

Calcul d'objets

Second examen partiel, MPRI 2-4

2021/01/06 — durée: 1h45

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

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

Outre les variables x , on a donc trois constructions syntaxiques, qui représentent respectivement la construction d'un objet, donné par la liste de ses méthodes (I est un ensemble d'indices, possiblement vide, typiquement un ensemble d'entiers de 1 à n); l'appel de la méthode ℓ de l'objet a ; et la redéfinition de la méthode ℓ de l'objet a . La construction $\varsigma(x) a$ lie la variable x dans l'expression a (et les expressions sont donc considérées égales par renommage des variables liées). Informellement, la variable x 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 ,

$$\begin{aligned} \{(\ell_i = \varsigma(x) a_i)^{i \in I}\}.\ell_j &\rightsquigarrow a_j[x \leftarrow \{(\ell_i = \varsigma(x) a_i)^{i \in I}\}] && \text{R-SEND} \\ \{(\ell_i = \varsigma(x) a_i)^{i \in I}\}.\ell_j \Leftarrow \varsigma(x) a' &\rightsquigarrow \{(\ell_i = \varsigma(x) a_i)^{i \in I \setminus \{j\}}, \ell_j = \varsigma(x) a'\} && \text{R-OVER} \\ E[a] &\rightsquigarrow E[a'] \quad \text{if} \quad a \rightsquigarrow a' && \text{R-CONTEXT} \end{aligned}$$

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

$$\begin{aligned} v &::= \{(\ell_i = \varsigma(x) a_i)^{i \in I}\} \\ E &::= [].\ell \quad | \quad [].\ell \Leftarrow \varsigma(x) a \end{aligned}$$

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 :

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

Question 1

- a) Donner les séquences de réduction pour les expressions a_0 et a_1 .
- b) Donner un exemple de programme bloqué.

Réponse \square

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

sont les suivants :

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

Notez l'absence de variables de types. Nous ne manipulerons donc que des types clos.

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

$$\begin{array}{c} \text{OBJECT} \\ \frac{\tau = \{(\ell_i : \tau_i)^{i \in I}\} \quad \forall i \in I \quad \Gamma, x : \tau \vdash a_i : \tau_i}{\Gamma \vdash \{(\ell_i = \varsigma(x) a_i)^{i \in I}\} : \tau} \end{array} \quad \begin{array}{c} \text{SEND} \\ \frac{\Gamma \vdash a : \{(\ell_i : \tau_i)^{i \in I}\} \quad j \in I}{\Gamma \vdash a.\ell_j : \tau_j} \end{array}$$

$$\begin{array}{c} \text{OVER} \\ \frac{\tau = \{(\ell_i : \tau_i)^{i \in I}\} \quad \Gamma \vdash a : \tau \quad \Gamma, x : \tau \vdash a' : \tau_j \quad j \in I}{\Gamma \vdash a.\ell_j \Leftarrow \varsigma(x) a' : \tau} \end{array} \quad \begin{array}{c} \text{VAR} \\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \end{array}$$

Question 2

a) Donner un exemple d'expression mal typée (sans justification).

b) Donner les types et les dérivations de typage pour les expressions a_0 et a_1 . (On pourra utiliser le fait que $\vdash a : \tau$ implique $\Gamma \vdash a : \tau$.) *Réponse* \square

On admet le lemme de substitution, dont l'énoncé est le suivant : *si $\Gamma, x : \tau', \Gamma' \vdash a : \tau$ et $\Gamma \vdash a' : \tau'$, alors $\Gamma, \Gamma' \vdash a[x \leftarrow a'] : \tau$.*

Question 3

On veut montrer que la règle de réduction R-SEND préserve le typage. a) Énoncez la propriété.

b) Montrez-la. *Réponse* \square

On dit qu'une traduction d'un langage L_1 dans un langage L_2 est une bisimulation si on peut fournir une fonction de traduction totale $\llbracket \cdot \rrbracket$ préservant la sémantique, c'est-à-dire :

- les valeurs de L_1 se traduisent en des valeurs de L_2
- Si $a \rightsquigarrow a'$ dans L_1 alors $\llbracket a \rrbracket \rightsquigarrow^+ \llbracket a' \rrbracket$ dans L_2 .
- Si $\llbracket a \rrbracket \rightsquigarrow e$ dans L_2 alors $a \rightsquigarrow a'$ dans L_1 et $e \rightsquigarrow^* \llbracket a' \rrbracket$.

Question 4

Peut-on donner une traduction du ς -calcul simplement typé dans le λ -calcul simplement typé avec juste une constante unité $()$ de type **unit** qui soit une bisimulation ? (On justifiera brièvement la réponse.) *Réponse* \square

Inversement, on définit la traduction suivante du λ -calcul dans le ς -calcul, où **val** et **arg** sont deux étiquettes arbitraires distinctes fixées. On note e les expressions du λ -calcul implicite-ment et simplement typé, étendu avec une seule constante unité $()$ de type **unit**.

$$\begin{array}{lll} \llbracket () \rrbracket \triangleq \{\} & \llbracket x \rrbracket \triangleq x.\mathbf{arg} & \llbracket \lambda x. e \rrbracket \triangleq \{\mathbf{arg} = \perp, \mathbf{val} = \varsigma(x) \llbracket e \rrbracket\} \\ & & \llbracket e_1 e_2 \rrbracket \triangleq (\llbracket e_1 \rrbracket.\mathbf{arg} \Leftarrow \llbracket e_2 \rrbracket).\mathbf{val} \end{array}$$

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

Question 5

Quel terme peut-on prendre pour \perp ?

Réponse \square

Question 6

On note e_1 le terme $(\lambda x. x) ()$ du λ -calcul.

a) Donnez la réduction, pas à pas, de $\llbracket e_1 \rrbracket$ jusqu'à obtenir une valeur v du ς -calcul.

b) Combien la réduction contient-elle de pas élémentaires ?

c) Que suggère cet exemple en lien avec la définition d'une bisimulation ?

Réponse \square

Question 7

Répondre aux trois mêmes questions que l'exercice précédent pour le terme e_2 égal à $(\lambda x. \lambda z. x) ()$.

Réponse \square

Question 8

Quelle stratégie d'évaluation du λ -calcul cette traduction vers le ς -calcul implémente-t-elle ?
(On justifiera brièvement la réponse.)

Réponse \square

On souhaite à présent montrer que la traduction présentée ci-dessus préserve le typage. On notera σ les types du λ -calcul simplement typé, définis par :

$$\sigma ::= \text{unit} \mid \sigma \rightarrow \sigma$$

Question 9

Donner la traduction des types du λ -calcul vers les types du ς -calcul.

Réponse \square

On note Δ les environnements de typage du λ -calcul et on traitera la constante $()$ de façon primitive avec l'axiome $\text{UNIT} : \Delta \vdash () : \text{unit}$. Les autres règles de typage sont celles du λ -calcul simplement typé. Nous voulons montrer la préservation du typage : Si $\vdash e : \sigma$ alors $\vdash \llbracket e \rrbracket : \llbracket \sigma \rrbracket$. Pour cela, il nous faut renforcer la propriété avec des termes ouverts. Cependant la traduction $\llbracket \Delta \rrbracket$ n'est pas uniquement un environnement de typage, mais un ensemble d'environnements de typages (pour une raison à découvrir). On pourra alors montrer

Si $\Delta \vdash e : \sigma$ alors $\Gamma \vdash \llbracket e \rrbracket : \llbracket \sigma \rrbracket$ pour tout Γ dans $\llbracket \Delta \rrbracket$.

Question 10

a) Donner la définition de $\llbracket \Delta \rrbracket$ qui permette de montrer la propriété ci-dessus.

b) Montrer la propriété ci-dessus. (On donnera clairement le schéma de preuve, mais on pourra ne traiter que les cas des variables et des abstractions.)

Réponse \square

Question 11

a) Pourquoi, bien que bien typé, cette traduction n'est-elle pas entièrement satisfaisante ?

b) Que pourrait-on attendre du typage ?

Réponse \square

1 Solutions

Question 1

a)

$$\begin{array}{c} a_0 \xrightarrow{\text{Send}} a_0 \xrightarrow{\text{Send}} \dots \\ a_1 \xrightarrow{\text{Send}} (a_1.\ell_0 \Leftarrow \{\}) . \ell_0 \xrightarrow{\text{Over}} \{\ell_0 = \{\}, \ell_1 = \varsigma(x) (x.\ell_0 \Leftarrow \{\}) . \ell_0\} . \ell_0 \xrightarrow{\text{Send}} \{\} \end{array}$$

b) L'expression $\{\}. \ell$ ne se réduit pas mais ce n'est pas une valeur. \square

Question 2

a) $\{\}. \ell$ est mal typé.

b) On peut dériver $\vdash a_0 : \tau_0$ pour τ_0 quelconque :

$$\frac{\text{VAR} \quad x : \{\ell : \tau_0\} \vdash x : \{\ell : \tau_0\}}{x : \{\ell : \tau_0\} \vdash x.\ell : \tau_0} \text{SEND} \quad \frac{x : \{\ell : \tau_0\} \vdash x.\ell : \tau_0}{\vdash \{\ell = \varsigma(x) x.\ell\} : \tau_0} \text{OBJECT} \quad \frac{\vdash \{\ell = \varsigma(x) x.\ell\} : \tau_0}{\vdash \{\ell = \varsigma(x) x.\ell\} . \ell : \tau_0} \text{SEND}$$

En particulier, nous pouvons choisir $\tau_0 \triangleq \{\}$. Notons τ_1 le type $\{\ell_0 : \tau_0, \ell_1 : \tau_0\}$. On peut dériver $\vdash a_1 : \tau_0$, c'est-à-dire $\vdash a_1 : \{\}$ ainsi :

$$\frac{\text{VAR} \quad x : \tau_1 \vdash x : \tau_1 \quad \text{OBJECT} \quad x : \tau_1, z : \tau_1 \vdash \{\} : \tau_0}{x : \tau_1 \vdash x.\ell_0 \Leftarrow \varsigma(z) \{\} : \tau_1} \text{OVER} \quad \frac{x : \tau_1 \vdash a_0 : \tau_0 \text{ (1)} \quad x : \tau_1 \vdash (x.\ell_0 \Leftarrow \varsigma(z) \{\}) . \ell_0 : \tau_0}{x : \tau_1 \vdash \{\ell_0 = a_0; \ell_1 = \varsigma(x) (x.\ell_0 \Leftarrow \varsigma(z) \{\}) . \ell_0\} : \tau_1} \text{OBJECT} \quad \frac{x : \tau_1 \vdash \{\ell_0 = a_0; \ell_1 = \varsigma(x) (x.\ell_0 \Leftarrow \varsigma(z) \{\}) . \ell_0\} : \tau_1}{a_1 : \tau_0} \text{SEND}$$

Le jugement (1) se déduit de la dérivation précédente de $\vdash a_0 : \tau_0$.

Question 3

a) **Fait** : Pour tout v de la forme $\{(\ell_i = \varsigma(x) a_i)^{i \in I}\}$ (1) et $j \in I$ (2), si $\Gamma \vdash v.\ell_j : \tau$ (3), alors $\Gamma \vdash a_j[x \leftarrow v] : \tau$ (4).

b) **Preuve** : Par inversion du typage, une dérivation de (3) est de la forme :

$$\text{OBJECT} \frac{(\Gamma, x : \sigma \vdash a_i : \tau_i)^{i \in I} \text{ (5)}}{\Gamma \vdash v : \sigma \text{ (6)}} \quad \text{SEND} \frac{\Gamma \vdash v : \sigma \text{ (6)}}{\Gamma \vdash v.\ell_j : \tau_j} \quad (2)$$

pour σ égal à $\{(\ell_i : \tau_i)^{i \in I}\}$ et τ_j égal à τ . La conclusion (4) est alors directement impliquée par le lemme de substitution appliqué à (6) et la prémisse j de la propriété (5).

Question 4

Non, car a_0 diverge, ce qui implique que sa traduction diverge également (par la deuxième clause), ce qui n'est pas possible dans le λ -calcul simplement typé (en l'absence de combinateur de point fixe).

Question 5

a_0 .

Question 6

$$\begin{aligned}
\text{a)} \quad \llbracket e_1 \rrbracket &\triangleq \llbracket (\lambda x. x) () \rrbracket \\
&= (\{\text{arg} = \perp, \text{val} = \varsigma(x) x.\text{arg}\}.\text{arg} \Leftarrow \{\}).\text{val} \\
&\rightsquigarrow \{\text{arg} = \{\}, \text{val} = \varsigma(x) x.\text{arg}\}.\text{val} && \text{OVER} \\
&\triangleq w.\text{val} \\
&\rightsquigarrow x.\text{arg}[x \leftarrow w] && \text{SEND} \\
&= w.\text{arg} \\
&\rightsquigarrow \{\} && \text{SEND} \\
&= \llbracket () \rrbracket
\end{aligned}$$

b) 3.

c) Sur cet exemple $e_1 \rightsquigarrow^* w_1$ dans le λ -calcul et $\llbracket e_1 \rrbracket \rightsquigarrow^* \llbracket w_1 \rrbracket$ dans le ς -calcul, ce qui suggère que la traduction est une bisimulation.

Question 7

$$\begin{aligned}
\text{a)} \quad \llbracket e_2 \rrbracket &\triangleq \llbracket (\lambda x. \lambda z. x) () \rrbracket \\
&= (\{\text{arg} = \perp, \text{val} = \varsigma(x) \{\text{val} = \varsigma(z) x.\text{arg}\}\}.\text{arg} \Leftarrow \{\}).\text{val} \\
&\rightsquigarrow \{\text{arg} = \{\}, \text{val} = \varsigma(x) \{\text{val} = \varsigma(z) x.\text{arg}\}\}.\text{val} && \text{OVER} \\
&\triangleq w.\text{val} \\
&\rightsquigarrow \{\text{arg} = \perp, \text{val} = \varsigma(z) w.\text{arg}\} && \text{SEND} \\
&\neq \{\text{arg} = \perp, \text{val} = \varsigma(z) \{\}\} \\
&= \llbracket \lambda z. () \rrbracket
\end{aligned}$$

b) 2.

c) Sur cet exemple $e_2 \rightsquigarrow^* w_2$ dans le λ -calcul mais $\llbracket e_2 \rrbracket \rightsquigarrow^* v_2 \neq \llbracket w_2 \rrbracket$ dans le ς -calcul. Ce qui suggère que la traduction n'est pas une bisimulation au sens défini ci-dessus.

La différence se trouve au niveau du champ **val** : l'un est égal à $\varsigma(z) w.\text{arg}$, qui se réduira vers $\{\}$, tandis que l'autre est égal à $\varsigma(z) \{\}$ (qui se réduira aussi vers $\{\}$). Ces deux objets sont observationnellement équivalents, mais syntaxiquement différents.

Question 8

Ce codage correspond à une stratégie d'évaluation en appel par nom. En effet, l'argument d'une application est évalué non pas lors de l'application (le calcul est suspendu sous un ς), mais à chacune de ses utilisations.

Question 9

$$\llbracket \text{unit} \rrbracket = \{\} \qquad \llbracket \sigma_1 \rightarrow \sigma_2 \rrbracket = \{\text{arg} : \llbracket \sigma_1 \rrbracket, \text{val} : \llbracket \sigma_2 \rrbracket\}$$

Question 10

a) $\llbracket \Delta \rrbracket$ est défini par induction

$$\llbracket \emptyset \rrbracket = \{\emptyset\} \qquad \llbracket \Delta, x : \sigma \rrbracket = \{\Gamma, x : \{\text{arg} : \llbracket \sigma \rrbracket; \text{val} : \tau\} \mid \Gamma \in \Delta \text{ et } \tau \text{ quelconque}\}$$

b) Nous montrons que si $\Delta \vdash e : \sigma$ (1) alors $\Gamma \vdash \llbracket e \rrbracket : \llbracket \sigma \rrbracket$ (2) pour tout Γ dans $\llbracket \Delta \rrbracket$ par induction sur la dérivation de typage de (1).

Cas Var (1) devient $\Delta \vdash x : \sigma$. Par inversion du typage, nous avons $x : \sigma \in \Delta$. Quel que soit Γ dans $\llbracket \Delta \rrbracket$, nous avons donc $x : \{\text{val} : \llbracket \sigma \rrbracket; \text{arg} : \sigma'\}$ dans Γ ce qui implique $\Gamma \vdash x : \{\text{val} : \llbracket \sigma \rrbracket; \text{arg} : \sigma'\}$ par la règle VAR et la conclusion (2) qui s'écrit $\Gamma \vdash x.\text{arg} : \llbracket \sigma \rrbracket$ est satisfaite par la règle SEND.

Cas Lam (1) devient $\Delta \vdash \lambda x. e_1 : \sigma$. Par inversion du typage, nous savons $\Delta, x : \sigma_0 \vdash e_1 : \sigma_1$ (3) où σ est égal à $\sigma_0 \rightarrow \sigma_1$. Remarquons que $\llbracket \sigma \rrbracket$ est égale à $\{\text{arg} : \llbracket \sigma_0 \rrbracket; \text{val} : \llbracket \sigma_1 \rrbracket\}$. Soit Γ dans $\llbracket \Delta \rrbracket$. Nous avons $\Gamma, x : \llbracket \sigma \rrbracket$ dans $\llbracket \Delta, x : \sigma_0 \rrbracket$. Par IH appliqué à (3), nous en déduisons $\Gamma, x : \llbracket \sigma \rrbracket \vdash \llbracket e_1 \rrbracket : \llbracket \sigma_1 \rrbracket$. Comme par ailleurs \perp a tous les types, nous en déduisons, par la règle OBJECT, $\Gamma \vdash \{\text{arg} = \perp; \text{val} = \llbracket e_1 \rrbracket\} : \llbracket \tau \rrbracket$ qui n'est autre que la conclusion (2).

Cas Unit et App omis.

Question 11

La traduction utilise une boucle (le terme \perp) pour coder un champ indéfini. Les expressions issues du codage n'accéderont jamais à ce champ avant qu'il soit redéfini, sinon le programme bouclerait. Ce bon comportement est garanti par la traduction, mais pas par le typage. En particulier, si on combine des expressions obtenues par traduction avec des expressions écrites directement, on pourrait déclencher une boucle qui en fait cacherait une erreur du genre "accès en dehors du domaine".

Il serait préférable que le typage permette de ne pas définir le champ **arg** d'une abstraction tout en garantissant que ce champ sera effectivement défini avant tout appel à la méthode **val** (qui peut déclencher à son tour un appel à la méthode **arg**). Mais il faudrait, pour cela, munir le ς -calcul d'un système de types plus complexe.