Programmation fonctionnelle et systèmes de types (MPRI 2-4-2)
Examen 2009-2010

Giuseppe Castagna, François Pottier et Didier Rémy

Durée 3h

1 Calcul d'objets: sous-typage et exceptions

On s'intéresse au $\mathbf{Ob}_{1<:}$-calcul, décrit dans le Chapitre 8 de [2] ainsi que dans [1], un calcul qui a pour vocation de modéliser les constructions élémentaires des langages orientés objets. Les expressions du $\mathbf{Ob}_{1<:}$-calcul sont définies par la grammaire:

$$\begin{array}{lcll} a & ::= & 0 \mid \mathsf{S}(a) \mid x \mid \{\ell_i : \tau_i = \varsigma(x)\, a_i{}^{i\in I}\} \mid a.\ell \mid a.\ell \Leftarrow \varsigma(x)\, a & \text{Expressions} \\ \tau & ::= & \mathtt{int} \mid \{\ell_i : \tau_i{}^{i\in I}\} & \text{Types} \end{array}$$

Les entiers sont construits inductivement à partir de 0 par l'application du constructeur S. Les trois dernières expressions représentent respectivement la construction d'un objet, donné par la liste de ses méthodes; l'appel de méthode; et la redéfinition 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. Le type spécifié après l'étiquette correspond au type de la méthode elle-même. Par exemple, un objet possédant 2 méthodes *val* et *next* de type `int` s'écrit en $\mathbf{Ob}_{1<:}$-calcul:

$$\{\mathit{val} : \mathtt{int} = \varsigma(x)\, 0\ ,\ \ \mathit{next} : \mathtt{int} = \varsigma(x)\, \mathsf{S}(x.\mathit{val})\}$$

La sémantique du $\mathbf{Ob}_{1<:}$-calcul est donnée par les règles de réduction suivantes, où j appartient à I,

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

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

$$\begin{array}{lcl} v & ::= & \{\ell_i = \varsigma(x)\, a_i{}^{i\in I}\} \mid 0 \mid \mathsf{S}(v) \\ E & ::= & [\,] \mid \mathsf{S}(E) \mid E.\ell \mid E.\ell \Leftarrow \varsigma(x)\, a \end{array}$$

Pour alléger les notations, on note a au lieu de $\varsigma(x)\, a$ si x n'est pas libre dans a. Il est alors possible de définir

$$(\{\mathit{val} : \mathtt{int} = 0\ ,\ \ \mathit{next} : \mathtt{int} = \varsigma(x)\, \mathsf{S}(x.\mathit{val})\}.\mathit{val} \Leftarrow 2).\mathit{next} \qquad (1)$$

qui se réduit en $\mathsf{S}(2)$, où 2 est du sucre syntaxique pour $\mathsf{S}(\mathsf{S}(0))$.

Question 1 *Donner tous les pas de la réduction du terme décrit ci-dessus (1).* □

Question 2 *Montrer qu'il peut y avoir des calculs infinis: donner un terme et un nombre suffisant de pas de réduction de ce terme qui montre que la réduction de ce terme peut ne pas terminer.* □

On veut définir un système de types simples pour le langage, c'est-à-dire définir un jugement de la forme $\Gamma \vdash a : \tau$ où de façon usuelle Γ est un environnement de type, c'est-à-dire une suite d'hypothèses $x_i : \tau_i$ où une variable ne peut apparaître plusieurs fois dans Γ. On note $\mathsf{dom}(\Gamma)$ le domaine de Γ défini comme l'ensemble des x_i's qui apparaissent dans Γ. Voici les principales règles de typage:

$$\textsc{Object}\ \frac{\tau = \{\ell_i : \tau_i\ ^{i\in I}\} \qquad x \notin \mathsf{dom}(\Gamma) \qquad \forall\, i \in I,\quad \Gamma, x : \tau \vdash a_i : \tau_i}{\Gamma \vdash \{\ell_i : \tau_i = \varsigma(x)\, a_i\ ^{i\in I}\} : \tau}$$

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

Question 3 *Expliquer (brièvement) pourquoi la condition $x \notin \mathsf{dom}(\Gamma)$ n'est pas restrictive.* □

Question 4 *Donner (toutes) les autres règles de typage.* □

On munit le $\mathbf{Ob}_{1<:}$-calcul d'un système de sous-typage *en largeur*, de manière que les objets avec plus de méthodes puissent être utilisés où des objets avec moins de méthodes sont attendus. À cet effet, on ajoute aux règles de l'exercice précédent la règle de subsumption (aussi appelée en cours règle de sous-typage):

$$\textsc{Subsumption}\ \frac{\Gamma \vdash a : \tau \qquad \tau <: \tau'}{\Gamma \vdash a : \tau'}$$

On n'autorisera pas le sous-typage *en profondeur* qui consisterait à permettre à un objet d'être utilisé là où un objet ayant les mêmes méthodes mais dont le type de chaque méthode serait un sous-type du type de la méthode dans l'objet attendu. On donne les deux règles de sous-typage suivantes:

$$\textsc{Refl}\ \frac{}{\tau <: \tau} \qquad\qquad \textsc{Trans}\ \frac{\tau_1 <: \tau_2 \quad \tau_2 <: \tau_3}{\tau_1 <: \tau_3}$$

Question 5 *a) Donner la ou les règles manquantes qui définissent la relation de sous-typage en largeur. b) Montrer que la règle de transitivité peut être éliminée, c'est-à-dire qu'en la retirant, on ne réduit pas l'ensemble des relations de sous-typage dérivables, mais seulement l'ensemble des dérivations possibles. c) Peut-on simplifier le règle* Refl*?* □

Question 6 *La règle de sous-typage en profondeur*

$$\textsc{Depth-Subtyping}\ \frac{\forall i \in I \quad \tau_i' <: \tau_i}{\{\ell_i : \tau_i'\ ^{i\in I}\} <: \{\ell_i : \tau_i\ ^{i\in I}\}}$$

n'est pas correcte.

Montrer qu'en utilisant cette règle et la construction $\Leftarrow$ on pourrait écrire un terme bien typé dont la réduction conduirait à une erreur (une expression irréductible qui n'est pas une valeur). On donnera la dérivation de typage du terme source et toutes les étapes de sa réduction jusqu'à une expression erronée. □

Question 7 *Dire lesquels parmi les termes suivants sont typables dans le système avec sous-typage en largeur. Pour les termes typables donner la dérivation de typage; pour les autres indiquer où se trouve l'erreur.*

1. $\{val : \texttt{int} = \varsigma(x)\, 0\ ,\ incr : \texttt{int} = \varsigma(x)\, (x.val \Leftarrow \varsigma(y)\, \mathsf{S}(x.val))\}$
2. $\{\ell : \texttt{int} = \varsigma(x)\, x.\ell\}$
3. $\{\ell : \{\ell_1 : \texttt{int}\} = \varsigma(x)\, \{\ell_1 : \texttt{int} = \varsigma(y)\, 0\, ,\, \ell_2 : \texttt{int} = \varsigma(y)\, x.\ell.\ell_1\}\}$

□

On cherche à écrire un algorithme de typage. Pour cela on voudrait un système de typage équivalent mais dirigé par la syntaxe (c'est-à-dire, qu'une dérivation de typage se termine par une règle qui soit uniquement déterminée par la forme de l'expression donnée) et satisfaisant la propriété de la sous-formule (tout ce qui apparait dans les premisses est déterminé par les consequences). Pour cela, il faut éliminer la règle de subsumption et à la place insérer une ou plusieurs prémisses de sous-typage dans certaines règles.

Question 8 *Donner une autre présentation du système de typage qui soit algorithmique (se limiter aux règles qui doivent être modifiées).* □

Nous voulons maintenant ajouter les exceptions au langage. Pour cela, nous étendons la grammaire avec les expressions suivantes:

$$a \quad ::= \quad \ldots \mid \texttt{raise}\ a \mid \texttt{try}\ a\ \texttt{with}\ x \to a$$

La sémantique de ces constructions est standard (similaire à celle de ML): de façon informelle, si a retourne une valeur alors l'expression $\texttt{try}\ a\ \texttt{with}\ x \to b$ retourne cette valeur; si a lève une exception, c'est-à-dire évalue une expression de la forme $\texttt{raise}\ v$, alors $\texttt{try}\ a\ \texttt{with}\ x \to b$ s'évalue comme le ferait l'expression $b[x \leftarrow v]$.

Question 9 *Décrire la sémantique opérationnelle des exceptions. Autrement dit, donner les règles de réductions et les contextes de réduction pour exprimer de façon formelle la sémantique décrite ci-dessus de façon informelle et incomplète. (On pourra utiliser un nouveau genre de contextes, dit de propagation, pour décrire de façon plus concise la propagation des exceptions au travers de toutes les autres constructions que* `try`*.)* □

On veut donner les règles de typage des deux nouvelles expressions, en supposant l'existence de la règle de sous-typage. Pour simplifier le typage nous imposons que l'expression spécifiée dans un raise ne puisse être que de type `int`. Puisqu'une exception peut apparaître dans n'importe quel contexte nous introduisons un nouveau type de base `exn` pour les exceptions, et nous étendons le sous-typage avec l'axiome Exc ci-dessous. On peut alors utiliser la règle de typage Raise ci-dessous:

$$\textsc{Exc}\ \frac{}{\Gamma \vdash \texttt{exn} <: \tau} \qquad \textsc{Raise}\ \frac{\Gamma \vdash a : \texttt{int}}{\Gamma \vdash \texttt{raise}\ a : \texttt{exn}}$$

Question 10 *Donner la règle de typage* Try *pour* `try`. □

Question 11 *Donner la version algorithmique des règles* Raise *et* Try*, qui peuvent donc utiliser si nécessaire le jugement de sous-typage dans leur prémisse (suggestion: pour le typage de try, on pourra utiliser un opérateur sur les types que l'on définira précisément).-* □

Un meilleur contrôle des exceptions Nous voulons modifier le système de types pour assurer que toute exception levée sera rattrapée par un try et n'atteindra jamais le "toplevel".

Pour cela, nous distinguons les expressions *pures* dont nous pouvons déterminer statiquement qu'aucune exception ne pourra s'échapper lors de son évaluation et caractérisées par un jugement de typage $\Gamma \vdash a : \tau$ des expressions *impures* d'où une exception peut éventuellement s'échapper et caractérisées par un jugement de typage auxiliaire $\Gamma \vdash^* a : \tau$. Un programme complet a ne sera typable que s'il est pur.

Dans un premier temps, nous pouvons récupérer *toutes* les règles pour les expressions pures, qui sont celles disponibles à l'issue de la question 5, *avant l'ajout des exceptions* (en particulier, nous abandonnons la règle Raise définie ci-dessus). Nous gardons le type `exn` et la règle de sous-typage Exn.

Comme une expression pure peut toujours être considérée comme une expression impure, nous ajoutons la règle Lift* ci-dessous (qui est une forme de sous-typage sur les jugements).

$$\frac{\Gamma \vdash a : \tau}{\Gamma \vdash^* a : \tau}\ \textsc{Lift}^*$$

Nous définissons également la règle Raise* ci-dessous pour typer les expressions impures et une règle Try pour typer les expressions pures selon le modèle ci-dessous.

$$\frac{\Gamma \vdash^* a : \texttt{int}}{\Gamma \vdash^* \texttt{raise}\ a : \texttt{exn}}\ \textsc{Raise}^* \qquad \frac{?}{\Gamma \vdash \texttt{try}\ a\ \texttt{with}\ x \to b : \tau}\ \textsc{Try}$$

Question 12 *Donner les prémisses de la règle* Try. □

On ajoutera également la règle suivante:

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

Question 13 *Donner un exemple qui justifie cette règle.* □

Question 14 *Cela suggère l'ajout de règles* Send* *et* Try* *dont les conclusions sont des jugements impurs et que l'on donnera sans justification.* □

Question 15 *Vérifier que* $\{\ell = \texttt{raise}\ 0\}$ *est une valeur. En déduire que la règle*

$$\frac{\tau = \{\ell_i : \tau_i\ ^{i \in I}\} \qquad \forall\ i \in I, \quad \Gamma, x : \tau \vdash^* a_i : \tau_i}{\Gamma \vdash^* \{\ell_i : \tau_i = \varsigma(x)\, a_i\ ^{i \in I}\} : \tau}\ \textsc{Object}^*$$

n'a pas de sens. On donnera au moins un terme qui est statiquement bien typé mais qui peut lever une exception non rattrapée, mais on pourra justifier brièvement davantage. □

Question 16 *Le système actuel n'accepte donc que des objets dont les méthodes sont pures. Qu'en déduisez vous sur la solution proposée.* □

Question 17 (facultative) *Que pourrait-on faire pour lever cette restriction?* □

References

[1] M. Abadi and L. Cardelli. A theory of primitive objects: untyped and first-order systems. In *Proc. of Theoretical Aspects of Computer Software*, LNCS. Springer, 1994.

[2] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer, 1996.

2 Corrigé

Question 2, page 1

$$\{\ell : \{\} = \varsigma(x)\, x.\ell\}.\ell$$

qui se réduit en lui même après un pas de réduction

Question 3, page 2

Parce que la variable x apparaissant dans les méthodes $\varsigma(x)\, a_i$ et $\varsigma(x)\, a'$ peut toujours être renommée pour satisfaire la condition.

Question 4, page 2

$$\textsc{Zero}\ \frac{}{\Gamma \vdash 0 : \texttt{int}} \qquad \textsc{Succ}\ \frac{\Gamma \vdash a : \texttt{int}}{\Gamma \vdash \mathsf{S}(a) : \texttt{int}} \qquad \textsc{Send}\ \frac{\Gamma \vdash a : \{\ell_i : \tau_i\ ^{i \in I}\} \qquad j \in I}{\Gamma \vdash a.\ell_j : \tau_j} \qquad \textsc{Var}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

Question 5, page 2

1) Il faut ajouter la règle pour les types d'objets:

$$\textsc{Width-Subtyping}\ \frac{}{\{\ell_i : \tau_i\ ^{i \in I \cup J}\} <: \{\ell_i : \tau_i\ ^{i \in I}\}}$$

2) La transitivité dérive immédiatement de la transitivité de l'inclusion ensembliste. 3) On peut restreindre la règle au cas où τ est `int`.

Question 6, page 2

$$(\{a_1 : \{\ell_1 : \texttt{int}, \ell_2 : \texttt{int}\} = \ldots\, , a_2 : \texttt{int} = \varsigma(x)\, x.a_1.\ell_2\}.a_1 \Leftarrow \{\ell_1 : \texttt{int} = 0\}).a_2$$

qui se réduit en deux pas à $\{\ell_1 : \texttt{int} = 0\}.\ell_2$. Cette expression est bloquée: elle ne se réduit pas et n'est pas une valeur.

Question 7, page 3

1. Mal typé car la méthode *incr* retourne un objet, pas un entier;
2. Bien typé (dérivation laissée comme exercice);

$$\textsc{Object}\ \frac{\textsc{Send}\ \frac{\textsc{Var}\ \frac{x : \{\ell : \texttt{int}\} \in x : \{\ell : \texttt{int}\}}{x : \{\ell : \texttt{int}\} \vdash x : \{\ell : \texttt{int}\}}}{x : \{\ell : \texttt{int}\} \vdash x.\ell : \texttt{int}}}{\vdash \{\ell : \texttt{int} = \varsigma(x)\, x.\ell\}}$$

3. Bien typé.

Question 8, page 3

Il suffit de modifier les règles Objet et Over, car Send ne demande aucune modification.

$$\textsc{Object}\quad \frac{\tau = \{\ell_i : \tau_i \ ^{i\in I}\} \qquad \forall\, i \in I \quad \Gamma, x : \tau \vdash a_i : \tau_i' \qquad \tau_i' <: \tau_i}{\Gamma \vdash \{\ell_i : \tau_i = \varsigma(x)\, a_i \ ^{i\in I}\} : \tau}$$

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

Question 9, page 3

On étend la définition des contextes d'évaluation E et on définit des contextes de propagation P:

$$\begin{array}{lcl} E & ::= & [\,] \mid \mathsf{S}(E) \mid E.\ell \mid E.\ell \Leftarrow \varsigma(x)\, a \mid \texttt{raise}\ E \mid \texttt{try}\ E\ \texttt{with}\ x \to a \\ P & ::= & [\,] \mid \mathsf{S}(P) \mid P.\ell \mid P.\ell \Leftarrow \varsigma(x)\, a \mid \texttt{raise}\ P \end{array}$$

On ajoute les règles de réduction:

$$\begin{array}{rcll} \texttt{try}\ v\ \texttt{with}\ x \to a & \rightsquigarrow & v & \\ \texttt{try raise}\ v\ \texttt{with}\ x \to a & \rightsquigarrow & a[x \leftarrow v] & \\ P[\texttt{raise}\ v] & \rightsquigarrow & \texttt{raise}\ v & \text{si } P \neq [\,] \end{array}$$

Solution alternative A la place des contextes de propagation (et de leur règle de reduction) on ajoute les règles suivantes

$$\begin{array}{rcl} \mathsf{S}(\texttt{raise}\ v) & \rightsquigarrow & \texttt{raise}\ v \\ (\texttt{raise}\ v).\ell & \rightsquigarrow & \texttt{raise}\ v \\ \texttt{raise}\ v.\ell \Leftarrow \varsigma(x)\, a & \rightsquigarrow & \texttt{raise}\ v \\ \texttt{raise}\ (\texttt{raise}\ v) & \rightsquigarrow & \texttt{raise}\ v \end{array}$$

Question 10, page 3

$$\textsc{Try}\quad \frac{\Gamma \vdash a : \tau \quad \Gamma, x : \texttt{int} \vdash b : \tau}{\Gamma \vdash \texttt{try}\ a\ \texttt{with}\ x \to b : \tau}$$

Question 11, page 3

$$\textsc{Raise}\quad \frac{\Gamma \vdash a : \tau \quad \tau <: \texttt{int}}{\Gamma \vdash \texttt{raise}\ a : \texttt{exn}} \qquad \textsc{Try}\quad \frac{\Gamma \vdash a : \tau_1 \quad \Gamma, x : \texttt{int} \vdash b : \tau_2}{\Gamma \vdash \texttt{try}\ a\ \texttt{with}\ x \to b : \tau_1 \vee \tau_2}$$

où $\tau_1 \vee \tau_2$ dénote le plus petit majorant de τ_1 et τ_2, s'il existe: puisque la règle de sous-typage peut être appliquée au dessus de chaque prémisse, il faut prendre le sup. (Pour la

règle RAISE le sous-typage est nécessaire dans le cas où $\tau = \texttt{exn}$ étant ce dernier le seul sous-type de `int`).
Quoique incomplète, les deux règles suivantes seront acceptées (mais moins bien notées):

$$\textsc{Try1}\ \frac{\Gamma \vdash a : \tau_1 \quad \Gamma, x : \texttt{int} \vdash b : \tau_2 \quad \tau_2 <: \tau_1}{\Gamma \vdash \texttt{try}\ a\ \texttt{with}\ x \to b : \tau_1} \qquad \textsc{Try2}\ \frac{\Gamma \vdash a : \tau_1 \quad \Gamma, x : \texttt{int} \vdash b : \tau_2 \quad \tau_1 <: \tau_2}{\Gamma \vdash \texttt{try}\ a\ \texttt{with}\ x \to b : \tau_2}$$

Question 12, page 4

$$\textsc{Try}\ \frac{\Gamma \vdash^* a : \tau \quad \Gamma, x : \texttt{int} \vdash b : \tau}{\Gamma \vdash \texttt{try}\ a\ \texttt{with}\ x \to b : \tau}$$

Question 13, page 4

Elle permet de typer $\texttt{try}\ (\texttt{raise}\ 0).\ell \Leftarrow \varsigma(x)\, 0\ \texttt{with}\ x \to 0$ qui s'évalue en 0.

Question 14, page 4

$$\textsc{Send}^*\ \frac{\Gamma \vdash^* a : \{\ell_i : \tau_i\ ^{i \in I}\} \qquad j \in I}{\Gamma \vdash^* a.\ell_j : \tau_j} \qquad \textsc{Try}^*\ \frac{\Gamma \vdash^* a : \tau \quad \Gamma, x : \texttt{int} \vdash^* b : \tau}{\Gamma \vdash^* \texttt{try}\ a\ \texttt{with}\ x \to b : \tau}$$

Question 15, page 4

La règle est incorrecte, car $(\texttt{try}\ \{\ell : \texttt{int} = \texttt{raise}\ 0\}\ \texttt{with}\ x \to \{\ell : \texttt{int} = 0\}).\ell$ est bien typé mais lève une exception.

Question 16, page 4

La solution proposée n'est pas bonne, car elle ne passe pas à l'abstraction—les méthodes étant la seule façon de faire de l'abstraction dans ce langage tout objets.

Question 17, page 4

Une solution possible consiste à ajouter des annotations * aux *étiquettes dans les types* pour indiquer l'effet latent que l'appel de la méthode pourra éventuellement lever une exception et modifier les règles de typage OBJECT et SEND comme suit:

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

où ϵ est soit vide soit $*$. On aurait alors $\vdash^* \{\ell : \texttt{int} = \texttt{raise}\ 0\}.\ell : \texttt{int}$ (mais pas $\vdash \{\ell : \texttt{int} = \texttt{raise}\ 0\}.\ell : \texttt{int}$) obligeant à protéger cette expression.