Complétude de systèmes de réécriture d'ordre supérieur

Jacek Chrząszcz

Résumé

Ce travail étend la notion de complétude de systèmes de réécriture sur les systèmes d'ordre supérieur. Comme dans le premier ordre, cette notion est reliée à la complétude de définitions de fonctions. Une procédure de vérification de cette propriété est également présentée. Deux disciplines classiques de types sont étudiées : types simples et types polymorphes. Pour les types simples, plusieurs approches sur le traitement des arguments fonctionnels sont données.

Mémoire de stage de D.E.A. en informatique

Laboratoire de Recherche en Informatique
Université de Paris-Sud

Directeur de stage : Jean-Pierre Jouannaud

Orsay, 13 janvier 1998

Complétude de systèmes de réécriture d'ordre supérieur

Jacek Chrząszcz

13 janvier 1998

1 Introduction

La réécriture a des multiples applications dans l'informatique. Par rapport aux équations, utilisées traditionnellement en mathématiques, les règles de réécriture, qui sont tout simplement des équations orientées, sont beaucoup mieux adaptées au calcul symbolique. Elles éliminent une source potentielle de non-déterminisme — le choix d'une direction. L'autre source, le choix d'une sous-expression à réécrire, peut être aussi négligée si la relation de réécriture est confluente. Lorsqu'elle possède la propriété de terminaison, chaque réduction d'un terme donné aboutit à l'issue d'un nombre fini d'étapes à un unique terme irréductible, dit forme normale. Si nous traitons les termes en forme normale comme des valeurs et tous les termes comme des expressions algébriques, notre système de réécriture nous donne un moyen automatique de calculer la valeur d'une expression arbitraire. Si nous voulons nous concentrer sur l'aspect calculatoire de la réécriture, nous pouvons considérer une règle de réécriture, dont le coté gauche a un symbole f en tête, comme un cas de définition par filtrage de la fonction f et, en général, la réécriture comme un moyen d'exécution de programmes fonctionnels.

Étant donné un système de réécriture, nous pouvons en extraire l'ensemble de symboles qui n'apparaissent en tête d'aucun coté gauche. Ces symboles vont être ensuite appelés constructeurs et tous les autres — symboles définis. Remarquons qu'un terme construit uniquement à partir de constructeurs est forcément irréductible, nous avons ainsi trouvé un ensemble des valeurs.

Complétude

Un système de réécriture convergent (qui est confluent et qui termine) est appelé *complet* si chaque terme sans variables se réécrit en un terme constructeur. Ce fait est essentiel pour faire des preuves inductives de différentes propriétés sémantiques des objets décrits par notre système. Dans le cas d'un système complet, il suffit de démontrer la propriété donnée pour des termes constructeurs, parce que la valeur sémantique de chaque autre terme correspond à la valeur d'un terme constructeur.

La propriété globale de complétude du système est équivalente à une autre propriété plus locale et plus facile à tester automatiquement. Si les définitions de toutes les fonctions du système sont localement complètes, alors le système lui-même est complet. Dans le cadre de la réécriture, la complétude locale de la fonction f donnée signifie que tout terme qui est une application de f à des valeurs est réductible. Cette dernière propriété est appelée aussi la *réductibilité close*, mais on va utiliser le nom de $<<$ complétude locale $>>$ pour la distinguer de la complétude globale du système.

Pour tester la complétude de fonctions définies par des règles de réécriture (et aussi, automatiquement, la complétude globale du système de réécriture) Kounalis [10] a conçu une technique d'*arbres de motifs*. Cette technique a été ensuite adaptées par Bouhoula dans [1] pour des règles conditionnelles dans le cadre de son système de prototypage de spécifications parametrées SPIKE et par Bouhoula et Jouannaud dans [2] pour un langage de clauses de Horn avec égalité et appartenance, implanté sous le nom de MAUDE [13].

Systèmes de réécriture d'ordre supérieur

Les règles de réécriture constituent souvent un fondement de la sémantique de différents formalismes mathématiques. Par exemple, les études de la sémantique opérationnelle du λ-calcul, basée sur la règle de β-réduction ont beaucoup de points communs avec les études de systèmes de réécriture. On étudie aussi des théories de λ-termes, dont la sémantique est donnée, outre la règle β, par d'autres règles de réécriture.

Un exemple classique d'un tel système est le système $\mathbf{T}$ de Gödel (voir par exemple [5]), le λ-calcul des entiers représentés par $0 : \mathtt{nat}$, $succ : \mathtt{nat} \to \mathtt{nat}$ et une famille de récurseurs, pour chaque type τ une fonction $rec_\tau : \tau \times (\tau \times \mathtt{nat} \to \tau) \times \mathtt{nat} \to \tau$, définie par les règles suivantes :

$$\begin{array}{rcl} rec_\tau(u, v, 0) & \to & u \\ rec_\tau(u, v, succ(x)) & \to & v(rec_\tau(u, v, x), x) \end{array}$$

Il y a deux moyens de définir la réécriture d'ordre supérieur. Soit on travaille avec des termes en forme β-normale, η-expansée et après chaque application d'une règle de réécriture, on normalise le terme obtenu, soit on permet aux pas de réécriture et de β-réduction de s'entrelacer librement. La façon choisie dépend de l'application et Van de Pol dans [3] montre qu'aucune de deux méthodes n'est plus générale que l'autre. Le système $\mathbf{T}$, mentionné ci-dessus, est habituellement présenté dans ce deuxième cadre.

Les problèmes classique de la réécriture, comme la confluence et la terminaison deviennent plus complexes dans le cas de systèmes d'ordre supérieur. Du point de vue pratique, la situation est encore pire. On ne connaît toujours pas d'algorithme qui résout le problème de filtrage d'ordre supérieur, le mécanisme fondamental de la réécriture. Il est donc possible que ce problème, dans toute sa généralité, soit indécidable, comme le problème d'unification d'ordre supérieur [6], qui lui est très proche.

Comme toujours, quand l'informatique découvre qu'un problème est intraitable (p. ex. NP-difficile), ou même indécidable, on doit chercher des sous-classes de ce problème qui sont plus facilement maniables, mais qui en même temps sont assez grandes pour être utiles en pratique. Dans le cadre de la réécriture d'ordre supérieur, la restriction consiste à limiter la forme des règles. Dans [14] Miller propose une forme particulière de $<<$ patterns $>>$ d'ordre supérieur pour lesquels le problème d'unification reste décidable et même quasi-linéaire [16].

Grâce à cela l'étude du problème de confluence a pu avancer. Le théorème de Knuth-Bendix, qui réduit la confluence d'un système de réécriture à un simple test de confluence de paires critiques, a été généralisé par Nipkow [15] aux systèmes de réécriture de $<<$ patterns $>>$ d'ordre supérieur, d'où décidabilité de cette propriété lorsque la réécriture en question termine.

Pendant les années 90 les premières techniques de preuve de terminaison ont aussi été développées (p. ex. [11]), mais même pour des $<<$ patterns $>>$, l'état de l'art n'est pas comparable avec la multitude de possibilités pour le premier ordre. Seulement récemment une analogie du puissant ordre récursif sur les chemins est apparu [9], mais le théorème de Kruskal qui est très souvent utilisé dans le premier ordre pour démontrer que de nouvelles relations d'ordres sont noetheriennes n'a pas de bonne généralisation connue à l'ordre supérieur.

Complétude de systèmes de réécriture d'ordre supérieur

Ce travail étend la notion de complétude de systèmes de réécriture aux systèmes d'ordre supérieur. Dans ce cadre, la définition de la complétude doit être reformulée. Ce n'est plus raisonnable de demander à certains termes, par exemple $\lambda x.f(x)$, pour un symbole défini f, de se réécrire. Si c'était notre intention, on devrait pouvoir exprimer chaque fonction par un λ-terme constructeur unique, indépendant des arguments. On perdrait donc la possibilité d'avoir un comportement dépendant des valeur des arguments actuels, ce qui est l'essence de la définition par cas et donc des définitions par des règles de réécriture. Pour y remédier on restreint l'ensemble de termes considéré aux seuls termes dont les types sont observables, donc de type de base, qui devront se réécrire en termes constructeurs pour que le système puisse être appelé complet.

Les λ-termes de type supérieur sont donc considérés comme des éléments de l'espace fonctionnel sur l'ensemble de valeurs. Dès qu'on applique une telle fonction à tous les arguments nécessaires, on obtient un terme de type de base, dont la valeur peut être déterminée si le système de règles est complet.

De même, la notion de complétude locale de définition de fonction doit changer. Les fonctions peuvent maintenant prendre des arguments qui sont eux-mêmes de type fonctionnel. Étant donné une fonction f, on demande que chaque terme, qui est une application de f à des termes constructeurs à la place des paramètres de type de base et des termes quelconques à la place des paramètres de type fonctionnels, soit réductible. Ce mémoire présente aussi la technique d'arbres de motifs adaptée aux systèmes de réécriture d'ordre supérieur pour tester la complétude locale de définitions de fonctions.

Pour les arguments fonctionnels de la fonction testée trois approches sont étudiées. L'approche classique consiste à traiter les arguments fonctionnels comme des $<<$ boîtes noires $>>$. Ils peuvent être appliqués à quelque chose ou éliminés, mais on ne peut pas modifier la structure interne de la fonction ni faire de récurrence. Dans le cadre de la réécriture cela consiste à limiter la forme des membres gauches de règles utilisées pour définir des fonctions. Les sous-termes de membres gauches qui filtrent les arguments de types fonctionnels doivent être triviaux, la distinction de cas de définitions est donc limitée aux arguments de premier ordre.

Cette approche a un grand avantage : la modularité. Si on ajoute des nouvelles fonctions complètement définies (par des règles appropriées) à un système globalement complet, le nouveau système reste complet. L'inconvénient est l'impossibilité d'utiliser le filtrage d'ordre supérieur ce qui constitue la vraie puissance de la réécriture d'ordre supérieur.

Dans la deuxième approche le filtrage d'ordre supérieur peut être utilisé librement à la seule condition que les cotés gauches des règles soient des << patterns >>. Par exemple considérons la fonction $Norm$ qui calcule la forme normale de fonctions de type `bool` $\rightarrow$ `bool`. Ces deux règles pourrait en faire partie de sa définition :

$$\begin{array}{lcl} Norm(\lambda x^{\texttt{bool}}.\mathrm{T} \vee \phi(\mathrm{x})) & \rightarrow & \lambda x^{\texttt{bool}}.\mathrm{T} \\ Norm(\lambda x^{\texttt{bool}}.\mathrm{F} \vee \phi(\mathrm{x})) & \rightarrow & Norm(\lambda x^{\texttt{bool}}.\phi(x)) \end{array}$$

Cependant on perd la propriété de modularité : les nouvelles fonctions (même complètement définies), ajoutée à un système complet R peuvent devenir elles-mêmes des arguments des fonction de R. Comme elles ne sont pas considérées par des règles de R, il est possible que le système étendu ne soit plus complet.

La troisième approche non-standard, décrite dans la section 3.8, est un essai d'unifier les avantages des deux premières. On introduit les fonctions à syntaxe limitée, qui sont les fonctions exprimable par des λ-termes utilisant un ensemble spécifié de symboles et qui ont des propriétés similaires aux celles de termes de types de base, sauf qu'elles peuvent être appliquées à des arguments. Les définitions par filtrage sur les arguments fonctionnels de ce nouveau type sont autorisées et la propriété de complétude reste modulaire.

Ce travail présente la problématique de complétude de systèmes d'ordre supérieur dans deux disciplines de types : types simples et types polymorphes. Pour des types simples les trois approches sur le traitement d'arguments fonctionnels sont étudiés dans le cadre de la réécriture de termes normalisés. Dans la discipline de types polymorphes on se restreint à la première approche, la réécriture étant la version où on entrelace librement l'application de règles avec la β-réduction.

Comme dans le premier ordre, la liaison entre la propriété globale de complétude du système et la propriété de complétude locale de définition de fonction est établie par le théorème 2, pour des types simples, et le théorème 4, pour des types polymorphes

L'organisation de ce mémoire est la suivante. Après la présente introductions, la section 2 rappelle d'une façon plus détaillée la problématique de la complétude de systèmes de réécriture de premier ordre. La section 3 décrit l'extension au λ-calcul simplement typé et la section 4 aux types polymorphes. Les conclusions sont données dans la section 5.

2 Premier ordre

La présente section constitue une introduction à la problématique de la complétude de définitions [7, 17, 10]. Les notions de complétude locale, complétude globale ainsi que la procédure de vérification qui seront introduites et expliquées dans cette section, sont assez intuitives dans le cas du premier ordre. Les mêmes notions réapparaîtront dans les sections consacrées à l'ordre supérieur, mais elles seront bien plus complexes et moins intuitives.

2.1 Types et termes

Nous allons travailler avec l'ensemble fini de types de base T. Pour faciliter la présentation, aucun opérateur sur les types ne sera introduit dans cette section.

Pour chaque type τ on a un ensemble dénombrable de variables $\mathcal{X}_\tau$. La variable x de type τ peut être aussi notée x^τ. Les ensembles $\mathcal{X}_{\tau_1}$ et $\mathcal{X}_{\tau_2}$ sont disjoints pour $\tau_1 \neq \tau_2$ et l'ensemble de toutes les variables est noté $\mathcal{X}$.

En outre, nous avons aussi besoin d'un ensemble fini de symboles de fonction $\mathcal{F}$. Avec chaque symbole $f \in \mathcal{F}$ vient une spécification du nombre et des types de ses arguments ainsi que de son type de sortie, ce qui va être noté $f : \tau_1 \times \cdots \times \tau_n \to \tau$ pour $n \geq 0$. Si $n = 0$, f est une constante et on écrit $f : \tau$.

Les termes bien-typés sont construits selon les règles suivantes :

$$\textbf{(var)} \qquad \frac{}{x : \tau} \text{ pour } x \in \mathcal{X}_\tau$$

$$\textbf{(fun)} \qquad \frac{u_1 : \tau_1, \ldots, u_n : \tau_n}{f(u_1, \ldots, u_n) : \tau} \text{ pour } \begin{array}{l} f \in \mathcal{F} \\ f : \tau_1 \times \cdots \times \tau_n \to \tau \end{array}$$

L'ensemble de termes bien-typés, pour $\mathcal{X}$ et $\mathcal{F}$ donnés, est noté $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Les termes peuvent être aussi traités comme des arbres étiquetés par des symboles de $\mathcal{F} \cup \mathcal{X}$, on va donc utiliser les suites d'entiers positifs pour symboliser les positions dans les termes ; $t|_p$ va signifier un sous-terme de t à la position p et $t[u]_p$ un terme t avec un terme u à la place du sous-terme $t|_p$. L'ensemble des variables qui apparaissent dans un terme t est noté $FV(t)$. Une variable est dite *linéaire* si elle apparaît dans le terme une seule fois. Un terme dont toutes les variables sont linéaire est aussi appelé linéaire. L'ensemble de termes clos (sans variables) est noté $\mathcal{T}(\mathcal{F})$.

Les symboles de fonction, qui forment l'ensemble $\mathcal{F}$, sont divisés en deux parties disjointes : l'ensemble des constructeurs C et l'ensemble des symboles définis D. On va toujours supposer que pour tout type $\tau \in T$, il existe au moins un terme constructeur $t \in \mathcal{T}(C)$ tel que $t : \tau$.

Une substitution est une fonction partielle de l'ensemble des variables vers l'ensemble des termes, $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, dont le domaine $Dom(\sigma)$ est fini. D'une façon naturelle cette fonction se prolonge à tout l'ensemble de termes. La fonction étendue (par abus de notation) est toujours notée σ. L'application de la substitution σ à un terme t est notée $t\sigma$. Les variables de t qui n'appartiennent pas à $Dom(\sigma)$ ne changent pas. Une substitution σ de domaine $\{x_1, \ldots, x_n\}$ telle que $x_i\sigma = t_i$ peut être aussi notée $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. Si $x\sigma \in \mathcal{T}(\mathcal{F})$

pour tout $x \in Dom(\sigma)$, alors σ est appelé une substitution close. D'autres propriétés des termes se prolongent sur les substitutions de la même manière.

2.2 Réécriture du premier ordre

Un système de réécriture est un ensemble fini de règles $R = \{l_i \to r_i\}$, pour les termes $l_i, r_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ de même type, qui satisfont les contraintes suivantes :

- les variables du coté droit apparaissent dans le coté gauche, $FV(r_i) \subseteq FV(l_i)$,
- le coté gauche n'est pas une variable
- le coté gauche ne commence pas par un constructeur $l_i \neq c(t_1, \ldots, t_n)$ pour $c \in C$,

Étant donné un tel système R, un terme $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ se réécrit en un terme t en utilisant la règle $l \to r$ à la position p, ce qui va être noté $s \xrightarrow[l \to r]{p} t$ (où tout simplement $s \longrightarrow_R t$), s'il existe une substitution σ telle que $s|_p = l\sigma$ et $t = s[r\sigma]_p$. La cloture reflexive et transitive de la relation $\longrightarrow_R$ sera notée $\longrightarrow_R^*$.

On va supposer en plus que le système R est linéaire, donc chaque coté gauche l_i est un terme linéaire. La propriété de linéarité de règles n'est pas fondamentale pour l'étude de la complétude des définitions, mais facilite les définitions d'une façon non-négligeable.

2.3 Complétude

Le but général de la réécriture, compris comme une manière de définir les fonctions et d'exécuter des programmes fonctionnels, est de transformer toute expression close, utilisant des fonctions, en une valeur, donc (en premier ordre) un terme construit uniquement à partir de constructeurs.

Définition 2.3.1 `(complétude globale)`
Un système de réécriture R est complet *ssi pour tout terme clos $t \in \mathcal{T}(\mathcal{F})$ il existe un terme $s \in \mathcal{T}(C)$ tel que $t \longrightarrow_R^* s$.*

La notion de complétude globale étant complexe, nous allons introduire une propriété auxiliaire qui sera plus facile à démontrer. La complétude locale de la définition de fonction signifie que l'application de cette fonction à des valeurs donne toujours un terme réductible.

Définition 2.3.2 `(complétude locale)`
Un système de réécriture R définit une fonction $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \tau$ d'une façon localement complète *ssi pour tous termes $u_1, \ldots, u_n \in \mathcal{T}(C)$, tels que $u_1 : \tau_1, \ldots, u_n : \tau_n$, le terme $f(u_1, \ldots, u_n)$ est réductible par R.*

La complétude locale de tous les symboles définis implique la complétude globale du système, à condition que le système soit fortement normalisant.

Théorème 1 `(théorème de complétude)`
Soit R un système de réécriture du premier ordre convergent. R est globalement complet si et seulement si R définit tous les symboles $f \in D$ d'une façon localement complète.

Preuve. Il est évident que la complétude globale implique la complétude locale de tous les symboles.

Afin de démontrer la réciproque il suffit de prouver que tout terme clos irréductible est un terme constructeur. Supposons que t soit un terme irréductible qui contient des symboles définis. Nécessairement il existe dans t un sous-terme $f(u_1, \ldots, u_n)$ qui commence par un symbole défini dont tous les arguments $u_1, \ldots, u_n$ sont des termes constructeurs. Puisque t est clos, ses sous-termes $u_1, \ldots, u_n$ le sont aussi et par la complétude locale de f, le terme $f(u_1, \ldots, u_n)$ est réductible, ce qui contredit l'irréductibilité de t. $\square$

2.4 Procédure de vérification

Par la suite on va présenter une procédure qui, pour une fonction f donnée, vérifie la complétude locale de f. La procédure crée un arbre de motifs, dont la racine est le motif le plus général $f(x_1, \ldots, x_n)$. Les fils d'un nœud interne sont des motifs plus précis que leur père, mais l'ensemble de termes filtrés par les fils est le même que celui filtré par leur père. Les feuilles de l'arbre sont des motifs réductibles, ou bien ceux dont le développement n'aboutirait jamais à un motif réductible.

Avant d'expliquer comment détecter ces derniers, on doit introduire la notion de variable d'induction :

Définition 2.4.1 `(variable d'induction)`
Soit $f \in D$, soit $t = f(t_1, \ldots, t_n)$ un terme linéaire, où $t_1, \ldots, t_n \in \mathcal{T}(C, \mathcal{X})$. Soit p une position dans t telle que $t|_p = x$ soit une variable. La variable x est une variable d'induction *dans t s'il existe une règle $l_i \to r_i \in R$ telle que t et l_i soient unifiables et que p soit une position dans l_i telle que $l_i|p$ ne soit pas une variable.*

Exemple 1. On définit le système suivant dont le type de base est << mots binaires >> :

$$\begin{array}{rcl} T & = & \{\mathtt{bin}\} \\ C & = & \{\epsilon : \mathtt{bin},\ 0 : \mathtt{bin} \to \mathtt{bin},\ 1 : \mathtt{bin} \to \mathtt{bin}\} \\ D & = & \{g : \mathtt{bin} \times \mathtt{bin} \to \mathtt{bin},\ h : \mathtt{bin} \times \mathtt{bin} \times \mathtt{bin} \to \mathtt{bin}\} \end{array}$$

avec le système de règles R suivant :

$$\begin{array}{rrcl} 1: & g(\,1(x),\, y\,) & \to & 0(\epsilon) \\ 2: & g(\,0(\epsilon),\, y\,) & \to & 1(\epsilon) \\ 3: & g(\,x,\, \epsilon\,) & \to & x \\ 4: & h(\epsilon, y, z) & \to & g(y, z) \\ 5: & h(0(x), y, z) & \to & x \\ 6: & h(1(x), y, z) & \to & x \end{array}$$

Dans le terme $g(x, y)$, x et y sont des variables d'induction, mais dans $g(1(x), y)$, seulement y. Le terme $g(0(x), 1(y))$ a une seule variable d'induction x, qui est aussi la seule variable d'induction de $h(x, y, z)$.

Ce sont bien évidemment des motifs possédant des variables d'induction qui vont être développés au cours de la construction de l'arbre de motifs. Les autres motifs sont ou bien réductibles, ou bien leur développement ne donne plus aucun espoir pour la réductibilité. Une variable d'induction de type ι peut être remplacé par :

Définition 2.4.2 `(ensemble couvrant)`
Soit $t = f(u_1, \ldots, u_n)$ pour $f \in D$ et $u_1, \ldots, u_n \in \mathcal{T}(C, \mathcal{X})$. Soit p une position de variable d'induction x^ι dans t. Un ensemble fini de termes non-variables CS est un ensemble couvrant *de x^ι dans t ssi pour toute substitution σ telle que $y\sigma \in \mathcal{T}(C)$ si $y \in FV(t)$, il existe un terme $u \in CS$ et une substitution $\hat{\sigma}$ tels que $t\sigma = t[u]_p\hat{\sigma}$.*

Dans le cas du premier ordre un bon candidat pour un ensemble couvrant est un ensemble de tous les constructeurs du type donné :

Proposition 2.4.3 `(ensemble couvrant simple)`
Soit $CS_\emptyset(\iota)$ un ensemble de tous les termes de la forme $c(x_1^{\iota_1}, \ldots, x_m^{\iota_m})$ pour $x_i^{\iota_i}$ nouvelles et distinctes, où $c : \iota_1 \times \cdots \times \iota_m \to \iota$ est un constructeur. $CS_\emptyset(\iota)$ est un ensemble couvrant de toute variable d'induction de type ι. □

Le nom d'ensemble couvrant << simple >>, ainsi que l'indice $\emptyset$ viennent du fait que le même ensemble sera utilisé également à l'ordre supérieur, à coté des autres ensembles couvrants dont la construction dépendra du contexte de la variable d'induction remplacée. Ces derniers seront décorés par l'information venant du contexte et leur nom figurera les endroits potentiels d'utilisation.

Maintenant on peut présenter la procédure de vérification, qui développe un arbre de motifs. Elle travaille sur des triplets d'ensembles : nœuds de travail (N_T), feuilles réductibles (F_r) et feuilles irréductibles (F_{ir}).

(succès) $$\frac{(\emptyset, F_r, \emptyset)}{\textbf{succès}}$$

(échec) $$\frac{(\emptyset, F_r, F_{ir})}{F_{ir}}$$

(décompose) $$\frac{(N_T \cup \{u\}, F_r, F_{ir})}{(N_T \cup \{u[v_i]_p\}_{i \in I}, F_r, F_{ir})}$$
$$\text{si} \begin{cases} u \text{ n'est pas réductible par } R \\ u|_p = x \text{ est une variable d'induction de } u \end{cases}$$
$$\text{où} \begin{cases} \{v_i\}_{i \in I} \text{ est un ensemble couvrant de } u|_p \\ \text{les variables libres de } \{v_i\}_{i \in I} \text{ sont nouvelles} \end{cases}$$

(feuille-réd) $$\frac{(N_T \cup \{u\}, F_r, F_{ir})}{(N_T, F_r \cup \{u\}, F_{ir})}$$
si u est réductible par R

(feuille-irréd) $\dfrac{(N_T \cup \{u\}, F_r, F_{ir})}{(N_T, F_r, F_{ir} \cup \{u\})}$

$$\text{si } \begin{cases} u \text{ est irréductible par } R \\ u \text{ n'a pas de variables d'induction} \end{cases}$$

Pour tester la complétude locale de définition d'une fonction $f \in \mathcal{F}$, où $f : \iota_1 \times \cdots \times \iota_n \to \iota$ il suffit de lancer la procédure ci-dessus avec le triplet suivant :

$$(\{f(x_1^{\iota_1}, \ldots, x_n^{\iota_n})\}, \emptyset, \emptyset)$$

Pendant le déroulement de la procédure la structure complète de l'arbre n'est pas mémorisée. Une fois un nœud développé (par la règle **(décompose)**) seuls ses fils restent dans l'ensemble N_T. Donc à tout instant $N_T \cup F_r \cup F_{ir}$ représente une << frontière >> de l'arbre en cours de construction.

Lorsque la construction de l'arbre est terminée et toutes les feuilles classées << réductible >> ou << irréductible >>, l'une des deux premières règles est déclenchée. Le **succès** signifie que tous les motifs sont réductibles, et sinon la procédure retourne l'ensemble de cas pour lesquels la fonction f n'est pas définie.

Exemple 1 (suite). On reprend l'exemple de la page 7. Voici la construction de l'arbre de motifs de la fonction g. Les variables d'induction sont indiquées par un caractère gros.

$g(\mathbf{x}, \mathbf{y})$
  $g(\mathbf{x}, \epsilon)$ réductible par 3
  $g(\mathbf{x}, 0(y))$
    $g(\epsilon, 0(y))$ irréductible
    $g(0(\mathbf{x}), 0(y))$
      $g(0(\epsilon), 0(y))$ réductible par 2
      $g(0(0(x)), 0(y))$ irréductible
      $g(0(1(x)), 0(y))$ irréductible
    $g(1(x), 0(y))$ réductible par 1
  $g(\mathbf{x}, 1(y))$
    $g(\epsilon, 1(y))$ irréductible
    $g(0(\mathbf{x}), 1(y))$
      $g(0(\epsilon), 1(y))$ réductible par 2
      $g(0(0(x)), 1(y))$ irréductible
      $g(0(1(x)), 1(y))$ irréductible
    $g(1(x), 1(y))$ réductible par 1

Feuilles irréductibles : $g(\epsilon, 0(y)),\ g(0(0(x)), 0(y)),\ g(0(1(x)), 0(y))$
$g(\epsilon, 1(y)),\ g(0(0(x)), 1(y)),\ g(0(1(x)), 1(y))$

Remarquons que le choix de la variable d'induction $\mathbf{y}$ dans la première ligne est arbitraire. Le choix complémentaire créerait un arbre différent, mais le résultat de la procédure serait toujours le même. Cependant il est possible d'avoir á chaque fois un ensemble de motifs irréductibles différents (s'il y avait une seule règle $g(\epsilon, \epsilon) \to \epsilon$ par exemple), mais dernière règle utilisée, **(succès)** ou **(échec)** est toujours la même.

La vérification de la fonction h est beaucoup plus simple et retourne un **succès** :

$$\begin{array}{l} h(\mathbf{x}, y, z) \\ \quad h(\epsilon, y, z) \text{ réductible par } 4 \\ \quad h(0(x), y, z) \text{ réductible par } 5 \\ \quad h(1(x), y, z) \text{ réductible par } 6 \end{array}$$

Toutes les feuilles réductibles : **succès**

2.5 Correction de la procédure de vérification

On va maintenant démontrer la correction de la procédure. Le premier lemme explique la signification de la réponse **succès**.

Lemme 2.5.1 *Soit $f \in D$. Si la procédure de vérification lancée pour f retourne un* **succès***, alors le système R définit f d'une façon localement complète.*

Preuve. Soit $Arg(f)$ l'ensemble de toutes les instances constructeurs closes de $f(x_1^{\iota_1}, \ldots, x_n^{\iota_n})$, donc des termes $f(u_1, \ldots, u_n)$ pour $u_1, \ldots, u_n \in \mathcal{T}(C)$. Vue la définition 2.4.2 de l'ensemble couvrant, l'utilisation de la règle **(décompose)** ne diminue pas l'ensemble des instances constructeurs de tous les termes N_T. Comme la règle **(feuille-réd)** ne change pas l'ensemble $N_T \cup F_r \cup F_{ir}$, donc pendant tout déroulement de la procédure, l'invariant suivant est satisfait :

tout terme $t \in Arg(f)$ est une instance d'un terme $\hat{t} \in N_T \cup F_r \cup F_{ir}$

Donc si la règle **(succès)** est déclenchée, alors tout terme de l'ensemble $Arg(f)$ est réductible et donc la définition de f est localement complète. □

La correction de la réponse négative est expliquée par le lemme suivante :

Lemme 2.5.2 *Soit $f \in D$. Si la procédure de vérification lancée pour f, utilisant des ensembles couvrants simples $CS_\emptyset$ retourne un ensemble de feuilles irréductibles F_{ir}, alors la définition de f n'est pas localement complète.*

Preuve. Soit $t \in F_{ir}$. La condition sur la règle **(feuille-irréd)** dit qu'il n'y a pas de variables d'induction dans t. Montrons que t n'unifie avec aucun coté gauche de R. Par contradiction, supposons que l_i soit unifiable avec t. D'après la définition 2.4.1 de variable d'induction, il y a deux possibilités :

- t est clos,
- la position de chaque variable $x \in FV(t)$ n'est pas une position stricte de l_i.

Puisque le terme l_i est linéaire, dans les deux cas t est une instance de l_i, alors t est réductible, contradiction. Donc chaque instance constructeur close de t est irréductible. Comme il existe des termes constructeurs clos de chaque type et vu la définitions de $CS_\emptyset$ (voir Proposition 2.4.3), il y a des instances closes de t de la forme $f(u_1, \ldots, u_n)$ pour $u_1, \ldots, u_n \in \mathcal{T}(C)$ qui ne sont pas réductibles. □

Pour finir la preuve de correction de la procédure de vérification on doit encore démontrer sa terminaison.

Lemme 2.5.3 *La procédure de vérification lancée pour un symbole de fonction f donné termine.*

Preuve. Les règles **(succès)** et **(échec)** ne peuvent être utilisées qu'une fois. Les règles **(feuille-irréd)** et **(feuille-réd)** diminuent l'ensemble N_T et seule la règle **(décompose)** pourrait éventuellement boucler. Mais comme un ensemble couvrant est composé de termes non-variables, la règle **(décompose)** fait grossir les termes dans N_T. Comme la position d'une variable d'induction a une profondeur bornée par celle des règles, les termes dans N_T ne peuvent grossir à l'infini. Alors, comme l'ensemble N_T est toujours fini, par le lemme de König, tout déroulement de la procédure doit être fini. □

Grâce au théorème 1 ainsi qu'aux lemmes 2.5.1, 2.5.2 et 2.5.3, afin de vérifier la complétude globale d'un système R linéaire fortement normalisant, il suffit de lancer la procédure de vérification pour tous les symboles définis. Le système est complet si et seulement si la procédure retourne un **succès** pour tous les symboles. Le cas échéant, elle nous fournit pour chaque symbole un ensemble de motifs non-spécifiés.

3 Ordre supérieur : types simples

Dans le cas de termes d'ordre supérieur, la situation est un peu plus complexe. Maintenant des fonctions peuvent prendre des arguments qui, eux aussi, sont des fonctions. Donc pour vérifier que la fonction donnée est définie pour tous ses arguments potentiels, il faut trouver un moyen de décrire d'une façon finie non seulement l'ensemble des arguments de type de base mais aussi les arguments fonctionnels.

3.1 Types et termes

Étant donné un ensemble fini de sortes B, on définit l'ensemble de types simples T par la grammaire suivante :

$$T ::= B \mid T \to T$$

Au cas d'ambiguïté on va écrire $T(B)$ pour l'ensemble de type simples construit à partir d'un ensemble de sortes B donné. Les types $\iota \in T$, qui ne contiennent aucun symbole $\to$, pour $\iota \in B$ (puisque $B \subseteq T$), vont être appelés types de base. Le symbole $\to$ est associatif à droite, donc on va écrire $\tau_1 \to \tau_2 \to \cdots \to \tau_n \to \tau$ pour $\tau_1 \to (\tau_2 \to \cdots \to (\tau_n \to \tau) \ldots)$, et τ va être appelé *type de sortie* si $\tau \in B$. Dans ce cas on va représenter ce type aussi par $\tau_1 \times \cdots \times \tau_n \to \tau$.

Les termes sont construits à partir d'un ensemble de variables $\mathcal{X}$, un ensemble de symbole de fonction $\mathcal{F}$, l'opérateur d'application -(-) et d'abstraction λ.

Comme dans la section précédente l'ensemble des variables est la somme disjointe des ensembles de variables de chaque type $\mathcal{X} = \bigcup_{\tau \in T} \mathcal{X}_\tau$. Remarquons qu'une variable peut désormais avoir un type d'ordre supérieur, par exemple $\mathtt{bin} \to \mathtt{bin}$.

Les symboles de fonctions dans l'ensemble fini $\mathcal{F}$ sont aussi équipés d'un type, $f : \tau_1 \times \cdots \times \tau_n \to \iota$. On va supposer toujours que $\iota \in B$, mais $\tau_1, \ldots, \tau_n$ peuvent être arbitraires.

On va travailler dans cette section avec des λ-termes algébriques, alors suivant la convention de [8], les éléments de $\mathcal{F}$ ne vont pas être tout simplement traités comme des constantes d'ordre supérieur. Dans des λ-termes algébriques un symbole n-aire f est toujours suivi de ses n-arguments. Le langage des termes est défini par la grammaire suivante :

$$\Lambda ::= \mathcal{X} \mid \Lambda(\Lambda) \mid \lambda\mathcal{X}.\Lambda \mid \mathcal{F}(\Lambda, \ldots, \Lambda)$$

On ne regarde pas l'arité de symboles de fonction pour l'instant, parce que cela est pris en charge par les règles de typage qui limitent l'ensemble Λ aux termes bien-typés. Afin de suivre la convention de termes algébriques, on va souvent écrire $u(v_1, \ldots, v_n)$ pour $u(v_1) \ldots (v_n)$. Pour éviter la confusion on peut toujours supposer que les variables libres et liées sont disjointes, et par conséquent, que les variables libres du terme u restent libres dans $t\{x \mapsto u\}$. Les positions dans les λ-termes sont aussi des suites d'entiers positifs, la construction λx étant considéré comme un symbole unaire pour tout $x \in \mathcal{X}$.

Les règles qui associent des types simples à certains termes sont les suivantes :

$$\textbf{(var)} \quad \frac{}{x^\tau : \tau}$$

$$\textbf{(fun)} \quad \frac{u_1 : \tau_1, \ldots, u_n : \tau_n}{f(u_1, \ldots, u_n) : \tau} \quad \text{pour} \quad \begin{array}{l} f \in \mathcal{F} \\ f : \tau_1 \times \cdots \times \tau_n \to \tau \end{array}$$

$$\textbf{(abs)} \quad \frac{u : \rho}{\lambda x^\tau . u : \tau \to \rho}$$

$$\textbf{(app)} \quad \frac{u : \tau \to \rho \qquad v : \tau}{u(v) : \rho}$$

L'ensemble de tous les termes bien-typés, pour $\mathcal{F}$ et $\mathcal{X}$ donnés est (comme auparavant) noté $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Les termes clos sont maintenant les termes qui ne contiennent pas de variables libres, mais ils peuvent contenir des variables liées (par exemple $\lambda x^\tau . x^\tau$ est un terme clos de type $\tau \to \tau$), les termes clos, bien-typés sont toujours noté $\mathcal{T}(\mathcal{F})$.

Comme dans la section précédente, l'ensemble de symboles $\mathcal{F}$ est divisé en deux parties disjointes : les constructeurs C et les symboles définis D. Maintenant on met une condition supplémentaire sur les constructeurs. Leurs types doivent être du premier ordre, donc si $c : \tau_1 \times \cdots \times \tau_n \to \tau$ est un constructeur, alors $\tau_1, \ldots, \tau_n, \tau \in B$. Comme dans la section précédente l'ensemble de termes constructeurs clos bien-typés est noté $\mathcal{T}(C)$ et on suppose toujours que pour chaque type de base $\iota \in B$ il existe un terme constructeur $t \in \mathcal{T}(C)$ tel que $t : \iota$.

3.2 Forme β-normale η-expansée

On va travailler presque uniquement sur les termes en forme β-normale η-expansée. Pour calculer cette forme on va utiliser les règles :

$$\begin{array}{rcl} (\lambda x^\tau . v)(u) & \longrightarrow_\beta & u\{x^\tau \mapsto v\} \\ C[u]_p & \longrightarrow_\eta & C[\lambda x^\tau . u(x^\tau)]_p \quad \text{pour} \quad x^\tau \notin FV(u) \end{array}$$

La règle de η-expansion ne pourra s'appliquer que sous les conditions suivantes :

- $u : \tau \to \sigma$
- u n'est pas une abstraction
- $C[u]|_q$ n'est pas une application pour $p = q \cdot 1$

On va noter $u\updownarrow_\beta^\eta$ la forme normale de u, et si on est sûr que la normalisation n'utilise que la règle η (notamment pour des variables d'ordre supérieur) on écrira $u\uparrow^\eta$; $\mathcal{T}(\mathcal{F}, \mathcal{X}) \updownarrow_\beta^\eta$ va signifier l'ensemble de termes bien-typés en forme normale.

A cause de confluence de règles β et η on obtient le lemme suivant :

Lemme 3.2.1 *Si u est un terme en forme β-normale η-expansée, de type $\tau_1 \times \cdots \times \tau_m \to \iota$ pour $m \geq 0$, $\iota \in B$, alors $u = \lambda x_1^{\tau_1} \ldots x_m^{\tau_m} . \phi(u_1, \ldots, u_n)$, $\phi \in \mathcal{F} \cup \mathcal{X}$, $\phi : \rho_1 \times \cdots \times \rho_n \to \iota$ et les termes $u_1, \ldots u_n$ sont eux aussi en forme normale.*

Corollaire 3.2.2 *Soient s, t deux termes en forme normale tels que $s|_p$ et t soient de même type. Alors $s[t]_p$ est aussi en forme normale.*

3.3 Réécriture d'ordre supérieur de termes normalisés

La définition d'un système de réécriture reste la même que celle de premier ordre. Un système de réécriture d'ordre supérieur est donc un ensemble fini de règles $\{l_i \to r_i\}$, où l_i, r_i sont des termes d'ordre supérieur, normalisés. Outre des conditions nécessaires (l_i et r_i de même type, $FV(r_i) \subseteq FV(l_i)$, $l_i \notin \mathcal{X}$, $l_i \neq c(u_1, \ldots, u_n)$ pour $c \in C$), qui ont été déjà introduites dans la section précédente, on va restreindre la forme du coté gauche aux *patterns*.

Définition 3.3.1 (`pattern`)
Un terme normalisé t est un pattern *ssi pour tout sous-terme s de t, si $s = Y(u_1, \ldots, u_n)$ pour $Y \in FV(t)$, la suite $u_1, \ldots, u_n$ est η-équivalente à une suite de variables liées distinctes.*

Voici quelque exemples de << patterns >> : $\lambda x.Y(\lambda z.x(z))$, $\lambda x.c(\lambda z.Y(z, x))$, Y, $\lambda x.c(x)$ et quelque termes qui ne le sont pas : $\lambda x.Y(x, x)$, $\lambda xy.Y(x, c)$, $\lambda x.Y(Z(x))$, $Y(d)$. Les << patterns >> ont toutes les bonnes propriétés de décidabilité des termes de premier ordre, l'unification peut être calculée en temps quasi-linéaire [16], et cette classe de termes est assez large pour la plupart des application.

La définition de la relation de réécriture associée à un système de règles donné est une simple généralisation de celle de premier ordre. Un terme normalisé $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ se réécrit en un terme t en utilisant la règle $l \to r$ à la position p, ce qui va être noté $s \xrightarrow[l \to r]{p} t$ (où tout simplement $s \longrightarrow t$), s'il existe une substitution σ telle que $s|_p =_{\beta\eta} l\sigma$ et $t = s[r\sigma\updownarrow_{\beta}^{\eta}]_p$. La différence se manifeste seulement par le fait que le coté gauche de la règle doit être égal à un sous-terme approprié modulo les règles β et η, et qu'on normalise le coté droit substitué, avant de le mettre dans le terme réécrit. D'après le corollaire 3.2.2, le terme t est normalisé.

3.4 Complétude

La notion de complétude globale diffère légèrement de celle de premier ordre. Comme on a dit dans l'introduction, on ne va plus exiger des termes fonctionnels ($\lambda x.f(x)$, pour $f \in D$, par exemple) de se réécrire en termes constructeurs. Seuls les termes de type de base doivent avoir une forme normale dans l'ensemble de valeurs du premier ordre $\mathcal{T}(C)$.

Définition 3.4.1 (`complétude globale`)
Un système de réécriture R est complet *ssi pour tout terme clos $t \in \mathcal{T}(\mathcal{F})$ de type de base il existe un terme $s \in \mathcal{T}(C)$ tel que $t \longrightarrow_R^* s$.*

De même que dans le premier ordre on va introduire une notion de complétude locale de définition d'une fonction, mais ici on doit également faire un petit changement. Dans le premier ordre on a défini la complétude locale d'une fonction f, comme la réductibilité des tous les termes $f(u_1, \ldots, u_n)$ pour $u_1, \ldots, u_n$ construits à partir de constructeurs. Ici on permet aux arguments fonctionnels d'être arbitraires. Cette nouvelle définition vient du même raisonnement que celle de complétude globale.

Définition 3.4.2 (`$f$-terme canonique`)
Soit $f : \tau_1 \times \cdots \times \tau_n \to \iota$ où $\iota \in B$. Un terme normalisé $u = f(u_1, \ldots, u_n)$ va être appelé f-terme canonique *si pour tout $i = 1 \ldots n$, u_i une des conditions suivantes est satisfaite :*

- *u_i a un type fonctionnel*
- *u_i a un type de base et $u_i \in \mathcal{T}(C, \mathcal{X})$*

Les arguments $u_1, \ldots, u_n$ de f construit comme dans la définition du f-terme canonique vont être appelés *arguments canoniques.*

Pour un terme avec variables libres $v = f(v_1, \ldots, v_n)$, une substitution σ telle que $v\sigma \updownarrow_\beta^\eta$ soit un f-terme canonique, va être appelée une *substitution canonique* de v.

La complétude locale signifie bien sûr la réductibilité de tous les f-termes canoniques clos :

Définition 3.4.3 (`complétude locale`)
Un système de réécriture R définit une fonction $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \iota$ d'une façon localement complète *ssi tout f-terme canonique clos est réductible par R.*

Heureusement à l'ordre supérieur le rapport entre la complétude locale de définition de tous les symboles et la complétude globale du système fortement normalisant reste le même qu'au premier ordre :

Théorème 2 (`théorème de complétude`)
Soit R un système de réécriture d'ordre supérieur convergent. R est globalement complet si et seulement si R définit tous les symboles $f \in D$ d'une façon localement complète.

Preuve. Comme dans la section précédente la difficulté existe seulement dans un sens. On va utiliser la recurrence sur l'union de l'ordre de sous-terme et de l'ordre de réduction, impliqué par la relation noetherienne $\longrightarrow_R$. Le fait que cette union est bien-fondée peut être trouvé dans [4]. On va montrer que pour chaque terme de type de base t, il existe un terme $t' \in \mathcal{T}(C)$ tel que $t \longrightarrow_R^* t'$, si tous les symboles sont localement complets.

Soit t un terme clos de type de base. Si t=$c(s_1, \ldots, s_n)$, pour $c \in C$, alors les sous-termes $s_1, \ldots, s_n$ de t sont aussi de type de base. Par hypothèse de récurrence, il existe des termes $s'_1, \ldots, s'_n \in \mathcal{T}(C)$ tels que pour tous i, $s_i \longrightarrow_R^* s'_i$. Donc $t' = c(s'_1, \ldots, s'_n)$ et bien évidemment $t' \in \mathcal{T}(C)$ et $t \longrightarrow_R^* t'$.

Si $t = f(s_1, \ldots, s_n)$, pour $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \iota$, alors pour tout i tel que $\tau_i \in B$ (par hypothèse de récurrence) il existe des termes $s'_i \in \mathcal{T}(C)$ tels que $s_i \longrightarrow_R^* s'_i$. Pour les autres i on pose $s'_i = s_i$. Soit $t_1 = f(s'_1, \ldots, s'_n)$. Comme f est localement complète, t_1 est réductible, disons $t_1 \longrightarrow_R t_2$. Comme t_2 est strictement plus petit que t dans l'ordre de réduction (puisque $t \longrightarrow_R^* t_1 \longrightarrow_R t_2$), par hypothèse de récurrence, il existe un terme $t_3 \in \mathcal{T}(C)$ tel que $t_2 \longrightarrow_R^* t_3$. Donc comme $t \longrightarrow_R^* t_2 \longrightarrow_R^* t_3$, on obtient le résultat.

Le cas de base est un terme irréductible sans sous-termes, donc une constante qui appartient à C. □

3.5 Procédure de vérification

Cette section présente une adaptation de la procédure de vérification de la complétude locale pour la réécriture d'ordre supérieur. On va d'abord définir une notion auxiliaire :

Définition 3.5.1 `(sous-terme transparent)`
Soit u un terme normalisé, et p une position dans u. On va appelle $u|_p$ un sous-terme transparent *de u s'il n'y a pas de variables libres sur le chemin entre la racine et le sous-terme $u|_p$ et si $u|_p = X(x_1{\uparrow^\eta}, \ldots, x_n{\uparrow^\eta})$ où $X : \tau_1 \times \cdots \times \tau_n \to \iota$, pour $\iota \in B$, est une variable libre linéaire dans u et $x_1^{\tau_1}, \ldots, x_n^{\tau_n}$ sont des variables liées distinctes.*

On redéfinit maintenant quelque notions de la section précédente.

Définition 3.5.2 `(variable d'induction)`
Soit $f \in D$. Étant donné un f-terme normalisé u et une position p, tels que $u|_p =_\eta X(x_1, \ldots, x_n)$ soit un sous-terme transparent, on appelle X variable d'induction *s'il existe une règle $l_i \to r_i$ dans R, telle que u soit unifiable avec l_i et $l_i|p$ ne soit pas un sous-terme transparent de la forme (η-égale à) $Y(y_1, \ldots, y_n)$, pour Y libre et $y_1, \ldots, y_n$ liées.*

Au cours de la construction de l'arbre de motifs, les sous-termes transparents dont le symbole de tête est une variables d'induction vont être remplacés par des ensembles couvrants. Mais la définition d'ensemble couvrant doit être adaptée à la notion de f-terme canonique.

Définition 3.5.3 `(ensemble couvrant)`
Soit $f : \tau_1 \times \cdots \times \tau_n \to \iota$ où $\iota \in B$. Soit $u = f(u_1, \ldots, u_n)$ un f-terme canonique et p une position dans u, telle que $u|_p$ soit un sous-terme transparent. L'ensemble fini CS de termes linéaires normalisés va être appelé ensemble couvrant *de $u|_p$, ssi pour toute substitution canonique close σ de u il existe un terme $v \in CS$ et une substitution $\hat{\sigma}$ telle que $u\sigma \updownarrow_\beta^\eta = u[v]_p \hat{\sigma} \updownarrow_\beta^\eta$.*

La procédure de vérification de la section 2 (dans la page 8) n'est presque pas modifiée. Ce qui change est la définition de variable d'induction et donc la règle **(décompose)** qui l'utilise, ainsi que les définitions des ensembles couvrants qui vont être étudiés dans les sous-sections qui suivent, pour les différentes approches traitant les arguments fonctionnels.

$$\textbf{(succès)} \quad \frac{(\emptyset, F_R, \emptyset)}{\textbf{succès}}$$

$$\textbf{(échec)} \quad \frac{(\emptyset, F_R, F_I)}{F_I}$$

$$\textbf{(décompose)} \quad \frac{(N_T \cup \{u\}, F_R, F_I)}{(N_T \cup \{u[v_i]_p\}_{i \in I}, F_R, F_I)}$$

$$\text{si} \begin{cases} u \text{ n'est pas réductible par } R \\ u|_p = X(x_1{\uparrow^\eta}, \ldots, x_n{\uparrow^\eta}) \text{ est un sous-terme transparent} \\ X \text{ est une variable d'induction de } u \end{cases}$$

$$\text{où} \begin{cases} \{v_i\}_{i\in I} \text{ est un ensemble couvrant de } u|_p \\ \text{les variables libres de } \{v_i\}_{i\in I} \text{ sont nouvelles} \end{cases}$$

(feuille-réd) $\dfrac{(N_T \cup \{u\}, F_R, F_I)}{(N_T, F_R \cup \{u\}, F_I)}$
si u est réductible par R

(feuille-irréd) $\dfrac{(N_T \cup \{u\}, F_R, F_I)}{(N_T, F_R, F_I \cup \{u\})}$
$$\text{si} \begin{cases} u \text{ est irréductible par } R \\ u \text{ n'a pas de variables d'induction} \end{cases}$$

Comme auparavant, pour tester la complétude locale de la définition de la fonction $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \iota$, on lance la procédure avec le triplet initial :

$$\{(f(x_1^{\tau_1}\uparrow^\eta, \ldots, x_n^{\tau_n}\uparrow^\eta), \emptyset, \emptyset)\}$$

La correction de la réponse positive de la procédure peut être démontrée maintenant, avant de présenter les ensembles couvrants utilisés. L'argumentation reste la même que dans le cas de premier ordre.

Lemme 3.5.4 *Soit $f \in D$. Si la procédure de vérification lancée pour f retourne un* **succès***, alors le système R définit f d'une façon localement complète.*

Preuve. On va montrer que l'application successive des règles **(décompose)** et **(feuille-réd)** ne viole pas l'invariant suivant :

tout f-terme canonique clos t est une instance d'un terme $\hat{t} \in N_T \cup F_r \cup F_{ir}$

Pour la règle **(feuille-réd)** c'est évident puisqu'elle ne change pas l'ensemble $N_T \cup F_r \cup F_{ir}$.

La règle **(décompose)** change l'ensemble N_T, mais vu la définition 3.5.3 de l'ensemble couvrant, ne diminue pas l'ensemble de f-termes canoniques qui sont instances de N_T.

Chaque déroulement de la procédure qui résulte en un **succès** est une suite d'applications des règles **(décompose)** et **(feuille-réd)** suivi par **(succès)**. La règle **(feuille-irréd)** ne peut pas être appliquée puisque à la fin on a $F_{ir} = \emptyset$, pour déclencher la règle **(succès)**.

Comme tout terme canonique clos est un instance du terme initiale, l'invariant est satisfait et toutes les feuilles sont réductibles, donc chaque f-terme canonique clos est réductible. □

3.6 Approche classique sur les arguments fonctionnels

Cette sous-section va considérer l'approche classique qui est utilisé dans des langages fonctionnels existants. Il n'est pas permis de regarder la structure interne des arguments fonctionnels qui sont traités comme des << boîtes noires >> et la pleine puissance du filtrage d'ordre supérieur n'est pas utilisée.

Définition 3.6.1 *Un système de réécriture $R = \{l_i \to r_i\}$ est appelé* classique *si toutes les règles satisfont la condition suivante : si $l_i = f(u_1, \ldots, u_n)$ pour $f : \tau_1 \times \cdots \times \tau_n \to \iota$ alors tous les sous-termes u_j tels que $\tau_j \notin B$ sont (η-équivalents à) des variables linéaires.*

On va montrer que dans ce cas toutes les variables d'induction sont du premier ordre et qu'on peut utiliser l'ensemble couvrant simple $CS_\emptyset$ (de la proposition 2.4.3) dans la procédure de vérification.

Lemme 3.6.2 *Soit R un système de réécriture classique, soit $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \iota$ pour $\iota \in B$ et u un f-terme canonique. Soit p une position d'un sous-terme transparent dans u avec une variable d'induction X en tête. Alors, le type de X est un type de base.*

Preuve. Soit $u = f(u_1, \ldots, u_n)$ et l_j le coté gauche (d'une règle de R) qui s'unifie avec u. Puisque p n'est pas la position de la racine, donc $p = i \cdot q$ pour $i \in 1 \ldots n$. Supposons que τ_i ne soit pas de type de base, $\tau_i = \rho_1 \times \cdots \times \rho_m \to \rho$, pour $\rho \in B$. Le symbole de tête de l_j est f et alors $l_j = f(t_1, \ldots, t_n)$, où $t_i = \lambda x_1^{\rho_1} \ldots x_m^{\rho_m}.Y(x_1, \ldots, x_m)$ pour une variable $Y : \tau_i$ linéaire dans l_j. Comme u_i est normalisé, alors $u_i = \lambda x_1^{\rho_1} \ldots x_m^{\rho_m}.u_i'$. Donc la position q dans u_i doit être au moins aussi profonde que la position du sous-terme transparent $Y(x_1, \ldots, x_m)$ dans t_i, contradiction.

Donc τ_i est un type de base. Puisque les constructeurs ont des types du premier ordre, il n'y a pas de λ sur le chemin de la racine vers p. Donc $u|_p = X$, parce qu'il n'y a pas de variables liées qui peuvent être arguments de X. □

Alors la procédure de vérification utilisée pour tester les fonctions définies par des systèmes de réécriture classique n'est pas différente de celle pour le premier ordre. Nous avons donc gratuitement les lemmes manquants (2.5.2 correction de l'échec et 2.5.3 terminaison) pour prouver la correction de la procédure pour des systèmes classiques linéaires.

La bonne propriété des systèmes classiques, mentionnée dans l'introduction, c'est la modularité.

Lemme 3.6.3 *Soit R_1 et R_2 deux systèmes classiques convergeants, sur le même ensemble de types T sur les signatures $\mathcal{F}_1$ et $\mathcal{F}_2$ respectivement, tels que $\mathcal{F}_1 \cap \mathcal{F}_2 = C_1 = C_2$. Si R_1 et R_2 sont complets et $R_1 \cup R_2$ termine, alors $R_1 \cup R_2$ est aussi complet.*

Preuve. Soit $R = R_1 \cup R_2$, $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. On va prouver par contradiction que chaque terme de type de base clos irréductible est un terme constructeur. Soit t un terme irréductible, qui contient des symboles non-constructeurs, minimal dans le sens de sous-terme, donc tel que tous ses sous-termes immédiat de premier ordre sont constructeurs. On a donc $t = f(t_1, \ldots, t_n)$ pour $f \in D_1$ (par exemple), $f : \tau_1 \times \cdots \times \tau_n \to \iota$ pour $\iota \in B$. Pour tout i, tel que $\tau_i \in B$, on a $t_i \in \mathcal{T}(C)$ et donc on pose $s_i = t_i$. Pour les autres i, s_i est un terme arbitraire (de type fonctionnel) clos, irréductible, mais $s_i \in T(\mathcal{F}_1)$. Comme R_1 est complet, il y a une règle $l_j \to r_j$ qui réduit le terme $s = f(s_1, \ldots, s_n)$ et elle peut être appliquée seulement en tête. Mais R_1 est un système classique et ne fait pas de filtrage sur les arguments fonctionnels. Donc l_j doit filtrer aussi le terme t. Contradiction. □

3.7 Approche moins restrictive

Dans cette approche, on ne met aucune condition supplémentaire sur la forme des règles (sauf celle que les membres gauches doivent être des << patterns >>), donc pour les arguments de type fonctionnel on a besoin de l'ensemble couvrant suivant, qui remplace un sous-terme transparent $X(x_1^{\tau_1}\uparrow^\eta, \ldots, x_n^{\tau_n}\uparrow^\eta) : \iota$.

Définition 3.7.1 (grand ensemble couvrant)
Le grand ensemble couvrant, $CS_{x_1^{\tau_1},\ldots,x_n^{\tau_n}}(\iota)$ *contient tous les termes de la forme*

$$f(X_1(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta, \ldots, X_m(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta)$$

pour $f \in \mathcal{F}$, $f : \rho_1 \times \ldots \times \rho_m \to \iota$ *et* $X_j : \tau_1 \to \cdots \to \tau_n \to \rho_i$ *distinctes.*

Il contient également tous les termes de la forme :

$$x_i^{\tau_i}(X_1(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta, \ldots, X_m(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta)$$

pour x_i *variable liée de type* $\tau_i = \rho_1 \times \ldots \times \rho_m \to \iota$ *(*$X_1, \ldots, X_m$ *comme avant).*

Sa correction (vis-à-vis la définition 3.5.3) est démontrée par la proposition suivante :

Proposition 3.7.2 *Soit* $f : \xi_1 \times \cdots \times \xi_n \to \xi$ *où* $\xi \in B$. *Soit* $u = f(u_1, \ldots, u_k)$ *un* f*-terme canonique et* p *une position dans* u, *telle que* $p = i \cdot q$ *pour* $\xi_i \notin B$, *et telle que* $u|_p$ *est un sous-terme transparent* η*-égale à* $X(x_1^{\tau_1}, \ldots, x_n^{\tau_n})$, *pour* $X : \tau_1 \times \cdots \times \tau_n \to \iota$, $\iota \in B$. *L'ensemble* $CS_{x_1^{\tau_1},\ldots,x_n^{\tau_n}}(\iota)$ *est un ensemble couvrant de* $u|_p$.

Preuve. Soit σ une substitution canonique close de u. D'après le lemme 3.2.1, $X\sigma\updownarrow_\beta^\eta = \lambda y_1^{\tau_1} \ldots y_n^{\tau_n}.\phi(t_1, \ldots, t_n)$ pour $\phi \in \mathcal{F} \cup \mathcal{X}$. La substitution est close, donc il y a deux possibilités :

- ϕ est une variable liée, $\phi = y_j^{\tau_j}$ et dans ce cas $\tau_j = \rho_1 \times \cdots \times \rho_m \to \iota$ on prend $v = x_j^{\tau_i}(X_1(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta, \ldots, X_m(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta)$
- ϕ est un symbole de fonction, $\phi = g$ pour $g : \rho_1 \times \cdots \times \rho_m \to \iota$ et on prend $v = g(X_1(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta, \ldots, X_m(x_1^{\tau_1}, \ldots, x_n^{\tau_n})\uparrow^\eta)$

Dans les deux cas, on pose

$$\begin{aligned} X_1\hat{\sigma} &= \lambda y_1^{\tau_1} \ldots y_n^{\tau_n}.t_1 \\ &\vdots \\ X_m\hat{\sigma} &= \lambda y_1^{\tau_1} \ldots y_n^{\tau_n}.t_m \end{aligned}$$

Pour les $Y \in Dom(\sigma)$ tels que $Y \neq X$ on pose $Y\hat{\sigma} = Y\sigma$. Il est évident que $u\sigma\updownarrow_\beta^\eta = u[v]_p\hat{\sigma}\updownarrow_\beta^\eta$. □

La terminaison de la procédure est établie par le lemme suivant, dont la preuve est presque identique à celle du premier ordre.

Lemme 3.7.3 *La procédure de vérification, qui utilise les ensembles couvrants* $CS_\emptyset$ *et* $CS_{\overline{x}}$, *lancée pour un symbole de fonction* f *donné, termine.*

Preuve. Comme les ensembles couvrants $CS_\emptyset$ et $CS_{\overline{x}}$ sont composés de termes non-transparents, la règle **(décompose)** fait grossir les termes dans N_T, dans le sens que le multi-ensemble de distances de la racine de sous-termes transparents dans les termes introduits dans N_T est plus grand que celui du terme supprimé. Comme la position d'un sous-terme transparent dominé par une variable d'induction a une profondeur bornée par celle des règles, les termes dans N_T ne peuvent grossir à l'infini. Comme l'ensemble N_T est toujours fini, par le lemme de König, toute suite d'applications de la règle **(décompose)** doit être fini. Puisque toute déroulement éventuel infini de la procédure doit comprendre une infinité d'applications de la règle **(décompose)**, on obtient le résultat. □

Malheureusement, dans cette approche on n'a pas de lemme de correction de la réponse négative. Il se peut que la procédure retourne des termes irréductibles, mais qu'il n'existe aucun f-terme canonique clos irréductible. Cependant, si un tel terme existe, il est forcément une instance d'un terme retourné par la procédure. La procédure peut donc être vue, comme un outil de preuve semi-automatique. Elle fait tout le travail mécanique et laisse seulement des cas plus difficiles pour l'utilisateur.

Exemple 2. Supposons avoir les définitions suivantes :

$$\begin{array}{rcl} B_2 & = & \{\alpha, \beta, \gamma\} \\ C_2 & = & \{a, b, c\} \\ D_2 & = & \{f, g\} \\ & & \\ & & \end{array} \qquad\qquad \begin{array}{rcl} a & : & \beta \to \alpha \\ b & : & \beta \\ f & : & (\beta \to \beta) \to \gamma \\ g & : & \alpha \to \beta \\ c & : & \gamma \end{array}$$

Considérons maintenant le système de règles suivant :

$$\begin{array}{rrcl} 1: & g(a(x)) & \to & b \\ 2: & f(\lambda x^\beta.x) & \to & c \\ 3: & f(\lambda x^\beta.b) & \to & c \end{array}$$

et procédons à la verification de complètude de définition de f.

$$\begin{array}{lll} f(\lambda x^\beta.\mathbf{X}(x)) & & \\ & f(\lambda x^\beta.x) & \text{réd. par 2} \\ & f(\lambda x^\beta.b) & \text{réd. par 3} \\ & f(\lambda x^\beta.g(Y(x))) & \text{irréd.} \end{array}$$

La procédure retourne le terme $f(\lambda x^\beta.g(Y(x)))$ qui est irréductible par notre système de règles. Mais la seule possibilité d'avoir une instance close du terme rétourné est de mettre à la place de $Y(x)$ le terme $a(\ldots)$, puisque l'argument de g doit être de type α et a seul possède un tel type de sortie. Dans ce cas :

$$f(\lambda x^\beta.g(a(\mathbf{X}(x)))) \underset{1}{\longrightarrow} f(\lambda x^\beta.b) \underset{3}{\longrightarrow} c$$

et on s'apperçoit que f est tout de même complètement définie.

Quand même pour un système linéaire, si dans un des termes retournés par la procédure toutes les variables libres sont de type de base, on peut trouver un

f-terme canonique clos irréductible. L'argument est le même que dans la preuve du lemme 2.5.2 de la section 2.

Comme l'on a dit dans l'introduction la complétude des systèmes sans restriction n'est pas une propriété modulaire.

Exemple 3. On définit un système R_1 des entiers avec la fonction Biz de type d'ordre supérieur (intuitivement $Biz(f) \longrightarrow^* f(1)$, mais ce n'est pas du tout important dans cet exemple) :

$$\begin{array}{rcl} T & = & \{\mathtt{nat}\} \\ C & = & \{0 : \mathtt{nat},\ succ : \mathtt{bin} \to \mathtt{bin}\} \\ D_1 & = & \{Biz : (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat}\} \end{array}$$

avec les règles :

$$\begin{array}{lrcl} 1: & Biz(\lambda x^{\mathtt{nat}}.x) & \to & succ(0) \\ 2: & Biz(\lambda x^{\mathtt{nat}}.0) & \to & 0 \\ 3: & Biz(\lambda x^{\mathtt{nat}}.succ(Y(x))) & \to & succ(Biz(\lambda x^{\mathtt{nat}}.Y(x))) \\ 4: & Biz(\lambda x^{\mathtt{nat}}.Biz(\lambda y^{\mathtt{nat}}.Y(y,\, x)) & \to & Biz(\lambda y^{\mathtt{nat}}.Y(y,\, succ(0))) \end{array}$$

Le système R_1 est complet, puisque sa seule fonction est localement complète. Examinons la construction de l'arbre de motifs :

$$\begin{array}{lll} Biz(\lambda x^{\mathtt{nat}}.\mathbf{Z}(x)) & & \\ & Biz(\lambda x^{\mathtt{nat}}.0) & \text{réd. par 2} \\ & Biz(\lambda x^{\mathtt{nat}}.succ(Y(x))) & \text{réd. par 3} \\ & Biz(\lambda x^{\mathtt{nat}}.Biz(\lambda y^{\mathtt{nat}}.Z(x,\, y)) & \text{réd. par 4} \\ & Biz(\lambda x^{\mathtt{nat}}.x) & \text{réd. par 1} \end{array}$$

Utilisant une fois la règle **(décompose)** pour remplacer le sous-terme transparent $Z(x)$ par l'ensemble couvrant $CS_x(\mathtt{nat})$, plusieurs fois la règle **(feuille-réd)** et une fois la règle **(succès)**, on a montré la complétude locale de la définition de Biz et donc (par le théorème 2) la complétude globale de R_1.

Maintenant on prend un autre système complet R_2, sur le même ensemble de types et de constructeurs, avec la fonction + :

$$D_2 \;=\; \{+ : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}\}$$

avec les règles bien connues :

$$\begin{array}{lrcl} 5: & x + 0 & \to & x \\ 6: & x + succ(y) & \to & succ(x + y) \end{array}$$

La somme de deux systèmes $R = R_1 \cup R_2$ n'est pas complète puisque le terme de type `nat`, $Biz(\lambda x^{\mathtt{nat}}.x + x)$, est une forme normale.

3.8 Nouvelle approche : << fonctions à syntaxe limitée >>

La troisième approche est un essai de joindre les avantage des deux premières. On permet l'utilisation du filtrage d'ordre supérieur, mais seulement pour les arguments de certains types fonctionnels. Pour chaqun de ces types on pré-définit l'ensemble de symboles qui peuvent être utilisés.

3.8.1 Extension du système de types

On va maintenant considérer l'ensemble de types simples construit à partir de l'ensemble de sortes $B = \mathit{SPEC} \cup \mathit{BASE}$, $\mathit{SPEC} \cap \mathit{BASE} = \emptyset$, pour un ensemble BASE de types de base et l'ensemble SPEC de nouveaux types fonctionnels. On peut distinguer maintenant les ensembles de types

$$\begin{aligned} T_{\to BASE} &= \{\tau \mid \tau = \rho_1 \times \cdots \times \rho_n \to \iota \text{ pour } \iota \in \mathit{BASE}\} \\ T_{\to SPEC} &= \{\tau \mid \tau = \rho_1 \times \cdots \times \rho_n \to s \text{ pour } s \in \mathit{BASE}\} \end{aligned}$$

D'une façon similaire on définit les ensembles $\mathcal{X}_{\to BASE}$ et $\mathcal{X}_{\to SPEC}$ (ainsi que $\mathcal{F}_{\to BASE}$ et $\mathcal{F}_{\to SPEC}$), comme ensembles de variables (symboles) de types $T_{\to BASE}$ et $T_{\to SPEC}$ respectivement.

A chaque type $s \in \mathit{SPEC}$ on va associer un vrai type fonctionnel $\mathit{Type}(s) \in T(\mathit{BASE})$, construit uniquement à partir de types de base et un ensemble de symboles $\mathit{symb}(s)$. Intuitivement les λ-termes de type $\mathit{Type}(s)$, construits à partir des $\mathit{symb}(s)$ vont avoir un type s. On va exiger que $\mathit{Type}(s)$ ainsi que le type de tout symbole $h \in \mathit{symb}(s)$ soient du premier ordre, donc de la forme $\iota_1 \times \cdots \times \iota_n \to \iota$ pour $\iota_1, \ldots, \iota_n, \iota \in \mathit{BASE}$, comme les types des constructeurs.

Pour indiquer le type d'une fonction $f \in \mathcal{F}_{\to SPEC}$ on va écrire à l'occasion $f : \tau_1 \times \cdots \times \tau_n \to [\iota_1 \times \cdots \times \iota_m \to \iota]_s$ si $f : \tau_1 \times \cdots \times \tau_n \to s$ et $\mathit{Type}(s) = \iota_1 \times \cdots \times \iota_m \to \iota$. Si un tel symbole f est suivi de tous ses $n + m$ arguments (ce qui va être souvent le cas), on va le noter $f(u_1, \ldots, u_n)(v_1, \ldots, v_m)$.

Exemple 4. Supposons avoir les définitions suivantes :

$$\begin{array}{rcl@{\qquad}rcl} \mathit{BASE}_4 &=& \{\mathtt{nat}\} & 0 &:& \mathtt{nat} \\ C_4 &=& \{0, s\} & s &:& \mathtt{nat} \to \mathtt{nat} \\ D_4 &=& \{+, *\} & +, * &:& \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat} \end{array}$$

On introduit un type spécial `Lin` — les fonctions linéaires d'une variable. On pose :

$$\begin{aligned} \mathit{SPEC}_4 &= \{\mathtt{Lin}\} \\ \mathit{Type}(\mathtt{Lin}) &= \mathtt{nat} \to \mathtt{nat} \\ \mathit{symb}(\mathtt{Lin}) &= \{0, s, +\} \end{aligned}$$

Les termes $\lambda x^{\mathtt{nat}}.0$, $\lambda x^{\mathtt{nat}}.s(x)$ et $\lambda x^{\mathtt{nat}}.x+s(x+s(s(0)))$ (on utilise une notation infixe pour $+$ et $*$) ont un type `Lin`, mais le terme $\lambda x^{\mathtt{nat}}.x * x$ utilise le symbole interdit $*$.

Étant donné un ensemble de symboles $\mathcal{F}$, avec des types de $T(\mathit{BASE} \cup \mathit{SPEC})$ associés à chaque symbole, en utilisant l'extension évidente de la fonction Type, on peut leur associer des types de $T(\mathit{BASE})$ de la manière suivante (on va utiliser le signe $\bar{:}$) :

Si $f : \tau_1 \times \cdots \times \tau_n \to \tau$ alors $f \mathrel{\bar{:}} \overline{\mathit{Type}}(\tau_1) \times \cdots \times \overline{\mathit{Type}}(\tau_n) \to \overline{\mathit{Type}}(\tau)$

où $\overline{\mathit{Type}}$ est une extension de Type sur $T(\mathit{BASE} \cup \mathit{SPEC})$ dans $T(\mathit{BASE})$ définie inductivement comme suit :

$$\begin{array}{rcll} \overline{Type}(\iota) & = & \iota & \text{pour } \iota \in BASE \\ \overline{Type}(s) & = & \overline{Type(s)} & \text{pour } s \in SPEC \\ \overline{Type}(\tau_1 \to \tau_2) & = & \overline{Type}(\tau_1) \to \overline{Type}(\tau_2) & \end{array}$$

3.8.2 Nouvelles règles de typage

Afin de déterminer les λ-termes qui ont un type $s \in SPEC$ donné, on a besoin d'un prédicat qui vérifie la bonne construction des termes. Pour chaque type $s \in SPEC$ on définit un prédicat BON_s :

Définition 3.8.1 (prédicat de construction)
Soit $s \in SPEC$, où $Type(s) = \iota_1 \times \cdots \times \iota_m \to \iota$. Un terme normalisé $u \in BON_s$ si et seulement si une des conditions suivantes est vraie :

1. *$u = x^\iota$ où, $x^\iota \in \mathcal{X}$, $\iota \in BASE$,*
2. *$u = c(u_1, \dots, u_n)$ pour $c \in symb(s)$, si $u_1, \dots, u_n \in BON_s$,*
3. *$u = f(u_1, \dots, u_n)(v_1, \dots, v_m)$ pour $f \in \mathcal{F}$, si $f : \tau_1 \times \cdots \times \tau_n \to s$ et $v_1, \dots, v_m \in BON_s$, ($u_1, \dots, u_n$ arbitraires)*
4. *$u = x(u_1, \dots, u_n)(v_1, \dots, v_m)$ pour $x^{\tau_1 \times \cdots \times \tau_n \to s} \in \mathcal{X}$, si $u_i \in BON_s$ pour tout i, où $\tau_i \in T_{\to BASE}$ et $v_j \in BON_s$*
5. *$u = \lambda x^\tau . u'$ si $u' \in BON_s$*

Remarquons que des termes de différents types peuvent appartenir à BON_s pour un type s donné. Maintenant on est prêt à présenter les règles de typage des termes dans la forme β-normale η-expansée (au sens de ?), prenant en compte les types fonctionnels à syntaxe limitée.

(fun-base) $$\frac{u_1 :_{\mapsto} \tau_1, \dots, u_n :_{\mapsto} \tau_n}{\phi(u_1, \dots, u_n) :_{\mapsto} \iota} \quad \text{si} \begin{cases} \phi \in \mathcal{F}_{\to BASE} \cup \mathcal{X}_{\to BASE}, \\ \phi : \tau_1 \times \cdots \times \tau_n \to \iota \end{cases}$$

(fun-spec) $$\frac{u_1 :_{\mapsto} \tau_1, \dots, u_n :_{\mapsto} \tau_n, v_1 :_{\mapsto} \iota_1, \dots, v_m :_{\mapsto} \iota_m}{\phi(u_1, \dots, u_n)(v_1, \dots, v_m) :_{\mapsto} \iota} \quad \text{si} \begin{cases} \phi \in \mathcal{F}_{\to SPEC} \cup \mathcal{X}_{\to SPEC}, \\ \phi : \tau_1 \times \cdots \times \tau_n \to s, \\ Type(s) = \iota_1 \times \cdots \times \iota_m \to \iota \end{cases}$$

(lambda) $$\frac{u :_{\mapsto} \sigma}{\lambda x^\tau . u :_{\mapsto} \tau \to u}$$

(spec) $$\frac{u :_{\mapsto} Type(s)}{u :_{\mapsto} xs} \quad \text{pour } s \in SPEC \text{ si } u \in BON_s$$

L'axiome $x^\tau : \tau$, qu'on peut trouver dans chaque système de typage, est ici caché dans la règle **(fun-base)**. Voici quelques exemples de termes typables par le nouveau système :

Exemple 5. Afin de montrer le fonctionnement du système de typage, on va enrichir l'exemple précédent de nouvelles fonctions définies, qui travaillent sur des types spéciaux. On a donc :

$$
\begin{array}{rcllcl}
BASE_5 & = & BASE_4 = \{\mathtt{nat}\} & & & \\
SPEC_5 & = & SPEC_4 = \{\mathtt{Lin}\} & & & \\
C_5 & = & \{0, s\} & A & : & \mathtt{nat} \to \mathtt{Lin} \\
D_5 & = & \{+, *, A, M\} & M & : & \mathtt{Lin} \to \mathtt{Lin}
\end{array}
$$

Les termes $u_1 = \lambda x^{\mathtt{nat}}.A(0)(x+0)$, $u_2 = \lambda y^{\mathtt{nat}}.M(\lambda x^{\mathtt{nat}}.A(s(0))(x))(y)$ et $u_3 = \lambda x^{\mathtt{nat}}.x^{\mathtt{nat}} + A(x * s(0))(x)$ ont tous le type Lin. Remarquons que le mauvais symbole $*$ apparaît dans u_3 dans le premier argument de la fonction A, mais selon la condition 3 de la définition du prédicat BON_s, on doit seulement assurer la bonne construction de la deuxième liste d'arguments.

Par contre les termes $v_1 = \lambda x^{\mathtt{nat}}.A(0)(x * x)$, $v_2 = \lambda x^{\mathtt{nat}}.A(0,0) * x$ et $v_3 = \lambda y^{\mathtt{nat}}.M(\lambda x^{\mathtt{nat}}.A(0)(x * y))(y)$ n'ont pas de type Lin. Les termes v_1 et v_2 ont uniquement le type $\mathtt{nat} \to \mathtt{nat}$, mais le terme v_3 n'est pas du tout typable par notre système.

3.8.3 Complétude

Dans cette section on va étudier la complétude de systèmes de réécriture qui peuvent utiliser le filtrage d'ordre supérieur, mais seulement dans les arguments de types fonctionnels à syntaxe limitée.

Définition 3.8.2 *Un système de réécriture $R = \{l_i \to r_i\}$ est appelé* limité *si toutes les règles sont de type de base et satisfont les conditions suivantes :*

- *si $l_i = f(u_1, \ldots, u_n)$ pour $f : \tau_1 \times \cdots \times \tau_n \to \tau$, alors tous les sous-termes u_j tels que $\tau_j \notin BASE \cup SPEC$ sont (η-égaux à) des variables linéaires*
- *si $l_i = f(u_1, \ldots, u_n)(v_1, \ldots, v_m)$ pour $f : \tau_1 \times \cdots \times \tau_n \to s$, $s \in SPEC$, alors les sous-termes $v_1, \ldots, v_m$ sont des variables linéaires.*

Il faut encore faire la remarque que les deux conditions ne s'excluent pas. Donc, si on prendre les types de l'exemple précédent, le membre gauche d'une règle qui définit la fonction $f : (\mathtt{nat} \to \mathtt{nat}) \times \mathtt{Lin} \times \mathtt{nat} \to \mathtt{Lin}$ doit être de la forme $f(\lambda x^{\mathtt{nat}}.Y(x), u, v)(z)$, pour Y et z libres. Par contre les termes u et v peuvent être plus détaillés.

La notion de complétude, qu'on va présenter ici pour des systèmes limités, est plus forte que celle pour des systèmes standard. Comme on l'a dit dans l'introduction, les nouveaux types fonctionnels ont beaucoup de points communs avec les types de base. Une similarité est le fait qu'on va demander, que les termes clos d'un type $s \in SPEC$ donné aient une forme construit uniquement à partir de symboles pré-définis $symb(s)$. On pourrait donc écrire un terme compliqué de type s, faire le calcul au niveau de la fonction, le simplifier jusqu'à la forme normale construit de $symb(s)$, et ensuite appliquer la forme simple à des arguments, peut-être plusieurs fois, sans recalculer la fonction de nouveau.

Un des inconvénients de cette approche est le fait que la propriété essentielle de préservation de types par réécriture, qui est normalement assurée par la simple condition que le membre gauche et le membre droit de chaque règle aient le même type, devient non-triviale dans le cadre de types fonctionnels à syntaxe limitée. De plus il y a une autre condition sur les termes qui doit être préservée par le système.

Définition 3.8.3 (condition VAR)
Un terme normalisé $t :_{\mapsto} \tau$ satisfait la condition VAR *si les conditions suivantes sont satisfaites :*

1. *si u est un sous-terme de t et $u :_{\mapsto} s$ pour $s \in SPEC$, alors $FV(u) \subseteq \mathcal{X}_{\to SPEC}$*
2. *si $f(u_1, \ldots, u_n)(v_1, \ldots, v_m)$ est un sous-terme de t, pour $f : \tau_1 \times \cdots \times \tau_n \to [\iota_1 \times \cdots \times \iota_m \to \iota]_s$, $s \in SPEC$, alors $FV(u_1, \ldots, u_n) \subseteq \mathcal{X}_{\to SPEC}$*

On donnera quelques outils de preuve de préservation de typabilité et la condition VAR par des règles de réécriture dans 3.8.5. Pour l'instant on peut présenter la nouvelle version de la notion de complétude.

Définition 3.8.4 (complétude globale)
Soit R un système de réécriture sur la signature contenant des types fonctionnels à syntaxe limitée. On dit que R est globalement complet *si pour tout terme clos t, qui satisfait la condition VAR,*

- *si $t :_{\mapsto} \iota$ pour $\iota \in BASE$ alors il existe $t' \in \mathcal{T}(C)$ tel que $t \longrightarrow_R^* t'$*
- *si $t :_{\mapsto} s$ pour $s \in BASE$ alors il existe $t' \in \mathcal{T}(symb(s))$ tel que $t \longrightarrow_R^* t'$*

La condition VAR sert en principe à éliminer les termes de nouveaux types fonctionnels qui sont de la mauvaise forme. Si le système de réécriture est limité, un terme clos $\lambda x^{\texttt{nat}}.A(x)(x)$ qui a le type `Lin` peut être réduit uniquement par une règle dont le membre gauche est de la forme $A(y)(z)$, qui filtre tout autre terme avec A en tête. Si on demande la réductibilité de ce terme, on perd la possibilité de distinction de cas.

Le changement de la notion de complétude globale nécessite la modification correspondante de la complétude locale. On commence par la nouvelle définition de f-terme canonique.

Définition 3.8.5 (f-terme canonique)
Soit $f : \tau_1 \times \cdots \times \tau_n \to \tau$ où $\tau \in BASE$ (respectivement $\tau \in SPEC$ et $Type(\tau) = \iota_1 \times \cdots \times \iota_m \to \iota$). Un terme $u = f(u_1, \ldots, u_n)$ (resp. $u = f(u_1, \ldots, u_n)(\perp_1, \ldots, \perp_m)$) va être appelé f-terme canonique *si pour tout $i = 1 \ldots n$, u_i est clos et l'une des conditions suivantes est satisfaite dans u :*

- *u_i a un type fonctionnel*
- *u_i a un type de base et $u_i \in \mathcal{T}(C, \mathcal{X})$*
- *u_i a un type $s \in SPEC$ et $u_i \in \mathcal{T}(symb(s), \mathcal{X})$*

(les symboles $\perp_1, \ldots, \perp_m$ sont des nouvelles constantes de types appropriés qui n'appartiennent pas à $\mathcal{F}$)

La *complétude locale* (voir la définition 3.4.3) de f est comme toujours la réductibilité de tout f-terme canonique clos.

Comme les systèmes de réécriture considérés ici sont des systèmes limités, la complétude locale d'un symbole de fonction f dont le type de sortie s appartient à $SPEC$ ($Type(s) = \iota_1 \times \cdots \times \iota_m \to \iota$), garantit aussi la réductibilité des termes

$\lambda x_1^{\iota_1} \dots x_m^{\iota_m}.f(u_1, \dots, u_n)(x_1, \dots, x_m)$ pour $u_1, \dots, u_n$ clos ainsi que de tous les termes $f(u_1, \dots, u_n)(v_1, \dots, v_m)$ pour $u_1, \dots, u_n$ clos et $v_1, \dots, v_m$ quelconques.

Afin de prouver l'équivalence entre complétude locale de tout les symboles et complétude globale du système, on a besoin d'un lemme encore plus fort, pour pouvoir faire la preuve par récurrence.

Lemme 3.8.6 *Soit R un système de réécriture limité convergent, qui préserve la typabilité au sens de $:_{\mapsto}$ et la condition VAR. Si R définit tous les symboles $f \in D$ d'une façon localement complète, alors pour un terme normalisé t qui satisfait VAR, les trois conditions suivantes sont satisfaites :*

1. *si $t :_{\mapsto} \iota$ pour $\iota \in BASE$ et t est clos, alors il existe $t' \in \mathcal{T}(C)$ tel que $t \longrightarrow_R^* t'$*
2. *si $t :_{\mapsto} s$ pour $s \in SPEC$ et t clos, alors il existe $t' \in \mathcal{T}(symb(s))$ tel que $t \longrightarrow_R^* t'$*
3. *si $t :_{\mapsto} \iota$ pour $\iota \in BASE$ et $t \in BON_s$ pour un $s \in SPEC$ donné tel que $Type(s) = \tau_1 \times \dots \times \tau_m \to \tau$ et si $FV(t) \subseteq \{x_1^{\tau_1}, \dots, x_m^{\tau_m}\}$, alors il existe $t' \in \mathcal{T}(symb(s), \mathcal{X})$ tel que $t \longrightarrow_R^* t'$*

Preuve. On prouve le lemme par récurrence sur l'union d'ordre de sous-terme et d'ordre de réduction (impliqué par la relation noetherienne $\longrightarrow_R$). Soit R et t comme dans le lemme.

1. si $t :_{\mapsto} \iota$ pour $\iota \in BASE$ et t est clos, alors il y a trois possibilités selon le symbole de tête de t (qui n'est pas une variable puisque t est clos) :
   - $t = c(t_1, \dots, t_n)$ pour $c \in C$. Les sous-terme $t_1, \dots, t_n$ de t sont clos du premier ordre et alors, par hypothèse de récurrence, il existe des termes $t'_1, \dots, t'_n \in \mathcal{T}(C)$ tels que pour tout i, $t_i \longrightarrow_R^* t'_i$. Dans ce cas $t \longrightarrow_R^* c(t'_1, \dots, t'_n)$.
   - $t = f(t_1, \dots, t_n)$ pour $f \in D$, $f : \tau_1 \times \dots \times \tau_n \to \iota$, $\iota \in BASE$.
   - $t = f(t_1, \dots, t_n)(u_1, \dots, u_m)$ pour $f \in D$, $f : \tau_1 \times \dots \times \tau_n \to [\iota_1 \times \dots \times \iota_m \to \iota]_s$, $s \in SPEC$.

   Dans les deux derniers cas on calcule par récurrence les formes normales t'_i pour les i où $\tau_i \in BASE \cup SPEC$, pour les autres on pose $t'_i = t_i$. Le terme $t' = f(t'_1, \dots, t'_n)$ (ou bien $t' = f(t'_1, \dots, t'_n)(u_1, \dots, u_m)$) est réductible. Disons que $t' \longrightarrow_R t''$. On a donc $t \longrightarrow_R^* t' \longrightarrow_R t''$ et par récurrence sur $\longrightarrow_R$ on peut trouver dans $\mathcal{T}(C)$ la forme normale de t''.
2. si $t :_{\mapsto} s$ pour $s \in SPEC$, $Type(s) = \iota_1 \times \dots \times \iota_m \to \iota$ et t clos, alors $t = \lambda x_1^{\iota_1} \dots x_m^{\iota_m}.t_1$, pour $t_1 :_{\mapsto} \iota$ et $t_1 \in BON_s$. Alors par récurrence (le point 3), $t_1 \longrightarrow_R^* t'_1$ pour $t'_1 \in \mathcal{T}(symb(s), \mathcal{X})$. Donc $t \longrightarrow_R^* \lambda x_1^{\iota_1} \dots x_m^{\iota_m}.t'_1$ où ce dernier terme est clos.
3. si $t :_{\mapsto} \iota$ pour $\iota \in BASE$ et $t \in BON_s$ pour un $s \in SPEC$ donné tel que $Type(s) = \tau_1 \times \dots \times \tau_m \to \tau$ et si $FV(t) \subseteq \{x_1^{\tau_1}, \dots, x_m^{\tau_m}\}$ alors on a les possibilités suivantes :
   - $t \in \mathcal{X}$. Donc forcément $t = x_i^{\tau_i}$ pour un i.
   - $t = c(t_1, \dots, t_n)$ pour $c \in symb(s)$. Les types de t_i sont dans $BASE$ et donc par récurrence, on a les formes souhaitées $t'_i \in \mathcal{T}(symb(s), \mathcal{X})$ et alors $t \longrightarrow_R^* c(t'_1, \dots t'_n)$

– $t = f(t_1, \ldots, t_n)(u_1, \ldots, u_m)$ pour $f : \tau_1 \times \cdots \times \tau_n \to s$. Comme t satisfait la condition *VAR*, les termes $t_1, \ldots, t_n$ sont clos. Par hypothèse de récurrence, on obtient les formes normales t'_i souhaitées pour les i où $\tau_i \in BASE \cup SPEC$. Pour les autres i on pose $t'_i = t_i$. On a donc $t = f(t'_1, \ldots, t'_n)(u_1, \ldots, u_m)$, où $t \longrightarrow^*_R t'$. Par complétude locale, t' se réduit à t''. Donc on peut utiliser l'hypothèse de récurrence pour t''.

Le terme t ne peut pas avoir un autre symbole en tête puisque $t \in BON_s$. □

On peut maintenant prouver immédiatement le théorème de complétude pour des systèmes utilisant les types fonctionnels à syntaxe limitée.

Théorème 3 *Soit R un système de réécriture limité convergent, qui préserve la typabilité au sens de $:_\mapsto$ et la condition VAR. Le système R est globalement complet si et seulement si R définit tous les symboles $f \in D$ d'une façon localement complète.*

Preuve. L'implication dans un sens est une simple conséquence du lemme précédent, et dans l'autre, comme toujours, est évidente. □

3.8.4 Vérification de complétude

Pour vérifier la complétude locale d'un symbole f donné, on peut utiliser la procédure de la page 16. Pour un symbole $f \in \mathcal{F}_{\to SPEC}$ on va utiliser le triplet initial suivant :

$$(\{f(X_1^{\tau_1}\uparrow^\eta, \ldots, X_n^{\tau_n}\uparrow^\eta)(\perp_1, \ldots, \perp_m)\}, \emptyset, \emptyset)$$

pour $f : \tau_1 \times \cdots \times \tau_n \to [\iota_1 \times \cdots \times \iota_m \to \iota]_s$. Il faut encore définir l'ensemble couvrant à utiliser pour des arguments d'un type $s \in SPEC$ donné, pour remplacer un sous-terme transparent $X(x_1^{\iota_1}, \ldots, x_n^{\iota_n})$ de type ι. Sa construction est très similaire à celle du grand ensemble couvrant, sauf que dans ce dernier on prenait en compte tous les symboles de la signature (de type de sortie approprié) et maintenant seulement les symboles autorisés de l'ensemble $symb(s)$.

Définition 3.8.7 `(ensemble couvrant limité)`
L'ensemble couvrant limité, $s\text{-}CS_{x_1^{\iota_1}, \ldots, x_n^{\iota_n}}(\iota)$, (pour $\iota_1, \ldots, \iota_n \in BASE$) contient tous les termes de la forme

$$f(X_1(x_1^{\iota_1}, \ldots, x_n^{\iota_n}), \ldots, X_m(x_1^{\iota_1}, \ldots, x_n^{\iota_n}))$$

où $f \in symb(s)$ et $f : \tau_1 \times \cdots \times \tau_m \to \iota$, $X_1, \ldots, X_n$ sont des nouvelles variables libres de type $X_i : \iota_1 \times \cdots \times \iota_n \to \tau_i$.

De plus, l'ensemble $s\text{-}CS_{x_1^{\iota_1}, \ldots, x_n^{\iota_n}}(\iota)$ contient toutes les variables liées parmi $x_1^{\iota_1}, \ldots, x_n^{\iota_n}$ pour lesquelles $\iota_i = \iota$.

La preuve de correction (vis-à-vis la définition 3.5.3) est très similaire à celle du grand ensemble couvrant :

Proposition 3.8.8 *Soit $f : \xi_1 \times \cdots \times \xi_n \to \xi$ où $\xi \in BASE$ (respectivement $\xi \in SPEC$ et $Type(\xi) = \iota_1 \times \cdots \times \iota_m \to \iota$). Soit $u = f(u_1, \ldots, u_k)$ (resp.*

$u = f(u_1, \ldots, u_k)(\perp_1, \ldots, \perp_m))$ un f-terme canonique et p une position dans u, telle que $p = i \cdot q$ pour $\xi_i \in SPEC$, et telle que $u|_p$ est un sous-terme transparent η-égale à $X(x_1^{\tau_1}, \ldots, x_n^{\tau_n})$, pour $X : \tau_1 \times \cdots \times \tau_n \to \tau$, $\tau_1, \ldots, \tau_n, \tau \in BASE$. L'ensemble $CS_{x_1^{\tau_1}, \ldots, x_n^{\tau_n}}(\iota)$ est un ensemble couvrant de $u|_p$.

Preuve. Soit σ une substitution canonique close de u. D'après le lemme 3.2.1, $X\sigma\updownarrow_\beta^\eta = \lambda y_1^{\tau_1} \ldots y_n^{\tau_n}.\phi(t_1, \ldots, t_n)$ pour $\phi \in \mathcal{F} \cup \mathcal{X}$. La substitution est close, donc il y a deux possibilité :

- ϕ est une variable liée, $\phi = y_j^{\tau_j}$ et dans ce cas $m = 0$, parce que $\tau_j \in BASE$, donc on prend $v = x_j$
- ϕ est un symbole de fonction, $\phi = g$ pour $g : \rho_1 \times \cdots \times \rho_m \to \iota$. Comme $u\sigma \updownarrow_\beta^\eta$ est un terme canonique, alors $g \in symb(s)$ et on peut prendre $v = g(X_1(x_1^{\tau_1}, \ldots, x_n^{\tau_n}), \ldots, X_m(x_1^{\tau_1}, \ldots, x_n^{\tau_n}))$

Dans le deuxième cas, on pose

$$\begin{array}{rcl} X_1\hat{\sigma} & = & \lambda y_1^{\tau_1} \ldots y_n^{\tau_n}.t_1 \\ & \vdots & \\ X_m\hat{\sigma} & = & \lambda y_1^{\tau_1} \ldots y_n^{\tau_n}.t_m \end{array}$$

Pour les $Y \in Dom(\sigma)$ tels que $Y \neq X$ on pose $Y\hat{\sigma} = Y\sigma$. Il est évident que $u\sigma \updownarrow_\beta^\eta = u[v]_p\hat{\sigma}\updownarrow_\beta^\eta$. □

Remarquons que dans le triplet initial, à la position d'un argument du type $s \in SPEC$ donné, il y a toujours un terme $\lambda x_1^{\iota_1} \ldots x_m^{\iota_m}.X(x_1, \ldots, x_m)$ pour $Type(s) = \iota_1 \times \cdots \times \iota_m \to \iota$. Alors le premier sous-terme transparent dans cette position est de la forme décrite dans la proposition (des types $\iota_1, \ldots, \iota_m \in BASE$ et comme la construction de $s - CS_{x_1^{\iota_1} \ldots x_m^{\iota_m}}$ ne le change pas, à chaque fois que l'on veut déclencher la règle **(décompose)** pour un sous-terme transparent d'un argument de type fonctionnel à syntaxe limitée, on peut utiliser un ensemble couvrant limité approprié.

La preuve de terminaison de la procédure de vérification peut être directement copiée du lemme 3.7.3, parce que l'utilisation des ensembles couvrants limités éloigne les variables libres de la racine de la même façon que l'utilisation des grands ensembles couvrants. Donc on peut se servir du même argument dans la preuve.

Comme pour l'approche précédente, il est aussi facile de trouver un exemple de fonction dont la définition est localement complète, mais pour laquelle la fonction retourne des f-termes canoniques irréductibles. Reprenons l'exemple 3 de la page 20. On introduit un type fonctionnel à syntaxe limité S par :

$$\begin{array}{rcl} SPEC_2 & = & \{S\} \\ Type(S) & = & \beta \to \beta \\ symb(S) & = & \{a, b, g\} \end{array}$$

et on change légèrement le type de la fonction $f : S \to \gamma$. Le test de la fonction f se déroule d'une façon identique et l'exemple irréductible retourné par la procédure est inductivement réductible pour la même raison.

Mais par contre la propriété de complétude de systèmes limités est à nouveau modulaire.

Lemme 3.8.9 *Soient R_1 et R_2 deux systèmes limités convergents, préservant la typabilité et la condition VAR, utilisant le même ensemble de types T et les signatures $\mathcal{F}_1$ et $\mathcal{F}_2$ respectivement, tels que $\mathcal{F}_1 \cap \mathcal{F}_2 = C_1 = C_2$ et que les spécifications de tous les types $s \in SPEC$ soient les mêmes. Si R_1 et R_2 sont complets et $R_1 \cup R_2$ termine et préserve typabilité et VAR, alors $R_1 \cup R_2$ est aussi complet.*

Preuve. Soit $R = R_1 \cup R_2$, $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$. On va montrer par contradiction que chaque symbole $f \in \mathcal{F}$ est complètement défini par R. Supposons qu'il y ait un f-terme canonique $u \in \mathcal{T}(\mathcal{F})$ clos irréductible, pour $f \in D_1$, $f : \tau_1 \times \cdots \times \tau_n \to \tau$. Si $\tau \in BASE$, alors $u = f(u_1, \ldots, u_n)$ et si $\tau \in SPEC$, alors $u = f(u_1, \ldots, u_n)(\bot_1, \ldots, \bot_m)$. On échange maintenant tous les arguments u_i de types $\tau_i \notin BASE \cup SPEC$ pour des termes $u'_i \in \mathcal{T}(\mathcal{F}_\infty)$ (de types appropriés). Soit u' le terme après le changement. Le système R_1 est complet, u est réductible par une règle $l_j \to r_j$. Mais comme R_1 est un système limité, $l_j = f(t_1, \ldots, t_n)$ (où $l_j = f(t_1, \ldots, t_n)(x_1, \ldots, x_m)$ pour $f \in \mathcal{F}_{\to SPEC}$) pour $t_i = Y{\uparrow^\eta}$ si $\tau_i \notin BASE \cup SPEC$. Or si l_j filtre u', il filtre également le terme u. Contradiction. $\square$

La preuve était essentiellement identique à celle de la modularité de la complétude des systèmes classiques (lemme 3.6.3).

3.8.5 Conditions sur des règles de réécriture

On va donner quelque conseils pour prouver qu'un système de règles donné préserve la typabilité au sens de $:_{\mapsto}$ et la condition VAR. Pour préserver la typabilité, il faut absolument que le système préserve le prédicat BON_s pour tout $s \in SPEC$. Pour prouver la préservation de typabilité, voici quelque remarques :

- pour des règles de premier ordre, si les symboles du membre gauche appartiennent à $symb(s)$ il faut que les symboles du membre droit y appartiennent aussi.
- pour des règles d'ordre supérieur, si le membre gauche commence par un symbole d'ordre supérieur $f \in \mathcal{F}_{\to BASE}$, le membre gauche ne satisfait de prédicat BON_s pour aucun s.
- pour des règles d'ordre supérieur, si le membre gauche commence par un symbole $f \in \mathcal{F}_{\to SPEC}$, avec le type de sortie $s \in SPEC$, le membre gauche peut éventuellement satisfaire le prédicat BON_s, donc il faut vérifier cela dans le membre droit. Un tel membre droit ne satisfait pas de prédicat $BON_{s'}$ pour les autres types $s' \in SPEC$.

Le lemme suivant pourrait aider aussi à prouver la préservation de ces propriétés par le système de réécriture.

Lemme 3.8.10 *Soit t un terme normalisé typable, qui satisfait VAR et σ une substitution normalisée typable qui satisfait VAR. Le terme $t\sigma\updownarrow_\beta^\eta$ satisfait VAR. De plus si $t \in BON_s$, pour un $s \in SPEC$ donné, et pour tout $x \in \mathcal{X}_{\to BASE}$, tel que $x \in Dom(\sigma)$, $x\sigma \in BON_s$, alors $t\sigma\updownarrow_\beta^\eta \in BON_s$.*

Preuve. Puisque t et σ sont typables, ils sont également typables au sens de $\bar{:}$ (voir la page 22) et donc $t\sigma$ est un terme typable du λ-calcul simplement typé, alors il est fortement normalisant. On prouve par récurrence sur le nombre de β-réductions nécessaires pour obtenir $t\sigma\updownarrow_{\beta}^{\eta}$ à partir de $t\sigma$. On fait également la récurrence interne sur la construction de t.

- $t = x$, c'est trivial
- $t = \lambda x.t'$. Comme t satisfait VAR et BON_s, alors t' aussi, donc on prouve le résultat par récurrence.
- $t = f(t_1, \ldots, t_n)$, f - de premier ordre. Les sous-termes satisfont VAR et BON_s. Si $f \in Fbase$ d'ordre supérieur, $t \notin BON_s$, mais les sous-termes satisfont VAR. On obtient donc le résultat par récurrence.
- $t = f(t_1, \ldots, t_n)(u_1, \ldots, u_m)$ pour $f \in \mathcal{F}_{\rightarrow SPEC}$. Les sous-termes $t_1, \ldots, t_n$ n'ont pas de variables dans $\mathcal{X}_{\rightarrow BASE}$ et les termes substitués par σ non plus. Les $u_1, \ldots, u_m$ satisfont VAR par récurrence. Si le type de sortie de f est s alors on a $u_1, \ldots, u_m \in BON_s$ et on conclut par récurrence. Si le type de sortie de f est un autre type dans $SPEC$, alors $t \notin BON_s$.
- $t = X(t_1, \ldots, t_n)$. Si $X \in \mathcal{X}_{\rightarrow BASE}$ alors $t \notin BON_s$. Si $X \in \mathcal{X}_{\rightarrow SPEC}$, et type de sortie de X est s, tous les termes t_i de type $T_{\rightarrow BASE}$ sont BON_s. Par hypothèse de récurrence, $t_i\sigma\updownarrow_{\beta}^{\eta}$ sont BON_s aussi. Maintenant on peut appliquer la récurrence sur le nombre de β-réductions. $X\sigma = \lambda x_1 \ldots x_n.t'$, on peut construire un terme normalisé t' et une substitution normalisée $\sigma' = \{x_i \mapsto t_i\sigma\updownarrow_{\beta}^{\eta}\}$. Comme $t'\sigma'$ est obtenu de $t\sigma$ par n β-réductions nécessaires, le nombre de β-réductions pour normaliser $t'\sigma'$ et plus petit que celui pour normaliser $t\sigma$, on peut utiliser la récurrence. $\square$

L'étude de systèmes avec types fonctionnels à syntaxe limitée est donc très compliquée. Cette approche constitue un intermédiaire entre l'approche classique et l'approche sans restrictions. On essaie de joindre les avantages de ces deux approches, mais les difficultés qui apparaissent sont très grandes. On doit mettre plusieurs conditions non-triviales sur les règles de réécriture, le typage est non-standard et peu clair, les résultats de complétude s'appliquent uniquement à des certains termes. On voit donc que cette approche est trop complexe pour être utilisée en pratique.

4 Ordre supérieur : types polymorphes

Cette section est consacrée à l'étude de la problématique de complétude de systèmes de réécriture d'ordre supérieur dans le cadre de types polymorphes. Les types polymorphes sont dans ce travail équipés d'opérateurs algébriques, comme dans l'article [8], par exemple l'opérateur `list`$(\cdot)$, qui pour un type quelconque τ donne le type `list`(τ) de listes d'éléments de type τ.

Même si ces opérateurs, et leurs constructeurs appropriés, peuvent être représentés directement par des types et des termes du système F pur (sans constantes de types et de termes), leur codage est très compliqué et peu intuitif. Par exemple l'ouvrage [5] propose la traduction suivante de l'opérateur `list`$(\cdot)$ et de ses constructeur Nil_τ : `list`(τ) (liste vide) et $Cons_\tau : \tau \times \texttt{list}(\tau) \to \texttt{list}(\tau)$ (tête) :

$$\begin{aligned}
\texttt{list}(\tau) &= \forall\alpha.\alpha \to (\tau \to \alpha \to \alpha) \to \alpha \\
Nil_\tau &= \Lambda\alpha.\lambda x^\alpha.\lambda y^{\tau\to\alpha\to\alpha}.x \\
Cons_\tau(h,t) &= \Lambda\alpha.\lambda x^\alpha.\lambda y^{\tau\to\alpha\to\alpha}.y(h, t\alpha(x,y))
\end{aligned}$$

4.1 Types polymorphes

Étant donné un ensemble dénombrable de variables de types Φ, ainsi qu'un ensemble fini d'*opérateurs de sortes* $S = \bigcup_{n\geq 0} S_n$, on définit l'ensemble de types $T_\forall$, comme suit :

$$T_\forall ::= \Phi \mid S_n(T_\forall^n) \mid T_\forall \to T_\forall \mid \forall\Phi.T_\forall$$

L'ensemble d'opérateurs de sortes d'arité 0, S_0 prend la place de l'ensemble de sortes B de la section précédente. Dans le cadre de types polymorphes, les types de base sont tous les types construits uniquement à partir d'opérateurs de sortes. On va aussi écrire qu'un type τ est de premier ordre, $\tau \in T(S,\Phi)$, s'il est construit à partir d'opérateurs de sortes et de variables de types Φ. La convention sur l'utilisation du symbole $\to$, reste la même que dans la section précédente.

Le nouveau symbole $\forall$ lie des variables de type de la même façon que l'opérateur λ lie des variables de terme dans les termes du λ-calcul, on peut donc définir analogiquement la notion de variable de type libre et liée ainsi que la substitution de type. Le quantificateur signifie intuitivement qu'un terme qui a le type $\forall\alpha.\tau$ a tous les types $\tau\{\alpha \mapsto \rho\}$ pour tous les ρ possible.

4.2 Termes du λ-calcul polymorphe

On va utiliser un ensemble dénombrable de variables $\mathcal{X}$ qui est, comme avant, une somme disjointe de variables de chaque type. L'ensemble de toutes les variables dont le type commence par un opérateur de sortes $s \in S$, va être noté par le symbole $\mathcal{X}_S$. On va avoir également un ensemble fini de symboles de fonction

$$\mathcal{F} = \bigcup_{\tau_1,\ldots,\tau_n,\tau} \mathcal{F}_{\tau_1\times\cdots\times\tau_n\to\tau}$$

où l'ensemble $\mathcal{F}_{\tau_1\times\cdots\times\tau_n\to\tau}$ contient tout les symboles algébriques dont les types des arguments sont $\tau_1,\ldots,\tau_n$ et le type de sortie τ. Remarquons que dans la section précédente le type de sortie de chaque fonction était le type de base. Maintenant le type de sortie peut être arbitraire. Comme avant, $\mathcal{F} = C \cup D$ où C est un ensemble de constructeurs. Pour chaque opérateur de sorte $s \in S_n$ on a un nombre fini de constructeurs

$$c : \tau_1 \times \cdots \times \tau_m \to s(\alpha_1,\ldots,\alpha_n) \quad \text{pour} \quad \tau_1,\ldots,\tau_m \in T(S,\{\alpha_1,\ldots,\alpha_n\})$$

À partir de $\mathcal{X}$ et $\mathcal{F}$ on va construire l'ensemble des termes algébriques du λ-calcul polymorphe. En outre d'opérateurs d'abstraction et d'application les termes vont aussi utiliser l'opérateur d'abstraction de type et d'application de type.

$$\mathcal{T} ::= \mathcal{X} \mid \lambda\mathcal{X}.\mathcal{T} \mid \mathcal{T}(\mathcal{T}) \mid \mathcal{F}(\mathcal{T},\ldots,\mathcal{T}) \mid \Lambda\Phi.\mathcal{T} \mid \mathcal{T}T_\forall$$

L'ensemble de variables de type libre dans un terme t donné va être noté $FVT(t)$. L'ensemble de termes typables, noté toujours $\mathcal{T}(\mathcal{F},\mathcal{X})$, est calculé en utilisant les règles de typage suivantes :

$$\textbf{(var)} \qquad \frac{}{x^\tau : \tau}$$

$$\textbf{(fun)} \qquad \frac{u_1 : \tau_1,\ldots,u_n : \tau_n}{f(u_1,\ldots,u_n) : \tau} \quad \text{pour} \quad \begin{array}{l} f \in \mathcal{F} \\ f : \tau_1 \times \cdots \times \tau_n \to \tau \end{array}$$

$$\textbf{(abs-terme)} \qquad \frac{u : \rho}{\lambda x^\tau.u : \tau \to \rho}$$

$$\textbf{(app-terme)} \qquad \frac{u : \tau \to \rho \qquad v : \tau}{u(v) : \rho}$$

$$\textbf{(abs-type)} \qquad \frac{u : \rho}{\Lambda\alpha.u : \forall\alpha.\rho} \quad \text{si } \alpha \text{ n'est pas libre dans les types de } FV(u)$$

$$\textbf{(app-type)} \qquad \frac{u : \forall\alpha.\rho}{u\tau : \rho\{\alpha \mapsto \tau\}}$$

On va supposer que pour tout type de premier ordre il y a des termes constructeurs de ce type et l'ensemble de termes constructeurs typables clos va être noté $\mathcal{T}(C)$.

La sémantique du λ calcul polymorphe, en plus de la règle β (voir 3.2) est donné aussi par la règle $T\beta$ suivante :

$$(\Lambda\alpha.u)\tau \quad \longrightarrow_{T\beta} \quad u\{\alpha \mapsto \tau\}$$

4.3 Réécriture

Comme on l'a dit dans l'introduction, dans cette section on ne va plus considérer la réécriture de termes normalisés, mais la version ou les réductions par des règles du système et les β-réductions peuvent s'entrelacer arbitrairement. Étant donné un système de règles $R = \{l_i \to r_i\}$ qui satisfont les conditions de la section 2.2, un terme $u \in \mathcal{T}(\mathcal{F},\mathcal{X})$ se réécrit à un terme t à la position p

en utilisant la règle $l_i \to r_i$ ce qui va être noté $s \xrightarrow[l \to r]{p} t$ (où tout simplement $s \longrightarrow t$), s'il existe une substitution de type σ_T et une substitution σ telle que $s|_p = l\sigma_T\sigma$ et $t = s[r\sigma_T\sigma]_p$.

On va se restreindre par contre aux systèmes algébriques, où tous les membres gauches sont de la forme $f(u_1, \ldots, u_n)$ pour $f \in D$.

On va dire que R est convergent si la relation $\longrightarrow_{R\beta} = \longrightarrow_R \cup \longrightarrow_\beta \cup \longrightarrow_{T\beta}$ est convergente. Les propriétés de tels systèmes mixtes sont étudiées entre autres dans l'article [8].

4.4 Complétude

La notion de complétude de systèmes de réécriture dans le cadre de types polymorphes ne diffère pas beaucoup de celle de la section précédente :

Définition 4.4.1 `(complétude globale)`
Un système R convergent est appelé globalement complet *si pour tout terme t clos, dont le type est clos de premier ordre, il existe un terme $s \in \mathcal{T}(C)$ tel que $t \longrightarrow_{R\beta} s$.*

La notion de complétude locale nécessite une modification dans le cadre de types polymorphes. La fonction localement complète doit être définie pour les arguments clos qui sont construits à partir de constructeurs jusqu'aux sous-termes qui ne sont pas de premier ordre.

Définition 4.4.2 `(complétude locale)`
Un système R définit une fonction $f \in D$ pour $f : \tau_1 \times \cdots \times \tau_n \to \tau$ d'une façon localement complète *si tout terme $f(t_1, \ldots, t_n)$ est réductible pour une substitution de types σ_T et si pour tout i, $t_i : \tau_i\sigma_T$, $t_i = t'_i\gamma_i$ pour une substitution close γ_i et $t'_i \in \mathcal{T}(C, \mathcal{X})$ et $FV(t'_i) \cap \mathcal{X}_S = \emptyset$.*

Afin de démontrer le théorème de complétude, on doit prouver la proposition suivante :

Proposition 4.4.3 *Soit t un terme typable clos. Si $t = t_1(t_2)$ ou bien $t = t_1\tau$, alors t contient un β-redex, un $T\beta$-redex ou un sous-terme clos de la forme $f(u_1, \ldots, u_n)$ pour $f \in D$*

Preuve. Par récurrence sur la construction de t_1, qui ne peut pas commencer par un constructeur ni par une variable.

- Si $t = t_1(t_2)$ alors $t_1 : \tau_1 \to \tau_2$. Donc on a quatre possibilités :
  1. $t_1 = f(u_1, \ldots, u_n)$ pour $f \in D$,
  2. $t_1 = \lambda x.t'_1$ donc t est un redex,
  3. $t_1 = t_3(t_4)$ pour $t_3 : \tau_4 \to (\tau_1 \to \tau_2)$, par hypothèse de récurrence on trouve un sous-terme voulu dans t_1,
  4. $t_1 = t_3\rho$ pour $t_3 : \forall\alpha.\tau'_1 \to \tau'_2$, par hypothèse de récurrence, t_1 contient un sous-terme cherché.
- Si $t = t_1\tau$ alors $t_1 : \forall\alpha.\tau'$ et on a les possibilités suivantes :
  1. $t_1 = f(u_1, \ldots, u_n)$ pour $f \in D$,

2. $t_1 = \Lambda\alpha.t'$, donc t est un redex,
3. $t_1 = t_3(t_4)$ pour $t_3 : \tau_4 \to \forall\alpha.\tau'$, par hypothèse de récurrence, on trouve un sous-terme voulu dans t_1,
4. $t_1 = t_3\rho$ pour $t_3 : \forall\beta.\tau_3'$ — par hypothèse de récurrence, t_1 contient un sous-terme expecté. □

Maintenant on va prouver un lemme qui implique directement le théorème de complétude. Mais avant de faire cela on a besoin de quelques notions auxiliaires.

Définition 4.4.4 *Soit τ un type polymorphe. Il existe un type de premier ordre ρ et une substitution de types σ_T tels que $\tau = \rho\sigma_T$, $\rho \in T(S, \Phi)$ et pour toute $\alpha \in FVT(\rho)$, $\alpha\sigma_T \neq s(\xi_1, \ldots, \xi_n)$ pour $s \in S$. Alors le* chapeau *de τ, noté $cap(\tau)$, est égal à ρ et l'ensemble de ses* aliens *$aliens(\tau) = \{\alpha\sigma_T \mid \alpha \in FVT(\tau)\}$*

On a donc, intuitivement introduit une partition d'un type τ donné en un chapeau d'opérateurs de sortes et le reste — l'ensemble des aliens — les sous-termes (de type) qui ne commence pas par un opérateur de sortes. Pour un type de premier ordre clos $\rho \in T(S)$, $cap(\rho) = \rho$ et $aliens(\rho) = \emptyset$.

Le lemme qui suit montre que dans une partition pareille de chaque terme clos $t : \tau$ en forme normale — en chapeau maximale de constructeurs et les autres sous-termes — les types de ces derniers appartiennent dans $aliens(\tau)$. Pour faciliter la formulation, on va écrire $Types(A)$ pour dénoter l'ensemble des types des termes d'un ensemble A donné.

Lemme 4.4.5 *Soit R un système de réécriture, qui définit tout ses symboles d'une façon localement complète. Soit t un terme clos de type τ. Si t est une forme normale, alors il existe un terme t' et une substitution γ tels que $t = t'\gamma$ pour $t' \in \mathcal{T}(C, \mathcal{X})$, $Types(FV(t')) \subseteq aliens(\tau)$.*

Preuve. Par récurrence sur la construction de t :

- Si τ ne commence pas par un opérateur de sorte, alors $cap(\tau) = \alpha$, $aliens(\tau) = \{\tau\}$, et alors $t = x^\tau\{x \mapsto t\}$.
- Si τ commence par un opérateur de sorte, alors $\tau = s(\overline{\tau})$, pour $s \in S$.

  Alors t ne contient pas de sous-terme clos $f(t_1, \ldots, t_m)$ pour $f \in D$, parce que vue la condition de complétude locale de f et le fait que $t_1, \ldots, t_m$ sont dans la forme normale, chaque sous-terme de cette forme serait réductible.

  Si $t = t_1(t_2)$ ou $t = t_1\tau'$, alors par la proposition 4.4.3 et la dernière remarque, t contient un redex, ce qui est impossible.

  Donc forcément $t = c(t_1, \ldots, t_m)$ pour $c \in C$, $c : \rho_1(\overline{\alpha}) \times \cdots \times \rho_m(\overline{\alpha}) \to s(\overline{\alpha})$. Les sous-termes t_i ont des types $\rho_i(\overline{\tau})$ et comme $\rho_i \in T(S, \Phi)$, alors $aliens(\rho_i(\overline{\tau})) \subseteq aliens(s(\overline{\tau})) = aliens(\tau)$. Donc puisque les t_i sont en la forme normale, on a $t_i = t_i'\gamma_i$ pour $Types(FV(t_i')) \subseteq aliens(\rho_i(\overline{\tau}))$. Soit maintenant $t' = c(t_1', \ldots, t_m')$, $\gamma = \gamma_1 \cup \ldots \cup \gamma_n$ (on renomme les variables libres dans t_i' en cas de conflit), et $t = t'\gamma$ pour $Types(FV(t')) \subseteq aliens(\tau)$, donc on obtient le résultat. □

On prouve maintenant facilement le théorème de complétude de systèmes convergents.

Théorème 4 `(théorème de complétude)`
Soit R un système de réécriture d'ordre supérieur convergent. R définit tous les symboles $f \in D$ d'une façon localement complète si et seulement si R est globalement complet.

Preuve. La preuve dans un sens est, comme toujours, évidente, et dans l'autre sens est une simple conséquence du lemme précédent, puisque pour un terme t de type de base τ, $aliens(\tau) = \emptyset$ et alors $t = t'$ et la substitution γ est une fonction vide. Donc comme chaque terme possède une forme normale, et comme cette forme normale est un terme constructeur, on prouve le théorème. □

4.5 Vérification de complétude

Pour les types polymorphes on va étudier seulement l'approche classique sur les arguments fonctionnels. Le deuxième type d'arguments dont on ne trouve pas dans la discipline de types simples, les arguments de types universellement quantifiés, vont être traités de la même manière.

Définition 4.5.1 `(système classique)`
Le système de réécriture algébrique $R = \{l_i \to r_i\}$ de termes polymorphes est classique *si les membres gauches l_i sont de la forme $f(t_1, \ldots, t_n)$ pour $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \tau$, les termes $t_i : \tau_i$ et si $s : \rho$ est un sous-terme d'un t_i (pas nécessairement propre) tel que $\rho \neq s(\overline{\rho})$ pour aucun $s \in S$, alors s est une variable linéaire.*

À cause de cette restriction sur le système de règles, on peut utiliser la définition 2.4.1 de la variable d'induction présenté dans la section consacrée au premier ordre. L'unification qui apparaît dans cette définition doit prendre en compte aussi les types qui doivent s'unifier.

Puisque les sous-termes des membres gauches d'autres types sont des variable linéaires, seules les variables dont les types commencent par un symbole $s \in S$ peuvent être des variables d'induction. Il suffit donc la définition de l'ensemble couvrant suivant

Définition 4.5.2 `(ensemble couvrant)`
Soit $t = f(u_1, \ldots, u_n)$ pour $f \in D$, $f : \tau_1 \times \cdots \times \tau_n \to \tau$ et $u_1, \ldots, u_n \in \mathcal{T}(C, \mathcal{X})$, $u_i : \tau_i$ Soit p une position de variable d'induction x^τ dans t. Un ensemble fini de termes non-variables CS est un ensemble couvrant *de x^τ dans t ssi pour toute substitution σ telle que $u_i\sigma \in \mathcal{T}(C, \mathcal{X})$, $u_i\sigma : \tau_i$ et $FV(u_i\sigma) \cap \mathcal{X}_S = \emptyset$, il existe un terme $v \in CS$ et une substitution $\hat{\sigma}$ tels que $t\sigma = t[v]_p\hat{\sigma}$.*

Dans la procédure de vérification on va utiliser l'ensemble couvrant qui est une légère généralisation du petit ensemble couvrant. Pour une variable d'induction de type $s(\overline{\tau})$, on définit

Proposition 4.5.3 `(ensemble couvrant polymorphe)`
Un ensemble couvrant polymorphe *$CS_\forall(s(\overline{\tau}))$ contient tous les termes de la forme*

$$c(x_1, \ldots, x_m) \quad pour \quad c : \rho_1(\overline{\alpha}) \times \cdots \times \rho_m(\overline{\alpha}) \to s(\overline{\alpha})$$

ou les variables $x_1, \ldots, x_m$ sont des nouvelles variables de types $x_i : \rho_i(\overline{\tau})$. $CS_\forall(s(\overline{\tau}))$ est un ensemble couvrant d'une variable d'induction $x : s(\overline{\tau})$.

Preuve. Soient t et σ comme dans la définition de l'ensemble couvrant et p une position dans t d'une variable d'induction $x : s(\overline{\tau})$. Le terme $t\sigma$ doit être de la forme $c(t_1, \ldots, t_m)$ pour c constructeur de type de sortie $s(\overline{\tau})$, qui est élément de $CS_\forall(s(\overline{\tau}))$. On pose $x_i\hat{\sigma} = t_i$ et pour les autres variables $y\hat{\sigma} = y\sigma$. On a certainement $t\sigma = t[c(x_1, \ldots, x_m]_p\hat{\sigma}$. $\square$

La preuve de correction de la réponse positive de la procédure nécessite une proposition supplémentaire, qui dit qu'un terme de type $\tau = \rho\sigma_T$ qui a des constructeurs jusqu'aux sous-termes qui ne sont pas du premier ordre est une instance de type et de terme d'un terme de type ρ qui, lui aussi, a des constructeurs le plus loin possible.

Proposition 4.5.4 *Soit t un terme de type $\tau = \rho\sigma_T$ pour une substitution de type σ_T. Si $t = t'\sigma$ pour une substitution σ, $t' \in \mathcal{T}(C, \mathcal{X})$ avec $FV(t') \cap \mathcal{X}_S = \emptyset$, alors il existe un terme $t'' : \rho$ tel que $t'' \in \mathcal{T}(C, \mathcal{X})$, $FV(t'') \cap \mathcal{X}_S = \emptyset$ et une substitution γ tels que $t = t''\sigma_T\gamma$.*

Preuve. On procède par récurrence sur la construction de t. Si $\rho = \alpha$ est une variable de type ou bien $\rho = \rho_1 \to \rho_2$ ou bien $\rho = \forall\alpha.\rho'$ alors on pose $t'' = x^\rho$, $\gamma = \{x^\tau \mapsto t\}$, et on obtient $t''\sigma_T\gamma = x^\rho\gamma = t$, $\{x^\rho\} \cap \mathcal{X}_S = \emptyset$, pour $t'' \in \mathcal{T}(C, \mathcal{X})$.

Si $\rho = s(\overline{\xi})$, pour $s \in S$, alors $\tau = s(\overline{\xi\sigma_T})$, et $t = c(t_1, \ldots, t_m)$ pour un constructeur $c : \iota_1(\overline{\alpha}) \times \cdots \times \iota_m(\overline{\alpha}) \to s(\overline{\alpha})$. Certainement $t_i : \iota_i(\overline{\xi})\sigma_T$, $t_i = t'_i\sigma_i$ pour $t'_i \in \mathcal{T}(C, \mathcal{X})$, $FV(t'_i) \cap \mathcal{X}_S = \emptyset$.

Donc par hypothèse de récurrence $t_i = t''_i\sigma_T\gamma_i$ pour $t''_i \in \mathcal{T}(C, \mathcal{X})$, $t''_i : \iota_i(\overline{\xi})$, $FV(t''_i) \cap \mathcal{X}_S = \emptyset$. Si on prend $t'' = c(t''_1, \ldots, t''_m)$, $\gamma = \gamma_1 \cup \ldots \cup \gamma_m$ (en cas de conflit on change des noms de variables dans t''_i), on obtient $t'' : s(\overline{\xi}) = \rho$, $t'' \in \mathcal{T}(C, \mathcal{X})$, $FV(t'') \cap \mathcal{X}_S = \emptyset$ et $t''\sigma_T\gamma = c(t''_1, \ldots, t''_m)\sigma_T\gamma = c(t''_1\sigma_T\gamma, \ldots, t''_m\sigma_T\gamma) = t$. $\square$

On peut maintenant prouver la correction de la procédure de vérification.

Lemme 4.5.5 *Soit R un système classique. Si la procédure, lancée pour un symbole $f \in D$ donné, $f : \tau_1 \times \cdots \times \tau_n \to \tau$ retourne un* **succès**, *alors R définit f d'une façon localement complète.*

Preuve. D'après la définition de l'ensemble couvrant et la réductibilité de toutes les feuilles par la procédure, tout terme de la forme $f(t_1, \ldots t_n)$ pour $t_i : \tau_i$, $t_i \in \mathcal{T}(C, \mathcal{X})$, $FV(t_i) \cap \mathcal{X}_S$ est réductible par R.

Soit $f(u_1, \ldots, u_n)$ une instance close de $f(x_1, \ldots, x_m)$ telle que pour tout i, $u_i : \tau_i\sigma_T$ pour une substitution de types σ_T, $u_i = u'_i\sigma_i$ pour $u'_i \in \mathcal{T}(C, \mathcal{X})$, $FV(u'_i) \cap \mathcal{X}_S = \emptyset$, comme dans la définition de la complétude locale.

Selon le lemme précédent pour chaque i, il existe un terme u''_i tel que $u''_i \in \mathcal{T}(C, \mathcal{X})$, $FV(u''_i) \cap \mathcal{X}_S = \emptyset$ et une substitution γ_i tels que $u_i = u''_i\sigma_T\gamma_i$. Soit $\gamma = \gamma_1 \cup \ldots \cup \gamma_n$ (au cas de conflit on change des noms de variables dans u''_i). Alors le terme $f(u_1, \ldots, u_n)$ est une instance du terme $f(u''_1, \ldots, u''_n)$ avec les substitutions $\sigma_T\gamma$, donc c'est une instance d'un terme réductible. $\square$

La preuve de terminaison de la procédure de vérification peut être faite de la même façon que d'habitude, puisque chaque utilisation de l'ensemble couvrant polymorphe par la règle **(décompose)**, remplace un terme de l'ensemble de nœuds par des autres où il y a un constructeur ajouté dans chaque terme. On a donc, grâce au théorème 4, un outil de preuve de complétude de systèmes de réécriture classiques.

5 Conclusions

On a présenté dans ce travail une adaptation de la notion de complétude de systèmes de réécriture à des systèmes d'ordre supérieur. Les systèmes considérés utilisent une signature divisée en deux parties disjointes : l'ensemble de constructeurs libres du premier ordre et l'ensemble des fonctions définies. On décrit une procédure qui, pour un symbole de fonction donné, vérifie si sa définition par des règles de réécriture couvre l'ensemble de tous ses arguments possibles. On démontre que lorsque cette dernière propriété est vraie pour toutes les fonctions du système, le système lui même est complet.

Deux disciplines de types sont étudiées dans ce travail : les types simples et les types polymorphes. La problématique de complétude de la réécriture typée par d'autres système de typage, par exemple les types dépendants ou le calcul de construction, nécessite un travail supplémentaire.

Pour les types simples, on présente trois approches de traitement des arguments fonctionnels. L'approche non-standard, étudiée dans la section 3.8, constitue un essai de permettre dans les définitions de fonctions l'utilisation du filtrage d'ordre supérieur pour les arguments de certains types fonctionnels, tout en préservant la modularité de la complétude du système. Ces types fonctionnels sont décrits d'une façon syntaxique ce qui provoque de multiples complications. On peut espérer que dans une discipline de types plus riche, on puisse trouver un moyen sémantique de décrire les types fonctionnels, pour lesquels l'utilisation du filtrage d'ordre supérieur n'empêcherait pas la modularité de la complétude.

Une autre extension possible de ce travail consiste à considérer les systèmes où les constructeurs ne sont pas libres, comme par exemple dans la spécification des entiers relatifs, $pred(succ(x)) \to x$. Les notions de complétude présentées dans ce mémoire peuvent être facilement généralisées pour ce type de systèmes. Par contre la généralisation devient plus compliquée si on enlève la condition que les constructeurs soient du premier ordre. Un exemple pratique où les constructeurs ont les types d'ordre supérieur est le système qui décrit les formules logiques de premier ordre, avec des constructeurs $\forall, \exists : (\texttt{terme} \to \texttt{formule}) \to \texttt{formule}$, où la formule $\forall x.\Phi(x)$ est représentée par le terme $\forall(\lambda x.\Phi(x))$.

La technique utilisée pour tester la complétude de définitions de fonctions est celle des arbres de motifs, adaptée de [1, 2]. Une généralisation très intéressante, du point de vue de l'application de la réécriture comme moyen de spécifier des systèmes informatiques, serait de considérer les règles conditionnelles d'ordre supérieur. On pourrait également considérer l'utilisation d'autres moyens de preuve de réductibilité close, par exemple la technique d'automates d'arbres, ou la technique de la résolution de formules quantifiées sur le λ-termes étudiés par Lugiez [12] dans le cadre de types simples.

Références

[1] Adel Bouhoula. Using Induction and Rewriting to Verify and Complete Parameterized Specifications. *Theoretical Computer Science*, 170, 1996. To appear.

[2] Adel Bouhoula and Jean-Pierre Jouannaud. Automata-driven automated induction.
Technical report, SRI-CSL, 1996. http ://www.loria.fr/ bouhoula.html, //www.lri.fr/people/jouannau.html.

[3] Jaco Van de Pol. *Termination of Higher-Order Rewrite Systems.* Phd thesis, Utrecht University, Utrecht, Netherlands, 1996.

[4] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[5] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[6] Warren D. Goldfarb. Note on the undecidability of the second-order unification problem. In *Theoretical Computer Science*, pages 225–230, 1981.

[7] John V. Guttag and James J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10(1), 1978.

[8] Jean-Pierre Jouannaud and Mitsuhiro Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2) :349–391, February 1997.

[9] Jean-Pierre Jouannaud and Albert Rubio. A recursive path ordering for higher-order terms in η-long β-normal form. In Ganzinger H., editor, *Proc. 7th Rewriting Techniques and Applications, New-Jersey, LNCS 1103.* Springer-Verlag, 1996.

[10] Emmanuel Kounalis. Completeness in data type specifications. In *Proc. EUROCAL 85, Linz, LNCS 204*, pages 348–362. Springer-Verlag, April 1985.

[11] C. Loría-Sáenz and J. Steinbach. Termination of combined (rewrite and λ-calculus) systems. In *Proc. 3rd Int. Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson, LNCS 656*, volume 656 of *Lecture Notes in Computer Science*, pages 143–147. Springer-Verlag, 1992.

[12] D. Lugiez. Positive and negative results for higher-order disunification. *Journal of Symbolic Computation*, 20 :431–470, 1995.

[13] P. Lincoln M. Clavel, S. Eker and J. Meseguer. Principles of maude. In J. Meseguer, editor, *Proceedings of the 1st International Workshop on Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science 4*, 1996.

[14] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In P. Schroeder-Heister, editor, *Extensions of Logic Programming.* LNCS 475, Springer Verlag, 1991.

[15] T. Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.

[16] Zhenyu Qian. Linear unification of higher-order patterns. In *TAP-SOFT'93 : Theory and Practice of Software Development*, volume 668 of *Lecture Notes in Computer Science*, pages 391–405. Springer-Verlag, 1993.

[17] J.-J. Thiel. Stop loosing sleep over incomplete specifications. In *Proc. 11th ACM Symp. on Principles of Programming Languages, Salt Lake City*, 1984.

Table des matières