Université d'Orléans

Habilitation à diriger les recherches
Résumé des travaux

Spécialité : Informatique

Langages synchronisés d'arbres et applications

Pierre Réty

LIFO, Université d'Orléans
rety@lifo.univ-orleans.fr

Soutenue le 30 juin 2001 devant le jury composé de

Président :	Christel Vrain	Professeur	(LIFO - Orléans)
Rapporteurs :	Bernhard Gramlich	Professeur	(TU - Vienne)
	Denis Lugiez	Professeur	(LIM - Marseille)
	Sophie Tison	Professeur	(LIFL - Lille)
Examinateurs :	Françoise Bellegarde	Professeur	(LIFC - Besançon)
	Helmut Seidl	Professeur	(Univ. Trèves)

Table des matières

1 Introduction **3**

2 Les langages synchronisés de n-uplets d'arbres **5**
2.1 Les grammaires synchronisées (TTSG) 5
2.1.1 Les TTSG à trans-classes bornées 9
2.1.2 Les TTSG à inclinaison bornée 11
2.2 Les systèmes de contraintes (CS) 12
2.2.1 Propriétés 15
2.2.2 Les relations régulières (RR) 16
2.2.3 Les relations faiblement régulières (WRR) 17
2.3 Travaux apparentés 20

3 Applications **23**
3.1 Les algèbres de processus 23
3.1.1 PA 23
3.1.2 BCPC 25
3.1.3 Model-checking 26
3.1.4 Conclusion et perspectives 27
3.2 La validation des programmes logiques 27
3.3 Les relations de type réécriture 28
3.4 La théorie de la réécriture en une étape 28

4 L'unification équationnelle **31**
4.1 En exprimant les solutions par un langage d'arbres 31
4.1.1 Avec les TTSG à trans-classes bornées 33
4.1.2 Avec les WRR 35
4.1.3 Résoudre les diséquations avec les TTSG 37
4.1.4 Perspectives 39
4.2 A l'aide d'un langage régulier de descendants 40
4.3 Résultats de décidabilité apparentés 43
4.4 Par surréduction dirigée 44

5 Perspectives **47**

Remerciements

Je tiens à remercier sincèrement tous les membres du jury pour avoir bien voulu s'intéresser à mes travaux. Je voudrais aussi remercier chaleureusement les chercheurs avec qui j'ai travaillé, ainsi que mes collègues du LIFO pour leur disponibilité à mes nombreuses questions scientifiques et techniques.

Chapitre 1

Introduction

Ce document est un résumé des résultats de recherche que j'ai obtenus, seul ou le plus souvent en collaboration avec d'autres chercheurs. Les articles complets, hormis les plus anciens, sont disponibles sur ma page web [1]. Nous présentons une nouvelle classe de langages d'arbres, les langages synchronisés, et leurs applications à la vérification, à la réécriture, et à la démonstration automatique (unification équationnelle). Des résultats sur l'unification équationnelle obtenus par d'autres moyens (langages réguliers d'arbres, surréduction [2]) sont aussi mentionnés.

Deux thèses ont été encadrées de manière effective : la moitié de celle de Jacques Chabin (soutenue en janvier 1994) et la totalité de celle de Sébastien Limet (soutenue en décembre 1996). Une thèse est en cours : celle de Benoit Lecland, et une autre va démarrer en septembre 2001 : celle de Julie Vuotto.

Sébastien Limet et moi-même avons introduit les langages synchronisés de n-uplets d'arbres à l'aide des grammaires synchronisées, pour résoudre un problème d'unification équationnelle (CAAP'97 [64], version complète dans la revue électronique DMTCS [65]). Par la suite nous les avons appliqués à la résolution de diséquations (RTA'98 [67]), puis à la théorie de la réécriture en une étape (RTA'99 [68], version complète dans [66]). Avec l'aide d'Helmut Seidl, nous avons redéfini avantageusement les langages synchronisés par un autre formalisme, les systèmes de contraintes, et avec l'aide de Valérie Gouranton, nous les avons appliqués à la vérification des processus concurrents et aux relations de type réécriture (FoSSaCs'2001 [43], version complète dans [42]). Puis nous avons défini les relations faiblement régulières, sous-classe des langages synchronisés stable par intersection, et les avons appliquées à la théorie de la réécriture en une étape, et à l'unification équationnelle (RTA'2001 [70], version complète dans [69]).

Dans certains de ces articles, les langages synchronisés ont été abordés d'un point de vue utilitaire, pour résoudre un problème précis, et leur présentation est parfois mélangée avec le problème à résoudre. Or de notre point de vue, les langages synchro-

1. http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety/
2. Narrowing en anglais.

nisés méritent d'être considérés en tant que tels, et pas seulement comme une technique de preuve utilisée pour obtenir tel ou tel résultat. C'est pourquoi, dans ce document, l'étude des langages synchronisés et de leurs propriétés est séparée des applications.

L'unification équationnelle est traitée à part. Il s'agit plus précisément d'unification modulo un système de réécriture, que nous abordons par plusieurs méthodes : en utilisant les langages synchronisés, grâce à un calcul de descendants (exprimés par un langage régulier) que j'ai proposé (LPAR'99 [94]) et implanté en Java, et enfin par surréduction dirigée. Les travaux sur la surréduction ont été effectués avec Jacques Chabin (RTA'91 [19], et améliorés dans [20]), puis Siva Anantharaman pour une version dans le cadre de la complétion sans échec UKB [18], puis Sébastien Limet pour des comparaisons précises avec les autres stratégies de surréduction [61] et une version dans le cadre de la réécriture conditionnelle [60] implantée en C (AMAST'96 [62]).

Notions de base et notations

Pour tirer profit de ce document, le lecteur doit connaître les bases de la logique équationnelle et de la réécriture : égalité modulo R, ensembles complets de R-unificateurs, confluence, terminaison, surréduction (voir [33]), et des langages d'arbres : automates finis, grammaires régulières (voir [37, 25]).

Précisons seulement quelques notations. Etant donné une signature Σ, T_Σ est l'ensemble des termes clos (sans variables), et si on fait la distinction entre constructeurs et fonctions définies, C est l'ensemble des constructeurs et T_C est l'ensemble des *termes-constructeur* clos, c-à-d des termes clos qui ne contiennent que des constructeurs. Pour $t \in T_\Sigma$, $Pos(t)$ est l'ensemble des positions (occurrences) de t, et pour $p \in Pos(t)$, $t(p)$ est le symbole apparaissant dans t à la position p, alors que $t|_p$ est le sous-terme de t à la position p.

Un système de réécriture est *basé sur les constructeurs*, ou encore respecte une *discipline de constructeurs*, si pour toute règle $l \to r$, l possède une fonction définie à la racine, et des constructeurs ou des variables partout ailleurs. La surréduction est notée par le symbole $\rightsquigarrow$.

Les champs d'un n-uplet sont appelés *composants*. Etant donné un langage S de l-uplets, et T de n-uplets, et pour $i \in \{1,\dots,l\}$, $j \in \{1,\dots,n\}$, la *i,j-jointure* de S et T est un langage de $l+n-1$-uplets, noté $S \bowtie_{i,j} T$, et défini par :

$$S \bowtie_{i,j} T = \{(s_1,\dots,s_l,t_1,\dots,t_{j-1},t_{j+1},\dots,t_n) \mid (s_1,\dots,s_l) \in S \wedge (t_1,\dots,t_n) \in T \wedge s_i = t_j\}$$

$S \bowtie T$ signifie $S \bowtie_{l,1} T$.

Pour $i_1,\dots,i_k \in \{1,\dots,l\}$, la *projection* de S sur les composants $i_1,\dots,i_k$ est le langage de k-uplets, noté $\Pi_{i_1,\dots,i_k}(S)$, défini par :

$$\Pi_{i_1,\dots,i_k}(S) = \{(s_{i_1},\dots,s_{i_k}) \mid \forall j \neq i_1,\dots,i_k, \exists s_j, (s_1,\dots,s_l) \in S\}$$

Chapitre 2

Les langages synchronisés de n-uplets d'arbres

Dans de nombreux domaines de l'informatique théorique, il est souvent nécessaire de manipuler des ensembles infinis de termes. Les automates ou les grammaires d'arbres peuvent alors rendre de grands services, sous réserve que ces automates/grammaires possèdent de bonnes propriétés, comme la stabilité par les opérations booléennes, la décidabilité de l'appartenance ou du vide, etc... Les automates d'arbres ont déjà été utilisés en démonstration automatique, réécriture, vérification de propriétés de sûreté, construction de compilateurs, analyse statique de programmes.

Sous l'appellation "langages synchronisés de n-uplets d'arbres", nous regroupons trois formalismes : les TTSG, les CS, et les grammaires de J.C. Raoult.

2.1 Les grammaires synchronisées (TTSG)

Les *TTSG* (Tree Tuple Synchronized Grammar) ont été introduites par Sébastien Limet et moi-même [64, 65]. Ce sont des grammaires régulières d'arbres dans lesquelles certaines productions sont synchronisées, c-à-d doivent être appliquées en même temps. Ainsi les différentes branches d'un arbre ne sont plus forcément indépendantes. Dans ce formalisme, les non-terminaux représentent des arbres (termes clos), et on obtient des n-uplets quand l'axiome est un n-uplet de non-terminaux.

Exemple 2.1 La TTSG suivante contient quatre paquets de productions synchronisées :

$$\left\{\begin{array}{l} X \Rightarrow f(X,X') \\ Y \Rightarrow f(Y,Y') \end{array}\right. \qquad \left\{\begin{array}{l} X \Rightarrow b \\ Y \Rightarrow b \end{array}\right. \qquad \left\{\begin{array}{l} X' \Rightarrow f(X,X') \\ Y' \Rightarrow f(Y,Y') \end{array}\right. \qquad \left\{\begin{array}{l} X' \Rightarrow b \\ Y' \Rightarrow b \end{array}\right.$$

Le premier paquet veut dire que si X est dérivé en $f(X,X')$, alors Y doit être dérivé en $f(Y,Y')$ en même temps. Donc si X apparaît dans un arbre mais pas Y, on ne peut pas appliquer ce paquet. Avec le couple (X,Y) pour axiome, une dérivation possible est :

$$(X,Y) \Rightarrow (f(X,X'),f(Y,Y')) \Rightarrow (f(X,f(X,X')),f(Y,f(Y,Y')))$$

Mais il y a une ambiguïté : quel X doit être synchronisé (dérivé en même temps) avec quel Y ? Pour lever cet ambiguïté, un contrôle (un entier) est attaché à chaque non-terminal le long de la dérivation, et est incrémenté à une nouvelle valeur chaque fois qu'un paquet de productions synchronisées est appliqué. La règle est : seuls des non-terminaux ayant le même contrôle peuvent être dérivés en même temps. Ainsi la dérivation précédente s'écrit (pour la lisibilité, le contrôle apparaît en indice) :

$$(X_0,Y_0) \Rightarrow (f(X_1,X_1'),f(Y_1,Y_1')) \Rightarrow (f(X_1,f(X_2,X_2')),f(Y_1,f(Y_2,Y_2')))$$

On voit maintenant que X et Y peuvent être dérivés ensemble seulement s'ils apparaissent à des positions identiques dans les deux arbres. Par conséquent cette grammaire génère les couples de termes identiques. $\diamond$

Pour obtenir quelque chose d'intéressant, le contrôle est indispensable. Malheureusement, le contrôle simple utilisé dans l'exemple précédent n'est pas suffisant quand on fait la jointure naturelle ou l'intersection de langages synchronisés : on obtient une TTSG dite de rang 2, plus compliquée dans son fonctionnement, et dont le contrôle est un couple d'entiers.

Exemple 2.2 Soit G_1 la grammaire définie par l'axiome (X,Y) et les productions :

$$G_1 : \qquad X' \Rightarrow 0,\ Y' \Rightarrow 0,\ X' \Rightarrow s(X),\ Y' \Rightarrow s(Y''),\ Y'' \Rightarrow s(Y),\ \begin{cases} X \Rightarrow X' \\ Y \Rightarrow Y' \end{cases}$$

et soit G_2 définie par l'axiome (Z,T) et les productions :

$$G_2 : \qquad Z' \Rightarrow 0,\ T' \Rightarrow 0,\ Z' \Rightarrow s(Z),\ T' \Rightarrow s(T''),\ T'' \Rightarrow s(T),\ \begin{cases} Z \Rightarrow Z' \\ T \Rightarrow T' \end{cases}$$

G_1 et G_2 génèrent le même langage, à savoir les couples de la forme $(s^n(0),s^{2n}(0))$. La jointure du second composant (champ) de G_1 avec le premier composant de G_2 est le langage de triplets :

$$L_3 = \{(s^n(0),s^{2n}(0),s^{4n}(0))\}$$

La question est : comment construire à partir des deux TTSG précédentes, une nouvelle TTSG G_3 qui reconnaît le langage L_3 ? L'idée est que l'on va générer le premier composant de L_3 avec les productions de G_1 et le troisième composant de L_3 par les productions de G_2. Par conséquent G_3 contiendra les non-terminaux et les productions de G_1 et de G_2. Enfin le second composant de L_3 sera généré par l'intersection du deuxième composant de G_1 avec le premier de G_2. Cette intersection sera calculée dans le même esprit que les intersections de grammaires régulières d'arbres, c'est à dire en calculant le produit cartésien des grammaires. Voici plus précisément comment on procède. Tout d'abord, remarquons que Y,Y',Y'' sont les seuls non-terminaux qui peuvent apparaître dans le deuxième composant d'un calcul de G_1 (il suffit de remarquer que le deuxième composant de l'axiome est Y que Y ne peut se dériver qu'en Y', Y' en $s(Y'')$ et Y'' en $s(Y)$). De même, Z et Z' sont les seuls non-terminaux qui peuvent apparaître dans le premier composant d'un calcul de G_2. Par conséquent, l'ensemble des non-terminaux utiles pour l'intersection est le produit cartésien de ces deux ensembles c'est à dire :

$$\{YZ, Y'Z, Y''Z, YZ', Y'Z', Y''Z'\}$$

Les productions *libres* (non synchronisées) seront donc :

$$Y'Z' \Rightarrow s(Y''Z),\ Y'Z' \Rightarrow 0,\ Y''Z' \Rightarrow s(YZ)$$

Les paquets de productions synchronisées seront construits de telle manière que lorsqu'une synchronisation était possible dans une des grammaires initiales, elle le sera dans la grammaire G_3. Plus précisément, pour chaque paquet de productions de G_1 (resp. G_2) qui contient un des non-terminaux Y, Y' ou Y'' (resp. Z ou Z'), on crée un nouveau paquet dans G_3. Voici les paquets que l'on obtient :

- $\{X \Rightarrow X', YZ \Rightarrow Y'Z\}$, $\{X \Rightarrow X', YZ' \Rightarrow Y'Z'\}$ provenant des paquets de G_1 et
- $\{YZ \Rightarrow YZ', T \Rightarrow T'\}$, $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}$, $\{Y''Z \Rightarrow Y''Z', T \Rightarrow T'\}$ provenant des paquets de G_2.

L'axiome est (X, YZ, T).

Malheureusement, la grammaire G_3 n'est pas celle que l'on recherche car elle reconnaît le langage vide. En effet, si on utilise le paquet $\{X \Rightarrow X', YZ \Rightarrow Y'Z\}$, en partant d'un contrôle égal à 0 l'axiome est dérivé en

$$(\,X'_1, Y'Z_1, T_0)$$

Maintenant le paquet $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}$ ne peut plus être appliqué à cause des numéros de contrôle et plus aucune autre production ne peut être appliquée sur les deux derniers composants. On peut remarquer qu'on arrive à une situation similaire si on commence par appliquer le paquet $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}$. Le langage reconnu par G_3 est donc bien vide. Le problème vient du fait que l'on mélange, dans G_3, le contrôle des productions provenant de G_1 avec celui des productions provenant de G_2. Afin d'y remédier, nous allons introduire des couples d'entiers dans le contrôle, ce qui nous permettra de différencier les deux niveaux de contrôle. Comme cela, lorsqu'on appliquera un paquet provenant de G_1 on utilisera le premier niveau de contrôle et lorsque le paquet proviendra de G_2, on utilisera le second niveau. Concrètement, voici les modifications qui sont faites dans G_3. Tout d'abord le contrôle initial de l'axiome est :

$$(\,X_{(0,\perp)}, YZ_{(0,0)}, T_{(\perp,0)}\,)$$

$\perp$ signifiant que le non-terminal n'est pas concerné par ce niveau de contrôle. Ensuite les paquets de productions synchronisées de G_3 provenant de G_1 incrémenteront le premier composant du contrôle alors que ceux provenant de G_2 incrémenteront le deuxième. En notant en indice des paquets le numéro (appelé aussi rang du paquet) du composant du contrôle qui sera incrémenté, G_3 devient :

- $\{X \Rightarrow X', YZ \Rightarrow Y'Z\}_1$, $\{X \Rightarrow X', YZ' \Rightarrow Y'Z'\}_1$

– $\{YZ \Rightarrow YZ', T \Rightarrow T'\}_2$, $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2$, $\{Y''Z \Rightarrow Y''Z', T \Rightarrow T'\}_2$

Une dérivation possible pour notre nouvelle grammaire G_3 est :

$$(\, X_{(0,\perp)}, YZ_{(0,0)}, T_{(\perp,0)}\,) \Rightarrow (\, X'_{(1,\perp)}, Y'Z_{(1,0)}, T_{(\perp,0)}\,)$$

Maintenant $\{Y'Z \Rightarrow Y'Z', T \Rightarrow T'\}_2$ est applicable et on obtient :

$$\begin{aligned}
&(\, X'_{(1,\perp)}, Y'Z'_{(1,1)}, T'_{(\perp,1)}\,)\\
&\quad \Rightarrow^*_{[prods-libres.]} (\, \boldsymbol{s}(X_{(1,\perp)}), \boldsymbol{s}(Y''Z_{(1,1)}), \boldsymbol{s}(\boldsymbol{s}(T_{(\perp,1)}))\,)\\
&\quad \Rightarrow (\, \boldsymbol{s}(X_{(1,\perp)}), \boldsymbol{s}(Y''Z'_{(1,2)}), \boldsymbol{s}(\boldsymbol{s}(T'_{(\perp,2)}))\,)\\
&\quad \Rightarrow^*_{[prods-libres.]} (\, \boldsymbol{s}(X_{(1,\perp)}), \boldsymbol{s}(\boldsymbol{s}(YZ_{(1,2)})), \boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(T_{(\perp,2)}))))\,)\\
&\quad \Rightarrow (\, \boldsymbol{s}(X'_{(2,\perp)}), \boldsymbol{s}(\boldsymbol{s}(Y'Z_{(2,2)})), \boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(T_{(\perp,2)}))))\,)\\
&\quad \Rightarrow (\, \boldsymbol{s}(X'_{(2,\perp)}), \boldsymbol{s}(\boldsymbol{s}(Y'Z'_{(2,3)})), \boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(T'_{(\perp,3)}))))\,)\\
&\quad \Rightarrow^*_{[prods-libres.]} (\, \boldsymbol{s}(\boldsymbol{0}), \boldsymbol{s}(\boldsymbol{s}(\boldsymbol{0})), \boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(\boldsymbol{s}(\boldsymbol{0}))))\,)
\end{aligned}$$

$\diamond$

On voit que la jointure de G_3 avec une autre TTSG nécessitera un contrôle sous forme de triplets, c'est à dire une TTSG de rang 3. D'où la définition générale d'une TTSG de rang quelconque. Pour faciliter l'algorithme de jointure, on suppose que les productions libres sont en forme Greibach, et que les productions synchronisées sont vides.

Définition 2.3 *Soit NT un ensemble de non-terminaux. Une* production *est une règle de la forme $X \Rightarrow Y$ ou $X \Rightarrow c(Y^1, \ldots, Y^n)$, avec $X, Y^1, \ldots, Y^n \in NT$ et $c \in \Sigma$. Un* paquet de productions *est un ensemble de productions associé à un entier positif (appelé* rang du paquet*). Il se note $\{X^1 \Rightarrow t^1, \ldots, X^n \Rightarrow t^n\}_k$.*

- *Lorsque $k = 0$ le paquet est un singleton de la forme $\{X \Rightarrow c(Y^1, \ldots, Y^n)\}_0$. Cette production est dite* libre, *elle s'écrit plus simplement $X \Rightarrow c(Y^1, \ldots, Y^n)$.*
- *Lorsque $k > 0$ le paquet est de la forme $\{X^1 \Rightarrow Y^1, \ldots, X^n \Rightarrow Y^n\}_k$. Les productions du paquet sont dites* synchronisées.

Définition 2.4 *Une* TTSG *est un quintuplet (Sz, Σ, NT, PP, TI) où*

- *Sz est un entier positif, appelé* rang de la TTSG, *qui définit la taille des n-uplets de contrôle,*
- *Σ est un ensemble de symboles avec arités (terminaux),*
- *NT est un ensemble fini de non-terminaux,*
- *PP est un ensemble fini de paquets de productions de rang inférieur ou égal à Sz,*
- *TI est l'axiome de la TTSG. C'est un n-uplet $(X^1_{ct_1}, \ldots, X^n_{ct_n})$ où $X^i \in NT$ et chaque ct_i est un Sz-uplet de contrôle qui contient uniquement des 0 et des $\perp$.*

Une production libre $X \Rightarrow c(Y^1, \ldots, Y^n)$ peut être appliquée dès que X apparaît. Les non-terminaux $Y_1, \ldots, Y_n$ gardent alors le même n-uplet de contrôle que X. D'autre part le paquet de productions synchronisées $\{X^1 \Rightarrow Y^1, \ldots, X^n \Rightarrow Y^n\}_k$ peut être appliqué si $X^1, \ldots X^n$ sont présents et que les k^{eme} composants des contrôles de $X^1, \ldots X^n$ sont identiques (et ne valent pas $\perp$). Les X^i sont alors remplacés par les Y^i et on incrémente les k^{eme} composants des contrôles à une nouvelle valeur.

La définition ci-dessous exprime une condition un peu plus forte que la régularité, qui permet la stabilité par jointure. Dans le cas des TTSG de rang $Sz > 1$, une hypothèse supplémentaire doit être ajoutée à cette définition. Cette hypothèse étant très technique, nous ne l'avons pas donnée ici (voir [65, 63]).

Définition 2.5 *Le i-ème composant d'un langage L de n-uplets d'arbres est dit* libre, *ou encore* sans synchronisations internes, *s'il existe une TTSG G qui génère L telle que pour tout n-uplet tp généré par G et tout paquet de productions synchronisées $\{X^1 \Rightarrow Y^1, \ldots, X^n \Rightarrow Y^n\}_k$ de G applicable sur tp (aux positions $p_1, \ldots, p_n$), il existe au plus une position parmi $p_1, \ldots, p_n$, appartenant au i-ème composant de tp.*

Proposition 2.6 *Si le i-ème composant de L est libre, alors la projection de L sur ce composant est un langage régulier d'arbres.*

On dit alors que i-ème composant de L est *régulier*. La réciproque est fausse.

Exemple 2.7 Le premier composant de $L = \{(c(f^n(a), g^m(a)), s(f^n(g^m(a))))\}$ est régulier. Mais pour compter n, il faut dériver de manière synchronisée les branches 1 et 3 du n-uplet. Il faut aussi que la branche 2 soit synchronisée avec les branches 1 et 3, afin de pouvoir ensuite dériver de manière synchronisée les branches 2 et 3 pour compter m. Puisque les branches 1 et 2 appartiennent au même composant (le premier) du n-uplet, ce composant n'est pas libre. $\diamond$

Théorème 2.8 *[64, 65, 63] Les TTSG sont stables par produit cartésien, union, intersection avec un langage régulier*[1]*, et par i,j-jointure si un des deux composants i ou j est libre.*

Pour obtenir des propriétés plus intéressantes (décidabilité du vide, stabilité par intersection), nous avons étudié deux sous-classes.

2.1.1 Les TTSG à trans-classes bornées

Dans un langage régulier d'arbres, deux branches différentes d'un même arbre sont indépendantes l'une de l'autre : si p et p' sont deux positions incomparables de t, on ne peut pas imposer de mettre à la position p' le symbole f' si $t(p) = f$ et g' si $t(p) = g$. En effet rien ne pourra empêcher d'avoir $t(p) = f$ et $t(p') = g'$. Avec les TTSG, on peut lier deux branches différentes grâce à la synchronisation. Une TTSG est à trans-classes

1. L'intersection d'une TTSG d'arbres avec un langage régulier est encore une TTSG.

bornées si le nombre de branches liées (synchronisables) ensemble est borné, quelle que soit la taille de l'arbre généré.

Définition 2.9 *Soit t un arbre dérivé par une TTSG G, tel que $t(p) = X_{ct} \in NT$ et $t(p') = X'_{ct'} \in NT$. Les positions p et p' sont* synchronisables en un coup *s'il existe un composant (disons le i^{eme}) de ct et ct' tel que $ct(i) = ct'(i) \neq \perp$.*
On définit la relation 'synchronisable' *comme étant la fermeture transitive de 'synchronisable en un coup'*[2]. *Remarquons que 'synchronisable' est une relation d'équivalence.*
Une trans-classe *est une classe d'équivalence de 'synchronisable'.*
Une TTSG G est à trans-classes bornées *s'il existe un naturel k tel que la taille de toutes les trans-classes de tous les arbres générés par G ne dépasse pas k.*

Le langage $Id_2 = \{(t,t) \mid t \in T_\Sigma\}$ peut être généré par une TTSG à trans-classes bornées, car il suffit de synchroniser les positions identiques du membre gauche et du membre droit du couple, autrement dit la taille des trans-classes est 2.

Par contre, l'ensemble des termes clos équilibrés (toutes les feuilles sont à la même profondeur) n'est pas généré par une TTSG à trans-classes bornées car si à une profondeur donnée il existe deux branches non synchronisables, elles seront dérivées ensuite indépendamment et rien ne permettra plus d'imposer qu'elles soient de même longueur. Autrement dit, pour chaque k, toutes les positions de profondeur k sont dans la même trans-classe. D'où des trans-classes de plus en plus grandes.

Théorème 2.10 *[64, 65, 63] Les TTSG à trans-classes bornées sont stables par produit cartésien, union, intersection avec un langage régulier, et par i,j-jointure si un des deux composants i ou j est libre. De plus le vide est décidable.*

Les propriétés de stabilité sont obtenues en appliquant le théorème 2.8 et en vérifiant que les TTSG résultats sont à trans-classes bornées. Le vide est obtenu par une technique de pompage similaire à celle des langages réguliers d'arbres, sauf qu'au lieu d'appliquer le pompage sur des non-terminaux, on l'applique sur des NT-trans-classes (obtenues en remplaçant les positions des trans-classes par des non-terminaux), qui sont en nombre fini car les trans-classes sont bornées et l'ensemble des non-terminaux est supposé fini.

Lemme 2.11 *[63] Le vide de l'intersection de deux TTSG à trans-classes bornées est indécidable.*

Preuve : Par codage du problème de Post : soient A et C deux alphabets et Φ, Φ' deux morphismes de A^* vers C^*. Considérons les deux langages d'arbres synchronisés $L = \{(\alpha,\phi(\alpha)) | \, \alpha \in A^+\}$ et $L' = \{(\alpha,\phi'(\alpha)) | \, \alpha \in A^+\}$. Pour chaque $a_i \in A$, posons $\phi(a_i) = c_{i,1} \dots c_{i,p_i}$. Alors L est généré par la TTSG Gr qui, pour tout $a_i \in A$, contient les productions : $\{X \Rightarrow X_i, Y \Rightarrow Y_i\}$, $X_i \Rightarrow a_i Z$, $Y_i \Rightarrow c_{i,1} Y_{i,2}$, $Y_{i,2} \Rightarrow c_{i,2} Y_{i,3}$, $\dots$, $Y_{i,p_i} \Rightarrow c_{i,p_i} T$, $\{Z \Rightarrow X, T \Rightarrow Y\}$, $\{Z \Rightarrow \perp, T \Rightarrow \perp\}$, et dont l'axiome

2. Pour une TTSG de rang 1, 'synchronisable' et 'synchronisable en un coup' sont identiques.

est (X,Y). La TTSG Gr' qui génère L' est définie de la même manière. Ainsi tester le vide de $L \cap L'$ revient à tester l'existence de $\alpha \in A^+$ tel que $\phi(\alpha) = \phi'(\alpha)$, donc à résoudre le problème de Post.

Une conséquence immédiate est :

Corollaire 2.12 *Les TTSG à trans-classes bornées ne sont pas stables par intersection, ni donc par complément.*

Le résultat suivant est dû à S. Limet et F. Saubion.

Théorème 2.13 *[72, 71] L'inclusion d'un langage généré par une TTSG à trans-classes bornées de rang 1, dans une relation régulière[3], est décidable.*

2.1.2 Les TTSG à inclinaison bornée

Ce sont les TTSG pour lesquelles la différence de profondeur entre deux positions synchronisables est bornée.

Définition 2.14 *La TTSG G est* à inclinaison bornée *s'il existe un entier k tel que pour tout arbre t généré par G et toute trans-classe Tr de t, $p,p' \in Tr \implies |prof(p) - prof(p')| \leq k$ ($prof(p)$ est la profondeur de la position p). Si $k = 0$ convient, G est dite* horizontale.

L'ensemble $\{(t,f(t)) \mid t \in T_\Sigma\}$ peut être généré par une TTSG à inclinaison bornée, puisque la différence de profondeur entre deux points de synchronisation est toujours de 1. L'ensemble des termes équilibrés peut être généré par une TTSG horizontale. Par contre l'ensemble $\{(s^n(0),s^{2n}(0))\}$ n'est pas généré par une TTSG à inclinaison bornée.

Au rang 1, on peut toujours se ramener au cas horizontal.

Théorème 2.15 *[66, sect. 4.2] Pour toute TTSG à inclinaison bornée de rang 1, il existe une TTSG horizontale (de rang 1) qui génère le même langage.*

Théorème 2.16 *Les TTSG horizontales sont stables par produit cartésien, union, intersection. Par conséquent elles sont stables par i,j-jointure sans aucune restriction.*

La stabilité par produit cartésien et union est obtenue en appliquant le théorème 2.8 et en vérifiant que les TTSG résultats sont horizontales. La stabilité par intersection est prouvée dans [68, 66]. L'intersection de TTSG horizontales de rang n et m donne une TTSG horizontale de rang $n + m$.

Malheureusement, le test du vide par pompage utilisé pour les TTSG à trans-classes bornées ne s'applique pas ici, puisque justement les trans-classes ne sont pas forcément bornées.

Question ouverte : Le vide des TTSG à inclinaison bornée est-il décidable ?

3. Les relations régulières sont définies en section 2.2.2.

2.2 Les systèmes de contraintes (CS)

Les *CS* (Constraint Systems) sont en fait une reformulation des TTSG à trans-classes bornées de rang 1, que nous avons effectuée pour obtenir un formalisme plus simple. Il s'agit de systèmes de contraintes ensemblistes, dont le langage reconnu est par définition le plus petit point fixe. On peut tout à fait les considérer comme des grammaires en remplaçant $\supseteq$ par $\Rightarrow$.

Par rapport aux TTSG, les CS ont pour particularités :

1. chaque non-terminal (appelé variable) désigne un n-uplet d'arbres, au lieu d'un arbre,
2. le contrôle n'existe pas, car il est alors inutile.

La grande force des CS réside dans la disparition de ce contrôle fort désagréable.

Exemple 2.17 Dans la signature $\Sigma = \{f^{\backslash 2}, b^{\backslash 0}\}$ considérons le langage $L_{id} = \{(t,t) \mid t \in T_\Sigma\}$ des couples de termes identiques. Ce langage a déjà été défini au moyen des TTSG dans l'exemple 2.1. Il peut aussi être défini par le CS :

$$\begin{array}{lcl} X_{id} & \supseteq & (b,b) \\ X_{id} & \supseteq & (f(1_1,2_1), f(1_2,2_2))\ (X_{id},X_{id}) \end{array}$$

où 1_1,2_1,1_2,2_2 sont des abréviations de couples (pour la lisibilité). Par exemple, 2_1 veut dire (2,1), et représente le premier composant du second argument (le second X_{id}). Remarquons que puisque 1_1 et 1_2 proviennent du même X_{id}, ils représentent deux termes identiques, autrement dit ils sont liés, c-à-d synchronisés au sens des TTSG : ils sont dans la même trans-classe d'une TTSG de rang 1. Par contre 1_1 et 2_1 sont indépendants. $\diamond$

Exemple 2.18 Le CS :

$$\begin{array}{lcl} X_{sym} & \supseteq & (b,b) \\ X_{sym} & \supseteq & (f(1_1,2_1), f(2_2,1_2))\ (X_{sym},X_{sym}) \end{array}$$

reconnaît le langage $L_{sym} = \{(t,t_{sym}) \mid t_{sym}$ est le symétrique de $t\}$. $\diamond$

Exemple 2.19 Dans la signature $\Sigma = \{s^{\backslash 1}, b^{\backslash 0}\}$, le langage $L_{dble} = \{(s^n(b),s^{2n}(b))\}$ est reconnu par le CS :

$$\begin{array}{lcl} X_{dble} & \supseteq & (b,b) \\ X_{dble} & \supseteq & (s(1_1), s(s(1_2)))\ X_{dble} \end{array}$$

$\diamond$

Les CS ressemblent aux contraintes ensemblistes définies [50], car tous deux possèdent un mécanisme de projection, et n'ont pas de négation. Pourtant ils n'ont pas la même expressivité, car si on essaye d'exprimer le langage d'arbres $L'_{sym} = \{c(t,t_{sym})\}$ par des contraintes ensemblistes, on obtient :

$$X \supseteq c(a,a) \cup c(b,b) \cup c\left(f(c_1^{-1}(\boldsymbol{X}),c_1^{-1}(X)),\ f(c_2^{-1}(X),c_2^{-1}(\boldsymbol{X}))\right)$$

Mais il n'y a aucun moyen de forcer les deux occurrences de $\boldsymbol{X}$ à être instanciées par la même valeur, alors qu'avec un CS, on aurait pu utiliser les deux composant d'une seule occurrence de X. Par conséquent, la solution la plus petite est $\{c(t,t') \mid t,t' \in T_{\{f,a,b\}}\}$ qui est un langage régulier. En fait, un système de contraintes ensemblistes a toujours une plus petite solution, qui est régulière.

Formalisation générale

On se donne un ensemble d'indices N pour les composants de n-uplets. Pour $I \subseteq N$ et un ensemble quelconque M, l'ensemble des I-uplets $a : I \to M$ est noté M^I. On écrira aussi $a = (a_i)_{i \in I}$ où $a_i = a(i)$, et lorsque $I = \{1,\ldots,k\} \subseteq \mathbb{N}$ on écrira $a = (a_1,\ldots,a_k)$.

Des langages différents de n-uplets d'arbres peuvent utiliser des n-uplets de longueurs différentes, et des ensembles d'indices différents. Chaque variable contrainte (c-à-d non-terminal en utilisant un vocabulaire de grammaire) représente un langage, et on lui associe un ensemble d'indices. Un ensemble *classifié* de variables de n-uplets (sur N) est un couple $(\mathcal{X},\rho)$ où $\rho : \mathcal{X} \to 2^N$ associe à chaque variable X un sous-ensemble d'indices, appelé *classe* de X. Lorsque ρ est implicitement connu, on confondra $(\mathcal{X},\rho)$ avec $\mathcal{X}$. La cardinalité maximale des classes des éléments de $\mathcal{X}$ est appelée *largeur* de $\mathcal{X}$. Sur l'exemple 2.17, on a $N = \{1,2\}$, $\mathcal{X} = \{X_{id}\}$, et $\rho(X_{id}) = \{1,2\}$. Ainsi la largeur de $\mathcal{X}$ est 2.

Un *système de contraintes (CS)* est formé d'un ensemble classifié de variables contraintes $(\mathcal{X},\rho)$, et d'un ensemble $\mathcal{E}$ d'inéquations de la forme :

$$X \supseteq \Box(X_1,\ldots,X_k) \tag{1}$$

où $X,X_1,\ldots,X_k \in \mathcal{X}$ et $\Box$ est un opérateur fournissant un n-uplet d'arbres pour X à partir de la concaténation des n-uplets d'arbres des variables X_i. Plus précisément, soit :

$$J = \{(i,x) \mid 1 \leq i \leq k, x \in \rho(X_i)\} \tag{2}$$

l'*union disjointe* des ensembles d'indices des variables $X_1,\ldots,X_k$ (dans l'exemple 2.17, $J = \{(1,1),(1,2),(2,1),(2,2)\}$, abrégé en $\{1_1,1_2,2_1,2_2\}$). En fait $\Box$ est une application $T_\Sigma^J \to T_\Sigma^{\rho(X)}$, dont chaque composant est spécifié par une expression d'arbres t qui peut accéder aux composants des variables $X_1,\ldots,X_k$, et appliquer des symboles de la signature Σ. Ainsi t peut être représenté par un élément de $T_\Sigma(J)$ où $T_\Sigma(J)$ sont les arbres sur Σ qui peuvent contenir en plus des symboles d'arité 0 pris dans l'ensemble d'indices J. Dans l'exemple 2.17, le premier composant de l'opérateur de la seconde contrainte est $t = f(1_1,2_1)$.

L'application induite par t est définie par :

$$t(s_j)_{j \in J} = t\{j \mapsto s_j\}_{j \in J}$$

pour tout $(s_j)_{j \in J} \in T_\Sigma^J$. Par conséquent, $\Box$ est donné par le n-uplet :

$$\Box \in T_\Sigma(J)^{\rho(X)}$$

La *taille* d'un CS est la somme des tailles de ses contraintes. La *taille* de la contrainte (1) est définie par :

$$|(1)| = k + \sum_{x \in \rho(X)} (1 + |\Box_x|)$$

où $|\Box_x|$ est la taille du terme définissant le x-ème composant $\Box_x$ de $\Box$. Ainsi, dans l'exemple 2.17, la taille de la deuxième contrainte est $2 + (1 + 3) + (1 + 3) = 10$.

Le CS est dit :

- *non-copiant* si aucun indice $j \in J$ apparaît deux fois dans $\Box$;
- *vide* si $\Box$ n'applique aucun symbole de Σ, c-à-d $\Box_x \in J$ pour tout $x \in \rho(X)$ (où $\Box_x$ est le x-ème composant de $\Box$);
- *Greibach* si tout composant $\Box_x$ est contenu dans J, ou est l'application d'un seul symbole de Σ, c-à-d $\Box_x = a(j_1, \ldots, j_n)$, $a \in \Sigma, j_1, \ldots, j_n \in J$;
- *régulier* si chaque composant de $\Box_x$ de $\Box$ est de la forme $\Box_x = a((1,x), \ldots, (n,x))$ où $a \in \Sigma$;
- *horizontal* si pour tous composants $\Box_x$, $\Box_y$ et toutes positions p, q; $\Box_x|_p = (i,x')$, $\Box_y|_q = (i,y')$ implique p et q sont à la même profondeur.

Un CS est dit non-copiant (non-vide, Greibach, régulier, horizontal) si toutes ses contraintes le sont.

Par exemple, les CS qui définissent L_{id} et L_{sym} sont non-copiants, Greibach et horizontaux. De plus L_{id} est régulier. Par ailleurs L_{dble} n'est ni Greibach, ni horizontal. Remarquons qu'il est toujours possible de mettre un CS en forme Greibach. Par exemple, la forme Greibach de L_{dble} est :

$$\begin{array}{lcl} X_{dble} & \supseteq & (b,b) \\ X_{dble} & \supseteq & (s(1_1), s(1_2))\ Y \end{array} \qquad Y \supseteq (1_1, s(1_2))\ X_{dble}$$

Une instanciation de variables est une application σ affectant à chaque variable $X \in \mathcal{X}$ un sous-ensemble de $T_\Sigma^{\rho(X)}$ (c-à-d un ensemble de n-uplets de termes clos). σ satisfait la contrainte (1) ssi :

$$\sigma(X) \supseteq \{\Box(t_1, \ldots, t_k) \mid t_i \in \sigma(X_i)\} \tag{3}$$

Remarquons que l'opérateur $\Box$ est en fait appliqué au produit cartésien des affectations des arguments $\sigma(X_i)$.

σ est une *solution* du CS si elle satisfait toutes ses contraintes. Puisque dans (3), l'application d'un opérateur est monotone (par rapport à l'inclusion), un CS admet une plus petite solution, qui est donc unique.

2.2.1 Propriétés

Avertissement : dans toute la suite, nous considérons uniquement des CS non-copiants.

Théorème 2.20 *Les CS et les TTSG de rang 1 à trans-classes bornées ont la même expressivité.*

Preuve : On transforme un CS en une TTSG de rang 1 équivalente en remplaçant chaque contrainte $X_0 \supseteq \Box(X_1,\dots,X_k)$ par le paquet de productions synchronisées :

$$\{X_0^j \Rightarrow Y_0^j \mid j \in \rho(X_0)\}$$

et les productions libres :

$$Y_0^j \Rightarrow \Box_j((i,y)/X_i^y), \quad j \in \rho(X_0)$$

où $(i,y)/X_i^y$ est le remplacement de (i,y) par X_i^y. Puisque le CS est non-copiant, chaque (i,y) apparaît une seule fois dans $\Box$, et donc en dérivant la TTSG ainsi obtenue, on ne pourra pas produire plusieurs occurrences du même non-terminal avec le même contrôle. Le nombre de non-terminaux étant fini (disons n), et la TTSG étant de rang 1, la taille d'une trans-classe est au plus n.

Réciproquement, pour chaque paquet $\{X^j \Rightarrow Y^j \mid j = 1,\dots,n\}$ et pour chaque sous-ensemble de productions libres :

$$\{\ Y^j \Rightarrow t^j(Z^{p_1}_{i_{(j,1)}},\dots,Z^{p_{k_j}}_{i_{(j,k_j)}}) \mid j = 1,\dots,n\ \}$$

on crée la contrainte $X \supseteq \Box(\dots,Z_{i_{(j,l)}},\dots)$ où

$$\forall j \in \{1,\dots,n\},\ \Box_j = t^j\left((Z^{p_m}_{i_{(j,m)}} \,/\, (i_{(j,m)},p_m)) \mid m = 1,\dots,k_j\right)$$

La TTSG de départ étant de rang 1 et à trans-classes bornées, le même $Z^{p_m}_{i_{(j,m)}}$ ne peut pas apparaître plusieurs fois dans les t^j. C'est pourquoi le système de contraintes ainsi obtenu est non-copiant.

Les propriétés énoncées ci-dessous proviennent de [43, 42].

Théorème 2.21 *Soit t un n-uplet de termes clos, et L un langage défini par un CS de taille n utilisant un ensemble classifié de variables de largeur r. L'appartenance ($t \in L$?) est décidable en temps $\mathcal{O}(n \cdot (|t| + 1)^r)$.*

D'autre part le vide est décidable en temps linéaire par rapport à la taille du CS.

Théorème 2.22 *Les CS sont stables par union, produit cartésien, projection.*

Théorème 2.23 *Si L_1, L_2 sont définies par des CS, et le composant i de L_1 est libre, alors $L_1 \bowtie_{i,j} L_2$ peut être défini par un CS. De plus si k est un composant libre de L_2, alors le composant correspondant dans L_3 est encore libre.*

Si L_1 est un langage régulier d'arbres, son unique composant est libre. Par conséquent, les CS sont stables par jointure avec un langage régulier d'arbres.

Corollaire 2.24 *Soient L_1 un langage d'arbres et Rel une relation binaire d'arbres. Si L_1 et Rel sont définies par des CS, et le premier composant de Rel est libre, alors l'image $Rel(L_1)$ de L_1 par Rel est un langage d'arbres défini par un CS.*

Preuve : $Rel(L_1) = \Pi_2(L_1 \bowtie Rel)$.

Lemme 2.25 *[70, 69] Le vide de l'intersection de deux CS est indécidable.*

Par conséquent, les CS ne sont pas stables par intersection.

Par rapport aux TTSG à trans-classes bornées, les CS apportent :

- la disparition du contrôle, en conservant de plus la stabilité par jointure,
- un test du vide de complexité linéaire, au lieu d'un test exponentiel basé sur une technique de pompage compliquée à cause des n-uplets de contrôle,
- la stabilité par projection.

Un test d'appartenance sur les TTSG de rang 1 avait été proposé par F. Saubion et I. Stéphan [99] en utilisant la logique linéaire. Le test sur les CS a l'avantage d'être beaucoup plus simple.

Grâce à l'équivalence entre CS et TTSG de rang 1 à trans-classes bornées, on obtient :

Corollaire 2.26 *Les TTSG à trans-classes bornées de rang 1 sont stables par union, produit cartésien, projection, et par i,j-jointure si un des deux composants i ou j est libre. De plus le vide est décidable.*

La stabilité par jointure des TTSG à trans-classes bornées de rang 1 est inattendue. Cela signifie en effet qu'il était inutile de compliquer le contrôle. On peut alors se demander s'il existe des TTSG à trans-classes bornées de rang supérieur à 1 qui ne proviendraient pas de la jointure de TTSG de rang 1 ? Intuitivement, d'après la preuve de stabilité par jointure des CS, il semble possible de transformer les trans-classes de rang n en trans-classes de rang 1 en augmentant la taille des paquets de productions synchronisées de la grammaire. D'où la conjecture forte :

Conjecture 2.27 *Pour tout n, les TTSG à trans-classes bornées de rang n et de rang 1 ont la même expressivité.*

2.2.2 Les relations régulières (RR)

La notion de langage régulier de n-uplets (appelée ici Relation Régulière) a d'abord été définie au moyen de transducteurs finis par M. Dauchet et S. Tison [31], qu'on peut aussi considérer comme des automates sur l'alphabet produit [25].

Une relation R est *régulière* si le langage d'arbres sur l'alphabet produit obtenu en superposant entre eux les composants de chaque n-uplet, est régulier. Formellement, soit $\Sigma_\oplus$ l'alphabet produit défini par $\Sigma_\oplus = (\Sigma \cup \{\perp\}) \times (\Sigma \cup \{\perp\}) - \{\perp\perp\}$ où $\perp$ est une nouvelle constante. Pour $s,t \in T_\Sigma$, on définit récursivement $s \oplus t \in T_{\Sigma_\oplus}$ par :

$$f(s_1,\ldots,s_n) \oplus g(t_1,\ldots,t_m) = \begin{cases} fg(s_1 \oplus t_1,\ldots,s_n \oplus t_n,\perp \oplus t_{n+1},\ldots,\perp \oplus t_m) \\ \quad \text{si } n < m \\ fg(s_1 \oplus t_1,\ldots,s_m \oplus t_m,s_{m+1} \oplus \perp,\ldots,s_n \oplus \perp) \\ \quad \text{sinon} \end{cases}$$

Par exemple $f(a,g(b)) \oplus f(f(a,a),b)$ est $ff(af(\perp a,\perp a),gb(b\perp))$. Cette définition s'étend trivialement au produit de k termes $t_1 \oplus \ldots \oplus t_k$.

Une relation n-aire $R \subseteq T_\Sigma^n$ est régulière si le langage d'arbres $\{t_1 \oplus \ldots \oplus t_n \mid (t_1,\ldots,t_n) \in R\}$ est régulier. Par exemple $\{(t,t) \mid t \in T_\Sigma\}$, et pour des symboles fixés $f,g \in \Sigma$, $\{(t,t[f \leftarrow g]) \mid t \in T_\Sigma\}$ sont des RR, où $t[f \leftarrow g]$ est le terme obtenu en remplaçant dans t toutes les occurrences du symbole f par g. Par ailleurs $\{(t,t_{sym}) \mid t \in T_\Sigma, t_{sym}$ est le symétrique de $t\}$ n'est pas une RR.

Proposition 2.28 *[70, 69] Les CS réguliers et les RR définissent la même classe de langages.*

On remarquera que les RR ont les mêmes propriétés que les langages réguliers d'arbres.

Le logiciel TAJA [4], initialement développé à Angers par des étudiants de F. Saubion, puis poursuivi à Orléans par Benoit Lecland, implante les RR, et donc aussi les langages réguliers d'arbres. Une particularité intéressante de TAJA est qu'il comporte une interface graphique.

2.2.3 Les relations faiblement régulières (WRR)

Il s'agit d'une sur-classe des RR, et d'une sous-classe des langages synchronisés, stable par intersection. Elle est définie à l'aide des CS.

Pour comprendre les restrictions nécessaires à la stabilité par intersection, étudions d'abord quelques exemples.

Exemple 2.29 Soient L_1, L_2 générés respectivement par X_1 et X_2 :

L_1	L_2
$X_1 \supseteq (1_1,g(1_2))Y_1$	$X_2 \supseteq (f(1_1),1_2)Y_2$
$Y_1 \supseteq (f(1_1),f(1_2))Z_1$	$Y_2 \supseteq (g(1_1),g(1_2))Z_2$
$Z_1 \supseteq (g(b),b)$	$Z_2 \supseteq (b,f(b))$

On voit que $L_1 = L_2 = \{(f(g(b)),g(f(b)))\}$. Donc $L_1 \cap L_2$ n'est pas vide, mais dans L_1 la position 1 (c-à-d la racine du premier du premier composant) est synchronisée

4. Voir l'URL http://www.univ-orleans.fr/SCIENCES/LIFO/Members/lecland/taja.html pour télécharger.

avec la position 2.1, alors que dans L_2, c'est 1.1 et 2 qui sont synchronisées. Un CS qui reconnaitraît $L_1 \cap L_2$ doit simuler en même temps les CS de L_1 et de L_2. En dérivant un tel CS, on abouti alors à une situation d'interblocage. $\diamond$

Ce problème provient du fait que dans L_1, L_2 il y a des synchronisations penchées, c-à-d les deux points de la synchronisation ne sont pas à la même profondeur.

Pour éviter cela, considérons uniquement des CS horizontaux.

Exemple 2.30 Soient $Id_2 = \{(t,t) \| t \in T_\Sigma\}$ et $L_{sym} = \{(t,t_{sym}) \| t \in T_\Sigma$, t_{sym} est le symétrique de $t\}$, avec $\Sigma = \{f^{\backslash 2}, b^{\backslash 0}\}$. Ainsi, $Id_2 \cap L_{sym}$ est l'ensemble des couples (t,t') tels que $t = t'$ et t est le symétrique de t'. Cela signifie que t est lui-même un arbre symétrique (par rapport à sa racine). Donc $Id_2 \cap L_{sym}$ est l'ensemble $\{(b,b)\} \cup \{(f(t,t_{sym}), f(t,t_{sym}))\}$. Soit $X = \{(t,t_{sym},t,t_{sym})\}$, alors $Id_2 \cap L_{sym}$ peut être défini par :

$$\begin{aligned} Id_2 \cap L_{sym} &\supseteq (b,b) | (f(1_1,1_2), f(1_3,1_4))X \\ X &\supseteq (f(1_1,2_1), f(2_2,1_2), f(1_3,2_3), f(2_4,1_4))(X,X) | (b,b,b,b) \end{aligned}$$

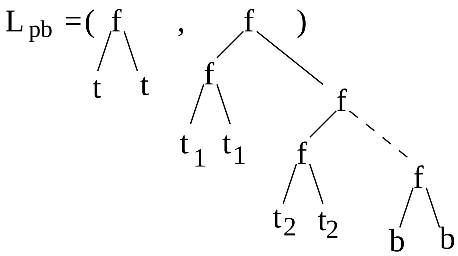
$\diamond$

Dans l'exemple ci-dessus on a besoin du langage intermédiaire de quadruplets X pour définir le langage de couples $Id_2 \cap L_{sym}$. D'une manière générale, ce processus d'augmentation de la taille des n-uplets peut nécessiter plusieurs étapes, et même ne pas terminer :

Exemple 2.31 Soit L_{pb} le langage défini par :

$$L_{pb} \supseteq (f(1_1,1_2), f(2_1,2_2))(Id_2, L_{pb}) \qquad L_{pb} \supseteq (b,b)$$

Le schéma ci-dessous donne une idée de l'allure de L_{pb} :



$L_{pb} \cap Id_2$ est le langage des couples de termes équilibrés [5] et identiques. Pour l'exprimer par un CS, il faut synchroniser toutes les positions de même profondeur ensemble, ce qui nécessite des n-uplets de plus en plus grands, donc un nombre infini de langages intermédiaires. Ceci est impossible puisqu'un CS est toujours supposé fini. $\diamond$

Ainsi donc, l'horizontalité ne suffit pas. Il faut de plus garantir que la taille des n-uplets n'est augmentée qu'un nombre fini de fois.

5. Toutes les branches ont la même longueur.

La définition ci-dessous est une condition suffisante, mais pas nécessaire comme le montre l'exemple 2.30 (L_{sym} n'est pas une WRR).

Définition 2.32 *(préliminaire) Soit $\mathcal{C}$ un CS. On dit que $X > Y$ s'il existe dans $\mathcal{C}$ une contrainte de la forme $X \supseteq \Box(\ldots,Y,\ldots)$.*
Une contrainte $X \supseteq \Box(X_1,\ldots,X_n)$ est récursive *si au moins un des X_i vérifie $X_i >^* X$.*

Définition 2.33
Un CS est faiblement régulier *si toute contrainte récursive $X \supseteq \Box(X_1,\ldots,X_n)$ est régulière, et parmi les langages définis par $X_1,\ldots,X_n$, au plus un n'est pas une RR.*
*Une relation est faiblement régulière (*WRR *pour Weakly Regular Relation) si elle est défini par un CS faiblement régulier.*

Remarquons qu'aucune restriction n'est imposée sur les contraintes non récursives.

Exemple 2.34 Soient $\Sigma = \{f^{\backslash 2},g^{\backslash 2},a^{\backslash 0}\}$ et le système de réécriture $R = \{f(x,y) \to g(y,x)\}$, et soit $S = \{(t_1,t_2) \mid t_1 \to_R t_2\} = \{(C[f(x,y)\sigma],C[g(y,x)\sigma])\}$. S est défini par le CS suivant :

$$
\begin{array}{lllllll}
S & \supseteq & (f(1_1,2_1),f(1_2,2_2))(S,Id_2) & \quad & S & \supseteq & (g(1_1,2_1),g(1_2,2_2))(S,Id_2) \\
S & \supseteq & (f(1_1,2_1),f(1_2,2_2))(Id_2,S) & \quad & S & \supseteq & (g(1_1,2_1),g(1_2,2_2))(Id_2,S)
\end{array}
$$

$$S \supseteq (f(1_1,2_1),g(2_2,1_2))(Id_2,Id_2)$$

$$
\begin{array}{lllllll}
Id_2 & \supseteq & (f(1_1,2_1),f(1_2,2_2))(Id_2,Id_2) & \quad & Id_2 & \supseteq & (g(1_1,2_1),g(1_2,2_2))(Id_2,Id_2)
\end{array}
$$

$$Id_2 \supseteq (a,a)$$

S est une WRR car toutes les règles récursives sont régulières et $Id_2 = \{(t,t) \mid t \in T_\Sigma\}$ est une RR. $\diamond$

Théorème 2.35 *[70, 69] Les WRR sont stables par union, projection, produit cartésien, intersection.*

Corollaire 2.36 *Les WRR sont stables par jointure.*

Preuve : Supposons que L_1 (resp. L_2) soit un langages de n-uplets de taille l (resp. n).

$$
\begin{array}{rl}
L_1 \bowtie_{i,j} L_2 = & \Pi_{1,\ldots,l+j-1,l+j+1,\ldots,l+n}(L_1 \times L_2) \\
& \cap\{(t_1,\ldots,t_i,\ldots,t_l,t_{l+1},\ldots,t_{l+j-1},t_i,t_{l+j+1},\ldots,t_{l+n}) \mid \forall k,t_k \in T_\Sigma\}
\end{array}
$$

Lemme 2.37 *Les WRR ne sont pas stables par complément.*

Preuve : L'ensemble des termes non-équilibrés sur la signature $\Sigma = \{f^{\backslash 2},a^{\backslash 0}\}$ est généré par le CS (V est l'axiome) :

$$
\begin{array}{lll}
V & \supseteq & f(1_1,2_1)(V,Id) \\
V & \supseteq & f(1_1,2_1)(Id,V) \\
V & \supseteq & f(1_1,1_2)\,U
\end{array}
\qquad
\begin{array}{lll}
U & \supseteq & (f(1_1,2_1),f(1_2,2_2))(U,Id^2) \\
U & \supseteq & (f(1_1,2_1),f(1_2,2_2))(Id^2,U) \\
U & \supseteq & (a,f(Id,Id)) \\
U & \supseteq & (f(Id,Id),a)
\end{array}
$$

où $Id = T_\Sigma$ et $Id^2 = Id \times Id$. C'est une WRR car Id est régulier, et la seule contrainte non régulière est $V \supseteq f(1_1,1_2)U$, qui n'est pas récursive.

Le complément de l'ensemble des termes déséquilibrés est l'ensemble des termes équilibrés, et ce n'est pas une WRR car comme on l'a vu dans l'exemple 2.31, il ne peut pas être défini par un CS.

2.3 Travaux apparentés

Les TTSG sont apparues en 1997. La même année, J.C. Raoult a publié un formalisme de grammaires de n-uplets d'arbres, très proche des TTSG [92]. Le contrôle (appelé instance) est composé d'entiers, et contrairement aux TTSG, les non-terminaux des membres droits d'un même paquet de productions synchronisées peuvent hériter de contrôles différents. Ceci dit, on peut se ramener à une TTSG en augmentant le nombre de paquets de productions synchronisées. De plus, vu les restrictions supposées, on voit trivialement [6] que les grammaires de Raoult sont équivalent aux TTSG de rang 1 à trans-classes bornées, qui, rappelons le, sont elles-mêmes équivalentes aux CS.

J.C. Raoult montre de plus que ses grammaires sont équivalentes aux relations rationnelles de n-uplets, c-à-d les relations définies par des expressions régulières de n-uplets d'arbres (à ne pas confondre avec les relations régulières (RR) de la section 2.2.2), et aux transducteurs d'arbres descendants k-copiants. Il montre aussi la stabilité par produit cartésien (appelé concaténation), union, et projection. Une stabilité par jointure (appelée composition [7]) est établie pour une sous-classe, les transductions, qui revient à supposer que tous les composants des n-uplets sont libres. Cette sous-classe contient les transductions d'Arnold et Dauchet [8].

Par rapport à ceux de Raoult, nos résultats sur les CS comprennent en plus le test d'appartenance, le test du vide, la stabilité par jointure avec une restriction moins forte (un seul composant doit être libre). Or ces trois propriétés sont très importantes, car sans elles les applications présentées dans le chapitre suivant n'existeraient pas. Le tableau ci-dessous résume les résultats, où x signifie que la propriété a été prouvée dans le formalisme en question.

6. A condition de modifier la définition 2.7 de [92] (dérivations de la grammaire), qui ne mentionne pas l'incrémentation du contrôle. Il doit s'agir d'un oubli puisque les exemples donnés par ailleurs supposent que cette incrémentation a bien lieu.

7. Contrairement à la jointure, la composition ne conserve pas dans la relation résultat le composant de jointure.

	CS's non-copiants	Raoult	TTSG's à TC bornées de rang 1
Prop. de décision			
$\in$	x	x	x
$= \emptyset$	L-time		Exp-time
Prop. de stabilité			
$\cup$	x	x	x
$\times$	x	x	x
Π	x	x	?
$\bowtie$	restreinte	très restreinte	restreinte, = rang 2
$\bowtie$ lang. d'arbres. rég.	x		= rang 2
$\cap$	non		non

Si on compare maintenant les CS avec les automates avec contraintes d'égalité et de diségalité [15, 29], les CS permettent des contraintes qui ne sont ni l'égalité ni la diségalité, ils peuvent exprimer par exemple $\{f(s^n(a), p^n(a))\}$, mais ne peuvent pas exprimer l'ensemble des termes équilibrés. Cela provient du fait qu'une synchronisation entre deux positions p, q effectue un test sur les symboles apparaissant à ces deux positions, alors qu'une contrainte de (dis)égalité entre p, q effectue un test sur les symboles apparaissant à toutes les positions situées en dessous (au sens large) de p et q.

Par rapport aux schématisations récurrentes de termes : ρ-termes [21], I-termes [23], R-termes [98], grammaires primales [51, 2], les CS ont l'avantage de contenir les langages réguliers d'arbres. Par contre, ils n'expriment que des termes clos.

Chapitre 3

Applications

Ce chapitre présente quatre applications : une à la concurrence, une à la programmation logique, et deux à la réécriture.

3.1 Les algèbres de processus

Nous avons introduit dans [43, 42] un nouveau formalisme de concurrence, appelé BCPC (Bounded Communication Process Calculus), intermédiaire entre les algèbres de processus PA [10] (sans communications) et CCS [82] (avec communications). Dans BCPC, la communication entre processus est autorisée, mais seulement à l'intérieur d'un terme de communication, l'ensemble des termes de communication étant fixé et fini. Ainsi, partant d'un nom de processus x, et dérivant un terme processus t, on ne pourra pas faire communiquer (synchroniser) deux noms de processus y, z apparaissant dans t si y et z sont trop loin l'un de l'autre car ils ne pourront pas être châpeautés par un même terme de communication. On peut voir cela comme un time-out, le temps maximal autorisé étant délimité par les termes de communication.

Nous avons montré que sur les termes processus (donc une structure infinie), le model-checking des formules d'une restriction de la logique temporelle CTL ne contenant que les opérateurs temporels EX et EF, est décidable. Une conséquence est que l'absence d'interblocages est décidable.

3.1.1 PA

PA est une algèbre de processus qui permet le non-déterminisme, les compositions séquentielles et parallèles, et la récursivité.

Syntaxe et sémantique

Act est un ensemble d'*actions*, $Act = \{a,b,c,\ldots\}$. $Const$ est un ensemble de *noms de processus*, $Const = \{x,y,z,\ldots\}$. T est l'ensemble $\{t_1,t_2,\ldots\}$ des PA-termes définis par :

$$t_1,t_2 ::= 0 \;\mid\; x \;\mid\; t_1 \parallel t_2 \;\mid\; t_1 \,.\, t_2$$

où $x \in Const$. 0 est un processus terminé (qui ne fait donc rien), $t_1 \parallel t_2$ représente la composition parallèle, et $t_1 \,.\, t_2$ représente la composition séquentielle. On se donne un ensemble fini de règles de dérivation de processus :

$$\Delta = \{x_i \xrightarrow{a_i} t_i \mid x_i \in Const,\ t_i \in T,\ a_i \in Act,\ i = 1,\ldots,n\}$$

Pour $x \in Const$ on définit $\Delta(x) = \{t \mid (x \xrightarrow{a} t) \in \Delta\}$. Pour $t \in T$, $Const(t)$ représente l'ensemble des noms de processus de $Const$ apparaissant dans t. Un processus t est terminé s'il ne contient pas de noms de processus : on définit donc le prédicat $Terminé(t) = (Const(t) = \emptyset)$.

Δ induit une relation étiquetée $\rightarrow_\Delta \subseteq T \times Act \times T$ grâce à la sémantique donnée en figure 3.1. Remarquons simplement que dans $t_1.t_2$, on peut dériver (exécuter) t_2 seulement si t_1 est terminé. On retrouve ainsi la notion classique de séquence.

$$x \xrightarrow{a} t \quad \text{si } (x \xrightarrow{a} t) \in \Delta$$

$$\frac{t_1 \xrightarrow{a} t_1'}{t_1 \parallel t_2 \xrightarrow{a} t_1' \parallel t_2} \qquad \frac{t_2 \xrightarrow{a} t_2'}{t_1 \parallel t_2 \xrightarrow{a} t_1 \parallel t_2'}$$

$$\frac{t_1 \xrightarrow{a} t_1'}{t_1 \,.\, t_2 \xrightarrow{a} t_1' \,.\, t_2} \qquad \frac{t_2 \xrightarrow{a} t_2'}{t_1 \,.\, t_2 \xrightarrow{a} t_1 \,.\, t_2'} \quad \text{si } Terminé(t_1)$$

FIG. 3.1 – *La sémantique*

Dérivation de processus

Intéressons nous maintenant à la dérivation de processus en plusieurs étapes. On considère que $\rightarrow$ est une relation binaire (les actions sont omises), et on s'intéresse à sa fermeture transitive $\rightarrow^*$.

Théorème 3.1 *[43, 42] $\rightarrow$ et $\rightarrow^*$ peuvent être générés par des CS réguliers.*

Autrement dit, ce sont des RR de couple d'arbres. Ceci avait déjà été démontré par D. Lugiez et P. Schnoebelen [75].

3.1.2 BCPC

$Act = \{a,b,c,\ldots,\overline{a},\overline{b},\overline{c},\ldots\}$, $\Delta = \Delta_1 \cup \Delta_2$ où Δ_2 est un ensemble de règles synchronisées :

$$\Delta_1 = \{x_i \xrightarrow{a_i} t_i \mid x_i \in Const,\ t_i \in T,\ a_i \in Act,\ i = 1,\ldots,n\}$$

$$\Delta_2 = \{\begin{cases} y_i \xrightarrow{a_i} t_{i,1} \\ z_i \xrightarrow{\overline{a_i}} t_{i,2} \end{cases} \mid \begin{array}{l} y_i,z_i \in Const,\ t_{i,1},t_{i,2} \in T, \\ a_i,\overline{a_i} \in Act,\ i = 1,\ldots,n \end{array} \}$$

On définit :

$$\Delta_1(x) = \{t \mid (x \xrightarrow{a} t) \in \Delta_1\} \qquad \Delta_2(y,z) = \{(t_1,t_2) \mid \begin{cases} y \xrightarrow{a} t_1 \\ z \xrightarrow{\overline{a}} t_2 \end{cases} \in \Delta_2\}$$

On se donne un ensemble fini CT (Communication Terms) de *termes de communication.* $\rightarrow_\Delta$ est maintenant induite par la sémantique de la figure 3.1 et

$$s \xrightarrow{\epsilon} s[y/t_1,z/t_2] \quad \text{si} \begin{cases} y \xrightarrow{a} t_1 \\ z \xrightarrow{\overline{a}} t_2 \end{cases} \in \Delta_2 \text{ et } s \in CT$$

où $s[y/t_1,z/t_2]$ est obtenu en remplaçant y par t_1 et z par t_2 dans s.

Exemple 3.2

$$\Delta_1 = \{x \xrightarrow{a} y\|z,\ z \xrightarrow{a} y_2,\ y_1 \xrightarrow{a} 0\},\ \Delta_2 = \begin{cases} y \xrightarrow{a} (0\ .\ y_1) \\ z \xrightarrow{\overline{a}} 0 \end{cases}$$

avec $CT = \{y\|z\}$. On a par exemple la dérivation :

$$x \xrightarrow{a} y\|z \xrightarrow{\epsilon} (0.y_1)\|0 \xrightarrow{a} (0.0)\|0$$

$\diamond$

Restrictions :

1. Aucun sous-terme strict d'un terme de communication n'est identique à un membre droit d'une règle de Δ.
2. Un terme de communication ne contient pas de composition séquentielle.

Puisqu'on ne s'occupe pas des actions, une manière de satisfaire la première restriction est de concaténer plusieurs règles de Δ en une seule afin d'obtenir des membres droits plus gros. Par contre on ne peut pas satisfaire la deuxième sans changer $\rightarrow^*$.

Théorème 3.3 *$\rightarrow$ et $\rightarrow^*$ peuvent être générés par des CS.*

$\rightarrow$ (resp. $\rightarrow^*$) n'est pas une RR, mais :

Proposition 3.4 *Les premiers composants de $\rightarrow$ et $\rightarrow^*$ sont libres*[1].

1. La liberté d'un composant d'un CS est définie de la même manière que pour une TTSG. Voir la définition formelle dans [42].

3.1.3 Model-checking

Soit $Prop = \{P_1, P_2, \ldots\}$ un ensemble de *propositions atomiques*. Pour $P \in Prop$, on note $Mod(P)$ l'ensemble des termes processus qui valident P, et on suppose que $Mod(P)$ est un langage régulier d'arbres.

Considérons la logique définie par la syntaxe suivante :

$$\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathsf{EX}_\varphi \mid \mathsf{EF}_\varphi$$

et la sémantique :

$$\begin{array}{rcl} t \models P & \Leftrightarrow & t \in Mod(P) \\ t \models \neg\varphi & \Leftrightarrow & t \not\models \varphi \\ t \models \varphi \wedge \varphi' & \Leftrightarrow & t \models \varphi \text{ et } t \models \varphi' \end{array} \qquad \begin{array}{rcl} t \models \mathsf{EX}_\varphi & \Leftrightarrow & t' \models \varphi \text{ pour un } t' \text{ t.q. } t \to t' \\ t \models \mathsf{EF}_\varphi & \Leftrightarrow & t' \models \varphi \text{ pour un } t' \text{ t.q. } t \to^* t' \end{array}$$

Définition 3.5 *Le* model-checking *consiste à tester si $t \models \varphi$ pour t et φ donnés.*

En définissant $Mod(\varphi) = \{s \in T \mid s \models \varphi\}$, le model-checking pour t et φ revient à tester si $t \in Mod(\varphi)$. Or Mod vérifie :

$$\begin{array}{rcl} Mod(\neg\varphi) & = & T - Mod(\varphi) \\ Mod(\varphi \wedge \varphi') & = & Mod(\varphi) \cap Mod(\varphi') \end{array} \qquad \begin{array}{rcl} Mod(\mathsf{EX}_\varphi) & = & Pre(Mod(\varphi)) \\ Mod(\mathsf{EF}_\varphi) & = & Pre^*(Mod(\varphi)) \end{array}$$

Lemme 3.6 *Dans BCPC, si $Mod(\varphi)$ est un langage régulier, alors $Mod(\mathsf{EX}_\varphi)$ et $Mod(\mathsf{EF}_\varphi)$ sont aussi des langages réguliers.*

Preuve : Remarquons que :

$$Mod(\mathsf{EX}_\varphi) = \Pi_1(\to \bowtie \mathsf{Mod}(\varphi)), \qquad \mathsf{Mod}(\mathsf{EF}_\varphi) = \Pi_1(\to^* \bowtie \mathsf{Mod}(\varphi))$$

Or les premiers composants de $\to$ et de $\to^*$ sont libres, et les CS sont stables par jointure et préservent la liberté des composants. La projection sur le premier composant est donc un langage régulier.

On en déduit par récurrence structurelle que pour toute formule φ, $Mod(\varphi)$ est régulier, et il suffit alors de tester si $t \in Mod(\varphi)$. D'où :

Théorème 3.7 *Le model-checking dans BCPC (et à fortiori dans PA) est décidable.*

Ce résultat a déjà été prouvé pour PA dans [28, 76]. Il est intéressant de constater que pour calculer le langage régulier $Mod(\varphi)$, il a fallu passer par des langages synchronisés ($\to$ et $\to^*$). D'où la question : peut-on prouver ce résultat en utilisant uniquement des langages réguliers ?

Corollaire 3.8 *L'absence d'interblocages est décidable en BCPC.*

Preuve : Il n'y a pas d'interblocage issu de t ssi :

$$t \models \neg\mathsf{EF}(\neg \mathit{Terminé} \wedge \neg\mathsf{EX}(\mathsf{True}))$$

où $Mod(True) = T$ et $Mod(\mathit{Terminé}) = \{t \in T \mid \mathit{Terminé}(t)\}$, qui sont bien des langages réguliers.

3.1.4 Conclusion et perspectives

La décidabilité du model-checking provient évidement du fait qu'il y a un nombre fini de termes de communication[2]. D'ailleurs, si la communication était autorisée dans n'importe quel terme, notre formalisme se rapprocherait de CCS.

Nous souhaiterions améliorer le travail actuel dans les directions suivantes :

- étendre BCPC pour permettre des rendez-vous de n processus, au lieu de 2 actuellement,
- lever la deuxième restriction, c-à-d autoriser la séquence dans les termes de communication.

Le premier point est quasiment trivial, puisque les CS peuvent tout aussi bien définir des relations non binaires. Le deuxième point est plus difficile car si on veut dériver un membre droit d'une séquence, il faut vérifier que le membre gauche est terminé. Cette vérification devrait être possible dans le CS qui reconnaît $\rightarrow^*$ (voir [42, figure 5]) en générant pas à pas chaque symbole du terme de communication s, au lieu de le générer d'un bloc comme c'est le cas actuellement.

Ni PA ni BCPC ne tiennent compte des propriétés algébriques des compositions séquentielles et parallèles (associativité, neutre 0, et commutativité de $\|$). La prise en compte de ces propriétés dans PA a été étudiée dans [78, 28]. En ce qui concerne BCPC, la difficulté est alors de donner un sens à la notion de terme de communication.

3.2 La validation des programmes logiques

Cette application a été développée par S. Limet et F. Saubion [72, 71].

M.P. Bonacina et J. Hsiang ont montré qu'on pouvait transformer un programme logique en un système de réécriture, et résoudre un but en appliquant sur le système de réécriture une procédure appelée *complétion linéaire* [16]. Il s'agit d'une extension de la procédure de complétion de Knuth-Bendix, qui est équivalente à la SLD-résolution, mais qui termine plus souvent car elle peut éviter des branches infinies non productives, et elle peut même synthétiser des ensembles infinis de réponses par un ensemble fini de règles de réécriture. La complétion linéaire a de plus été étendue à la programmation logique avec négation [3, 4] et avec contraintes [95, 96].

Quand la SLD-résolution ne termine pas, il est difficile de savoir si le programme logique est correct, c-à-d conforme à une sémantique attendue. Mais si la complétion linéaire termine et qu'elle produit un système de réécriture linéaire, on peut exprimer les solutions du but, même s'il y en a une infinité, par une TTSG de rang 1 à trans-classes bornées. Par ailleurs on demande à l'utilisateur d'exprimer la sémantique attendue, c-à-d l'ensemble des solutions prévues, par un langage régulier. Grâce au test d'inclu-

2. La taille du CS qui génère $\rightarrow^*$ est proportionelle à ce nombre.

sion entre TTSG de rang 1 à trans-classes bornées et automates finis (théorème 2.13), on peut alors vérifier que le programme logique est correct, c-à-d ne calcule que des solutions attendues.

3.3 Les relations de type réécriture

Etant donnés une relation binaire sur les termes Rel et deux langages réguliers d'arbres L_1, L_2, la décidabilité de $Rel(L_1) \subseteq L_2$ a été étudiée par F. Seynhaeve, S. Tison et M. Tommasi [101, 100] quand Rel est une relation de type réécriture (réécritures parallèle, en une passe, etc...).

Quand Rel est défini par un CS, grâce au corollaire 2.24 nous obtenons :

Proposition 3.9 *[43, 42] Si L_1,Rel sont définis par des CS et L_2 est régulier, alors*

$$\left.\begin{array}{r}L_1 \text{ est régulier} \\ \text{ou le premier composant de } Rel \text{ est libre}\end{array}\right\} \implies Rel(L_1) \subseteq L_2 \text{ est décidable}$$

Preuve : $Rel(L_1) \subseteq L_2 \Leftrightarrow Rel(L_1) \bowtie \overline{L_2} = \emptyset$

Remarquons que ceci peut être utilisé de manière incrémentale pour composer des relations. Si le premier composant de chaque relation $Rel_1, \ldots, Rel_n$ est libre, alors $Rel_1(\ldots\ldots(Rel_n(L_1)\ldots) \subseteq L_2$ est décidable.

Appliquons maintenant ce résultat aux relations de type réécriture. On peut facilement définir la réécriture en une étape, et la réécriture parallèle par des CS (voir l'exemple 2.34 et l'annexe de [42] pour le cas général). Si le système de réécriture est linéaire, on peut aussi définir la réécriture en une passe, et la réécriture en une passe par la racine par des CS. Donc si L_1 est régulier, on obtient les mêmes résultats de décidabilité que dans [101] (ou plus faible à cause de la linéarité). Par contre, si L_1 n'est pas régulier, par exemple si L_1 est formé des instances closes d'un terme non linéaire, nous obtenons des nouveaux résultats, en supposant toutefois la linéarité gauche pour les réécritures en une étape et parallèle, afin que le premier composant de Rel soit libre.

De plus, en supposant toujours la linéarité gauche, on peut composer n fois la relation de réécriture en zéro ou une étape avec elle-même, et on obtient la décidabilité de $\xrightarrow{\leq n}(L_1) \subseteq L_2$, où $\xrightarrow{\leq n}$ est la réécriture en en une nombre d'étapes inférieur ou égal à n.

3.4 La théorie de la réécriture en une étape

Etant donné une signature Σ et un système de réécriture, la théorie de la réécriture en une étape est la théorie du premier ordre sur l'univers des Σ-termes clos dont le seul symbole de prédicat est $\rightarrow$, où $x \rightarrow y$ signifie x se réécrit en y en une étape [3]. La confluence en une étape peut par exemple être exprimée dans cette théorie, par la formule :

$$\forall x,y,z\,((x \rightarrow y \wedge x \rightarrow z) \implies \exists u\,(y \rightarrow u \wedge z \rightarrow u))$$

Cette théorie est indécidable [104], même dans le cas de ses fragments $\exists^*\forall^*$ [102, 77] et $\exists^*\forall^*\exists^*$ [106]. Elle est décidable si la signature est unaire [54], dans les systèmes de réécriture linéaires dont les membres gauches et droits n'ont pas de variables communes [25] [4]. Le fragment existentiel positif est décidable [88], et tout le fragment existentiel pour les systèmes de réécriture 'quasi-shallow' [5] [17].

Dans [68] nous avons affirmé la décidabilité du fragment existentiel pour les systèmes de réécriture linéaires, non superposants (non overlappping), et non-ϵ-gauche-droite-superposants [6]. Il y a cependant un trou dans la preuve de ce résultat car on obtient une sous-classe des TTSG à inclinaison bornée, dont on ignore si le vide est décidable. L'utilisation des WRR ne permet pas de résoudre le problème car la mise en forme horizontale (théorème 2.15) d'une TTSG à inclinaison bornée n'est pas toujours une WRR. Cependant, les WRR nous ont permis d'obtenir un résultat un peu différent, sous une hypothèse d'horizontalité.

Définition 3.10 *Un système de réécriture R est* ϵ-gauche-droite-collisionnant *si pour toute règle $l \rightarrow r$, $l(\epsilon) \neq r(\epsilon)$. R est* faiblement horizontal *si dans chaque règle de réécriture, toutes les occurrences d'une même variable sont à la même profondeur.*

Remarquons qu'un système ϵ-gauche-droite-collisionnant et faiblement horizontal est non-ϵ-gauche-droite-superposant. De plus il ne possède pas de règles effondrantes (collapsing rules).

Théorème 3.11 *[70, 69] Le fragment existentiel de la théorie en une étape est décidable pour les systèmes de réécriture faiblement horizontaux ϵ-gauche-droite-collisionnants.*

Puisque la profondeur faible (quasi-shallowness) est un cas particulier de l'horizontalité faible, ce résultat étend celui de [17], sauf qu'on suppose en plus la collision ϵ-gauche-droite. Remarquons d'autre part que les règles de réécriture peuvent être non linéaires et qu'il peut y avoir des paires critiques.

Notre procédure de décision fonctionne de la manière suivante. Considérons un système de réécriture $R = \{ru_1, \ldots, ru_n\}$ et une formule existentielle en forme prénexe :

1. Puisque les symboles de Σ ne sont pas autorisés dans les formules, les atomes sont de la forme $x \rightarrow x$ ou $x \rightarrow y$. Du fait de l'hypothèse de collision ϵ-gauche-droite, $x \rightarrow x$ n'a pas de solutions et est remplacé par le prédicat sans solutions $\perp$, et $x \rightarrow y$ est remplacé par la proposition équivalente $(x \overset{?}{\rightarrow}_{[ru_1]} y \vee \ldots \vee x \overset{?}{\rightarrow}_{[ru_n]} y) \wedge x \neq y$,

3. Les symboles de Σ ne sont pas autorisés dans les formules.

4. Dans ce cas, même la théorie de la réécriture en plusieurs étapes est décidable.

5. Dans les règles de réécriture, les variables sont toutes à la profondeur 1.

6. c-à-d aucun membre gauche ne se superpose en tête avec le membre droit correspondant.

où $\overset{?}{\rightarrow}_{[ru_i]}$ est la réécriture par la règle ru_i en zéro ou une étape. Puis la formule est transformée en une disjonction de conjonctions (éventuellement de négations) d'atomes de la forme $x \overset{?}{\rightarrow}_{[ru_i]} y$ ou $x = y$. Or les ensembles de solutions SOL de ces atomes sont des WRR, car $SOL(x = y)$ et $SOL(x \neq y)$ sont des RR, et :

Lemme 3.12
Sous les hypothèses du théorème 3.11, $SOL(x \overset{?}{\rightarrow}_{[ru_i]} y)$ *et* $SOL(\neg(x \overset{?}{\rightarrow}_{[ru_i]} y))$ *sont des WRR.*

$SOL(x \overset{?}{\rightarrow}_{[ru_i]} y)$ est une WRR grâce à l'horizontalité faible (voir l'exemple 2.34), et on peut calculer son complément car la collision ϵ-gauche-droite est une sorte de déterminisme.

2. Les solutions d'un facteur conjonctif sont obtenues en faisant des produits cartésiens avec l'ensemble T_Σ des termes clos (qui est une RR), et des intersections. Par exemple soit

$$C = \; x \overset{?}{\rightarrow}_{[ru_1]} y \wedge \neg(y \overset{?}{\rightarrow}_{[ru_2]} z)$$

D'après le lemme précédent, les solutions des littéraux sont des WRR de couples, et on peut calculer

$$SOL(C) = \;\; SOL(x \overset{?}{\rightarrow}_{[ru_1]} y)) \times T_\Sigma \;\; \cap \;\; T_\Sigma \times SOL(\neg(y \overset{?}{\rightarrow}_{[ru_2]} z))$$

qui est encore une WRR (de triplets).

3. La validité de la formule complète est testée en appliquant le test du vide des WRR sur chaque facteur conjonctif.

L'unification de contextes est un cas particulier de l'unification du second ordre, et il y a un lien entre ce problème et la réécriture en une étape, puisqu'il existe des termes x,y t.q. $x \rightarrow_{[l \rightarrow r]} y$ ssi $X(l) \doteq X(r)$ admet des solutions (X est une variable du second ordre).

On peut alors se demander si le résultat que nous avons obtenu sur la réécriture en une étape a des retombées sur la disunification de contextes, par le fait que nous pouvons traiter la négation. Une équivalence précise entre l'unification de contextes stratifiée et une petite extension du fragment positif de la réécriture en une étape [89] a été établie. Malheureusement, déjà dans ce cas, le système de réécriture ainsi obtenu contient des règles effondrantes, ce qui n'est pas compatible avec nos restrictions.

Chapitre 4

L'unification équationnelle

L'unification équationnelle a déjà été beaucoup étudiée. Une synthèse des résultats existants est donnée dans [9]. Dans ce chapitre, nous nous limitons au cas de l'unification modulo un système de réécriture.

4.1 En exprimant les solutions par un langage d'arbres

Dans cette section, nous abordons le problème de l'unification modulo un système de réécriture confluent et basé sur les constructeurs. Le but est de décider l'existence d'unificateurs (unifiabilité), et de les exprimer par un langage de n-uplets d'arbres. Nous nous intéresserons de plus en sous-section 4.1.3 à la résolution de diséquations (c-à-d de buts de la forme $s \neq s'$).

Comme dans les langages de programmation logico-fonctionnels, nous considérons la notion d'unificateur selon une *sémantique de constructeurs* (appelée égalité stricte dans [86, 7]), c-à-d la substitution close et ne contenant que des constructeurs σ est une *solution* de l'équation $s \doteq s'$ s'il existe un terme-constructeur t tel que $\sigma s \rightarrow^* t$ and $\sigma s' \rightarrow^* t$. Dans le même esprit, σ est une *solution* de la diséquation $s \not\doteq s'$ s'il existe des termes-constructeur t,t' tels que $\sigma s \rightarrow^* t$ and $\sigma s' \rightarrow^* t'$ and $t \neq t'$. Le système de réécriture étant supposé confluent et basé sur les constructeurs, t et t' sont uniques (s'ils existent), donc σ ne peut pas être à la fois solution de $s \doteq s'$ et $s \not\doteq s'$. Selon cette sémantique, l'équation $1/x \div 1/0$ où $\div$ est la division euclidienne sur les entiers, n'a pas de solutions.

Nous avons établi un résultat de décidabilité de l'unifiabilité en utilisant les TTSG [64, 65]. Puis la méthode a été généralisée par Sébastien Limet et Frédéric Saubion [73] : tout langage de n-uplet d'arbres peut être utilisé, si

1. il est stable par jointure,
2. le vide est décidable,
3. pour toute fonction définie f, il peut exprimer l'ensemble de n-uplets

$N_f = \{(r,\sigma x_1,\ldots,\sigma x_n) \mid f(x_1,\ldots,x_n) \rightsquigarrow^*_{[\sigma]} r$ et $r,\sigma x_1,\ldots,\sigma x_n$ sont des termes-constructeur clos$\}$,

4. pour tout constructeur c, il peut exprimer l'ensemble de n-uplets

$$N_c = \{(c(t_1,\ldots,t_n),t_1,\ldots,t_n) \mid t_i \text{ sont des termes-constructeur clos}\}$$

S'il est en plus stable par intersection et projection, le but à unifier peut être non linéaire.

Ainsi, le travail utilisant les TTSG est une instance de cette généralisation, et [73] présente aussi une autre instance utilisant les grammaires primales [51]. Plus tard, nous avons développé une troisième instance utilisant les WRR [69]. Pour que le point 3. soit vérifié, il faut des restrictions sur le système de réécriture, qui dépendent du langage d'arbres utilisé. Chaque instance fournit donc un résultat théorique différent.

Je vais d'abord présenter la méthode générale à l'aide d'un exemple, puis les deux instances sur lesquelles j'ai travaillé.

Exemple 4.1

$$R = \{f(s(x)) \xrightarrow{1} p(f(x)),\ \ f(p(x)) \xrightarrow{2} s(f(x)),\ \ f(0) \xrightarrow{3} 0\}$$

où s,p,0 sont des constructeurs, et considérons le but linéaire $p(f(x)) \doteq f(s(f(x')))$. Supposons que nous disposons d'un langage de n-uplets d'arbres qui satisfait les quatre propriétés précédentes. Il peut donc exprimer N_f, N_s, N_p.

La méthode consiste à simuler les dérivations de surréduction 'innermost' issues du but. On calcule :

$$\begin{array}{rcl} N = N_p \bowtie N_f & = & \{(r_1,r_2,t) \mid (r_1,r_2) \in N_p \wedge (r_2,t) \in N_f\} \\ & = & \{(r_1,r_2,t) \mid p(f(x)) \rightsquigarrow^*_{[x/t]} p(r_2) = r_1\} \end{array}$$

$$\begin{array}{rcl} N' & = & N_f \bowtie N_s \bowtie N_f \\ & = & \{(r'_1,r'_2,r'_3,t') \mid (r'_1,r'_2) \in N_f \wedge (r'_2,r'_3) \in N_s \wedge (r'_3,t') \in N_f\} \\ & = & \{(r'_1,r'_2,r'_3,t') \mid f(s(f(x'))) \rightsquigarrow^*_{[x'/t']} f(s(r'_3)) = f(r'_2) \rightarrow^* r'_1\} \end{array}$$

D'après la sémantique de constructeurs et les propriétés de la surréduction [36], le but admet des solutions ssi il existe des instances t, t' telles que $r_1 = r'_1$, c-à-d telles que $N \bowtie_{1,1} N' \neq \emptyset$. De plus $N \bowtie_{1,1} N'$ exprime les solutions grâce à t, t'.

Maintenant, si le but à unifier n'est pas linéaire, comme $p(f(x)) \doteq f(s(f(x)))$, t doit en plus être égal à t'. Par projection, on ne garde que t, t', r_1, r'_1, et on force les égalités par intersection : le but admet des solutions ssi $\Pi_{1,3}(N) \cap \Pi_{1,4}(N') \neq \emptyset$.

Faisons ce test sur l'exemple. Pour toute étape de surréduction apparaissant dans une dérivation de surréduction $f(x) \rightsquigarrow^*_{[\sigma]} r$ où r est un terme-constructeur clos, ou bien s est ajouté dans σ et p dans r (par la règle 1), ou bien c'est le contraire (par la règle 2), et la dérivation se termine en ajoutant 0 dans les deux (par la règle 3). Donc N_f est donné par l'expression :

$$N_f \equiv ((p,s) \cup (s,p))^*.(0,0)$$

Par ailleurs

$$N_s = \{(s(t),t)\} \equiv (s,\epsilon).((s,s) \cup (p,p))^*.(0,0)$$
$$N_p = \{(p(t),t)\} \equiv (p,\epsilon).((s,s) \cup (p,p))^*.(0,0)$$

où ϵ est la chaîne vide. Par conséquent,

$$\begin{array}{rcl} N & = & (p,\epsilon,\epsilon).((s,s,p) \cup (p,p,s))^*.(0,0,0) \\ N_f \bowtie N_s & = & (p,s,\epsilon).((p,s,s) \cup (s,p,p))^*.(0,0,0) \\ N' = (N_f \bowtie N_s) \bowtie N_f & = & (p,s,\epsilon,\epsilon).((p,s,s,p) \cup (s,p,p,s))^*.(0,0,0,0) \end{array}$$

$$\begin{array}{rl} \Pi_{1,3}(N) \cap \Pi_{1,4}(N') & = (p,\epsilon).((s,p) \cup (p,s))^*.(0,0) \bigcap (p,\epsilon).((p,p) \cup (s,s))^*.(0,0) \\ & = (p(0),0) \end{array}$$

Les solutions sont données par le second composant. Il n'y en a qu'une : $x/0$.

C'est exact : si x/t est un unificateur de $p(f(x)) \doteq f(s(f(x)))$ ne contenant que des constructeurs, forcément $p(f(t)) = p(f(f(t)))$, donc $f(t) = f(f(t))$, donc $f(t) = t$ car $f(f(t)) = t$. Par conséquent $t = 0$. $\diamond$

4.1.1 Avec les TTSG à trans-classes bornées

Pour tout constructeur $c \in C$, N_c est généré par la TTSG de rang 1 suivante :

$$X \Rightarrow c(X_1, \ldots, X_n)$$
$$\left\{\begin{array}{l} X_i \Rightarrow X^d \\ Y_i \Rightarrow Y^d \end{array}\right. \quad X^d \Rightarrow d(X_1, \ldots, X_{ar(d)}) \quad Y^d \Rightarrow d(Y_1, \ldots, Y_{ar(d)}) \quad \left| \begin{array}{l} \forall d \in C, \\ \forall i \in \{1, \ldots, armax\} \end{array}\right.$$

où n est l'arité de c et $armax$ est l'arité des constructeurs la plus grande. L'axiome est $(X,Y_1, \ldots, Y_n)$.

Exemple 4.2 Avec $C = \{c^{\backslash 2}, a^{\backslash 0}, b^{\backslash 0}\}$, on peut dériver (le contrôle est écrit au dessus des non-terminaux) :

$$\begin{array}{l} (\overset{0}{X}, \overset{0}{Y_1}, \overset{0}{Y_2}) \Rightarrow (c(\overset{0}{X_1}, \overset{0}{X_2}), \overset{0}{Y_1}, \overset{0}{Y_2}) \Rightarrow (c(\overset{1}{X^c}, \overset{0}{X_2}), \overset{1}{Y^c}, \overset{0}{Y_2}) \\ \Rightarrow (c(c(\overset{1}{X_1}, \overset{1}{X_2}), \overset{0}{X_2}), c(\overset{1}{Y_1}, \overset{1}{Y_2}), \overset{0}{Y_2}) \Rightarrow^* (c(c(\overset{2}{X^a}, \overset{3}{X^b}), \overset{4}{X^a}), c(\overset{2}{Y^a}, \overset{3}{Y^b}), \overset{4}{Y^a}) \\ \Rightarrow^* (c(c(a,b), a), c(a,b), a) \end{array}$$

$\diamond$

Théorème 4.3 *[64, 65], [63, théorème 6.5] Pour toute fonction définie f, N_f est généré par une TTSG de rang 1, sous les conditions suivantes :*

1. *le système de réécriture est totalement linéaire,*
2. *dans les membres droits, les arguments des fonctions définies sont des variables[1].*

1. Cette condition peut être affaiblie en des conditions plus techniques : pas de fonctions imbriquées et pas de σ_{in}.

De plus le premier composant de N_f est libre[2].

Exemple 4.4

$$R = \{f(s(x)) \xrightarrow{1} p(f(x)),\ f(a) \xrightarrow{2} a\}$$

A chaque position des membres des règles de réécriture, on associe un non-terminal (écrit au-dessus des membres des règles) :

$$R = \{ \overset{R^1_\epsilon}{f}\ (\overset{L^1_1}{s}\ (\overset{X^1}{x})) \xrightarrow{1} \overset{R^1_\epsilon}{p}\ (\overset{R^1_1}{f}\ (\overset{X^1}{x})),\quad \overset{R^2_\epsilon}{f}\ (\overset{L^2_1}{a}) \xrightarrow{2} \overset{R^2_\epsilon}{a} \}$$

Les unifications syntaxiques de la surréduction sont simulées par des paquets de productions synchronisées, alors que les descentes dans les sous-termes le sont par des productions libres. La TTSG générant N_f est :

$$\begin{cases} R^1_1 \Rightarrow R^1_\epsilon \\ X^1 \Rightarrow L^1_1 \end{cases},\quad \begin{cases} R^1_1 \Rightarrow R^2_\epsilon \\ X^1 \Rightarrow L^2_1 \end{cases},\quad L^1_1 \Rightarrow s(X^1),\ \ R^1_\epsilon \Rightarrow p(R^1_1),\ \ L^2_1 \Rightarrow a,\ \ R^2_\epsilon \Rightarrow a$$

L'axiome[3] est (R^1_1,X^1). Cette grammaire génère bien $N_f = \{p^n(a), s^n(a)\}$. $\diamond$

Le vide des TTSG à trans-classes bornées est décidable. Par contre elles ne sont pas stables par intersection, ce qui empêche l'unification de buts non linéaires. Nous supposons donc que le but est linéaire ($f(x) \doteq x$ est considéré comme non linéaire).

La stabilité par i,j-jointure n'est garantie que si un des composants i ou j est libre. Or le premier composant de N_f l'est, et les deux composants de N_c le sont trivialement. De plus :

Lemme 4.5 *([63, lemme 7.11] redémontré dans [42, théorème 3.10]) Si L_1, L_2 sont des langages générés par des TTSG à trans-classes bornées, et si le k-ème composant de L_1 et le premier composant de L_2 sont libres, alors le k-ème composant de $L_1 \bowtie L_2$ est encore libre.*

Revenons à l'exemple 4.1. En appliquant plusieurs fois ce lemme (avec $k = 1$), on en déduit que $N \bowtie_{1,1} N' = (N_p \bowtie N_f) \bowtie (N_f \bowtie (N_s \bowtie N_f))$ est généré par une TTSG à trans-classes bornées. Ceci se généralise, et grâce à la décidabilité du vide on obtient :

Théorème 4.6 *L'unifiabilité des buts linéaires est décidable modulo les systèmes de réécriture confluents, basés sur les constructeurs, linéaires, et tels que dans les membres droits les arguments de fonctions sont des variables. De plus les solutions peuvent être exprimées par une TTSG à trans-classes bornées.*

Ces restrictions peuvent paraître fortes. Toutefois, si l'une d'elle n'est pas satisfaite l'unifiabilité est indécidable.

Théorème 4.7 *L'unifiabilité modulo un système de réécriture confluent et basé sur les constructeurs est indécidable si une des trois conditions suivantes n'est pas satisfaite :*

- *le but est linéaire,*

2. Car les membres droits sont linéaires.
3. Le cas général est légèrement plus compliqué.

- *le système de réécriture est linéaire droit,*
- *dans les membres droits, les arguments de fonction sont des variables.*

Ce résultat est prouvé par codage du problème de Post. On remarquera que la linéarité gauche n'est pas mentionnée. Cela provient du fait que le théorème 4.6 est encore vrai même si les membres gauches ne sont pas linéaires. Mais ceci sera prouvé par une autre méthode : un calcul de descendants (voir section 4.2).

4.1.2 Avec les WRR

Les résultats présentés ici proviennent de [69]. Les WRR sont stables par intersection et projection, et le vide est décidable. Cependant, il va falloir modifier un peu la méthode générale car N_c n'est pas horizontal.

$\boldsymbol{N_c}$ **:** $N_c = \{(c(t_1,\dots,t_n),t_1,\dots,t_n)\}$ ne peut pas s'exprimer par une WRR (ni d'ailleurs par une RR), car il faut une synchronisation entre les deux exemplaires de t_i, qui ne sont pas à la même profondeur. N_c n'est donc pas horizontal.

Modifions un peu la méthode générale pour qu'il le soit. En utilisant un symbole supplémentaire $\natural$, nous définissons $N_c = \{(c(t_1,\dots,t_n),\natural t_1,\dots,\natural t_n)\}$ pour tout constructeur c et $\natural L = \{(\natural t_1,\dots,\natural t_n) \mid (t_1,\dots,t_n) \in L\}$ pour tout langage L. Ils sont générés par les systèmes de contraintes :

$$\begin{array}{rcl} N_c & \supseteq & (c(1_1,\dots,n_1),\natural 1_2,\dots,\natural n_2)(Id_2,\dots,Id_2) \\ \natural L & \supseteq & (\natural 1_1,\dots,\natural 1_n)(L) \end{array}$$

où $Id_2 = \{(t,t) \mid t \in T_C\}$ et T_C est l'ensemble des termes-constructeur. Ces contraintes ne sont pas récursives et Id_2 est régulier. Donc N_c est une WRR et $\natural L$ est une WRR si L l'est.

La modification de la définition de N_c impose aussi des modifications dans les calculs.

Exemple 4.8 Considérons de nouveau l'exemple 4.1. Maintenant N et N' sont calculés par :

$$\begin{array}{rcl} N = N_p \bowtie \natural N_f & = & \{(r_1,\natural r_2,\natural t) \mid (r_1,\natural r_2) \in N_p \wedge (\natural r_2,\natural t) \in \natural N_f\} \\ & = & \{(r_1,\natural r_2,\natural t) \mid p(f(x)) \rightsquigarrow^*_{[x/t]} p(r_2) = r_1\} \end{array}$$

$$\begin{array}{rcl} N' & = & N_f \bowtie N_s \bowtie \natural N_f \\ & = & \{(r_1',r_2',\natural r_3',\natural t') \mid (r_1',r_2') \in N_f \wedge (r_2',\natural r_3') \in N_s \wedge (\natural r_3',\natural t') \in \natural N_f\} \\ & = & \{(r_1',r_2',\natural r_3',\natural t') \mid f(s(f(x'))) \rightsquigarrow^*_{[x'/t']} f(s(r_3')) = f(r_2') \rightarrow^* r_1'\} \end{array}$$

Si le but est linéaire, on vérifie $N \bowtie_{1,1} N' \neq \emptyset$ comme dans la méthode générale. Sinon on peut toujours imposer que $t = t'$ en calculant $\Pi_{1,3}(N) \cap \Pi_{1,4}(N')$, car le nombre de $\natural$ au-dessus de t dans N et de t' dans N' est le même (un). Ceci est dû au fait que le nombre de constructeurs apparaissant au-dessus des deux occurrences de x dans le but est le même (un). $\diamond$

Définition 4.9 *Soient t un terme et $u \in Pos(t)$, et soit $\|u\|$ le nombre de constructeurs apparaissant dans t au-dessus de u. Le terme t est* faiblement horizontal *si*

$$\forall u,u' \in Pos(t),\, t(u) = t(u') = x \in Var(t) \implies \|u\| = \|u'\|$$

Dans ce cas, nous définissons aussi la notation $\|x\| \stackrel{def}{=} \|u\|$. Le but $t \doteq t'$ (resp. la règle de réécriture $l \to r$) est faiblement horizontale *si $t \doteq t'$ (resp. $l \to r$) est un terme faiblement horizontal, en considérant que $\doteq$ (resp. $\to$) est un symbole binaire.*

Ainsi, si le but est faiblement horizontal la méthode modifiée marche, à condition qu'on puisse exprimer N_f.

$\boldsymbol{N_f}$ **:** Voici comment un système de réécriture peut être transformé en un système de contraintes qui exprime N_f. Bien que technique, l'algorithme de transformation doit être donné ici car il intervient ensuite dans la définition 4.14. Etudions d'abord un exemple.

Exemple 4.10 Soit la règle de réécriture $f(c(x,y)) \to d(f(y),x)$. Par surréduction on a $f(x) \rightsquigarrow_{[\sigma=x/c(x,y)]} t' = d(f(y),x)$. Ainsi $(t',\sigma) = (d(f(y),x), c(x,y))$. Pour obtenir des termes-constructeur clos, x doit être instanciée par un terme clos et $f(y)$ doit être surréduit encore. La contrainte est donc :

$$F \supseteq (d(1_1,2_1), c(2_2,1_2))(F,Id_2)$$

◇

Définition 4.11 *Etant donné un système de réécriture R, un préordre sur les fonctions définies par $f > g$ si $f(t_1,\ldots,t_n) \to r \in R$ et g apparaît dans r. $f \equiv g$ signifie $f > g \wedge g > f$. La règle de réécriture $f(t_1,\ldots,t_n) \to r \in R$ est* récursive *s'il existe une fonction g dans r telle que $f \equiv g$.*

Algorithme : supposons que dans les membres droits, les arguments de fonctions sont des variables, et que les règles récursives sont linéaires. Notons les règles de réécriture sous la forme $f(t_1,\ldots,t_n) \to C[u_1,\ldots,u_m]$ où C est un contexte ne contenant que des constructeurs (et donc pas de variables), et chaque u_i est soit une variable, soit un appel de fonction de la forme $f_i(x_1^i,\ldots,x_{k_i}^i)$. Pour chaque règle, on crée la contrainte :

$$F \supseteq (C[1_1,\ldots,m_1],\theta t_1,\ldots,\theta t_n)(X_1,\ldots,X_m)$$

où $X_i = Id_2$ si u_i est une variable et $X_i = F_i$ si $u_i = f_i(x_1^i,\ldots,x_{k_i}^i)$, et θ est la substitution $\{u_i/i_2 \mid u_i \text{ est une variable}\}$.

Si cette règle de réécriture n'est ni linéaire ni récursive, les F_i's sont définis indépendamment de F. On peut alors forcer l'égalité de certaines variables en faisant l'intersection[4] des arguments de la contrainte $(X_1,\ldots,X_n)$ avec la relation régulière

$$\{(s_1,\ldots s_k,t,s_{k+2},\ldots s_p,t,s_{p+2},\ldots,s_q) | s_i,t \in T_C\}$$

4. si un des F_i's n'est pas une WRR, l'algorithme échoue.

Toutefois, le système de contraintes obtenu n'est pas forcément une WRR : dans l'exemple précédent, la contrainte est récursive et non régulière. La non-régularité provient du fait qu'en parcourant les feuilles de gauche à droite, on obtient x, y pour le membre gauche, et y, x pour le membre droit : il y a une permutation de variables. Si on supprime cette permutation en considérant la règle $f(c(x,y)) \rightarrow d(x,f(y))$, la contrainte obtenue est alors régulière. Mais l'absence de permutation de variables ne garantie pas la régularité.

Exemple 4.12 Soit $f(a, c(x,y)) \rightarrow s(f(x,y))$.
La contrainte obtenue $F \supseteq (s(1_1),a,c(1_2,1_3))\, F$ n'est pas régulière car il y a une synchronisation interne dans le troisième composant. $\diamond$

De plus la présence de permutations ne garantit pas la non-régularité.

Exemple 4.13 Soit $f(c(x,y), s(z)) \rightarrow c(f(x,z), y)$.
La contrainte obtenue $F \supseteq (c(1_1,2_1), c(1_2,2_2), s(1_3))(F,Id_2)$ est régulière. $\diamond$

C'est pourquoi nous introduisons la définition suivante :

Définition 4.14 *Un système de réécriture basé sur les constructeurs est* faiblement régulier *si les arguments des fonctions des membres droits sont des variables, les règles récursives sont linéaires, et le système de contraintes obtenu par l'algorithme précédent est une WRR.*

Remarquons que la régularité faible implique en particulier l'horizontalité faible des règles de réécriture.

On obtient donc :

Théorème 4.15 *L'unifiabilité des buts faiblement horizontaux est décidable modulo les systèmes de réécriture confluents, basés sur les constructeurs, et faiblement réguliers. De plus les unificateurs peuvent être exprimés par une WRR.*

L'avantage de ce résultat par rapport au précédent (avec les TTSG) est que le but peut être non linéaire.

4.1.3 Résoudre les diséquations avec les TTSG

Les TTSG à trans-classes bornées permettent aussi de résoudre les diséquations, sous les mêmes hypothèses que pour les équations.

Définition 4.16 *L'*anti-jointure *des langages de n-uplets L_1, L_2 sur les composants i,j est définie par :*

$$L_1 \bowtie_{i,j} L_2 \;=\; \{(tp_1\backslash_i).(tp_2\backslash_j) \;\mid\; tp_1 \in L_1 \text{ et } tp_2 \in L_2 \text{ et } tp_1|_i \neq tp_2|_j\}$$

où $tp_1\backslash_i$ est le n-uplet tp_1 dont on a supprimé le i-ème composant, et . est la concaténation.

Exemple 4.17
Si $L_1 = \{(1,a),(0,b)\}$ et $L_2 = \{(0,a),(1,c)\}$ alors $L_1 \overline{\bowtie}_{1,1} L_2 = \{(a,a),(b,c)\}$. ◇

Remarquons que si L_1 est composé de n-uplets de taille n_1 (n_2 pour L_2), alors les n-uplets de $L_1 \overline{\bowtie}_{i,j} L_2$ sont de taille $n_1 + n_2 - 2$, alors que ceux de $L_1 \bowtie_{i,j} L_2$ sont de taille $n_1 + n_2 - 1$. Ceci est dû au fait que $L_1 \bowtie_{i,j} L_2$ conserve le composant de jointure, alors que $L_1 \overline{\bowtie}_{i,j} L_2$ ne le fait pas.

Revenons à la méthode générale d'unification à l'aide des langages d'arbres.

Exemple 4.18 Considérons de nouveau l'exemple 4.1. On avait

$$\begin{array}{rcl} N & = & \{(r_1,r_2,t) \mid p(f(x)) \rightsquigarrow^*_{[x/t]} p(r_2) = r_1\} \\ N' & = & \{(r'_1,r'_2,r'_3,t') \mid f(s(f(x'))) \rightsquigarrow^*_{[x'/t']} f(s(r'_3)) = f(r'_2) \to^* r'_1\} \end{array}$$

D'après la sémantique de constructeurs et les propriétés de la surréduction [36], la diséquation $p(f(x)) \doteq f(s(f(x')))$ admet des solutions ssi il existe des instances t, t' telles que $r_1 \neq r'_1$, c-à-d telles que $N \overline{\bowtie}_{1,1} N' \neq \emptyset$. De plus $N \overline{\bowtie}_{1,1} N'$ exprime les solutions grâce à t, t'. ◇

Or :

Théorème 4.19 *[67] Les TTSG à trans-classes bornées sont stables par anti-jointure.*

L'algorithme d'anti-jointure est en fait plus compliqué que l'algorithme de jointure. Il est en effet nécessaire de rajouter un composant supplémentaire à $L_1 \overline{\bowtie}_{i,j} L_2$ (composant d'anti-jointure), dans lequel on génère à la fois $tp_1|_i$ et $tp_2|_j$ (notations de la définition précédente) afin de conserver les synchronisations de L_1 entre le i-ème composant et les autres (même chose pour L_2), et on impose que sur une branche arbitraire et à une profondeur arbitraire, il y ait une collision (symboles différents) entre $tp_1|_i$ et $tp_2|_j$.

Par conséquent :

Théorème 4.20 *La satisfiabilité des diséquations linéaires est décidable modulo les systèmes de réécriture confluents, basés sur les constructeurs, linéaires, et tels que dans les membres droits les arguments de fonctions sont des variables. De plus les solutions peuvent être exprimées par une TTSG à trans-classes bornées.*

Que signifie ce résultat, si on tient compte de la sémantique ?

Définition 4.21 *Un système de réécriture R confluent et basé sur les constructeurs est* suffisamment complet *si tout terme clos peut se réécrire en un terme-constructeur.*

Pour un système de réécriture quelconque, la notion de complétude suffisante est définie par $\forall t \in T_\Sigma, \exists t' \in T_C \mid t =_R t'$, et c'est une propriété indécidable [44]. Mais dans le cas d'un système confluent et basé sur les constructeurs, c'est bien équivalent à la définition ci-dessus. Si le système est de plus terminant, cette propriété est décidable en testant la réductibilité inductive de $f(x_1,\ldots,x_n)$ pour toute fonction définie f.

Lemme 4.22 *[67] Supposons que R est suffisamment complet. La diséquation $s \neq s'$ a des solutions ssi il existe une substitution close θ telle que $\theta s \neq_R \theta s'$.*

Remarquons que θ peut contenir des fonctions définies. Considérons maintenant le raisonnement suivant :

$$
\begin{array}{ll}
& s \neq s' \text{ n'a pas de solutions} \\
\Longleftrightarrow & \text{non}(s \neq s' \text{ a des solutions}) \\
\Longleftrightarrow & \text{non(il existe une substitution close } \theta \text{ telle que } \theta s \neq_R \theta s') \\
\Longleftrightarrow & \text{pour toute substitution close } \theta,\ \theta s =_R \theta s' \\
\Longleftrightarrow & s \doteq s' \text{ est un théorème inductif.}
\end{array}
$$

On obtient donc ainsi un test de décision des théorèmes inductifs linéaires.

4.1.4 Perspectives

L'hypothèse de linéarité du but, nécessaire quand on utilise les TTSG, réduit un peu l'intérêt des résultats obtenus. Par contre cette restriction n'existe pas quand on utilise les WRR ou les grammaires primales. D'où l'idée d'étendre ces deux instances à la résolution de diséquations non nécessairement linéaires, ce qui permettrait de prouver des théorèmes inductifs non linéaires.

Exemple 4.23 Considérons encore l'exemple 4.1. On avait

$$
\begin{array}{rcl}
N & = & \{(r_1,r_2,t) \mid p(f(x)) \rightsquigarrow^*_{[x/t]} p(r_2) = r_1\} \\
N' & = & \{(r'_1,r'_2,r'_3,t') \mid f(s(f(x'))) \rightsquigarrow^*_{[x'/t']} f(s(r'_3)) = f(r'_2) \rightarrow^* r'_1\}
\end{array}
$$

La diséquation non linéaire $p(f(x)) \neq f(s(f(x)))$ admet des solutions ssi il existe des instances t, t' telles que $r_1 \neq r'_1$ et $t = t'$. Pour forcer $t = t'$ dans le langage $N \overline{\bowtie}_{1,1} N' = \{(r_2,t,r'_2,r'_3,t')\}$, il suffit de faire l'intersection avec l'ensemble $L = \{(s_1,u,s_2,s_3,u) \mid s_1,s_2,s_3,u \in T_C\}$. Autrement dit, on teste si $(N \overline{\bowtie}_{1,1} N') \cap L \neq \emptyset$. $\diamond$

Ainsi, si le langage d'arbres dont on dispose est, en plus des autres propriétés, stable par anti-jointure et peut exprimer L, on peut décider la satisfiabilité des diséquations, et donc les théorèmes inductifs dans les systèmes de réécriture suffisamment complets.

Or on peut s'apercevoir qu'une classe de langages stable par jointure et projection, et qui peut exprimer l'ensemble $\overline{Id_2} = \{(t,t') \in T^2_\Sigma \mid t \neq t'\}$ est forcément stable par anti-jointure, car si n est la taille des n-uplets de L_1 :

$$L_1 \overline{\bowtie}_{i,j} L_2 = \Pi_{\{k \mid k \notin \{i,j+n\}\}}(L_1 \bowtie_{i,1} \overline{Id_2} \bowtie_{2,j} L_2)$$

Cette remarque s'applique aux TTSG à trans-classes bornées, en supposant qu'un des composants i ou j est libre, et par le fait que les deux composants de $\overline{Id_2}$ sont libres. On obtient ainsi une nouvelle preuve plus simple du résultat de [67]. Elle s'applique aussi aux WRR, car $\overline{Id_2}$ est une RR, donc une WRR. On obtient ainsi un nouveau résultat de décision des théorèmes inductifs. Par contre elle ne s'applique probablement pas aux grammaires primales, car T_Σ n'étant pas exprimable par une grammaire primale, il est peu probable que $\overline{Id_2}$ le soit.

4.2 A l'aide d'un langage régulier de descendants

Définition 4.24 *Soit R un système de réécriture et t un terme clos. t' est un* descendant *ou* successeur *de t si $t \rightarrow_R^* t'$.*

Soit E un ensemble de termes clos. $R^*(E)$ dénote l'ensemble des descendants des éléments de E, et $R^!(E)$ est l'ensemble des formes normales, c-à-d $R^!(E) = R^*(E) \cap IRR(R)$ où $IRR(R)$ est l'ensemble des termes non réductibles par R.

Si t n'est pas clos, $\{t\theta\}$ dénote l'ensemble des instances closes de t.

Supposons que R est confluent et considérons deux termes t, t' tels que $Var(t) \cap Var(t') = \emptyset$. Alors d'après la propriété de Church-Rosser, t et t' sont unifiables modulo R ssi il existe σ, σ' tels que $t\sigma$ et $t'\sigma'$ se réécrivent en un même terme, c-à-d $R^*(\{t\theta\}) \cap R^*(\{t'\theta\}) \neq \emptyset$. On peut donc décider l'unifiabilité de deux termes sans variables communes par un calcul de descendants.

A cause des applications possibles en déduction automatique et en validation de programme, le problème d'exprimer $R^*(E)$ à l'aide d'un automate fini d'arbres, et plus généralement la classe d'équivalence de E modulo une théorie équationnelle, a déjà été étudié dans un certain nombre de cas [24, 30, 97, 27, 55] [5]. A quelques détails près, tous ces travaux supposent que les membres droits (les deux membres quand on travaille avec des règles non orientées) des règles de réécriture sont 'shallow', c-à-d que les variables apparaissent à la profondeur 0 ou 1.

Par ailleurs, T. Genet [38] a donné une méthode pour exprimer un sur-ensemble de $R^*(E)$ par un automate fini, en supposant uniquement la linéarité gauche. Cette restriction a ensuite été supprimée [39], ce qui a permis de prouver la sûreté de la version corrigée [74] du protocole NSPK (Needham-Schroeder Public Key).

J'ai proposé [94] un calcul exact de descendants à l'aide d'automates finis, qui permet des membres droits non 'shallow'. Les restrictions restent bien sûr fortes car en général $R^*(E)$ n'est pas régulier même si E l'est. On suppose que le système de réécriture est basé sur les constructeurs, et :

1. Pour toute règle $l \rightarrow r$ et toute position de fonction définie p dans r, $r|_p = f(r_1, \ldots, r_n)$ où pour tout i, r_i est une variable ou un terme clos [6].
2. Les membres droits sont linéaires et ne contiennent pas de fonctions imbriquées.
3. E est l'ensemble des instances-constructeur closes d'un terme linéaire donné t.

Ces restrictions sont bien nécessaires car si l'une d'entre elle n'est pas satisfaite, $R^*(E)$ n'est pas régulier. En effet, quand le système de réécriture est linéaire gauche, $IRR(R)$ est régulier, et donc $R^!(E) = R^*(E) \cap IRR(R)$ est régulier si $R^*(E)$ l'est. Or le tableau suivant montre que $R^!(E)$ n'est pas régulier.

5. [55] exprime l'ensemble des termes normalisables, ce qui revient à calculer l'ensemble des descendants en orientant les règles de réécriture dans le sens contraire.

6. Ceci peut être affaibli en une restriction plus technique, voir [94].

Restriction non satisfaite	Système de réécriture	E	$R^!(E)$
Linéarité des mbres droits	$f(x) \to c(x,x)$	$\{f(t)\}$	$\{c(t,t)\}$
Restriction 1	$f(s(x),y) \to s(f(x,s(y)))$	$f(s^*(0),0)$	$s^n(f(0,s^n(0)))$
Pas de fonctions imbriquées dans les membres droits	$f(s(x),y) \to s(f(x,g(y)))$ $g(x) \to s(x)$	$f(s^*(0),0)$	$s^n(f(0,s^n(0)))$
$E = \{t\theta\}$	$f(s(x)) \to s(f(x))$	$(fs)^*(0)$	$s^n(f^n(0))$

Théorème 4.25 *[94] Si les trois restrictions énoncées précédemment sont satisfaites, alors l'ensemble des descendants $R^*(E)$ est régulier.*
Il est reconnu par un automate fini dont le nombre d'états est dans le pire cas une tour d'exponentielles de hauteur n du nombre de positions de t, n étant le nombre maximal de fonctions imbriquées[7] dans t.

La méthode ne sera pas détaillée ici. Nous présentons juste une définition et un exemple pour donner l'idée.

Définition 4.26 *On définit l'automate $\mathcal{A}_{data}$ qui reconnaît l'ensemble des termes-constructeur clos T_C : $\mathcal{A}_{data} = (C,Q_{data},Q_{data_f},\Delta_{data})$ où*
$Q_{data} = Q_{data_f} = \{q_{data}\}$ et $\Delta_{data} = \{c(q_{data},\dots,q_{data}) \to q_{data} \mid c \in C\}$.

Etant donné un terme linéaire t, on définit l'automate $\mathcal{A}_{t\theta}$ qui reconnaît les instances-constructeur closes de t : $\mathcal{A}_{t\theta} = (C \cup F,Q_{t\theta},Q_{t\theta_f},\Delta_{t\theta})$ où

$$Q_{t\theta} = \{q^p \mid p \in \overline{Pos}(t)\} \cup \{q_{data}\}$$
$$Q_{t\theta_f} = \{q^\epsilon\} \ (q_{data} \ \textit{if } t \textit{ est une variable})$$
$$\Delta_{t\theta} = \left\{ t(p)(s_1,\dots,s_n) \to q^p \mid p \in \overline{Pos}(t),\ s_i = \left| \begin{array}{l} q_{data} \ \textit{si } t|_{p.i} \textit{ est une variable} \\ q^{p.i} \ \textit{sinon} \end{array} \right. \right\} \cup \Delta_{data}$$

Exemple 4.27 Soit a,s des constructeurs et f une fonction. Considérons le terme $t = f(s(s(y)))$ ainsi que l'automate $\mathcal{A}_{t\theta}$ qui reconnaît le langage $E = f(s(s(s^*(a))))$ des instances-constructeur closes de t. $\mathcal{A}_{t\theta}$ peut être résumé par $\overset{q^\epsilon}{f}(\overset{q^1}{s}(\overset{q^{1.1}}{s}(\overset{q_{data}}{s^*(a)})))$, ce qui signifie que :

$$s^*(a) \to^*_{[\Delta_{t\theta}]} q_{data},\ \ s(s^*(a)) \to^*_{[\Delta_{t\theta}]} q^{1.1},\ \ s(s(s^*(a))) \to^*_{[\Delta_{t\theta}]} q^1,\ \ f(s(s(s^*(a)))) \to^*_{[\Delta_{t\theta}]} q^\epsilon$$

Considérons maintenant le système de réécriture $R = \{f(s(x)) \to s(f(x))\}$. Trivialement $R^*(E) = E \cup s(f(s(s^*(a)))) \cup s(s(s^*(f(s^*(a)))))$.

En réécrivant E, des instances de membres droits de règles sont introduites. Donc pour construire un automate qui reconnaît $R^*(E)$, il faut reconnaître les instances des membres droits en des états, sans faire de confusion entre les différentes instances possibles d'un même membre droit. En effet, considérons les deux premières étapes issues de E :

7. Si la i-ème branche de t contient n_i fonctions imbriquées, $n = Max(n_i)$.

$$f(s(s(s^*(a)))) \to_{[x/s(s^*(a))]} \mathcal{L}' = s(f(s(s^*(a)))) \to_{[x/s^*(a)]} s(s(f(s^*(a))))$$

Le langage qui instancie x au cours de la première réécriture est $s(s^*(a))$ (reconnu en $q^{1.1}$), alors que c'est $s^*(a)$ (reconnu en q_{data}) au cours de la deuxième. Par conséquent, on code deux versions du membre droit : $\overset{d^{\epsilon}_{q^{1.1}}}{s}(\overset{d^{1}_{q^{1.1}}}{f}(\overset{q^{1.1}}{x}))$ et $\overset{d^{\epsilon}_{q_{data}}}{s}(\overset{d^{1}_{q_{data}}}{f}(\overset{q_{data}}{x}))$, en rajoutant les états $d^{\epsilon}_{q^{1.1}}, d^{1}_{q^{1.1}}, d^{\epsilon}_{q_{data}}, d^{1}_{q_{data}}$ et les transitions :

$$f(q^{1.1}) \to d^{1}_{q^{1.1}},\ s(d^{1}_{q^{1.1}}) \to d^{\epsilon}_{q^{1.1}},\ f(q_{data}) \to d^{1}_{q_{data}},\ s(d^{1}_{q_{data}}) \to d^{\epsilon}_{q_{data}}$$

Ainsi le langage reconnu en l'état $d^{\epsilon}_{q^{1.1}}$ est exactement le membre droit instancié par $\overset{q^{1.1}}{s}(\overset{q_{data}}{s^*(a)})$, donc $\mathcal{L}' = s(f(s(s^*(a)))) \to^* d^{\epsilon}_{q^{1.1}}$. De même le langage reconnu en $d^{\epsilon}_{q^{data}}$ est exactement le membre droit instancié par $\overset{q_{data}}{s^*(a)}$, donc $s(f(s^*(a))) \to^* d^{\epsilon}_{q_{data}}$. Plus généralement, on code une version du membre droit pour chaque état de $Q_{t\theta}$.

On simule maintenant les étapes de réécriture grâce aux langages, en ajoutant des transitions. Par exemple, considérons de nouveau la première étape de réécriture issue de E :

$$\overset{q^{\epsilon}}{f}(\overset{q^{1}}{s}(\overset{q^{1.1}}{s}(\overset{q_{data}}{s^*(a)}))) \to_{[x/s(s^*(a))]} \mathcal{L}' = s(f(s(s^*(a))))$$

Puisque $f(s(x))$ est le membre gauche de la règle, et $f(s(q^{1.1})) \to^*_{\Delta_{t\theta}} q^{\epsilon}$, on ajoute la transition $d^{\epsilon}_{q^{1.1}} \to q^{\epsilon}$. Ainsi $\mathcal{L}' = s(f(s(s^*(a)))) \to^* d^{\epsilon}_{q^{1.1}} \to q^{\epsilon}$ qui est l'état d'acceptation. Donc $\mathcal{L}'$ est reconnu par l'automate. Plus généralement, à chaque fois que $f(s(q)) \to^* q'$ où $q \in Q_{t\theta}$, on ajoute $d^{\epsilon}_{q} \to q'$. $\diamond$

Dans cet exemple, t contient une seule fonction. Quand il contient des fonctions imbriquées, on fabrique un automate par fonction et on utilise une opération d'imbrication d'automates.

Par ce calcul de descendants, on calcule tous les descendants, y compris ceux qui ne contiennent pas que des constructeurs. Pour tester l'unifiabilité selon une sémantique de constructeurs (comme en section 4.1), il suffit de faire l'intersection avec l'ensemble T_C des termes-constructeur clos, qui est régulier, donc de tester $R^*(\{t\theta\}) \cap R^*(\{t'\theta\}) \cap T_C \neq \emptyset$. D'où :

Théorème 4.28 *L'unifiabilité des buts linéaires est décidable modulo les systèmes de réécriture confluents, basés sur les constructeurs, linéaires droits, et tels que dans les membres droits les arguments de fonctions sont des variables ou des termes-constructeur clos.*

Ce résultat d'unifiabilité étend celui obtenu en utilisant les TTSG (théorème 4.6), puisque les membres gauches peuvent être non linéaires, et les arguments de fonction des membres droits peuvent être des termes-constructeur clos. Par contre, par cette méthode on n'exprime pas les solutions.

4.3 Résultats de décidabilité apparentés

La décidabilité de l'unifiabilité a été établie pour d'autres classes de systèmes de réécriture.

D'abord pour les systèmes de réécriture clos [58] par D. Kozen, résultat amélioré ensuite par J.M. Hullot [53] grâce à la surréduction basique, en permettant que les membres droits soient des termes clos ou des variables. De son coté, S. Mitra [83] suppose que les membres droits sont des termes-constructeur. Tous ces résultats sont très restrictifs car ils n'autorisent pas la récursivité.

Dans [22], J. Christian donne un nouveau critère : tous les membres gauches sont plats ($f(s_1,\ldots,s_n)$ est plat si $\forall i \in [1,n]$, s_i est une variable ou un terme-constructeur clos) et les règles de réécriture sont orientées par un ordre bien fondé. H. Comon, M. Haberstrau et J.P. Jouannaud [26] ont montré la décidabilité pour les théories peu-profondes (shallow, c-à-d les variables des membres des équations sont à la profondeur zéro ou un[8]), étendue ensuite par R. Nieuwenhuis [90] aux théories standards.

Un autre résultat a été établi par D. Kapur et P. Narendran [57], en supposant que pour chaque règle, tout appel de fonction dans le membre droit[9] est un sous-terme strict du membre gauche.

Les travaux ci-dessus portent en fait sur des théories finitaires, et les preuves sont souvent basées sur des stratégies complètes de surréduction générant un espace de recherche fini. Abordons maintenant les théories infinitaires.

Comme indiqué en section 4.1, une troisième instance de la méthode générale d'unification à l'aide des langages synchronisés a été développée par S. Limet et F. Saubion [73] en utilisant les grammaires primales. La décidabilité est obtenue pour les systèmes de réécriture confluents et primaux. Un système primal est un système linéaire gauche et basé sur les constructeurs, tel que chaque fonction récursive est définie par au plus deux règles, dont une seule est récursive, et dans les membres droits les arguments des appels de fonction sont des variables.

S. Mitra a montré la décidabilité [83, 84] pour les systèmes de réécriture confluents, linéaires, et basés sur les constructeurs, en supposant que pour chaque fonction définie f, il y a au plus une règle de la forme $f(t_1,\ldots,t_n) \to r$ où r n'est pas un terme-constructeur. De plus ce membre droit ne doit contenir qu'une fonction (disons g) et le sous-terme dont g est la racine doit être plat au sens de [22]. Grâce à la notion de substitution itérée, S. Mitra peut représenter finiment l'ensemble infini des unificateurs et décider l'unifiabilité.

Un autre résultat a été établi par H. Faßbender et S. Maneth [34] pour les systèmes de réécriture confluents et basés sur les constructeurs, en supposant que tous les construc-

8. Cela ressemble aux termes plats de J. Christian, mais ici il n'y a pas de notion de constructeurs. Pour d'autres auteurs le concept de 'shallow term' est plus restrictif : c'est une variable ou un terme de la forme $f(x_1,\ldots,x_n)$.

9. C-à-d tout sous terme du membre droit dont le symbole de tête est une fonction.

teurs sont monadiques et qu'il y a un seul symbole de fonction f. Cette fonction doit de plus être définie de la manière suivante : pour tout constructeur c, il existe exactement une règle de réécriture de la forme $f(c(x)) \to t$, où t est soit un terme-constructeur clos, soit un terme de la forme $t'[u \leftarrow f(x)]$, t' étant un terme-constructeur et u l'occurrence la plus grande de t' (il n'y en a qu'une car d'après les hypothèses t' ne contient qu'une seule branche). On peut d'ailleurs noter que ces restrictions font que tous les symboles (fonctions et constructeurs) sont monadiques (d'arité 1 ou 0). Par conséquent, on pourrait se ramener à des problèmes sur les langages de chaînes.

On peut aussi remarquer que les résultats de décidabilité que nous avons obtenus ne sont ni des extensions ni des restrictions de ceux cités ci-dessus. En effet contrairement à [53, 22, 26, 90] et [57], nous autorisons des théories infinitaires. Par rapport aux systèmes primaux [73] de S. Limet et F. Saubion et aux restrictions de S. Mitra [83, pages 100-109], nous autorisons qu'une fonction soit définie par plusieurs règles de réécriture récursives. Par rapport à H. Faßbender et S. Maneth [34], nous ne restreignons pas l'arité des fonctions et des constructeurs. Par contre ces auteurs ne supposent pas pas la linéarité du but, contrairement à nos résultats obtenus à l'aide des TTSG ou par un calcul de descendants, ni l'horizontalité, contrairement à notre résultat obtenu avec les WRR.

Y. Kaji, T. Fujiwara et T. Kasami ont donné une procédure qui, lorsqu'elle termine, décide l'unifiabilité [56]. Les restrictions supposées sont la linéarité du but et des membres droits, ainsi qu'une forme affaiblie de la linéarité des membres gauches. Le principe est de représenter l'ensemble des termes obtenus par surréduction par des automates finis d'arbres, ceci pour chaque membre du but, et de faire l'intersection des deux automates. Autrement dit cela ressemble à un calcul de descendants, mais contrairement à celui que j'ai présenté en section 4.2, les auteurs n'ont pas donné des restrictions suffisantes pour garantir la terminaison, et donc la décidabilité. De plus leur méthode est moins élaborée puisqu'elle ne termine pas en partant d'un système comme $\{s(x) + y \to s(x + y),\ 0 + x \to x\}$.

4.4 Par surréduction dirigée

Schématiquement, on peut considérer qu'il y a deux grandes approches de la surréduction (narrowing), chacune correspondant à une utilisation particulière :

1. En tant que sémantique opérationnelle des langages de programmation logico-fonctionnels [47], comme K-Leaf [40], Babel [86], Toy [41], Curry [48]. Ces langages utilisent la surréduction paresseuse (lazy narrrowing) [85] ou par nécessité (needed narrowing) [7]. Pour écrire un programme du type 'générer et tester', l'évaluation paresseuse à l'avantage d'effectuer un coroutinage automatique, c-à-d de commencer à tester un élément non complètement instancié, sans avoir besoin de programmer explicitement la fusion des procédures de génération et de

test [10]. De plus le risque d'enlisement (floundering) n'existe pas, contrairement au coroutinage en programmation logique (en utilisant le 'freeze').

Dans cette approche, le programme est écrit sous forme de règles de réécriture linéaires à gauche, et respectant une discipline de constructeurs. Certains langages autorisent les fonctions non déterministes [11], c-à-d les systèmes de réécriture non confluents. Bien évidemment, l'efficacité de la méthode d'évaluation est primordiale.

2. En tant que méthode d'unification dans les systèmes de réécriture confluents. Dans cette approche, la discipline de constructeurs n'est pas exigée. Différentes variantes de la surréduction ont été proposées : basique [53, 91], normalisante [35], à la fois basique et normalisante [93], LSE [59, 14], etc...

Nos travaux relèvent de la deuxième approche. Nous avons introduit la *surréduction dirigée par un graphe de termes* (RTA'91, [19]), qui consiste à diriger la surréduction par une sorte de graphe afin de ne calculer que des étapes susceptibles de conduire à des solutions de l'équation à unifier. On réduit ainsi l'espace de recherche.

Théorème 4.29 *La surréduction dirigée est complète dans les systèmes de réécriture confluents et terminants [19].*
Elle termine plus souvent que la surréduction basique, et si on suppose la discipline de constructeurs, que la surréduction paresseuse de Babel [61].

D'autre part, une comparaison informelle avec les techniques d'interprétation abstraite en réécriture [13, 11], plus précisément avec la surréduction abstraite de [1], a été réalisée [61].

Une extension à la réécriture conditionnelle a été effectuée et implantée [62].

Théorème 4.30 *[60] La surréduction dirigée conditionnelle est complète dans les systèmes de réécriture conditionnels confluents et décroissants d'une part, et dans les systèmes de réécriture conditionnels confluents par niveaux et terminants.*
Elle termine plus souvent que la surréduction basique conditionnelle. De plus, si le système de réécriture n'a pas de paires critiques, chaque solution est générée une seule fois (même modulo un renommage de variables).

Cette dernière propriété est intéressante car les conditions suffisantes habituelles qui garantissent la confluence par niveaux interdisent les paires critiques. Cela ne permet cependant pas de conclure que l'ensemble d'unificateurs généré est minimal, car pour cela, il faudrait que chaque solution soit générée une seule fois modulo le système de réécriture.

---

10. Voir par exemple la petite application musicale (génération automatique d'accords à partir d'une mélodie donnée) que j'ai programmée en Curry avec M. Hanus [49].

11. Même si tous les arguments de la fonction sont complètement instanciés, elle peut retourner plusieurs valeurs. D'un point de vue mathématique, ce n'est pas une fonction, mais une relation.

Une autre surréduction ressemblant à la surréduction dirigée, mais moins élaborée, est apparue quatre ans plus tard sous la direction d'A. Middeldorp ([80, 81, 79] pour la réécriture simple, [46, 45] pour la réécriture conditionnelle). Cette surréduction est qualifiée de paresseuse par leurs auteurs, car les termes sont parcourus de haut en bas. C'est aussi le cas dans la surréduction dirigée. Il ne faut cependant pas faire de confusion entre cette notion de paresse et celle utilisée en programmation logico-fonctionnelle, cette dernière étant définie à partir de la notion de constructeur.

En comparant la surréduction dirigée avec celle de Middeldorp, on voit que cette dernière :

- n'est pas dirigée (pas de graphe).
- utilise uniquement des équations ordinaires, alors que la surréduction dirigée utilise des équations orientées de la forme $s \vec{=}_R t$ dont les solutions sont $\{\theta$ substitution normalisée $\mid \theta s \rightarrow^*_R \theta t\}$. Par conséquent, quand elle surréduit s par la règle $l \rightarrow r$ à la position p, l'unification de $s|_p$ et l est faite modulo le système de réécriture, alors qu'elle devrait être syntaxique. Cela reste bien sûr correct, mais c'est source d'inefficacité.
- instancie les variables des équations, ce qui provoque des duplications de termes si la même variable apparait plusieurs fois (cas des règles non linéaires). C'est encore source d'inefficacité.
- n'est pas implantée.

Cette inefficacité lui permet toutefois d'avoir des résultats de complétude un peu plus larges, en affaiblissant l'hypothèse de terminaison du système de réécriture : terminant est affaibli en faiblement terminant pour la réécriture simple, et en terminant par niveaux pour la réécriture conditionnelle mais en supposant alors que le choix de l'équation à traiter est 'don't know'.

Chapitre 5

Perspectives

Il existe des liens entre les langages définis par des formules logiques et ceux définis par des automates. Il est en effet connu que la logique monadique faible du second ordre avec k successeurs WSkS coïncide par un codage avec les langages réguliers d'arbres, et plus généralement avec les RR. Concernant les langages non réguliers, R. Treinen a montré que les formules uniformes d'une logique appelée WS2Sy coïncidaient avec les automates d'arbres avec contraintes [105]. Un peut donc se demander s'il existe une logique qui coïnciderait avec les CS. Du fait de la non stabilité des CS par intersection et complémentation, cette logique ne devrait pas contenir les opérateurs $\wedge$ et $\neg$. Même question pour les WRR, avec une logique qui contiendrait $\wedge$, mais pas $\neg$.

Nous avons défini une sous-classe des langages synchronisés, stable par intersection : les WRR. Mais on a vu qu'on pouvait parfois calculer l'intersection de langages synchronisés qui ne sont pas des WRR. Peut-on alors définir d'autres sous-classes, plus expressives ou orthogonales aux WRR, stables par intersection ? Dans le même ordre d'idée, peut-on définir une sous-classe stable par complément ? C'est bien sûr possible en rajoutant une hypothèse forte de déterminisme sur les WRR, mais l'expressivité semble alors très réduite.

Dans les travaux sur les algèbres de processus, on a vu que si on sait exprimer $\rightarrow^*$ par un langage de couples d'arbres, et qu'on sait faire la jointure et la projection, on peut exprimer les descendants. D'une manière générale, exprimer $\rightarrow^*$ est nettement plus difficile que d'exprimer $\rightarrow$. D'où l'intérêt d'avoir des langages de relations, stables par fermeture transitive. Les langages synchronisés (les CS) ne le sont pas, mais contiennent les GTT (Ground Tree Transducers [32]) qui le sont. Malheureusement les GTT sont peu expressifs. D'où l'idée de chercher une sous-classe des langages synchronisés, plus expressive que les GTT, et stable par fermeture transitive. La difficulté sera toutefois de conserver les autres propriétés des CS.

La bissimulation est une propriété importante en concurrence. On sait que la bissimulation forte est décidable dans PA normé [52], alors qu'elle est indécidable dans CCS. BCPC se situant entre PA et CCS quant à l'expressivité, il serait intéressant de savoir si la bissimulation est décidable dans BCPC normé. Si c'est le cas, cela pourrait peut-être permettre de prouver sur des structures infinies, des propriétés fortes de sécurité comme la non-interférence entre niveaux de secret [87].

Il semblerait intéressant d'instancier la méthode générale d'unification présentée en section 4.1 par d'autres langages d'arbres, afin d'obtenir de nouveaux résultats sur l'unifiabilité et les preuves inductives. Pour qu'un langage d'arbres puisse convenir, il doit vérifier des propriétés tout à fait standards, plus une qui l'est moins : exprimer N_f, c-à-d simuler la surréduction en étant capable d'exprimer en même temps le terme obtenu et les instances correspondantes des variables. Cette dernière propriété est à priori la plus difficile à satisfaire, et nécessite à coups sûr des restrictions fortes sur le système de réécriture et/ou sur le but. Mais comment pourrait-il en être autrement puisqu'on sait bien que l'unification équationnelle est hautement indécidable (on peut très facilement coder le problème de Post) ? Autrement dit, les restrictions doivent être suffisamment fortes pour empêcher le codage du problème de Post, ou pour se trouver dans un cas particulier de Post qui est décidable.

Pour donner une orientation plus pratique à cette méthode, en vue par exemple de son intégration dans un prouveur de théorèmes, on pourrait sans doute affaiblir les restrictions et se contenter d'une semi-décision en calculant des approximations de l'ensemble des solutions : un sur-ensemble (resp. sous-ensemble) permettrait de prouver dans un certain nombre de cas que l'ensemble des solutions est (resp. n'est pas) vide. Pour cela, on pourrait s'inspirer de travaux existants sur l'approximation de la surréduction par des interprétations abstraites, comme [13, 11, 12].

Pour rechercher des unificateurs au moyen de la surréduction, nous avons dirigé celle-ci par un graphe de termes, afin de réduire les redondances. La programmation logico-fonctionnelle utilise une autre technique, les arbres de définition (Definitional trees [5, 6]), sorte de pré-compilation de la surréduction paresseuse. Les arbres de définition étant basés sur la notion de constructeur, il semble difficile de les adapter à une signature sans constructeurs. Par contre, il est sans doute possible d'introduire les graphes de termes dans la programmation logico-fonctionnelle.

Nous avons utilisé un calcul de descendants pour décider l'unifiabilité. Mais il peut aussi servir à d'autres choses, comme par exemple la décision de l'accessibilité par réécriture, la vérification de programmes écrits sous forme de systèmes de réécriture [94], la vérification de protocoles cryptographiques [39].

Il serait intéressant d'étudier le comportement de la méthode de T. Genet [38] (calcul d'un sur-ensemble des descendants) quand les restrictions que j'ai supposées pour mon calcul exact de descendants [94] sont satisfaites. Son calcul devient-il exact dans ce cas ? Une comparaison de ces travaux avec un autre calcul publié récemment [103] serait aussi souhaitable.

On pourrait d'autre part s'intéresser à des calculs de descendants selon certaines stratégies de réécriture. Le calcul que j'ai proposé peut par exemple être adapté à la réécriture 'innermost', en faisant l'intersection de l'automate qui reconnaît les descendants issus de la position p, avec l'automate des formes normales, avant de l'imbriquer dans l'automate global. Un premier résultat a été obtenu sur ce sujet pendant le DEA de Julie Vuotto [107].

Remarquons que notre méthode générale d'unification, par ses instances, induit des calculs particuliers de descendants. Etant donné un terme t, elle permet en effet de calculer l'ensemble des substitutions-constructeur closes σ telles que $\sigma t \rightarrow^* s$ par une dérivation à filtres-constructeur [1], où s est un terme-constructeur. On peut donc exprimer $T^c(\{\theta t\})$ par un langage d'arbres, où $T^c(E)$ sont les termes-constructeur descendants de E par une stratégie à filtres-constructeur. On obtient donc des calculs de descendants issus de langages non réguliers (les instances d'un terme non linéaire) lorsqu'on utilise les grammaires primales et les WRR. Ce genre de calcul pourrait sans doute être utile dans les systèmes de réécriture suffisamment complet [2].

1. Les variables des règles de réécriture sont instanciées par des termes-constructeur.
2. Dans ce cas, une stratégie est à filtres-constructeur ssi elle est innermost.

Bibliographie

[1] M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an Optimization for Equational logic programs. In *proceedings of the 5th International Symposium on Programming Language Implementation and Logic Programming*, volume 714 of *LNCS*, pages 391–409. Springer-Verlag, 1993.

[2] Ali Amaniss, Miki Hermann, and Denis Lugiez. Set Operations for Recurrent Term Schematizations. In *Colloquium on Trees in Algebra and Programming*, volume 1214. Lecture Notes in Computer Science, 1997. ISBN 3-540-62781-2.

[3] S. Anantharaman and G. Richard. A Rewrite Mechanism for Logic Programs with Negation. In *Proceedings 7th Conference on Rewriting Techniques and Applications, New Brunswick (USA)*, pages 163–178, 1995.

[4] S. Anantharaman and G. Richard. A Rewrite Mechanism for Logic Programs with Negation. In *Theoretical Computer Science, 192(1):77-106*. Elsevier Science Publishers, 1998.

[5] A. Antoy. Lazy Evaluation in Logic. In *Symposium on Programming Language Implementation and Logic Programming*, volume 528 of *LNCS*, pages 371–382. Springer-Verlag, 1991.

[6] A. Antoy. Definitional Trees. In *Conference on Algebric and Logic Programming*, volume 632 of *LNCS*, pages 143–157. Springer-Verlag, 1992.

[7] A. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. In *Proceedings 21st ACM Symposium on Principle of Programming Languages, Portland*, pages 268–279, 1994.

[8] A. Arnold and M. Dauchet. Morphismes et bimorphismes d'arbres. In *Theoretical Computer Science, 20:33-93*. Elsevier Science Publishers, 1982.

[9] F. Baader and J. Siekmann. Unification Theory. In D.M. Gabbay, C.J. Hogger, and Robinson J.A., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, Oxford, UK, 1993.

[10] J.C.M. Baeten and W.P. Weijland. Process algebra. In *Cambridge Tracts in Theoretical Computer Science*, volume 18, 1990.

[11] D. Bert and R. Echahed. Abstraction of Conditional Term Rewrite Systems. In J. Lloyd, editor, *proceedings 3rd Internationnal Symposium on Logic Programming*

ILPS'95, volume 906, pages 162–176. MIT Press, 1995.

[12] D. Bert, R. Echahed, and K. Adi. Resolution of goals with the functional and logic language lpg : impact of abstract interpretation. In M. Wirsing and M. Nivat, editors, *proceedings 5th Internationnal Conf. on Algebraic Methodology and Software Technology AMAST'96*, volume 1101 of *LNCS*, pages 629–632. Springer-Verlag, 1996.

[13] D. Bert, R. Echahed, and B. Østvold. Abstract Rewriting. In P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, editors, *proceedings 3rd Internationnal Workshop on Static Analysis WSA'93*, volume 724 of *LNCS*, pages 178–192. Springer-Verlag, 1993.

[14] A. Bockmayr, S. Krischer, and A. Werner. Narrowing Strategies for Arbitrary Canonical Systems. *Fundamenta Informaticae*, 24(1,2):125 – 155, 1995.

[15] B. Bogaert and S. Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In *Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *LNCS*. Springer-Verlag, 1992.

[16] M.P. Bonacina and J. Hsiang. On Rewrite Programs: Semantics and Relationship with Prolog. *Journal of Logic Programming*, 14:155–180, 1992.

[17] A.C. Caron, F. Seynhaeve, S. Tison, and M. Tommasi. Deciding the Satisfiability of Quantifier Free Formulae on One-Step Rewriting. In *Proceedings of 10th Conference RTA, Trento (Italy)*, volume 1631 of *LNCS*. Springer-Verlag, 1999.

[18] J. Chabin, S. Anantharaman, and P. Réty. E-Unification via Constrained Rewriting. Technical Report 93-7, Laboratoire d'Informatique Fondamentale d'Orléans, 1993. Also presented at UNIF'93, Boston (USA).

[19] J. Chabin and P. Réty. Narrowing Directed by a Graph of Terms. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *LNCS*, pages 112–123. Springer-Verlag, April 1991.

[20] J. Chabin and P. Réty. Directed Narrowing Including Merging. Technical Report 92-1, Laboratoire d'Informatique Fondamentale d'Orléans, 1992.

[21] H. Chen and J. Hsiang. Recurrence Domains : their Unification and Application to Logic Programming. In *Information and Computation, 122:45-69.* 1995.

[22] J. Christian. Some Termination Criteria for Narrowing and E-Unification. In Saragota Springs, editor, *Proceedings 11th International Conference on Automated Deduction, Albany (NY, USA)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 582–588. Springer-Verlag, 1992.

[23] H. Comon. On Unification of Terms with Integer Exponents. *Mathematical Systems Theory*, 28(1):67–88, 1995.

[24] H. Comon. Sequentiality, second order monadic logic and tree automata. In *Proc., Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 508–517. IEEE Computer Society Press, 26–29 June 1995.

[25] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications (TATA).* http://l3ux02.univ-lille3.fr/tata.

[26] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Syntacticness, Cycle-Syntacticness and Shallow Theories. *Information and Computation*, 111(1):154–191, 1994.

[27] J. Coquidé, M. Dauchet, R. Gilleron, and S. Vagvolgyi. Bottom-up Tree Push-down Automata and Rewrite Systems. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *LNCS*, pages 287–298. Springer-Verlag, April 1991.

[28] P. Schnoebelen D. Lugiez. The Regular Viewpoint on PA-Processes. In ?, editor, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 1–17, Nice, France, September 1998.

[29] M. Dauchet, A.C. Caron, and J.L. Coquidé. Reduction Properties and Automata with Constraints. In *Journal of Symbolic Computation, 20:215-233.* 1995.

[30] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc., Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 242–248, Philadelphia, Pennsylvania, 1990. IEEE Computer Society Press.

[31] M. Dauchet and S. Tison. Algebraic Complexity of Tree Languages. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages.* Elsevier, 1992.

[32] M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne. Decidability of the Confluence of Ground Term Rewriting Systems. In *INRIA Report 675.* 1987.

[33] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. Van Leuven, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320. Elsevier Science Publishers, 1990.

[34] H. Faßbender and S. Maneth. A Strict Border for the Decidability of E-Unification for Recursive Functions. In *proceedings of the intern. Conf. on Algebraic and Logic Programming. To appear.*, 1996.

[35] M. Fay. First Order Unification in an Equational Theory. In *Proceedings of the 4th Workshop on Automated Deduction*, pages 161–167, Austin, Texas. 1979.

[36] L. Fribourg. SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting. In *proceedings IEEE Symposium on Logic Programming*, pages 172–185, Boston, 1985.

[37] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó- Budapest, 1984.

[38] T. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In *Proceedings of 9th Conference on Rewriting Techniques and Applications, Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.

[39] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proceedings of CADE*, LNCS. Springer-Verlag, 2000.

[40] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel LEAF: a Logic plus Functional Language. *The Journal of Computer and System Sciences*, 42(3):139–185, 1991.

[41] J.C. González-Navarro, M.T. Hortalá-González, F.J. López-Fraguas, and M. Rodriguez-Artalejo. An Approach to Declarative Programming Based on a Rewrite Logic. *Journal of Logic Programming*, 40:47–87, 1999.

[42] V. Gouranton, P. Réty, and H. Seidl. Synchronized Tree Languages Revisited and New Applications. Research Report 2000-16, LIFO, 2000. http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety/publications.html.

[43] V. Gouranton, P. Réty, and H. Seidl. Synchronized Tree Languages Revisited and New Applications. In *Proceedings of FoSSaCs*, volume xxx of *LNCS*. Springer-Verlag, 2001.

[44] John V. Guttag. The Specification and Application to Programming of Abstract Data Types. Technical report, Computer System Research Report CSRG-59, University of Toronto, 1975.

[45] M. Hamada. Strong Completeness of a Narrowing Calculus for Conditional Rewrite Systems with Extra Variables. In *Proceedings of the 6th Austrialian Theory Symposium (CATS), Springer-Verlag*, 2000.

[46] M. Hamada and T. Middeldorp, A. Suzuki. Completeness Results for a Lazy Conditional Narrowing Calculus. In *Proceedings of the 2nd Discrete Mathematics and Theoretical Computer Science Conference and the 5th Austrialian Theory Symposium (CATS), Auckland, Springer-Verlag, p217-231*, 1999.

[47] M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19 & 20:583–628, May/July 1994.

[48] Michael Hanus. Curry: An Integrated Fonctional Logic Language. Technical report, Available at http://www-i2.informatik.rwth-aachen.de/ hanus/curry. 1998.

[49] Michael Hanus and Pierre Rety. Demand-driven Search in Functional Logic Programs. In *Proc. of the 13th Workshop on Logic Programming (WLP'98), (U. Egly, H. Tompits, Eds.), Vienna, 1998, pp. 13-24*. Research Report 1843-1998-10, Institute of Information Systems, Technical University Vienna.

[50] N. Heintze and J. Jaffar. A Decision Procedure for a Class of Set Constraints. In *Proc. of 5th Symposium on LICS*, pages 42–51, 1990.

[51] M. Hermann and R. Galbavý. Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176, 1997.

[52] Y. Hirshfeld and M. Jerrum. Bisimulation Equivalence is Decidable for Normed Process Algebra. In *Proc. of 26th International Colloquium on Automata, Languages and Programming*, volume 1644 of *LNCS*, pages 412–421. Springer-Verlag, 1999.

[53] J.-M. Hullot. Canonical Forms and Unification. In W. Bibel and R. Kowalski, editors, *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, volume 87 of *LNCS*, pages 318–334. Springer-Verlag, 1980.

[54] F. Jacquemard. *Automates d'Arbres et Réécriture de Termes.* Thèse de Doctorat d'Université, Université de Paris-sud, 1996. In french.

[55] F. Jacquemard. Decidable Approximations of Term Rewrite Systems. In H. Ganzinger, editor, *Proceedings 7th Conference on Rewriting Techniques and Applications, New Brunswick (USA)*, volume 1103 of *LNCS*, pages 362–376. Springer-Verlag, 1996.

[56] Y. Kaji, T. Fujiwara, and T. Kasami. Solving a Unification Problem under Constrained Substitutions Using Tree Automata. In *Proc. Fourteenth Conference on FST & TCS, Madras, India*, volume 880 of *LNCS*, pages 276–287. Springer-Verlag, 1994.

[57] D. Kapur and P. Narendran. Matching, Unification and Complexity. *Sigsam Bulletin*, 21(4):6–9, November 1987.

[58] D. Kozen. Positive First Order Logic is np-complete. *IBM Journal of Res. Developp.*, 25(4):327–332, 1981.

[59] S. Krischer and A. Bockmayr. Detecting Redundant Narrowing Derivations by the LSE-SL Reducibility Test. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *LNCS*, pages 74–85. Springer-Verlag, April 1991.

[60] S. Limet and P. Réty. Conditional Directed Narrowing (including proofs). Technical Report 95-11, Laboratoire d'Informatique Fondamentale d'Orléans, 1995.

[61] S. Limet and P. Réty. Directed Narrowing: An Optimization That More Often Terminates (including proofs). Technical Report 95-07, Laboratoire d'Informatique Fondamentale d'Orléans, 1995.

[62] S. Limet and P. Réty. Conditional Directed Narrowing. In M. Wirsing and M. Nivat, editors, *proc of the 5th Conf. on Algebraic Methodology and Software Technology*, volume 1101 of *LNCS*, pages 637–640. Springer-Verlag, 1996.

[63] S. Limet and P. Réty. E-unification by Means of Tree Tuple Synchronized Grammars. Research Report RR-LIFO-96-16, LIFO, 1996.

[64] S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 429–440. Springer-Verlag, 1997.

[65] S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. *Discrete Mathematics and Theoritical Computer Science (http://dmtcs.loria.fr/)*, 1:69–98, 1997.

[66] S. Limet and P. Réty. A New Result about the Decidability of the Existential One-step Rewriting Theory. Research Report RR-LIFO-98-12, LIFO, 1998.

[67] S. Limet and P. Réty. Solving Disequations modulo some Class of Rewrite Systems. In *Proceedings of 9th Conference on Rewriting Techniques and Applications, Tsukuba (Japon)*, volume 1379 of *LNCS*, pages 121–135. Springer-Verlag, 1998.

[68] S. Limet and P. Réty. A New Result about the Decidability of the Existential One-step Rewriting Theory. In *Proceedings of 10th Conference on Rewriting Techniques and Applications, Trento (Italy)*, volume 1631 of *LNCS*. Springer-Verlag, 1999.

[69] S. Limet, P. Réty, and H. Seidl. Weakly Regular Relations and Applications. Research Report RR-LIFO-00-17, LIFO, 2000. http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety/publications.html.

[70] S. Limet, P. Réty, and H. Seidl. Weakly Regular Relations and Applications. In *Proceedings of 12th Conference on Rewriting Techniques and Applications, Utrecht (The Netherlands)*, volume xxx of *LNCS*. Springer-Verlag, 2001.

[71] S. Limet and F. Saubion. A Proof Method for Logic Programming. Technical Report 97-14, Laboratoire d'Informatique Fondamentale d'Orléans, 1997.

[72] S. Limet and F. Saubion. On Partial Validation of Logic Programs. In M. Johnson, editor, *proc of the 6th Conf. on Algebraic Methodology and Software Technology, Sydney (Australia)*, volume 1349 of *LNCS*, pages 365–379. Springer Verlag, 1997.

[73] S. Limet and F. Saubion. A general framework for R-unification. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *proc of PLILP-ALP'98*, volume 1490 of *LNCS*, pages 266–281. Springer Verlag, 1998.

[74] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using CSP and FDR. In *Proceedings of the 2nd Conf. TACAS*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.

[75] D. Lugiez and P. Schnoebelen. Decidable first-order Transition logics for PA-Processes. In Springer, editor, *ICALP*, LNCS, Geneva, Switzerland, July 2000.

[76] D. Lugiez and P. Schnoebelen. The Regular Viewpoint on PA-Processes. *Theoretical Computer Science*, 2000.

[77] J. Marcinkowski. Undecidability of the First-order Theory of One-step Right Ground Rewriting. In *Proceedings 8th Conference on Rewriting Techniques and Applications, Sitges (Spain)*, volume 1232 of *LNCS*, pages 241–253. Springer-Verlag, 1997.

[78] R. Mayr. Model Checking PA-Processes. In?, editor, *CONCUR*, volume 1243 of *Lecture Notes in Computer Science*, pages 332–346, Warsaw, Poland, July 1997.

[79] A. Middeldorp and S. Okui. A Deterministic Lazy Narrowing Calculus. *Journal of Symbolic Computation*, 25(6):733–757, 1998.

[80] A. Middeldorp, S. Okui, and T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. In *procedings of the 20th Colloquium on Trees in Algebra and Programming, LNCS*, 1995.

[81] A. Middeldorp, S. Okui, and T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. *Theoretical Computer Science*, 167(1,2):95–130, 1996.

[82] R. Milner. *Communication and Concurrency*. Prentice Hall Int., 1989.

[83] S. Mitra. *Semantic Unification for Convergent Rewrite Systems.* Phd thesis, Univ. Illinois at Urbana-Champaign, 1994.

[84] S. Mitra. Semantic Unification for Convergent System. Technical Report CS-R-94-1855, University of Illinois at Urbana-Champaign, 1994.

[85] H. Moreno-Navarro, J.J.AND Kuchen, R. Loogen, and M. Rodriguez-Artalejo. Lazy narrowing in a graph machine. In H. Kirchner and W. Wechler, editors, *Proc. on the second International Conf. on Algebraic and Logic Programming (Nancy)*, volume 463, pages 298–31. Springer-Verlag, 1990.

[86] J.J. Moreno-Navarro and M. Rodriguez-Artalejo. Logic Programming with Functions and Predicates : The Language BABEL. *Journal of Logic Programming*, 12(3):191–223, 1992.

[87] J. Mullins. Nondeterministic Admissible Interference. *Journal of Universal Computer Science*, 6(11):1054–1070, 2000.

[88] J. Niehren, M. Pinkal, and P. Ruhrberg. On Equality up-to Constraints over Finite Trees ,Context Unification and One-step Rewriting. In W. Mc Cune, editor, *Proc. of CADE'97, Townsville (Australia)*, volume 1249 of *LNCS*, pages 34–48, 1997.

[89] J. Niehren, S. Tison, and R. Treinen. On Rewrite Constraints and Context Unification. *Information Processing Letters*, 1-2:35–40, 2000.

[90] R. Nieuwenhuis. Basic Paramodulation and Decidable Theories. In *procedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, 1996.

[91] Werner Nutt, Pierre Réty, and Gert Smolka. Basic Narrowing Revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.

[92] J.C. Raoult. Rational Tree Relations. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 4:149–176, 1997.

[93] P. Réty. Improving Basic Narrowing Techniques. In P. Lescanne, editor, *Proceedings 2nd Conference on Rewriting Techniques and Applications, Bordeaux (France)*, volume 256 of *LNCS*, pages 228–241, Bordeaux (France), May 1987. Springer-Verlag.

[94] P. Réty. Regular Sets of Descendants for Constructor-based Rewrite Systems. In *Proceedings of the 6th international conference on Logic for Programming and Automated Reasoning (LPAR), Tbilisi (Republic of Georgia)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1999.

[95] G. Richard and F. Saubion. A Rewrite Approach for Constraint Logic Programming. In *Proceedings of LATIN, Valparaiso*, LNCS. Springer-Verlag, 1995.

[96] G. Richard and F. Saubion. Answer Synthesis for CLP Programs with Negation. *Journal of the Interest Group in Pure and Applied Logic*, 5(3), 1997.

[97] K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *The Journal of Computer and System Sciences*, 37:367–394, 1988.

[98] G. Salzer. The Unification of Infinite Sets of Terms and its Applications. In *Proceedings of 3rd International Conference on Logic Programming and Automated Reasoning (LPAR)*, volume 624 of *LNCS*, pages 409–420. Springer-Verlag, 1992.

[99] F. Saubion and I. Stéphan. On Implementation of Tree Synchronized Languages. In *Proceedings of 10th Conference on Rewriting Techniques and Applications, Trento (Italy)*, LNCS. Springer-Verlag, 1999.

[100] F. Seynhaeve. *Automates, Réécriture et Contraintes : Résultats de Décidabilité et d'Indécidabilité*. Phd thesis, Université de Lille I, 1999.

[101] F. Seynhaeve, S. Tison, and M. Tommasi. Homomorphisms and Concurrent Term Rewriting. In G. Ciobanu and G. Paun, editors, *Proceedings of the twelfth International Conference on Fundamentals of Computation theory*, number 1684 in Lecture Notes in Computer Science, pages 475–487, Iasi, Romania, 1999.

[102] F. Seynhaeve, M. Tommasi, and R. Treinen. Grid Structures and Undecidable Constraint Theories. In *Proceedings of 6th Colloquium on Trees in Algebra and Programming*, volume 1214 of *LNCS*, pages 357–368. Springer-Verlag, 1997.

[103] T. Takai, Y. Kaji, and H. Seki. Right-linear finite path overlapping term rewrite systems efectively preserve recognizability. In Leo Bachmair, editor, *Proceedings of the 11th Conference on Rewriting Techniques and Applications*, volume 1833 of *LNCS*. Springer-Verlag, 2000.

[104] R. Treinen. The First-order Theory of One-step Rewriting is Undecidable. In H. Ganzinger, editor, *Proceedings 7th Conference on Rewriting Techniques and Applications, New Brunswick (USA)*, volume 1103 of *LNCS*. Springer-Verlag, 1996.

[105] R. Treinen. Predicate Logic and Tree Automata with Tests. In *Foundations of Software Science and Computation Structures (FOSSACS 2000)*, volume 1784 of *LNCS*, pages 329–343. Springer-Verlag, 2000.

[106] S. Vorobyov. The First-order Theory of One-step Rewriting in Linear Noetherian Systems is Undecidable. In *Proceedings 8th Conference on Rewriting Techniques and Applications, Sitges (Spain)*, volume 1232 of *LNCS*, pages 241–253. Springer-Verlag, 1997.

[107] J. Vuotto. Regular Sets of Descendants by Innermost Rewriting. Technical report, Rapport de DEA, LIFO, Université d'Orléans, 2001.