}}$ est une suite de solutions $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(f(X_1, \ldots, X_m))$. Or par hypothèse, $\tau \in \mathcal{S}(f(X_1, \ldots, X_m))$. On a donc*

$$\forall n \geq n_0,\ d_E(\mathcal{S}_n(f(X_1, \ldots, X_m)), \mathcal{S}(f(X_1, \ldots, X_m))) \geq 2^{-|\tau|}$$

impliquant $\forall n \geq n_0$, $max\{d_E(\mathcal{S}_n(X_i), \mathcal{S}(X_i)) \mid 1 \leq i \leq m\} \geq 2^{-(|\tau|-1)}$ avec $|\tau| \geq 1$ (puisque $\tau \in \mathcal{S}(f(X_1, \ldots, X_m))$).

- *pour une contrainte de la forme $X \subseteq f(X_1, \ldots, X_m)$: soit τ un terme clos tel que $\tau \notin \mathcal{S}(f(X_1, \ldots, X_m))$, selon la forme du terme τ :*
  - *si $\tau = g(\tau'_1, \ldots, \tau'_l)$ (avec $g \neq f$), posons n_0 le plus petit entier tel que $\forall n \geq n_0$, $d_S(\mathcal{S}_n, \mathcal{S}) < 2^{-|\tau|}$ avec $|\tau| \geq 0$.*
  - *si $\tau = f(\tau'_1, \ldots, \tau'_m)$, alors il existe un i tel que $\tau'_i \notin \mathcal{S}(X_i)$ avec $|\tau_i| = |\tau| - 1$ et $|\tau| \geq 1$; posons n_0 le plus petit entier tel que $\forall n \geq n_0$, $d_S(\mathcal{S}_n, \mathcal{S}) < 2^{-|\tau_i|}$ avec $|\tau_i| \geq 0$.*

  Dans les deux cas, on a donc $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(f(X_1, \ldots, X_m))$. Or, puisque cette suite est une suite de solution de $\mathcal{SC}$, $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(X)$. Or $\tau \in \mathcal{S}(X)$. Ainsi, $\forall n \geq n_0, d_E(\mathcal{S}_n(X), \mathcal{S}(X)) > 2^{-|\tau|}$.

- *pour une contrainte de la forme $\{x \mid \phi(x)\} \subseteq X$: soit τ un terme clos tel que $\tau \notin \mathcal{S}(X)$. La suite considérée étant convergente vers $\mathcal{S}$, posons n_0 le plus petit entier tel que $\forall n \geq n_0$, $d_S(\mathcal{S}_n, \mathcal{S}) < 2^{-|\tau|}$. Ceci implique que $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(X)$. La suite des $(\mathcal{S}_n)_{n\in\mathbb{N}}$ étant une suite de solutions, $\forall n \geq n_0$, $\tau \notin \mathcal{S}_n(\{x \mid \phi(x)\})$.*

  En considérant ϕ sous sa forme normale disjonctive, $\exists(\bigvee_i \bigwedge_j t_{ij} \in X_{ij})$, ceci permet d'affirmer que pour tout entier n ($n \geq n_0$) et pour toute substitution close σ,

$$\forall i,\ \exists j,\ (\sigma \circ [x \mapsto \tau])(t_{ij}) \notin \mathcal{S}_n(X_{ij}) \quad (1)$$

Or puisque $\tau \in \mathcal{S}(\{x \mid \phi(x)\})$, on peut affirmer qu'il existe une substitution close σ' telle que

$$\exists i',\ \forall j',\ (\sigma' \circ [x \mapsto \tau])(t_{i'j'}) \in \mathcal{S}(X_{i'j'}) \quad (2)$$

On peut donc en déduire que pour toute substitution σ vérifiant à la fois (1) et (2), il existe un i et un j tels que $(\sigma \circ [x \mapsto \tau])(t_{ij}) \notin \mathcal{S}_n(X_{ij})$ et $(\sigma \circ [x \mapsto \tau])(t_{ij}) \in \mathcal{S}(X_{ij})$. Or, ceci implique que $\forall n \geq n_0, d_S(\mathcal{S}_n, \mathcal{S}) \geq 1$.

Dans tous les cas précédemment inspectés, on peut déduire qu'il existe un réel k strictement positif et un entier n_0, tel que pour n supérieur ou égal à n_0, $d_S(\mathcal{S}_n, \mathcal{S}) \geq k$. Ceci impliquerait que la suite $(\mathcal{S}_n)_{n\in\mathbb{N}}$ n'est pas convergente, contredisant l'hypothèse.

■

Nous allons maintenant montrer que la proposition 13 n'est plus vraie lorsque l'on considère des systèmes de contraintes définies généralisées.

Considérons la contrainte définie généralisée suivante $\{x \mid \forall y,\ f(x,y) \in X\} \subseteq \bot$ et la suite de solutions $(\mathcal{S}_n)_{n\in\mathbb{N}}$ donnée par :

$$\forall n \in \mathbb{N},\ \mathcal{S}_n(X) = \{f(a,\tau) \mid \tau \in TERM(\Sigma) \text{ et } |\tau| \leq n\}$$

Cette suite est convergente au sens de d_S et sa limite est donnée par $\mathcal{S}(X) = \bigcup_{n\in\mathbb{N}} \mathcal{S}_n(X) = \{f(a,\tau) \mid \tau \in TERM(\Sigma)\}$. Or $\mathcal{S}$ n'est pas solution de la contrainte.

Nous allons voir dans la section suivante une autre propriété propre à l'extension que nous avons proposée et qui instaure un lien étroit entre contraintes définies (généralisées) et contraintes co-définies.

3.3.2 Lien avec la classe des contraintes co-définies

Nous allons voir dans cette section le lien existant entre les contraintes co-définies et les contraintes définies généralisées. Nous montrerons en fait comment le problème de satisfiabilité d'un système de contraintes co-définies [48] peut être encodé dans celui d'un système de contraintes définies généralisées et la relation existante entre les ensembles de solutions de ces deux systèmes. Ceci nous conduira à proposer une extension pour la classe de contraintes co-définies.

Dans [18], Charatonik et Podelski ont mentionné le fait que les classes des contraintes ensemblistes définies et co-définies, contrairement à ce que leurs noms pouvaient laisser croire, ne sont pas duales au sens de la relation d'inclusion. En effet, renverser le symbole d'inclusion pour une contrainte définie ($\subseteq$ en lieu et place de $\supseteq$) n'en faisait pas une contrainte co-définie. La réciproque est également vraie.

Cependant, il convient de remarquer qu'opérationnellement, ces deux classes de contraintes peuvent être considérées comme « duales » : nous avons dans ce chapitre proposé un algorithme de test de satisfiabilité pour un système de contraintes définies (généralisées) qui peut être assimilé à un algorithme calculant un plus petit point fixe. Nous verrons au chapitre suivant un algorithme répondant au problème de satisfiabilité pour une extension de la classe des contraintes co-définies, correspondant à un calcul de plus grand point fixe [49]. Cette remarque pouvait déjà être faite au regard de [25].

Comme il est mentionné dans [18], il n'existe pas non plus entre ces deux classes de « dualisation » par complémentation, principalement à cause des opérateurs de projection. Il est bien connu que les opérations ensemblistes d'intersection et d'union sont duales l'une de l'autre par complémentation [50], ce qui s'énonce généralement comme suit :

$$\complement(A \cup B) = (\complement A) \cap (\complement B)$$
$$\complement(A \cap B) = (\complement A) \cup (\complement B)$$

[48] Interprétées ici, contrairement au chapitre suivant, sur les ensembles de termes clos, *i.e.* d'arbres finis.

[49] Cet algorithme est conçu pour interpréter ces contraintes sur les ensembles d'arbres finis et infinis, cependant il peut être restreint aux ensembles d'arbres finis.

[50] C'est pourquoi ces deux opérateurs sont souvent appelés *opérateurs booléens*.

Cependant, une telle propriété n'existe pas pour les opérateurs de projection, ce qu'on pourrait résumer par « le complémentaire d'une projection n'est pas une projection ». Nous allons voir que les expressions d'appartenance offre une généralisation des projections appropriée, dans le sens où elles permettent une « dualisation » par complémentation.

Signalons le fait que pour une composition fonctionnelle, la complémentation s'opère comme suit : soit $Sexp$ une expression ensembliste égale à $f(Sexp_1, \ldots, Sexp_m)$, on a alors

$$\complement Sexp \equiv \bigcup_{g \neq f} g(\top, \ldots, \top) \cup \bigcup_{1 \leq i \leq m} f(\top, \ldots, (\complement Sexp_i), \ldots, \top)$$

où $\top$ est l'opérateur ensembliste nullaire dont la sémantique est l'ensemble $TERM(\Sigma)$.

Cette équivalence est donnée comme une conséquence de l'axiomatisation des contraintes ensemblistes proposée dans [20]. Elle peut être illustrée par : un terme τ appartient au complémentaire de $f(Sexp_1, \ldots, Sexp_m)$ ssi soit le symbole de tête de τ est différent du symbole de fonction f, soit son symbole de tête est f, *i.e.* $\tau = f(\tau_1, \ldots, \tau_m)$, et dans ce cas, au moins l'un des τ_i n'appartient pas à l'interprétation de l'expression $Sexp_i$ correspondante. Il est à noter que l'hypothèse selon laquelle la signature Σ est finie se révèle ici cruciale.

La complémentation pour les expressions d'appartenance est très naturelle, se basant uniquement sur des considérations ensembliste et logique. Les éléments de $\{x \mid \Phi(x)\}$ sous une certaine interprétation ensembliste $\mathcal{I}$ sont par définition les termes clos τ tels que $(\langle\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\rangle, [x \mapsto \tau]) \models \Phi(x)$. Les termes appartenant au complémentaire de cette expression sous l'interprétation $\mathcal{I}$ sont donc les termes τ tels que $(\langle\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\rangle, [x \mapsto \tau]) \not\models \Phi(x)$ ou de manière équivalente $(\langle\langle \mathsf{T}_\Sigma, \mathcal{I} \rangle\rangle, [x \mapsto \tau]) \models \neg\Phi(x)$. En considérant les équivalences suivantes,

$$\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2 \qquad \neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$$
$$\neg(\exists\varphi) \equiv \forall\neg\varphi \qquad \neg(\forall\varphi) \equiv \exists\neg\varphi$$

et le fait que $\neg(t \in Sexp)$ est équivalent à $t \in \complement sexp$, on peut en déduire qu'il existe une formule monadique positive Ψ tel que $\complement\{x \mid \Phi(x)\} \equiv \{x \mid \Psi(x)\}$.

Nous allons voir que de ces considérations, on peut montrer que le problème de satisfiabilité d'un système de contraintes co-définies peut se ramener à ce même problème mais pour un système de contraintes définies généralisées.

Des contraintes co-définies vers les contraintes définies généralisées

Rappelons que les contraintes co-définies sont des inclusions $Sexp_A \subseteq Sexp_B$, où :

$$Sexp_A ::= X \mid a \mid h(Sexp_A) \mid Sexp_A \cup Sexp_A$$
$$Sexp_B ::= X \mid f(Sexp_B, \ldots, Sexp_B) \mid Sexp_B \cup Sexp_B \mid Sexp_B \cap Sexp_B \mid f_k^{-1}(Sexp_B) \mid \bot$$

Un système de telles contraintes peut être mis en forme aplatie, c-à-d transformé en un système équivalent (du point de vue des solutions en se restreignant aux variables apparaissant

dans le système initial) dont les contraintes sont de l'une des formes suivantes[51] :

$$a \subseteq X$$
$$X \subseteq f(X_1, \ldots, X_m)$$
$$X \subseteq Y \cup Z$$
$$X \subseteq f_i^{-1}(Y)$$

L'idée de la transformation d'un système de contraintes ensemblistes co-définies (en forme aplatie) en un système de contraintes définies généralisées est basée sur l'équivalence suivante [52] :

$$A \subseteq B \Leftrightarrow (\complement B) \subseteq (\complement A)$$

Ainsi, en appliquant cette équivalence aux différentes formes possibles de contraintes apparaissant dans un système de contraintes ensemblistes co-définies en forme aplatie, on obtient les contraintes suivantes :

$$a^{\complement} \supseteq (\complement X)$$
$$(\complement X) \supseteq \bigcup_{g \neq f} g(\top, \ldots, \top) \cup \bigcup_{1 \leq i \leq m} f(\top, \ldots, (\complement X_i), \ldots, \top)$$
$$(\complement X) \supseteq (\complement Y) \cap (\complement Z)$$
$$(\complement X) \supseteq \{x_i \mid \forall_{-x_i} f(x_1, \ldots, x_m) \in (\complement Y)\}$$

La complémentation d'une expression utilisant un opérateur de projection $f_i^{-1}(Y)$ est simplement obtenue par le fait que cette expression est équivalente à $\{x_i \mid \exists_{-x_i} f(x_1, \ldots, x_m) \in Y\}$.

Puisque ces transformations appliquées sur un système de contraintes co-définies en forme aplatie produisent un système sémantiquement équivalent, on peut affirmer que l'ensemble des solutions est préservé. En particulier si le système co-défini n'a pas de solution, alors le système produit est lui aussi insatisfiable.

On peut alors remarquer que toutes les variables du système originel n'apparaissent plus dans le nouveau système que sous une forme complémentée $(\complement X)$ et que les symboles de complémentation n'apparaissent qu'appliqués à une variable ensembliste. Ainsi, en remplaçant simplement les expressions de la forme $\complement X$ par une nouvelle variable ensembliste $X^{\complement}$, alors le système ainsi obtenu est un système de contraintes définies généralisées.

Il est immédiat de voir que si le système de contraintes (contenant les variables complémentées) est insatisfiable, alors il en est de même pour le système de contraintes définies généralisées obtenu par remplacement de ces variables. De plus, pour toute solution $\mathcal{I}$ du système co-défini, on peut définir une unique solution $\mathcal{I}^{\complement}$ pour le système défini généralisé en posant $\mathcal{I}^{\complement}(X) = \complement\mathcal{I}(X)$.

En remarquant que la mise sous forme aplatie d'un système de contraintes ensemblistes co-définies peut être réalisée par un algorithme polynômial en temps (et donc, en espace) et qu'il en est de même pour la phase de complémentation, on peut déduire de cela un algorithme *EXPTIME* répondant au problème de satisfiabilité d'un système de contraintes ensemblistes co-définies interprété sur les ensembles de termes clos[53].

[51] Une telle transformation vient principalement du fait que $X \subseteq Y \cap Z \equiv \{X \subseteq Y, X \subseteq Z\}$ et $h(X) \subseteq Y \equiv X \subseteq h_1^{-1}(Y)$.

[52] Cette équivalence est une conséquence de la définition d'une interprétation ensembliste.

[53] Nous rappelons que ce problème est connu pour être *EXPTIME*-complet.

De plus, de par la définition de $\sqcap$ et $\sqcup$, nous pouvons en déduire que les solutions d'un système de contraintes ensemblistes co-définies forment un semi-treillis supérieur complet, dual par complémentation du semi-treillis inférieur complet des solutions du système de contraintes définies correspondant ; ceci implique notamment qu'un système de contraintes ensemblistes co-définies satisfiable admet une plus grande solution au sens de l'ordre sur les interprétations ensemblistes $\sqsubseteq$.

Une extension pour la classe des contraintes ensemblistes co-définies

Nous avons vu comment avec l'algorithme que nous avons proposé tester la satisfiabilité d'un système de contraintes ensemblistes co-définies (interprété sur les ensembles de termes clos). Deux questions restent cependant en suspens, à savoir :

- peut-on considérer une classe de contraintes englobant la classe des contraintes co-définies et telle que par complémentation, on puisse en tester la satisfiabilité avec l'algorithme que nous avons donné ?
- Si dualiser par complémentation un système de contraintes co-définies donne un système de contraintes définies généralisées, alors que donne la complémentation d'un système de contraintes définies (généralisé ou non) ?

C'est pour répondre à ces deux questions que nous proposons pour la classe des contraintes co-définies l'extension suivante : considérons les contraintes de la forme $Sexp_A \subseteq Sexp_B$ avec

$$Sexp_A ::= X \mid a \mid f(\top, \ldots, \top, Sexp_A, \top, \ldots, \top) \mid Sexp_A \cup Sexp_A \mid \top$$
$$Sexp_B ::= X \mid f(Sexp_B, \ldots, Sexp_B) \mid Sexp_B \cup Sexp_B \mid Sexp_B \cap Sexp_B \mid \bot \mid f_k^{-1}(Sexp_B) \mid \{x \mid \varphi(x)\}$$

L'expression $f(\top, \ldots, \top, Sexp_A, \top, \ldots, \top)$ signifie simplement que tous les arguments de f sont tous égaux à $\top$ à l'exception d'un seul qui est une expression $Sexp_A$.

Nous allons tout d'abord montrer que tout système de contraintes définies généralisées (en forme aplatie) peut se dualiser par complémentation (et renommage des variables complémentées) dans un système de cette classe.

Rappelons que les contraintes définies généralisées sont d'une des formes suivantes :

$$f(X_1, \ldots, X_m) \subseteq X$$
$$\{x \mid \varphi(x)\} \subseteq X$$
$$X \subseteq f(X_1, \ldots, X_m)$$
$$X \subseteq a^{\complement}$$

En dualisant par complémentation et renommage des variables complémentées ($\complement X$), on obtient des contraintes de la forme :

$$X \subseteq \bigcup_{g \neq f} g(\top, \ldots, \top) \cup \bigcup_{1 \leq i \leq m} f(\top, \ldots, X_i, \ldots, \top)$$
$$X \subseteq \{x \mid \psi(x)\}$$
$$\bigcup_{g \neq f} g(\top, \ldots, \top) \cup \bigcup_{1 \leq i \leq m} f(\top, \ldots, X_i, \ldots, \top) \subseteq X$$
$$a \subseteq X$$

Ces contraintes sont effectivement des contraintes de la classe que nous venons d'introduire.

En utilisant l'approche de dualisation par complémentation que nous avons présentée pour la classe des contraintes définies, on peut également en conclure que dualiser un tel système de contraintes produit un système de contraintes définies généralisées. On en déduit donc que :

- le problème de satisfiabilité d'un système de cette classe (interprété sur les ensembles de termes clos) est *EXPTIME*-complet.
- Un système de contraintes satisfiable de cette classe admet une plus grande solution et que l'ensemble de ces solutions forme un semi-treillis supérieur complet.

Nous appellerons cette classe de contraintes *la classe de contraintes ensemblistes co-définies généralisées* et un algorithme répondant au problème de satisfiabilité d'un système de cette classe cette fois-ci interprétée sur les ensembles d'arbres finis et infinis fera l'objet du chapitre suivant.

Chapitre 4

Les contraintes co-définies : une extension

Dans ce chapitre, nous allons proposer une extension de la classe des contraintes co-définies introduite dans [18][54]. Nous définirons syntaxiquement cette classe et préciserons la sémantique de cette extension, basée comme ce qui a été fait dans [18] sur une interprétation sur les ensembles d'arbres finis et infinis. Nous proposerons alors un algorithme répondant au problème de satisfiabilité d'un système de contraintes de cette extension avec la sémantique précédemment citée. Nous prouverons la correction de notre algorithme et donnerons une caractérisation de sa complexité. Finalement, nous étudierons deux cas de restriction de cette classe, l'une syntaxique et l'autre sémantique.

4.1 Les contraintes co-définies généralisées

Cette section définit syntaxiquement et sémantiquement l'extension de la classe des contraintes co-définies que nous allons considérer.

4.1.1 Syntaxe et sémantique

Définition 16 Une *contrainte co-définie généralisée* ψ_{CDSC} est définie par la syntaxe abstraire donnée à la figure 4.1. L'expression ensembliste$\{x \mid \Phi(x)\}$ est une expression d'appartenance construite à l'aide de $\Phi(x)$, une formule monadique positive étendue ayant x pour unique variable libre du premier ordre.

Nous nous sommes jusqu'ici principalement intéressés à une sémantique des contraintes ensemblistes basée sur l'algèbre des ensembles d'arbres finis PTr_Σ (ou de manière équivalente, sur l'algèbre des ensembles de termes clos). La sémantique que nous allons aborder pour cette classe de contraintes repose quant-à-elle sur l'algèbre des ensembles d'arbres finis et infinis $\mathsf{PTr}^\omega_\Sigma$. Une interprétation ensembliste est donc une application d'un ensemble de variables ensemblistes vers l'ensemble des ensembles d'arbres finis et infinis $\wp(\mathbb{T}\mathbb{R}^\omega_\Sigma)$. Cette interprétation est étendue aux expressions ensemblistes de la manière suivante :

- $\mathcal{I}(f(Sexp_1, \ldots, Sexp_m)) = \{f(\tau_1, \ldots, \tau_m) \mid \tau_1 \in \mathcal{I}(Sexp_1)$ *et* $\ldots$ *et* $\tau_m \in \mathcal{I}(Sexp_m)\}$[55]

[54] Une extension plus faible que celle que nous présenterons ici avait déjà été proposée dans [26].

[55] Cette sémantique est fixée par l'algèbre $\mathsf{PTr}^\omega_\Sigma$ en notant que la notation $f(\tau_1, \ldots, \tau_m)$ est un abus de notation pour $f^{\mathsf{PTr}^\omega_\Sigma}(\tau_1, \ldots, \tau_m)$.

$$
\begin{array}{rcl}
Sexp_A & ::= & X \mid a \mid f(\top, \ldots, \top, Sexp_A, \top, \ldots, \top) \mid Sexp_A \cup Sexp_A \mid \top \\
Sexp_B & ::= & X \mid f(Sexp_B, \ldots, Sexp_B) \mid Sexp_B \cup Sexp_B \mid Sexp_B \cap Sexp_B \mid \bot \mid \\
 & & f_k^{-1}(Sexp_B) \mid \{x \mid \Phi(x)\} \\
\Phi & ::= & t \in Sexp_B \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists y \Phi \mid \forall y \Phi \\
\psi_{CDSC} & ::= & Sexp_A \subseteq Sexp_B
\end{array}
$$

avec $X \in \mathcal{V}$, $f \in \Sigma$, $a \in \Sigma_0$ et $t \in TERM(\Sigma, \mathcal{X})$.

FIG. 4.1: Syntaxe abstraite des contraintes ensemblistes co-définies généralisées

- $\mathcal{I}(Sexp_1 \cup Sexp_2) = \mathcal{I}(Sexp_1) \cup \mathcal{I}(Sexp_2)$
- $\mathcal{I}(Sexp_1 \cap Sexp_2) = \mathcal{I}(Sexp_1) \cap \mathcal{I}(Sexp_2)$
- $\mathcal{I}(\top) = \mathbb{TR}_\Sigma^\omega$
- $\mathcal{I}(\bot) = \varnothing$
- $\mathcal{I}(f_k^{-1}(Sexp)) = \{\tau_k \mid f^{\mathsf{PTr}_\Sigma^\omega}(\tau_1, \ldots, \tau_m) \in \mathcal{I}(Sexp)\}$
- $\mathcal{I}(\{x \mid \Phi(x)\}) = \{\tau \mid (\langle\!\langle \mathsf{Tr}_\Sigma^\omega, \mathcal{I} \rangle\!\rangle, [x \mapsto \tau]) \models \Phi(x)\}$ où $[x \mapsto \tau]$ est une valuation du singleton $\{x\}$ dans $\mathbb{TR}_\Sigma^\omega$.

$(\langle\!\langle \mathsf{Tr}_\Sigma^\omega, \mathcal{I} \rangle\!\rangle, [x \mapsto \tau]) \models \Phi(x)$ est un abus de la notation introduite à la section 1.2.2.0 où la sémantique des formules atomiques $t \in Sexp$ (en considérant non plus une variable du second ordre, mais une expression ensembliste) est fixée pour une valuation ν (ayant pour domaine $Var_{\mathcal{X}}(t)$, l'ensemble des variables du premier ordre apparaissant dans t) et pour une interprétation ensembliste $\mathcal{I}$ par la valeur de vérité de l'expression $\lfloor t \rfloor_\nu^{\mathsf{Tr}_\Sigma^\omega} \in \mathcal{I}(Sexp)$, étendue aux formules monadiques du premier ordre (et en particulier, aux formules positives) comme présenté à la section 1.2.2.0.

Le but de ce chapitre est de proposer un algorithme répondant au problème de satisfiabilité d'un système de contraintes co-définies généralisées avec la sémantique que nous venons de fixer. Pour ce faire, nous allons sans perte de généralité, restreindre la syntaxe des contraintes que nous allons manipuler aux formes suivantes :

$$
\begin{array}{cc}
X \subseteq f(X_1, \ldots, X_m) & a \subseteq X \\
X \subseteq \{x \mid \varphi(x)\} & f(\top, \ldots, \top, Y, \top, \ldots, \top) \subseteq X
\end{array}
$$

où $\varphi(x)$ est une formule monadique positive.

On dira alors qu'un système ne comportant que de tels contraintes est *en forme aplatie.*

Tout système de contraintes co-définies généralisées peut être mis sous cette forme de telle manière que les solutions (sur les variables de ce système originel) soient préservées. Ceci peut être réalisé à l'aide du système d'inférence $\mathcal{S}^\omega$, présenté à la figure 4.2. Il est très simple de

$$\frac{\mathcal{SC} \cup \{Sexp \subseteq Sexp'\}}{\mathcal{SC} \cup \{Sexp \subseteq W \ , \ W \subseteq Sexp'\}}$$

$$\frac{\mathcal{SC} \cup \{Sexp \cup Sexp' \subseteq X\}}{\mathcal{SC} \cup \{Sexp \subseteq X \ , \ Sexp' \subseteq X\}} \qquad \frac{\mathcal{SC} \cup \{\top \subseteq X\}}{\mathcal{SC} \cup \bigcup_{f \in \Sigma} \{f(\top, \ldots, \top, W, \top, \ldots, \top) \subseteq W\}}$$

$$\frac{\mathcal{SC} \cup \{f(\top, \ldots, \top, Sexp, \top, \ldots, \top) \subseteq X\}}{\mathcal{SC} \cup \{f(\top, \ldots, \top, W, \top, \ldots, \top) \subseteq X \ , \ Sexp \subseteq W\}}$$

$$\frac{\mathcal{SC} \cup \{X \subseteq f(Sexp_1, \ldots, Sexp_m)\}}{\mathcal{SC} \cup \{X \subseteq f(W_1, \ldots, W_m) \ , \ W_1 \subseteq Sexp_1 \ , \ \ldots \ , \ W_m \subseteq Sexp_m\}}$$

$$\frac{\mathcal{SC} \cup \{ \ X \subseteq \{x \mid \varphi[t \in Sexp]\}}{\mathcal{SC} \cup \{ \ X \subseteq \{x \mid \varphi[t \in W]\} \ , \ W \subseteq Sexp\}} \qquad \frac{\mathcal{SC} \cup \{ \ X \subseteq Sexp \cap Sexp'\}}{\mathcal{SC} \cup \{ \ X \subseteq Sexp \ , \ X \subseteq Sexp'\}}$$

$$\frac{\mathcal{SC} \cup \{ \ X \subseteq Sexp \cup Sexp'\}}{\mathcal{SC} \cup \{ \ X \subseteq \{x \mid x \in W_1 \vee x \in W_2\} \ , \ W_1 \subseteq Sexp \ , \ W_2 \subseteq Sexp'\}}$$

$$\frac{\mathcal{SC} \cup \{ \ X \subseteq f_i^{-1}(Sexp) \ \}}{\mathcal{SC} \cup \{ \ X \subseteq \{x_i \mid \exists_{-x_i} f(x_1, \ldots, x_m) \in W\} \ , \ W \subseteq Sexp\}} \qquad \frac{\mathcal{SC} \cup \{X \subseteq \bot\}}{\mathcal{SC} \cup \{X \subseteq \tau \ , \ X \subseteq \tau'\}}$$

où :

- $W, W_1, \ldots, W_m$ sont des variables nouvelles (et ce à chaque application d'une règle d'inférence), *i.e.* n'apparaissant pas dans le système de contraintes sur lequel la règle est appliquée,
- $Sexp, Sexp_1, \ldots, Sexp_m$ ne sont pas des expressions ensemblistes égales à des variables de second ordre,
- $\tau, \tau' \in TERM(\Sigma)$ et $\tau \neq \tau'$.

FIG. 4.2: Le système d'inférence $\mathcal{S}^\omega$ de mise en forme aplatie

démontrer que tout système de contraintes co-définies généralisées $\mathcal{SC}$, normalisé par ce système d'inférence est bien en forme aplatie, que ces solutions (en ce qui concerne les variables de $\mathcal{SC}$) coïncident avec celle de $\mathcal{SC}$ et que cette normalisation peut se faire en temps polynômial par rapport à la taille de $\mathcal{SC}$, entraînant donc une augmentation polynômiale de la taille de celui-ci.

4.1.2 Outils

Comme nous l'avons fait dans le chapitre précédent, nous allons, avant de proposer l'algorithme testant la satisfiabilité d'un système de contraintes co-définies généralisées, présenter des notions qui permettront d'exposer cet algorithme et de prouver sa correction. La nature de ces notions est une nouvelle fois double, à savoir une extension des automates d'arbres infinis décrits à la section 1.3.3 et des considérations de logique.

Les automates d'arbres infinis à rang

Comme nous l'avions fait pour les automates ascendants d'arbres finis, nous allons ajouter à la définition des automates d'arbres infinis (sans condition d'acceptance) la notion de rang.

Définition 17 Un *automate d'arbres infinis de rang n (sans condition d'acceptance)* $\mathcal{A}$ est un quadruplet $(\Sigma, \mathcal{Q}, I, \Delta)$. Σ est une signature, $\mathcal{Q}$ est un ensemble fini d'états, $I = (I_1, \ldots, I_n)$ est un n-uplet d'ensembles d'états dits initiaux ($I_i \subseteq \mathcal{Q}$) et Δ est une relation sur $\bigcup_{i=0}^{amax} \Sigma_i \times \mathcal{Q}^i \times \mathcal{Q}$, dont les éléments appelés *règles de transition* sont notés :

$$f(q_1, \ldots, q_m) \to q \quad \textit{avec } q, q_1, \ldots, q_m \in \mathcal{Q} \textit{ et } f \textit{ un symbole d'arité } m$$

Les notions de calcul et d'accessibilité dans ces automates coïncident avec celles définies pour les automates d'arbres infinis (voir section 1.3.3). Nous rappelons simplement qu'un calcul r pour un arbre τ dans un automate $\mathcal{A}$ est une application de $dom(\tau)$, le domaine de l'arbre τ vers $\mathcal{Q}$, telle que pour toute position d de ce domaine,

$$f(r(d \cdot 1), \ldots, r(d \cdot m)) \to r(d) \quad \in \Delta$$

si $\tau(d)$ est égal au symbole f d'arité m.

L'absence de condition d'acceptance implique que les notions de calcul et de calcul réussi se confondent. Nous noterons $Run_{\mathcal{A}}(\tau)$ l'ensemble des calculs pour un arbre τ dans un automate $\mathcal{A}$.

Un état est dit accessible dans un automate $\mathcal{A}$ s'il existe un arbre τ, un calcul r pour τ et $\mathcal{A}$, une position d de $dom(\tau)$ tels que $r(d) = q$.

Le langage $L(\mathcal{A})$ reconnu par un automate d'arbres infinis de rang n (sans condition d'acceptance) est un n-uplet d'ensembles d'arbres finis et infinis $(L_1, \ldots, L_n)$ tel que τ appartient à L_i si et seulement si il existe un calcul r pour τ dans $\mathcal{A}$ tel que $r(\epsilon) \in I_i$.

Nous n'allons en fait considérer que les automates déterministes et complets (de manière ascendante[56]), à savoir que pour tout membre gauche de règles, il existe un unique membre droit tel que la règle de transition ainsi formée soit une règle de l'automate.

Nous pouvons remarquer notamment que cette notion de déterminisme n'implique pas l'unicité d'un calcul pour un arbre τ dans un tel automate. Ainsi,

[56]Nous précisons ici la mention « de manière ascendante », car les notions classiques de déterministe et de complétude associées aux automates d'arbres infinis correspondent en fait aux notions définies pour les automates d'arbres finis descendants.

Exemple 14 : *Considérons $\mathcal{A}$, un automate d'arbres infinis de rang 1 déterministe et complet « de manière ascendante» défini par $\Sigma = \{a/0,\ f/1\}$, $\mathcal{Q} = \{q_0, q_1, q_2, q_3\}$, $I = (\{q_0, q_1, q_2, q_3\})$ et*

$$\Delta = \begin{cases} a \to q_0 & f(q_2) \to q_2 \\ f(q_0) \to q_0 & f(q_3) \to q_3 \\ f(q_1) \to q_1 \end{cases}$$

Il est évident que pour l'arbre τ, tel que $dom(\tau) = \{1\}^\omega$ (l'ensemble des mots finis et infinis ne comportant que des 1) et pour toute position d de $dom(\tau)$, $\tau(d) = f$, les applications r_1 et r_2 de $dom(\tau)$ dans $\mathcal{Q}$ définies par : $\forall d \in dom(\tau), r_1(d) = q_1$ et $r_2(d) = q_2$ sont deux calculs différents pour τ dans $\mathcal{A}$.

On notera cependant qu'un arbre fini admet un calcul unique pour un tel automate.

De plus, nous n'allons considérer qu'une sous-classe de ces automates définie pour un ensemble de règles possédant une propriété particulière.

Monotonie des règles et pré-calculs

Nous considérons dans cette section des automates d'arbres infinis de rang n déterministes et complets (de manière ascendante) définis sur une signature Σ et un ensemble fini d'états $\mathcal{Q}$. Nous allons supposer de plus que l'ensemble des états $\mathcal{Q}$ est muni d'une relation d'ordre partiel $\preceq^{\mathcal{Q}}$, telle que $(\mathcal{Q}, \preceq^{\mathcal{Q}}, \sqcup^{\mathcal{Q}}, \sqcap^{\mathcal{Q}}, q_m, q_M)$ forme un treillis complet, où q_m, q_M désignent respectivement le plus petit et le plus grand élément du treillis.

Définition 18 On dit qu'un automate d'arbres infinis de rang n déterministe et complet (de manière ascendante) $\mathcal{A} = (\Sigma, \mathcal{Q}, I, \Delta)$ satisfait à la *propriété de monotonie des règles* si pour toute paire de règles $f(q_1, \ldots, q_m) \to q$ et $f(q'_1, \ldots, q'_m) \to q'$,

$$(\bigwedge_{1 \le i \le m} q_i \preceq^{\mathcal{Q}} q'_i) \Longrightarrow q \preceq^{\mathcal{Q}} q'$$

On peut noter qu'ajouter cette condition n'implique pas l'unicité du calcul pour un arbre dans un tel automate. Ainsi, en reprenant l'exemple 14 et en le complétant en posant que l'ensemble $\mathcal{Q}$ est ordonné de la manière suivante $\{q_0 \preceq^{\mathcal{Q}} q_1,\ q_0 \preceq^{\mathcal{Q}} q_2,\ q_1 \preceq^{\mathcal{Q}} q_3,\ q_2 \preceq^{\mathcal{Q}} q_3\}$, le contre-exemple est toujours valide. Cependant, nous allons voir que cette propriété supplémentaire permet de distinguer un calcul particulier. Pour ce faire, nous allons tout d'abord introduire la notion de pré-calcul.

Définition 19 Un pré-calcul pr pour un arbre τ dans un automate $\mathcal{A}$ d'arbres infinis de rang n déterministe et complet (de manière ascendante) est une application de $dom(\tau)$ dans $\mathcal{Q}$ telle que pour toute position d de $dom(\tau)$, si $\tau(d) = f$ et l'arité de f est m, alors pour la règle de transition de $\mathcal{A}$ [57] $f(pr(d \cdot 1), \ldots, pr(d \cdot m)) \to q$, on a $pr(d) \preceq^{\mathcal{Q}} q$.

On note $PRun_{\mathcal{A}}(\tau)$ l'ensemble des pré-calculs pour un arbre τ dans un automate $\mathcal{A}$. On peut tout de suite remarquer que tout calcul est un pré-calcul, *i.e.* $Run_{\mathcal{A}}(\tau) \subseteq PRun_{\mathcal{A}}(\tau)$.

Nous allons définir sur l'ensemble $PRun_{\mathcal{A}}(\tau)$ une relation d'ordre partiel $\preceq^{PR}$ par

Définition 20 Soient pr et pr', deux pré-calculs pour un arbre τ dans un automate d'arbres infinis de rang n déterministe et complet, $pr \preceq^{PR} pr'$ si et seulement si pour toute position d de $dom(\tau)$, $pr(d) \preceq^{\mathcal{Q}} pr'(d)$.

[57] L'existence d'une telle règle est garantie par les notions de déterminisme et de complétude « ascendants» de l'automate considéré.

En combinant la notion de pré-calcul et la propriété de monotonie des règles, on a alors pour tout arbre τ et tout automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles,

Proposition 14 $(PRun_{\mathcal{A}}(\tau), \preceq^{PR}, \sqcup^{PR}, \sqcap^{PR}, pr_m, pr_M)$ est un treillis complet.
Preuve :

Soit E un ensemble de pré-calculs pour τ dans $\mathcal{A}$ ($E \subseteq PRun_{\mathcal{A}}(\tau)$). Posons ξ et ξ' deux applications de $dom(\tau)$ dans $\mathcal{Q}$ définies par : $\forall d \in dom(\tau)$, $\xi(d) = \sqcup^{\mathcal{Q}}\{pr(d) \mid pr \in E\}$ et $\xi'(d) = \sqcap^{\mathcal{Q}}\{pr(d) \mid pr \in E\}$.
Montrons que :

- *ξ et ξ' sont des pré-calculs pour τ dans $\mathcal{A}$ ($\xi, \xi' \in PRun_{\mathcal{A}}(\tau)$).*
- *$\forall pr \in E$, $pr \preceq^{PR} \xi$ et $\xi' \preceq^{PR} pr$.*
- *$\forall pr' \in PRun_{\mathcal{A}}(\tau)$*

$$\begin{cases} (\forall pr \in E,\ pr \preceq^{PR} pr') \Longrightarrow \xi \preceq^{PR} pr' \\ \text{et} \\ (\forall pr \in E,\ pr' \preceq^{PR} pr) \Longrightarrow pr' \preceq^{PR} \xi' \end{cases}$$

i.e. par définition que $\xi = \sqcup^{PR}E$ et $\xi' = \sqcap^{PR}E$ sont des pré-calculs de τ dans $\mathcal{A}$.

Montrons tout d'abord que ξ est un pré-calcul pour τ dans $\mathcal{A}$, i.e. pour toute position d de $dom(\tau)$, on a $\xi(d) \preceq^{\mathcal{Q}} q$ pour la règle $f(\xi(d \cdot 1), \ldots, \xi(d \cdot m)) \to q$ de $\mathcal{A}$ (en ayant supposé que $\tau(d) = f$ d'arité m).
Considérons l'ensemble R des règles de transition $f(q'_1, \ldots, q'_m) \to q'$ tel que pour tout i, $q'_i = pr_i(d \cdot i)$ pour des pr_i appartenant à E.
Soit pr un élément de E ; puisque pr est un pré-calcul, il existe un état q' apparaissant en membre droit d'une règle de R, tel que $pr(d) \preceq^{\mathcal{Q}} q'$. On en déduit donc que :

$$\xi(d) = \sqcup^{\mathcal{Q}}\{pr(d) \mid pr \in E\} \preceq^{\mathcal{Q}} \sqcup^{\mathcal{Q}}\{q' \mid f(q'_1, \ldots, q'_m) \to q' \ \in R\}$$

De plus, par définition, $\xi(d \cdot i) = \sqcup^{\mathcal{Q}}\{pr(d \cdot i) \mid pr \in E\}$ et donc, pour toute règle de transition $f(q'_1, \ldots, q'_m) \to q'$ de R, $q'_i \preceq^{PR} \xi(d \cdot i)$. Ainsi, en utilisant la propriété de monotonie des règles de $\mathcal{A}$, on peut en déduire que pour tout règle $f(q'_1, \ldots, q'_m) \to q'$ de R, $q' \preceq^{PR} q$, i.e.

$$\sqcup^{\mathcal{Q}}\{q' \mid f(q'_1, \ldots, q'_m) \to q' \ \in R\} \preceq^{\mathcal{Q}} q$$

Ainsi, $\xi(d) \preceq^{PR} q$. Nous ne détaillons pas la preuve pour ξ' qui est en fait duale de celle de ξ.

Le second point est immédiat, puisque $\forall d \in dom(\tau)$, $pr(d) \preceq^{\mathcal{Q}} \sqcup^{\mathcal{Q}}\{pr(d) \mid pr \in E\} = \xi(d)$ et $\sqcap^{\mathcal{Q}}\{pr(d) \mid pr \in E\} = \xi'(d) \preceq^{\mathcal{Q}} pr(d)$.

Pour le troisième point, comme nous l'avons fait précédemment, nous ne détaillons que le cas de ξ, celle de ξ' étant dual.
Soit pr', un pré-calcul tel que $\forall pr \in E$, $pr \preceq^{PR} pr'$. Par définition, pour tout pr de E et toute position d, $pr(d) \preceq^{\mathcal{Q}} pr'(d)$. Ainsi, pour toute position d, $\sqcup^{\mathcal{Q}}\{pr(d) \mid pr \in E\} \preceq^{\mathcal{Q}} pr'(d)$, ce qui est équivalent à $\xi \preceq^{PR} pr'$.

■

Nous allons maintenant démontrer que tout pré-calcul (pour un arbre τ dans un automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles) peut être « transformé » en un calcul pour ce même arbre plus grand au sens de $\preceq^{PR}$.

Définition 21 Soit $Lift_{\mathcal{A}}$ une fonction (définie pour un automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles) ayant pour domaine l'ensemble de tous les pré-calculs $\bigcup_{\tau \in \mathbb{TR}^{\omega}_{\Sigma}} PRrun_{\mathcal{A}}(\tau)$. Elle est définie pour $pr^{\tau}_{\mathcal{A}}$, un pré-calcul pour un arbre τ dans $\mathcal{A}$ par : pour toute position d de $dom(\tau)$, $(Lift_{\mathcal{A}}(pr^{\tau}_{\mathcal{A}}))(d) = q$ si $\mathcal{A}$ contient la règle $f(pr^{\tau}_{\mathcal{A}}(d \cdot 1), \ldots, pr^{\tau}_{\mathcal{A}}(d \cdot m)) \rightarrow q$.

Il est évident par définition que $Lift_{\mathcal{A}}(pr^{\tau}_{\mathcal{A}})$ est une application de $dom(\tau)$ dans $\mathcal{Q}$. On peut de plus montrer que :

Proposition 15 Soit $pr^{\tau}_{\mathcal{A}}$, un pré-calcul pour un arbre τ dans un automate $\mathcal{A}$ satisfaisant à la propriété de monotonie des règles,

- $Lift_{\mathcal{A}}(pr^{\tau}_{\mathcal{A}}) \in PRun_{\mathcal{A}}(\tau)$, *i.e.* $Lift_{\mathcal{A}}$ associe à un pré-calcul pour τ un autre pré-calcul pour ce même arbre.
- $pr^{\tau}_{\mathcal{A}} \preceq^{PR} Lift_{\mathcal{A}}(pr^{\tau}_{\mathcal{A}})$.

Preuve :

Pour le premier point : posons $\xi = Lift_{\mathcal{A}}(pr^{\tau}_{\mathcal{A}})$ *et montrons que* ξ *est un pré-calcul de* τ *dans* $\mathcal{A}$. *Selon la position* d *de* $dom(\tau)$:

- *si* d *est une feuille : puisque* $pr^{\tau}_{\mathcal{A}}$ *est un pré-calcul alors* $pr^{\tau}_{\mathcal{A}}(d) \preceq^{PR} q$ *pour la règle* $\tau(d) \rightarrow q$ *de l'automate* $\mathcal{A}$. *Or, par définition* $\xi(d) = q$, *donc la proposition est bien vérifiée dans ce cas.*
- *si* d *n'est pas une feuille : puisque* $pr^{\tau}_{\mathcal{A}}$ *est un pré-calcul alors* $pr^{\tau}_{\mathcal{A}}(d) \preceq^{PR} q$ *pour la règle* $f(pr^{\tau}_{\mathcal{A}}(d \cdot 1), \ldots, pr^{\tau}_{\mathcal{A}}(d \cdot m)) \rightarrow q$ *de l'automate* $\mathcal{A}$ *(si* $\tau(d)$ *est le symbole* f *d'arité* m*). Par définition,* $\xi(d) = q$ *et pour tout* i, $pr^{\tau}_{\mathcal{A}}(d \cdot i) \preceq^{PR} \xi(i)$. *Puisque l'automate est déterministe et complet, il existe dans* $\mathcal{A}$ *une unique règle* $f(\xi(d \cdot 1), \ldots, \xi(d \cdot m)) \rightarrow q'$ *et comme* $\mathcal{A}$ *vérifie la propriété de monotonie des règles,* $\xi(d) \preceq^{PR} q'$.

Le second point est directement impliqué par les définitions de $Lift_{\mathcal{A}}$ *et de* $\preceq^{PR}$.

■

Définissons l'itération de la fonction $Lift_{\mathcal{A}}$ comme suit :

$$Lift^{n}_{\mathcal{A}}(pr) = \begin{cases} pr & si\ n = 0 \\ Lift_{\mathcal{A}}(Lift^{n-1}_{\mathcal{A}}(pr)) & si\ n > 0 \end{cases}$$

et définissons $Lift^{\infty}_{\mathcal{A}}(pr)$ comme :

$$\forall d \in dom(\tau),\ (Lift^{\infty}_{\mathcal{A}}(pr))(d) = \sup_{n \rightarrow \infty} ((Lift^{n}_{\mathcal{A}}(pr))(d))$$

Cette fonction $Lift^{\infty}_{\mathcal{A}}(pr)$ est bien définie car à chaque position d de $dom(\tau)$, pour l'itération de la fonction $Lift_{\mathcal{A}}$, la suite d'états étiquetant cette position est croissante (au sens $\preceq^{\mathcal{Q}}$) et finie (puisque $\mathcal{Q}$ est fini). On en déduit donc que pour chaque position d, la limite est finiment atteinte, *i.e.*

$$\forall d,\ \exists n_0, \forall n,\ n_0 \leq n,\ (Lift^{n}_{\mathcal{A}}(pr))(d) = (Lift^{n_0}_{\mathcal{A}}(pr))(d)$$

On peut alors démontrer que cette limite est un calcul pour l'arbre τ dans l'automate $\mathcal{A}$.

Proposition 16 Soit pr un pré-calcul d'un arbre τ dans un automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles,

$$Lift^{\infty}_{\mathcal{A}}(pr) \in Run_{\mathcal{A}}(\tau)$$

Preuve :

Il est suffisant de démontrer que pour tout $d \in dom(t)$,

$$t(d)((Lift^{\infty}_{\mathcal{A}}(pr))(d \cdot 1), \ldots, (Lift^{\infty}_{\mathcal{A}}(pr))(d \cdot m)) \rightarrow (Lift^{\infty}_{\mathcal{A}}(pr))(d) \in \Delta$$

Pour $d \in dom(t)$, posons n_0 le plus petit entier naturel tel que

$$\forall n,\ n > n_0,\ (Lift^{n}_{\mathcal{A}}(pr))(d) = (Lift^{n_0}_{\mathcal{A}}(pr))(d) = (Lift^{\infty}_{\mathcal{A}}(pr))(d)$$

et $\{n_1, .., n_m\}$ l'ensemble des entiers naturels tels que pour tout i,

$$\forall n,\ n > n_i,\ (Lift^{n}_{\mathcal{A}}(pr))(d.i) = (Lift^{n_i}_{\mathcal{A}}(pr))(d.i) = (Lift^{\infty}_{\mathcal{A}}(pr))(d.i)$$

Soit $n = max\{n_0, n_1, .., n_m\}$, $q = (Lift^{n}_{\mathcal{A}}(pr))(d) = (Lift^{\infty}_{\mathcal{A}}(pr))(d)$ et $q_i = (Lift^{n}_{\mathcal{A}}(pr))(d.i) = (Lift^{n}_{\mathcal{A}}(pr))(d.i)$. Par définition de $Lift_{\mathcal{A}}$,

$$t(d)(q_1, .., q_m) \rightarrow (Lift^{n+1}_{\mathcal{A}}(pr))(d) \in \Delta$$

Ainsi, puisque $Lift^{n+1}_{\mathcal{A}}(pr)(d) = Lift^{n}_{\mathcal{A}}(pr)(d) = Lift^{\infty}_{\mathcal{A}}(pr)(d)$, la proposition est vraie.

■

Signalons cependant que cette limite $Lift^{\infty}_{\mathcal{A}}(pr)$ n'est pas finiment atteinte, *i.e.* il n'existe pas nécessairement d'entier n tel que $Lift^{n}_{\mathcal{A}}(pr) = Lift^{\infty}_{\mathcal{A}}(pr)$. Par exemple,

Exemple 15 : *Considérons un automate $\mathcal{A}$ tel que $\Sigma = \{a/0, g/1, f/2\}$, $\mathcal{Q} = \{q_1, q_2\}$ et*

$$\Delta = \begin{cases} a \rightarrow q_1 & f(q_1, q_2) \rightarrow q_2 \\ g(q_2) \rightarrow q_2 & f(q_2, q_1) \rightarrow q_2 \\ g(q_1) \rightarrow q_1 & f(q_1, q_1) \rightarrow q_1 \\ f(q_2, q_2) \rightarrow q_2 & \end{cases}$$

$\mathcal{Q}$ est muni d'un ordre $\preceq^{\mathcal{Q}}$ tel que $q_2 \preceq^{\mathcal{Q}} q_1$

Considérons un arbre τ dont le domaine est défini comme l'ensemble $\{1\}^{\omega} \cup \{1^n \cdot 2 \cdot 1^j \mid j < n,\ n \in \mathbb{N}\} \cup \{1^n \cdot 2 \cdot 1^n \mid n \in \mathbb{N}\}$ et un pré-calcul $pr^{\mathcal{A}}_{\tau}$ pour τ dans $\mathcal{A}$, définis comme suit :

- *$\tau(d) = f$ et $pr^{\mathcal{A}}_{\tau}(d) = q_2$ si $d \in \{1\}^{\omega}$.*
- *$\tau(d) = g$ et $pr^{\mathcal{A}}_{\tau}(d) = q_2$ si $d \in \{1^n \cdot 2 \cdot 1^j \mid 1 \leq j < n,\ n \in \mathbb{N}\}$.*
- *$\tau(d) = a$ et $pr^{\mathcal{A}}_{\tau}(d) = q_1$ si $d \in \{1^n \cdot 2 \cdot 1^n \mid n \in \mathbb{N}\}$.*

Un fragment de τ, de $pr^{\mathcal{A}}_{\tau}$, $Lift(pr^{\mathcal{A}}_{\tau})$, $Lift^2(pr^{\mathcal{A}}_{\tau})$ et de $Lift^{\infty}(pr^{\mathcal{A}}_{\tau})$ sont donnés à la figure 4.3.

On a alors pour tout entier m,

$$Lift^{m}_{\mathcal{A}}(pr^{\mathcal{A}}_{\tau})(d) = \begin{cases} q_2 & si\ d \in \{1\}^{\omega} \\ q_2 & si\ d \in \{1^n \cdot 2 \cdot 1^j \mid 1 \leq j < m < n,\ n \in \mathbb{N}\} \\ q_1 & si\ d \in \{1^n \cdot 2 \cdot 1^j \mid 1 \leq n - m < j < n,\ n \in \mathbb{N}\} \end{cases}$$

On a donc pour tout entier m et la position $d = 1^n \cdot 2 \cdot 1^{n-(m+1)}$ (pour $m < n$) le fait que $(Lift^{m}_{\mathcal{A}}(\tau))(d) \neq (Lift^{m+1}_{\mathcal{A}}(\tau))(d)$ (pour $m < n$); ainsi pour tout entier m, $Lift^{m}_{\mathcal{A}}(pr^{\mathcal{A}}_{\tau}) \neq Lift^{m+1}_{\mathcal{A}}(pr^{\mathcal{A}}_{\tau})$.

Cependant, lorsque m tend vers l'infini, la limite est le calcul

$$Lift^{\infty}_{\mathcal{A}}(pr^{\mathcal{A}}_{\tau})(d) = \begin{cases} q_2 & si\ d \in \{1\}^{\omega} \\ q_1 & si\ d \in \{1^n \cdot 2 \cdot 1^j \mid 1 \leq j \leq n,\ n \in \mathbb{N}\} \end{cases}$$

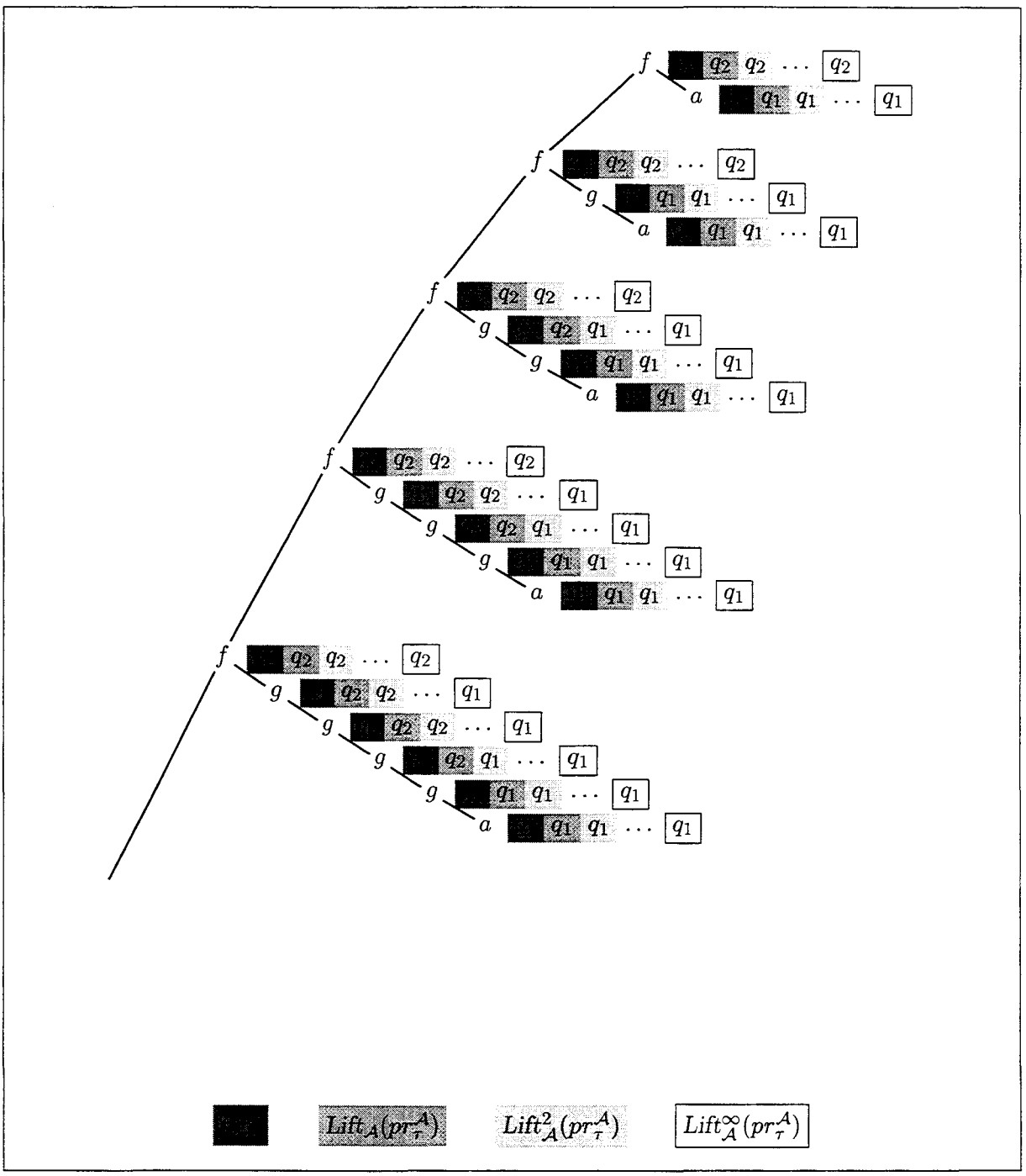

FIG. 4.3: Exemple de pré-calcul et d'application de la fonction $Lift_{\mathcal{A}}$

De cette dernière proposition, on déduit immédiatement que pour un automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles et pour tout arbre τ, il existe un plus grand calcul (au sens de $\preceq^{PR}$) pour τ dans $\mathcal{A}$.

Proposition 17

$$\exists \hat{r} \in Run_{\mathcal{A}}(\tau),\ \forall r \in Run_{\mathcal{A}}(\tau),\ r \preceq^{PR} \hat{r}$$

Preuve :

Par définition, tout calcul est un pré-calcul, on a par la proposition 14 que $\sqcup^{PR} Run_{\mathcal{A}}(\tau)$ *est un pré-calcul de* τ *dans* $\mathcal{A}$ *(plus grand que tous les calculs de* $Run_{\mathcal{A}}(\tau)$ *au sens de* $\preceq^{PR}$*). Du fait de la proposition 16, on peut en déduire que* $Lift^{\infty}_{\mathcal{A}}(\sqcup^{PR} Run_{\mathcal{A}}(\tau))$ *est un calcul de* τ *dans* $\mathcal{A}$ *et est plus grand que tout élément de* $Run_{\mathcal{A}}(\tau)$*. Ainsi,* $\hat{r} = Lift^{\infty}_{\mathcal{A}}(\sqcup^{PR} Run_{\mathcal{A}}(\tau))$.

■

Nous noterons $\hat{r}^{\mathcal{A}}_{\tau}$, le plus grand calcul pour un arbre τ dans un automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles.

On peut remarquer sur l'exemple 15 que le calcul limite n'est pas le plus grand calcul, puisque ce dernier est défini par $\forall d \in dom(\tau),\ \hat{r}^{\mathcal{A}}_{\tau}(d) = q_1$.

Nous terminons cette section en définissant la notion d'accessibilité maximale.

Définition 22 Soit $\mathcal{A}$ un automate d'arbres infinis déterministe et complet (au sens ascendant) à rang et vérifiant la propriété de monotonie des règles, un état q est dit *maximalement accessible* dans $\mathcal{A}$ si et seulement si il existe un arbre τ tel que $\hat{r}^{\mathcal{A}}_{\tau}(\epsilon) = q$.

Posons le problème suivant :

PROBLÈME D'ACCESSIBILITÉ MAXIMALE
Données : Soient $\mathcal{A}$ un automate d'arbres infinis déterministe et complet (au sens ascendant) à rang et vérifiant la propriété de monotonie des règles, et q un état de $\mathcal{A}$.
Question : q est maximalement accessible dans $\mathcal{A}$

Nous allons prouver que le problème d'accessibilité maximale pour un automate $\mathcal{A}$ d'arbres infinis déterministe et complet (au sens ascendant) à rang et vérifiant la propriété de monotonie des règles et pour un état q de $\mathcal{A}$ est polynômial dans la taille de $\mathcal{A}$.

Nous allons considérer un terme t de $TERM(\Sigma, \mathcal{X})$ comme un arbre sur Σ et $\mathcal{X}$ en assimilant les variables à des symboles d'arité 0 et donc n'apparaissant qu'aux feuilles de tels arbres.

Soit t un terme linéaire (*i.e.* chaque variable de t n'apparaît qu'à une seule position dans celui-ci) ayant pour variable $\{x_1, \ldots, x_l\}$. Pour des arbres $\tau_1, \ldots, \tau_l$ de $\mathbb{TR}^{\omega}_{\Sigma}$, nous notons $t[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]$, l'arbre de $\mathbb{TR}^{\omega}_{\Sigma}$ construit par greffe, $(..(t[d_1 \leftarrow \tau_1])..[d_l \leftarrow \tau_l])$, où pour tout i, d_i est l'unique position dans t de la variable x_i (t étant linéaire). On peut remarquer que l'ordre dans lequel s'effectuent les greffes successives peut être quelconque, puisque les positions d_i sont disjointes.

Soient $\mathcal{A}$ un automate d'arbres infinis déterministe et complet (au sens ascendant) à rang, t un terme de $TERM(\Sigma, \mathcal{X})$ et $\sigma : Var_{\mathcal{X}}(t) \mapsto \mathcal{Q}$, une application de l'ensemble des variables de t vers l'ensemble des états de l'automate $\mathcal{A}$. On notera $r^{\mathcal{A}}_{t,\sigma} : dom(t) \mapsto \mathcal{Q}$, l'unique application du domaine du terme t vers les états de l'automate telle que

- $r^{\mathcal{A}}_{t,\sigma}(d) = \sigma(x)$ si la variable x est à la position d dans le terme t,
- $r^{\mathcal{A}}_{t,\sigma}(d) = q$ si t à la position d n'est pas une variable et si $f(r^{\mathcal{A}}_{t,\sigma}(d \cdot 1), \ldots r^{\mathcal{A}}_{t,\sigma}(d \cdot m)) \rightarrow q$ est une règle de transition de $\mathcal{A}$.

On notera que cette application est bien définie puisque $\mathcal{A}$ est complet au sens ascendant et de manière unique puisque $\mathcal{A}$ est déterministe au sens ascendant.

On a alors

Lemme 3 q est maximalement accessible dans $\mathcal{A}$, un automate d'arbres infinis déterministe et complet (au sens ascendant) à rang vérifiant la propriété de monotonie des règles si et seulement si il existe un terme t de $TERM(\Sigma, \mathcal{X})$ linéaire tel que les deux affirmations suivantes sont valides :

- pour les variables $\{x_1, \ldots, x_l\}$ de t, il existe $\tau_1, \ldots, \tau_l$ tels que $r^{\mathcal{A}}_{t[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]}(\epsilon) = q$,
- pour l'application σ_M de $\{x_1, \ldots, x_l\}$ dans $\mathcal{Q}$ donnée par $\forall x \in \{x_1, \ldots, x_l\}$, $\sigma_M(x) = q_M$ (où q_M désigne l'élément maximum du treillis des états de $\mathcal{A}$), on a $r^{\mathcal{A}}_{t,\sigma_M}(\epsilon) = q$.

Preuve :

Montrons tout d'abord que les deux affirmations impliquent le fait que q soit maximalement accessible dans $\mathcal{A}$. Supposons ces deux affirmations valides et que q ne soit pas maximalement accessible. Posons $\tau = t[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]$.

Puisque $r^{\mathcal{A}}_{\tau}$ est un calcul pour τ dans $\mathcal{A}$, cela implique qu'il existe un autre calcul, que l'on peut lui supposer maximal, $\hat{r}^{\mathcal{A}}_{\tau}$ pour τ dans $\mathcal{A}$. On aurait en particulier pour ce calcul $r^{\mathcal{A}}_{\tau}(\epsilon) = q \prec^{\mathcal{Q}} \hat{r}^{\mathcal{A}}_{\tau}(\epsilon)$.

Soit σ : $\{x_1, \ldots, x_l\} \mapsto \mathcal{Q}$, définie par : pour tout $x_i \in \{x_1, \ldots, x_l\}$, $\sigma(x_i) = \hat{r}^{\mathcal{A}}_{\tau}(d_i)$, où d_i est la position de x_i dans t. Il est immédiat de voir que pour toute position d de $dom(t)$, $\hat{r}^{\mathcal{A}}_{\tau}(d) = r^{\mathcal{A}}_{t,\sigma}(d)$. On a alors pour tout x_i, $\sigma(x_i) \preceq^{\mathcal{Q}} r^{\mathcal{A}}_{\tau}(d_i) = \sigma_M(x_i) = q_M$. Alors, du fait de la propriété de monotonie des règles, on peut montrer par induction sur la structure de t que pour tout d de $dom(t)$, $r^{\mathcal{A}}_{t,\sigma}(d) = \hat{r}^{\mathcal{A}}_{\tau}(d) \preceq^{\mathcal{Q}} r^{\mathcal{A}}_{\tau}(d)$. En particulier pour la racine de t, $r^{\mathcal{A}}_{t,\sigma}(\epsilon) = \hat{r}^{\mathcal{A}}_{\tau}(\epsilon) \preceq^{\mathcal{Q}} r^{\mathcal{A}}_{\tau}(\epsilon) = q$. Ceci contredit l'hypothèse.

Montrons maintenant que si q est maximalement accessible, alors les deux affirmations sont vraies. Supposons que q soit maximalement accessible et que l'une des deux affirmations soient fausses, i.e. que pour tout terme linéaire t, soit

(i) il n'existe pas d'arbres $\tau_1, \ldots, \tau_l$ tel que $r^{\mathcal{A}}_{t[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]}(\epsilon) = q$, soit

(ii) $r^{\mathcal{A}}_{t,\sigma_M}(\epsilon) \neq q$.

Si la condition (i) est vraie, alors elle est vrai en particulier pour le terme réduit à une variable x. Il n'existerait alors pas d'arbre τ tel que $r^{\mathcal{A}}_{\tau}(\epsilon) = q$. Ceci impliquerait que q n'est pas un état accessible, donc pas maximalement accessible. Contradiction de l'hypothèse. La condition (i) est donc toujours fausse.

Supposons que la condition (ii) soit vraie. Puisque q est maximalement accessible il existe un arbre τ tel que $\hat{r}^{\mathcal{A}}_{\tau}(\epsilon) = q$. Selon cet arbre τ :

(a) τ est un arbre fini. Dans ce cas, pour le terme clos (et donc, linéaire) $t = \tau$, on a $r^{\mathcal{A}}_{t,\sigma_M}(\epsilon) = \hat{r}^{\mathcal{A}}_{\tau}(\epsilon) = q$. Contradiction de l'hypothèse.

(b) τ est un arbre infini. Soit $|d|$, la longueur d'une position définie par (i) $|\epsilon| = 0$, (ii) $|d.i| = |d| + 1$ pour $i \in \mathbb{N}$.

Considérons la suite (infinie) $(\xi_i)_{i \in \mathbb{N}}$ d'applications de $dom(\tau)$ dans $\mathcal{Q}$ données pour tout i, par

$$\xi_i(d) = \begin{cases} q_M & \text{si } |d| \leq i \\ q' & \text{si } \tau(d) = f \text{ et } f(\xi_i(d \cdot 1), \ldots, \xi_i(d \cdot m)) \rightarrow q' \text{ est une règle de } \mathcal{A} \end{cases}$$

Du fait de la monotonie des règles de l'automate, pour tout $d \in dom(\tau)$, $\xi_{i+1}(d) \preceq^{\mathcal{Q}} \xi_i(d)$. Or puisque $\mathcal{Q}$ est fini, pour tout $d \in dom(\tau)$, il existe un entier i tel que $\forall i' > i$, $\xi_{i'}(d) = \xi_i(d)$.

Notons que puisque par hypothèse, pour tout terme linéaire t, $r^{\mathcal{A}}_{t,\sigma_M}(\epsilon) \neq q$, on a : pour tout i, $q \prec^{\mathcal{Q}} \xi_i(\epsilon)$.
Posons l'application $\xi : dom(\tau) \mapsto \mathcal{Q}$ définie par : pour tout d de $dom(\tau)$,

$$\xi(d) = \xi_j(d) \text{ tel que } j = \min\{i \mid \forall i' > i,\ \xi_i(d) = \xi_{i'}(d)\}$$

Montrons maintenant que ξ est un calcul pour τ dans l'automate $\mathcal{A}$. Pour toute position d de $dom(\tau)$, considérons $\{j_1, \ldots, j_m\}$, les plus petits entiers tels que pour tout entier k, $\forall j'_k > j_k$, $\xi_{j_k}(d \cdot k) = \xi_{j'_k}(d \cdot k)$. Pour $j = \max\{j_1, \ldots, j_m\}$, on a pour tout $1 \leq k \leq m$, $\forall j' > j$, $\xi_j(d \cdot k) = \xi_{j'}(d \cdot k)$.
Ainsi, par définition de ξ, $\tau(d)(\xi(d \cdot 1), \ldots \xi(d \cdot m)) \rightarrow \xi(d)$ est une règle de transition de $\mathcal{A}$, donc ξ est un calcul pour τ dans l'automate $\mathcal{A}$.
Or, $q \prec^{\mathcal{Q}} \xi(\epsilon)$. Donc, ξ est un calcul pour τ dans $\mathcal{A}$ strictement plus grand que le calcul maximum $\hat{r}^{\mathcal{A}}_{\tau}$. Contradiction

■

Nous allons maintenant donner une caractérisation algorithmique de l'ensemble des états maximalement accessibles dans un automate $\mathcal{A}$ au moyen d'un opérateur $T_{\mathcal{A}} : \mathcal{Q} \times \mathcal{Q} \mapsto \mathcal{Q} \times \mathcal{Q}$.

Définition 23 Soit $\mathcal{A}$, un automate d'arbres infinis déterministe et complet (au sens ascendant) à rang vérifiant la propriété de monotonie des règles. Soit $T_{\mathcal{A}}$, l'opérateur défini sur $\mathcal{Q} \times \mathcal{Q}$ comme suit : pour $S_{\mathcal{Q}^2} \subseteq \mathcal{Q} \times \mathcal{Q}$,

$$(q, q') \in T_{\mathcal{A}}(S_{\mathcal{Q}^2}) \text{ ssi } \begin{cases} \text{il existe } f(q_1, \ldots, q_m) \rightarrow q \\ \text{et } f(q'_1, \ldots, q'_m) \rightarrow q' \text{ dans } \Delta \\ \text{tels que pour tout } i,\ (q_i, q'_i) \in S_{\mathcal{Q}^2} \end{cases}$$

On peut remarquer que l'opérateur $T_{\mathcal{A}}$ est monotone pour l'inclusion.

Lemme 4 *q* est maximalement accessible dans $\mathcal{A}$, un automate d'arbres infinis déterministe et complet (au sens ascendant) à rang vérifiant la propriété de monotonie des règles si et seulement si (q, q) appartient à $\bigcup_{n \in \mathbb{N}} T^n_{\mathcal{A}}(\{(q', q_M) \mid q' \in Reachable(\mathcal{A})\})$.
Preuve :

Soit $S^0_{\mathcal{Q}^2} = \{(q', q_M) \mid q' \in Reachable(\mathcal{A})\}$. Posons $T_{\mathcal{A},j}(S^0_{\mathcal{Q}^2}) = \bigcup_{1 \leq i \leq j} T^i_{\mathcal{A}}(S^0_{\mathcal{Q}^2})$.
Soit $|t|$, la taille d'un terme de $TERM(\Sigma, \mathcal{X})$, définie inductivement sur la structure de t par (i) $|t| = 0$ si $t \in \mathcal{X}$ (t est une variable), (ii) $|t| = 1$ si $t \in \Sigma_0$ (t est une constante), (iii) $|f(t_1, \ldots, t_m)| = 1 + \max(\{|t_i| \mid 1 \leq i \leq m\})$.

Montrons maintenant par induction sur n, que pour tout terme t de $TERM(\Sigma, \mathcal{X})$, si $|t| \leq n$ alors $(r^{\mathcal{A}}_{t[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]}(\epsilon), r^{\mathcal{A}}_{t,\sigma_M}(\epsilon)) \in T_{\mathcal{A},n}(S^0_{\mathcal{Q}^2})$ pour certains $\tau_1, \ldots, \tau_l$ de $\mathbb{T}\mathbb{R}^{\omega}_{\Sigma}$.
(a) pour $n = 0$, le terme t est une variable x. Alors, par définition, pour tout arbre τ de $\mathbb{T}\mathbb{R}^{\omega}_{\Sigma}$, $(r^{\mathcal{A}}_{t[x \leftarrow \tau]}(\epsilon), r^{\mathcal{A}}_{t,\sigma_M}(\epsilon)) = (r^{\mathcal{A}}_{\tau}(\epsilon), q_M)$ appartient à $T_{\mathcal{A},0}(S^0_{\mathcal{Q}^2}) = \{(q', q_M) \mid q' \in Reachable(\mathcal{A})\}$, puisque $r^{\mathcal{A}}_{\tau}(\epsilon)$ est nécessairement un état accessible.
(b) pour $n > 0$, le terme t est de la forme $f(t_1, \ldots, t_m)$ (incluant le cas où f est une constante). Par hypothèse d'induction, pour tout i, $(r^{\mathcal{A}}_{t_i[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]}(\epsilon), r^{\mathcal{A}}_{t_i,\sigma_M}(\epsilon))$ appartient à $T_{\mathcal{A},n-1}(S^0_{\mathcal{Q}^2})$. Alors, par définition de $T_{\mathcal{A}}$ et d'un calcul dans l'automate déterministe au sens ascendant, $(r^{\mathcal{A}}_{t[x_1 \leftarrow \tau_1, \ldots, x_l \leftarrow \tau_l]}(\epsilon), r^{\mathcal{A}}_{t,\sigma_M}(\epsilon))$ appartient à $T_{\mathcal{A},n}(S^0_{\mathcal{Q}^2})$.

On peut maintenant affirmer que si q est maximalement accessible, alors pour le terme t pour lequel les deux affirmations du lemme 3 sont valides, $(r^{\mathcal{A}}_{t[x_1\leftarrow\tau_1,\ldots,x_l\leftarrow\tau_l]}(\epsilon), r^{\mathcal{A}}_{t,\sigma_M}(\epsilon)) = (q,q)$ appartient à $T_{\mathcal{A},|t|}(S^0_{\mathcal{Q}^2})$ et donc à $\bigcup_{n\in\mathbb{N}} T^n_{\mathcal{A}}(S^0_{\mathcal{Q}^2})$.

Montrons maintenant que si (q,q') appartient à $\bigcup_{n\in\mathbb{N}} T^n_{\mathcal{A}}(S^0_{\mathcal{Q}^2})$, alors il existe un terme t et des arbres $\tau_1,\ldots,\tau_m$ tels que $(q,q') = (r^{\mathcal{A}}_{t[x_1\leftarrow\tau_1,\ldots,x_l\leftarrow\tau_l]}(\epsilon), r^{\mathcal{A}}_{t,\sigma_M}(\epsilon))$. Par induction sur l'entier n :
(a) pour $n=0$, $(q,q') = (q_r, q_M)$ avec q_r accessible. La propriété est trivialement vérifiée pour $t\in\mathcal{X}$ et pour $q_r = r^{\mathcal{A}}_{\tau}$, où τ est un arbre quelconque.
(b) pour $n>0$, (q,q') est obtenu par définition pour deux règle de transition de l'automate $f(q_1,\ldots,q_m)\rightarrow q$ et $f(q'_1,\ldots,q'_m)\rightarrow q'$ avec pour tout i, (q_i,q'_i) appartenant à $T^{n-1}_{\mathcal{A}}(S^0_{\mathcal{Q}^2})$. Par hypothèse d'induction pour tout i, il existe un terme t_i et des arbres $\tau^i_1,\ldots,\tau^i_{l^i}$ vérifiant la propriété. Ainsi, la propriété est vérifiée pour $t = f(t_1,\ldots,t_m)$ et pour les arbres $\tau^i_1,\ldots,\tau^i_{l^i}$. Ainsi, si (q,q) appartient à $\bigcup_{n\in\mathbb{N}} T^n_{\mathcal{A}}(S^0_{\mathcal{Q}^2})$, alors il existe un j tel que (q,q) appartient $T_{\mathcal{A},j}(S^0_{\mathcal{Q}^2})$. On peut ainsi affirmer qu'il existe un terme t (de taille inférieure à j) et des arbres $\tau_1,\ldots,\tau_m$ de $\mathbb{TR}^{\omega}_{\Sigma}$ tels que $(q,q) = (r^{\mathcal{A}}_{t[x_1\leftarrow\tau_1,\ldots,x_l\leftarrow\tau_l]}(\epsilon), r^{\mathcal{A}}_{t,\sigma_M}(\epsilon))$, ce qui implique que les deux conditions du lemme 3 sont satisfaites.

■

Théorème 14 Le problème d'accessibilité maximale est polynômial.
Preuve :

Notons tout d'abord que le test d'accessibilité d'un état dans un automate automate d'arbres infinis déterministe et complet (au sens ascendant) à rang est polynômial (ces automates étant des automates de Büchi sans état d'acceptance).
Répondre au problème d'accessibilité maximale pour un état revient à calculer l'ensemble $\bigcup_{n\in\mathbb{N}} T^n_{\mathcal{A}}(\{(q', q_M) \mid q'\in Reachable(\mathcal{A})\})$. Il très simple de voir que calculer cet ensemble (dont la taille est au plus quadratique dans la taille de l'automate) peut être réalisé par un algorithme polynômial dans la taille de $\mathcal{A}$.

■

Notons le fait que si un état q' est accessible, alors il existe un état q maximalement accessible tel que $q' \preceq^{\mathcal{Q}} q$. Nous noterons $Reach\_max(\mathcal{A})$, l'ensemble des états maximalement accessibles dans un automate $\mathcal{A}$ vérifiant la propriété de monotonie des règles.

Automates et logique

On peut associer, à un automate $\mathcal{A}$ d'arbres infinis déterministe et complet (dans le sens ascendant) de rang n et vérifiant la propriété de monotonie des règles, deux algèbres $\mathsf{A}_{\mathcal{A}}$ et $\hat{\mathsf{A}}_{\mathcal{A}}$, définies comme suit :

- $\mathsf{A}_{\mathcal{A}} = \langle \mathcal{Q}, \{f^{\mathcal{A}}\}_{f\in\Sigma}\rangle$ où $f^{\mathcal{A}}(q_1,\ldots,q_m) = q$ ssi $f(q_1,\ldots,q_m)\rightarrow q \in \Delta$.
- $\hat{\mathsf{A}}_{\mathcal{A}} = \langle Reach\_max(\mathcal{A}), \{f^{rchm(\mathcal{A})}\}_{f\in\Sigma}\rangle$ où $f^{rchm(\mathcal{A})}(q_1,\ldots,q_m) = q$ ssi $f(q_1,\ldots,q_m)\rightarrow q \in \Delta$.

Il est évident de par leur définition respective que $\hat{\mathsf{A}}_{\mathcal{A}}$ est une sous-algèbre de $\mathsf{A}_{\mathcal{A}}$.

Signalons de plus que du fait de l'unicité pour un arbre τ du plus grand calcul $\hat{r}^{\mathcal{A}}_{\tau}$ dans un tel automate $\mathcal{A}$, à chaque arbre τ on peut associer un unique état q tel que $\hat{r}^{\mathcal{A}}_{\tau}(\epsilon) = q$.

On peut donc en déduire qu'il existe un morphisme surjectif de $\mathsf{Tr}^{\omega}_{\Sigma}$ dans $\hat{\mathsf{A}}_{\mathcal{A}}$ associant à chaque arbre de $\mathbb{TR}^{\omega}_{\Sigma}$ l'unique élément de $Reach\_max(\mathcal{A})$, $\hat{r}^{\mathcal{A}}_{\tau}(\epsilon)$. Ce morphisme est noté $run_{\mathcal{A}}$.

Ces deux Σ-algèbres sont étendues respectivement pour un ensemble de variables du second ordre $\{V_1,\ldots,V_n\}$ en deux $(\Sigma,\{V_1,\ldots,V_n\})$-structures $\mathsf{R}_{\mathcal{A}}$ et $\hat{\mathsf{R}}_{\mathcal{A}}$, où $\mathsf{R}_{\mathcal{A}} = \langle\langle \mathsf{A}_{\mathcal{A}}, \mathcal{I}\rangle\rangle$ avec

$\forall i, \mathcal{I}(V_i) = I_i$ et $\hat{\mathsf{R}}_{\mathcal{A}} = \langle\langle \hat{\mathsf{A}}_{\mathcal{A}}, \mathcal{I}^{\hat{r}} \rangle\rangle$ avec $\forall i, \mathcal{I}^{\hat{r}}(V_i) = I_i \cap Reach\_max(\mathcal{A})$.

Comme dans le chapitre précédent, nous allons également ici considérer des formules monadiques $\top$-bornées (voir définition 13) et plus particulièrement les formules monadiques $\top$-bornées associées aux formules monadiques positives comme présentées dans la définition 14. On va supposer comme précédemment que si φ est une formule monadique positive, alors $\tilde{\varphi}$ dénotera la formule monadique $\top$-bornée associée.

Pour prendre en compte ces formules monadiques $\top$-bornées définies pour un ensemble de variables du second ordre $\{V_1, \ldots, V_n\}$ et une variable du second ordre particulière $\top$, nous étendons les structures $\mathsf{R}_{\mathcal{A}}$ et $\hat{\mathsf{R}}_{\mathcal{A}}$ (définies pour $\{V_1, \ldots, V_n\}$) en deux structures respectivement $\mathsf{R}^{\top}_{\mathcal{A}}$ et $\hat{\mathsf{R}}^{\top}_{\mathcal{A}}$ (définies pour $\{\top, V_1, \ldots, V_n\}$) en fixant pour chacune d'elles la sémantique de la variable $\top$ à $Reach\_max(\mathcal{A})$.

Logique

Nous terminons cette section par la remarque suivante :

Remarque 1 Soient R_A, R_B deux structures, Φ_1 et Φ_2 deux formules du premier ordre, et ν_A, ν_B deux valuations définies de $Var_{\mathcal{X}}(\Phi_1) \cup Var_{\mathcal{X}}(\Phi_2)$ respectivement dans les supports de R_A et R_B,

Si pour $e \in \{1, 2\}$, $((\mathsf{R}_A, \nu_A) \not\models \Phi_e) \Rightarrow ((\mathsf{R}_B, \nu_B) \not\models \Phi_e)$ alors

$$((\mathsf{R}_A, \nu_A) \not\models \Phi_1 \wedge \Phi_2) \Rightarrow ((\mathsf{R}_B, \nu_B) \not\models \Phi_1 \wedge \Phi_2)$$
$$((\mathsf{R}_A, \nu_A) \not\models \Phi_1 \vee \Phi_2) \Rightarrow ((\mathsf{R}_B, \nu_B) \not\models \Phi_1 \vee \Phi_2)$$

4.2 Une solution au problème de satisfiabilité

Nous allons dans cette section proposer un algorithme répondant au problème de satisfiabilité d'un système de contraintes co-définies généralisées (en forme aplatie) interprétées sur les ensembles d'arbres finis et infinis. Notre algorithme manipulera des automates complets et déterministes (ascendants) de rang n, qui de plus, vérifieront la propriété de monotonie des règles. Comme dans le chapitre précédent, cet algorithme sera présenté comme un système de règles d'inférence (appliquées avec une stratégie particulière) qui selon les contraintes, soit transforme un tel automate en un autre, soit produit la valeur spéciale $\Box$. Après avoir décrit cet algorithme, nous prouverons sa correction et donnerons sa complexité.

4.2.1 L'algorithme

Nous allons proposer un algorithme pour répondre au problème de satisfiabilité d'un système de contraintes co-définies généralisées supposé en forme aplatie et interprété sur les ensembles d'arbres finis et infinis. Cet algorithme est basé sur un ensemble $\mathcal{R}^{\omega}$ de règles d'inférence. Ces règles d'inférence transforment un automate d'arbres infinis déterministe et complet (de manière ascendante), soit en un automate de ce même type, soit en la valeur $\Box$. La forme des automates que nous considérons est définie en fonction des variables du système de contraintes ensemblistes et les états formeront un treillis fini complet. Les automates considérés devront de plus vérifier la propriété de monotonie des règles. Ceci imposera une stratégie particulière d'application des règles de $\mathcal{R}^{\omega}$, afin de préserver cette propriété.

Préliminaires

Le but de cette section est de présenter la forme des automates d'arbres infinis que l'algorithme manipulera, mais également de fixer quelques notations.

La forme des automates dépend des variables ensemblistes apparaissant dans le système de contraintes en forme aplatie considéré. Soient $\mathcal{SC}$ un système de contraintes co-définies généralisées en forme aplatie et $\{X_1, \ldots, X_n\}$, l'ensemble des variables ensemblistes apparaissant dans $\mathcal{SC}$.

Nous allons considérer les automates d'arbres infinis de rang n déterministes et complets (de manière ascendante) de la forme $\mathcal{A} = (\Sigma, \mathcal{Q}, I, \Delta)$ tels que Σ est la signature fonctionnelle sur laquelle est construit $\mathcal{SC}$, l'ensemble des états $\mathcal{Q} = \{0,1\}^n$, *i.e.* l'ensemble des n-uplets de 0 et de 1, et le n-uplet I d'ensembles d'états initiaux est égal à $(I_1, \ldots, I_n)$ tel que pour tout i $(1 \leq i \leq n)$ $q \in I_i$ si et seulement si la $i^{ème}$ composante de q est égale à 1.

L'ensemble des automates ainsi définis est noté $TA^{\omega}_{\mathcal{SC}}$.

Comme nous l'avons fait au chapitre précédent, à chaque variable ensembliste du système est associée une des composantes du langage reconnu par l'automate. L'idée motivant ce choix est de construire un automate comme précédemment décrit reconnaissant un langage $(L_1, \ldots, L_n)$ tel que si le système de contraintes $\mathcal{SC}$ considéré est satisfiable alors ce langage décrit la plus grande solution du système, *i.e.* une arbre τ appartient à L_i si et seulement si il existe une solution $\mathcal{I}$ de $\mathcal{SC}$ telle que $\tau \in \mathcal{I}(X_i)$. Lorsque le système de contraintes est insatisfiable, l'impossibilité de construire un tel automate sera détectée.

Pour un automate $\mathcal{A}$ de $TA^{\omega}_{\mathcal{SC}}$ reconnaissant un langage $(L_1, \ldots, L_n)$, nous noterons $\mathcal{I}_{\mathcal{A}}$, l'interprétation ensembliste pour les variables $\{X_1, \ldots, X_n\}$ par l'automate $\mathcal{A}$ définie par : pour tout X_i, $\mathcal{I}_{\mathcal{A}}(X_i) = L_i$ ou de manière équivalente, $\tau \in \mathcal{I}_{\mathcal{A}}(X_i)$ si et seulement si $\hat{r}^{\mathcal{A}}_{\tau}(\epsilon) \in I_i$.

Les éléments Σ, $\mathcal{Q}$ et I étant fixés pour les automates considérés, nous ne distinguerons donc plus un tel automate de son ensemble de règles de transition Δ. Nous noterons $switch_0(q, i)$, l'état q' identique à l'état q sauf (éventuellement) sur la $i^{ème}$ composante qui est égale à 0 pour q', *i.e.* $\forall j$, $q' \notin I_j \Leftrightarrow q \notin F_j \vee i = j$. Nous allons noter Δ_1, l'automate de $TA^{\omega}_{\mathcal{SC}}$ dit « universel », dont tous les membres droits de règles de transition sont égaux à $\{1\}^n$.

Le système d'inférence

L'idée sous-jacente à l'algorithme que nous proposons est similaire à celle présentée dans le chapitre précédent. Cet algorithme est basé sur un système d'inférence $\mathcal{R}^{\omega}$ transformant un automate de la classe $TA^{\omega}_{\mathcal{SC}}$, soit en un autre automate ce cette classe, soit en la valeur $\Box$. Partant de l'« automate universel » Δ_1, on applique les règles de ce système d'inférence jusqu'à obtenir soit un automate sur lequel plus aucune règle n'est applicable, soit la valeur $\Box$. Une application d'une règle d'inférence modifie en fonction d'une contrainte considérée du système une règle de transition de l'automate (qui est ici choisie de manière particulière) en remplaçant dans le membre droit de cette règle un 1 en un 0 (à l'aide de $switch_0$).

Ceci a en fait pour but de calculer la valeur $\Box$ si le système de contraintes $\mathcal{SC}$ est insatisfiable et, dans le cas contraire, un automate Δ_f de $TA^{\omega}_{\mathcal{SC}}$ reconnaissant un langage $(L_1, \ldots, L_n)$ tel que pour tout arbre τ, $\tau \in L_i$ (ou de manière équivalente, $\hat{r}^{\Delta_f}_{\tau}(\epsilon) \in I_i$) si et seulement si il existe une solution $\mathcal{I}$ de $\mathcal{SC}$ telle que $\tau \in \mathcal{I}(X_i)$.

Nous présentons maintenant le système d'inférence $\mathcal{R}^{\omega}$ défini pour la classe des automates $TA^{\omega}_{\mathcal{SC}}$ vérifiant la propriété de monotonie des règles ; ces règles d'inférence ont pour résultat soit

- *(Compose)*

$$\frac{\Delta \cup \{f(q_1,\dots,q_m) \to q\}}{\Delta \cup \{f(q_1,\dots,q_m) \to switch_0(q,i)\}} \text{ si } \begin{cases} X_i \subseteq g(X_{i_1},\dots,X_{i_l}) \in \mathcal{SC} \\ et \mid \text{- soit } g \neq f \\ \quad \text{- soit } f = g \text{ et } \exists k,\ q_k \notin I_{i_k} \end{cases}$$

- *(Clash)*

$$\frac{\Delta \cup \{a \to q\}}{\Box} \text{ si } \begin{cases} a \subseteq X_i \in \mathcal{SC} \text{ et} \\ q \notin I_i \end{cases}$$

- *(Decomp)*

$$\frac{\Delta \cup \{lhs \to q\}}{\Delta \cup \{lhs \to switch_0(q,i)\}} \text{ si } \begin{cases} g(\top,\dots,\top,X_i,\top,\dots\top) \subseteq X_j \in \mathcal{SC} \\ (X_i \text{ est le } k^{\text{ième}} \text{ fils de } f) \text{ et} \\ \exists\, g(q'_1,\dots,q'_m) \to q' \ tq \\ \quad \mid -q' \notin I_j \\ \quad \mid -q = q'_k \\ \quad \mid -\{q'_1,\dots,q'_m\}\backslash\{q'_k\} \in Reach\_max(\Delta \cup \{lhs \to q\}) \end{cases}$$

- *(Member)*

$$\frac{\Delta \cup \{lhs \to q\}}{\Delta \cup \{lhs \to switch_0(q,i)\}} \text{ si } \begin{cases} X_i \subseteq \{x \mid \varphi(x)\} \\ (\mathsf{R}^{\top}_{\Delta \cup \{lhs \to q\}}, [x \mapsto q]) \not\models \tilde{\varphi}(x) \end{cases}$$

FIG. 4.4: Le système d'inférence $\mathcal{R}^\omega$ du test de satisfiabilité

un autre automate de $TA^\omega_{\mathcal{SC}}$, soit la valeur spéciale $\Box$.

Les règles du système $\mathcal{R}^\omega$ sont données à la figure 4.4 : intuitivement, selon la contrainte, la règle d'inférence *(Compose)* signifie qu'un arbre τ de la forme $f(\tau_1,\dots,\tau_m)$ n'appartenant pas à $\mathcal{I}(g(X_{i_1},\dots,X_{i_l}))$, pour n'importe quelle solution $\mathcal{I}$ (soit parce que $f \neq g$, soit parce qu'il existe un k tel que $\tau_k \notin \mathcal{I}(X_{i_k})$), ne peut appartenir à $\mathcal{I}(X_i)$. Selon la contrainte dans la condition de la règle *(Clash)*, pour toute solution $\mathcal{I}$, a doit appartenir à $\mathcal{I}(X_i)$. Or, d'après la condition sur la règle de transition, pour toute solution $\mathcal{I}$, $a \notin \mathcal{I}(X_i)$. Ceci entraîne donc que le système de contraintes ne peut avoir de solution. La règle d'inférence *(Decomp)* traite le cas suivant : il existe un arbre de la forme $g(\tau_1,\dots,\tau_m)$ tel que pour toute solution $\mathcal{I}$, $g(\tau_1,\dots,\tau_m)$ n'appartient pas à $\mathcal{I}(X_j)$. On peut donc en déduire du fait de la contrainte que pour toute solution $\mathcal{I}$, $\tau_k \notin \mathcal{I}(X_i)$. La règle d'inférence *(Member)* doit être interprétée de la manière suivante : il existe un arbre τ tel que pour toute solution $\mathcal{I}$, $\tau \notin \mathcal{I}(\{x \mid \varphi(x)\})$ (*i.e.* $(\langle\!\langle \mathsf{Tr}^\omega_\Sigma, \mathcal{I}\rangle\!\rangle, [x \mapsto \tau]) \not\models \varphi(x)$) ; on peut donc en conclure que pour toute solution $\mathcal{I}$, $\tau \notin \mathcal{I}(X_i)$.

Le système d'inférence $\mathcal{R}^\omega$ est utilisé pour définir l'algorithme que nous proposons. Soient Δ, Δ' deux automates de $TA^\omega_{\mathcal{SC}}$, on notera $\Delta \vdash_{\mathcal{R}^\omega} \Delta'$ si et seulement si soit $\Delta = \Delta'$[58], soit Δ' est obtenu à partir de Δ et de l'application d'une des règles d'inférence de $\mathcal{R}^\omega$ pour une règle

[58] La relation $\vdash_{\mathcal{R}^\omega}$ est réflexive.

de transition $f(q_1, \ldots, q_m) \to q$ de Δ vérifiant les conditions de la règle d'inférence considérée et telle que cette règle soit une règle de membre gauche minimum (au sens de $\preceq^{\mathcal{Q}}$) [59] telle que l'application de la règle d'inférence entraîne une modification de la règle de transition (*i.e.* $q \in F_i$).

Posons de plus que $\Box \vdash_{\mathcal{R}^\omega} \Box$ et notons $\vdash^*_{\mathcal{R}^\omega}$ la clôture transitive de $\vdash_{\mathcal{R}^\omega}$.

Notre algorithme calcule une valeur e_f définie par $S_1 \vdash^*_{\mathcal{R}^\omega} e_f$ et telle que $\forall e\ e_f \vdash_{\mathcal{R}^\omega} e \Rightarrow e = e_f$, *i.e.* e_f est un point fixe de la relation $\vdash_{\mathcal{R}^\omega}$.

Cette définition de la relation $\vdash^*_{\mathcal{R}^\omega}$ n'a de sens que si le système de transition appliqué avec la stratégie précédemment décrite préserve la propriété de monotonie des règles dans l'automate calculé. En effet, la règle *(Member)* nécessite que la structure $\mathsf{R}^\top_\Delta$ soit définie ce qui est le cas uniquement dans un automate satisfaisant à la propriété de monotonie des règles. Ainsi, cette condition est vérifiée si $\vdash_{\mathcal{R}^\omega}$ transforme un automate de $TA^\omega_{\mathcal{SC}}$ vérifiant cette propriété en un autre automate la vérifiant également,*i.e.* $\vdash_{\mathcal{R}^\omega}$ préserve la propriété de monotonie des règles.

Pour prouver cette invariance, nous allons tout d'abord montrer le lemme suivant :

Lemme 5 Soient $\tilde{\varphi}(x_1, \ldots, x_m)$ une formule monadique $\top$-bornée (ayant $\{x_1, \ldots, x_m\}$ pour variables libre du premier ordre) et deux valuations (dans l'ensemble des états d'un automate Δ vérifiant la propriété de monotonie des règles) $\nu = [x_1 \mapsto q_1, \ldots, x_m \mapsto q_m]$ et $\nu' = [x_1 \mapsto q'_1, \ldots, x_m \mapsto q'_m]$ telles que $\forall j,\ q'_j \preceq^{\mathcal{Q}} q_j$,

$$(\mathsf{R}^\top_\Delta, \nu) \not\models \tilde{\varphi}(x_1, \ldots, x_m) \implies (\mathsf{R}^\top_\Delta, \nu') \not\models \tilde{\varphi}(x_1, \ldots, x_m)$$

Preuve :

Par induction sur la structure de la formule $\tilde{\varphi}(x_1, \ldots, x_m)$:

- *Si $\tilde{\varphi}(x_1, \ldots, x_m)$ est une formule atomique $t \in X_i$, ceci est équivalent à démontrer que*

$$\lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_\nu \notin I_i \Rightarrow \lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'} \notin I_i$$

Par induction sur la structure de t :

- *si t est une constante $t = a$, alors $\lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_\nu = \lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'}$.*
- *si t est une variable x_j, alors par hypothèse, $\lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'} \preceq^{\mathcal{Q}} \lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_\nu$ et $\lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_\nu \notin I_i$.*
- *si t est égal à $f(t_1, \ldots, t_l)$: par hypothèse d'induction, on a pour tout i et pour tout j $(1 \leq j \leq l)$:*

$$\lfloor t_j \rfloor^{\mathsf{R}^\top_\Delta}_\nu \notin I_i \Rightarrow \lfloor t_j \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'} \notin I_i$$

Par définition, ceci est équivalent à $\forall j$, $\lfloor t_j \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'} \preceq^{\mathcal{Q}} \lfloor t_j \rfloor^{\mathsf{R}^\top_\Delta}_\nu$. Or par définition d'un calcul,

$$f(\lfloor t_1 \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'}, \ldots \lfloor t_l \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'}) \to \lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'} \quad et \quad f(\lfloor t_1 \rfloor^{\mathsf{R}^\top_\Delta}_\nu, \ldots \lfloor t_l \rfloor^{\mathsf{R}^\top_\Delta}_\nu) \to \lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_\nu$$

sont deux règles de transition de Δ. Or celui-ci vérifiant la propriété de monotonie des règles, on a $\lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_{\nu'} \preceq^{\mathcal{Q}} \lfloor t \rfloor^{\mathsf{R}^\top_\Delta}_\nu$, ce qui implique l'expression souhaitée.

- *si $\tilde{\varphi} = \tilde{\varphi}_1 \wedge \tilde{\varphi}_2$ ou $\tilde{\varphi} = \tilde{\varphi}_1 \vee \tilde{\varphi}_2$, alors la proposition est immédiate par la remarque 1.*

[59] Pour deux règles $f(q_1, \ldots, q_m) \to q$ et $f(q'_1, \ldots, q'_m) \to q'$, la première a un plus petit membre gauche au sens de $\preceq^{\mathcal{Q}}$ ssi $\forall i,\ q_i \preceq^{\mathcal{Q}} q'_i$.

- *si* $\tilde{\varphi} = \exists y\,(y \in \top \wedge \tilde{\varphi}')$ *: par définition* $(\mathsf{R}^{\top}_{\Delta}, \nu) \not\models \tilde{\varphi}(x_1, \ldots, x_m)$ *est équivalent à dire que : pour tout état* q *maximalement accessible,* $(\mathsf{R}^{\top}_{\Delta}, \nu \circ [y \mapsto q]) \not\models \tilde{\varphi}'(y, x_1, \ldots, x_m)$.
  Par hypothèse d'induction, cela implique : pour tout état q *maximalement accessible,* $(\mathsf{R}^{\top}_{\Delta}, \nu' \circ [y \mapsto q]) \not\models \tilde{\varphi}'(y, x_1, \ldots, x_m)$, *ce qui est équivalent à* $(\mathsf{R}^{\top}_{\Delta}, \nu') \not\models \tilde{\varphi}(x_1, \ldots, x_m)$.
- *si* $\tilde{\varphi} = \forall y\,(y \in \top \Rightarrow \tilde{\varphi}')$ *: ce cas est similaire au cas précédent.*

■

On a alors

Proposition 18 Soit Δ un automate de TA^{ω}_{SC} vérifiant la propriété de monotonie des règles, si $\Delta \vdash_{\mathcal{R}^{\omega}} \Delta'$ (avec $\Delta' \neq \Box$), alors Δ' vérifie la propriété de monotonie des règles.

Preuve :

Le seul cas à examiner est le cas où $\Delta \vdash_{\mathcal{R}^{\omega}} \Delta'$ *avec* $\Delta \neq \Delta'$. *Soit* $f(q_1, \ldots, q_m) \to q$ *(avec* $q \in I_i$*) la règle de* Δ *transformée dans* Δ' *en* $f(q'_1, \ldots, q'_m) \to switch_0(q, i)$, *il est alors suffisant de prouver que pour toute autre règle* $f(q''_1, \ldots, q''_m) \to q''$ *de* Δ *(et donc de* Δ'*), on a* $\forall j,\ q''_j \preceq^{\mathcal{Q}} q_j \Rightarrow q'' \preceq^{\mathcal{Q}} switch_0(q, i)$.
Supposons que cela ne soit pas le cas, alors cela implique que $q'' \in I_i$. *On peut alors montrer que dans ce cas, l'hypothèse de minimalité du membre gauche de la règle transformée est fausse. Selon la règle d'inférence utilisée pour* $\Delta \vdash_{\mathcal{R}^{\omega}} \Delta'$ *:*

- *Pour la règle (Compose) :*
  - *si* $f \neq g$, *alors il existe une règle* $f(q''_1, \ldots, q''_m) \to q''$ *ayant un membre gauche plus petit que* $f(q_1, \ldots, q_m) \to q$, *pour laquelle la règle d'inférence peut s'appliquer.*
  - *si* $f = g$, *puisqu'il existe un* k *tel que* $q_k \in I_{i_k}$ *et que* $q''_{i_k} \preceq^{\mathcal{Q}} q_{i_k}$, *selon l'hypothèse de minimalité, la règle d'inférence aurait du s'appliquer pour* $f(q''_1, \ldots, q''_m) \to q''$.
- *Pour la règle (Decomp) : pour la règle de transition* $g(q^l_1, \ldots, q^l_m) \to q^l$ *utilisée dans la condition de la règle d'inférence, i.e. telle que* $q^l \notin I_j$, $q^l_k = q$ *et* $\{q^l_1, \ldots, q^l_m\} \backslash \{q^l_k\}$ *soient des états maximalement accessibles, considérons la règle* $g(q^l_1, \ldots, q'', \ldots, q^l_m) \to q^{l'}$ *dont le membre gauche est identique à la précédente à l'exception du* $k^{ème}$ *état du membre gauche qui est égal dans celle-ci à* q'' *(et non, à* q*). Or, on a bien* $q'' \in I_i$ *(par hypothèse), pour les états en membre gauche de cette règle* $\{q^l_1, \ldots, q^l_m\} \backslash \{q''\}$ *sont maximalement accessibles et* $q^{l'} \notin I_j$ *(puisque* $q'' \preceq^{\mathcal{Q}} q$ *et par monotonie des règles de* Δ, $q^{l'} \preceq^{\mathcal{Q}} q^l$ *et* $q^l \notin I_j$*). C'est donc cette règle* $f(q''_1, \ldots, q''_m) \to q''$ *qui aurait du être transformée, selon l'hypothèse de minimalité.*
- *Pour la règle (Member) : puisque* Δ *vérifie la propriété de monotonie des règles, on a* $q'' \preceq^{\mathcal{Q}} q$. *De par, le lemme 5 et la condition de la règle d'inférence, on a*

$$(\mathsf{R}^{\top}_{\Delta \cup \{lhs \to q\}}, [x \mapsto q'']) \not\models \tilde{\varphi}(x)$$

Ainsi, puisque la règle $f(q''_1, \ldots, q''_m) \to q''$ *a un membre gauche plus petit que celui de* $f(q_1, \ldots, q_m) \to q$ *et que* $q'' \in I_i$, *ceci contredit l'hypothèse de minimalité.*

■

Puisqu'il est évident que l'« automate universel » $\mathcal{S}_1$ vérifie la propriété de monotonie des règles, on peut en déduire que $\vdash_{\mathcal{R}^{\omega}}$ est bien définie et que tous les automates au cours du calcul vérifient cette propriété.

Nous allons dans la section suivante montrer la correction partielle et la complétude de l'algorithme que nous avons proposé, ainsi que sa complexité.

4.2.2 Correction - complexité

Nous allons commencer par montrer la correction partielle en prouvant que si l'algorithme produit un automate Δ_f, alors le langage reconnu par cet automate définit une solution du système de contraintes ensemblistes $\mathcal{SC}$.

Correction partielle

Soit Δ_f, l'automate de $TA^{\omega}_{\mathcal{SC}}$ ($\Delta_f \neq \Box$) produit par notre algorithme, *i.e.* $\forall e$, $\Delta_f \vdash_{\mathcal{R}^\omega} e \Rightarrow e = \Delta_f$; posons $\mathcal{I}_{\Delta_f}$, l'interprétation ensembliste associée à Δ_f et définie par $\forall i$, $\mathcal{I}_{\Delta_f}(X_i) = L_i$, en supposant que le langage reconnu par Δ_f est $(L_1, \ldots, L_n)$ ou de manière équivalente, pour tout i, $\tau \in \mathcal{I}_{\Delta_f}(X_i)$ si et seulement si $\hat{r}_\tau^{\Delta_f}(\epsilon) \in I_i$.

Théorème 15 $\mathcal{I}_{\Delta_f}$ est une solution de $\mathcal{SC}$.
Preuve :

Supposons que $\mathcal{I}_{\Delta_f}$ ne soit pas une solution ; il existe donc alors une contrainte de $\mathcal{SC}$ qui est violée. Selon la forme de cette contrainte :

- *pour une contrainte de la forme $X_i \subseteq g(X_{i_1}, \ldots, X_{i_l})$: si la contrainte est violée, cela signifie qu'il existe un arbre τ tel que $\tau \in \mathcal{I}_{\Delta_f}(X_i)$ (ceci implique que $\hat{r}_\tau^{\Delta_f}(\epsilon) \in I_i$) et $\tau \notin \mathcal{I}_{\Delta_f}(g(X_{i_1}, \ldots, X_{i_l}))$. Alors, selon la forme de τ :*
  - *si $\tau = f(\tau_1, \ldots, \tau_m)$ (avec $f \neq g$), alors il existe dans Δ_f une règle de transition $f(q_1, \ldots, q_m) \to q$ telle que $q \in I_i$.*
  - *si $\tau = g(\tau_1, \ldots, \tau_l)$, alors il existe un k tel que $\tau_k \notin \mathcal{I}_{\Delta_f}(X_{i_k})$. Donc, il existe dans Δ_f une règle de transition $g(q_1, \ldots, q_l) \to q$ telle que $q \in I_i$ et $q_{i_k} \notin I_{i_k}$.*
- *pour une contrainte de la forme $a \subseteq X_i$: si une telle contrainte est violée, cela signifie $a \notin \mathcal{I}_{\Delta_f}(X_i)$, i.e. $\hat{r}_a^{\Delta_f}(\epsilon) \notin I_i$. Cela implique que pour la règle $a \to q$ de Δ_f, on a $q \notin I_i$.*
- *pour une contrainte de la forme $g(\top, \ldots, \top, X_i, \top, \ldots \top) \subseteq X_j$: si la contrainte est violée, cela signifie qu'il existe un arbre τ de la forme $g(\tau_1, \ldots, \tau_l)$ tel que $\tau_k \in \mathcal{I}_{\Delta_f}(X_i)$ et $\tau \notin \mathcal{I}_{\Delta_f}(X_j)$. Cela implique qu'il existe une règle $g(q'_1, \ldots, q'_m) \to q'$ de Δ_f telle que $q' \notin I_j$, $q'_k \in I_i$ et $\{q'_1, \ldots, q'_m\} \backslash \{q'_k\}$ sont maximalement accessibles*
- *pour une contrainte de la forme $X_i \subseteq \{x \mid \varphi(x)\}$: si une telle contrainte est violée, cela signifie qu'il existe un arbre τ tel $\tau \in \mathcal{I}_{\Delta_f}(X_i)$ (ce qui implique qu'il existe une règle $lhs \to q$ dans Δ tel que $q = \hat{r}_\tau^{\Delta_f}(\epsilon) \in I_i$) et $\tau \notin \mathcal{I}_{\Delta_f}(\{x \mid \varphi(x)\})$, i.e.*

$$(\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \mathcal{I}_{\Delta_f} \rangle\!\rangle, [x \mapsto \tau]) \not\models \varphi(x)$$

Par la définition de $\hat{\mathsf{R}}_{\Delta_f}$ et la proposition 5, on en déduit que $(\hat{\mathsf{R}}_{\Delta_f}, [x \mapsto \hat{r}_\tau^{\Delta_f}(\epsilon)]) \not\models \varphi(x)$. Par la définition de $\hat{\mathsf{R}}^{\top}_{\Delta_f}$ et la proposition 6, cela implique que $(\hat{\mathsf{R}}^{\top}_{\Delta_f}, [x \mapsto \hat{r}_\tau^{\Delta_f}(\epsilon)]) \not\models \tilde{\varphi}(x)$. Enfin, par la proposition 7, $(\mathsf{R}^{\top}_{\Delta_f}, [x \mapsto \hat{r}_\tau^{\Delta_f}(\epsilon)]) \not\models \tilde{\varphi}(x)$.

Dans chacun des cas précédents, on peut voir qu'une règle d'inférence peut être appliquée (modifiant l'automate), entraînant le fait qu'on aurait $\Delta_f \vdash_{\mathcal{R}^\omega} e$ avec $e \neq \Delta_f$. Ceci contredit donc l'hypothèse initiale.

■

Complétude

La preuve de complétude de l'algorithme présenté se fait en deux étapes : la première montre qu'à toute solution du système $\mathcal{SC}$, on peut associer un unique pré-calcul dans chacun des automates calculés. On pourra, dans un deuxième temps, en déduire la complétude.

Considérons $\mathcal{I}$, une interprétation des variables ensemblistes apparaissant dans le système $\mathcal{SC}$; on définit l'application $pr^{\mathcal{I}}_{\tau}$ (de $dom(\tau)$ dans $\mathcal{Q}$) par : $pr^{\mathcal{I}}_{\tau}(d) = q$ si et seulement si $q \in I_i \Leftrightarrow \tau_{|d} \in \mathcal{I}(X_i)$. Nous allons montrer que si $\mathcal{I}$ est une solution de $\mathcal{SC}$, alors $pr^{\mathcal{I}}_{\tau}$ est un pré-calcul dans tout automate calculé, en montrant tout d'abord que cette propriété est préservée par $\vdash_{\mathcal{R}^\omega}$. Pour ce faire, nous prouvons deux lemmes.

Soient $\mathcal{I}$, une interprétation ensembliste définie pour des variables de $\mathcal{SC}$, étendue en une interprétation $\tilde{\mathcal{I}}$ fixant la sémantique de la variable additionnelle $\top$ à $\mathbb{TR}^{\omega}_{\Sigma}$, Δ un automate de $TA^{\omega}_{\mathcal{SC}}$ vérifiant la propriété de monotonie des règles et tel que pour tout arbre τ, $pr^{\mathcal{I}}_{\tau}$ est un pré-calcul dans Δ,

Lemme 6 $\forall \tau_1, \ldots, \tau_m \in \mathbb{TR}^{\omega}_{\Sigma}$, $\forall f(q_1, \ldots, q_m) \to q \in \Delta$

$$\forall i,\ pr^{\mathcal{I}}_{\tau_i}(\epsilon) \preceq^{\mathcal{Q}} q_i \Longrightarrow pr^{\mathcal{I}}_{f(\tau_1,\ldots,\tau_l)}(\epsilon) \preceq^{\mathcal{Q}} q$$

Preuve :

Par monotonie des règles de Δ, il existe dans cet automate une règle $(pr^{\mathcal{I}}_{\tau_1}(\epsilon), \ldots, pr^{\mathcal{I}}_{\tau_l}(\epsilon)) \to q'$ telle que $q' \preceq^{\mathcal{Q}} q$ et puisque pour tout arbre τ, $pr^{\mathcal{I}}_{\tau}$ est un pré-calcul dans Δ, alors $pr^{\mathcal{I}}_{f(\tau_1,\ldots,\tau_l)}(\epsilon) \preceq^{\mathcal{Q}} q'$.

■

On en déduit alors

Lemme 7 $\forall \tau_1, \ldots, \tau_l$,

$$(\mathsf{R}^{\top}_{\Delta}, [y_1 \mapsto pr^{\mathcal{I}}_{\tau_1}(\epsilon), \ldots, y_l \mapsto pr^{\mathcal{I}}_{\tau_l}(\epsilon)]) \not\models \tilde{\varphi}(y_1, \ldots, y_l)$$
$$\Longrightarrow$$
$$(\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [y_1 \mapsto \tau_1, \ldots y_l \mapsto \tau_l) \not\models \tilde{\varphi}(y_1, \ldots, y_l)$$

Preuve :

Posons tout d'abord $\nu = [y_1 \mapsto pr^{\mathcal{I}}_{\tau_1}(\epsilon), \ldots, y_l \mapsto pr^{\mathcal{I}}_{\tau_l}(\epsilon)]$ et $\sigma = [y_1 \mapsto \tau_1, \ldots y_l \mapsto \tau_l]$.
Par induction sur la structure de la formule $\tilde{\varphi}$ (ou φ) :

- *si $\tilde{\varphi}$ est une formule atomique $t \in X_i$: par induction sur la structure de t :*
  - *si t est une constante ($t = a$), alors $(\mathsf{R}^{\top}_{\Delta}, \nu) \not\models a \in X_i$ est équivalent à dire que pour la règle $a \to q$ de Δ, $q \notin I_i$. Or, $pr^{\mathcal{I}}_{a}$ est un pré-calcul pour a dans Δ, donc $pr^{\mathcal{I}}_{a}(\epsilon) \preceq^{\mathcal{Q}} q$. Ceci implique $a \notin \mathcal{I}(X_i)$, c-à-d, $(\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, \sigma) \not\models a \in X_i$.*
  - *si t est une variable (du premier ordre) ($t = x$) : par définition, $(\mathsf{R}^{\top}_{\Delta}, \nu \circ [x \mapsto pr^{\mathcal{I}}_{\tau}]) \not\models x \in X_i$ est équivalent à $pr^{\mathcal{I}}_{\tau}(\epsilon) \notin I_i$, i.e. $\tau \notin \mathcal{I}(X_i)$. Ainsi, $(\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, \sigma \circ [x \mapsto \tau]) \not\models x \in X_i$.*
  - *si t est un terme de la forme $f(t_1, \ldots, t_m)$: par hypothèse d'induction, on a : pour tout i et tout e,*

$$(\mathsf{R}^{\top}_{\Delta}, \nu) \not\models t_i \in X_e \Longrightarrow (\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, \sigma) \not\models t_i \in X_e$$

Soit q_i, l'état tel que $\forall e$, $q_i \notin I_e \Leftrightarrow (\mathsf{R}^{\top}_{\Delta}, \nu) \not\models t_i \in X_e$. Ainsi, en posant $\tau'_i = \lfloor t_i \rfloor_{\sigma}^{\mathsf{Tr}^{\omega}_{\Sigma}}$, l'hypothèse d'induction peut se reformuler comme : $pr^{\mathcal{I}}_{\tau'_i}(\epsilon) \preceq^{\mathcal{Q}} q_i$.

Pour la règle $f(q_1, \ldots, q_m) \to q$ de Δ, par le lemme 6, $pr^{\mathcal{I}}_{\tau}(\epsilon) \preceq^{\mathcal{Q}} q$ pour $\tau = f(\tau'_1, \ldots, \tau'_m)$. Ceci peut être reformulé comme : pour tout i,

$$(\mathsf{R}^{\top}_{\Delta}, \nu) \not\models f(t_1, \ldots, t_m) \in X_i \Longrightarrow (\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, \sigma) \not\models f(t_1, \ldots, t_m) \in X_i$$

- *pour une formule $\tilde{\varphi}$ de la forme $\tilde{\varphi}_1 \wedge \tilde{\varphi}_2$ ou $\tilde{\varphi}_1 \vee \tilde{\varphi}_2$: immédiat par la remarque 1.*
- *pour une formule $\tilde{\varphi}$ de la forme $\exists x(x \in \top \wedge \tilde{\varphi}')$: par hypothèse de récurrence, on a : pour tout arbre τ,*

$$(\mathsf{R}^{\top}_{\Delta}, [x \mapsto pr^{\mathcal{I}}_{\tau}(\epsilon), y_1 \mapsto pr^{\mathcal{I}}_{\tau_1}(\epsilon), \ldots, y_l \mapsto pr^{\mathcal{I}}_{\tau_l}(\epsilon)]) \not\models \tilde{\varphi}(x, y_1, \ldots, y_l)$$
$$\Longrightarrow$$
$$(\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle, [x \mapsto \tau, y_1 \mapsto \tau_1, \ldots y_l \mapsto \tau_l) \not\models \tilde{\varphi}(x, y_1, \ldots, y_l)$$

puisque $pr^{\mathcal{I}}_{\tau}(\epsilon)$ (resp. τ) appartient à $\mathcal{Q}$ (resp. $\mathbb{TR}^{\omega}_{\Sigma}$), l'interprétation de $\top$ dans $\mathsf{R}^{\top}_{\Delta}$ (resp. $\langle\!\langle \mathsf{Tr}^{\omega}_{\Sigma}, \tilde{\mathcal{I}} \rangle\!\rangle$), on en déduit immédiatement la proposition dans ce cas.

- *pour une formule $\tilde{\varphi}$ de la forme $\exists x(x \in \top \Rightarrow \tilde{\varphi}')$: la preuve est similaire au cas précédent.*

■

On peut alors montrer l'invariance par l'application de la relation $\vdash_{\mathcal{R}^{\omega}}$ du pré-calcul défini à partir d'une solution du système,

Proposition 19 Soient $\mathcal{I}$ une solution de $\mathcal{SC}$ et Δ un automate de $TA^{\omega}_{\mathcal{SC}}$ (vérifiant la propriété de monotonie des règles) et $\mathcal{I}$ tel que pour tout arbre τ, $pr^{\mathcal{I}}_{\tau}$ est un pré-calcul pour τ dans Δ. Si $\Delta \vdash_{\mathcal{R}^{\omega}} \Delta'$ (avec $\Delta' \neq \Box$), alors pour tout arbre τ', $pr^{\mathcal{I}}_{\tau'}$ est un pré-calcul pour τ' dans Δ'.

Preuve :

Une nouvelle fois, nous ne considérons que le cas où $\Delta' \neq \Delta$, le cas où ils sont égaux étant évidemment trivial.

Par définition d'un pré-calcul, il est suffisant de montrer que pour tout arbre τ et toute position d de $dom(\tau)$, si $\tau(d)(pr^{\mathcal{I}}_{\tau}(d \cdot 1), \ldots, pr^{\mathcal{I}}_{\tau}(d \cdot m)) \to q \in \Delta$ et $\tau(d)(pr^{\mathcal{I}}_{\tau}(d \cdot 1), \ldots, pr^{\mathcal{I}}_{\tau}(d \cdot m)) \to q' \in \Delta'$, alors $pr^{\mathcal{I}}_{\tau}(d) \preceq^{\mathcal{Q}} q'$ (en supposant que l'arité de $\tau(d)$ est m). Si les règles sont les mêmes ($q = q'$), alors cette proposition est vraie. Le seul cas à considérer est donc le cas où ces règles de transition sont différentes, i.e. de par le système d'inférence, $q' = switch_0(q, i)$. Ceci revient à prouver pour la règle d'inférence utilisée pour $\Delta \vdash_{\mathcal{R}^{\omega}} \Delta'$ que $pr^{\mathcal{I}}_{\tau}(d) \notin I_i$.

Selon la règle d'inférence :

- *pour la règle (Compose) : il est suffisant de montrer que $\tau_{|d} \notin \mathcal{I}(g(X_{i_1}, \ldots, X_{i_l}))$. En effet, puisque $\mathcal{I}$ est une solution, cette affirmation implique que $\tau_{|d} \notin \mathcal{I}(X_i)$ et donc que, $pr^{\mathcal{I}}_{\tau}(d) \notin I_i$. Selon la forme de $\tau_{|d}$:*
  - *si $\tau(d) = f$ et $f \neq g$, alors $\tau_{|d} \notin \mathcal{I}(g(X_{i_1}, \ldots, X_{i_l}))$, par définition d'une interprétation ensembliste.*
  - *si $\tau(d) = g$, alors par les conditions de la règle d'inférence, il existe un k tel que $pr^{\mathcal{I}}_{\tau}(d \cdot k) \notin I_{i_k}$. Donc par définition de $pr^{\mathcal{I}}_{\tau}$, $\tau_{|d \cdot k} \notin \mathcal{I}(X_{i_k})$. Ainsi, $\tau_{|d} \notin \mathcal{I}(g(X_{i_1}, \ldots, X_{i_l}))$.*
- *pour la règle (Decomp) : il est suffisant de montrer qu'il existe un arbre $g(\tau_1, \ldots, \tau_m)$ tel que $\tau_k = \tau_{|d}$ et $g(\tau_1, \ldots, \tau_m) \notin \mathcal{I}(X_j)$, i.e. $pr^{\mathcal{I}}_{g(\tau_1, \ldots, \tau_m)}(\epsilon) \notin I_j$. En effet, puisque $\mathcal{I}$ est une solution, ceci implique que $g(\tau_1, \ldots, \tau_m) \notin g(\top, \ldots, \top, X_i, \top, \ldots \top)$ et donc que $\tau_k = \tau_{|d} \notin \mathcal{I}(X_i)$, i.e. $pr^{\mathcal{I}}_{\tau}(d) \notin I_i$.*

Or, du fait des conditions de la règle d'inférence, il existe dans Δ une règle de transition $g(q_1^l, \ldots, q_m^l) \to q^l$ avec $q^l \notin I_j$ et $\{q_1^l, \ldots, q_m^l\} \backslash \{q_k^l\} \in Reach\_max(\Delta \cup \{lhs \to q\})$ et $q_k^l = q$. Puisque les états de $\{q_1^l, \ldots, q_m^l\} \backslash \{q_k^l\}$ sont maximalement accessibles, il existe des arbres $\tau_1, \ldots, \tau_{k-1}, \tau_{k+1}, \ldots, \tau_m$ tels que $q_i^l = \hat{r}_{\tau_i}^{\Delta}(\epsilon)$.

Or, puisque par hypothèse $pr_\tau^{\mathcal{I}}(d) \preceq^{\mathcal{Q}} q$ et que par définition d'un pré-calcul et d'un plus grand calcul,

$$\forall i \in \{1, \ldots, m\} \backslash \{k\},\ pr_{\tau_i}^{\mathcal{I}}(\epsilon) \preceq^{\mathcal{Q}} \hat{r}_{\tau_i}^{\Delta}(\epsilon)$$

Alors par monotonie des règles, on a $pr_{g(\tau_1,\ldots,\tau_m)}^{\mathcal{I}}(\epsilon) \preceq^{\mathcal{Q}} q^l$ et donc, $pr_{g(\tau_1,\ldots,\tau_m)}^{\mathcal{I}}(\epsilon) \notin I_j$.

- *pour la règle (Member) : il est suffisant de montrer que $(\langle\langle \mathsf{Tr}_\Sigma^\omega, \mathcal{I} \rangle\rangle, [x \mapsto \tau_{|d}]) \not\models \varphi(x)$. En effet, ceci est équivalent à $\tau_{|d} \notin \mathcal{I}(\{x \mid \varphi(x)\})$. Puisque $\mathcal{I}$ est une solution, cette dernière proposition implique que $\tau_{|d} \notin \mathcal{I}(X_i)$, i.e. $pr_\tau^{\mathcal{I}}(d) \notin I_i$.*

  Par les conditions de la règle d'inférence, on a $(\mathsf{R}_\Delta^\top, [x \mapsto q]) \not\models \tilde{\varphi}(x)$. Or, par hypothèse, $pr_\tau^{\mathcal{I}}(d) \preceq^{\mathcal{Q}} q$, donc on a par le lemme 5, $(\mathsf{R}_\Delta^\top, [x \mapsto pr_\tau^{\mathcal{I}}(d)]) \not\models \tilde{\varphi}(x)$.

  Par le lemme 7, on déduit de cette dernière affirmation que $(\langle\langle \mathsf{Tr}_\Sigma^\omega, \tilde{\mathcal{I}} \rangle\rangle, [x \mapsto \tau_{|d}]) \not\models \tilde{\varphi}(x)$, ce qui implique par la proposition 6, $(\langle\langle \mathsf{Tr}_\Sigma^\omega, \mathcal{I} \rangle\rangle, [x \mapsto \tau_{|d}]) \not\models \varphi(x)$

■

On peut alors démontrer que l'algorithme proposé est complet en montrant que si la valeur calculée par notre algorithme est un automate, alors l'interprétation ensembliste qu'il définit est plus grande que toute solution du système de contraintes (au sens de $\sqsubseteq$) et que si cette valeur est $\square$, alors le système est insatisfiable.

Théorème 16 Soit e_f, la valeur finale calculée par notre algorithme (*i.e.*, pour tout e tel que $e_f \vdash_{\mathcal{R}^\omega} e$, on a $e = e_f$),

- si e_f est un automate ($e_f \neq \square$), alors $\forall \mathcal{I} \in SOL(\mathcal{SC}),\ \mathcal{I} \sqsubseteq \mathcal{I}_{e_f}$.
- si $e_f = \square$, alors $\mathcal{SC}$ est insatisfiable.

Preuve :

Il est évident que pour toute solution $\mathcal{I}$ et pour tout arbre τ, $pr_\tau^{\mathcal{I}}$ est un pré-calcul dans l'automate Δ_1. Ainsi, pour tout automate Δ tel que $\Delta_1 \vdash_{\mathcal{R}^\omega}^* \Delta$, par le lemme 19, pour toute solution $\mathcal{I}$ et pour tout arbre τ, $pr_\tau^{\mathcal{I}}$ est un pré-calcul dans l'automate Δ. Pour un tel automate Δ (vérifiant par la proposition 18), il existe pour l'arbre τ un plus grand calcul $\hat{r}_\tau^{\Delta}$. On a donc pour tout arbre τ, $pr_\tau^{\mathcal{I}}(\epsilon) \preceq^{\mathcal{Q}} \hat{r}_\tau^{\Delta}(\epsilon)$,

- *pour le premier point : la remarque précédente est vraie en particulier pour $\Delta = e_f$ et est, par définition, équivalente à : pour tout variable X_i, $\tau \in \mathcal{I}(X_i) \Longrightarrow \tau \in \mathcal{I}_{e_f}(X_i)$. Ainsi, $\mathcal{I} \sqsubseteq \mathcal{I}_{e_f}$.*
- *pour le second point : soit Δ, un automate tel que $\Delta_1 \vdash_{\mathcal{R}^\omega}^* \Delta$ et $\Delta \vdash_{\mathcal{R}^\omega} \square$. Par la règle d'inférence (Clash), on a $\hat{r}_a^{\Delta}(\epsilon) \notin I_i$ et donc par la remarque précédente, pour toute solution $\mathcal{I}$, $pr_a^{\mathcal{I}}(\epsilon) \notin I_i$, i.e. $a \notin \mathcal{I}(X_i)$. Du fait, de la contrainte $a \subseteq X_i$, on en déduit immédiatement que le système ne peut avoir de solution.*

■

On peut déduire des preuves de correction partielle et de complétude que notre algorithme produit la valeur $\square$ si et seulement le système de contraintes $\mathcal{SC}$ est insatisfiable. Dans le cas contraire, Δ_f, l'automate produit, définit la plus grande solution du système $\mathcal{SC}$. Ainsi, de par l'unicité du résultat, en appliquant notre algorithme à l'aide d'une stratégie « terminante», la confluence du calcul est assurée.

Remarque : Nous avons montré dans la section 3.3.2 le lien existant entre les contraintes définies et co-définies généralisées. Ceci nous a permis d'en conclure (avant même de proposer un algorithme pour cette classe) que les solutions d'un système de contraintes co-définies généralisées (lorsqu'il est interprété sur les ensembles d'arbres finis) forment un semi-treillis supérieur complet. Il est très simple de vérifier comme cela a été fait à la proposition 10 que cette propriété est vraie également lorsque ces contraintes sont interprétées sur des ensembles d'arbres finis et infinis.

Dans la section suivante, nous allons la complexité du problème de satisfiabilité d'un système de contraintes co-définies généralisées lorsqu'elles sont interprétées sur les ensembles d'arbres finis et infinis.

Complexité

La classe des contraintes co-définies généralisées étend (syntaxiquement) la classe des contraintes co-définies introduites par Charatonik et Podelski dans [18] et sont toutes les deux interprétées sur les ensembles d'arbres finis et infinis. Ainsi, la complexité du problème de satisfiabilité de la classe des contraintes co-définies, prouvée *EXPTIME*-complète dans [18], nous donne une borne inférieure pour la complexité de notre problème.

Proposition 20 Le problème de satisfiabilité de la classe des contraintes co-définies généralisées (interprétée sur les ensembles d'arbres finis et infinis) est *EXPTIME*-dur.

Nous allons maintenant exprimer la complexité de notre algorithme en fonction de la complexité du test d'accessibilité maximale en raffinant la stratégie d'application des règles déjà définie (nécessaire à la préservation de la propriété de monotonie des règles pour les automates calculés).

Cette fonction sera paramétrée par le temps nécessaire pour tester l'appartenance d'état à la valeur de l'interprétation de la variable $\top$ dans la structure $\mathsf{R}_\Delta^\top$, *i.e.* dans ce cas précis pour tester si un état est ou non maximalement accessible dans un automate Δ ; pour un automate de taille n, ce coût en temps sera noté $Sem^\top(n)$.

Proposition 21 Soit $\mathcal{SC}$ le système de contraintes de taille T pris en entrée de notre algorithme. Le test de satisfiabilité de ce système est effectué en (au plus) $O(T^3\, 2^{k\,(T^2+T)} * Sem^\top(T\, 2^{k'\,T})$, où k, k' sont des constantes dépendantes de la signature Σ.

Preuve :

Commençons par poser quelques notations : soit T la taille du système $\mathcal{SC}$ pris en entrée par notre algorithme. Soit $N_\mathcal{V}$ (resp. $N_{\mathcal{SC}}$) le nombre de variables ensemblistes (resp. de contraintes ensemblistes) apparaissant dans le système de contraintes donné en entrée. Soit S_φ la longueur de la plus grande formule monadique positive apparaissant dans les expressions ensemblistes des contraintes de $\mathcal{SC}$ et S_Q la longueur de la plus grande quantification de ces formules. Soient N_f et amax respectivement le nombre de symboles de fonctions et l'arité maximale de la signature Σ.

Nous allons supposer que tester le fait qu'un état soit initial pour une composante et que transformer un 0 en un 1 dans un membre droit de règle sont des opérations s'effectuant en temps constant.

Nous rappelons que l'application d'une règle d'inférence sur une règle de transition (ayant un membre gauche minimal) entraîne que l'automate produit voit sa règle de même membre gauche comporter une composante de moins à 1. Puisque l'automate comporte au plus $2^{N_\mathcal{V}\,amax}N_f$ règles de transition (ayant chacune $N_\mathcal{V}$ composantes), on peut en déduire que l'algorithme réalise au plus $2^{N_\mathcal{V}\,amax}N_f N_\mathcal{V}$ applications de règles d'inférence.

Pour une contrainte donnée, la recherche de la règle de transition de membre gauche minimale sur laquelle la règle d'inférence peut être utilisée peut se faire naïvement en parcourant les $2^{N_\mathcal{V}\,amax}N_f$ de l'automate en testant (au pire) pour chacune d'elles leur déclenchabilité, i.e. l'initialité d'un état (correspondant à la composante à transformer) et la vérification des conditions de la règles d'inférence.

Selon le type de la règle d'inférence, le coût de ce test de déclenchabilité est : amax pour la règle (Compose) et $O(1)$ pour la règle (Clash). Pour la règle (Decomp), il convient de trouver une règle parmi au plus $2^{N_\mathcal{V}\,amax}$ pour laquelle de plus $(a-1)$ tests d'accessibilité maximale d'états seront réalisés (en $Sem^\top(|\Delta|)$ pour un chaque test). Pour la règle (Member), Il est nécessaire de tout d'abord calculer l'ensemble des états maximalement accessibles, ce qui peut être réalisé en $2^{N_\mathcal{V}} * Sem^\top(|\Delta|)$; dans un second temps, la non-validité de la forme $\tilde{\varphi}$ peut être tester en $O(2^{N_\mathcal{V}\,S_\mathcal{Q}}\,S_\varphi)$.

Pour chacune de ses règles, on peut majorer le test de déclenchabilité d'une règle d'inférence par $O(k_1 2^{k_2 * N_\mathcal{V}} * Sem^\top(|\Delta|) + 2^{N_\mathcal{V}\,S_\mathcal{Q}}\,S_\varphi)$, où k_1 et k_2 sont deux constantes dépendantes de la signature Σ.

En majorant $N_\mathcal{V}$, $S_\mathcal{Q}$, S_φ et N_{SC} par T, et en notant que la taille de l'automate Δ est (au plus) en $2^{N_\mathcal{V}\,amax}N_f\,N_\mathcal{V}\,(amax+1)$, une majoration grossière nous donne le résultat énoncé.

■

Théorème 17 Le problème de satisfiabilité de la classe des contraintes co-définies généralisées (interprétée sur les ensembles d'arbres finis et infinis) est *EXPTIME*.

Preuve :

Immédiat de par la proposition 21 et le théorème 14 affirmant que le test d'accessibilité maximale $Sem^\top$ est polynômial dans la taille de l'automate.

■

Dans la section suivante, nous allons considérer deux restrictions à la classe des contraintes co-définies généralisées que nous avons définies dans ce chapitre.

4.3 Contraintes co-définies généralisées : deux restrictions

Nous allons dans cette section proposer deux restrictions à la classe des contraintes co-définies généralisées : la première de ces restrictions sera syntaxique et nous permettra de caractériser exactement la complexité du problème de satisfiabilité. Elle se basera sur les expressions d'appartenance simples présentées dans le chapitre précédent à la section 3.3.1. La seconde sera sémantique, puisqu'elle consistera à interpréter les contraintes co-définies généralisées sur les ensembles d'arbres finis.

4.3.1 Contraintes co-définies avec expressions d'appartenance simples

Cette restriction correspond au travail publié dans [26].

Syntaxe

Nous rappelons la définition syntaxique de la classe des contraintes co-définies avec expressions d'appartenance simples. Une telle contrainte est une inclusion de la forme $Sexp_A \subseteq Sexp_B$,

où $Sexp_A$ et $Sexp_B$ sont deux expressions ensemblistes décrites par les grammaires suivantes :

$$Sexp_A ::= X \mid a \mid f(\top, \ldots, \top, Sexp_A, \top, \ldots, \top) \mid Sexp_A \cup Sexp_A \mid \top$$

$$\begin{aligned} Sexp_B ::= {} & X \mid f(Sexp_B, \ldots, Sexp_B) \mid Sexp_B \cup Sexp_B \mid Sexp_B \cap Sexp_B \mid \bot \mid \\ & f_k^{-1}(Sexp_B) \mid \{x \mid \Phi(x)\} \\ \Phi ::= {} & t \in Sexp_B \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists y \Phi \mid \forall y \Phi \end{aligned}$$

où $X \in \mathcal{V}$, $f \in \Sigma$, $a \in \Sigma_0$, $t \in TERM(\Sigma, \mathcal{X})$ et $\Phi(x)$ est une formule monadique positive étendue ayant x pour unique variable libre du premier ordre et ne comportant pas de quantificateur universel ($\forall$).

Comme pour les contraintes co-définies généralisées, nous pouvons sans perte de généralité[60] transformer un système de contraintes répondant à cette syntaxe en un système en forme aplatie, *i.e.* dont les contraintes sont de la forme :

$$f(\top, \ldots, \top, Y, \top, \ldots, \top) \subseteq X \qquad X \subseteq f(X_1, \ldots, X_m)$$
$$a \subseteq X \qquad X \subseteq \{x \mid \varphi(x)\}$$

où $\varphi(x)$ est une formule monadique positive ne comportant pas de quantificateur universel.

Un test de satisfiabilité

Nous allons dans cette section adapter le système d'inférence $\mathcal{R}^\omega$ à la restriction que nous considérons. Ce nouveau système d'inférence, que nous noterons $\mathcal{R}^\omega_\exists$, définira un algorithme d'une manière identique à $\vdash_{\mathcal{R}^\omega}$.

On peut remarquer que dans le système d'inférence $\mathcal{R}^\omega$ était présent, pour chaque type de contraintes apparaissant dans un système co-défini généralisé, une et une seule règle d'inférence. Il en sera de même pour $\mathcal{R}^\omega_\exists$.

Pour les contraintes de la forme $a \subseteq X$, $X \subseteq f(X_1, \ldots, X_m)$ et $f(\top, \ldots, \top, Y, \top, \ldots, \top) \subseteq X$ le système $\mathcal{R}^\omega_\exists$ contient les règles *(Clash)*, *(Compose)* et *(Decomp)* du système $\mathcal{R}^\omega$ (voir figure 4.4). Pour les contraintes de la forme $X \subseteq \{x \mid \varphi(x)\}$, $\mathcal{R}^\omega_\exists$ contient en fait une restriction de la règle *(Member)* de $\mathcal{R}^\omega$ en se basant sur les affirmations suivantes.

Nous avons vu qu'un automate $\mathcal{A}$ d'arbres infinis déterministe et complet (au sens ascendant) de rang n définissait une Σ-algèbre $\mathsf{A}_\mathcal{A}$, dont le domaine est l'ensemble des états de l'automate $\mathcal{Q}$. Pour des variables du second ordre (ou ensemblistes) $\{X_1, \ldots, X_n\}$, cette algèbre s'étend en une $(\Sigma, \{X_1, \ldots, X_n\})$-structure $\mathsf{R}_\mathcal{A}$ et pour les formules monadiques $\top$-bornées, en une $(\Sigma, \top, \{X_1, \ldots, X_n\})$-structure, notée $\mathsf{R}^\top_\mathcal{A}$ en fixant l'interprétation de la variable $\top$ dans cette structure à $Reach\_max(\mathcal{A})$, l'ensemble des états maximalement accessibles de $\mathcal{A}$. Basée sur la structure $\mathsf{R}_\mathcal{A}$, nous définissons la $(\Sigma, \{\top, X_1, \ldots, X_n\})$-structure $\tilde{\mathsf{R}}^\top_\mathcal{A}$ en posant que l'interprétation de la variable $\top$ est égale à $Reachable(\mathcal{A})$, l'ensemble des états accessibles de l'automate $\mathcal{A}$.

On peut alors montrer que :

[60] Il y a préservation des solutions restreintes aux variables du système initial.

Proposition 22 Soit χ, une formule monadique $\top$-bornée ne comportant pas de quantificateur universel, pour toute valuation ν de $Var_{\mathcal{X}}(\chi)$ dans $\mathcal{Q}$,

$$(\tilde{\mathsf{R}}^{\top}_{\mathcal{A}}, \nu) \models \chi \text{ ssi } (\mathsf{R}^{\top}_{\mathcal{A}}, \nu) \models \chi$$

Preuve :

La preuve est faite par induction sur la structure de la formule χ : de par la nature des deux structures $\tilde{\mathsf{R}}^{\top}_{\mathcal{A}}$ et $\mathsf{R}^{\top}_{\mathcal{A}}$, il est évident que la proposition est vraie si χ est une formule atomique et en utilisant l'hypothèse d'induction et la remarque 1, qu'elle est vraie également pour une formule de la forme $\chi_1 \wedge \chi_2$ ou $\chi_1 \vee \chi_2$. Alors, pour une formule χ de la forme $\exists x, (x \in \top \wedge \chi')$, par une double implication :

($\Rightarrow$) *si $(\tilde{\mathsf{R}}^{\top}_{\mathcal{A}}, \nu) \models \exists x, (x \in \top \wedge \chi')$, alors par définition, il existe un état accessible q tel que $(\tilde{\mathsf{R}}^{\top}_{\mathcal{A}}, \nu \circ [x \mapsto q]) \models \chi')$. Par hypothèse de récurrence, ceci est équivalent à dire qu'il existe un état accessible q tel que $(\mathsf{R}^{\top}_{\mathcal{A}}, \nu \circ [x \mapsto q]) \models \chi'$. Or, pour tout état accessible q, il existe un état q' maximalement accessible tel que $q \preceq^{\mathcal{Q}} q'$. De plus, par le lemme 5, on peut donc en déduire qu'il existe un état maximalement accessible q' tel que $(\mathsf{R}^{\top}_{\mathcal{A}}, \nu \circ [x \mapsto q']) \models \chi'$ et donc par définition, $(\mathsf{R}^{\top}_{\mathcal{A}}, \nu) \models \exists x, (x \in \top \wedge \chi')$.*

($\Leftarrow$) *si $(\mathsf{R}^{\top}_{\mathcal{A}}, \nu) \models \exists x, (x \in \top \wedge \chi')$, alors par définition, il existe un état maximalement accessible q tel que $(\mathsf{R}^{\top}_{\mathcal{A}}, \nu \circ [x \mapsto q]) \models \chi'$. Par hypothèse d'induction et puisque tout état maximalement accessible est accessible, on a $(\tilde{\mathsf{R}}^{\top}_{\mathcal{A}}, \nu \circ [x \mapsto q]) \models \chi')$ et donc, $(\tilde{\mathsf{R}}^{\top}_{\mathcal{A}}, \nu) \models \exists x, (x \in \top \wedge \chi')$.*

■

Il est bien évident que, de par la construction donnée dans la définition 14, pour une formule monadique positive φ sans quantificateur universel, la formule monadique $\top$-bornée $\tilde{\varphi}$ qui lui est associée ne comporte pas non plus de quantificateur universel.

Ainsi, pour des expressions ensemblistes simples (avec des formules sans quantificateur universel), la règle *(Member)* de $\mathcal{R}^{\omega}$ peut s'écrire comme

$$\frac{\Delta \cup \{lhs \rightarrow q\}}{\Delta \cup \{lhs \rightarrow switch_0(q,i)\}} \text{ si } \begin{cases} X_i \subseteq \{x \mid \varphi(x)\} \\ (\tilde{\mathsf{R}}^{\top}_{\Delta \cup \{lhs \rightarrow q\}}, [x \mapsto q]) \not\models \tilde{\varphi}(x) \end{cases}$$

Cette règle est la dernière règle d'inférence de $\mathcal{R}^{\omega}_{\exists}$.

Correction - complexité

La correction de l'algorithme (défini à partir des règles de $\mathcal{R}^{\omega}_{\exists}$) répondant au problème de satisfiabilité d'un système de contraintes co-définies avec expressions d'appartenance simples, est par la proposition 22 un corollaire de la correction de l'algorithme pour les contraintes co-définies généralisées.

Du point de vue de la complexité, on peut montrer que :

Proposition 23 Le problème de satisfiabilité d'un système de contraintes co-définies avec expressions d'appartenance simples (interprété sur les ensembles d'arbres finis et infinis) est un problème *EXPTIME*-complet.

Preuve :

Puisque syntaxiquement tout système de contraintes ensemblistes co-définies est un système de contraintes co-définis avec expressions d'appartenance simples, on peut déduire de [18] que le problème de satisfiabilité d'un système de cette dernière classe est EXPTIME-dur.

L'algorithme que nous proposons ici n'est qu'une restriction de l'algorithme pour la classe des contraintes co-définies généralisées. Ainsi, le résultat de la proposition 21 nous donne une borne supérieure pour la complexité de cette restriction. Cependant, il est à noter que si $Sem^{\top}(n)$ *désigne le coût pour un automate de taille n du fait de tester si un état q appartient à l'interprétation de la variable* $\top$ *sur* $\mathsf{R}^{\top}_{\Delta}$ *(i.e. qu'il est maximalement accessible dans* Δ*), du fait de la proposition 22, on se contente de tester uniquement le fait qu'il soit accessible. Ceci se traduit en disant que* $Sem^{\top}(n)$ *désigne pour la restriction le coût du test de l'appartenance d'un état q à l'interprétation de la variable* $\top$ *dans la structure* $\tilde{\mathsf{R}}^{\top}_{\Delta}$.
Or, le test d'accessibilité d'un état dans un tel automate est polynômial par rapport à la taille de celui-ci. En conséquence, on en déduit que l'algorithme est dans EXPTIME.

■

4.3.2 Interprétation sur les ensembles d'arbres finis

Si la première restriction que nous avons proposée, était d'ordre syntaxique, la seconde sera quant-à-elle d'ordre sémantique. Nous allons en effet nous intéresser aux contraintes co-définies généralisées, interprétées cette fois-ci sur les ensembles d'arbres finis (*i.e.* de termes clos).

Nous avons vu dans la section 3.3.2 qu'en raisonnant à l'aide de dualisation par complémentation, on pouvait tester la satisfiabilité d'un système de contraintes co-définies généralisées interprétées sur les ensembles d'arbres finis en encodant ce problème dans celui de la satisfiabilité d'un système de contraintes définies généralisées. On en déduisait que dans le cas, où le système co-défini était satisfiable, alors l'automate (de rang n) calculé pour son dual défini reconnaissait le complémentaire de la plus grande solution du système co-défini. Nous rappelons que cet automate d'arbres finis ascendants de rang n est déterministe et complet. Nommons cet automate Δ de la forme (Σ, Q, F, Δ), avec $F = (F_1, \ldots, F_n)$; le langage reconnu $(L_1, \ldots, L_n)$ par cet automate définit la plus petite solution du système de contraintes définies généralisées calculée par complémentation. Or, on peut aisément définir un automate $\Delta^{\complement}$ reconnaissant le langage $(\complement L_1, \ldots, \complement L_n)$ en posant $(\Sigma, Q, F^{\complement}, \Delta)$, $F^{\complement} = (F_1^{\complement}, \ldots, F_n^{\complement})$, où $F_i^{\complement}$ est l'ensemble des états ayant 0 sur leur $i^{ème}$ composante. On peut donc affirmer que l'automate $\Delta^{\complement}$ définit une interprétation ensembliste qui correspond à la plus grande solution du système de contraintes co-définies généralisées considéré.

Tout ceci nous a permis de définir un algorithme répondant au problème de satisfiabilité pour un système de la classe des contraintes co-définies généralisées interprétées sur les ensembles d'arbres finis en se basant sur celui proposé pour la classe des contraintes définies, mais également de conclure que ce problème était *EXPTIME*-complet.

Cette section a simplement pour but de montrer que l'algorithme proposé dans ce chapitre peut lui aussi être adapté pour permettre le test de satisfiabilité pour un système de la classe co-définie interprétée sur les ensembles de termes clos et de démontrer que cet algorithme est dans *EXPTIME*.

Cette adaptation est nécessaire, car comme il est montré dans l'exemple suivant pour un système de contraintes ensemblistes co-définies donné, la restriction aux arbres finis d'une solution calculée sur les ensembles d'arbres finis et infinis n'est généralement pas une solution ; ceci implique qu'un système co-défini satisfiable lorsqu'il est interprété sur les ensembles d'arbres finis

et infinis peut ne pas l'être lorsqu'il est interprété sur les ensembles d'arbres finis.

Exemple 16 : *Le système suivant est satisfiable sur les ensembles finis et infinis d'arbres, mais insatisfiable sur les ensembles d'arbres finis.*

$$X \subseteq f(X)$$
$$a \subseteq \{y \mid \exists x \in X \wedge y \in a\}$$

En effet, si on considère l'unique solution du système (interprété sur les ensembles d'arbres finis et infinis), elle associe à X le singleton $\{\tau\}$, tel $dom(\tau) = \{1\}^\omega$ et pour tout $d \in dom(\tau)$, $\tau(d) = f$. La restriction de cette solution sur un ensemble d'arbres finis l'interprétation associant à X l'ensemble vide $\varnothing$. Cette interprétation est (la seule) solution de la première contrainte (interprétée sur les ensembles d'arbres finis), mais viole la seconde.

Il n'existe pas de réelle différence entre les automates ascendants déterministes et complets de rang n et les automates d'arbres infinis (sans condition d'acceptance) déterministes et complets au sens ascendant de rang n à la fois sur le plan de la définition syntaxique de ces deux objets et sur les notions de calcul pour les arbres finis [61] , la définition donnée pour les premiers n'étant qu'une version opérationnelle de celle donnée pour les seconds. On peut même ajouter qu'en fait du point de vue du langage reconnu, les premiers peuvent être vus comme la restriction des seconds aux ensembles d'arbres finis.

En redéfinissant pour les automates de la classe TA^{ω}_{SC}, la notion d'accessibilité d'un état q dans un automate $\mathcal{A}$ comme l'existence d'un arbre τ **fini** et d'un calcul $r^{\mathcal{A}}_{\tau}$ (pour τ dans $\mathcal{A}$ tel que $r^{\mathcal{A}}_{\tau}(\epsilon) = q$, cette notion se confond du fait de l'unicité du calcul avec celle d'accessibilité maximale. De plus, adaptons la définition des structures $\hat{\mathsf{R}}_{\Delta}$ et $\mathsf{R}^{\top}_{\Delta}$ en fixant pour la première que son support est en fait les états accessibles par un arbre fini et que cet ensemble défini pour la seconde la sémantique de la variable $\top$ dans celle-ci.

En considérant le système d'inférence $\mathcal{R}^{\omega}$ et l'algorithme qu'il définit avec ces nouvelles définitions, les preuves de correction et de complétude s'adapte parfaitement dans ce nouveau cadre. Ceci nous permet de conclure que dans ce cas, l'algorithme répondant au problème de satisfiabilité du système de contraintes co-définies généralisées interprété sur les ensembles d'arbres finis. De plus, on peut affirmer que lorsque le résultat de cet algorithme est un automate d'arbres d'infinis de rang n, alors la restriction du langage reconnu par cet automate aux ensembles d'arbres finis est exactement la plus grande solution du système de contraintes interprété sur les ensembles d'arbres finis.

En ce qui concerne la complexité, reprenons la proposition 21. Le coût en temps qui y est défini est paramétré par la grandeur $Sem^{\top}(T\,2^{k'\,T})$, où $T\,2^{k'\,T}$ majore la taille de l'automate en fonction de la taille du système de contraintes T. En se rappelant du fait que $Sem^{\top}$ désigne le coût du test de l'appartenance d'un état q à l'interprétation de la variable $\top$ dans la structure $\mathsf{R}^{\top}_{\Delta}$ et que dans la restriction que nous avons définie, ceci revient alors simplement à tester le fait que q soit finiment accessible dans l'automate considéré, on peut en déduire que $Sem^{\top}(T\,2^{k'\,T})$ est de l'ordre de $O(T\,2^{k'\,T})$. Ainsi, il est immédiat de conclure que tout ceci défini un algorithme *EXPTIME* répondant au problème de satisfiabilité d'un système de contraintes co-définies généralisées, interprétées sur les ensembles d'arbres finis.

[61] Nous avons déjà mentionné le fait qu'il y a pour tout arbre fini unicité du calcul dans un automate d'arbres infinis déterministes et complets au sens ascendants de rang n.

Chapitre 5

Application à l'analyse de programmes logiques

Le but de ce chapitre est de présenter l'une des applications principales des contraintes ensemblistes, l'analyse de programmes et plus précisément l'analyse de programmes logiques. Il se veut être principalement une présentation de l'analyse ensembliste des programmes logiques, mais également d'une étude des apports de différentes techniques liées à ce qu'on appelle l'approximation des programmes logiques (dont l'analyse ensembliste est une technique particulière) permettant une amélioration de la qualité de l'analyse ensembliste. Nous verrons en particulier que l'approche « automate d'arbres »[62] de la résolution des contraintes ensemblistes permet de proposer une amélioration dans un cadre purement logique.

5.1 Généralités

L'analyse statique[63] de programmes a pour but d'extraire d'un programme des propriétés de celui-ci. Cette définition très générale regroupe au travers du terme *propriété* des domaines comme la preuve de terminaison, de correction partielle ou de complétude (vis-à-vis d'une spécification) ou d'équivalence selon différentes sémantiques, mais également des informations approchées sur les données manipulées ou le comportement d'un programme.

L'extraction d'informations d'un programme permet non seulement de pouvoir appréhender des problèmes de correction partielle ou de complétude dans une optique de typage ou déboguage, mais celles-ci peuvent également être utiles dans le cadre de l'optimisation, aussi bien d'un point de vue utilisateur, permettant à celui-ci d'améliorer le programme manuellement, que d'un point vue automatique en fournissant ces informations à un programme de « transformation de programmes », à un compilateur ou à un interpréteur.

C'est dans ce cadre que se situe l'*analyse ensembliste* de programmes dont le but principal est de calculer une approximation de la sémantique d'un programme. Ce type d'analyse est « dirigé » par les variables apparaissant dans un programme : l'idée essentielle est de calculer une sur-approximation [64] de l'ensemble des valeurs que peuvent prendre celles-ci au cours de l'exécution du programme et de capturer d'une certaine façon le comportement d'une exécution ensembliste du programme, c-à-d une forme de « sémantique ensembliste » du programme. Ceci

[62] En réalité, c'est surtout l'aspect algébrique de l'approche qui est importante.

[63] Le terme « statique » signifie ici « sans exécution », ou tout au moins sans exécution « réelle » du programme.

[64] L'ensemble exact des valeurs prises par des variables au cours de l'exécution d'un programme n'étant bien évidemment pas un ensemble récursif.

est réalisé en particulier du le fait que le raisonnement du comportement individuel d'une variable relativement à celui des autres variables est impossible : on ne peut avancer le fait qu'une variable x prend la valeur a lorsque la variable y prend la valeur b[65]. On pourra cependant savoir que la variable x peut prendre la valeur a et la variable y peut prendre la valeur b au cours de l'exécution du programme. On parle alors de perte de *dépendance inter-variables*. Il va de soi que cette perte de dépendance entraîne que l'ensemble des valeurs que peut prendre une variable n'est qu'un sur-ensemble approché des valeurs qu'elle prendra effectivement au cours de l'exécution. Cette approche d'analyse de programmes est très générale et s'adapte à différents paradigmes de programmation, comme la programmation impérative [37], fonctionnelle [6, 38, 47] ou logique [37]. Pour chacun de ces formalismes, cette idée de perte de dépendances inter-variables est formalisée au travers d'une sémantique ensembliste approchant la sémantique effective du paradigme.

La définition d'une sémantique ensembliste ne donne pas de méthode opérationnelle, ni pour le calcul d'une représentation de l'approximation, ni même de procédure de test d'appartenance d'une valeur à cette approximation de la sémantique. C'est sur ce plan qu'interviennent les contraintes ensemblistes : elle fournissent un moyen général d'approximation de la sémantique de manière purement syntaxique, le calcul de l'approximation n'étant plus lié qu'à la résolution de ces contraintes. Ainsi, la démarche de l'analyse ensembliste est la même pour les différents paradigmes de programmation et se découpe en deux phases [55] : la proposition d'une méthode d'extraction de la syntaxe d'un programme ou d'une des sémantiques du langage considéré d'un ensemble de contraintes ensemblistes, méthode devant être prouvée correcte (c-à-d qu'elle constitue bien une approximation de la sémantique du programme). Puis, la proposition d'un algorithme pour la classe des contraintes extraites qui selon le cas peut simplement tester la satisfiabilité du système de contraintes [50, 52] ou calculer une solution particulière (comme la plus petite ou la plus grande).

Comme l'indique le titre de ce chapitre, nous allons nous intéresser uniquement à l'analyse ensembliste des programmes logiques définis. Nous allons tout d'abord présenter comment l'idée intuitive de perte de dépendances inter-variables se traduit formellement au travers d'une approximation de l'opérateur de conséquences logiques immédiates (qui est un des moyens d'exprimer la sémantique d'un programme logique), en terme d'un autre opérateur que l'on qualifiera d'« ensembliste». Puis, dans un second temps, nous allons voir comment la sémantique définie par cet opérateur ensembliste peut être effectivement calculée par une méthode d'extraction syntaxique d'un système de contraintes du programme à analyser et par la résolution de ces contraintes.

5.2 Analyse ensembliste de programmes logiques

Nous allons maintenant préciser notre propos sur l'analyse ensembliste dans le cadre de la programmation logique.

L'analyse ensembliste de programmes logiques (définie dans [37]) s'inscrit dans une thématique plus générale qu'est le *typage « souple »*, également appelée *approximation de programmes logiques*. Cette notion de typage « souple »[66] de programmes logiques a été introduite par Mishra dans [51]. L'idée de celle-ci est simplement de calculer un sur-ensemble récursif de la dénotation du programme. Un atome clos mal-typé, *i.e.* n'appartenant pas à ce sur-ensemble, ne peut donc

[65] Tout du moins, pas pour n'importe quelles variables à n'importe quel moment de l'exécution.

[66] Ce qualificatif de « souple » provient du fait qu'il doit être utile au programmeur d'ajouter au formalisme de la programmation logique un formalisme *ad-hoc* de typage.

être succès pour le programme logique considéré. En pratique, dans la plupart des travaux existants ce sur-ensemble récursif est un ensemble régulier d'arbres finis.

Nous allons tout d'abord donner la formalisation de la perte de dépendance inter-variables proposée par Heintze et Jaffar.

5.2.1 L'opérateur ensembliste $\mathcal{T}_{P,\mathsf{T}_\Sigma}$

Nous avons précédemment dit que l'analyse ensembliste est intuitivement décrite comme « une perte de dépendance inter-variables » et que cette intuition se formalise au travers d'une modification (entraînant une approximation) de la sémantique pour le programme considéré.

Heintze et Jaffar donnent dans [40] cette formulation de l'analyse ensembliste en se basant sur la sémantique de point fixe des programmes logiques, à savoir l'opérateur T_{P,T_Σ}. Ils y présentent un opérateur de point fixe dit ensembliste, noté $\mathcal{T}_{P,\mathsf{T}_\Sigma}$, « approchant » l'opérateur T_{P,T_Σ}. Nous allons présenter cet opérateur ensembliste et voir en quoi il traduit l'idée intuitive de pertes de dépendance inter-variables.

Considérons S un ensemble de substitutions closes définies sur un même domaine[67]. La fonction App_α est une fonction d'approximation pour un tel ensemble de substitutions, calculant une interprétation ensembliste dont le domaine est celui des substitutions présentes dans S et qui associe à chacun des éléments de cet ensemble un ensemble de termes clos. Cette interprétation ensembliste est définie comme suit, pour toute variable x du domaine des éléments de S :

$$(App_\alpha(S))(x) = \{\sigma(x) \mid \sigma \in S\}$$

Considérons par exemple, $S = \{[x \mapsto a,\ y \mapsto b], [x \mapsto c,\ y \mapsto a]\}$, on a alors

$$(App_\alpha(S))(x) = \{a, c\}$$
$$(App_\alpha(S))(y) = \{a, b\}$$

On dit qu'une substitution σ' est représentée dans $App_\alpha(S)$ si pour toute variable x du domaine de σ', $\sigma'(x) \in (App_\alpha(S))(x)$, ce que l'on notera $\sigma' \hat{\in} App_\alpha(S)$.

Ainsi, pour l'exemple, l'ensemble des substitutions représentées est

$$[x \mapsto a,\ y \mapsto a] \qquad [x \mapsto a,\ y \mapsto b]$$
$$[x \mapsto c,\ y \mapsto a] \qquad [x \mapsto c,\ y \mapsto b]$$

Il est immédiat de noter que l'ensemble S est un sous-ensemble des substitutions représentées dans $App_\alpha(S)$, *i.e.* pour tout $\sigma \in S$, $\sigma \hat{\in} App_\alpha(S)$.

Ces notions sont le cœur de la formalisation de la sémantique ensembliste des programmes logiques, puisque l'ensemble des substitutions représentées par $App_\alpha(S)$ pour un ensemble S correspond bien à une perte de dépendance inter-variables : pour les substitutions représentées, il n'existe par construction aucun lien entre les valeurs des différentes variables (du domaine de la substitution). Il est à noter que cette approximation est bien de nature ensembliste de par la définition de $App_\alpha(S)$.

Nous allons maintenant voir comment cette approximation est introduite au niveau de la sémantique d'un programme logique permettant la définition d'une sémantique dite « ensembliste ».

[67] On appelle *domaine d'une substitution* l'ensemble des variables pour lequel elle est définie. Ce domaine pour une substitution σ est noté $dom(\sigma)$.

La sémantique considérée est celle de point fixe définie à l'aide de l'opérateur de conséquence logique immédiate T_{P,T_Σ}, défini de $\wp(BH)$ vers $\wp(BH)$, où BH est la base de Herbrand[68],

$$T_{P,\mathsf{T}_\Sigma}(I) = \left\{ H \;\middle|\; \begin{array}{l} H \Leftarrow B_1, \ldots, B_l \in INST_{\mathsf{T}_\Sigma}(P) \\ B_1, \ldots, B_l \in I \end{array} \right\}$$

où $INST_{\mathsf{T}_\Sigma}(P)$ est l'ensemble des instances closes des clauses de P. Cet opérateur est simplement une fonction de la base de Herbrand vers elle-même, produisant un résultat (un ensemble d'instances de tête de clause) en fonction de données (les atomes du corps des instances de clauses se trouvant dans I).

Considérons, pour exemple, $I = \{p(a,b),\, p(c,a)\}$ et le programme constitué de l'unique clause $p(f(x,y)) \Leftarrow p(x,y)$, on a donc $T_{P,\mathsf{T}_\Sigma}(I) = \{p(f(a,b)), p(f(c,a))\}$, calculé pour les deux instances closes correspondant aux substitutions $[x \mapsto a,\, y \mapsto b]$ et $[x \mapsto c,\, y \mapsto a]$.

Supposons maintenant qu'au lieu de considérer ces deux substitutions, on choisisse de considérer leurs approximations ensemblistes pour produire le résultat (l'ensemble des instances des têtes de clauses), c-à-d ici l'ensemble des substitutions

$$[x \mapsto a,\, y \mapsto a] \qquad [x \mapsto a,\, y \mapsto b]$$
$$[x \mapsto c,\, y \mapsto a] \qquad [x \mapsto c,\, y \mapsto b]$$

On obtient alors l'ensemble des atomes suivants

$$\{p(a,b),\, p(a,a),\, p(c,b),\, p(c,a)\}$$

Cette description informelle correspond à l'opérateur ensembliste de point fixe, introduit par Heintze et Jaffar dont la définition est :

$$\mathcal{T}_{P,\mathsf{T}_\Sigma}(I) = \left\{ \alpha(H) \;\middle|\; \begin{array}{l} H \Leftarrow B_1, \ldots, B_l \in P \\ \alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma(B_1) \in I \wedge \ldots \wedge \sigma(B_l) \in I\}) \end{array} \right\}$$

On peut noter que puisque pour tout ensemble de substitution S, tout élément de S est représenté dans $App_\alpha(S)$, on a alors, pour tout ensemble d'atomes I, $T_{P,\mathsf{T}_\Sigma}(I) \subseteq \mathcal{T}_{P,\mathsf{T}_\Sigma}(I)$, ce qui traduit le fait que l'opérateur ensembliste est une sur-approximation de T_{P,T_Σ}.

Heintze et Jaffar montrent que cet opérateur est monotone et continu ; il admet donc un plus petit point fixe atteint au premier ordinal transfini et, pour un programme P est noté $\mathcal{T}_{P,\mathsf{T}_\Sigma} \uparrow \omega$. De plus, du fait de la remarque précédente, on a $T_{P,\mathsf{T}_\Sigma} \uparrow \omega \subseteq \mathcal{T}_{P,\mathsf{T}_\Sigma} \uparrow \omega$, *i.e.* le plus petit point fixe de l'opérateur est un sur-ensemble de la dénotation du programme.

Ce plus petit point fixe $\mathcal{T}_{P,\mathsf{T}_\Sigma} \uparrow \omega$ est nommé *sémantique ensembliste* du programme P et est noté DEN_P^{sba}.

Nous avons vu dans cette section comment se traduit l'idée intuitive de l'analyse ensembliste, à savoir la perte de dépendance inter-variables au cours du calcul, dans le cadre de la programmation logique. Cependant, l'introduction de l'opérateur $\mathcal{T}_{P,\mathsf{T}_\Sigma}$ ne constitue nullement une définition opérationnelle pour calculer (une représentation) du résultat de cette analyse ensembliste. Nous allons voir dans la section suivante comment les contraintes ensemblistes jouent ce rôle.

[68] Cette définition est donnée dans la section 1.4.2.

5.2.2 De $\mathcal{T}_{P,\mathsf{T}_\Sigma}$ aux contraintes ensemblistes

Nous avons dans la section précédente la définition formelle de l'analyse ensembliste d'un programme logique, traduisant l'idée de pertes de dépendance inter-variables. Afin de rendre effective cette définition, nous allons voir comment les contraintes ensemblistes peuvent être utilisées en clarifiant la démarche en deux étapes d'extraction, puis résolution des contraintes ensemblistes, ceci en s'assurant bien-sûr de la correction de ces phases vis-à-vis de la définition formelle présentée.

L'analyse ensembliste d'un programme logique, *i.e.* le calcul d'une représentation de la sémantique ensembliste d'un programme logique, peut être vue comme l'extraction d'un système de contraintes de ce programme[69] et la résolution des contraintes nous donnant une solution représentant ce modèle ensembliste.

Nous allons maintenant présenter l'étape d'extraction des contraintes ensemblistes d'un programme logique, d'une manière très voisine de celle de [36].

Cette extraction est purement syntaxique et sera décrite à partir de la définition de l'opérateur de conséquence logique immédiate T_{P,T_Σ}. Pour être tout à fait précis, nous allons reformuler la définition de cet opérateur dans une optique plus proche des contraintes ensemblistes, ce qui permettra de définir plus aisément le mécanisme d'extraction.

Une reformulation de T_{P,T_Σ}

Cette reformulation de la définition de l'opérateur de conséquence logique immédiate se base sur une extension de la définition des expressions d'appartenance que nous avons proposée. Nous allons considérer des expressions d'appartenance de la forme $\{t \mid \varphi(x_1, \ldots, x_m)\}$, où t est un terme quelconque et $\varphi(x_1, \ldots, x_m)$ est une formule monadique (étendue) positive ayant $x_1, \ldots, x_m$ pour variables libres. Un exemple d'un telle expression est

$$\{f(x,y) \mid \exists z,\ f(z,x) \in X \wedge g(y,z) \in Y\}$$

Notons que les expressions d'appartenance que nous avons considérées jusqu'ici correspondent au cas où t est simplement une variable du premier ordre.

La sémantique associée à de telles expressions (pour une interprétation ensembliste $\mathcal{I}$ associant à chaque variable ensembliste un ensemble de termes clos) est l'ensemble des termes clos τ tels qu'il existe une valuation[70] ν de $Var_{\mathcal{X}}(\varphi)$ vers $\mathbb{TR}_\Sigma$ telle que $\tau = \lfloor t \rfloor_\nu^{\mathsf{Tr}_\Sigma}$ et $(\langle\!\langle \mathcal{I}, \mathsf{Tr}_\Sigma \rangle\!\rangle, \nu) \models \varphi$.

Ainsi, pour l'expression donnée en exemple précédemment, pour une interprétation ensembliste $\mathcal{I}$ telle que $\mathcal{I}(X) = \{b, f(d,a), f(d,e)\}$ et $\mathcal{I}(Y) = \{a, g(a,c), g(e,d)\}$, on a

$$\mathcal{I}(X) = \{f(a,e), f(e,e)\}$$

puisque par exemple pour $f(e,e)$, la substitution $\sigma = [x \mapsto e,\ y \mapsto e]$ est telle que $\sigma(f(x,y)) = f(e,e)$ et $(\langle\!\langle \mathcal{I}, \mathsf{Tr}_\Sigma \rangle\!\rangle, \sigma) \models \exists z,\ f(z,x) \in X \wedge g(y,z) \in Y$.

Nous allons maintenant donner la définition de l'opérateur de conséquence logique immédiate à l'aide de l'extension des expressions d'appartenance que nous avons présentée.

On peut affirmer que pour calculer la valeur $T_{P,\mathsf{T}_\Sigma}(I)$, chaque clause apporte sa « contribution » à cette valeur. En particulier, lorsqu'il n'y a qu'une clause dans le programme, comme dans celui de l'exemple de la section précédente $p(f(x,y)) \Leftarrow p(x,y)$, la contribution de cette clause pour une certaine valeur de I, qui peut être considérée comme une variable est

[69] En fait, ces contraintes seront extraites de la définition de l'opérateur de conséquence logique immédiate.

[70] On peut parler ici en l'occurrence de substitution.

$$\{p(f(x,y)) \mid p(x,y) \in I \wedge x \in HU \wedge y \in HU\}$$

où HU est une variable ensembliste dont la valeur est l'univers de Herbrand[71].

Ainsi, pour la valuation ensembliste $\mathcal{I}$ associant à la variable I l'ensemble $\{p(a,b), p(c,a)\}$, la sémantique de l'expression ensembliste $\{p(f(x,y)) \mid p(x,y) \in I \wedge x \in HU \wedge y \in HU\}$ est $\{p(f(a,b)), p(f(c,a))\}$, qui correspond bien à la valeur de $T_{P,\mathsf{T}_\Sigma}(I)$.

Lorsque les programmes comportent plusieurs clauses, alors $T_{P,\mathsf{T}_\Sigma}(I)$ est défini comme l'union de la contribution de chacune des clauses.

La reformulation de T_{P,T_Σ} en tenant compte de la sémantique que nous avons introduite, est donc

$$T_{P,\mathsf{T}_\Sigma}(I) = \bigcup_{c \in P} \left\{ H \;\middle|\; \wedge_i B_i \in I \wedge \bigwedge_{x \in Var_{\mathcal{X}}(c)} x \in HU \right\}$$

où la clause c est de la forme $H \Leftarrow B_1, \ldots, B_l$ et HU désigne l'univers de Herbrand.

Pour un programme logique P, nous allons noter SE^P l'expression

$$\bigcup_{c \in P} \left\{ H \;\middle|\; \wedge_i B_i \in I \wedge \bigwedge_{x \in Var_{\mathcal{X}}(c)} x \in HU \right\}$$

qui définit de manière syntaxique l'opérateur de conséquence logique immédiate de P.

Exemple 17 : *Pour le programme logique P suivant,*

$$p(a,b) \quad p(f(x,y)) \Leftarrow p(x,y)$$
$$p(c,a)$$

on obtient la définition de T_{P,T_Σ} suivante :

$$T_{P,\mathsf{T}_\Sigma}(I) = p(a,b) \cup p(c,a) \cup \{p(f(x,y)) \mid p(x,y) \in I \wedge x \in HU \wedge y \in HU\}$$

Extraction des contraintes

Nous allons maintenant montrer comment, à partir de cette description de l'opérateur de conséquence logique immédiate, on peut extraire un système de contraintes ensemblistes, qui se révélera être un système de contraintes définies généralisées. Nous montrerons de plus comment ce mécanisme purement opérationnel coïncide avec la définition formelle de l'analyse ensembliste de la section précédente.

Nous avons vu précédemment que la reformulation de la définition de $T_{P,\mathsf{T}_\Sigma}(I)$ amenait à considérer des expressions ensemblistes de la forme :

$$Sexp ::= Sexp \cup Sexp \mid \{t \mid \varphi\} \mid At$$

[71] Cette condition sur les variables du premier ordre de la clause vient du fait que du point de vue où nous nous plaçons, il n'existe pas de différence entre symboles de fonctions et de prédicats, tous les deux étant considérés comme constructeurs d'arbres. Cette condition a pour but de distinguer clairement atomes et termes du programme.

où At est un atome clos.

Le processus d'approximation est décrit à l'aide d'une fonction notée App_{SC} définie pour de telles expressions comme suit : pour une expression de la forme $\{t \mid \varphi\}$,

$$App_{SC}(\{t \mid \varphi\}) = \begin{cases} \{y \mid \exists_{-y}\, \varphi \wedge y \in \{a\}\} & \text{si } t \text{ est une constante } a. \\ \{x \mid \exists_{-x}\, \varphi\} & \text{si } t \text{ est une variable } x. \\ f(App_{SC}(\{t_1 \mid \varphi\}), \ldots, App_{SC}(\{t_m \mid \varphi\})) & \text{si } t = f(t_1, \ldots, t_m). \end{cases}$$

Cette fonction est étendue pour tenir compte de l'union et des atomes clos simplement en posant $App_{SC}(Sexp_1 \cup Sexp_2) = App_{SC}(Sexp_1) \cup App_{SC}(Sexp_2)$ et $App_{SC}(At) = At$.

Exemple 18 : *En reprenant le programme de l'exemple 17, l'expression SE^P correspondante était égale à*

$$p(a,b) \cup p(c,a) \cup \{p(f(x,y)) \mid p(x,y) \in I \wedge x \in HU \wedge y \in HU\}$$

En appliquant la fonction d'approximation App_{SC}, $App_{SC}(SE^P)$ est égal à

$$p(a,b) \cup p(c,a)$$
$$\cup p(f(\{\mathbf{x} \mid \exists \mathbf{y},\ \mathbf{p(x,y)} \in \mathbf{I} \wedge \mathbf{x} \in \mathbf{HU} \wedge \mathbf{y} \in \mathbf{HU}\}, \{\mathbf{y} \mid \exists \mathbf{x},\ \mathbf{p(x,y)} \in \mathbf{I} \wedge \mathbf{x} \in \mathbf{HU} \wedge \mathbf{y} \in \mathbf{HU}\}))$$

Il est à noter que pour tout programme P, l'approximation de l'expression SE^P, c-à-d $App_{SC}(SE^P)$ est une expression qui est de la forme des expressions ensemblistes pouvant apparaître en membre gauche des contraintes définies généralisées.

Nous allons maintenant prouver le lien existant entre ce mécanisme opérationnel et la définition de l'analyse ensembliste donnée dans la section précédente. Ceci se prouve simplement en montrant que

$$\mathcal{T}_{P,\mathsf{T}_\Sigma}(I) = App_{SC}(SE^P)$$

On peut remarquer que du fait des définitions de $\mathcal{T}_{P,\mathsf{T}_\Sigma}$ et de App_{SC}, il est suffisant de tester cette égalité clause par clause. Ainsi, pour le programme P, cela se ramène à tester pour toute clause c de P, que

Proposition 24

$$\mathcal{T}_{\{c\},\mathsf{T}_\Sigma}(I) = App_{SC}(SE^{\{c\}})$$

Preuve :

Nous allons supposer que la clause c est de la forme $H \Leftarrow B_1, \ldots, B_m$ et nous allons confondre la variable ensembliste I et sa valeur.

Par définition, on peut affirmer qu'un atome clos $\alpha(H)$ (α étant une substitution close définie sur les variables de la clause c) appartient à $\mathcal{T}_{\{c\},\mathsf{T}_\Sigma}(I)$ si et seulement si

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma(B_1) \in I \wedge \ldots \wedge \sigma(B_m) \in I \wedge \bigwedge_{x \in Var_\mathcal{X}(c)} x \in HU\})$$

On peut alors remarquer de par la définition de App_α que cette dernière proposition est équivalente à : pour toute variable x du domaine de α, il existe une substitution σ de $\{\sigma \mid \sigma(B_1) \in I \wedge \ldots \wedge \sigma(B_m) \in I \wedge \bigwedge_{x \in Var_\mathcal{X}(c)} x \in HU\}$ telle que $\alpha(x) = \sigma(x)$.

Nous allons maintenant montrer une propriété qui implique la proposition à prouver : pour tout terme t[72]*,*

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\}) \Leftrightarrow \alpha(t) \in App_{SC}(\{t \mid \varphi\})$$

[72] le mot « terme » désigne simplement ici un objet construit avec pour signature les signatures fonctionnelles et de prédicat du programme P.

en supposant que φ est une conjonction d'atomes $t \in X$ et que le domaine de α contient les variables libres de t et des φ.

Par induction sur la structure du terme t :

- *si t est une constante ($t = a$), alors il est suffisant de montrer que*

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\}) \Leftrightarrow a \in \{y \mid \exists\_y\, \varphi \wedge y \in a\}$$

Pour toute substitution σ telle que $\sigma \models \varphi$, on peut considérer la substitution σ' définie comme σ pour les variables de t et de φ et telle que $\sigma'(y) = a$. Ainsi, $\sigma' \models \varphi \wedge y \in a$.
On a bien alors que la restriction du domaine des substitutions α' telles que

$$\alpha' \hat{\in} App_\alpha(\{\sigma' \mid \sigma' \models \varphi \wedge y \in \{a\}\})$$

aux variables de t et φ sont exactement les α telles que $App_\alpha(\{\sigma \mid \sigma \models \varphi\})$.

- *si t est une variable x, alors par définition de App_{SC}, il est équivalent de démontrer que*

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\}) \Leftrightarrow \alpha(x) \in \{x \mid \varphi\}$$

Or, nous avons vu que $\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\})$ est équivalent à dire que pour toute variable y du domaine α, il existe une substitution σ telle que $\sigma \models \varphi$ et $\alpha(y) = \sigma(y)$. Ainsi, cette propriété implique bien pour $x = y$, le fait que pour un σ bien choisi que $\sigma(x) = \alpha(x) \in \{x \mid \varphi\}$.
La réciproque vient alors du fait que cette propriété énoncée est vraie pour toute variable appartenant aux variables libres de t et de φ.

- *si $t = f(t_1, \ldots, t_n)$, alors par hypothèse de récurrence, pour tout i,*

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\}) \Leftrightarrow \alpha(t_i) \in App_{SC}(\{t_i \mid \varphi\})$$

Ainsi,

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\})$$
$$\Leftrightarrow$$
$$f(\alpha(t_1) \ldots, \alpha(t_m)) \in f(App_{SC}(\{t_1 \mid \varphi\}), \ldots, App_{SC}(\{t_n \mid \varphi\}))$$

Donc, par définition de App_{SC},

$$\alpha \hat{\in} App_\alpha(\{\sigma \mid \sigma \models \varphi\}) \Leftrightarrow \alpha(t) \in App_\alpha(f(t_1, \ldots, t_n) \mid \varphi\})$$

Puisque $f(t_1, \ldots, t_n)$ est égal à t la proposition est donc vraie dans ce cas.

Ceci étant vrai en particulier pour t égal à H, la tête de la clause c, la proposition est donc vraie.

■

On déduit de cette preuve que pour un programme logique P, $\mathcal{T}_{P,\mathsf{T}_\Sigma}(I) = App_{SC}(SE^P)$. Ainsi, par définition, les solutions de l'équation ensembliste $I = App_{SC}(SE^P)$ sont exactement les points fixes de l'opérateur ensembliste $\mathcal{T}_{P,\mathsf{T}_\Sigma}$ et en particulier, la plus petite solution de cette équation (et donc, le plus petit point fixe de $\mathcal{T}_{P,\mathsf{T}_\Sigma}$) est la sémantique ensembliste du programme P, *i.e.* DEN^{sba}_P.

De plus, par la proposition 25 (voir ci-dessous), la plus petite solution de l'équation $I = App_{SC}(SE^P)$ est également la plus petite solution de la contrainte $I \supseteq App_{SC}(SE^P)$, qui du fait de la forme de $App_{SC}(SE^P)$ se trouve être une contrainte définie généralisée. Ainsi, analyser de manière ensembliste un programme logique se ramène à calculer la plus petite solution d'un système de contraintes définies généralisées en appliquant l'algorithme proposé dans le chapitre 3.

Proposition 25 Soit $\mathcal{I}^=$ la plus petite solution d'un ensemble d'équations ensemblistes de la forme $\{Sexp_i = X_i \mid 1 \leq i \leq m\}$ telle que

- $Sexp_i$ est une expression ensembliste de la forme

$$Sexp_i ::= X \mid f(Sexp_i \ldots, Sexp_i) \mid Sexp_i \cup Sexp_i \mid \{x \mid \varphi(x)\}$$

- X_i n'apparaît que dans un seul membre droit d'équation de l'ensemble

alors $\mathcal{I}^=$ est également la plus petite solution du système de contraintes définies généralisées $\{Sexp_i \subseteq X_i \mid 1 \leq i \leq m\}$.

Preuve :

Notons tout d'abord qu'on peut démontrer (par induction sur la structure) que les expressions ensemblistes considérées sont monotones, c-à-d pour deux interprétations ensemblistes $\mathcal{I}$,$\mathcal{I}'$ telles que $\mathcal{I} \sqsubseteq \mathcal{I}'$, on a pour tout Sexp, $\mathcal{I}(Sexp) \subseteq \mathcal{I}'(Sexp)$.

Il est tout d'abord évident que toute solution de $\{Sexp_i = X_i \mid 1 \leq i \leq m\}$ est solution du système défini généralisé.

Supposons maintenant que $\mathcal{I}^=$ ne soit pas la plus petite solution de ce système. De ce cas, la plus petite solution de celui-ci $\mathcal{I}^\subseteq$ est telle que

- *il existe un j vérifiant $\mathcal{I}^\subseteq(Sexp_j) \subseteq \mathcal{I}^\subseteq(X_j)$ et $\mathcal{I}^\subseteq(Sexp_j) \neq \mathcal{I}^\subseteq(X_j)$, puisque $\mathcal{I}^\subseteq$ n'est pas une solution égalitaire,*
- *$\mathcal{I}^\subseteq \sqsubseteq \mathcal{I}^=$, ceci venant du fait que l'ensemble des solutions d'un système de contraintes définies généralisées est clos par intersection $\sqcap$.*

Définissons une interprétation $\mathcal{I}$ en posant :

- *$\mathcal{I}(X_i) = \mathcal{I}^\subseteq(Sexp_i)$ si X_i est une variable ensembliste apparaissant en membre gauche d'une contrainte,*
- *$\mathcal{I}(Y) = \mathcal{I}^\subseteq(Y)$, dans le cas contraire.*

Puisque $\mathcal{I}^\subseteq$ est une solution mais non-égalitaire, on a nécessairement $\mathcal{I} \sqsubseteq \mathcal{I}^\subseteq$ et $\mathcal{I} \neq \mathcal{I}^\subseteq$.

Or, par monotonie des expressions ensemblistes et par définition de $\mathcal{I}$, on a, pour tout i, $\mathcal{I}(Sexp_i) \subseteq \mathcal{I}^\subseteq(Sexp_i) = \mathcal{I}(X_i)$. Ainsi, $\mathcal{I}$ est une solution du système défini généralisé strictement plus petite que $\mathcal{I}^\subseteq$. Contradiction. ■

5.2.3 Exemple et Discussion

Nous allons débuter cette section en proposant un exemple et en donnant le résultat de l'analyse ensembliste sur celui-ci.

Exemple

Pour le programme logique P suivant, qui a pour but de tester l'égalité de deux listes,

$$eql([\,],[\,])$$
$$eql([x|l1],[x|l2]) \Leftarrow eql(l1,l2)$$

on obtient la définition de T_{P,T_Σ} suivante :

$$T_{P,\mathsf{T}_\Sigma}(I) = eql([\,],[\,]) \cup \left\{ eql([x|l1],[x|l2]) \;\middle|\; \begin{array}{l} eql(l1,l2) \in I \wedge x \in HU \wedge \\ l1 \in HU \wedge l2 \in HU \end{array} \right\}$$

Le processus d'approximation de l'analyse ensembliste produit le système de contraintes (partiellement aplati) suivant :

$$eql([\,],[\,]) \subseteq I$$
$$eql([X|L1],[X|L2]) \subseteq I$$
$$\left\{ x \;\middle|\; \begin{array}{l} \exists l1,\, \exists l2, eql(l1,l2) \in I \wedge x \in HU \wedge \\ l1 \in HU \wedge l2 \in HU \end{array} \right\} \subseteq X$$
$$\left\{ l1 \;\middle|\; \begin{array}{l} \exists x,\, \exists l2, eql(l1,l2) \in I \wedge x \in HU \wedge \\ l1 \in HU \wedge l2 \in HU \end{array} \right\} \subseteq L1$$
$$\left\{ l2 \;\middle|\; \begin{array}{l} \exists x,\, \exists l1, eql(l1,l2) \in I \wedge x \in HU \wedge \\ l1 \in HU \wedge l2 \in HU \end{array} \right\} \subseteq L2$$
$$[\,] \cup zero \cup s(HU) \cup [HU|HU] \subseteq HU$$

La dernière contrainte donnant la représentation explicite de HU par rapport à la signature fonctionnelle Σ supposée ici égale à $\{[\,]/0, zero/0, s/1, [\cdot|\cdot]/2\}$.

La représentation sous forme de grammaire de la plus petite solution de ce système de contraintes définies généralisées est :

$$I \Rightarrow eql([\,],[\,]) \mid eql([X|L1],[X|L2])$$
$$X \Rightarrow HU$$
$$HU \Rightarrow [\,] \mid zero \mid s(HU) \mid [HU|HU]$$
$$L1 \Rightarrow [\,] \mid [X|L1]$$
$$L2 \Rightarrow [\,] \mid [X|L2]$$

Le résultat de cette analyse stipule simplement que les succès pour le prédicat eql ne peuvent être que des couples de listes $(l1, l2)$. L'analyse est en fait un petit peu plus précise, puisqu'elle affirme que soit les deux listes du couples sont vides, soit elles sont toutes les deux non-vides.

Discussion

L'apport principal de la présentation qui a été faite de l'analyse ensembliste dans la section précédente est la description à la fois précise et formelle de la technique d'extraction des contraintes ensemblistes d'un programme logique par rapport aux techniques présentées dans [40] et [36]. Ceci permet notamment d'exprimer simplement le lien existant entre la définition formelle de l'analyse ensembliste et le mécanisme plus opérationnel lié aux contraintes ensemblistes. De plus, cette simplicité rend ce mécanisme pleinement effectif pour ce qui est d'une implantation.

En ce qui concerne la technique de l'analyse ensembliste en elle-même, elle peut sembler trop simple pour produire un mécanisme utilisable en pratique. Cependant, comme l'affirme

Heintze dans [37], cette technique améliore de nombreux travaux antérieurs autant sur le plan de la simplicité de son formalisme que sur la précision de l'analyse. L'implantation de Heintze présentée dans [36] s'avère de plus efficace sur des programmes de taille moyenne, malgré la nature « exponentielle » de la complexité de l'algorithme [73]. Cet aspect de la complexité de l'analyse ensembliste, ou plus généralement du « typage souple » se retrouve dans divers autres problèmes d'analyse statique de programmes logiques et semble étroitement liée à ce paradigme de programmation [74].

Cependant, l'analyse ensembliste s'avère parfois très insuffisante sur le plan de la précision, y compris sur des programmes très simples. Nous allons exhiber un tel exemple et montrer comment deux techniques différentes (et de plus, combinables) peuvent pallier cette insuffisance. Ces aspects plus techniques feront l'objet d'une sous-section. Mais nous allons tout d'abord donner une autre présentation de l'analyse ensembliste en terme de transformations de programmes logiques.

5.2.4 Une approche par transformations de programmes

Nous avons déjà mentionné dans la section 2.4.4 que les travaux de Früwirth, Shapiro, Vardi et Yardeni [28] avaient pour but de clarifier ceux présentés par Heintze et Jaffar dans [40]. Le point de vue de ce propos était alors uniquement celui de la « résolution de contraintes avec opérateurs intensionnels ». En fait, ceci doit être pris dans un cadre plus général, dans le sens où cette remarque est également valide du point de vue de l'analyse ensembliste de programmes logiques, principale motivation de [40]. En effet, dans [28], Früwirth, Shapiro, Vardi et Yardeni avancent le fait (comme le titre de cet article l'indique) que les programmes logiques eux-même sont de « bons descripteurs » du typage d'un programme logique.

Ainsi, les programmes uniformes dont nous avons parlés dans la section 2.4.4 ont été introduits dans [28] dans le but d'exprimer l'analyse ensembliste de programmes logiques par une méthode de transformation de programmes. Nous rappelons qu'un programme est dit uniforme si les symboles de prédicats qui y apparaissent sont unaires et si ces clauses sont de la forme $p(t) \Leftarrow p_1(t_1), \ldots, p_l(t_l)$, où t est un terme linéaire [75].

Nous allons considérer un programme logique P en supposant sans perte de généralité que deux clauses différentes de P ne partagent pas de variables. La première transformation appliquée à P est de rendre unaire les symboles de prédicats. Ceci est réalisé en ajoutant à la signature fonctionnelle du programme un symbole f_p pour tout symbole de prédicat p de P et en ne considérant plus qu'un seul symbole de prédicat noté *type*. Dans toutes les clauses du programme, on remplace l'atome $p(t_1, \ldots, t_m)$ par l'atome $type(f_p(t_1, \ldots, t_m))$. De plus, afin d'assurer une correction de cette transformation vis-à-vis de la signature originelle de P, dans chaque clause, pour toute variable x de celle-ci, on ajoute l'atome $all(x)$. On termine cette première phase en complétant le programme avec la définition du prédicat *all*, *i.e.* pour chaque symbole f d'arité m de la signature originelle de P, on ajoute la clause $all(f(x_1, \ldots, x_l)) \Leftarrow all(x_1), \ldots, all(x_m)$ où les variables $x_1, \ldots, x_m$ sont deux-à-deux distinctes.

Nous noterons P' le programme ainsi obtenu.

Exemple 19 : *Pour le programme P testant l'égalité de deux listes donné à la section 5.2.3, on obtient le programme P' suivant (en supposant que la signature fonctionnelle du programme*

[73] Les travaux antérieurs (donnant des résultats comparables en précision) que l'analyse ensembliste améliore étaient eux-aussi de nature « exponentielle ».

[74] Cette remarque est dûe à Debray.

[75] Un terme t dit « linéaire » si chaque variable possède une seule occurrence.

P est $\{[]/0, zero/0, s/1, [\cdot|\cdot]/2\}$*) :*

$$type(f_{eql}([\,],[\,]))$$
$$type(f_{eql}([x|l1],[x|l2])) \Leftarrow type(f_{eql}(l1,l2)), all(x), all(l1), all(l2)$$
$$all([\,])$$
$$all(zero)$$
$$all(s(y)) \Leftarrow all(y)$$
$$all([z|w]) \Leftarrow all(z), all(w)$$

On peut remarquer que cette transformation préserve la sémantique dans le sens suivant : pour un atome clos $p(\tau_1, \ldots, \tau_m)$,

$$p(\tau_1, \ldots, \tau_m) \in DEN_{\mathsf{T}_\Sigma}(P) \text{ ssi } type(f_p(\tau_1, \ldots, \tau_m)) \in DEN_{\mathsf{T}_\Sigma}(P')$$

Notons tout d'abord que les clauses définissant le prédicat *all* sont des clauses uniformes ; elles ne seront pas concernées par la seconde phase de transformation. Cette seconde phase assure, elle, l'approximation. Cette transformation se fait clause par clause :

considérons une clause de P' de la forme $type(t) \Leftarrow B_1, \ldots, B_l$, où t est un terme quelconque et $B_1, \ldots, B_l$ sont des atomes. Soit t', un terme linéaire dont les variables n'apparaissent pas dans le programme P' et tel qu'il existe une substitution σ associant à chaque variable de t' une variable de t telle que $\sigma(t') = t$. La transformation consiste à remplacer la clause $type(t) \Leftarrow B_1, \ldots, B_l$ par les clauses :

$$type(t') \Leftarrow \bigwedge_{x \in Var_{\mathcal{X}}(t')} p_{\sigma(x)}(x)$$
$$\bigwedge_{x \in Var_{\mathcal{X}}(t)} (p_{\sigma(x)}(\sigma(x)) \Leftarrow B_1, \ldots, B_l)$$

Les $p_{\sigma(x)}$ sont de nouveaux symboles de prédicats unaires introduits pour chacune des variables apparaissant dans la tête de la clause à transformer. L'idée intuitive de cette transformation est de dire que le prédicat $p_{\sigma(x)}$ est le type de la variable $\sigma(x)$.

Exemple 20 : *Pour le programme P' de l'exemple 19, la seule clause concernée par cette transformation est*

$$type(f_{eql}([x|l1],[x|l2])) \Leftarrow type(f_{eql}(l1,l2)), all(x), all(l1), all(l2)$$

et est remplacée par les clauses :

$$type(f_{eql}([x_1|l1]),[x_2|l2]))) \Leftarrow p_x(x_1), p_{l1}(l1), p_x(x_2), p_{l2}(l2)$$
$$p_x(x) \Leftarrow type(f_{eql}(l1,l2)), all(x), all(l1), all(l2)$$
$$p_{l1}(l1) \Leftarrow type(f_{eql}(l1,l2)), all(x), all(l1), all(l2)$$
$$p_{l2}(l2) \Leftarrow type(f_{eql}(l1,l2)), all(x), all(l1), all(l2)$$

Le programme ainsi obtenu est noté P_{sba}.

L'équivalence de cette approche de l'analyse ensembliste par transformation de programme et les approches précédentes est montrée dans [28] et est résumée par le fait : soit $p(\tau_1, \ldots, \tau_m)$ un atome clos, $p(\tau_1, \ldots, \tau_m) \in DEN_P^{sba}$ si et seulement si $type(f_p(\tau_1, \ldots, \tau_m)) \in DEN_{\mathsf{T}_\Sigma}(P_{sba})$.

Le programme issu de ces deux phases de transformation est un programme uniforme. Ainsi, par l'approche de [28] résumée à la section 2.4.4, on peut représenter le plus petit modèle de Herbrand d'un tel programme à l'aide d'un automate ascendant d'arbres finis.

Nous avons également proposé dans [24] une méthode très semblable à celle décrite dans le chapitre 3 pour donner cette représentation sous la forme d'un automate ascendant déterministe et complet d'arbres finis de rang n. Le rang n correspond en fait aux nombres de symboles de prédicats du programme uniforme[76], et on associe une composante du langage reconnu par l'automate à chacun des symboles de prédicats unaires.

Dans la section suivante, nous allons montrer les limitations de l'analyse ensembliste[77] et proposer deux solutions visant à améliorer la précision de l'analyse.

5.3 Limitations et solutions

Dans cette section, nous allons voir que l'analyse ensembliste peut sur des programmes se révéler peu précise, y compris pour des programmes simples. Nous allons présenter deux méthodes permettant de limiter cette imprécision : la première de ces deux méthodes se base sur l'aspect « logique » des programmes logiques, et peut être vue comme une méthode basée sur la théorie des modèles. La seconde est plus technique et se base sur une méthode de transformation de programmes, transformant l'analyse ensembliste (qui s'appuie sur la sémantique de point fixe) en une méthode dirigée par le but.

L'exemple « classique »[78] pour lequel l'analyse ensembliste se révèle peu précise est le programme codant le renversement d'une liste avec un accumulateur :

Exemple 21 :

$$rev(L, L') \Leftarrow rev\text{-}acc(L, [\,], L')$$
$$rev\text{-}acc([\,], L'', L'')$$
$$rev\text{-}acc([X|L1], L2, L3) \Leftarrow rev\text{-}acc(L1, [X|L2], L3)$$

L'analyse ensembliste de ce programme a pour résultat la grammaire suivante :

$$I \Rightarrow rev(L, L') \mid rev\text{-}acc([\,], L'', L'') \mid rev\text{-}acc([X|L1], L2, L3)$$
$$L'' \Rightarrow \top$$
$$X \Rightarrow \top$$
$$L2, L3 \Rightarrow \top$$
$$L1 \Rightarrow [\,] \mid [X|L1]$$
$$L' \Rightarrow \top$$
$$L \Rightarrow L1$$

[76] Pour être plus précis, aux nombres de symboles de prédicats du programme uniforme transformé dans une forme appelée *quasi-automate*.

[77] Cette limitation n'est pas propre à cette technique d'analyse et se retrouve dans toutes celles qui se basent sur la sémantique de point fixe.

[78] Il est utilisé dans [28] comme critique de l'analyse ensembliste et plus généralement des méthodes définies à partir de la sémantique de point fixe, méthodes appelées « bottom-up ».

où $\top$ est l'ensemble de tous les termes clos.

En résumé, ce résultat, en ce qui concerne le prédicat *rev*, affirme que les atomes succès de celui-ci ne peuvent avoir qu'une liste pour premier argument, mais ne donne aucune information concernant le second argument (affirmant qu'il s'agit d'un terme clos). Or, il est évident que pour ce prédicat, le second argument doit lui aussi être une liste, mais ce fait n'est pas capturé par l'analyse ensembliste.

Intuitivement, ce manque de précision provient du fait que le prédicat *rev-acc* est défini à l'aide du fait $rev\text{-}acc([], L'', L'')$ qui possède des succès pour n'importe quelle valeur de L''. L'analyse ensembliste n'est pas suffisamment fine pour tenir compte du fait que *rev* utilise *rev-acc* de manière particulière par l'atome $rev\text{-}acc(L, [], L')$.

5.3.1 Analyse statique basée sur les modèles

Dans [29], Gallagher, Boulanger et Saglam proposent une méthodologie d'analyse statique pour les programmes logiques basée sur les fondements même de la programmation logique (et de la logique), à savoir la théorie des modèles. L'idée de cette approche est de fixer une algèbre, adaptée à la nature de la propriété à analyser, et de calculer le plus petit modèle du programme logique dans cette algèbre[79].

Cette approche peut très simplement être utilisée pour calculer une sur-approximation de l'ensemble des conséquences logiques d'un programme. Nous allons le montrer sur un exemple.

Exemple 22 : *Considérons le programme suivant codant l'addition :*

$$add(zero, W, W)$$
$$add(s(X), Y, s(Z)) \Leftarrow add(X, Y, Z)$$

et l'algèbre finie A *ayant pour support* $\{\mathsf{entier}, \mathsf{non\text{-}entier}\}$ *et associant aux éléments de la signature* $\{zero/0, s/1, f/1\}$ *les fonctions suivantes :*

$$zero^{\mathsf{A}} = \mathsf{entier}$$
$$s^{\mathsf{A}}(\mathsf{entier}) = \mathsf{entier}$$
$$s^{\mathsf{A}}(\mathsf{non\text{-}entier}) = \mathsf{non\text{-}entier}$$
$$f^{\mathsf{A}}(\mathsf{entier}) = \mathsf{non\text{-}entier}$$
$$f^{\mathsf{A}}(\mathsf{non\text{-}entier}) = \mathsf{non\text{-}entier}$$

Nous pouvons remarquer que pour chacun des éléments du support de l'algèbre a_{A}*, il existe un terme clos* τ *tel que* $\lfloor \tau \rfloor_{\mathsf{A}} = a_{\mathsf{A}}$.

Il est très simple de calculer le plus petit modèle du programme dans l'algèbre A *; ce plus petit modèle est l'ensemble (nécessairement fini) de* A*-atomes*

$$\{add(\mathsf{entier}, \mathsf{entier}, \mathsf{entier}), add(\mathsf{entier}, \mathsf{non\text{-}entier}, \mathsf{non\text{-}entier})\}$$

Si on considère l'algèbre fini comme un automate d'arbres dont les états sont les éléments du support alors on obtient bien une approximation régulière de la dénotation du programme.

Il existe un lien indirect entre l'analyse ensembliste d'un programme logique et la notion d'algèbre finie au travers de l'algorithme de résolution des contraintes ensemblistes définies généralisées : rappelons tout d'abord que l'analyse ensembliste d'un programme logique revient

[79] Pour que ce calcul soit effectif et en particulier, termine, il est nécessaire que le domaine de l'algèbre soit fini.

essentiellement à résoudre un système de contraintes définies avec expressions d'appartenance simples. Rappelons également que résoudre un tel système de contraintes (en fait, calculer la plus petite solution) revient par l'algorithme présenté au chapitre 3 à calculer un automate d'arbres finis, représentant une algèbre dont le domaine est l'ensemble des états accessibles, les règles de transition de l'automate donnant l'interprétation des symboles de fonction dans cette algèbre, algèbre que nous avons notée $\mathsf{A}_{rch(\mathcal{A})}$.

On peut alors se demander si l'analyse ensembliste d'un programme logique P, produisant un automate $\mathcal{A}$, correspond au plus petit modèle de P dans l'algèbre $\mathsf{A}_{rch(\mathcal{A})}$, *i.e.* à $T_{P,\mathsf{A}_{rch(\mathcal{A})}}$. La réponse à cette question est négative et nous allons en donner un contre-exemple.

Considérons le programme logique suivant :

$$p(X,X)$$
$$q(a)$$
$$r(b)$$

construit sur le signature fonctionnelle ne contenant que les deux constantes a et b.

L'analyse ensembliste de ce programme produit un automate, qui en se focalisant uniquement sur les états accessibles [80] et sur les règles impliquant des symboles de fonction, ici a et b, contient les deux règles suivantes :

$$\mathbf{a} \rightarrow \{q, p_1, p_2\}$$
$$\mathbf{b} \rightarrow \{r, p_1, p_2\}$$

Le plus petit modèle du programme dans cette algèbre est

$$\mathbf{q}(\{q,p_1,p_2\}), \mathbf{r}(\{r,p_1,p_2\}), \mathbf{p}(\{q,p_1,p_2\},\{q,p_1,p_2\}), \mathbf{p}(\{r,p_1,p_2\},\{r,p_1,p_2\})$$

L'ensemble des atomes clos représentés par ce modèle sont $\{q(a), r(b), p(a,a), p(b,b)\}$, tandis que la sémantique ensembliste de ce programme est défini par l'ensemble fini (et donc régulier) $\{q(a), r(b), p(a,a), p(b,b), p(a,b), p(b,a)\}$.

Il est en fait très simple de démontrer que l'analyse ensembliste, c-à-d l'ensemble d'atomes clos approximant la sémantique du programme, peut être défini à partir de $\mathcal{T}_{P,\mathsf{A}_{rch(\mathcal{A})}} \uparrow \omega$, correspondant au plus petit point fixe de l'opérateur de conséquence logique ensembliste, calculé dans l'algèbre $\mathsf{A}_{rch(\mathcal{A})}$. Cette relation peut être décrite par $p(\tau_1, \ldots, \tau_m) \in DEN_P^{sba}$ si et seulement si $p(\lfloor \tau_1 \rfloor_{\mathsf{A}_{rch(\mathcal{A})}}, \cdots, \lfloor \tau_m \rfloor_{\mathsf{A}_{rch(\mathcal{A})}}) \in \mathcal{T}_{P,\mathsf{A}_{rch(\mathcal{A})}} \uparrow \omega$.

Or, il se trouve que pour une algèbre quelconque A, si E_{A} est un ensemble de A-atomes, alors $T_{P,\mathsf{A}}(E_{\mathsf{A}}) \subseteq \mathcal{T}_{P,\mathsf{A}}(E_{\mathsf{A}})$ [81]. On peut donc en déduire, l'inclusion précédente étant généralement stricte, une méthode permettant d'augmenter la précision de l'analyse ensembliste, en calculant simplement $T_{P,\mathsf{A}_{rch(\mathcal{A})}}$, où $\mathcal{A}$ est l'automate produit par la résolution du système de contraintes ensemblistes provenant de l'analyse de P. La complexité de ce calcul est similaire à celui-ci de la phase d'analyse ensembliste proprement dite, puisque ce calcul de point fixe se réalise en temps linéaire par rapport à la taille de l'automate.

Nous allons reprendre l'exemple du programme de renversement d'une liste avec accumulateur :

[80] Le sens que l'on peut donner à l'état $\{q, p_1, p_2\}$ est qu'il désigne les termes succès pour q et qui apparaissent comme premier (p_1) et second argument (p_2) dans les succès pour p.

[81] Ceci est la base de la correction de l'analyse ensembliste, lorsqu'elle est décrite au moyen de l'opérateur $\mathcal{T}_{P,\mathsf{T}_\Sigma}$.

$$rev(L, L') \Leftarrow rev\text{-}acc(L, [\,], L')$$
$$rev\text{-}acc([\,], L'', L'')$$
$$rev\text{-}acc([X|L1], L2, L3) \Leftarrow rev\text{-}acc(L1, [X|L2], L3)$$

Nous allons supposer que $\{[\,]/0, zero/0, s/1, [\cdot|\cdot]/2\}$ est la signature fonctionnelle de ce programme. L'automate produit par l'analyse ensembliste sera de la forme suivante[82] :

$$\begin{array}{ll} [\,] \to q_{nil} & zero \to q_o \\ s(q_{nil}) \to q_o & s(q_o) \to q_o \\ s(q_{l+}) \to q_o & [q_{nil}|q_{nil}] \to q_{l+} \\ [q_o|q_{nil}] \to q_{l+} & [q_{l+}|q_{nil}] \to q_{l+} \\ [q_{nil}|q_{l+}] \to q_{l+} & [q_o|q_{l+}] \to q_{l+} \\ [q_{l+}|q_{l+}] \to q_{l+} & [q_{nil}|q_o] \to q_o \\ [q_o|q_o] \to q_o & [q_{l+}|q_o] \to q_o \end{array}$$

Intuitivement, l'état q_{nil} représente la liste vide $[\,]$, l'état q_{l+} les listes non vides et l'état q_o tous les termes clos de la signature qui ne sont pas des listes.

Cet automate est vu comme une algèbre, dont les éléments du support sont les états. Le plus petit modèle du programme dans cette algèbre est :

$$\begin{array}{ll} rev\text{-}acc(q_{nil}, q_{nil}, q_{nil}), & rev\text{-}acc(q_{l+}, q_{nil}, q_{l+}), \\ rev\text{-}acc(q_{nil}, q_o, q_o), & rev\text{-}acc(q_{l+}, q_{l+}, q_{l+}), \\ rev\text{-}acc(q_{nil}, q_{l+}, q_{l+}), & rev(q_{nil}, q_{nil}), \\ rev\text{-}acc(q_{l+}, q_o, q_o) & rev(q_{l+}, q_{l+}) \end{array}$$

Ce plus petit modèle traduit une approximation qui stipule que les succès du prédicat *rev* sont obtenus soit lorsque les deux arguments sont égaux à la liste vide, soit lorsque ceux-ci sont tous les deux des listes non-vides.

Il est clair sur cet exemple que cette méthode permet d'améliorer sensiblement la méthode d'analyse ensembliste « pure ». Ce principe peut se décrire comme le fait que l'analyse ensembliste est utilisée pour générer à partir du programme un système de typage « pertinent », *i.e.* une algèbre dont les éléments du domaine sont des types et l'interprétation des symboles de fonction sont les « profils » (ou déclarations) de ces derniers. Le programme alors est simplement typé dans ce système par un simple calcul du plus petit point fixe de l'opérateur de conséquence logique pour cet ensemble de types.

Nous allons voir dans la sous-section suivante une autre technique permettant elle aussi d'améliorer la précision de l'approximation produite par l'analyse ensembliste.

5.3.2 Analyse ensembliste dirigée par le but

La principale limitation de l'analyse ensembliste des programmes logiques vient du fait qu'elle se base sur l'opérateur de conséquence logique immédiate. On qualifie généralement ce type d'analyse de « bottom-up ». Comme nous l'avons déjà dit, pour le programme codant le renversement d'une liste avec accumulateur, l'analyse ensembliste du fait de la perte de dépendance

[82] L'automate produit est en réalité beaucoup plus grand. Cependant, celui donné ici peut être vu comme une « approximation » de l'automate réellement produit pour lequel l'état q_o regroupe tous les états différents de q_{nil} et q_{l+}.

inter-variables qu'elle engendre, ne peut prendre en compte l'utilisation « particulière» par le prédicat *rev* du prédicat *rev-acc*.

Pour répondre à ce problème, il conviendrait donc que l'analyse ensembliste ne soit plus globale, mais puisse prendre en compte de manière isolée le prédicat *rev*. Ceci revient à considérer une analyse dirigée par un but, en l'occurrence ici, $\Leftarrow rev(l, l')$, dont le résultat est précisément un sur-ensemble des instances closes de $rev(l, l')$, succès pour ce but.

Dans cette optique, Heintze a proposé dans [36, 37] une approche de l'analyse ensembliste dirigée par un but (on parle alors d'analyse « top-down»). Cette méthode est prouvée correcte dans [37] dans le sens où pour un programme P et un but G, l'analyse produit un sur-ensemble (régulier) des instances closes de G succès pour le programme P. L'extraction des contraintes d'un programme est alors différente de celle que nous avons présentée, et simule d'une certaine façon le comportement de la résolution OLD sur le programme pour le but donné. Le résultat de l'application de cette méthode d'extraction est un ensemble de contraintes définies, dont la plus petite solution (pour une variable correspondant au but) donne l'approximation désirée. Par exemple, pour le programme de l'exemple 21 et le but $\Leftarrow rev(l, l')$, on obtient pour approximation pour la variable l', l'ensemble des listes.

Cependant, développer une nouvelle approche, comme l'a fait Heintze, peut être évité comme il est mentionné dans [30]. Dans cet article, Gallagher et de Waal proposent une méthode d'approximation régulière de programmes logiques de type « bottom-up» [83] et font également allusion à ce manque de précision de ce type d'analyse. La solution qu'ils proposent est d'utiliser un mécanisme *ad-hoc* de transformation de programmes logiques, appelé *transformation « des ensembles magiques»*. Ce type de transformation a été introduit dans le cadre des bases de données déductives [9] et vise à fournir un mécanisme d'évaluation des réponses à une requête donnée (*i.e.* un but) plus efficace que la SLD-résolution. Cette transformation pour un programme P et un but donné $\Leftarrow p(t_1, \ldots, t_m)$ produit un nouveau programme logique pour lequel les conséquences logiques du programme pour un nouveau prédicat ans_p sont exactement les instances closes du but $\Leftarrow p(t_1, \ldots, t_m)$ conséquences logiques du programme P. Ce programme issu de la transformation simule le comportement de la SLD-résolution (pour une règle de sélection fixée pour chacune des clauses) pour le but $\Leftarrow p(t_1, \ldots, t_m)$ et le programme P. Ainsi, dans le cas de bases de données déductives, il est possible d'utiliser en lieu et place de la SLD-dérivation, un calcul par chaînage avant des conséquences du nouveau programme en considérant l'opérateur de point fixe qui lui est associé.

Ainsi, Gallagher et de Waal proposent la démarche suivante : pour calculer pour un symbole de prédicat p d'arité m, une sur-approximation de l'ensemble des atomes clos de la forme $p(\tau_1, \ldots, \tau_m)$, conséquence logique de P, il suffit d'appliquer une transformation « des ensembles magiques» de P avec le but $\Leftarrow p(x_1, \ldots, x_m)$, où les variables $x_1, \ldots, x_m$ sont deux-à-deux distinctes. Gallagher et de Waal soulignent le fait que la démarche prise par Heintze dans [36] est en tout point similaire à la leur.

Cette approche, contrairement à celle proposée dans [36] a l'avantage de ne nécessiter aucune preuve de correction puisque celle-ci est immédiatement déductible de la correction de l'approximation de l'opérateur T_{P,T_Σ} et de la transformation « des ensembles magiques».

L'idée de cette transformation « des ensembles magiques» peut être vue comme la simulation du comportement d'une SLD-résolution avec une règle de sélection des atomes fixée (nous allons considérer ici la OLD-résolution, *i.e.* le choix systématique de l'atome le plus gauche dans la

[83] Cette méthode, bien que moins précise que l'analyse ensembliste que nous avons présentée, se veut principalement plus efficace

résolvante courante) pour un programme et un but donnés. Cette transformation correspond en fait à une description très procédurale de la résolution, distinguant les notions des valeurs d'appels et de valeurs retournées.

Considérons le but $\Leftarrow p(x)$ et le programme suivant

$$p(x) \Leftarrow q(x), r(x)$$
$$q(a)$$
$$q(b)$$
$$r(a)$$

Le but $\Leftarrow p(x)$ correspond donc à un appel au prédicat p. Le calcul de la « valeur retournée » par le prédicat p peut être décrite selon le mécanisme suivant : un appel à $p(x)$ a du être effectué et des valeurs ont du être retournées par $q(x)$ et $r(x)$. Ceci peut être schématisé en utilisant des nouveaux symboles de prédicats $call\_p$ et ans_p, correspondant aux appels et aux retours du prédicat p, par la clause :

$$ans\_p(x) \Leftarrow call\_p(x), ans\_q(x), ans\_r(x)$$

L'appel à $r(x)$ se fait quant-à-lui après un appel à $p(x)$ et est précédé d'un retour de résultat pour $q(x)$. La valeur calculée par cet appel l'est simplement par un appel à la dernière clause. Ceci est schématisé par les deux clauses suivantes :

$$call\_r(x) \Leftarrow call\_p(x), ans\_q(x)$$
$$ans\_r(a) \Leftarrow call\_r(a)$$

On ajoute de plus à ce programme le seul fait de celui-ci, qui traduit qu'un appel à $p(x)$ a été réalisé, *i.e.* tout simplement le fait $call\_p(x)$.

Exemple 23 : *En appliquant la transformation « des ensembles magiques » au programme de l'exemple « renverse » et au but* $\Leftarrow rev(L, L')$*, on obtient le programme suivant :*

$ans\_rev(L, L') \Leftarrow call\_rev(L, L'), ans\_rev\text{-}acc(L, [\,], L')$
$ans\_rev\text{-}acc([\,], L'', L'') \Leftarrow call\_rev\text{-}acc([\,], L'', L'')$
$ans\_rev\text{-}acc([X|L1], L2, L3) \Leftarrow call\_rev\text{-}acc([X|L1], L2, L3), ans\_rev\text{-}acc(L1, [X|L2], L3)$

$call\_rev\text{-}acc(L, [\,], L') \Leftarrow call\_rev(L, L')$
$call\_rev\text{-}acc(L1, [X|L2], L3) \leftarrow call\_rev\text{-}acc([X|L1], L2, L3)$

$call\_rev(L, L')$

L'application de l'analyse ensembliste à ce programme permet d'obtenir pour le prédicat ans_rev *(i.e. le prédicat* rev *du programme initial) que ses succès sont nécessairement de la forme* $ans\_rev(\tau, \tau')$*, où* τ,τ' *sont soit tous les deux égaux à la liste vide, soit deux listes non-vides.*

L'avantage de l'utilisation de cette transformation « des ensembles magiques » pour mener à bien une analyse dirigée par le but est de séparer celle-ci de la phase d'approximation de l'opérateur de conséquence logique. Ainsi, toute amélioration de cette dernière phase d'approximation

entraîne son intégration immédiate dans une méthode dirigée par le but. En particulier, la méthode basée sur les modèles décrite à la sous-section précédente peut parfaitement s'appliquer sur le programme issu de la transformation des « ensembles magiques ».

Nous allons dans la section suivante présenter une autre analyse ensembliste pour des programmes logiques pour lesquels la sémantique d'intérêt n'est pas donnée par le plus petit modèle de Herbrand du programme. Ces programmes logiques sont dits « réactifs » et la sémantique qui leur est associée capture les comportements succès (donc finis) et les comportements infinis vis-à-vis de la SLD-résolution.

5.4 Analyse de programmes logiques réactifs

Si l'analyse ensembliste des programmes logiques définis a motivé l'introduction de la classe des contraintes ensemblistes définies, nous allons voir dans cette section, la motivation de la classe des contraintes co-définies [17, 18], ainsi que celle du choix (habituel) pour cette classe d'une résolution sur les ensembles d'arbres finis et infinis. Cette motivation est l'analyse de programme logique réactif dont la sémantique tient compte des comportements finis succès d'un programme, mais également des comportements infinis. Une méthode d'extraction des contraintes a été proposée par Charatonik et Podelski dans [17] ; Ces contraintes appartiennent à la classe de contraintes co-définies et le calcul de la plus grande solution de celles-ci donne une approximation de la sémantique du programme logique réactif.

Nous allons dans cette section donner une formalisation de l'analyse ensembliste des programmes logiques réactifs très voisines de celle donnée pour l'approximation de la sémantique « classique » des programmes logiques et proposer une méthode d'extraction correspondante basée sur une définition de la sémantique d'un programme logique réactif à l'aide d'un opérateur de point fixe. Nous verrons que ces contraintes appartiennent à la classe des contraintes co-définies avec expressions d'appartenance simples[84].

5.4.1 Sémantique « réactive » des programmes logiques

Un *programme logique réactif* (appelé aussi « processus perpétuel » dans [45]) correspond syntaxiquement à un programme logique défini. C'est donc sur le plan sémantique qu'apparaît la différence. L'idée est d'associer à un programme logique une sémantique ne prenant pas en compte uniquement les comportements terminants sous la forme d'un succès d'un programme au sens de la SLD-résolution. En effet, la sémantique usuelle d'un programme logique donnée sous la forme du plus petit modèle de Herbrand capture le fait que si un atome clos A appartient à ce plus petit modèle, alors il existe une SLD-dérivation succès fini pour le but $\Leftarrow A$. Cependant dans diverses situations, il peut être intéressant de savoir que pour un but donné, il existe une SLD-dérivation infinie et donc, n'entraînant pas d'échecs finis pour ce but. C'est ce comportement à l'infini que vise à modéliser la sémantique des programmes logiques réactifs.

Si l'on souhaite pouvoir modéliser de tels comportements, il va de soi que l'univers de Herbrand (dont le domaine est un ensemble de termes clos et, donc finis) ne convient plus. L'univers de Herbrand est donc *complété* et il correspond alors à l'algèbre des arbres finis et infinis étiquetés sur Σ, la signature fonctionnelle du programme logique considéré. L'*univers de Herbrand complet* ainsi défini est noté HU^*[85] et la base de Herbrand qui lui est associée (comme l'ensemble de

[84] Ce travail a été publié dans [26].

[85] Nous utiliserons la notation équivalente $\mathrm{Tr}^{\omega}_{\Sigma}$.

tous les HU^*-atomes), dite *base de Herbrand complète*, est notée BH^*.

Ainsi, un atome (de la base de Herbrand complète) A appartient à la sémantique d'un programme logique réactif P s'il existe pour le but $\Leftarrow A$ soit une SLD-dérivation succès, soit une SLD-dérivation infinie[86].

Exemple 24 : *Considérons le programme logique (réactif) P défini par l'unique clause*

$$p(f(x)) \Leftarrow p(x)$$

et le but $\Leftarrow p(\tau)$, où τ est l'arbre tel que $dom(\tau) = \{1\}^\omega$ et $\forall d \in dom(\tau)$, $\tau(d) = f$. Par définition de $\vdash_{SLD}$, la séquence infinie $G_0, G_1, \ldots, G_n, \ldots$ telle que pour tout i, $G_i = \Leftarrow p(\tau)$, constitue une SLD-dérivation infinie. Ainsi, l'atome $p(\tau)$ appartient à la sémantique « réactive » du programme P.

La sémantique « réactive » d'un programme logique P peut, comme dans le cas de la sémantique « classique », être définie à l'aide d'un opérateur de point fixe. Cet opérateur, noté $T^*_{\mathsf{Tr}^\omega_\Sigma, P}$ est défini comme suit :

$$T^*_{P,\mathsf{Tr}^\omega_\Sigma}(I) = \left\{ H \;\middle|\; \begin{array}{l} H \Leftarrow B_1, \ldots, B_l \in INST_{\mathsf{Tr}^\omega_\Sigma}(P) \\ B_1, \ldots, B_l \in I \end{array} \right\}$$

où I est un sous-ensemble de BH^* et $INST_{\mathsf{Tr}^\omega_\Sigma}(P)$, l'ensemble des $\mathsf{Tr}^\omega_\Sigma$-instances des clauses de P.

L'idée de la sémantique réactive étant de capturer les comportements succès et les comportements non-terminants d'un programme logique, (*i.e.* de ne pas considérer les atomes menant à un échec fini), cette sémantique correspond en fait à $T^*_{\mathsf{Tr}^\omega_\Sigma, P} \downarrow \omega$ ($T^*_{\mathsf{Tr}^\omega_\Sigma, P} \downarrow 0$ étant la base de Herbrand complète) et Lloyd démontre dans [45] que $T^*_{\mathsf{Tr}^\omega_\Sigma, P} \downarrow \omega$ est égal au plus grand point fixe de l'opérateur $T^*_{\mathsf{Tr}^\omega_\Sigma, P}$ [87].

Exemple 25 : *Considérons le programme logique P suivant (qui code le test de parité d'un entier) :*

$$\begin{array}{l} e(zero) \\ o(s(zero)) \\ e(s(x)) \Leftarrow o(x) \\ o(s(x)) \Leftarrow e(x) \\ eo(x) \Leftarrow e(x), o(x) \end{array}$$

La dernière clause affirme que les succès du prédicat eo sont les entiers qui sont à la fois pairs et impairs. D'un point de vue de la sémantique du plus petit modèle de Herbrand, il n'existe aucun terme clos τ tel que l'atome $eo(\tau)$ appartiennent à cette sémantique. Cependant, d'un point de vue opérationnel, il existe pour le but $\Leftarrow eo(x)$ une SLD-dérivation infinie. Cela vient du fait que pour le but $\Leftarrow eo(s(s(s(s(\ldots)))))$, noté $\Leftarrow eo(s^\omega)$, on peut construire une telle SLD-dérivation.

La sémantique du plus petit modèle de Herbrand de ce programme est donnée par l'ensemble suivant :

$$\{e(zero), o(s(zero)), e(s(s(zero))), o(s(s(s(zero)))), \ldots\}$$

[86] Cette définition contredit quelque peu la définition donnée au chapitre 1. Ceci peut être vu comme une extension où le terme « SLD-dérivation » désigne une suite finie ou infinie telle que pour deux résolvantes G_i, G_{i+1} consécutives de la suite, $G_i \vdash_{SLD} G_{i+1}$ correspond à une étape de SLD-résolution.

[87] Si pour une algèbre A, le plus petit point fixe de $T_{\mathsf{A},P}$ est le plus petit A-modèle du programme P, il n'en va pas de même pour le plus grand point fixe qui n'est pas le plus grand A-modèle. Ceci vient du fait que l'ensemble de tous les A-atomes est toujours A-modèle d'un programme logique défini. Cependant, ce plus grand point fixe correspond au plus grand modèle du complété de Clark du programme P.

tandis que la sémantique réactive est donnée par l'ensemble :

$$\{e(zero), o(s(zero)), e(s(s(zero))), o(s(s(s(zero)))),$$
$$\ldots, e(s^\omega), o(s^\omega), eo(s^\omega)\}$$

5.4.2 Formalisation de l'analyse ensembliste de programmes réactifs

La formalisation de l'analyse ensembliste des programmes logiques réactifs est en fait similaire à celle des programmes logiques dans le sens du plus petit modèle de Herbrand.

Elle se base sur la fonction d'approximation App_α adaptée à la nouvelle sémantique, *i.e.* définie de manière identique mais non plus pour des ensembles de substitutions, mais pour des ensembles de valuations associant à chaque variable un arbre (fini ou infini). Cette fonction comme précédemment à pour but la perte de dépendance inter-variables au cours du calcul.

On peut alors, comme l'on fait Heintze et Jaffar, définir un opérateur ensembliste simulant un calcul avec perte de dépendance inter-variables approchant l'opérateur de point fixe défini pour les programmes réactifs,

$$\mathcal{T}^*_{P,\mathsf{Tr}^\omega_\Sigma}(I) = \left\{ \alpha(H) \;\middle|\; \begin{array}{l} H \Leftarrow B_1, \ldots, B_l \in P \\ \alpha\hat{\in} App_\alpha(\{\nu \mid \lfloor B_1 \rfloor_\nu^{\mathsf{Tr}^\omega_\Sigma} \in I \wedge \ldots \wedge \lfloor B_l \rfloor_\nu^{\mathsf{Tr}^\omega_\Sigma}\}) \end{array} \right\}$$

On peut alors affirmer de par la définition de App_α que le plus grand point fixe de cet opérateur est une sur-approximation du plus grand point fixe de $T^*_{P,\mathsf{Tr}^\omega_\Sigma}$, qui est la sémantique réactive du programme P.

Nous allons voir dans la section suivante comment à l'aide des contraintes ensemblistes, on peut donner à partir de cette définition formelle un mécanisme opérationnel permettant de calculer une représentation de cette plus grande solution.

5.4.3 Analyse ensembliste et contraintes

La démarche effective de l'analyse ensembliste de programmes logiques réactifs au moyen de contraintes ensemblistes est semblable à celle que nous avons présentée pour la sémantique du plus petit modèle de Herbrand. Elle se base sur l'extraction d'un système de contraintes de (la reformulation de) la définition de l'opérateur de $T^*_{P,\mathsf{Tr}^\omega_\Sigma}$.

Comme nous l'avions fait à la section 5.2.2, on peut reformuler la définition de cet opérateur de la manière suivante :

$$T^*_{P,\mathsf{Tr}^\omega_\Sigma}(I) = \bigcup_{c\in P} \left\{ H \;\middle|\; \bigwedge_i B_i \in I \wedge \bigwedge_{x\in Var_{\mathcal{X}}(c)} x \in HU^* \right\}$$

En notant SE^P_* l'expression

$$\bigcup_{c\in P} \left\{ H \;\middle|\; \bigwedge_i B_i \in I \wedge \bigwedge_{x\in Var_{\mathcal{X}}(c)} x \in HU^* \right\}$$

et en utilisant la fonction App_{SC} définie dans la section précédente, on peut montrer que

$$\mathcal{T}^*_{P,\mathsf{Tr}^\omega_\Sigma}(I) = App_{SC}(SE^P_*)$$

Ainsi, on en déduit que les solutions de l'équation ensembliste $I = App_{SC}(SE_*^P)$ sont exactement les points fixes de l'opérateur $\mathcal{T}^*_{P,\mathsf{Tr}^\omega_\Sigma}$. En particulier, la plus grande solution de cette équation est la sémantique ensembliste réactive du programme logique P.

Il est à noter que $App_{SC}(SE_*^P)$ est en fait une expression ensembliste (comportant des expressions d'appartenance dit « simples ») pouvant apparaître en membre droit des contraintes co-définies. De plus, on peut démontrer que de manière duale à l'analyse pour le plus petit modèle, que les plus grandes solutions de $I = App_{SC}(SE_*^P)$ et de $I \subseteq App_{SC}(SE_*^P)$ coïncident.

L'analyse ensembliste de programmes logiques réactifs se résume donc à l'extraction du programme d'un système de contraintes co-définies, dont on peut calculer la plus grande solution sur les ensembles d'arbres finis et infinis avec l'algorithme proposé au chapitre 4.

5.4.4 Autres applications

L'analyse ensembliste de la sémantique réactive d'un programme logique basée sur le calcul de la plus grande solution d'un système de contraintes possède d'autres applications.

L'une d'elles, assez immédiate, est la possibilité de donner une sous-approximation régulière de l'ensemble des échecs finis d'un programme logique. Se basant sur la possibilité de représenter divers formalismes opérationnels au travers des programmes logiques, les méthodes, proposées ici, peuvent être adaptées et utilisées comme une des principales composantes pour l'analyse statique d'autres paradigmes de programmation ou pour des systèmes réactifs décrits par un système de transitions.

Echecs finis L'analyse, ou plutôt l'approximation, que nous avons proposée à la section précédente, possède une autre application, celle de permettre de définir une sous-approximation régulière de l'ensemble des échecs finis d'un programme logique P.

Rappelons que l'ensemble des échecs finis d'un programme logique P que nous noterons FF_{P,T_Σ} et est égal au complémentaire de $T_{P,\mathsf{T}_\Sigma} \downarrow \omega$.

On peut alors en remarquant que l'opérateur $T_{P,\mathsf{Tr}^\omega_\Sigma}$ est monotone et que si E est un ensemble d'arbres finis, *i.e.* de termes clos, alors $T^*_{P,\mathsf{Tr}^\omega_\Sigma}(E) = T_{P,\mathsf{T}_\Sigma}(E)$, déduire que $T_{P,\mathsf{T}_\Sigma} \downarrow \omega \subseteq T^*_{P,\mathsf{Tr}^\omega_\Sigma} \downarrow \omega$. L'analyse ensembliste de la sémantique réactive du programme P donnant une sur-approximation de $T^*_{P,\mathsf{Tr}^\omega_\Sigma} \downarrow \omega$, le complémentaire de cet ensemble restreint aux arbres finis est donc une sous-approximation de FF_{P,T_Σ}.

CC-langages L'analyse ensembliste de la sémantique réactive de programmes logiques est également à la base d'une méthode d'analyse statique pour les *langages concurrents contraints* (CC-langages) [61]. Dans [56], Podelski, Charatonik et Müller utilisent cette analyse ensembliste pour détecter statiquement des erreurs de type « deadlock ou échec » pour des programmes Oz [66].

Vérification de systèmes réactifs Dans [19], Charatonik et Podelski proposent une méthode de vérification (partielle) basée sur l'analyse ensembliste pour des systèmes d'états infinis réactifs modélisés par des programmes logiques. Elle se base sur les approximations des plus grand et plus petit points fixes de l'opérateur de conséquences logiques immédiates et offre des procédures de semi-validation et semi-réfutation pour le système et un ensemble d'assertions, représenté comme des formules de la logique temporelle CTL.

Conclusion

Nous allons conclure cette thèse en rappelant les points essentiels qui y sont développés. Nous avons proposé une extension de la classe des contraintes ensemblistes définies, principalement en considérant ce que nous avons nommé « expressions d'appartenance », qui sont une description intensionnelle d'ensembles de la forme $\{x \mid \varphi(x)\}$, $\varphi(x)$ étant une formule positive du premier ordre (c-à-d ne comportant ni négation, ni implication, ni équivalence) et dont les atomes sont de la forme $t \in Sexp$, où $Sexp$ est une expression ensembliste. Nous proposons un algorithme répondant au problème de satisfiabilité d'un système de contraintes de cette extension et qui dans le cas, où le système est satisfiable donne une représentation de la plus petite solution sous la forme d'un automate d'arbres finis. Nous montrons également que cette extension préserve deux propriétés des contraintes ensemblistes définies : la complexité du problème de satisfiabilité (à savoir, *EXPTIME*-complet) et la propriété que les solutions forment un semi-treillis inférieur complet.

Cette extension de la classe définie nous a amené à proposer une extension de la classe co-définie, rendant ces deux classes duales par complémentation. Cette propriété fait que l'algorithme testant la satisfiabilité d'un système de contraintes définies généralisées peut être utilisé pour répondre à ce même problème, mais pour l'extension de la classe co-définie interprétée sur les ensembles d'arbres finis.

Nous nous sommes ensuite intéressés à cette extension de la classe des contraintes co-définies et avons proposé un algorithme pour tester la satisfiabilité d'un système de cette classe lorsque ces contraintes sont interprétées sur les ensembles d'arbres finis et infinis. La complexité de cet algorithme est montrée dans *EXPTIME*, alors que le problème est *EXPTIME*-dur. Enfin, nous montrons que lorsqu' une restriction est fait au problème de satisfiabilité sur les ensembles d'arbres finis, alors l'algorithme proposé est optimum pour ce problème, *i.e.* dans *EXPTIME*.

Les algorithmes donnés dans cette thèse ne peuvent, du fait de la taille des automates qu'ils manipulent, être implantés tels qu'ils ont été décrits. Cependant, une implantation (du cas « défini ») est en cours, se basant sur les principes de ces algorithmes et sur une représentation plus compacte des automates manipulés. Cependant, divers pistes s'offrent encore à nous, il conviendra de trouver celle donnant l'implantation la plus efficace.

Ce travail d'implantation constitue un des prolongements naturels de ce travail, mais des considérations plus théoriques mériteraient également d'être abordées.

Dans le domaine des contraintes ensemblistes, nous avons utilisé les expressions d'appartenance pour des classes de contraintes limitées. Ces expressions peuvent apparaître 'a gauche du symbole d'inclusion dans le cas des contraintes ensemblistes définies et à gauche pour les contraintes co-définies. Nous avons mentionné en outre le fait que les expressions d'appartenance permettent d'exprimer les opérateurs d'union, d'intersection, de projection et ceci restant toujours vraie même si on se restreint aux expressions d'appartenance simple, *i.e.* ne comportant

pas de quantification universelle.

La classe la plus « naturelle » à étudier est alors celle définie comme une inclusion de la forme $Sexp \subseteq Sexp$, où $Sexp$ est soit une composition fonctionnelle $f(X_1, \ldots, X_m)$, soit une expression d'appartenance $\{x \mid \varphi(x)\}$. Cette classe englobe la classe la plus générale, *i.e.* la classe *(PSC)* augmentée des symboles de projections[88]. Cependant, cette classe se révèle être indécidable, puisque que peut être encoder dans le problème de satisfiabilité d'un système de cette classe le problème de validité d'une formule monadique du premier ordre de la théorie des arbres finis, problème connu comme indécidable.

En effet, toute formule φ de la théorie monadique du premier ordre peut s'écrire comme une formule du premier ordre positive (sans négation, implication, ni équivalence) dont les atomes sont soit de la forme $t \in X$, soit $t \notin X$. D'un point de vue ensembliste, ce dernier type d'atomes peut se traduire en utilisant l'opérateur de complémentation par $t \in \complement X$. Cette expression $\complement X$ est équivalente d'un point de vue des contraintes ensemblistes à

$$\complement X \cup X = \top$$
$$\complement X \cap X = \bot$$

exprimable dans la classe à laquelle nous nous intéressons.

Ainsi, la contrainte $\{x \mid \varphi \wedge x \in a\} \subseteq b$ est satisfiable si et seulement si la formule φ de la théorie du premier ordre monadique des arbres finis n'est pas satisfiable.

Cependant, le problème reste ouvert lorsque lorsque les expressions d'appartenance sont « simples », cette classe généralisant également celle introduite dans [15].

Une autre voie de recherche, qui mériterait également d'être prospectée, est celle des caractérisations d'ensembles d'arbres finis et infinis à l'aide des termes de points fixes [53, 54]. Ces termes de points fixes Tpf peuvent être décrits par la grammaire suivante

$$Tpf ::= X \mid f(Tpf, \ldots, Tpf) \mid Tpf \cup Tpf \mid \mu X.Tpf \mid \nu X.Tpf$$

et sont interprétés sur les ensembles d'arbres finis et infinis.

Les opérateurs μ et ν sont respectivement les opérateurs de plus petit et de plus grand points fixes. Ainsi, les expressions $\mu X.a \cup f(X)$ et $\nu X.a \cup f(X)$ sont respectivement interprétées comme les plus grande et plus petite solutions de l'équation ensembliste $X = a \cup f(X)$, interprétée sur les ensembles d'arbres finis et infinis, *i.e.* resp. $\{f^n(a) \mid n \in \mathbb{N}\}$ et $\{f^n(a) \mid n \in \mathbb{N}\} \cup \{f(f(f(f(\ldots))))\}$.

Il est noté dans [54] qu'enrichir la définition des termes de points fixes pas des expressions de la forme $Tpf \cap Tpf$, où $\cap$ dénote un opérateur d'intersection, ne change par l'expressivité des termes de points fixes.

Une extension de ses termes de points fixes a été proposée dans [13] sous le nom de *mu-calcul de Horn*, se basant sur les programmes logiques uniformes de [28], qui correspondent à une classe de contraintes définies avec la composition fonctionnelle et ce que nous avons appelé expressions d'appartenance simples.

On peut donc se demander s'il est possible de considérer les expressions d'appartenance dans toute leur généralité, à savoir de définir des termes de points fixes de la manière suivante

$$Tpf ::= X \mid f(Tpf, \ldots, Tpf) \mid \{x \mid \varphi(x)\} \mid \mu X.Tpf \mid \nu X.Tpf$$

où φ est une formule monadique positive.

[88] Comme noté dans [15], la relation de non-inclusion est inutile en présence de symboles de projection. Ainsi, cette classe englobe la classe *(NPSC)*

Index

– R –

– S –

– T –

– V –

Bibliographie

[1] ACKERMANN, W. *Solvable Cases of the Decision Problem.* North-Holland, 1954.

[2] AIKEN, A., KOZEN, D., VARDI, M., AND WIMMERS, E. The Complexity of Set Constraints. In *Proceedings of Computer Science Logic* (1993), pp. 1–17.

[3] AIKEN, A., KOZEN, D., AND WIMMERS, E. Decidability of Systems of Set Constraints with Negative Constraints. Techn. report 93-1362, Computer Science Departement, Cornell University, 1993.

[4] AIKEN, A., KOZEN, D., AND WIMMERS, E. Decidability of Systems of Set Constraints with Negative Constraints. *Information and Computation 1*, 122 (oct 1995), 30–44.

[5] AIKEN, A., AND WIMMERS, E. Solving Systems of Set Constraints. In *Proceedings of the 7^{th} IEEE Symposium on Logic in Computer Science* (1992), pp. 329–340.

[6] AIKEN, A., AND WIMMERS, E. L. Type Inclusion Constraints and Type Inference. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture* (Jun 1993), ACM Press, pp. 31–41.

[7] BACHMAIR, L., GANZINGER, H., AND WALDMANN, U. Set Constraints are the Monadic Class. In *Proceedings of the 8^{th}IEEE Symposium on Logic in Computer Science* (1993), pp. 75 – 83.

[8] BACHMAIR, L., GANZINGER, H., AND WALDMANN, U. Superposition with Simplification as a Decision Procedure for the Monadic Class with Equality. Technical Report MPI-I-93-204, Max-Planck-Institut für Informatik, feb 1993.

[9] BANCILHON, F., MAIER, D., SAGIV, Y., AND ULLMAN, J. Magic Sets and other Strange Ways to Implement Logic Programs. In *Proceedings of the 5^{th} ACM Symposium on Principles of Database Systems* (1986).

[10] BOGAERT, B., AND TISON, S. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In *Proceedings of the 1992 Symposium on Theoretical Aspects of Computer Science* (1992), LNCS 577, pp. 161–171.

[11] CHANDRA, A., KOZEN, D., AND STOCKMEYER, L. Alternation. *Journal of ACM 28* (1981), 114–133.

[12] CHARATONIK, W. *Set Constraints in Some Equational Theories.* PhD thesis, University of Wroclaw, 1994.

[13] CHARATONIK, W., MCALLESTER, D., NIWIŃSKI, D., PODELSKI, A., AND WALUKIEWICZ, I. The Horn Mu-Calculus. In *Proceedings of the 13^{th} IEEE Symposium on Logic in Computer Science* (1998), pp. 58–69.

[14] CHARATONIK, W., AND PACHOLSKI, L. Negative Set Constraints with Equality. In *Proceedings of the 9^{th} IEEE Symposium on Logic in Computer Science* (1994), IEEE, Ed., pp. 128–136.

[15] CHARATONIK, W., AND PACHOLSKI, L. Set Constraints with Projections are in NEXPTIME. In *Proceedings of the 35th Symposium on Foundations of Computer Science* (1994), pp. 642–653.

[16] CHARATONIK, W., AND PODELSKI, A. Set Constraints with Intersection. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science* (1997).

[17] CHARATONIK, W., AND PODELSKI, A. Solving Set Constraints for Greatest Models. Tech. Rep. MPI-I-97-2-004, Max-Planck-Institut für Informatik, 1997.

[18] CHARATONIK, W., AND PODELSKI, A. Co-definite Set Constraints. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications* (1998), LNCS.

[19] CHARATONIK, W., AND PODELSKI, A. Set-based Analysis of Reactive Infinite-state Systems. In *Proceedings of the 1st International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (mar 1998), vol. LNCS 1384, pp. 358–375.

[20] CHENG, A., AND KOZEN, D. A Complete Gentzen-style Axiomatization for Set Constraints. In *Proceedings of the 23th International Colloquium on Automata, Languages, and Programming* (jul 1996), LNCS 1099, pp. 134–145.

[21] COMON, H., DAUCHET, M., GILLERON, R., LUGIEZ, D., TISON, S., AND TOMMASI, M. Tree automata : Techniques and applications, 1997.

[22] COURCELLE, B. Fundamental Properties of Infinite Trees. *Theoretical Computer Science 25* (1983), 95–169.

[23] DERANSART, P., AND MALUSZYNSKI, J. *A Grammatical View of logic Programming.* MIT-Press, 1993.

[24] DEVIENNE, P., TALBOT, J.-M., AND TISON, S. Set-based Analysis for Logic Programming and Tree Automata. In *Proceedings of the 4th International Static Analysis Symposium* (sep 1997), P. V. Hentenryck, Ed., LNCS 1302, Springer, pp. 127–140.

[25] DEVIENNE, P., TALBOT, J.-M., AND TISON, S. Solving Classes of Set Constraints with Tree Automata. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming* (oct 1997), G. Smolka, Ed., LNCS 1330, pp. 62–76.

[26] DEVIENNE, P., TALBOT, J.-M., AND TISON, S. Co-definite Set Constraints with Membership Expressions. In *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming* (jun 1998), J. Jaffar, Ed., MIT-Press, pp. 25–39.

[27] FILÉ, G. Tree Automata and Logics Programs. In *Proceedings of the 2nd Symposium on Theoretical Aspects of Computer Science* (1985), LNCS 182, Spinger, pp. 119–130.

[28] FRÜHWIRTH, T., SHAPIRO, E., VARDI, M., AND YARDENI, E. Logic Programs as Types for Logic Programs. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science* (jun 1991), pp. 300–309.

[29] GALLAGHER, J., BOULANGER, D., AND SAGLAM, H. Practical Model-Based Static Analysis for Definite Logic Programs. In *Proceedings of the International Logic Programming Symposium* (dec 1995), MIT-Press, pp. 351–368.

[30] GALLAGHER, J., AND DE WAAL, D. Fast and Precise Regular Approximations of Logic Programs. In *Proceedings of the 11th International Conference on Logic Programming* (1994), MIT-Presse, pp. 599–613.

[31] GÉCSEG, F., AND STEINBY, M. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[32] GILLERON, R., TISON, S., AND TOMMASI, M. Solving System of Set Constraints using Tree Automata. In *Proceedings of the* 10^{th} *Symposium on Theoretical Aspects of Computer Science* (feb 1993), LNCS 665, pp. 505–514.

[33] GILLERON, R., TISON, S., AND TOMMASI, M. Solving Systems of Set Constraints with Negated Subset Relationships. In *Proceedings of the* 34^{th} *Symposium on Foundations of Computer Science* (1993), pp. 372–380.

[34] GILLERON, R., TISON, S., AND TOMMASI, M. Some New Decidability Results on Positive and Negative Set Constraints. In *Proceedings of the* 1^{st} *International Conference on Constraints in Computational Logics* (1994), LNCS 845, pp. 336–351.

[35] GIVAN, R., KOZEN, D., MCALLESTER, D., AND WITTY, C. Tarskian Set constraints. In *Proceedings of the* 11^{th} *IEEE Symposium on Logic in Computer Science* (jul 1996), pp. 138–147.

[36] HEINTZE, N. Practical Aspects of Set Based Analysis. In *Proceedings of the Joint International Conference and Symposium on Logic Programming* (nov 1992), MIT-Press.

[37] HEINTZE, N. *Set Based Program Analysis.* PhD thesis, Carnegie Mellon University, sep 1992.

[38] HEINTZE, N. Set-based Analysis of ML Programs. In *Lisp and Functional Programming.* ACM, 1994, pp. 306–317.

[39] HEINTZE, N., AND JAFFAR, J. A Decision Procedure for a Class of Herbrand Set Constraints. In *Proceedings of the* 5^{th} *IEEE Symposium on Logic in Computer Science* (jun 1990), pp. 42–51.

[40] HEINTZE, N., AND JAFFAR, J. A Finite Presentation Theorem for Approximating Logic Programs. In *Proceedings of the* 17^{th} *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (jan 1990), pp. 197–209.

[41] JAFFAR, J., AND MAHER, M. Constraint logic programming : A survey. *Journal of Logic Programming 19/20* (1994), 503–581.

[42] KOZEN, D. Logical Aspects of Set Constraints. In *Proceedings of the* 1993 *Conference on Computer Science Logic* (1993), LNCS 832, pp. 175–188.

[43] KOZEN, D. Rational Spaces and Set Constraints. *Theoretical Computer Science 167* (1996), 73–94.

[44] LECOUTRE, C. *Interprétation Abstraite en Programmation Logique avec Contraintes.* PhD thesis, Université des Sciences et Technologies de Lille, 1994.

[45] LLOYD, J. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[46] LÖWENHEIM, L. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen 76* (1915), 228–251.

[47] MACALLESTER, D., AND HEINTZE, N. On the Complexity of Set-Based Analysis. In *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming* (1997), vol. 32(8) of *SIGPLAN Notices*, ACM Press, pp. 150–163.

[48] MAHER, M. Complete Axiomatizations of the Algebras of Finite, Rationnal and Infinite Trees. In *Proceedings of the* 3^{rd} *IEEE Symposium on Logic in Computer Science* (1988), pp. 348–357.

[49] MAL'CEV, A. On the Elementary Theories of Locally Free Universal Algebras. *Soveit Math. Doklady* (1961), 768–771.

[50] MÜLLER, M., NIEHREN, J., AND PODELSKI, A. Inclusion Constraints over Non-Empty Sets of Trees. In *Proceedings of 7th International Joint Conference CAAP/FASE - (TAPSOFT'97)* (apr 1997), LNCS 1214, pp. 345–356.

[51] MISHRA, P. Toward a Theory of Types in PROLOG. In *Proceedings of the 1st International Logic Programming Symposium* (jul 1984), pp. 289–298.

[52] MÜLLER, M. *Set-based Failure Diagnosis for Concurrent Constraint Programming.* PhD thesis, Universität des Saarlandes, Technische Fakultät, D–66041 Saarbrücken, jul 1998.

[53] NIWIŃSKI, D. On Fixed-point Clones. In *Proceedings of the 13th International Colloquium on Automata, Languages, and Programming* (1986), LNCS 226, pp. 464–473.

[54] NIWIŃSKI, D. Fixed Point Characterization of Infinite Behaviour of Finite State System. *Theoretical Computer Science 189*, 1–2 (1997), 1–69.

[55] PACHOLSKI, L., AND PODELSKI, A. Set Constraints : A Pearl in Research on Constraints. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming* (oct 1997), G. Smolka, Ed., LNCS 1330, pp. 549–561. Tutorial.

[56] PODELSKI, A., CHARATONIK, W., AND MÜLLER, M. Set-based Error Diagnosis of Concurrent Constraint Programs. Tech. rep., Programming Systems Lab, Universität des Saarlandes, 1998.

[57] POST, E. L. Recursively Enumerable Sets of Positive Integers and their Decision Problems. *Bulletion of the American Mathematical Society 50* (1944), 284–316.

[58] RABIN, M. Decidability of Second-order Theories and Automata on Infinite Trees. In *Transactions of American Mathematical Society* (1969), vol. 141, pp. 1–35.

[59] RABIN, M. *Mathematical Logic and Foundations of Set Theory.* Y. Bar-Hillel, 1970, ch. Weakly Definable Relations and Special Automata, pp. 1–23.

[60] ROBINSON, J. A Machine-oriented Logic Based on the Resolution Principle. *Journal of ACM 12*, 1 (jan 1965).

[61] SARASWAT, V. *Concurrent Constraint Programming.* MIT-Press, 1993.

[62] SEIDL, H. Haskell Overloading is DEXPTIME-complete. *Information Processing Letter 52* (1994), 57–60.

[63] SEYNHAEVE, F. Contraintes ensemblistes. Rapport de DEA, LIFL, Université des Sciences et Technologies de Lille, jul 1994.

[64] SEYNHAEVE, F. Set constraints and automata. 1996 CCL Workshop, 1996.

[65] SEYNHAEVE, F., TOMMASI, M., AND TREINEN, R. Grid Structures and Undecidable Constraint Theories. In *Proceedings of 7th International Joint Conference CAAP/FASE - (TAPSOFT'97)* (apr 1997), LNCS 1214, pp. 357–368.

[66] SMOLKA, G. The Oz Programming Model. In *Computer Science Today*, J. van Leeuwen, Ed., LNCS 1000. Springer-Verlag, Berlin, 1995, pp. 324–343.

[67] STEFANSSON, K. Systems of Set Constraints with Negative Constraints are NEXPTIME-Complete. In *Proceedings of the 9th IEEE Symposium on Logic in Computer Science* (1994).

[68] TAITSLIN, M. Some Further Examples of Undecidable Theories. *Algebra and Logic*, 7 (1968), 127–129.

[69] TALBOT, J.-M., DEVIENNE, P., AND TISON, S. Generalized Definite Set Constraints. *CONSTRAINTS - An International Journal* (1999). To appear.

[70] TARSKI, A. A Lattice Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics 5* (1955), 285–310.

[71] THOMAS, W. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science* (1990), vol. B, Elsevier, pp. 133–191.

[72] TOMMASI, M. *Automates et Contraintes Ensemblistes.* PhD thesis, Université des Sciences et Technologies de Lille, 1994.

[73] URIBE, T. E. Sorted Unification and the Solution of Semi-linear Membership Cconstraints. Master's thesis, University of Illinois, 1992.

[74] URIBE, T. E. Sorted Unification using Set Constraints. In *Proceedings of the* 11^{th} *International Conference on Automated Deduction* (1992), LNAI 607, pp. 163–177.

[75] VAN EMDEN, M., AND KOWALSKI, R. The Semantics of Predicate Logic as a Programming Language. *Journal of ACM 23*, 4 (oct 1976), 733–742.