DEA Informatique
Université de Rennes I
ENS Cachan -antenne de Bretagne-

Automates d'arbres pour l'approximation régulière des états accessibles dans les systèmes de réécriture conditionnels

Rapport de stage
Juin 2002

Guillaume Feuillade

Encadrant :
Thomas Genet

Je tiens à remercier Thomas Genet, mon responsable, pour sa patience et son positivisme.

Je remercie également les stagiaires qui partageaient le même espace de travail que moi : Soazig Bars, Hong Tuan Ha Stéphane et Ponscarme Francois-Xavier, qui m'ont supporté durant ce stage.

Enfin, je remercie les stagiaires de DEA Sabine Carbonnel, Hervé Jegou, Boyer Fabrice, Delhumeau Jonathan, Millet Aymeric et Nicolas Yohan pour les échanges instructifs que j'ai pu avoir avec eux.

Table des matières

Introduction

Dans notre société, les domaines d'application de l'informatique sont en nombre croissant. Dans certains domaines, dits critiques, la sûreté d'exécution des systèmes informatiques est cruciale. C'est le cas par exemple des systèmes embarqués ou des systèmes devant assurer la protection de données sensibles. Il existe plusieurs manières de garantir le bon fonctionnement de tels systèmes. Citons par exemple les tests exhaustifs (pour des systèmes rudimentaires), la programmation à partir d'une spécification formelle dont la validité est prouvée, ou alors la modélisation formelle *a posteriori* d'un programme suivi de sa preuve.

Ici nous nous intéressons à un ensemble de spécifications portant sur un formalisme particulier, utile dans le cas des systèmes pouvant accéder à une infinité d'états, et qui ne se terminent pas nécessairement. Une spécification formelle est une description du système sous une forme mathématique qui se prête à des preuves formelles. Il existe un vaste ensemble de représentations envisageables dont chacune présente des avantages quand à la facilité de preuve de certaines propriétés ou d'implémentation. Dans notre cas, l'idée est de s'intéresser à pouvoir prouver qu'un système, sous certaines conditions, ne peut accéder à certains états dits *critiques*. La représentation formelle s'appuie sur des *systèmes de réécriture* (spécification algébrique sous forme d'équations orientées nommées *règles de réécriture*). Ce formalisme permet de décrire des programmes fonctionnels, ou des protocole, en cryptographie. L'objectif est de pouvoir obtenir une description de l'ensemble des états qui peuvent être pris par un programme, ou à défaut, un sur-ensemble suffisamment proche.

La première partie de ce rapport est consacrée à la description du formalisme et des outils l'accompagnant. La partie suivante s'intéressera à ce qui est actuellement réalisable, dans le cas de programmes décrits par des *systèmes de réécriture non conditionnels*, de manière exacte ou, plus généralement, par approximation. Ensuite, dans le cadre de la réalisation du stage seront exposées les deux méthode envisagées pour effectuer le traitement dans le cas des *systèmes de réécriture conditionnels*. Dans un premier temps, l'approche consistera à tenter de se ramener à des systèmes de réécriture non conditionnels pour exploiter ce qui est déjà existant. Puis sera présentée une méthode étendant le calcul d'approximation pour des systèmes non conditionnels à celui dans le cadre conditionnel. Enfin un bilan sera présenté, avec une comparaison des deux stratégies et une discussion sur les perspectives ouvertes.

Chapitre 1

Notions préliminaires

1.1 Réécriture

Dans ce chapitre, nous donnons les définitions de base utilisées dans ce rapport. Nous ne définirons pas intégralement les objets et leurs propriétés. Pour plus de détail, le lecteur pourra se reporter aux documents de référence sur la réécriture [DJ90] et sur les automates d'arbres [CDG+97].

1.1.1 Alphabet, termes et réécriture de termes

Alphabet

Un *alphabet* $\mathcal{F}$ est un ensemble de fonctions avec une arité $\in \mathbb{N}$. Les fonctions constantes sont d'arité 0. Soit $\mathcal{X}$ un ensemble dénombrable de symboles de variables. On définit récursivement l'ensemble des *termes* sur $\mathcal{F}$, noté $\mathcal{T}(\mathcal{F}, \mathcal{X})$, de la manière suivante:

1. $\mathcal{X} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$, et
2. pour tout $f \in \mathcal{F}$ avec arité$(f) = n \in \mathbb{N}$, et tous termes $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, on a $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Un terme se représente comme un arbre étiqueté par les éléments de $\mathcal{F} \cup \mathcal{X}$. Ainsi le terme $f(t_1, \ldots, t_n)$ est représenté de la manière suivante :

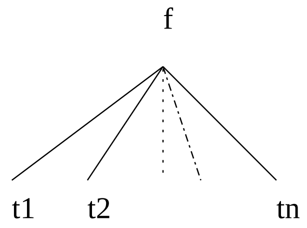

FIG. 1.1 – *structure arborescente d'un terme*

On note $\mathrm{Var}(t)$ l'ensemble des variables d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.
$\mathcal{F} = \{f : 3, cons : 2, a : 0\}$ désigne un alphabet dans lequel f est un symbole d'arité 3, $cons$ est un symbole d'arité 2 et a est une constante. Par commodité, une constante $c()$ est notée c. Un terme ne contenant aucune variable est dit *clos*, et l'ensemble des termes clos est noté $\mathcal{T}(\mathcal{F})$.

L'ensemble des *positions* d'un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, noté $\mathcal{P}os(t)$, est un mot sur $\mathbb{N}$ défini inductivement par :

1. $\mathcal{P}os(t) = \{\epsilon\}$ si $t \in \mathcal{X}$,
2. $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ et } p \in \mathcal{P}os(t_i)\}$ si $f \in \mathcal{F}$, $arite(f) = n$ et $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, où ϵ représente la suite d'entiers vide.

Si $p \in \mathcal{P}os(t)$ alors le *sous-terme* de t à la position p est noté $t|_p$. Les sous-termes $t|_p$ tels que $p \in \mathcal{P}os(t) \setminus \{\epsilon\}$ sont nommés *sous-termes stricts*. Le terme obtenu après remplacement dans t de $t|_p$ à la position p par le terme s est noté $t[s]_p$. Le symbole étiquetant le terme à la position ϵ est noté $\mathcal{R}oot(t)$.

Définition 1.1 *Une* substitution *est une application* $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$ *qui s'étend à* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ *en un endomorphisme:* $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$. *Le résultat de l'*application *d'une substitution* σ *à un terme* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ *est le terme noté* $t\sigma$.

Exemple 1.1 *Soit* $\mathcal{F} = \{f : 3, h : 1, g : 1, a : 0\}$, *soit* $\sigma = \{x \mapsto g(a), y \mapsto h(z)\}$ *et* $t = f(h(x), x, g(y))$. *On a* $t\sigma = f(h(g(a)), g(a), g(h(z)))$.

Définition 1.2 *Un contexte est un terme* $C[\,]$ *de* $\mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{X})$ *dans lequel le nouveau symbole de constante* $\Box \notin \mathcal{F}$ *n'apparaît qu'une seule fois. Pour tout contexte* $C[\,]$ *et tout terme* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $C[t]$ *est le terme obtenu par remplacement de* $\Box$ *par* t *dans* $C[\,]$.

Règles de réécriture

Une *règle de réécriture* est de la forme $l \to r$, où $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X}), l \notin \mathcal{X}$, et $\mathrm{Var}(l) \supseteq \mathrm{Var}(r)$. Un ensemble de règles de réécriture R définit une relation binaire, dite *relation de réécriture* $\to_R$. La fermeture transitive (resp. réflexive transitive) de $\to_R$ est notée $\to_R^+$ (resp. $\to_R^*$). Deux termes s et t sont *joignables* par $\mathcal{R}$, noté $s \downarrow_R t$, s'il existe un terme v tel que $s \to_R^* v$ et $t \to_R^* v$. Un terme t est dit *forme normale* de $\mathcal{R}$ si il n'admet aucune réécriture ($\nexists t'$ t.q. $t \to_{\mathcal{R}} t'$). On note $s \to_{\mathcal{R}}^! t$ si s admet t comme forme normale par $\mathcal{R}$.

1.1.2 Systèmes de réécriture

Les systèmes de réécriture constituent un formalisme adapté à la description de programmes fonctionnels, de processus communiquants ou de systèmes à états transitoires. Il existe des langages permettant d'interpréter un programme écrit sous la forme d'un *système de réécriture* tels que les langages Elan [BKK$^+$96] et Maude [CDE$^+$98]. En pratique ils constituent plus généralement un formalisme intermédiaire, destiné dans notre cas à vérifier que certains états ne puissent être atteints. Nous considérerons par la suite un système de réécriture comme la description d'un programme ou d'un protocole, même si les méthodes envisagées resterons générales. Les systèmes de réécriture possèdent la même expressivité que les machines de Turing [HL78], la programmation par réécriture n'est par conséquent pas restrictive.

Systèmes de réécriture non conditionnels

Un *système de réécriture* $\mathcal{R}$ défini sur un alphabet $\mathcal{F}$ est un ensemble de *règles de réécriture*[1].

Exemple 1.2 *Voici un exemple simple de système de réécriture définissant la multiplication et l'addition sur les entiers où* $\mathcal{F} = \{0:0, s:1, +:2, *:2\}$, *et* $x, y \in \mathcal{X}$. *La fonction* s *désigne la fonction successeur.*

$$\begin{array}{ll} (1) & 0 * x \to 0 \\ (2) & s(x) * y \to (x * y) + y \\ (3) & x + 0 \to x \\ (4) & x + s(y) \to s(x + y) \end{array}$$

Définition 1.3 *(Relation de réécriture) Soit* $\mathcal{R}$ *un système de réécriture. Pour tous termes* $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \to_R t$ *s'il existe une règle de réécriture* $l \to r \in \mathcal{R}$, *une position* $p \in \mathcal{P}os(s)$ *et une substitution* σ *tels que* $l\sigma = s|_p$ *et* $t = s[r\sigma]_p$.

Exemple 1.3 *Soit* t *le terme* $s(s(0)) * s(s(0))$ *et* $\mathcal{R}$ *le système de l'exemple 1.2. Sur* t *seule la règle* (2) *de* $\mathcal{R}$ *peut être appliquée: à la position* ϵ *de* t *on a* $t = (s(x) * y)\sigma$ *et* $\sigma = \{x \mapsto s(0), y \mapsto s(s(0))\}$. *Le redex*[2] *est donc* $t|_\epsilon$ *soit* t *lui-même. Le terme* t *est réductible en* $t_1 = t[((x * y) + y)\sigma]_\epsilon$, *i.e.* $t_1 = (s(0) * s(s(0))) + s(s(0))$. *Nous noterons cette réduction de la façon suivante:*

$$\underline{s(s(0)) * s(s(0))} \to_R^{(2)} (s(0) * s(s(0))) + s(s(0))$$

où le redex réduit à cette étape est souligné, et $\to_R$ *a pour exposant le numéro de la règle appliquée. Nous donnons maintenant la réécriture du terme* t *jusqu'à sa forme normale.*

1. en pratique nous nous restreindrons à des ensembles finis de règles de réécriture.
2. redex désigne le sous-terme qui satisfait au membre gauche d'une règle.

$$
\begin{aligned}
&\underline{s(s(0)) * s(s(0))} \to_R^{(2)}\\
&(\underline{s(0) * s(s(0))}) + s(s(0)) \to_R^{(2)}\\
&((\underline{0 * s(s(0))}) + s(s(0))) + s(s(0)) \to_R^{(1)}\\
&\underline{(0 + s(s(0))) + s(s(0))} \to_R^{(4)}\\
&s(\underline{(0 + s(s(0))) + s(0)}) \to_R^{(4)}\\
&s(s(\underline{(0 + s(s(0))) + 0})) \to_R^{(3)}\\
&s(s(\underline{0 + s(s(0))})) \to_R^{(4)}\\
&s(s(s(\underline{0 + s(0)}))) \to_R^{(4)}\\
&s(s(s(s(\underline{0 + 0})))) \to_R^{(3)}\\
&s(s(s(s(0))))
\end{aligned}
$$

La réécriture s'achève sur le terme $s(s(s(s(0))))$ en forme normale. On a donc $s(s(0)) * s(s(0)) \to^!_{\mathcal{R}} s(s(s(s(0))))$. Nous venons de montrer que $2*2 = 4$! Il faut remarquer que la réduction que nous venons de présenter n'est pas la seule possible. Par exemple, soit t_1 le premier terme obtenu par réécriture, i.e.

$$s(0) * s(s(0))) + s(s(0)).$$

Il existe plusieurs façons de réduire t_1: il existe dans le même terme deux redex à des positions différentes et réductibles par des règles distinctes. Le terme t_1 peut être réduit par la règle (4) à la position 1, comme précédemment, mais également par la règle (4) à la position ϵ, i.e.

$$\underline{(s(0) * s(s(0))) + s(s(0))} \to_R^{(4)} s((s(0) * s(s(0))) + s(0))$$

Sur ce dernier terme, là encore plusieurs règles peuvent s'appliquer à des positions différentes.

La réécriture est également applicable à un ensemble de termes. Pour un système de réécriture $\mathcal{R}$ et un ensemble de termes E, nous définissons maintenant l'ensemble de $\mathcal{R}$-descendants de E, noté $\mathcal{R}^*(E)$, qui est l'ensemble des termes accessibles par $\to_R^*$ à partir des termes de E, ainsi que l'ensemble des $\mathcal{R}$-formes normales de E, noté $\mathcal{R}^!(E)$, qui est l'ensemble des formes normales accessibles par $\to_R^*$ à partir des termes de E.

Définition 1.4 *($\mathcal{R}$-descendants et $\mathcal{R}$-formes normales) Soit $\mathcal{R}$ un système de réécriture et E un ensemble (un langage) de termes.*

$$\mathcal{R}^*(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ t.q. } s \to_{\mathcal{R}}^* t\}.$$

$$\mathcal{R}^!(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ t.q. } s \to_{\mathcal{R}}^! t\}.$$

Terminaison et confluence

Un système de réécriture peut présenter plusieurs propriétés [DJ90] parmi lesquelles la *confluence* qui garantit que deux exécutions sur les mêmes données mènent au même résultat si les exécutions terminent. La terminaison pour toute exécution est assurée si l'ensemble des règles de réécriture est *nœterien*. Dans ce cas, la réécriture d'un terme aboutit systématiquement à une forme normale en un nombre fini de pas ; de plus, cette forme est unique si le système est confluent. Avec les outils classiques de la réécriture, ces propriétés sont généralement prérequises pour prouver d'autres propriétés sur des systèmes de réécriture ; or le problème de la terminaison reste indécidable dans le cas général. Il existe un ensemble de critères de terminaison, fondés généralement sur les *bons ordres* permettant d'assurer la terminaison de certains sous-ensembles de systèmes [DJ90]. Cependant, cette indécidabilité, ainsi que le fait que certains systèmes de réécriture intéressants ne soient pas terminants, amène à rechercher d'autres critères de validation. C'est dans cette optique que nous allons nous intéresser dans le chapitre 2 à l'ensemble des états atteints afin d'être en mesure de prouver que certains programmes ne peuvent entrer dans des états *critiques*.

1.1.3 Systèmes de réécriture conditionnels

Il est possible d'étendre les systèmes de réécriture en ajoutant des conditions sur les règles de réécriture. Un *système de réécriture conditionnel* est un *système de réécriture* dont les règles sont soumises à des conjonctions de conditions exprimées sous la forme d'équations de réécriture [BN98]. Les conditions peuvent se présenter sous différentes formes :

- $l \leftrightarrow^* r$: on peut passer de l à r en utilisant les règles de réécriture de manière bidirectionnelle.
- $l \downarrow r$: il est possible de réécrire les deux termes l et r en un même terme en un nombre fini de règles
- $l \downarrow^! r$: les deux termes l et r peuvent être réécrits en une même forme normale
- $l \rightarrow^* r$: il est possible de réécrire l en r
- $l \rightarrow^! r$: r est une forme normale de l

Cet ensemble de types de conditions n'est pas exhaustif (il en existe également qui imposent des conditions sur la structure des termes). Généralement, dans un but d'homogénéité, il est préférable de n'utiliser qu'un seul type de condition pour décrire un programme. Les conditions portent sur des termes dont les variables sont un sous-ensemble des termes de la règle de réécriture associée. Par conséquent, une règle du type : $l \rightarrow r$ if $s \downarrow t$, vérifie nécessairement : $\mathrm{Var}(l) \supseteq \mathrm{Var}(r), \mathrm{Var}(l) \supseteq \mathrm{Var}(s), \mathrm{Var}(l) \supseteq \mathrm{Var}(t)$.

Exemple 1.4 *Voici un exemple simple de système de réécriture conditionnel qui étend l'exemple 1.3 en ajoutant les tests de parité.*
$\mathcal{F} = \{true : 0, false : 0, 0 : 0, odd : 2, even : 2, s : 1, + : 2, * : 2\}$, *et* $x, y \in \mathcal{X}$.

$$
\begin{array}{lllll}
(1) & 0 * x \rightarrow 0 & & \\
(2) & s(x) * y \rightarrow (x * y) + y & & \\
(3) & x + 0 \rightarrow x & & \\
(4) & x + s(y) \rightarrow s(x + y) & & \\
(5) & even(s(x)) \rightarrow true & if & odd(x) \rightarrow^* true \\
(6) & odd(s(x)) \rightarrow true & if & even(x) \rightarrow^* true \\
(7) & odd(0) \rightarrow false & & \\
(8) & even(0) \rightarrow true & &
\end{array}
$$

Pour pouvoir réécrire le terme $odd(s(s(s(x))))$ *d'après la règle* (6) *en true, il faut considérer les réécriture de* $even(s(s(x)))$ *et montrer que ce terme peut se réécrire en true.*

1.2 Automates d'arbres et langages réguliers

Les *automates d'arbres* constituent un outil permettant de représenter, de manière finie, des ensembles réguliers d'arbres et, par conséquent, de termes. Les propriétés de ces automates sont décrites dans [CDG+97].

Un *automate d'arbres* est un quadruplet $\mathcal{A} =< \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta >$, où $\mathcal{Q}$ est un ensemble fini de symboles d'arité 0 qui sont les *états* de l'automate, $\mathcal{Q}_f$ est un ensemble d'états terminaux et Δ est un ensemble de *transitions normalisées*. Une transition normalisée est une règle de réécriture où le terme de gauche est de hauteur 1 et dont les feuilles sont des états de l'automate et le terme de droite est un état de l'automate ($f(q_1, q_2, \ldots, q_n) \rightarrow q$ est un exemple de transition normalisée).
La relation de réécriture induite par Δ est notée $\rightarrow_\Delta$. Un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ est dit reconnu par l'état q de l'automate $\mathcal{A}$ si $t \rightarrow_\Delta^* q$. Un terme est reconnu par l'automate $\mathcal{A}$ s'il est reconnu par un état de $\mathcal{Q}_f$. Le langage reconnu par $\mathcal{A}$ est donc $\mathcal{L}(\mathcal{A}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \text{ tel que } t \rightarrow_\Delta^* q\}$. Par extension, on nomme *langage reconnu par l'état* q_i ou $\mathcal{L}(\mathcal{A}, q_i)$ l'ensemble des termes qui peuvent se réécrire en l'état q_i de l'automate. On nomme *langage régulier* (ou *ensemble régulier de termes*) un ensemble de termes reconnaissable par un automate d'arbre.

Exemple 1.5 *Nous donnons ici l'exemple de l'automate d'arbres* A_0 *reconnaissant l'ensemble des "listes plates non-vides composées de a et de b". On a* $A_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, *où* $\mathcal{F} = \{cons : 2, a : 0, b : 0, nil : 0\}$, $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\mathcal{Q}_f = \{q_0\}$ *et*

$$\Delta = \begin{cases} [b]lr(1) & cons(q_2, q_1) \rightarrow q_0 \\ (2) & nil \rightarrow q_1 \\ (3) & cons(q_2, q_1) \rightarrow q_1 \\ (4) & a \rightarrow q_2 \\ (5) & b \rightarrow q_2 \end{cases}$$

Dans cet exemple, on a $\mathcal{L}(A_0, q_2) = \{a, b\}$, $\mathcal{L}(A_0, q_1) = \{nil, cons(x, y)$ *tel que* $x \in \mathcal{L}(A_0, q_2)$ *et* $y \in \mathcal{L}(A_0, q_1)\}$, *ou encore l'ensemble*
$\{nil, cons(a, nil), cons(b, nil), cons(a, cons(a, nil)), cons(a, cons(b, nil)), \dots\}$, *ou en utilisant la représentation usuelle des listes dans les langages fonctionnels ou logiques* $\{[], [a], [b], [a, b], [a, a], [b, a, b, b] \dots\}$, *c'est à dire l'ensemble des listes plates formées de "a" et de "b". Enfin,*

$$\mathcal{L}(A_0, q_0) = \mathcal{L}(A_0) = \{cons(x, y) \ | \ x \in \mathcal{L}(A_0, q_2) \text{ et } y \in \mathcal{L}(A_0, q_1)\}$$

c'est à dire l'ensemble des listes plates constituées de a et de b, non-vides.

Montrons comment le terme $cons(b, cons(a, nil))$ *est reconnu par l'automate* A_0. *Par commodité, les transitions sont numérotées et le redex en passe d'être réduit par une transition de* Δ *est souligné:*

$$\begin{array}{l} cons(b, cons(\underline{a}, nil)) \rightarrow_{\Delta}^{(4)} \\ cons(b, cons(q_2, \underline{nil})) \rightarrow_{\Delta}^{(2)} \\ cons(b, \underline{cons(q_2, q_1)}) \rightarrow_{\Delta}^{(3)} \\ cons(\underline{b}, q_1) \rightarrow_{\Delta}^{(5)} \\ \underline{cons(q_2, q_1)} \rightarrow_{\Delta}^{(1)} q_0. \end{array}$$

Comme $q_0 \in \mathcal{Q}_f$, *le terme est reconnu par l'automate. On peut remarquer que dans cet exemple la dérivation n'est pas unique et peut mener vers un terme irréductible ou vers un état n'appartenant pas à* $\mathcal{Q}_f$.

Les automates d'arbres vont nous permettre de représenter les ensembles de termes que peut atteindre un système de réécriture à partir d'un ensemble E de termes initiaux. Le fait qu'ils soient eux-mêmes définis par une relation de réécriture est un avantage supplémentaire pour en faire des outils adéquats.

Chapitre 2

Termes accessibles dans les systèmes de réécriture non conditionnels

Dans ce chapitre nous abordons la problématique des termes accessibles dans les système de réécriture non conditionnels. Dans un premier temps, nous dressons la liste des cas ou $\mathcal{R}^*(E)$ est descriptible exactement à l'aide d'un automate d'arbre, puis nous discutons de la nécessité d'effectuer des approximations. Nous présentons alors l'algorithme de construction par complétion d'un automate qui reconnaît une approximation de l'ensemble $\mathcal{R}^*(E)$, ainsi que les propriétés et notations relatives à la méthode qui seront utiles pour le cas des systèmes conditionnels.

2.1 Langages de termes accessibles

2.1.1 Définition

Soit $\mathcal{T}(\mathcal{F}, X)$ un alphabet de termes et $\mathcal{R}$ un système de réécriture sur cet alphabet. Soit un ensemble $E \in \mathcal{T}(\mathcal{F})$. On cherche à déterminer l'ensemble $\mathcal{R}^*(E)$ quelles que soient les propriétés du système de réécriture. Dans la mesure où nous utilisons des automates d'arbres comme représentation, nous ne sommes pas en mesure de reconnaître $\mathcal{R}^*(E)$ lorsque cet ensemble n'est pas régulier.

L'objectif est de prouver que le système décrit par $\mathcal{R}$ ne peut amener à des configurations (ou états) *critiques*, cela revient à montrer la non-appartenance de certains termes critiques à $\mathcal{R}^*(E)$. Dans cette optique, lorsque $\mathcal{R}^*(E)$ n'est pas régulier, il est alors intéressant de reconnaître un sur-ensemble régulier $approx(\mathcal{R}, E) \supseteq \mathcal{R}^*(E)$. En effet, si un terme critique n'apparaît pas dans une telle sur-approximation, alors il n'appartient pas à $\mathcal{R}^*(E)$.

2.1.2 Langages réguliers de termes accessibles

Il existe des cas où $\mathcal{R}^*(E)$ est régulier. L'exemple le plus simple est lorsque cet ensemble est fini, auquel cas la construction de l'automate est triviale. Dans le cas général, il faut trouver une construction de l'automate en fonction du système de réécriture et des termes *en entrée* [GT95]. Les ensembles de ce type recensés à ce jour sont ceux où E est régulier et où $\mathcal{R}$ est soit :

- un système de réécriture clos [Bra69, DT90],
- un système de réécriture linéaire à droite et monadique [Sal88], où monadique signifie que le membre droit de chaque règle de $\mathcal{R}$ est soit une variable, soit un terme $f(t_1, \ldots, t_n)$ où $f \in \mathcal{F}$, $arite(f) = n$, et t_i, $i = 1, \ldots, n$ est une variable ou une constante,
- un système de réécriture linéaire et semi-monadique [CDGV91], où semi-monadique signifie que le membre droit de chaque règle de $\mathcal{R}$ est soit une variable, soit un terme $f(t_1, \ldots, t_n)$ où $f \in \mathcal{F}$, $arite(f) = n$, et $\forall i = 1, \ldots, n$, t_i est une variable ou un terme clos,
- un système "décroissant" [Jac96], où décroissant signifie que le membre droit de chaque règle de $\mathcal{R}$ est soit une variable, soit un terme $f(t_1, \ldots, t_n)$ où $f \in \mathcal{F}$, $arite(f) = n$, et $\forall i = 1, \ldots, n$, t_i est une variable, un terme clos, ou un terme dont les variables n'apparaissent pas dans le membre gauche.

Cependant, pour un langage régulier E, $\mathcal{R}^*(E)$ n'est pas nécessairement régulier, même si $\mathcal{R}$ est un système linéaire confluent et terminant [GT95]. D'autre part, si on oublie l'une des restriction des systèmes décroissants, alors $\mathcal{R}^*(E)$ n'est pas nécessairement régulier [Jac96].

2.2 approximation régulière des états accessibles

Dans la perspective de calculer $\mathcal{R}^*(E)$ pour des systèmes de réécriture représentant des programmes, la classe des systèmes décrits ci-dessus est bien trop restrictive. Il est alors intéressant de définir une *approximation* de $\mathcal{R}^*(E)$, c'est à dire un sur-ensemble régulier de $\mathcal{R}^*(E)$ pour tout ensemble E régulier et $\mathcal{R}$ linéaire à gauche[1].

2.2.1 Introduction

Puisque nous ne sommes pas en mesure, dans le cas général, de décrire de manière finie à l'aide d'automates d'arbres, l'ensemble des états atteints par un programme, on s'attache à en décrire un sur-ensemble régulier. Les

1. En pratique il est possible de lever cette restriction en introduisant des vérifications supplémentaires [GVTT01b]

automates d'arbres possèdent des similitudes avec les systèmes de réécriture qui peuvent être exploités à cette fin. Dans cette optique, il est nécessaire de définir des méthodes d'approximation par sur-ensemble pour ces systèmes, ceci a été réalisé par T.Genet [Gen98].

2.2.2 Réécriture de langages de termes

Soit $\mathcal{L}$ un langage sur un alphabet de termes clos $\mathcal{T}(\mathcal{F})$. Soit $\mathcal{R}$ un système de réécriture sur $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Considérons une règle $l \to r$ de $\mathcal{R}$.

Définition 2.1 *Une* substitution de langage *est une application* $\sigma_{\mathcal{L}} : \mathcal{X} \mapsto \mathcal{P}art(\mathcal{T}(\mathcal{F}, \mathcal{X}))$*, où* $\mathcal{P}art(\mathcal{T}(\mathcal{F}, \mathcal{X}))$ *désigne l'ensemble des parties de* $\mathcal{T}(\mathcal{F}, \mathcal{X})$*. On peut l'étendre à un morphisme* $\sigma_{\mathcal{L}} : \mathcal{T}(\mathcal{F}, \mathcal{X}) \mapsto \mathcal{P}art(\mathcal{T}(\mathcal{F}, \mathcal{X}))$*.*

Définition 2.2 *(Relation de réécriture de langage) Soit* $\mathcal{R}$ *un système de réécriture. Pour tous langages* $\mathcal{L}_s, \mathcal{L}_t \subset \mathcal{T}(\mathcal{F}, \mathcal{X}), \mathcal{L}_s \to_{\mathcal{R}} \mathcal{L}_t$ *s'il existe une règle de réécriture* $l \to r \in \mathcal{R}$*, une position* $p \in \mathcal{P}os$ *et une substitution* $\sigma_{\mathcal{L}}$ *tels que* $\forall s \in \mathcal{L}_s, \forall l_i \in l\sigma_{\mathcal{L}}, l_i = s|_p$ *et* $\forall t \in \mathcal{L}_t, \exists s \in \mathcal{L}_s$ *tel que* $t = s[r\sigma]_p$*.*

2.2.3 Algorithme

Nous présentons une méthode de calcul de $\mathcal{R}^*(E)$ par sur-approximation. Elle consiste en la construction incrémentale d'un automate reconnaissant $approx(\mathcal{R}, E)$ par complétion.

Construction de $\mathcal{R}^*(E)$ par complétion

Soit $\mathcal{A}$ un automate sur $\mathcal{T}(\mathcal{F})$, Soit q_i l'état reconnaissant le langage $\mathcal{L}(\mathcal{A}, q_i)$, pour tout $1 \leq i \leq n$. On note $\sigma_{\mathcal{L}}(x_1 : q_1, x_2 : q_2, \ldots, x : q_n)$ la substitution qui à un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $var(t) \subset \{x_i \mid 1 \leq i \leq n\}$, associe le langage :

$$t\sigma_{\mathcal{L}} = \{r \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \mid \forall 1 \leq i \leq n, \exists s_i \in \mathcal{L}(\mathcal{A}, q_i) \text{ t.q. } t[x_1/s_1][x_2/s_2]\ldots[x_n/s_n] = r\}$$

Dans le système de réécriture $\mathcal{R}$, si E désigne un ensemble régulier d'entrées et $\mathcal{A}$ un automate reconnaissant E, il est possible de construire incrémentalement l'ensemble $\mathcal{R}^*(E)$ avec le principe suivant décrit par T. Genet [Gen98] :
s'il existe $s, t \in \mathcal{T}(\mathcal{F}), l \to r \in \mathcal{R}$, un état $q \in \mathcal{Q}$ tel que $s \to_{\Delta}^* q$, $t \not\to_{\Delta}^* q$ et une substitution de terme σ tel que $s = l\sigma, t = r\sigma$, alors on génère un nouvel automate $\mathcal{A}'$ tel que l'état q reconnaisse également le terme t.

Dans ce cas, on a $s \to_{\mathcal{R}} t$, $s \to_{\Delta}^* q \in \mathcal{Q}$ mais l'on a pas $t \to_{\Delta}^* q \in \mathcal{Q}$. L'idée consiste à ajouter cette transition dans l'automate, après l'avoir normalisée (la normalisation de règle est décrite en 2.2.3).

$$\begin{tikzcd} l\sigma \arrow[r, "\Delta"', "*"] \arrow[d, "\mathcal{R}"'] & q \\ r\sigma \arrow[ur, bend right, "/" marking, "\Delta^{*}"'] & \end{tikzcd}$$

FIG. 2.1 – *Un* paire critique *entre* Δ *et* $\mathcal{R}$. *On souhaite que l'état* q *de l'automate reconnaisse tous les descendants par* $\mathcal{R}$ *des termes qu'il reconnaît déjà. On résout donc la paire critique en ajoutant* $r\sigma \to_{\Delta} q$*, après normalisation, aux transitions de* $\mathcal{A}$

Nous avons donc défini une procédure de construction par complétion. Elle génère une suite $(\mathcal{A}_i)_{i \in \mathbb{N}}$ avec $\mathcal{A}_0$ l'automate reconnaissant les entrées du programme. La complétion est terminée lorsque qu'une nouvelle étape ne modifie pas l'automate, c'est à dire $\exists n \in \mathbb{N}$ tel que $\mathcal{A}_{n+1} = \mathcal{A}_n$ alors dans ce cas, $\mathcal{A}_n$ reconnaît un sur ensemble régulier de $\mathcal{R}^*(E)$.

Cependant, une telle construction a diverge souvent car les normalisation génèrent de nouveaux états qui sont à leur tour générateurs de paires critiques. De plus $\mathcal{R}^*(E)$ n'est pas nécessairement régulier.

Approximation par normalisation

Avant toute chose, nous donnons l'algorithme classique permettant de ramener tout ensemble de transitions Δ à un ensemble de transitions normalisées Δ' reconnaissant le même langage, i.e. pour tout terme $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ et tout état $q \in \mathcal{Q}$, $s \to_{\Delta'}^{*} q$ si et seulement si $s \to_{\Delta}^{*} q$.

Définition 2.3 *Soit* $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ *un automate d'arbres,* $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$*, et* $q \in \mathcal{Q}$*. L'ensemble* $Norm(s \to q)$ *est défini par:*

- *si* $s = q \in \mathcal{Q}$*, alors* $Norm(s \to q) = \emptyset$*,*
- *si* $s \in \mathcal{Q}$ *et* $s \neq q$*, alors* $Norm(s \to q) = \{s \to q\}$*,*
- *si* $s = f(t_1, \ldots, t_n)$ *avec* $f \in \mathcal{F}^n$*, alors*

$$Norm(s \to q) = \{f(q_1, \ldots, q_n) \to q\} \cup \bigcup_{i=1}^{n} Norm(t_i \to q_i)$$

où $q_i = t_i$ *si* $t_i \in \mathcal{Q}$ *et* q_i *est un nouvel état, sinon.*

Exemple 2.1 *Soit* $\mathcal{F} = \{f, g, a\}$ *et* $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$*, où* $\mathcal{Q} = \{q_0, q_1\}$*,* $\mathcal{Q}_f = \{q_0\}$*, et* $\Delta = \{f(q_1) \to q_0, g(q_1, q_1) \to q_1, a \to q_1\}$.

$Norm(q_1 \to q_0) = \{q_1 \to q_0\}$. $Norm(f(g(q_1, f(a))) \to q_0) = \{f(q_2) \to q_0, g(q_1, q_3) \to q_2, f(q_4) \to q_3, a \to q_4\}$, *où* q_2, q_3, q_4 *sont des nouveaux états.*

Cette méthode basique de normalisation génère des nouveaux état qui reconnaissent les sous-termes de $r\sigma$. Il est possible d'endiguer la création de nouveau états dans l'automate en spécifiant une stratégie de normalisation particulière. L'idée est de réutiliser dans la mesure du possible des états déjà existants, même si cette opération étend leur langage reconnu. Lors de cette normalisation, en incrémentant les langages reconnus par certains états, le langage reconnu par l'automate est étendu d'au moins le terme $r\sigma$ mais aussi de nouveaux termes. Ces nouveaux termes ne font pas nécessairement partie de $\mathcal{R}^*(E)$. Ainsi, nous limitons la création de nouveau états en reconnaissant des termes supplémentaires, ce qui constitue la sur-approximation lors du calcul de $\mathcal{R}^*(E)$.

Exemple 2.2 $\mathcal{F} = \{f : 1, h : 1, a : 0\}$, $\mathcal{X} = \{x\}$. *Soit le système de réécriture* $\mathcal{R}$ *:*

$$\begin{array}{lll} (1) & f(x) & \to f(f(h(x))) \\ (2) & h(h(x)) & \to a \end{array}$$

Soit l'automate des entrées $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, *avec* $\mathcal{Q} = \{q_0, q_1, q_2\}$, $\mathcal{Q}_f = \{q_2\}$, *et* $\Delta = \{a \to q_0, h(q_0) \to q_1, h(q_1) \to q_0, f(q_0) \to q_2\}$. *Cet automate reconnaît* $E = \mathcal{L}(\mathcal{A}) = \{f(h^{2n}(a)) \mid n \in \mathbb{N}\}$

Il existe une paire critique :

$$f(a) \to_\Delta q_2, f(a) \to_\mathcal{R} f(f(h(a))) \text{ et } f(f(h(a))) \not\to_\Delta q_2$$

Il faut alors normaliser la transition $f(f(h(a))) \to_\Delta q_2$. *Utilisons déjà* $h(a) \to_\Delta q_1$, *normalisons ensuite en ajoutant* $f(q_1) \to q_2$, *et* $f(q_2) \to q_2$. *On obtient alors* Δ *:*

$$\begin{array}{ll} a & \to q_0 \\ h(q_0) & \to q_1 \\ h(q_1) & \to q_0 \\ f(q_0) & \to q_2 \\ f(q_1) & \to q_2 \\ f(q_2) & \to q_2 \end{array}$$

L'automate reconnaît alors $\mathcal{L}(\mathcal{A}) = \{f^m(h^n(a)) \mid n \in \mathbb{N}, m \in \mathbb{N} \setminus \{0\}\}$. *Il n'existe plus de paire critique donc* $\mathcal{R}^*(E) \subseteq \mathcal{L}(\mathcal{A})$ *mais cependant,* $f(h(a)) \in \mathcal{L}(\mathcal{A})$ *alors que* $f(h(a)) \notin \mathcal{R}^*(E)$, *d'où* $\mathcal{R}^*(E) \not\supset \mathcal{L}(\mathcal{A})$. *Cette normalisation à permis d'effectuer une sur-approximation de* $\mathcal{R}^*(E)$ *en un nombre limité d'opérations. La méthode a consisté ici à «replier» les appels récursifs de la fonction* f *sur l'état* q_2.

2.2.4 Algorithme et langages de termes

Un état de l'automate ne reconnaît pas un unique terme mais un langage de termes. Par conséquent il n'est pas envisageable de considérer les paires critiques terme à terme, mais plutôt de chercher à réécrire une famille de termes qui satisfont un même membre gauche de règle. En pratique, pour trouver les paires critiques d'une règle $l \to r \in \mathcal{R}$, on cherche une substitution de langage $\sigma_{\mathcal{L}}(x_1 : q_1, x_2 : q_2, \ldots, x : q_n)$ avec $\{x_1, x_2, \ldots x_n\} = var(l)$ et $\forall t \in l\sigma_{\mathcal{L}}, t \to_{\Delta}^{*} q \in \mathcal{Q}$ tel que $r\sigma_{\mathcal{L}} \not\to_{\Delta}^{*} q$. En effet, si $l\sigma_{\mathcal{L}} \subset \mathcal{R}^{*}(E)$ alors $r\sigma_{\mathcal{L}} \subset \mathcal{R}^{*}(E)$, il faut donc ajouter, après normalisation, la transition[2] $r\sigma_{\mathcal{L}} \to_{\Delta}^{*} q \in \mathcal{Q}_f$.

Pour chaque règle de $\mathcal{R}$, il faut alors rechercher les substitutions des variable par des états de $\mathcal{A}$ qui permettent de réécrire par $\to_{\Delta}$ le membre gauche de la règle en un état final de $\mathcal{A}$ puis vérifier si, pour cette substitution, le membre droit est effectivement reconnu par l'automate.

2.2.5 Timbuk, une implémentation

Cette méthode a mené à l'implémentation de la bibliothèque d'automates d'arbres Timbuk [GVTT01b]. Le principe est de reconnaître, à partir d'un automate et d'un système de réécriture, l'ensemble des termes accessibles, soit $\mathcal{R}^{*}(E)$. L'utilisateur intervient principalement lors des normalisations pour "diriger" l'approximation. En effet, il est intéressant d'avoir un certaine précision au abords des termes critiques et une approximation large pour les ensemble de termes moins importants. Notons également que la forme de l'automate des entrées du système de réécriture permet elle aussi d'effectuer une partie de l'approximation. Pour plus d'informations sur le fonctionnement et les fonctionnalités de Timbuk, se reporter à [GVTT01a]

2.3 Conclusion et perspectives

2.3.1 Bilan

Nous disposons d'un outil, qui s'il présente des inconvénients de complexité, reste utilisable dans le cas général dans la mesure où il n'impose aucune restriction sur les systèmes à étudier. Nous somme donc à même, à partir d'un protocole ou d'un programme fonctionnel traduit en système de réécriture, de vérifier que certaines configurations ne peuvent être atteintes. Par exemple, dans le cas d'un protocole cryptographique, il est possible de vérifier que des états non sûrs sont incessibles, et donc d'en valider la conception.

2. Note : $r\sigma_{\mathcal{L}}$ représente un langage de termes dans $\mathcal{T}(\mathcal{F}, \mathcal{X})$ mais aussi un unique terme du point de vue de la réécriture $\to_{\Delta}$ car $r\sigma_{\mathcal{L}} \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$

2.3.2 Perspectives

À partir de ce point, il est légitime de s'interroger sur les extensions de la méthode. Une idée est de tenter d'exploiter ces résultats pour permettre de travailler sur des formalisations sous forme de systèmes de réécriture conditionnels, afin de permettre une plus grande souplesse dans le formalisme à partir duquel nous souhaitons prouver les propriétés.

Chapitre 3

Conditions et méthode de traduction

3.1 Problématique des états accessibles dans la réécriture conditionnelle

Dans un système de réécriture conditionnel, l'évaluation des états accessible est compliquée par le fait que, pour évaluer les conditions, il est nécessaire de connaître les états accessibles à partir de certains termes. Une règle du type $l \to r$ if $s \downarrow t$ indique que pour une instance de l (i.e. une substitution σ des variables du terme l), l'instance correspondante de r est accessible si et seulement si l'intersection de l'ensemble des états accessibles à partir des termes $s\sigma$ et de $t\sigma$ est non vide ($\mathcal{R}^*(t\sigma) \cap \mathcal{R}^*(s\sigma) \neq \emptyset$). De manière générale, l'évaluation d'une condition ramène à un problème d'accessibilité qui doit être à son tour résolu en tenant compte des règles conditionnelles.

Exemple 3.1 *Voici un exemple simple de système de réécriture conditionnel définissant partiellement la fonction member, où* $\mathcal{F} = \{true : 0, false : 0, nil : 0, cons : 2, member : 2\}$, *et* $x, y, l \in \mathcal{X}$.

$$\begin{array}{lll} (1) & member(x, cons(y,l)) & \to true \quad \textit{if } x \downarrow y \\ (2) & member(x, cons(y,l)) & \to true \quad \textit{if } member(x,l) \downarrow true \\ (3) & member(x, nil) & \to false \end{array}$$

On souhaite évaluer le terme :

$$member(true, cons(false, cons(true, nil))) \to^? true$$

D'après (2) il faut précédement évaluer la condition :

$$member(true, cons(true, nil)) \downarrow^? true$$

On s'interroge alors sur l'applicabilité de la règle (1) :

$$member(true, cons(true, nil) \rightarrow^? true$$

qui nécessite à son tour l'évaluation de :

$$true \downarrow^? true$$

cette derniere condition étant trivialement vérifiée :

$$member(true, cons(true, nil) \downarrow true$$

est également vérifiée, par conséquent :

$$member(true, cons(false, cons(true, nil))) \rightarrow true$$

3.2 Traduction vers un système non conditionnel

Une méthode envisageable pour le calcul des $\mathcal{R}$-descendants d'un ensemble de termes dans un système de réécriture conditionnel consiste à tenter de traduire un tel système en un équivalent non conditionnel, puis d'utiliser la méthode existante présentée en 2.2.3. Le problème de la traduction a déja été abordé par [AK96] dans le cadre de la réécriture concurrente, ou dans [DOS87] pour la démonstration de certaines propriétés. Cependant, toutes les transformations envisagées requièrent des propriétés contraignantes sur le système. Ces propriétés sont par exemple, un ordre sur les règles, des stratégies particulières de réécriture, la terminaison et la confluence du système. À notre connaissance, il n'existe pas de traduction telle que le système de réécriture non conditionnel résultat modélise exactement le système conditionnel considéré. Dans le cadre de notre travail, ces restrictions sont soit incompatibles avec la méthode (l'utilisation d'une stratégie de réécriture est incompatible avec l'algorithme présenté en 2.2.3), soit nous font sortir du cadre fixé au départ (la confluence du système est une propriété dont on souhaite s'affranchir). De plus, la plupart des méthodes présentées génèrent beaucoup de nouveaux termes et de règles, compliquant énormément la procédure de complétion. Ainsi, il apparait intéressant de définir une méthode de traduction suffisement générale pour être exploitable et suffisement simple pour ne pas compromettre la convergence de la méthode de complétion.

3.2.1 Forme des conditions et traductibilité

On se restreint pour cette partie à des conditions simple, c'est à dire une unique condition par règle. Notons qu'il est aisé de se ramener à ce cas en éclatant une règle possédant de multiples conditions en plusieurs règles comme sur l'exemple suivant :

Exemple 3.2

$$\begin{array}{llll}
l & \rightarrow r & \mathit{if} & c_1(s_1,t_1) \wedge c_2(s_2,t_2) \wedge \ldots \wedge c_n(s_n,t_n) \\
 & & \Downarrow & \\
l & \rightarrow f(l,1) & \mathit{if} & c_1(s_1,t_1) \\
f(l,1) & \rightarrow f(l,2) & \mathit{if} & c_2(s_2,t_2) \\
\vdots & \vdots & \vdots & \vdots \\
f(l,n-1) & \rightarrow f(l,n) & \mathit{if} & c_n(s_n,t_n) \\
f(l,n) & \rightarrow r & &
\end{array}$$

On «eclate» la règle à l'aide de la fonction f et d'un compteur qui indique le nombre de conditions restant à évaluer.

Types de conditions et implications

Les types de conditions que nous considérons ici sont les suivants :

– $s \downarrow t$, il est possible de joindre s et t en un terme commun. Ce type de condition est particulièrement intuitif dans le mesure où, dans le cadre de l'évaluation de fonction (ou de programmes) nous pouvons le comprendre de la manière suivante : le calcul de la valeur des fonctions (ou les sortie des sous-programmes) représentés par les termes s et t avec les entrées définies par la fonction de substitution choisie pour l'invocation de la règle, ont même résultat (ou un résultat commun dans le cadre du non-determinisme).

– $s \downarrow^! t$, il est possible de joindre s et t dans une même forme normale. Ceci nécessite que les termes s et t possèdent une forme normale, en d'autre termes, que le calcul de la valeur de s et de celle de t terminent tous deux sur un (au moins) résultat identique. Notons que si $s \downarrow t$, et que le terme de jonction possède une forme normale, alors elle constitue la jonction normale permettant de satisfaire la condition $s \downarrow^! t$. De plus, $s \downarrow^! t \Rightarrow s \downarrow t$.

– $s \rightarrow^* t$, il est possible d'obtenir t par réécriture de s. Ceci signifie que t est un descendant de s, il est possible de l'interpréter comme le fait que t doive être un résultat partiel ou final de l'évaluation de la fonction ou du sous-programme décrit par le terme s. Notons que $s \rightarrow^* t \Rightarrow s \downarrow t$.

– $s \xrightarrow{!}^* t$, t est une forme normale de s. Cela signifie que le résultat de l'évaluation de s, ou du moins un des résultats est le terme t. Notons que $s \xrightarrow{!}^* t \Rightarrow x \rightarrow^* t$ et $s \xrightarrow{!}^* t \Rightarrow x \downarrow^! t$

On obtient donc la hiérarchie des conditions suivante :

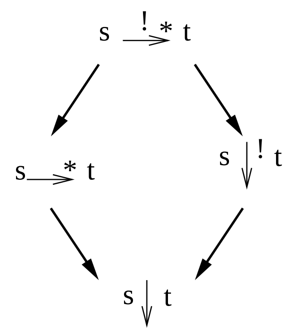

FIG. 3.1 – *Hiérarchie sur la validité des conditions*

Conditions traductibles

Afin de traduire une condition, la méthode intuitive est d'associer une fonction de condition à chaque règle conditionnelle. Le problème qui se pose est que dans le cadre d'une réécriture non conditionnelle, d'une part, il est impossible de différencier lors de substitutions des termes conditionnels de termes non conditionnels ; et d'autre part, étant donné qu'il est nécessaire d'effectuer un nombre indéterminé d'étapes de réécriture pour arriver à valider une condition, et que l'on s'interdit une stratégie particulière de réécriture, alors il est possible que les termes informatifs que l'on insérerait soient altérés par réécriture et ne permettent plus de retrouver le contexte de la règle conditionnelle initiale.

Exemple 3.3 *considérons une méthode de traduction naïve :*

$$\mathcal{R} = \{l \to r \text{ if } s \to^* t\} \quad \Rightarrow \quad \mathcal{R}' \begin{cases} l & \to c(r, s) \\ c(r, t) & \to r \end{cases}$$

supposons maintenant qu'il existe σ_1 et σ_2, substitutions, telles que $r\sigma_1 \to^* r\sigma_2$ et $s\sigma_1 \to^* t\sigma_2$ et $s\sigma_1 \not\to^* t\sigma_1, s\sigma_2 \not\to^* t\sigma_2$ alors la réécriture suivante est possible :

$$l\sigma_1 \to c(r\sigma_1, s\sigma_1) \to^* c(r\sigma_2, s\sigma_1) \to^* c(r\sigma_2, t\sigma_2) \to r\sigma_2$$

Or $s\sigma_2 \not\to^* t\sigma_2$, la méthode est donc invalide.

Exemple 3.4 *Imaginons maintenant une traduction plus forte pour les conditions de type $\downarrow$:*

$$\{l \to r \text{ if } s \downarrow t\} \quad \Rightarrow \quad \begin{cases} l & \to \boldsymbol{\downarrow}(l, s, t, r) \\ \boldsymbol{\downarrow}(l, x, x, r) & \to r \end{cases}$$

Considérons maintenant le système de réécriture suivant :

$$\begin{array}{lll}
F(x,y,z) & \rightarrow a \textit{ if } f(x,y) \downarrow f(x,z) & \\
f(x,y) & \rightarrow d & \textit{if } g(y) \downarrow h(y) \\
f(a,x) & \rightarrow f(a,a) & \\
g(b) & \rightarrow g(c) &
\end{array}$$

Il se traduit alors de la manière suivante :

$$\begin{array}{ll}
F(x,y,z) & \rightarrow\!\downarrow\!(F(x,y,z), f(x,y), f(x,z), a) \\
f(x,y) & \rightarrow\!\downarrow\!(f(x,y), g(y), h(y), d) \\
f(a,x) & \rightarrow a \\
g(b) & \rightarrow g(c) \\
h(b) & \rightarrow h(c) \\
\downarrow\!(f(x,y), z, z, d) & \rightarrow d
\end{array}$$

Considérons maintenant la réécriture des termes $f(a,b)$ et $f(a,c)$:

$$\begin{array}{ll}
f(a,b) & \rightarrow\!\downarrow\!(\underline{f(a,b)}, g(b), h(b), d) \\
 & \rightarrow\!\downarrow\!(f(a,a), \underline{g(b)}, h(b), d) \\
 & \rightarrow\!\downarrow\!(f(a,a), g(c), \underline{h(c)}, d) \\
 & \rightarrow\!\downarrow\!(f(a,a), g(c), h(c), d)
\end{array}$$

$$\begin{array}{ll}
f(a,c) & \rightarrow\!\downarrow\!(\underline{f(a,c)}, g(c), h(c), d) \\
 & \rightarrow\!\downarrow\!(f(a,a), g(c), h(c), d)
\end{array}$$

D'après ces résultats, évaluons le terme $F(a,b,c)$:

$$\begin{array}{lll}
F(a,b,c) & \rightarrow & \downarrow\!(F(a,b,c), \underline{f(a,b)}, f(a,c), a) \\
 & \rightarrow^{*} & \downarrow\!(F(a,b,c), \downarrow\!(f(a,a), g(c), h(c), d), \underline{f(a,c)}, a) \\
 & \rightarrow^{*} & \underline{\downarrow\!(F(a,b,c), \downarrow\!(f(a,a), g(c), h(c), d), \downarrow\!(f(a,a), g(c), h(c), d), a)} \\
 & \rightarrow & a
\end{array}$$

Malgré l'introduction des termes l et r comme sous-termes dans les termes intermédiaires conditionnels, dans le but d'essayer de conserver les substitutions, le système traduit présente une sur estimation des descendants du terme $F(a,b,c)$. En effet, il est impossible de vérifier, lors de l'étape de passage du terme conditionnel intermédiaire au résultat, que la jonction des deux termes conditionnels est un terme de l'alphabet de départ (et non un terme conditionnel). Ici, la condition $\downarrow(f(a,a), g(c), h(c), d)$ n'est pas vérifiée, cependant ce terme conditionnel permet la dernière réécriture et induit pas conséquent un résultat faux.

Des problèmes du type de celui présenté dans l'exemple ci-dessus se posent à chaque fois qu'il est nécessaire de vérifier, lors de l'étape de validation de la condition, deux termes non clos. Ceci ferme les perspectives de traduction exepté dans les cas où t est un terme clos (pas d'ambiguïté de réécriture).

3.2.2 Réécriture en terme clos et traduction

Condition de réécriture vers un terme clos

Afin de se dégager des problèmes de croisement des substitutions, nous nous limiterons aux conditions pour lesquelles ceci est impossible.

Définition 3.1 *Notons $s \rightarrow^{\circ}_{\mathcal{R}} t$ lorsque $s \rightarrow_{\mathcal{R}}$ et $t \in \mathcal{T}(\mathcal{F})$, et $s \rightarrow^{\circledast}_{\mathcal{R}} t$ lorsque $s \rightarrow^{*}_{\mathcal{R}} t$ et $t \in \mathcal{T}(\mathcal{F})$. Ceci définit la nouvelle condition : s se réécrit par un système de réécriture $\mathcal{R}$ en un terme clos (dans la règle) t.*

Méthode

À partir de ce type de conditions, définissons la méthode de traduction suivante :

$$
\begin{array}{llllll}
\mathcal{R} = \{ & (k) & l & \rightarrow & r & \text{if} \quad s \rightarrow^{\circledast} t \\
 & & & \Downarrow & & \\
\mathcal{R}' = \left\{ \begin{array}{l} \\ \\ \end{array} \right. & \begin{array}{l} (k_1) \\ (k_2) \end{array} & \begin{array}{l} l \\ c_k(x,t) \end{array} & \begin{array}{c} \rightarrow \\ \rightarrow \end{array} & \begin{array}{l} c_k(r,s) \\ x \end{array} &
\end{array}
$$

Chaque règle conditionnelle ajoute par conséquent une nouvelle fonction d'arité 2 et se traduit par 2 nouvelles règles. Soir $\mathcal{R}$ un système de réécriture conditionnel dont les conditions s'expriment par $\rightarrow^{\circledast}$, soit $\mathcal{R}'$ sa traduction.

Soit $\mathcal{F}' = \mathcal{F} \cup \{c_k : 2 \mid k \leq \mathcal{C}ard(\mathcal{R})\}$. Nommons $\mathcal{R}'$ le système de réécriture non conditionnel obtenu par la traduction à partir du système conditionnel $\mathcal{R}$. Nommons $\mathcal{R}'^{*}_{c-}(E)$ l'ensemble des termes accessibles de ce système, à partir de l'ensemble E, auquel on supprime tout les termes comportant au moins une occurrence d'un c_k :

$$\mathcal{R}'^{*}_{c-}(E) = \mathcal{R}'^{*}(E) - \{t | \exists p_{\in Pos}, k \in \mathbb{N}, t_1, t_2 \in \mathcal{T}(\mathcal{F}) \text{ tel que } t|_p = c_k(t_1, t_2)\}$$

Définition 3.2 *Soit $\mathcal{R}_{decond}$ l'ensemble des règles non conditionnelles de $\mathcal{R}$*

Définition 3.3 *Pour un système de réécriture conditionnel $\mathcal{R}$, on note $\xrightarrow{\downarrow n}_{\mathcal{R}}$ la relation définie par :*

- $\xrightarrow{\downarrow 0}_{\mathcal{R}} = \rightarrow_{\mathcal{R}_{decond}}$
- *$a \xrightarrow{\downarrow n+1}_{\mathcal{R}} b \Leftrightarrow a \xrightarrow{\downarrow 0}_{\mathcal{R}} b$ ou alors $\exists \sigma$ substitution, $p \in \mathcal{P}os(a)$ et $(l \rightarrow r$ if $s \downarrow t) \in \mathcal{R}$ tels que*
  $a|p = l\sigma, b = a[r]p$ et $\exists u$ tel que $s \xrightarrow{\downarrow n}_{\mathcal{R}} u$ et $t \xrightarrow{\downarrow n}_{\mathcal{R}} u$.

Théorème 3.1 $\forall t_1, t_2 \in \mathcal{T}(\mathcal{F}), t_1 \rightarrow^{*}_{\mathcal{R}} t_2 \Leftrightarrow t_1 \rightarrow^{*}_{\mathcal{R}'} t_2$. *La traduction est donc valide.*

Preuve Effectuons une récurence sur n pour montrer $\forall a, b \in \mathcal{T}(\mathcal{F}), a \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} b \Rightarrow a \rightarrow_{\mathcal{R}'} b$

- $a \xrightarrow{\downarrow 0}{}^{*}_{\mathcal{R}} b$ alors il existe une suite de termes $(t_i)_{1\leq i\leq k}$ telle que $a = t_1 \xrightarrow{\downarrow 0}_{\mathcal{R}} t_2 \xrightarrow{\downarrow 0}_{\mathcal{R}} \dots \xrightarrow{\downarrow 0}_{\mathcal{R}} t_n = b$. $\forall 1 \leq i \leq k, \exists ((j) l \rightarrow_{\mathcal{R}} r \in \mathcal{R}$ t.q. $\exists p \in \mathcal{P}os(t_i), \sigma$ substitution, $t_i|_p = l\sigma, t_i[r\sigma]p$. Or, la traduction de (j) est la règle (j) elle-même, donc $t_i \rightarrow_{\mathcal{R}'} t_{i+1}$. Ainsi, $a \rightarrow_{\mathcal{R}'} b$.
- supposons $\forall a, b \in \mathcal{T}(\mathcal{F}), a \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} b \Rightarrow a \rightarrow_{\mathcal{R}'} b$, alors $a \xrightarrow{\downarrow n+1}{}^{*}_{\mathcal{R}} b$ alors il existe une suite de termes $(t_i)_{1\leq i\leq k}$ telle que

$$a = t_1 \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_2 \xrightarrow{\downarrow n+1}_{\mathcal{R}} \dots \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_k = b$$

  Démontrons par induction sur i $a \rightarrow^*_{\mathcal{R}'} t_i$:

  - $a = t_1$ donc $a \rightarrow^*_{\mathcal{R}' t_1}$
  - supposons $a \rightarrow^*_{\mathcal{R}' t_i}$, on à $t_i \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{i+1}$, par définition cela signifie $\exists ((j) l \rightarrow r \text{ if } s \rightarrow^{\circledast} t) \in \mathcal{R}$ t.q. $\exists p \in \mathcal{P}os(t_i), \sigma$ substitution, $t_i|_p = l\sigma, t_i[r\sigma]p$, et $s\sigma \xrightarrow{\downarrow n}{}^{\circledast}_{\mathcal{R}} t$. Ceci implique $s\sigma \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} t$, ce qui par hypothèse de récurrence (sur n) nous permet de déduire : $s\sigma \rightarrow^*_{\mathcal{R}'} t$. Par conséquent, selon la méthode de traduction, on peut obtenir la réécriture suivante :

$$l\sigma \xrightarrow{(j_1)}_{\mathcal{R}'} c_j(r\sigma, s\sigma) \rightarrow^*_{\mathcal{R}'} c_j(r\sigma, t) \xrightarrow{(j_2)}_{\mathcal{R}'} r\sigma$$

    Ceci signifie $t_i \rightarrow^*_{\mathcal{R}'} t_{i+1}$, donc par hypothèse d'induction sur i, $a \rightarrow^*_{\mathcal{R}'} t_i \rightarrow^*_{\mathcal{R}'} t_{i+1}$. Donc $a \rightarrow^*_{\mathcal{R}'} t_{i+1}$

  Nous avons donc au final $a \rightarrow^*_{\mathcal{R}'} t_k = b$

La partie réciproque de la preuve n'est à ce jour pas terminée. □

Exemple 3.5 *Voici un exemple de traduction à partir d'un système de réécriture conditionnel. La fonction SetIn teste si l'ensemble des valeurs de la liste passée en premier argument se trouve dans la liste passée en second argument. Le système de réécriture vérifie la validité de SetIn.* $\mathcal{F} = \{true : 0, a : 0, b : 0, nil : 0, cons : 2, member : 2, SetIn : 2\}$, *et* $x, y, l, l_1, l_2 \in \mathcal{X}$.

(1)	$SetIn(cons(x, l_1), l_2)$	$\rightarrow SetIn(l_1, l_2)$	$if\ member(x, l_2) \rightarrow^{\circledast} true$
(2)	$SetIn(nil, l)$	$\rightarrow true$	
(3)	$member(x, cons(x, l))$	$\rightarrow true$	
(4)	$member(x, cons(y, l))$	$\rightarrow true$	$if\ member(x, l) \rightarrow^{\circledast} true$

Ce système se traduit de la manière suivante :

$$\begin{array}{lll}
(1_1) & SetIn(cons(x,l_1),l_2) & \to c_1(SetIn(l_1,l_2), member(x,l_2)) \\
(1_2) & c_1(x, true) & \to x \\
(2) & SetIn(nil, l) & \to true \\
(3) & member(x, cons(x,l)) & \to true \\
(4_1) & member(x, cons(y,l)) & \to c_4(true, member(x,l)) \\
(4_2) & c(x, true) & \to x
\end{array}$$

Un exemple de réécriture possible dans ce système :

$$\begin{array}{l}
t = \underline{SetIn(cons(a,cons(b,nil)),cons(b,cons(a,nil)))} \\
\xrightarrow{(1_1)} c_1(SetIn(cons(b,nil),cons(b,cons(a,nil))),\underline{member(a,cons(b,cons(a,nil)))}) \\
\xrightarrow{(4_1)} c_1(SetIn(cons(b,nil),cons(b,cons(a,nil))),c_4(true,\underline{member(a,cons(a,nil))})) \\
\xrightarrow{(3)} c_1(SetIn(cons(b,nil),cons(b,cons(a,nil))),\underline{c_4(true,true)}) \\
\xrightarrow{(4_2)} \underline{c_1(SetIn(cons(b,nil),cons(b,cons(a,nil))),true,true)} \\
\xrightarrow{(1_2)} \underline{SetIn(cons(b,nil),cons(b,cons(a,nil))} \\
\xrightarrow{(1_1)} c_1(\underline{SetIn(nil,cons(b,cons(a,nil)))},member(b,cons(b,cons(a,nil)))) \\
\xrightarrow{(2)} c(true,\underline{member(b,cons(b,cons(a,nil)))}) \\
\xrightarrow{(3)} \underline{c_1(true,true,true)} \\
\xrightarrow{(1_2)} true
\end{array}$$

Dans cet exemple, il y a jusqu'à 3 conditions en attente d'évaluation dans un même terme intermédiaire. Les termes ont tendance à se complexifier au fil des conditions ; toutefois, le résultat peut aussi bien être réécrit à l'intérieur du terme conditionnel avant vérification de la condition qu'après cette vérification. Cet exemple de réécriture n'illustre pas les conditions qui ne sont pas vérifiées, pour lesquelles on peut toujours effectuer des réécritures sur le résultat sans que ceci soit exploitable pour $\mathcal{R}^*(E)$.

Généralisation à la traduction d'une conjonction de conditions

Une règle peut être soumise à un ensemble de conditions. Il existe des méthodes pour se ramener à une seule condition par règle (voir la partie 3.2.1). Toutefois, ces méthodes engendrent un certain nombre de règles et de termes intermédiaires qui complexifient de façon non négligeable le système de réécriture. Il est cependant possible de généraliser la méthode de traduction à plus d'une condition.
Considérons pour chaque règle conditionnelle (i), sa fonction de condition c_i, d'arité 1+ nombre de conditions de (i). Pour la traduction, la méthode est la même que pour une unique condition exepté le fait que la fonction

c_i véhicule plus de termes conditionnels, qui doivent tous être réécrits en la forme close attendue pour produire le résultat.

Exemple 3.6 *Exemple de traduction de conditions multiples dans le cadre de l'évaluation de valeurs booléennes.* $\mathcal{F} = \{true : 0, false : 0, not : 1, \oplus : 2, and : 2, or : 2\}$*, et* $x, y, z \in \mathcal{X}$.

$$
\begin{array}{llll}
(1) & \oplus(x,y) & \to\ true & \textit{if } x \to^{\circledast} true \wedge y \to^{\circledast} false \\
(2) & \oplus(x,y) & \to \oplus(y,x) & \\
(3) & \oplus(x,y) & \to\ false & \textit{if } and(x,y) \to^{\circledast} true \\
(4) & \oplus(x,y) & \to\ false & \textit{if } x \to^{\circledast} false \wedge y \to^{\circledast} false \\
(5) & and(x,y) & \to\ true & \textit{if } x \to^{\circledast} true \wedge y \to^{\circledast} true \\
(5) & and(x,y) & \to\ false & \textit{if } x \to^{\circledast} false \\
(6) & and(x,y) & \to and(y,x) & \\
(7) & or(x,y) & \to\ true & \textit{if } x \to^{\circledast} true \\
(8) & or(x,y) & \to\ false & \textit{if } x \to^{\circledast} false \wedge y \to^{\circledast} false \\
(9) & or(x,y) & \to or(y,x) & \\
(10) & not(x) & \to\ true & \textit{if } x \to^{\circledast} false \\
(11) & not(x) & \to\ false & \textit{if } x \to^{\circledast} true
\end{array}
$$

Après traduction cela donne :

$$
\begin{array}{lll}
(1_1) & \oplus(x,y) & \to c_1(true, x, y) \\
(1_2) & c_1(x, true, false) & \to x \\
(2) & \oplus(x,y) & \to \oplus(y,x) \\
(3_1) & \oplus(x,y) & \to c_3(false, and(x,y)) \\
(3_2) & c_1(x, true) & \to x \\
(4_1) & \oplus(x,y) & \to c_4(false, x, y) \\
(4_2) & c_4(x, false, false) & \to x \\
\vdots & \vdots & \\
(11_1) & not(x) & \to c_{11}(false, x) \\
(11_2) & c_{11}(x, true) & \to x
\end{array}
$$

3.3 Approximation des termes accessibles dans un système traduit

Cette méthode de traduction permet l'emploi de l'algorithme d'approximation décrit en 2.2.3.Une fois un terme conditionnel $c_i(r,s)$ introduit dans l'automate, les itérations successives permettent de reconnaître les réécritures de ce terme et, par conséquent, le terme composé de c et des réécritures de r et s. Ainsi dans le cas ou la condition est vérifiée, r et ses descendants vont effectivement être des descendants de l et seront donc reconnus par le même état de l'automate. Dans le cas contraire, l'algorithme va calculer tout de même les descendants de r, qui vont alors être reconnus par l'automate, alors qu'il n'aboutissent à aucun terme dépourvu de conditions, et donc de calculer $\mathcal{R}^*(E)$. Il peut alors être intéressant de rendre prioritaires les évaluations des réécritures de s afin de pouvoir s'éviter ces calculs inutiles dans le cadre des conditions fausses.

Du point de vue des états générés, il est nécessaire de remarquer que chaque condition nécessite la création d'au moins un état supplémentaire de l'automate, et ce pour chaque langage de termes (i.e. ensemble de termes reconnus par un même état) pour lesquels la règle $l \to r$ est applicable. De plus chaque condition peut amener à utiliser d'autres règles conditionnelles (voire la même règle conditionnelle avec une substitution différente). Par conséquent, cette méthode sera coûteuse dans les cas ou les conditions s'avèrent très restrictives (peu souvent vérifiées) ou sont difficiles à évaluer.

Chapitre 4

Évaluation des conditions dans des automates d'arbres

Dans cette partie nous abordons l'extension de l'algorithme de calcul et d'approximation de $\mathcal{R}^*(E)$ décrit en 2.2.3 aux systèmes de réécriture conditionnels. Afin de pouvoir exploiter une méthode similaire à celle employée pour les systèmes de réécriture non conditionnels, il faut être en mesure de déterminer, étant donné une règle conditionnelle, si elle est applicable ou non. Dans le cadre de la réécriture conditionnelle, l'applicabilité d'une règle peut être soumise à l'applicabilité d'une autre règle et ainsi de suite. Ce problème amène à rechercher, au sein de ce qui est déjà construit, les conditions qui sont validées, sous réserve que cet ensemble soit croissant au fil de la construction de l'automate reconnaissant $\mathcal{R}^*(E)$.

4.1 Automate et validité des conditions

Considérons une règle conditionnelle de la forme $l \to r$ if $s \downarrow t$. Pour une substitution σ, le terme $l\sigma$ se réécrit en $r\sigma$ si et seulement si il existe un terme u tel que $s\sigma \to^* u$ et $t\sigma \to^* u$ dans le système considéré. donc $l \to r$ si $\mathcal{R}^*(s\sigma) \cap \mathcal{R}^*(t\sigma) \neq \emptyset$. Par conséquent, le problème de validation de condition renvoie à un problème de calcul de descendants. Si l'on applique aucune stratégie particulière, il faut alors générer les ensembles $\mathcal{R}^*(s\sigma)$ et $\mathcal{R}^*(t\sigma)$ avant de vérifier s'ils possèdent ou non des descendants communs. Remarquons qu si nous effectuons une sur-approximation des ensemble de descendants alors nous allons, par cette méthode de vérification des conditions, obtenir une sur-évaluation de la validité des conditions ; ceci est acceptable car une telle sur-évaluation induit des réécritures supplémentaires, ce qui est cohérent avec le fait de sur-approximer $\mathcal{R}^*(E)$.

4.1.1 Sémantique d'un état dans la construction de $\mathcal{R}^*(E)$

Considérons la méthode de construction de $\mathcal{R}^*(E)$ décrite en 2.2.3 sur un automate d'entrées $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ et un système de réécriture non conditionnel $\mathcal{R}$. Soit $q \in \mathcal{Q}$ un état de l'automate reconnaissant le terme t. Lorsque t fait partie d'une paire critique, $t \to_{\mathcal{R}} r$, $r \notin \mathcal{L}(\mathcal{A}, q)$, alors la méthode consiste à normaliser la transition $r \to_\Delta q$ pour l'ajouter dans l'automate. Ainsi, tous les descendants de t par $\mathcal{R}$, obtenus grâce à une paire critique, sont reconnus par le même état q.

Corollaire 4.1 *Pour un système $\mathcal{R}$ non conditionnel, dans la construction de l'automate $\mathcal{A}$ reconnaissant un sur-ensemble de $\mathcal{R}^*(E)$ par complétion, si $l \to_{\mathcal{R}}^* r$ et $l \in \mathcal{L}(\mathcal{A}, q_l)$ alors $r \in \mathcal{L}(\mathcal{A}, q_l)$.*

La preuve de ce corollaire constitue un lemme permettant de prouver la validité de la construction. Elle est décrite dans [Gen97].

4.1.2 Calcul des descendants d'un terme provenant d'une condition

En conservant cette propriété lors de l'adaptation de l'algorithme aux système de réécriture conditionnels, on peut obtenir une méthode d'évaluation des conditions. Il suffit en effet, pour une réécriture $(i)\ l\sigma \to r\sigma$ if $s\sigma \downarrow t\sigma$ ((i) étant l'identifiant de la règle utilisée), de créer deux états $q_{l\sigma,(i),c_1}$ et $q_{l\sigma,(i),c_2}$. Lors de la complétion, ces deux états reconnaîtrons alors les descendants de $s\sigma$ et $t\sigma$ respectivement, et la condition sera effectivement vérifiée, si à partir d'une certaine étape, nous obtenons $\mathcal{L}(\mathcal{A}(q_{l\sigma,(i),c_1}) \cap \mathcal{L}(\mathcal{A}(q_{l\sigma,(i),c_2}) \neq \emptyset$.

4.2 Stratégie de construction de $\mathcal{R}^*(E)$

Partant de cette idée, il est alors possible de définir une méthode de construction de l'automate qui reconnaît $\mathcal{R}^*(E)$ dans le cadre d'un système de réécriture conditionnel.

4.2.1 Construction par complétion

A partir d'un automate $\mathcal{A}_0$ reconnaissant l'ensemble de termes E qui constituent les entrées d'un système de réécriture conditionnel R sur un alphabet de termes $\mathcal{T}(\mathcal{F}, \mathcal{X})$, on définit la procédure de complétion suivante :

1. à partir de l'automate $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}_i, \mathcal{Q}_f, \Delta_i \rangle$, on construit l'automate $\mathcal{A}_{i+1} = \langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_f, \Delta_{i+1} \rangle$ avec initialement $\mathcal{Q}_{i+1} = \mathcal{Q}_i, \Delta_{i+1} = \Delta_i$.
2. on considère chaque paire critique sans tenir compte des conditions. Une paire $(l\sigma_{\mathcal{L}}, r\sigma_{\mathcal{L}})$ est dite critique sur une règle non conditionnelle $l \to r$ ou conditionnelle $l \to r$ if $s \downarrow t$, identifiée par (i) si $\sigma_{\mathcal{L}}$ est

une substitution de langage $\sigma_{\mathcal{L}} = (x_1 : q_{i_1}, x_2 : q_{i_2} .... x_n : q_{i_n})$ avec $(x_1, x_2 ... x_n) \in var(l)$ et $\exists q_l \in \mathcal{Q}_i$ t.q. $l\sigma_{\mathcal{L}} \rightarrow^*_{\Delta_i} q_l$ et $r\sigma_{\mathcal{L}} \not\rightarrow^*_{\Delta_i} q_l$.

3. pour chacune de ces paires critique, (i) est soit une règle :
   - conditionnelle : $l \rightarrow r$ if $s \downarrow t$, dans ce cas deux choses sont possibles :
     - il existe 2 états $q_{l\sigma,(i),c_1}$ et $q_{l\sigma,(i),c_2}$ dans $\mathcal{Q}_i$. Il faut alors calculer $\mathcal{L}(\mathcal{A}_i(q_{l\sigma,(i),c_1}) \cap \mathcal{L}(\mathcal{A}_i(q_{l\sigma,(i),c_2})$. Si cet ensemble est vide, alors on considère temporairement la condition comme fausse et aucune modification n'a lieu. S'il est non vide, alors la condition est vérifiée et on continue à traiter la paire critique comme s'il s'agissait d'une règle non conditionnelle.
     - $q_{l\sigma,(i),c_1} \notin \mathcal{Q}_i$ ou $q_{l\sigma,(i),c_2} \notin \mathcal{Q}_i$, alors on crée ces deux états et on ajoute à l'automate $\mathcal{A}_{i+1}$ les transitions
     $$s\sigma_{\mathcal{L}} \rightarrow^*_{\Delta_{i+1}} q_{l\sigma,(i),c_1} \text{ et } t\sigma_{\mathcal{L}} \rightarrow^*_{\Delta_{i+1}} q_{l\sigma,(i),c_2}$$
   - non conditionnelle (ou alors conditionnelle et la condition vient d'être vérifiée à l'étape ci-dessus), alors on ajoute la transition $r\sigma_{\mathcal{L}} \rightarrow^*_{\Delta_{i+1}} q_l$.
4. le nouvel automate $\mathcal{A}_{i+1} = \langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_{f,i+1}, \Delta_{i+1} \rangle$ constitue le résultat d'une étape de complétion à partir de $\mathcal{A}_i$.

Ainsi, lorsque qu'il existe $i \in \mathbb{N}$ tel que $\mathcal{A}_i = \mathcal{A}_{i+1}$, la procédure de complétion est terminée et $\mathcal{A}_i$ est le résultat. Nous verrons plus loin des considérations d'approximation pour arriver à un tel point fixe (section 4.3).

4.2.2 Suivi des conditions

Dans cet algorithme de complétion, l'estimation des conditions est initialement fausse pour toute condition. Sont d'abord créés les états qui vont permettre de vérifier ou non ces conditions, et ensuite, dans le cas ou une réécriture permet de joindre les deux termes conditionnels, la condition devient vraie et la règle est prise en compte (pour la substitution considérée) pour la complétion de l'automate.

Exemple 4.1 *Soit* $\mathcal{F} = \{true : 0, nil : 0, conj : 1 cons : 2\}$, $\mathcal{X} = \{l\}$ *et* $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, *où* $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{Q}_f = \{q_4\}$, *et*

$$\Delta = \begin{cases} true & \rightarrow q_0 \\ nil & \rightarrow q_1 \\ cons(q_0, q_1) & \rightarrow q_2 \\ cons(q_0, q_2) & \rightarrow q_3 \\ conj(q_3) & \rightarrow q_4 \end{cases}$$

Soit le système de réécriture

$$\mathcal{R} = \begin{cases} (1) & conj(cons(true, l)) \quad \to true \text{ if } conj(l) \downarrow true \\ (2) & conj(nil) \quad \to true \end{cases}$$

1. *Première étape de complétion, une unique paire critique :*

$$\begin{array}{ll} conj(cons(true, q_2)) & \to_{\mathcal{R}} true \text{ if } conj(q_2) \downarrow true \\ conj(cons(true, q_2)) & \to^*_{\Delta} q_4 \end{array}$$

   Création des états $q_5 = q_{conj(cons(true,q_2)),(1),c_1}$ *et* $q_6 = q_{conj(cons(true,q_2)),(1),c_2}$. $\mathcal{Q}_1 = \mathcal{Q} \cup \{q_5, q_6\}$, $\Delta_1 = \Delta \cup \{conj(q_2) \to q_5, true \to q_6\}$.

2. *Deuxième étape de complétion, deux paires critiques :*

$$\begin{array}{ll} conj(cons(true, q_2)) & \to_{\mathcal{R}} true \text{ if } conj(q_2) \downarrow true \\ conj(cons(true, q_2)) & \to^*_{\Delta} q_4 \end{array}$$

$$\begin{array}{ll} conj(cons(true, q_1)) & \to_{\mathcal{R}} true \text{ if } conj(q_1) \downarrow true \\ conj(cons(true, q_1)) & \to^*_{\Delta} q_5 \end{array}$$

   Aucune opération à effectuer pour la première, la condition est toujours évaluée comme fausse. La deuxième paire critique engendre la création des états $q_7 = q_{conj(cons(true,q_1)),(1),c_1}$ *et* $q_8 = q_{conj(cons(true,q_1)),(1),c_2}$. $\mathcal{Q}_2 = \mathcal{Q}_1 \cup \{q_7, q_8\}$, $\Delta_2 = \Delta_1 \cup \{conj(q_1) \to q_7, true \to q_8\}$.

3. *Troisième étape de complétion, trois paires critiques :*

$$\begin{array}{ll} conj(cons(true, q_2)) & \to_{\mathcal{R}} true \text{ if } conj(q_2) \downarrow true \\ conj(cons(true, q_2)) & \to^*_{\Delta} q_4 \end{array}$$

$$\begin{array}{ll} conj(cons(true, q_1)) & \to_{\mathcal{R}} true \text{ if } conj(q_1) \downarrow true \\ conj(cons(true, q_1)) & \to^*_{\Delta} q_5 \end{array}$$

$$\begin{array}{ll} conj(q_1) & \to_{\mathcal{R}} true \\ conj(q_1) & \to^*_{\Delta} q_7 \end{array}$$

   Aucune opération à effectuer pour les 2 premières, la troisième induit $\Delta_3 = \Delta_2 \cup \{true \to q_7\}$.

4. *Quatrième étape de complétion :*

$$\begin{array}{ll} conj(cons(true, q_2)) & \to_{\mathcal{R}} true \text{ if } conj(q_2) \downarrow true \\ conj(cons(true, q_2)) & \to^*_{\Delta} q_4 \end{array}$$

$$\begin{array}{ll} conj(cons(true, q_1)) & \to_{\mathcal{R}} true \text{ if } conj(q_1) \downarrow true \\ conj(cons(true, q_1)) & \to^*_{\Delta} q_5 \end{array}$$

La première condition est toujours évaluée comme fausse. La deuxième vérifie maintenant $true \rightarrow_\Delta q_7 = q_{conj(cons(true,q_1)),(1),c_1}$, *elle est donc validée :* $\mathcal{L}(\mathcal{A}, q_8) \cap \mathcal{L}(\mathcal{A}, q_7) = \{true\}$. $\Delta_4 = \Delta_3 \cup \{true \rightarrow q_5)\}$.

5. *Cinquième étape de complétion :*

$$\begin{array}{ll} conj(cons(true, q_2)) & \rightarrow_\mathcal{R} true \text{ if } conj(q_2) \downarrow true \\ conj(cons(true, q_2)) & \rightarrow_\Delta^* q_4 \end{array}$$

Ici, on a $true \rightarrow_\Delta q_5 = q_{conj(cons(true,q_2)),(1),c_1}$, *et* $conj(q_2) \rightarrow_\Delta q_5$, *ce qui valide la condition. Par conséquent, après normalisation, on obtient* $\Delta_5 = \Delta_4 \cup \{true \rightarrow q_4\}$.

À partir de ce point, il n'existe plus de paire critique, et l'automate $\mathcal{A}_5 = \langle \mathcal{F}, \mathcal{Q}_5, \mathcal{Q}_f, \Delta_5 \rangle$, *où* $\mathcal{Q}_5 = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$, $\mathcal{Q}_f = \{q_4\}$, *et*

$$\Delta_5 = \left\{ \begin{array}{llll} true & \rightarrow q_0 & true & \rightarrow q_6 \\ nil & \rightarrow q_1 & conj(q_1) & \rightarrow q_7 \\ cons(q_0, q_1) & \rightarrow q_2 & true & \rightarrow q_8 \\ cons(q_0, q_2) & \rightarrow q_3 & true & \rightarrow q_7 \\ conj(q_3) & \rightarrow q_4 & true & \rightarrow q_5 \\ conj(q_2) & \rightarrow q_5 & true & \rightarrow q_4 \end{array} \right.$$

Ici le langage reconnu par $\mathcal{A}_5$ *est* $\{conj(cons(true, cons(true, nil))), true\}$. *La condition a été évaluée quand les réécritures des termes conditions ont été suffisantes.*

4.2.3 Validité

Considérons toujours un automate $\mathcal{A}_0$ reconnaissant un ensemble d'entrées E pour un système de réécriture conditionnel $\mathcal{R}$. Supposons qu'il existe un automate $\mathcal{A}'$ point fixe pour la complétion 4.2.1 (en pratique obtenu par complétions successives à partir de $\mathcal{A}_0$) et dont le langage reconnu vérifie $\mathcal{L}(\mathcal{A}') \subset \mathcal{L}(\mathcal{A}_0)$.

Théorème 4.1 $\mathcal{L}(\mathcal{A}')$ *est clos par* $\mathcal{R}$ *et* $\mathcal{R}^*(E) \subseteq \mathcal{L}(\mathcal{A}')$

Preuve Montrons que $\forall t \in \mathcal{T}(\mathcal{F})$ t.q. $\exists q \in \mathcal{Q}_0, t \in \mathcal{L}(\mathcal{A}', q), \forall u \in \mathcal{T}(\mathcal{F})$ t.q. $t \rightarrow_\mathcal{R}^* u$, on a $u \in \mathcal{L}(\mathcal{A}', q)$
Procédons par récurrence, prouvons que $\forall n \in \mathbb{N}, q \in \mathcal{Q}_0, t \in \mathcal{L}(\mathcal{A}', q)$, u t.q. $t \xrightarrow{\downarrow n}{}^*_\mathcal{R} u$, alors $u \in \mathcal{L}(\mathcal{A}', q)$

- $\forall n \in \mathbb{N}, q \in \mathcal{Q}_0, t \in \mathcal{L}(\mathcal{A}', q)$, u t.q. $t \xrightarrow{\downarrow 0}{}^*_\mathcal{R} u$, alors $u \in \mathcal{L}(\mathcal{A}', q)$. En effet, $\xrightarrow{\downarrow 0}{}^*_\mathcal{R}$ signifie que l'on considère le système $\mathcal{R}$ sans les règles conditionnelles, la preuve est donc la même que pour l'algorithme de complétion dans les système de réécriture non conditionnels [Gen97].

- Supposons maintenant que pour $n, \forall k \leq n, t \xrightarrow{\downarrow k}{}^{*}_{\mathcal{R}} u$ et $t \rightarrow^{*}_{\Delta'} q$ implique $u \rightarrow^{*}_{\Delta'} q$. Montrons que l'on peut en déduire :

$$t \xrightarrow{\downarrow n+1}{}^{*}_{\mathcal{R}} u \text{ et } t \rightarrow^{*}_{\Delta'} q \Rightarrow u \rightarrow^{*}_{\Delta'} q$$

. $t \xrightarrow{\downarrow n+1}{}^{*}_{\mathcal{R}} u$ signifie qu'il existe $\{t_1, t_2, \ldots, t_{n-1}\} \subset \mathcal{T}(\mathcal{F})$ tels que $t_0 = t \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_1 \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_2 \xrightarrow{\downarrow n+1}_{\mathcal{R}} \ldots \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{j-1} \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_j = u$

supposons pour $i < j, t_i \rightarrow^{*}_{\Delta'} q$, 2 cas se présentent :

  - $t_i \xrightarrow{\downarrow n}_{\mathcal{R}} t_{i+1}$, alors, d'après l'hypothèse de récurrence, $t_{i+1} \rightarrow^{*}_{\Delta'} q$.
  - $t_i \not\xrightarrow{\downarrow n}_{\mathcal{R}} t_{i+1}$ et $t_i \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{i+1}$, alors il existe une règle $(k)\, l \rightarrow r$ if $c_1 \downarrow c_2 \in \mathcal{R}$ un contexte clos $C[]$, et une substitution σ tels que :

$$t_i = C[l\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] = t_{i+1} \text{ if } c_1\sigma \downarrow c_2\sigma$$

$$\text{et } \exists u \text{ t.q. } c_1\sigma \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} u, \text{ et } c_2\sigma \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} u$$

Comme il n'existe pas de paire critique entre $\mathcal{R}$ et Δ' qui induise une modification de l'automate (car l'automate est un point fixe pour la complétion), alors $\exists q_{l\sigma,(k),c_1}, q_{l\sigma,(k),c_2} \in \mathcal{Q}'$. Par conséquent, on a :

$$\begin{array}{lll} c_{k,1}\sigma & \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} u & \text{et } c_{k,1} \rightarrow_{\Delta'} q_{l\sigma,(k),c_1} \\ c_{k,2}\sigma & \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} u & \text{et } c_{k,1} \rightarrow_{\Delta'} q_{l\sigma,(k),c_2} \end{array}$$

Par hypothèse de récurrence, ceci induit $u \in \mathcal{L}(\mathcal{A}', q_{l\sigma,(k),c_1}))$ et $u \in \mathcal{L}(\mathcal{A}', q_{l\sigma,(k),c_2}))$.Par conséquent $t_i = C[l\sigma] \rightarrow_{\mathcal{R}} C[r\sigma] = t_{i+1}$ if $c_{k,1}\sigma \downarrow c_{k,2}\sigma$ voit sa condition vérifiée dans le cadre de l'automate et $t_i \rightarrow^{*}_{\Delta'} q$ implique par méthode de complétion $t_{i+1} \rightarrow^{*}_{\Delta'} q$

Or, $t = t_0 \in \mathcal{L}(\mathcal{A}', q)$, donc par induction, $\forall i \leq n, t_i \in \mathcal{L}(\mathcal{A}', q)$, en particulier $u = t_n$.

On retrouve donc $t \xrightarrow{\downarrow n+1}{}^{*}_{\mathcal{R}} u$ et $t \rightarrow^{*}_{\Delta} q$ implique $u \rightarrow^{*}_{\Delta} q$

Donc $\forall n \in \mathbb{N}, t \xrightarrow{\downarrow n}{}^{*}_{\mathcal{R}} u$ et $t \rightarrow^{*}_{\Delta'} q$ implique $u \rightarrow^{*}_{\Delta'} q$, par conséquent $t \rightarrow^{*}_{\mathcal{R}} u$ et $t \rightarrow^{*}_{\Delta'} q$ implique $u \rightarrow^{*}_{\Delta'} q$. Donc $\forall q \in \mathcal{Q}', \mathcal{L}(\mathcal{A}', q)$ est clos par R, en particulier pour $q \in \mathcal{Q}_f$, donc $\mathcal{L}(\mathcal{A}')$ est clos par R. Puisque $E \subseteq \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$, alors $\mathcal{R}^*(E) \in \mathcal{L}(\mathcal{A}')$.

□

4.3 Application, approximation, et améliorations possibles

4.3.1 Approximation

En pratique, la méthode de complétion seule ne produit un résultat que si elle arrive à un point fixe. Ceci est essentiellement déterminé par la création d'états supplémentaires lors de la normalisation des règles. C'est dans ce sens, qu'il faut effectuer une normalisation induisant des approximations ciblées afin de faire converger la suite des $(\mathcal{A}_i)_{i\in\mathbb{N}}$ vers un point fixe. L'objectif de la méthode étant de prouver la non-appartenance de certains états critiques à $\mathcal{R}^*(E)$, il est judicieux de diriger cette approximation afin d'obtenir un résultat précis pour les réécriture pouvant mener à de tels termes (en particulier, les sous-termes des termes critiques), et plus large pour des termes de moindre importance.

4.3.2 Restriction de la création d'états dédiés aux conditions

Lorsqu'une condition nouvelle est rencontrée, nous créons systématiquement un état dédié à celle-ci afin d'en calculer les descendants. Cependant, lorsqu'il existe un état q dans l'automate dont le langage reconnu est **exactement** un des termes de la condition, il est alors possible d'utiliser cet état pour l'évaluation des descendants, sans perte de précision ni biais dans la méthode. Par extension, s'il existe un état q reconnaissant un langage de terme incluant un tel terme conditionnel c, l'utilisation de cet état pour reconnaître les descendant de c constitue une sur-approximation de la validité de la condition. De telles approximation peuvent permettre d'anticiper l'évaluation de certaines conditions et de grouper le calcul des descendants pour des termes proches ou de sémantique proche.

4.3.3 Extension de la méthode à d'autre types de conditions

Nous avons défini une méthode qui fonctionne pour les conditions du type $\downarrow$. Cependant, à partir de ce traitement, il est possible de permettre en ajoutant quelques modification de traiter les cas d'autres types de conditions.

- Pour les conditions du type (i) $\quad l \to r$ if $s \to^* t$, la méthode consiste à créer un état $q_{l\sigma,(i)}$ et de normaliser la transition $s\sigma \to_\Delta^* q_{l\sigma,(i)}$. Puis lors des étapes suivantes de complétion de vérifier si $t\sigma \in \mathcal{L}(\mathcal{A}, q_{l\sigma,(i)})$ pour valider la condition.
- Pour les conditions introduites dans le chapitre 3, c'est à dire pour une règle $l \to r$ if $s \to^{\circledast} t$, la méthode est la suivante : on crée un état q_t s'il n'existe pas déja (il peut être partagé par plusieurs règles ou par

la même avec des redex différents). Puis pour valider la condition on vérifie si $t \in \mathcal{L}(\mathcal{A}, q_{s\sigma})$. Ce type de condition est le moins couteux à évaluer.

Il est également possible d'utiliser simultanément plusieurs types de conditions différent, les états produits ne présentant aucune ambiguïté de nom. Ainsi l'approche de calcul par complétion d'automate de $\mathcal{R}^*(E)$ est satisfaisante dans tous les cas, à l'exeption des réécritures sous une forme normale ($\downarrow^!$ ou $\rightarrow^!$) pour lesquels une autre approche aurait certainement plus de succès.

Chapitre 5

Conclusion

5.1 Comparaison des différentes méthodes

Les résultats obtenus dans les deux approches sont très inégaux. En effet, la recherche de méthodes de traduction amène à restreindre le type de conditions. Ceci n'est pas une barrière dans la mesure où la description fonctionnelle d'un protocole reste facilitée par l'utilisation de tels systèmes de réécriture conditionnels. Cependant, si l'on y ajoute les problèmes dus à la conservation de termes conditionnels au sein des termes étudiés, la méthode d'algorithme par complétion étendu aux systèmes conditionnels est plus intéressante. En effet elle sépare les analyse des termes conditionnels des résultats, et permet donc une convergence plus rapide vers un point fixe. De plus elle produit des automates plus lisibles et donc facilite la procédure d'approximation pour un utilisateur.

5.2 Bilan

Ce stage a permis d'explorer deux voies pour l'évaluation des $\mathcal{R}$-descendants d'un ensemble de termes dans le cadre de la réécriture conditionnelle. Dans les deux cas, nous avons obtenu un résultat. L'approche par traduction a permis de caractériser les difficultés d'expression d'un système de réécriture conditionnel par un système de réécriture non conditionnel. L'extension de l'algorithme de complétion pour les systèmes conditionnels nous fourni un résultat satisfaisant, et permet de montrer que ceux-ci sont exploitable dans le cadre des preuves de programmes formalisés par des règles conditionnelles. Ce résultat permet donc d'étendre la flexibilité du formalisme de la réécriture en permettant l'introduction de conditions. De plus, outre l'avantage obtenu sur la facilité de formalisation, l'exemple 4.1 montre que les résultat sont généralement plus lisibles dans la mesure où l'évaluation de conditions au travers du langage reconnu d'états dédiés de l'automate permet de séparer les calculs d'une réécriture.

Le calcul des $\mathcal{R}$-descendants dans des systèmes non terminants ouvre des perspectives de preuves, mais reste encore difficile dans le cadre de gros systèmes. Les automates d'arbres constituent des structures adaptées à la description d'ensembles de termes qui ont permis l'obtention des algorithmes pour les systèmes conditionnels. L'étude d'autre structures finies pouvant reconnaître des ensembles non bornés de termes, et la recherche de méthode de construction de $\mathcal{R}^*(E)$ au sein de ces structures pourrait mener à l'obtention de nouvelles méthodes. De même, si les algorithmes de complétion sont une procédure de construction de $\mathcal{R}^*(E)$ incrémentale, il faut aussi envisager d'autres méthodes de construction de cet ensemble.

Les méthodes développées ne permettent d'effectuer des preuves de programmes que dans un cadre restreint, car l'application de l'algorithme de complétion est en pratique difficile ; notamment la recherche de paires critiques qui reste le principal obstacle à l'accroissement de la taille des système prouvables. Toutefois, ces résultats ouvrent des perspectives de preuve de programme formalisés par des systèmes de réécriture car ils ne requièrent aucune propriété particulière des systèmes, en particulier la terminaison qui est un critère difficile à valider et pourtant généralement requis pour les procédures de preuves.

Bibliographie

[AK96] Ilies Alouini and Claude Kirchner. Toward the concurrent implementation of computational systems. In *Algebraic and Logic Programming*, pages 1–31, 1996.

[BKK+96] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Marian Vittek. ELAN: A logical framework based on computational systems. In José Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4 of *Electronic Notes in TCS*, Asilomar (California), September 1996.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[Bra69] Walter S. Brainerd. Tree generating regular systems. *Information and Control*, 14:217–231, 1969.

[CDE+98] M. Clavel, F. Durán, S. Eker, P. Lincoln, and J. Meseguer. An Introduction to Maude (Beta Version). Technical report, SRI, March 1998.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata/`, 1997.

[CDGV91] J.L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata and rewrite systems. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 287–298. Springer-Verlag, 1991.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[DOS87] N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay*

(France), volume 308 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1987.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, pages 242–248, June 1990.

[Gen97] T. Genet. Decidable approximations of sets of normal forms and some applications. In *Workshop on Construction of Computational Logics, Dagstuhl (Germany)*, 1997.

[Gen98] T. Genet. *Contraintes d'ordre et automates d'arbres pour les preuves de terminaison*. PhD thesis, Université Henri Poincaré – Nancy 1, Nancy, 1998.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.

[GVTT01a] T. Genet and V. Viet Triem Tong. Timbuk Documentation. IRISA / Université de Rennes 1, 2001. `http://www.irisa.fr/lande/genet/timbuk/`.

[GVTT01b] T. Genet and Valérie Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with *timbuk*. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Havana (Cuba)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 691–702. Springer-Verlag, 2001.

[HL78] G. Huet and D. S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, Laboria, France, 1978.

[Jac96] F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proceedings 7th Conference on Rewriting Techniques and Applications, New Brunswick (New Jersey, USA)*, pages 362–376. Springer-Verlag, 1996.

[Sal88] K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *Journal of Computer and System Sciences*, 37:367–394, 1988.