Programmation 1

TD n°12

Aliaume Lopez

16 décembre 2019

♻ : reprise d'un exercice ❗ : exercice de compréhension
👍 : exercice fondamental de cours 🏆 : une solution complète par mail = un gâteau

Exercise 1 : Quelques aides

1. Sortir une feuille pour noter la correction.
2. Ne pas attendre la correction pour réfléchir sur une feuille.
3. Ne pas hésiter à demander à son voisin, ou mieux, au chargé de TD.
4. Rédiger et ne pas se contenter d'avoir une idée.

❗ **Exercise 2 : Back to Luc's answer**

Posons $J \triangleq \{[a, b] \mid 0 \leq a \leq b \leq 1\}$ où a et b sont des nombres réels. Rappelons que $(J, \supseteq)$ est un DCPO et que cela donne lieu à une topologie τ sur J, la topologie de Scott.

1. Considérons M l'ensemble des éléments maximaux de J. Déterminer la topologie induite par τ sur M.
2. Montrer que toute propriété P continue sur M est constante.

1 Probably Correct Functions

Pour l'exercice suivant, se munir du poly de cours ou bien des transparents pour avoir accès aux sémantiques dénotationnelles et opérationnelles.

👍 **Exercise 3 : La sémantique n'est pas adéquate**

Considérons l'expression PCF u suivante

```
letrec f (x) = 3 in
letrec g (x) = g (x) in
f (g 0)
```

1. Ce n'est pas une expression valide, car il manque les annotations de type. Les ajouter.
2. Calculer la sémantique dénotationnelle de u.
3. Montrer qu'il n'y a aucune dérivation $E \vdash u \Downarrow v$ où E est un P-environnement, et v une valeur.

👍 **Exercise 4 : Jouons avec les types**

Pour chaque expression OCaml ci-dessous, donner s'il existe le type de l'expression. Justifier.

1. `let f x = x in (f 3, f "trois")`

2. `(fun f -> (f 3, f "trois")) (fun x -> x)`
3. `let f x = x in leg g = ref f in (!g 3, !g "trois")`

👍 **Exercise 5 : Boolean Function Calculus**

On considère le langage suivant

$$M := x \mid \lambda x : \tau.M \mid MN \mid \textbf{let } x : \tau = M \textbf{ in } N \mid \textbf{ff} \mid \textbf{tt} \mid \textbf{if } M \textbf{ then } N \textbf{ else } P$$

1. Proposer un système de typage adapté.
2. Donner une dérivation de $\vdash (\lambda x.\textbf{if } x \textbf{ then ff else } x)\textbf{tt} : \textbf{bool}$
3. Montrer que le système de type proposé est déterministe.
4. Quel élément de la syntaxe du langage de programmation est crucial pour garantir le déterminisme du typage ? Expliciter avec un exemple.
5. Montrer que la construction `let` s'encode à l'aide des autres constructions de manière bien typée.
6. Proposer une sémantique à petit pas pour ce langage.
7. Montrer qu'il y a un théorème de *subject reduction*, c'est-à-dire que la sémantique à petit pas préserve le typage.
8. On ajoute à la syntaxe les deux constructions suivantes

$$\textbf{try } M \textbf{ with } N \mid \textbf{abort}$$

   Proposer une extension du système de typage.
9. Proposer une extension de la sémantique à petit pas. Le théorème de *subject reduction* tient-il toujours ?

❗ **Exercise 6 : Les véritables exceptions**

On ajoute des constructeurs d'exception que l'on note $C_1, \ldots, C_n$. Ce sont par exemple des exceptions comme `KeyboardInterrupt`. Pour chaque C_i, on considère un type τ_i d'argument fixé et on ajoute les règles de déductions

$$\frac{}{C_i : \tau_i \to \textbf{exn}}$$

1. Adapter la syntaxe. Quelles sont les valeurs ? Quels sont les contextes ?
2. Adapter la sémantique à petit pas.
3. L'utiliser pour réduire le terme suivant en supposant que $M \to^* V$.

$$\textbf{try } (\lambda x.\lambda y.y)(\textbf{abort } M) \textbf{ with } C_i(x) \mapsto x$$

4. Le langage OCaml interdit la construction d'exceptions possédant un type polymorphe. Expliquer.

Exercise 7 : Relations logiques et terminaison

L'objectif de cet exercice est de montrer que tout programme bien typé en BoolPCF sans `let` termine. Pour cela, on donne la sémantique à petit pas suivante

$$\begin{aligned}(\lambda x.M)N &\to M[x \mapsto N]\\ \textbf{if tt then } M \textbf{ else } N &\to M\\ \textbf{if ff then } M \textbf{ else } N &\to N\\ C[M] &\to C[M'] \qquad \text{si } M \to M'\end{aligned}$$

On note V l'ensemble des valeurs.

$$C := \square \mid VC \mid CM \mid \textbf{if } C \textbf{ then } M \textbf{ else } M$$

On utilise la notation $M \Downarrow$ pour signifier que M termine.
On découpe la preuve en deux en utilisant une construction intermédiaire (une relation logique). Formellement, on prouve $\Gamma \vdash M : \tau \implies M \in \|\tau\|_C$ puis $M \in \|\tau\|_C \implies M \Downarrow$.
On donne ci-après la définition inductive de $\|\tau\|_V$ (pour valeurs) et $\|\tau\|_C$ (pour calculs).

$$\|\textbf{bool}\|_V \triangleq \{\textbf{tt}, \textbf{ff}\} \qquad \|\tau\|_C \triangleq \{M \mid M \to^* v \in \|\tau\|_V\}$$
$$\|\sigma \to \tau\|_V \triangleq \{\lambda x : \sigma.M \mid \forall v \in \|\sigma\|_V, M[x \mapsto v] \in \|\tau\|_C\}$$

La définition inductive sur τ ne prend en compte que des programmes sans variables libres. Pour pallier cela, on définit

$$\|\tau\|_X^f \triangleq \left\{ M \mid \exists \Gamma, \Gamma \vdash M : \tau \wedge \forall \rho : \prod_{x:\sigma \in \Gamma} \|\sigma\|_V, M\rho \in \|\tau\|_X \right\}$$

Ce qui se lit, « un terme avec des variables libres $x_i : \sigma_i$ est dans $\|\tau\|_X^f$ si et seulement s'il est bien typé et pour toute instanciation des x_i avec des éléments de $\|\sigma_i\|_V$, le terme est dans $\|\tau\|_X$ ». Remarquons que les variables sont instanciées avec des termes *clos* !

1. Montrer que la sémantique opérationnelle est déterministe.
2. Montrer que le typage est préservé par réduction.
3. Montrer que $\|\tau\|_V \subseteq \|\tau\|_C$, en déduire la même inclusion pour les termes ouverts.
4. Montrer que $\|\tau\|_C$ est clos par réduction et anti-réduction. En déduire que $\|\tau\|_C^f$ est clos par les mêmes opérations.
5. Montrer par induction sur τ la propriété suivante
$$\forall M, \forall \Gamma, \forall \tau, \Gamma \vdash M : \tau \implies M \in \|\tau\|_C^f$$
6. Montrer par induction sur τ la propriété suivante
$$\forall M, \forall \tau, M \in \|\tau\|_C^f \implies M \Downarrow$$
7. En déduire que tout terme typé termine.
8. En particulier, l'expression suivante n'est pas typable : $\Delta \triangleq \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. Que "calcule" cette expression ?
9. Si on ajoute Δ comme une construction primitive, quel serait son type ?
10. Cet ajout change-t-il le théorème de terminaison ?