Types et Interprétations pour des programmes de complexité polynomiale

Yann Duplouy,
encadré par Patrick Baillot,
dans l'équipe PLUME du Laboratoire de l'Informatique du Parallélisme
(ENS de Lyon)

21 août 2015

Le contexte général

Ce stage a porté sur l'étude de systèmes de types et d'interprétations pour montrer que certains programmes sont de complexité polynomiale. L'état de l'art principal étant :

Patrick Baillot a travaillé avec Ugo Dal Lago pour prouver, à l'aide d'interprétations polynomiales d'ordre supérieur, que dans certains systèmes de réécriture simplement typés (des STTRS), les programmes qui y sont construits sont calculés en temps polynomial.

D'autre part, Martin Hofmann a construit un lambda-calcul SLR qui généralise une caractérisation de PTIME faite par Bellantoni-Cook (qui est lui-même une classe spécifique, qu'on nommera BC, construite par définitions primitives récursives, permettant de construire l'ensemble des fonctions PTIME sur les entiers). Dans ce lambda-calcul, le système de types permet de montrer là aussi que tous les termes calculables le sont en temps polynomial.

Le problème étudié

Nous avons donc plusieurs manières de montrer que des programmes sont effectivement calculables en temps polynomial, mais est-ce que ces différentes manières sont des manières distinctes dans leur raisonnement, ou est-ce qu'il existe une manière de les relier qui montre qu'effectivement les restrictions proposées dans l'article de mon maitre de stage ne sont pas trop fortes, et que l'ensemble des fonctions calculables en temps $PTIME$ y sont bien ? On peut

donc s'intéresser à voir si le lambda-calcul SLR est au moins entièrement traduisible dans ce type de systèmes de réécriture...

La contribution proposée

Le travail principal de ce stage a été d'essayer de donner une traduction de SLR dans un système de types correspondant à ceux développés dans l'article de Patrick Baillot et de Ugo Del Lago, en montrant que les conditions énoncées dans cet article suffisent effectivement à montrer que dans cette traduction, les programmes construits correspondant à des termes valides sont aussi calculés en temps polynomial.

Cela n'a été fait au final que pour une restriction de SLR correspondant aux fonctions constructibles dans le système de réécriture de Bellantoni-Cook ; dans cette restriction, nous avons montré que toutes les interprétations des termes valides avaient une certaine forme, permettant de construire facilement les interprétations des nouveaux termes construits, et vérifiant bien que les conditions de l'article de Patrick Baillot et Ugo Del Lago sont vérifiées. L'idée a été donc de traduire BC dans la forme d'un sous-ensemble de SLR que l'on traduit ensuite sous la forme d'un STTRS sur lequel on a cherché (et trouvé) des interprétations montrant le caractère PTIME.

Les arguments en faveur de sa validité

Le système de réécriture construit pour cette restriction de SLR ne peut pas faire de calcul infini, et peut etre interprété par des polynômes qui vérifient les conditions nécessaires pour que le calcul soit effectué en temps PTIME : lorsque l'on effectue une étape de calcul, l'interprétation du nouveau terme est un polynôme inférieur à celui du terme de départ.

Le bilan et les perspectives

On a donc finalement montré que dans le cadre fourni par le travail de Patrick Baillot et Ugo Del Lago, il est possible d'attester que l'ensemble des fonctions PTIME sur les entiers l'est uniquement à l'aide de leurs systèmes de réécriture. On n'a néanmoins pas pu prouver que l'ensemble du langage SLR pouvait être traduit sous la forme de STTRS avec quasi-interprétations

1 Quelques briques de base

Nous allons voir, à travers ce rapport, certaines des méthodes développées ces dernières décennies pour montrer que des ensembles de fonctions sont calculables en temps polynomiales. Ces méthodes sont de deux ordres ; certains de ces ensembles de fonctions sont définis comme étant des classes de systèmes de réécriture, d'autres sont directement sous la forme d'un lambda-calcul.

L'objectif de cette partie est de donner les éléments nécessaires à la compréhension du travail effectué dans la partie suivante, en particulier les définitions ainsi que les théorèmes ; quelques exemples seront donnés en annexes pour permettre au lecteur de voir le fonctionnement de certaines des méthodes présentées dans cette section.

1.1 Systèmes de réécritures simplement typés à complexité polynomiale

Dans cette section, on présente les systèmes de réécritures simplement typés à complexité polynomiale, définis dans l'article de Patrick Baillot et Ugo Dal Lago[1], où l'idée principale est d'associer une interprétation, sous la forme d'un polynome, aux différents termes du système de réécriture et de montrer que si ces interprétations réduisent avec l'application des règles, alors le système de réécriture fait ses calculs en temps polynomial.

1.1.1 Rappels sur les systèmes de réécriture simplement typés

On considère en fait une sous-classe de ces systèmes de réécritures (qu'on va abbrévier en STTRS), où l'on considère un ensemble dénombrable de types de bases - des types de données, notés $D, E...$ et on construit des types à partir de ceux-là de la manière suivante :

$$A, B ::= D | A_1 \times ... \times A_n \to A$$

où les types construits avec $\to$ sont appelés types fonctionnels. On définit ensuite un ensemble de symboles de fonctions $\mathcal{F}$, un ensemble de constructeurs $\mathcal{C}$ et un ensemble $\mathcal{X}$ de variables. Les constructeurs $c \in \mathcal{C}$ sont de types $D_1 \times ... \times D_n \to D$. Les fonctions $f \in \mathcal{F}$ peuvent avoir n'importe quel type fonctionnel, et les variables $x \in \mathcal{X}$ n'importe quel type. De là, on peut définir des termes :

$$t, t_i ::= x^A \mid c^A \mid f^A \mid (t^{A_1 \times ... A_n \to A} t_1^{A_1} ... t_n^{A_n})$$

On note $\mathcal{T}$ l'ensemble de ces termes. On évitera par la suite de parler d'annoter les termes par leurs types. On dénote par $\mathrm{FV}(()t)$ l'ensemble des variables apparaissant dans t ; on dira que t est clos si $\mathrm{FV}(()t) = \emptyset$.

On définit ensuite un motif comme étant généré par la grammaire suivante :

$$p, p_i := x^A \mid (c^{D_1 ... D_n \to D} p_1^{D_1} ... p_n^{D_n})$$

et l'on définit finalement les règles de réécriture $l \to r$ telles que :

1. l et r sont des termes de même types, où les $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$ et chaque variable apparaît au plus une fois dans l
2. l doit avoir la forme $f p_1 ... p_k$ où chaque p_i n'est composé que de motifs. On définit l'*arité* de la règle comme étant le nombre de motifs présents dans la règle.

Définition 1. *Un système de réécriture simplement typé est un ensemble R de règles de réécritures ne se chevauchant pas*[1] *telles que pour chaque symbole de fonction f chaque règle pour f a la même arité, qui sera celle de la fonction.*

Définition 2. *On définit un programme $P = (f, R)$ par un* STTRS *R et un symbole de fonction $f \in \mathcal{F}$.*

On dit qu'un terme est une *valeur* si :
— soit il est d'un type de base D et est de la forme $(c\ v_1\ ...\ v_n)$ où $v_1, ..., v_n$ sont déjà des valeurs
— soit il est de type fonctionnel A et est de la forme $(f\ v_1\ ...v_n)$ où $v_1, ..., v_n$ sont des valeurs est n est strictement inférieur à l'arité de f

Substitution et contextes. L'évaluation des termes dans ces STTRS est alors défini par une relation de réécriture. On définit d'abord, pour cela, une *substitution* σ comme étant une fonction associant des variables $x \in \mathcal{X}$ d'un domaine fini à une valeur du même type ($\sigma(x^A)$ est de type A). On peut étendre cette substitution σ à une fonction de $\mathcal{T} \to \mathcal{T}$. On définit ensuite des *contextes* comme étant des termes contenant exactement une occurence d'une constante spéciale, $\bullet^A$ (un *trou*) de type A. Si $\mathcal{C}$ est un contexte avec un trou $\bullet^A$ alors $\mathcal{C}(t)$ est le terme obtenu à partir de $\mathcal{C}$ en remplaçant l'occurence de $\bullet^A$

1. c'est-à-dire que deux motifs différents ne peuvent pas reconnaître la même expression

Réduction des termes d'un STTRS**.** Finalement, soit un STTRS R. Alors t réduit vers s, $t \to_R s$, s'il existe une règle $l \to r$ de R, un contexte $\mathcal{C}$ et une substitution σt telle que $l\sigma$ est un terme clos, $t = \mathcal{C}(l\sigma)$ et $s = \mathcal{C}(r\sigma)$.

Proposition 1 (Confluence forte)**.** *Soit R un* STTRS*, et $t \to_R s_1$ et $t \to_R s_2$. Alors il existe s tel que $s_1 \to_R s$ et $s_2 \to_R s$*

On peut voir dans l'article présentant ces STTRS que cela suffit à simuler un lambda-calcul tel que PCF.
i

1.1.2 Quasi-interprétations et ordre supérieurs

Un des problèmes de l'approche que l'on vient de voir est que l'expressivité de ce que l'on peut interpréter est trop limitée. L'idée est donc d'étendre l'approche pour englober plus de STTRS et de fonctions calculables de cette manière.

Un système de types linéaires pour des STTRS On commence par définir un système de types pour ces systèmes de réécritures, qui nécessite de découper l'ensemble des fonctions $\mathcal{F}$ en deux ensembles, $\mathcal{NF}$ ne pouvant être défini d'une manière récursive et $\mathcal{RF}$ pouvant l'être :

$$\frac{f^A \in \mathcal{NF}}{\Gamma \mid \Delta \vdash f : A} \quad \frac{c^A \in \mathcal{C}}{\Gamma \mid \Delta \vdash c : A} \quad \frac{}{\Gamma \mid x : A, \Delta \vdash x : A} \quad \frac{}{x : D, \Gamma \mid \Delta \vdash x : D}$$

$$\frac{f^{A_1,\ldots,A_n \to B} \in \mathcal{RF} \text{ d'arité } n \quad \Gamma \mid \emptyset \vdash s_i : A_i}{\Gamma \mid \Delta \vdash (f\ s_1\ \ldots\ s_n) : B} \quad \frac{\Gamma \mid \Delta \vdash t : A_1 \times \ldots \times A_n \to B \quad \Gamma \mid \Delta_i \vdash s_i : A_i}{\Gamma \mid \Delta, \Delta_1, \ldots \Delta_n \vdash (t\ s_1\ \ldots s_n) : B}$$

Dans un jugement $\Gamma \mid \Delta \vdash t : A$ ainsi construit, le sous-contexte Δ est censé contenir des variables linéaires, et Γ les variables non-linéaires.

Critère de terminaison. L'objectif du système de type que l'on vient de définir est de définir un critère de terminaison. On suppose connu un ordre strict bien fondé $\sqsubset$ sur $\mathcal{F}$. Si t est un terme, alors $t \sqsubset f$ si pour tout $g \in \mathcal{F}$ apparaissant dans t, $g \sqsubset f$. On dira qu'un STTRS vérifie le *critère de terminaison* si, pour toute règle $(f\ \overline{p_1}\ \ldots\ \overline{p_k}) \to s$ [2] vérifie :

2. où les p_i sont toujours des motifs

— soit $f \in \mathcal{RF}$, et il existe un terme r et des motifs $\overline{q_1}...\overline{q_k}$ tels que : $s = r\{x/(f\ \overline{q_1}...\overline{q_k})\}$, on a $\Gamma \mid x : B, \Delta \vdash r : B, r \sqsubset f$ et pour tout i, j $q_{i,j}$ est un sous-terme de $p_{i,j}$ et il existe au moins un de ceux-là qui est un sous-terme strict.
— ou alors $\Gamma \mid \Delta \vdash s : B$ et $s \sqsubset f$

où $\overline{p_i} = c^{D_1...D_n \to D} p_{i,1}^{D_1} ... p_{i,n}^{D_n}$.

Remarque. *Remarquons que la condition sur le typage du cas récursif force à ce qu'il y ait au plus un appel récursif dans le côté droit d'une règle.*

Max-polynomes d'ordre supérieur On étend maintenant la définition des polynômes d'ordre supérieur. La nouvelle grammaire de type est la suivante :

$$S ::= \mathrm{N} | S \multimap S \quad A ::= S | A \to A$$

les types S sont appelés types linéaires, et le type fonction linéaire est un sous-type des types fonctions. On étend les constructeurs de la manière suivante :

$$D_P = \{+ : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}, \max : \mathrm{N} \multimap \mathrm{N} \multimap \mathrm{N}, \times : \mathrm{N}^2 \to \mathrm{N}\} \cup \{n : \mathrm{N} | n \in \mathbb{N}^*\}$$

et l'on définit la grammaire de termes :

$$M, P := x^A \mid c^A \mid (M^{A \to B} P^A)^B \mid (\lambda x^A . M^B)^{A \to B} \mid (M^{S \multimap R} P^S)^R \mid (\lambda x^S . M^R)^{S \multimap R}$$

où $c^A \in D_P$. On demande, de plus, à ce que pour $(\lambda x^A . M^B)^{A \to B}$, x^A apparaisse au moins une fois dans M^B et que pour $(\lambda x^S . M^R)^{S \multimap R}$ la variable x^S apparaisse exactement une fois dans M^R et dans une position linéaire (donc pas du côté droit d'une application $P^{A \to B} L^A$).

Cette classe est préservée par β-réduction, et on définit les *max-polynômes d'ordre supérieur* (HOMP [3]) comme un terme construit de cette manière et de forme β-réduite.

Interprétation sémantique On définit une catégorie $\mathbb{FPOS}$ avec les objets et constructions suivantes :

— $\mathcal{N}$ est le domaine des entiers *strictement positifs* équipés avec l'ordre partiel naturel noté ici $\leq_{\mathcal{N}}$
— 1 avec l'ordre trivial à un point
— si σ, τ sont des objets alors $\sigma \times \tau$ est obtenu par ordre produit

3. Higher-Order Max-Polynomials

— $\sigma \Rightarrow \tau$ est l'ensemble des fonctions totales monotones comme morphismes équipés de l'ordre $f \leq_{\sigma\Rightarrow\tau} g$ si pour tout a de σ on a $f(a) \leq_\tau g(a)$

On aura parfois à comparer les sémantiques de termes n'ayant pas les mêmes variables libres : dans ce cas, si $f \in \sigma_1 \times ... \times \sigma_n \to \tau$, et $g \in \sigma_1 \times ... \times \sigma_m \to \tau$ et $n \geq m$ alors $f \leq g$ si $\forall a_1 \in \sigma_1 ... \forall a_m \in \sigma_m : f(a_1...a_n) \leq_\tau g(a_1, ..., a_m)$

Tailles. Pour interpréter la construction de types fonctionnels linéaires dans les HOMP, on définit une notion de taille, un multiset fini d'éléments de $\mathbb{N}$.

Soit σ un objet de $\mathbb{FPOS}$, alors un élément $e \in \sigma$ admet une taille dans les cas suivants :

— si σ est $\mathcal{N}$, e est un entier n, et $\mathcal{S}$ est une taille de e si $\max \mathcal{S} \leq n \leq \sum \mathcal{S}$

— Si $\sigma = \sigma_1 \times ... \times \sigma_n$ alors $\mathcal{S}$ est une taille de $e = (e_1...e_n)$ si pour tout $i \in \{1, ..., n\}$ il existe $\mathcal{S}_i$ taille de e_i, tel que $\mathcal{S} = \cup_{i=1}^n S_i$

— Si $\sigma = \tau \Rightarrow \rho$, alors $\mathcal{S}$ est une taille de e si pour tout f de τ ayant une taille $\mathcal{T}$, $\mathcal{S} \cup \mathcal{T}$ est une taille de $e(f)$. On définit $\tau \multimap \rho$ comme le sous-ensemble des fonctions de σ admettant une taille.

On dénote toujours $[\![A]\!]_\leq$ la sémantique de A comme un objet de $\mathbb{FPOS}$ (où N est envoyé sur $\mathcal{N}$, $\to$ sur $\Rightarrow$ et $\multimap$ sur $\multimap$). On peut donc interpréter un HOMP M comme une fonction monotone $[\![M]\!]_\leq$. On parlera alors de la *taille* de M comme étant la taille de son inteprétation $[\![M]\!]_\leq$.

1.2 Définition de BC

Stephen Bellantoni et Stephen Cook définissent dans un article[2] une classe B de systèmes de réécritures terminant en temps polynomial, la plus petite classe contenant les constructeurs suivants :

— la constante 0

— les successeurs $S_0(x)$ et $S_1(x)$

— les projections $\pi_j^{n,m}(x_1...x_n; x_{n+1}, ..., x_{n+m}) \to x_j$ pour $1 \leq j \leq n+m$

— la fonction prédecesseur $p(;0) \to 0$, $p(; S_i(x) \to x)$

— la conditionnelle $C(;0,x,y) \to x$, $C(;S_0,x,y) \to x$ et $C(;S_1,x,y) \to y$

et qui soit close par :

— récursion prédicative :

$$\begin{aligned} f(0, x_1...x_n; y_1...y_m) &\to g(x_1...x_n; y_1...y_m) \\ f(S_i(z), x_1...x_n; y_1...y_m) &\to h_i(z, x_1...x_n; y_1...y_m, f(z, x_1...x_n; y_1...y_m)) \end{aligned}$$

pour $i \in \{0,1\}$ et g et h_i définies précédemment dans B

— composition sûre :

$$f(x_1...x_n; y_1...y_m) \rightarrow g(h_1(x_1...x_n), ..., h_p(x_1...x_n); l_1(x_1...x_n; y_1...y_m), ...l_q(x_1, ...x_n; y_1...y_m))$$

où g, h_i et l_j sont définies précédemment dans B.

Proposition 2. *Toute fonction calculable par un système de réécriture de B est de classe* PTIME.

1.3 Définition de SLR

On ne présentera SLR que de manière succinte dans cette section ; on trouvera plus de détails quant à ce lambda-calcul dans l'article de Martin Hofman[**?**]. Il s'agit d'un lambda-calcul visant à généraliser la caractérisation de PTIME que l'on vient de voir à des fonctions d'ordre supérieur.

Aspects Ce lambda-calcul repose sur des ensembles de fonctions permettant de restreindre la capacité de récurrence. Pour distinguer ces espaces de fonctions, on utilise des *aspects*, paires de la forme $a = (m, l)$ où $m \in \{\text{modal}, \text{non-modal}\}$ et $l \in \{\text{linéaire}, \text{non-linéaire}\}$. On peut alors définir une notation générique $A \xrightarrow{a} B$ pour des ensembles de fonctions. Par exemple, $A \multimap B$ désigne $A \xrightarrow{a} B$ où $a = (\text{non-modal}, \text{linéaire})$, et $\Box A \rightarrow B$ le cas où $a = (\text{modal}, \text{non-linéaire})$.

Types Les types de SLR sont définis par la grammaire suivante :

A, B	::=	X	variable de type
	|	N	les entiers
	|	L(A)	les listes
	|	T(A)	les arbres
	|	$A \xrightarrow{a} B$	
	|	$\forall X.A$	types polymorphes
	|	$A \times B$	produit cartésien
	|	$A \otimes B$	produit tensoriels

On caractérise un type comme étant *sûr* s'il est construit à partir de variables de types de N en utilisant les opérateurs L($--$), T($--$), $\multimap$, $\times$ et $\otimes$. On restreint de plus l'utilisation de L(A) et T(A) aux cas où A est sûr. Une relation de sous-typage est définie dans l'article ; elle n'est pas nécessaire pour la suite de ce rapport. Il est néanmoins intéressant de noter que, dans ce système, on peut dupliquer les entiers naturels sans casser la linéarité.

Termes On peut alors définir les expressions de SLR :

e	$::=$	x	variables
	$\mid$	$(e_1 e_2) \mid \lambda x : A.e$	application et abstraction
	$\mid$	$\Lambda X.e \mid e[A]$	abstraction et application de type
	$\mid$	$\langle e_1, e_2 \rangle \mid e.1 \mid e.2$	paire ($\times$) et projections
	$\mid$	$e_1 \otimes e_2 \mid \text{let } e_1 = x \otimes y \text{ in } e_2$	paire ($\otimes$) et élimination
	$\mid$	c	constantes

Constantes On dispose alors des constantes suivantes :

$$
\begin{array}{rcl}
0 & : & \mathrm{N} \\
\mathrm{S}_0, \mathrm{S}_1 & : & \mathrm{N} \multimap \mathrm{N} \\
\mathrm{rec}^{\mathrm{N}} & : & \forall X.X \multimap \Box(\Box\mathrm{N} \to X \multimap X) \to \Box\mathrm{N} \to X \\
\mathrm{case}^{\mathrm{N}} & : & \forall X.(X \times (\mathrm{N} \multimap X) \times (\mathrm{N} \multimap X)) \multimap \mathrm{N} \multimap X \\
\mathrm{nil} & : & \forall A.\mathrm{L}(A) \\
\mathrm{cons} & : & \forall A.A \multimap \mathrm{L}(A) \multimap \mathrm{L}(A) \\
\mathrm{rec}^{\mathrm{L}} & : & \forall A.\forall X.X \multimap \Box(\Box A \to \Box\mathrm{L}(A) \to X \multimap X) \to \Box\mathrm{L}(A) \to X \\
\mathrm{case}^{\mathrm{L}} & : & \forall A.\forall X.X \times (A \multimap \mathrm{L}(A) \multimap X) \multimap \mathrm{L}(A) \multimap X \\
\mathrm{leaf} & : & \forall A.A \multimap \mathrm{T}(A) \\
\mathrm{node} & : & \forall A.A \multimap \mathrm{T}(A) \multimap \mathrm{T}(A) \multimap \mathrm{T}(A)
\end{array}
$$

Le fonctionnement de 0, S_0, S_1, nil, cons, leaf et node est intuitif ; on remarquera néanmoins que les arbres ainsi construits sont binaires. Pour le fonctionnement des autres symboles, cela donne [4] :

- $\mathrm{rec}^{\mathrm{N}}(X, h, g, 0) \to 0$
- $\mathrm{rec}^{\mathrm{N}}(X, h, g, \mathrm{S}_i(x)) \to h(\mathrm{S}_i(x), \mathrm{rec}^{\mathrm{N}}(X, h, g, x))$
- $\mathrm{rec}^{\mathrm{L}}(A, X, g, h, \mathrm{nil}) \to g$
- $\mathrm{rec}^{\mathrm{L}}(A, X, g, h, \mathrm{cons}(a, l)) \to h(a, l, \mathrm{rec}^{\mathrm{L}}(A, X, g, h, l))$
- $\mathrm{rec}^{\mathrm{T}}(A, X, g, h, \mathrm{leaf}(a)) \to g(a)$
- $\mathrm{rec}^{\mathrm{T}}(A, X, g, h, \mathrm{node}(a, l, r)) \to h(a, l, r, \mathrm{rec}^{\mathrm{T}}(A, X, g, h, l), \mathrm{rec}^{\mathrm{T}}(A, X, g, h, r))$

ce qui permet effectivement de faire des récurrences sur des listes et des arbres. Pour les constantes permettant de faire de la distinction par cas, on envoie dans le cas attendu de la manière suivante :

- $\mathrm{case}^{\mathrm{N}}(X, g, h_0, h_1, 0) = g$
- $\mathrm{case}^{\mathrm{N}}(X, g, h_0, h_1, \mathrm{S}_i(x)) = h_i(\mathrm{S}_i(x))$
- $\mathrm{case}^{\mathrm{L}}(A, X, h_{nil}, h_{cons}, \mathrm{nil}) = h_{nil}$
- $\mathrm{case}^{\mathrm{L}}(A, X, h_{nil}, h_{cons}, \mathrm{cons}(a, l)) = h_{cons}(a, l)$
- $\mathrm{case}^{\mathrm{T}}(A, X, h_{node}, h_{leaf}, \mathrm{node}(a)) = h_{node}(a)$
- $\mathrm{case}^{\mathrm{T}}(A, X, h_{node}, h_{leaf}, \mathrm{leaf}(a, l, r)) = h_{leaf}(a, l, r)$

4. X, A, désignant des types permettant de faire l'application de types

Contextes Un *contexte* est une fonction partielle de variables vers des paires d'aspects et de types ; on l'écrit généralement sous une liste d'éléments de la forme $x^a : A$. Si Γ est un contexte, $\mathrm{dom}(\Gamma)$ est l'ensemble des variables liées dans Γ, et si $x^a : A \in \Gamma$ alors $\Gamma(x) = A$ et $\Gamma((x)) = a$. On dira en particulier que Γ est non-linéaire si l'ensemble de ses éléments est d'aspect non-linéaire.

Typage La relation de typage $\Gamma \vdash e : A$ entre contextes, expressions et types est définie de façon inductive par l'ensemble des règles décrite en aannexe A

2 De BC à un fragment de SLR pour une traduction en $STTRS$

2.1 Le fragment de SLR correspondant à BC

Nous avons vu dans la partie précédente les définitions (succintes) de SLR et BC. Nous allons définir une restriction de SLR qui correspond aux fonctions que l'on pourrait exprimer dans BC. Cette restriction, que l'on nommera $\mathrm{SLR}^{\mathrm{BC}}$, est construite comme suit :

Types Les types de $\mathrm{SLR}^{\mathrm{BC}}$ sont donnés par la grammaire suivante :

$$A ::= N | N \xrightarrow{a} A$$

où a désigne l'*aspect* d'une fonction, sous la forme toujours d'une paire $a = (m, l)$ où $m = \{\text{modal}, \text{non-modal}\}$ et $l \in \{\text{linéaire}, \text{non-linéaire}\}$. Si une fonction est modale, elle ne peut être linéaire, et il existe un ordre sur les aspects : (modal, non-linéaire) < (non-modal, non-linéaire) < (non-modal, linéaire). L'idée étant toujours de séparer les termes sur lesquels il reste possible de faire une récurrence de ceux où rajouter une récurrence nous ferait sortir du cadre $PTIME$.

Termes Les expressions de $\mathrm{SLR}^{\mathrm{BC}}$ sont données par la grammaire suivante :

$$\begin{array}{rcll} e & ::= & x & \text{(variables)} \\ & | & (e_1 e_2) & \text{(application)} \\ & | & \lambda x : A.e & \text{(abstraction)} \\ & | & c & \text{(constantes)} \end{array}$$

où x fait partie d'un ensemble dénombrable de variables, et où c peut être une des constantes suivantes, avec les types indiqués :

$$\begin{array}{rcl} 0 & : & \mathrm{N} \\ \mathrm{S}_0, \mathrm{S}_1 & : & \mathrm{N} \multimap \mathrm{N} \\ \mathrm{rec}^{\mathrm{N}} & : & \mathrm{N} \multimap \Box(\Box\mathrm{N} \to \mathrm{N} \multimap \mathrm{N}) \to \Box\mathrm{N} \to \mathrm{N} \\ \mathrm{case}^{\mathrm{N}} & : & (\mathrm{N} \times (\mathrm{N} \multimap \mathrm{N}) \times (\mathrm{N} \multimap \mathrm{N})) \multimap \mathrm{N} \multimap \mathrm{N} \end{array}$$

Typage Les règles de typage de $\mathrm{SLR}^{\mathrm{BC}}$ sont les suivantes :

$$\frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x^a : A \vdash e : B}{\Gamma \vdash \lambda x : A.e : A \xrightarrow{a} B} \quad \text{(T-Arr-I)}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : A \xrightarrow{a} B \quad \Gamma, \Delta_2 \vdash e_2 : A \quad \Gamma \text{ non-linéaire} \quad x^{a'} : X \in \Gamma, \Delta_2 \Rightarrow a' \leq a}{\Gamma, \Delta_1, \Delta_2 \vdash (e_1 e_2) : B} \quad \text{(T-Arr-E)}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : \mathrm{N} \quad \Gamma, \Delta_2 \vdash e_2 : \Box(\Box\mathrm{N} \to \mathrm{N} \multimap \mathrm{N})}{\Gamma, \Delta_1, \Delta_2 \vdash \mathrm{rec}^{\mathrm{N}}\ e_1\ e_2 : \Box\mathrm{N} \to \mathrm{N}} \quad \text{(T-recN)}$$

$$\frac{\Gamma, \Delta_1 \vdash e_0 : \mathrm{N} \quad \Gamma, \Delta_2 \vdash e_1 : \mathrm{N} \multimap N \quad \Gamma, \Delta_3 \vdash e_2 : \mathrm{N} \multimap N}{\Gamma, \Delta_1, \Delta_2, \Delta_3 \vdash \mathrm{case}^{\mathrm{N}}\ e_0\ e_1\ e_2 : \mathrm{N} \multimap \mathrm{N}} \quad \text{(T-caseN)}$$

$$\frac{c : A}{\Gamma \vdash c : A \text{ si } c \notin \{\mathrm{rec}^{\mathrm{N}}, \mathrm{case}^{\mathrm{N}}\}} \quad \text{(T-Const)}$$

Il est nécessaire d'avoir des règles spécifiques pour $\mathrm{case}^{\mathrm{N}}$ et $\mathrm{rec}^{\mathrm{N}}$, l'application – limitée à des fonctions de la forme $\mathrm{N} \to A$ – ne permettant pas de gérer ces deux fonctions.

En particulier, la règle qui va nous intéresser est T-Arr-E : c'est là que se joue la limitation de type des variables : en particulier, si la fonction e_1 demande un argument modal (est de type $\Box N \to A$), c'est alors toutes les variables (libres) de e_2 qui devront être modales. Cela jouera un rôle particulier dans la traduction dans le STTRS associé, que l'on verra maintenant.

2.2 Traduction vers un STTRS

2.2.1 Termes dans le STTRS et traduction de termes de $\mathrm{SLR}^{\mathrm{BC}}$

On suppose que tous les termes que l'on traduira sont correctement typés dans $\mathrm{SLR}^{\mathrm{BC}}$. La traduction effective se fera à l'aide de l'arbre de typage utilisé. La traduction d'un terme de $\mathrm{SLR}^{\mathrm{BC}}$ t en un terme du STTRS que l'on construit, $\langle t \rangle$, se fait de la manière suivante, suivant la dernière règle utilisée (et par induction) :

— s'il s'agit de T-recN ou de T-caseN :

$$\begin{aligned} \langle \mathrm{rec}^{\mathrm{N}}\ e_1\ e_2 \rangle &= \mathtt{recnat}\langle e_1 \rangle \langle e_2 \rangle \\ \langle \mathrm{case}^{\mathrm{N}}\ e_0\ e_1\ e_2 \rangle &= \mathtt{casenat}\langle e_0 \rangle \langle e_1 \rangle \langle e_2 \rangle \end{aligned}$$

— s'il s'agit de T-Cons, on traduit le constructeur correspondant dans un constructeur équivalent

$$\begin{aligned} \langle 0 \rangle &= \texttt{zero} \\ \langle \mathrm{S}_0 \rangle &= \mathtt{S_0} \\ \langle \mathrm{S}_1 \rangle &= \mathtt{S_1} \end{aligned}$$

— s'il s'agit de T-var, on garde la variable telle quelle dans le STTRS
— Si on utilise T-Arr-I, on est dans le cas d'une abstraction, et $t = \lambda x : A.e$. On désigne un ordre sur les variables libres de e respectant l'ordre des typages que l'on a donné au-dessus (les variables libres de type modales sont en premier) $x_1 < ... < x_n$, et la traduction est alors définie comme :

$$\langle \lambda x : A.e \rangle = \mathtt{abs}_{\langle e \rangle, x}\, x_1 ... x_n$$

— Si on utilise T-Arr-E, il faut différencier selon l'aspect – cela permettra (plus loin) d'écrire une définition efficace des interprétations :
  — s'il s'agit d'une fonction modale, e_2 ne peut contenir de variable libre non-modale (par les hypothèses de la règle) :

$$\langle e_1\ e_2 \rangle = \mathtt{app}^{\Box}\ (\mathtt{abs}_{\langle e_1 \rangle, \vec{x}\vec{y}})\ (\mathtt{abs}_{\langle e_2 \rangle, \vec{x}})$$

  — s'il s'agit d'une autre forme de fonction :

$$\langle e_1\ e_2 \rangle = \mathtt{app}\ (\mathtt{abs}_{\langle e_1 \rangle, \vec{x}\vec{y}})\ (\mathtt{abs}_{\langle e_2 \rangle, \vec{x}\vec{y}})$$

  où $\vec{x}$ représente l'ensemble des variables libres modales de e_1 et e_2, et $\vec{y}$ les variables libres non-modales (et le `abs` à plusieurs variables est exactement la même chose qu'une itération des `abs`)

2.2.2 Règles de réécriture

Le STTRS que l'on construit a les règles de réécritures suivantes :

$$\begin{aligned} \mathtt{app}\ e_1\ e_2\ \vec{x}\ \vec{y} &\to (e_1\ \vec{x}\ \vec{y})\ (e_2\ \vec{x}\ \vec{y}) \\ \mathtt{app}^{\Box}\ e_1\ e_2\ \vec{x}\ \vec{y} &\to (e_1\ \vec{x}\ \vec{y})\ (e_2\ \vec{x}\ \vec{y}) \\ (\mathtt{abs}_{\langle e \rangle, x}\ x_1\ ...\ x_n)\ y &\to e[x \backslash y] \\ \mathtt{recnat}\ h\ g\ \mathtt{zero} &\to g \\ \mathtt{recnat}\ h\ g\ \mathtt{S_0}(x) &\to h\ \mathtt{S_0}(x)\ (\mathtt{recnat}\ h\ g\ x) \\ \mathtt{recnat}\ h\ g\ \mathtt{S_1}(x) &\to h\ \mathtt{S_1}(x)\ (\mathtt{recnat}\ h\ g\ x) \\ \mathtt{casenat}\ g\ h_0\ h_1\ \mathtt{zero} &\to g \\ \mathtt{casenat}\ g\ h_0\ h_1\ \mathtt{S_0}(x) &\to h_0(x) \\ \mathtt{casenat}\ g\ h_0\ h_1\ \mathtt{S_1}(x) &\to h_1(x) \end{aligned}$$

Proposition 3. *Ces règles de réécriture terminent bien.*

Démonstration. Les règles correspondant à $\texttt{app}, \texttt{app}^{\Box}$ ne font pas apparaître de fonction dans la partie droite : $e_1\ e_2 \sqsubset \texttt{app}\ e_1\ e_2$. Pour la règle de `abs`, il suffit de faire en sorte que l'ordre $\sqsubset$, sur les `abs`, soit compatible avec l'ordre des sous-termes (ce qui permet, par exemple, de gérer des abstractions qui se suivent) :

$$e_1 \text{ est un sous-terme de } e_2 \Rightarrow \texttt{abs}_{e_1,x} \sqsubset \texttt{abs}_{e_2,x}$$

De même, pour les `casenat`, l'ordre fait en sorte que `casenat` soit plus grand que toutes les autres fonctions.

Pour `recnat`, il n'y a pas de récursion dans le cas du `zero` (le premier argument que l'on a donné fonctionne). Pour les deux autres règles, remarquons que $h\ \texttt{S}_0(x)\ (\texttt{recnat}\ h\ g\ x) = (h\ \texttt{S}_0(x)\ y)[(\texttt{recnat}\ h\ g\ x)\backslash y]$ et qu'on utilise bien seulement des sous-termes des termes initialement donnés dans la récursion, avec l'un d'entre eux strictement plus petit. □

2.2.3 Temps polynomial

L'objectif de cette section est de montrer que le STTRS que l'on a ainsi construit fait ses calculs en temps polynomial, ce qui avec la propriété de terminaison, ne nécessite plus que de trouver des quasi-interprétations aux termes du STTRS. On va donc chercher à montrer le lemme suivant :

Lemme 4. *Soit t un terme du* STTRS *représentant* $\mathrm{SLR}^{\mathrm{BC}}$*. Alors il existe un polynôme q tel que : $[t] = \lambda\vec{x}\lambda\vec{y}\lambda\vec{v}\ q(\vec{x}, \vec{v}_{\Box}) + \max(\vec{y}, \vec{v}_{\neg\Box})$ et ces polynômes (croissants en tous leurs arguments) vérifient la décroissance des interprétations avec l'application des règles de réécriture*

où l'on aura préalablement trié les variables libres du terme t de sorte à ce que $\vec{x}$ contienne toute les variables libres modales, et $\vec{y}$ toutes les variables libres non-modales. $\vec{v}$ représente l'ensemble des variables liées de t, $\vec{v}_{\Box}$ ses variables liées modales et $\vec{v}_{\neg\Box}$ le reste de ses variables liées.

Les interprétations suivantes conviennent :

$$\begin{aligned}
[\texttt{zero}] &= 1 \\
[\texttt{S}_0] &= \lambda x.x + 1 \\
[\texttt{S}_1] &= \lambda x.x + 1 \\
[\texttt{recnat}] &= \lambda\phi.\lambda\psi.\lambda x.\psi + x\phi(x, 0) \\
[\texttt{casenat}] &= \lambda x.\lambda\psi.\lambda\phi.(\max(x, \psi(x), \phi(x)) \\
[\texttt{abs}_{t,x}] &= [t] \text{ en replaçant le } \lambda x. \text{ en tête} \\
[\texttt{app}] &= \lambda f\lambda g\lambda\vec{x}\lambda\vec{y}.f(\vec{x}, \vec{0}, 0) + g(\vec{x}, 0) + \max(\vec{y}) \\
[\texttt{app}^{\Box}] &= \lambda f\lambda g\lambda\vec{x}\lambda\vec{y}.f(\vec{x}, \vec{0}, g(\vec{x}, 0)) + \max(\vec{y})
\end{aligned}$$

Remarquons que l'on a défini les interprétations de `recnat` et `casenat` dans le cas où il n'y a pas de variables libres. On peut adapter ces définitions dans le cas où les variables libres sont représentées par des vecteurs $\vec{x}$ et $\vec{y}$ (choisies dans le même ordre que $\vec{v}_{\Box}$ et $\vec{v}_{\neg\Box}$:

$$\begin{array}{rcl} [\texttt{recnat}] & = & \lambda\vec{x}.\lambda\vec{y}.\lambda\phi.\lambda\psi.\lambda x.\psi(\vec{x},\vec{y}) + x\phi(\vec{x},\vec{y},x,0) \\ [\texttt{casenat}] & = & \lambda\vec{x}.\lambda\vec{y}.\lambda x.\lambda\psi.\lambda\phi.(\max(x,\psi(\vec{x},\vec{y},x),\phi(\vec{x},\vec{y},x)) \end{array}$$

Démonstration. Il nous reste donc à montrer que les interprétations décroissent avec l'application des règles.

- Pour `casenat`, le résultat de la réduction est toujours exactement un des trois arguments (en plus de l'entier), lui-même argument du maximum (ce qui entraîne une décroissance non-stricte)
- Pour `recnat`, on a deux cas :
  - celui de l'initialisation (pour l'entier zéro), on ne retrouve qu'un élément de l'interprétation dans l'interprétation de la forme réduite (ψ)
  - pour le cas de la récurrence, l'interprétation du terme réduit est

$$\begin{array}{rcl} G & = & [h\ (\texttt{S}_\texttt{i}\ x)\ (\texttt{recnat}\ \ h\ g\ x)] = [h](x,[g] + x[h](x,0)) \\ & = & q_h(x) + [g] + x[h](x,0) = (x+1)q_h(x) + [g] \end{array}$$

    et l'interprétation du terme de départ est :

$$\begin{array}{rcl} D & = & [\texttt{recnat}\ \ h\ g\ \ \texttt{S}_\texttt{i}(x)] = [g] + (x+1)[h](x+1,0) \\ & = & [g] + (x+1)q_h(x+1) \end{array}$$

    en remarquant que les interprétations sont définies comme étant croissantes en tous leurs arguments, on remarque la décroissance de l'interprétation.
- Pour les applications, le terme réduit est le même (e_1e_2) mais son interprétation diffère suivant la modalité de la fonction e_1 :
  - si c'est une fonction modale, dans le cas sans variables libres, alors l'ensemble des variables utilisées dans e_2 est forcément modal (par fonctionnement de la règle E-Arr-E) :

$$G = [(e_1\ \vec{x}\ \vec{y})\ (e_2\ \vec{x})] = [e_1](\vec{x},\vec{y},e_2(\vec{x})) = q(\vec{x},e_2(\vec{x})) + \max(\vec{y})$$

  - sinon :

$$\begin{array}{rcl} G & = & [e_1e_2] = [(e_1\ \vec{x}\ \vec{y})\ (e_2\ \vec{x}\ \vec{y})] = [e_1](\vec{x},\vec{y},[e_2](\vec{x},\vec{y})) \\ & = & q_1(\vec{x}) + \max([e_2](\vec{x},\vec{y}),\vec{y}) = q_1(\vec{x}) + q_2(\vec{x}) + \max(\vec{y}) \end{array}$$

Dans les deux cas, cela correspond à l'interprétation que l'on a construit pour chacun des symboles $\texttt{app}$ et $\texttt{app}^{\Box}$ (et on a donc de nouveau une décroissance non-stricte)

$\Box$

3 Extension à la traduction de SLR et problèmes rencontrés

La création d'un sous-ensemble particulier de SLR, que l'on a nommé SLR^{BC}, permettait de se restreindre au cas où l'ensemble des variables était de type N ce qui permettait en particulier de ne plus avoir à prendre en compte l'aspect linéaire des fonctions (on a évoqué dans la première partie que l'on pouvait, dans SLR, dupliquer les entiers sans causer de souci). Si l'on peut traduire SLR en grande majorité, l'extension à de plus nombreux types pose des problèmes pour l'interprétation de l'application.

Dans cette section, nous verrons une traduction d'un sous-ensemble plus large de SLR ainsi que des interprétations permettant de montrer qu'un sous-ensemble de SLR est calculable en temps polynomial, et l'on verra ensuite les limitations pour le cas de l'application.

3.1 Traduction de SLR dans un STTRS

On peut effectuer une traduction entière de SLR vers un STTRS, de la même manière que pour SLR^{BC} (c'est-à-dire en remontant les règles de typages, qui sont définies dans l'annexe).

$$
\begin{aligned}
\langle x \rangle &= x \\
\langle (e_1 e_2) \rangle &= (\langle e_1 \rangle \langle e_2 \rangle) \\
\langle \lambda x : A.e \rangle &= \mathtt{abs}_{\langle e \rangle, x}\, x_1 ... x_n \\
\langle < e_1, e_2 > \rangle &= \mathtt{pair}\ \langle e_1 \rangle\ \langle e_2 \rangle \\
\langle e.1 \rangle &= \mathtt{proj1}\ e \\
\langle e.2 \rangle &= \mathtt{proj2}\ e \\
\langle e_1 \otimes e_2 \rangle &= \mathtt{lpair}\ \langle e_1 \rangle\ \langle e_2 \rangle \\
\langle \text{let } e_1 = x \otimes y \text{ in } e_2 \rangle &= \mathtt{let} \langle e_1 \rangle\ x\ y \langle e_2 \rangle \\
\langle 0 \rangle &= \mathtt{zero} \\
\langle \mathrm{S}_0 \rangle &= \mathtt{S}_0 \\
\langle \mathrm{S}_1 \rangle &= \mathtt{S}_1 \\
\langle \mathrm{rec}^{\mathbf{N}} \rangle &= \mathtt{recnat} \\
\langle \mathrm{case}^{\mathbf{N}} \rangle &= \mathtt{casenat} \\
\langle \mathrm{nil} \rangle &= \mathtt{nil} \\
\langle \mathrm{cons} \rangle &= \mathtt{cons} \\
\langle \mathrm{rec}^{\mathbf{L}} \rangle &= \mathtt{listrec} \\
\langle \mathrm{case}^{\mathbf{L}} \rangle &= \mathtt{listcase}
\end{aligned}
$$

Pour ce qui est des règles de réduction, on a :

$$\begin{array}{rcl}
\texttt{proj1}(\texttt{pair}\ t_1\ t_2) & \to & t_1 \\
\texttt{proj2}(\texttt{pair}\ t_1\ t_2) & \to & t_2 \\
\texttt{let}((\texttt{lpair}\ t_1\ t_2)\ x\ y\ t & \to & t\{t_1/x, t_2/y\} \\
\texttt{recnat}\ h\ g\ \texttt{zero} & \to & g \\
\texttt{recnat}\ h\ g\ \texttt{S}_0(x) & \to & h\ \texttt{S}_0(x)\ (\texttt{recnat}\ h\ g\ x) \\
\texttt{recnat}\ h\ g\ \texttt{S}_1(x) & \to & h\ \texttt{S}_1(x)\ (\texttt{recnat}\ h\ g\ x) \\
\texttt{casenat}\ g\ h_0\ h_1\ \texttt{zero} & \to & g \\
\texttt{casenat}\ g\ h_0\ h_1\ \texttt{S}_0(x) & \to & h_0(x) \\
\texttt{casenat}\ g\ h_0\ h_1\ \texttt{S}_1(x) & \to & h_1(x) \\
\texttt{listrec}\ g\ h\ \texttt{nil} & \to & g \\
\texttt{listrec}\ g\ h\ \texttt{cons}(a,l) & \to & h\ a\ (\texttt{listrec}\ g\ h\ l) \\
\texttt{listcase}\ h_{nil}\ h_{cons}\ \texttt{nil} & \to & h_{nil} \\
\texttt{listcase}\ h_{nil}\ h_{cons}\ \texttt{cons}(a,l) & \to & h_{cons}(a,l)
\end{array}$$

3.2 Interprétations dans les cas faciles

On peut montrer qu'une partie de SLR est interprétable et calculable en temps polynomial. On utilise pour cela les interprétations suivantes :

$$\begin{array}{rcl}
[x] & = & 1 \\
[(e_1 e_2)] & = & [e_1] + [e_2] \\
[\texttt{abs}_{e,x}] & = & \lambda x_1 ... \lambda x_n [\langle e \rangle] + [x_1] + ... + [x_n] \\
[\texttt{pair}] & = & \lambda x.\lambda y.x + y \\
[\texttt{proj1}] & = & \lambda x.x + 1 \\
[\texttt{proj2}] & = & \lambda x.x + 1 \\
[\texttt{lpair}] & = & \lambda x.\lambda y.x + y \\
[\texttt{let}] & = & \lambda x.\lambda y.\lambda z.\lambda w.x + y + z + w + 1 \\
[\texttt{zero}] & = & 1 \\
[\texttt{S}_0] & = & \lambda x.x + 1 \\
[\texttt{S}_1] & = & \lambda x.x + 1 \\
[\texttt{recnat}] & = & \lambda \phi.\lambda x.\lambda y.x + y.\phi(y,y) \\
[\texttt{casenat}] & = & \lambda x.\lambda \phi.\lambda \psi.\lambda y.x + y + \phi(y) + \psi(y) \\
[\texttt{nil}] & = & 1 \\
[\texttt{cons}] & = & \lambda x.\lambda y.x + y + 1 \\
[\texttt{listrec}] & = & \lambda x.\lambda \psi.\lambda y.(x + y + 1)\psi(x + y + 1) \\
[\texttt{listcase}] & = & \lambda y.\lambda \psi.\lambda x.y + \psi(x) + 1
\end{array}$$

La majorité des cas concerne des règles de réduction ne faisant pas apparaître de récurrence, ce qui permet de traiter ces cas rapidement. Il reste

deux cas particuliers, celui des récurrences sur les entiers et des récurrences sur les listes. On traite le cas des récurrence sur les entiers.

Démonstration.

Remarque. *On rappelle que le typage de* $\mathrm{rec}^{\mathrm{N}}$ *est* $\forall X.X \multimap \square(\square\mathrm{N} \to X \multimap X) \to \square\mathrm{N} \to X$ *; on conserve la même idée de typage dans le STTRS dans lequel on traduit, ce qui veut dire que la fonction* h*, à laquelle on fait la récurrence, est linéaire (et donc croissante en tous ses arguments)*

Remarque. *Pour une fonction linéaire* h*, de taille* $\mathcal{S}$*, à deux arguments entiers (dans les interprétations), on a :*

$$\max(\max(\mathcal{S}), n, m) \leq \phi(n, m) \leq n + m + \sum \mathcal{S}$$

On se rappelle que l'on travaille sur la règle de réécriture

$$\texttt{recnat}\ h\ g\ \mathtt{S_i}(x) \quad \to \quad h\ \mathtt{S_i}(x)\ (\texttt{recnat}\ h\ g\ x)$$

dont le membre de gauche a comme interprétation (une fois réduite)

$$G = [\texttt{recnat}\ h\ g\ \mathtt{S_i}(x)] = [g] + ([x] + 1)([h](x + 1, x + 1))$$

et le membre de droite

$$D = [h\ \mathtt{S_i}(x)\ (\texttt{recnat}\ h\ g\ x)] = [h]([x] + 1, [g] + x.[h](x, x))$$

et par les remarques précédentes

$$D \leq [x] + 1 + [g] + x.[h](x, x) \leq G$$

□

Le calcul est similaire pour la récurrence sur les listes ; il est ainsi possible de montrer que le sous-ensemble de SLR se limitant aux listes, naturels, à leurs récurrences et leurs tris par cas est calculable en temps PTIME à l'aide des STTRS.

3.3 Cas de l'application

Pour traiter le cas de l'application dans $\mathrm{SLR}^{\mathrm{BC}}$, on s'était servi du fait que l'ensemble des interprétations dans la traduction pouvait s'écrire sous forme max-polynomiale avec un polynôme ne portant que sur les variables sûres. Ce n'est plus le cas lorsque l'on rajoute les éléments de SLR qui

n'étaient pas encore présents, et encore moins si l'on rajoutait les arbres aux constructions que l'on a traduit dans cette section.

Après avoir observé le fonctionnement des interprétations pour $\mathrm{SLR}^{\mathrm{BC}}$, c'est-à-dire que l'on mettait l'ensemble des variables sûres dans un polynôme et que l'on rajoutait à cela le maximum des variables non-sûres, il est fort probable que, si on arrive à retrouer une structure équivalente, les variables non-linéaires soient quelque part entre les deux, possiblement une somme de ces variables non-linéaires, ou simplement, malgré que l'on ait étendu les constructions pour couvrir l'ensemble de SLR, présentes dans l'opérateur maximum au même titre que les variables linéaires.

A Règles de typage de SLR

On commence par présenter les quelques règles de sous-typages, où (modal,non-linéaire <: non-modal,non-linéaire <: non-modal, linéaire, avant de présenter le typage de SLR.

$$\frac{A' <: A \quad B' <: B \quad a' <: a}{A \xrightarrow{a} B <: A' \xrightarrow{a'} B'}$$

$$\frac{A' <: A \quad B' <: B}{A \times B <: A' \times B'}$$

$$\frac{A' <: A \quad B' <: B}{A \otimes B <: A' \otimes B'}$$

$$\frac{A <: B}{\forall X.A <: \forall X.B}$$

$$\frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \quad \text{(T-Var)}$$

$$\frac{\Gamma \vdash e : A \quad A <: B}{\Gamma \vdash e : B} \quad \text{(T-Sub)}$$

$$\frac{\Gamma, x^a : A \vdash e : B}{\Gamma \vdash \lambda x : A.e : A \xrightarrow{a} B} \quad \text{(T-Arr-I)}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : A \xrightarrow{a} B \quad \Gamma, \Delta_2 \vdash e_2 : A \quad \Gamma \text{ non-linéaire} \quad x^{a'} : X \in \Gamma, \Delta_2 \Rightarrow a' \leq a}{\Gamma, \Delta_1, \Delta_2 \vdash (e_1 e_2) : B} \quad \text{(T-Arr-E)}$$

$$\frac{\Gamma \vdash e : A \quad X \text{ n'est pas libre dans } \Gamma}{\Gamma \vdash \Lambda X.e : \forall X.A} \quad \text{(T-All-I)}$$

$$\frac{\Gamma \vdash e : \forall X.A \quad B \text{ est sûre}}{\Gamma \vdash e[B] : A[B/X]} \quad \text{(T-All-E)}$$

$$\frac{\Gamma \vdash e_1 : A_1 \quad \Gamma \vdash e_2 : A_2}{\Gamma \vdash \langle e_1, e_2 \rangle : A_1 \times A_2} \quad \text{(T-Prod-I)}$$

$$\frac{\Gamma \vdash e : A_1 \times A_2 \quad i \in \{1, 2\}}{\Gamma \vdash e.i : A_i} \quad \text{(T-Prod-E)}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : A_1 \quad \Gamma, \Delta_2 \vdash e_2 : A_2 \quad \Gamma \text{ non-linéaire}}{\Gamma, \Delta_1, \Delta_2 \vdash e_1 \otimes e_2 : A_1 \otimes A_2} \quad \text{(T-Tens-I)}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : A_1 \otimes A_2 \quad \Gamma, \Delta_2, x^a : A_1, y^a : A_2 \vdash e_2 : B \quad \Gamma \text{ non-linéaire }, a = (\mathrm{l}, \mathrm{nm})}{\Gamma, \Delta_1, \Delta_2 \vdash \text{let } e_1 = x \otimes y \text{ in } e_2 : B} \quad \text{(T-Tens-E)}$$

$$\frac{c : A}{\Gamma \vdash c : A} \quad \text{(T-Cons)}$$

Références

[1] Patrick Baillot and Ugo Dal Lago. Higher-order Interpretations and Program Complexity (Long Version). Technical report, February 2012. 21 pages.

[2] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2 :97–110, 1992.