Programmation fonctionnelle et systèmes de types (MPRI 2-4-2)
Examen 2008-2009

Giuseppe Castagna, François Pottier et Didier Rémy

Durée 3h

1 Calcul d'objets

On s'intéresse au ς-calcul, un calcul qui a pour vocation de modéliser les constructions élémentaires des langages orientés objets. Les expressions du ς-calcul sont définies par la grammaire:

$$a \quad ::= \quad \{\ell_i = \varsigma(x)\, a_i^{i\in I}\} \mid a.\ell \mid a.\ell \Leftarrow \varsigma(x)\, a$$

Ces trois constructions syntaxiques représentent respectivement la construction d'un objet, donné par la liste de ses méthodes; l'appel de méthode; et la re-définition de méthode. La construction $\varsigma(x)\, a$ lie la variable x dans l'expression a. Informellement, cette variable représente l'objet lui-même ("self"). Les méthodes n'attendent aucun argument hormis l'objet lui-même.

La sémantique du ς-calcul est donnée par les règles de réduction suivantes, où j est dans I,

$$\begin{array}{rcll} \{\ell_i = \varsigma(x)\, a_i^{i\in I}\}.\ell_j & \rightsquigarrow & a_j[x \leftarrow \{\ell_i = \varsigma(x)\, a_i^{i\in I}\}] & \text{R-Send} \\ \{\ell_i = \varsigma(x)\, a_i\}.\ell_j \Leftarrow \varsigma(x)\, a' & \rightsquigarrow & \{\ell_i = \varsigma(x)\, a_i^{i\in I\setminus\{j\}}, \ell_j = \varsigma(x)\, a'\} & \text{R-Over} \\ E[a] \rightsquigarrow E[a'] & \text{if} & a \rightsquigarrow a' & \text{R-Context} \end{array}$$

Les valeurs v et les contextes de réduction sont définis par:

$$\begin{array}{rcl} v & ::= & \{\ell_i = \varsigma(x)\, a_i^{i\in I}\} \\ E & ::= & [] \mid E.\ell \mid E.\ell \Leftarrow \varsigma(x)\, a \end{array}$$

Pour alléger les notations, on note a au lieu de $\varsigma(x)\, a$ si x n'est pas libre dans a. On définit les expressions suivantes:

$$\begin{array}{rcl} a_0 & \triangleq & \{\ell = \varsigma(x)\, x.\ell\}.\ell \\ a_1 & \triangleq & \{\ell_0 = a_0; \ell_1 = \varsigma(x)\, (x.\ell_0 \Leftarrow \{\}).\ell_0\}.\ell_1 \end{array}$$

Question 1 *Donner les séquences de réduction pour les expressions a_0 et a_1. Donner un exemple de programme bloqué.* □

On munit le ς-calcul d'un système de types simples. Les types et environnements de typage sont les suivants:

$$\begin{array}{lcll} \tau & ::= & \alpha \mid \{\ell_i = \tau_i^{\,i\in I}\} & \text{types} \\ \Gamma & ::= & \emptyset \mid \Gamma, x:\tau & \text{environnements} \end{array}$$

On définit le jugement $\Gamma \vdash a : \tau$ comme la plus petite relation satisfaisant les règles suivantes:

$$\textsc{Object}\ \frac{\tau = \{\ell_i : \tau_i^{\,i\in I}\} \qquad \forall i \in I \quad \Gamma, x:\tau \vdash a_i : \tau_i}{\Gamma \vdash \{\ell_i = \varsigma(x)\, a_i^{\,i\in I}\} : \tau} \qquad \textsc{Send}\ \frac{\Gamma \vdash a : \{\ell_i : \tau_i^{\,i\in I}\} \qquad j \in I}{\Gamma \vdash a.\ell_j : \tau_j}$$

$$\textsc{Over}\ \frac{\tau = \{\ell_i : \tau_i^{\,i\in I}\} \qquad \Gamma \vdash a : \tau \qquad \Gamma, x:\tau \vdash a' : \tau_j \qquad j \in I}{\Gamma \vdash a.\ell_j \Leftarrow \varsigma(x)\, a' : \tau} \qquad \textsc{Var}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

Question 2 *Donner un exemple d'expression mal typée (sans justification). Donner les types et les dérivations de typage pour les expressions a_0 et a_1.* □

On admet le lemme de substitution: *Si $\Gamma, x:\tau', \Gamma' \vdash a : \tau$ et $\Gamma \vdash a' : \tau'$, alors $\Gamma, \Gamma' \vdash a[x \leftarrow a'] : \tau$.* On écrit $a \subset a'$ si pour tout environnement Γ et tout type τ tels que $\Gamma \vdash a : \tau$, on a aussi $\Gamma \vdash a' : \tau$.

Question 3 *Montrer la propriété d'auto-réduction,* i.e. *si $a \rightsquigarrow a'$, alors $a \subset a'$. (On décrira clairement le schéma de la preuve, mais on pourra ne pas détailler le cas de l'override, similaire à l'appel de méthode.)* □

On ne montrera pas la propriété de progression.

On dit qu'on peut coder un langage L_1 dans L_2 si on peut fournir une fonction de traduction totale $[\![\cdot]\!]$ préservant la sémantique.

Question 4 *Peut-on coder le ς-calcul simplement typé dans le λ-calcul simplement typé? (On justifiera brièvement la réponse.)* □

Inversement, on définit le codage suivant où et val et arg sont deux étiquettes arbitraires distinctes fixées. On note M les expressions du λ-calcul.

$$[\![x]\!] \triangleq x \qquad [\![\lambda(x)\, M]\!] \triangleq \{\mathsf{val} = \varsigma(x)\, ([\![M]\!][x \leftarrow x.\mathsf{arg}]), \mathsf{arg} = \bot\}$$

$$[\![M_1\ (M_2)]\!] \triangleq ([\![M_1]\!].\mathsf{arg} \Leftarrow [\![M_2]\!]).\mathsf{val}$$

où $\bot$ est un terme du ς-calcul qui admet tous les types.

Question 5 *Quel terme peut-on prendre pour $\bot$?* □

Question 6 *Vers quelle valeur se réduit l'expression a égale à $[\![(\lambda(x)\, x)\ (y)]\!]$?* □

Question 7 *Quelle stratégie d'évaluation du λ-calcul ce codage vers le ς-calcul implémente-t-il? (On justifiera brièvement la réponse.)* □

On souhaite à présent montrer que le codage présenté ci-dessus préserve le typage.

Question 8 *Donner la traduction des types du λ-calcul (simplement typé) vers le ς-calcul.* □

Pour pouvoir définir comment un jugement de typage du λ-calcul est traduit en un jugement de typage du ς-calcul, nous devons généraliser la notion d'environnement de typage du λ-calcul. En effet, nous devons distinguer, parmi les variables, les variables dites libres de celles liées par λ. Pour les premières, nous introduisons dans l'environnement des liaisons de la forme $x : \tau$, comme d'habitude. Pour les secondes, nous introduisons des liaisons de la forme $x : \tau \triangleright \tau'$. Une telle liaison est censée indiquer que la variable x a été introduite par une λ-abstraction de type $\tau \to \tau'$.

Sur le modèle du jugement de typage du λ-calcul, nous construisons alors un jugement $\Gamma \vdash M : \tau \triangleright a$, qui doit impliquer:

1. d'une part, $\Gamma \vdash M : \tau$;
2. d'autre part, $[\![\Gamma]\!] \vdash a : [\![\tau]\!]$;
3. enfin, a est égal à $[\![M]\!]$, dans lequel x est remplacé par $x.\mathsf{arg}$, pour toute variable x telle qu'une liaison de la forme $x : \tau \triangleright \tau'$ apparaît dans Γ.

Question 9 *Donner la traduction des environnements de typage, en tenant compte du fait que ceux-ci contiennent deux sortes de liaison, comme indiqué ci-dessus.* □

Question 10 *Donner les règles qui définissent le jugement $\Gamma \vdash M : \tau \triangleright a$.* □

Question 11 *Démontrer le point 2 ci-dessus, à savoir le fait que le terme a produit par la traduction est bien typé. On se contentera de traiter le cas de la λ-abstraction.* □

Question 12 *Pourquoi, bien que bien typé, ce codage n'est-il pas entièrement satisfaisant? Que pourrait-on attendre du typage?* □

2 Sous-typage

Nous considérons un langage simple réduit à un calcul d'enregistrements. Ses types sont construits inductivement à partir d'un ensemble de types atomiques (par exemple, `int`, `real`, `bool`,...) en appliquant le constructeur de type enregistrement. Cependant, pour simplifier, on ne considérera formellement qu'un seul type atomique $\diamond$ avec une constante c de ce type:

$$\begin{array}{rcll} T & ::= & \diamond \mid \langle\!\langle \ell : T \ldots \ell : T \rangle\!\rangle & \text{Types} \\ e & ::= & c \mid \langle \ell = e, \ldots \ell = e \rangle \mid e.\ell & \text{Expressions} \end{array}$$

avec la sémantique usuelle pour l'accès aux enregistrements (la stratégie d'évaluation n'est.pas importante).

Nous ajoutons à ce calcul de base les opérations sur les ensembles telles qu'elles sont définies dans certains langages de programmation: le singleton, l'union de deux ensembles, la différence de deux ensembles et l'itération sur un ensemble:

$$\begin{array}{rcll} T & ::= & \ldots \mid \texttt{set}\ T & \text{Types} \\ e & ::= & \ldots \mid x \mid \texttt{set}\ e \mid e + e \mid e - e \mid \texttt{for}(x\ \texttt{in}\ e)\{e\} & \text{Expressions} \end{array}$$

L'itération sur un ensemble donne comme résultat un autre ensemble. Ainsi, si e dénote l'ensemble $\{a_1, \ldots, a_n\}$ alors $\texttt{for}(x\ \texttt{in}\ e)\{e'\}$ dénote l'ensemble $\{e'[x := a_1], \ldots, e'[x := a_n]\}$. La sémantique des autres opérations est standard. Les opérations d'union et de soustraction ne doivent bien entendu opérer qu'entre des ensembles dont les éléments sont *compatibles*. Par exemple, l'élément c n'est pas compatible avec un enregistrement, et un ensemble n'est pas compatible avec un enregistrement. Deux enregistrements sont toujours compatibles si, par sous-typage on ignore les étiquettes sur lesquelles leurs projections ne sont pas compatibles.

Question 13 *Le sous-typage est défini comme la plus petite relation $\leq$ réflexive et transitive entre les types satisfaisant un ensemble de règles.*

Donner les deux règles de sous-typage nécessaires pour définir cette relation (l'une pour pour $\langle\langle\rangle\rangle$, et pour `set`*()).* □

Question 14 *Le typage est défini par un jugement de la forme $\Gamma \vdash e : T$ comme la plus petite relation satisfaisant un ensemble de règles d'inférence. Parmi ces règles on donne la règle de sous-typage*

$$\frac{\Gamma \vdash e : T \qquad T \leq T'}{\Gamma \vdash e : T'}$$

Écrire les trois règles de typage du langage original (constante, enregistrement, sélection de champ) ainsi que les cinq autres règles de typage des expressions introduites (respectivement, variable, singleton, union, différence et itération). (On évitera une redondance, i.e. *d'inclure des hypothèses de sous-typage inutiles dans les prémisses qui pourraient être dérivables par la règle de sous-typage).* □

Question 15 *Définir par induction sur tous les types un opérateur* $\mathbf{sup}(S, T)$ *qui dénote la borne supérieure de deux types S et T.*

□

Question 16 *Écrire les règles de* l'algorithme *de typage en présence de sous-typage, en utilisant l'opérateur* **sup** *défini à la question précédente.*

□

Question 17 *Quelle modification faut-il apporter à la question 15 si le langage qu'on étendait par les opérations sur les ensembles avait, en plus des enregistrements aussi les types (et, bien sûr, les termes) des fonctions ?*

□

Question 18 *La réponse à la question 13 comporte deux règles de sous-typage pour les constructeurs de types, plus les deux règles pour la réflexivité et la transitivité de tous les types. Le système ainsi obtenu n'est pas algorithmique. Écrire la/les règle(s) qu'il faut utiliser en remplacement de la transitivité et de la réflexivité afin d'obtenir un système algorithmique équivalent à celui de la question 13. Justifier la réponse.*

□

Question 19 ***(Facultative)*** *Le système donné ne satisfait pas la propriété de progression. Donner une expression qui montre cela (suggestion: définir les valeurs du calcul et utiliser l'opérateur différence).*

□