TD 10

Exercice 1. On considère l'expression PCF u suivante:

$$\begin{array}{l}\texttt{letrec } f_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) = \dot{3} \texttt{ in}\\ \texttt{letrec } g_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) = g_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) \texttt{ in}\\ \quad f_{\texttt{int}\to\texttt{int}}(g_{\texttt{int}\to\texttt{int}}\dot{0}).\end{array}$$

Montrer que $[\![u]\!]\rho = 3$ pour n'importe quel environnement ρ. Montrer que pourtant, on ne peut pas dériver $E \vdash u \Rightarrow 3$ pour aucun P-environnement E: en fait, il n'y a pas de dérivation $E \vdash u \Rightarrow p$ pour aucun E et aucun p.

En déduire que la sémantique opérationnelle de PCF_V n'est pas adéquate par rapport à la sémantique dénotationnelle de PCF.

Exercice 2. Pour chaque expression Caml ci-dessous, donnez son type s'il existe. Commentez.

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 let g = ref f in (!g 3,!g "trois")`

Exercice 3. On considère le langage de programmation de fonctions booléenes suivant:

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

où τ est un type de la forme:

$$\tau ::= \textbf{bool} \mid \sigma \to \tau.$$

1. Proposez un système de types pour ce langage de programmation. Donnez la dérivation de $\vdash (\lambda x : \textbf{bool}.\ \textbf{if } x \textbf{ then ff else } x \textbf{ fi})\ \textbf{tt} : \textbf{bool}$.
2. Montrez que le typage est déterministe: si $\Gamma \vdash M : \tau$ et $\Gamma \vdash M : \tau'$, alors $\tau = \tau'$.
3. Quel élément de la syntaxe du langage de programmation est crucial pour garantir le déterminisme du typage? Explicitez avec un exemple.
4. Montrer que la construction let...in s'encode à l'aide des autres constructions de manière bien typée.

Exercice 4. On reprend le langage de l'exercice précédent. On définit la relation $M \to N$ comme suit:

$$\begin{array}{r}(\lambda x : \tau.\ M)\ N \to M[N/x]\\ \textbf{let } x : \tau = M \textbf{ in } N \to N[M/x]\\ \textbf{if tt then } M \textbf{ else } N \textbf{ fi} \to M\\ \textbf{if ff then } M \textbf{ else } N \textbf{ fi} \to N\end{array}$$

et $M \to N$ implique $C[M] \to C[N]$ pour tout contexte C.

1. Soit M, N tels que $M \to N$. Montrez que si $\Gamma \vdash M : \tau$, alors $\Gamma \vdash N : \tau$.
2. La réciproque est-elle vraie?

Exercice 5. On veut écrire un vérificateur de type pour le langage des exercices précédents.

1. Définissez les types Caml `typ` et `exp` des types et expressions.
2. Écrivez une fonction `check_type` prenant en paramètres une expression M et un type τ , et renvoyant vrai si $\vdash M : \tau$, et faux sinon.

Exercice 6. On étend le langage précédent avec des exceptions.

1. On se place tout d'abord dans le cadre d'un système d'exception très rudimentaire, défini par la syntaxe
$$M ::= \cdots \mid \textbf{try } M \textbf{ recovering with } N \mid \textbf{abort}$$
Proposez une extension du système de type pour supporter ce mécanisme d'exception.
2. On se rapproche un peu plus du système d'exception de Caml. On ajoute des constructeurs d'exceptions $C_1, \ldots, C_n$ et on associe à C_i le type $\tau_i \to$`exn`, où `exn` est un nouveau type, que l'on peut voir comme le type somme dont les constructeurs sonts $C_1, \ldots, C_n$. Proposez une syntaxe, un système de types, et une sémantique à petit pas pour ce langage.
3. Caml interdit les constructeurs d'exceptions ayant un type polymorphe. Expliquez pourquoi sur un exemple.

Exercice 7. Le langage *PCF finitaire* est une variante de PCF où le type de base est celui des booléens et où il n'y a pas de points fixes. Plus précisément, les types sont définis par

$$\tau ::= \textbf{bool} \mid \sigma \to \tau$$

et les termes par

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

où **tt**, **ff**, et $\bot$ sont de type **bool**, et où la conditionnelle n'est définie que pour les M, N, P de type **bool**. Les contextes d'évaluation sont définis par

$$C[.] ::= .M \mid \textbf{if } . \textbf{ then } M \textbf{ else } N \textbf{ fi}$$

La sémantique *small-step* est définie par

$$\begin{array}{r} \bot \to \bot \\ (\lambda x : \tau.M)N \to M[N/x] \\ \textbf{if tt then } M \textbf{ else } N \textbf{ fi} \to M \\ \textbf{if ff then } M \textbf{ else } N \textbf{ fi} \to N \end{array}$$

avec la règle supplémentaire $C[M] \to C[M']$ si $M \to M'$ et si $C[.]$ est un contexte d'évaluation.

1. PCF finitaire est-il un langage par nom ou par valeur? Que faudrait-il changer à la sémantique *small-step* pour obtenir l'autre sémantique?
2. On dit qu'un terme M *converge* (noté $M \downarrow$) si il existe un terme N tel que $M \to^* N \not\to$. Donnez deux exemples de termes M de type **bool** tels que $M \not\downarrow$.
3. Proposez une sémantique dénotationelle de PCF finitaire. Montrez que votre sémantique est correcte.

4. On appelle λ-terme un terme de PCF qui ne contient pas $\bot$. On admet que tout λ-terme converge.

   Déduisez-en que votre sémantique est adéquate.

5. On définit la notion d'équivalence observationnelle suivante: $M \simeq N$ si pour tout contexte $C[.]$ (contexte d'évaluation toujours), $C[M] \to^* \mathbf{tt}$ ssi $C[N] \to^* \mathbf{tt}$. Montrer que $M \simeq N$ ssi pour tout contexte C,

   $$C[M]\downarrow \quad \Leftrightarrow \quad C[N]\downarrow .$$

6. Montrez que le ou parallèle n'est pas définissable en PCF finitaire.

7. Montrez que le modèle dénotationnel n'est pas complètement abstrait pour PCF finitaire.