

TD 11

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

$$\begin{aligned} & \mathbf{letrec} \ f_{\text{int} \rightarrow \text{int}}(x_{\text{int}}) = \dot{3} \ \mathbf{in} \\ & \mathbf{letrec} \ g_{\text{int} \rightarrow \text{int}}(x_{\text{int}}) = g_{\text{int} \rightarrow \text{int}}(x_{\text{int}}) \ \mathbf{in} \\ & \quad f_{\text{int} \rightarrow \text{int}}(g_{\text{int} \rightarrow \text{int}} \dot{0}) \end{aligned}$$

Montrer que $\llbracket u \rrbracket \rho = 3$ 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 booléennes suivant :

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

où τ est un type de la forme :

$$\tau ::= \mathbf{bool} \mid \sigma \rightarrow \tau.$$

1. Proposez un système de types pour ce langage de programmation. Donnez la dérivation de $\vdash (\lambda x : \mathbf{bool}. \mathbf{if} \ x \ \mathbf{then} \ \mathbf{ff} \ \mathbf{else} \ x \ \mathbf{fi}) \ \mathbf{tt} : \mathbf{bool}$.
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` 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 $M \rightarrow N$ comme suit :

$$\begin{aligned} & (\lambda x : \tau. M) \ N \rightarrow M[N/x] \\ & \mathbf{let} \ x : \tau = M \ \mathbf{in} \ N \rightarrow N[M/x] \\ & \mathbf{if} \ \mathbf{tt} \ \mathbf{then} \ M \ \mathbf{else} \ N \ \mathbf{fi} \rightarrow M \\ & \mathbf{if} \ \mathbf{ff} \ \mathbf{then} \ M \ \mathbf{else} \ N \ \mathbf{fi} \rightarrow N \end{aligned}$$

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

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

Exercice 5. 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

$$M ::= \dots \mid \mathbf{try} \ M \ \mathbf{recovering} \ \mathbf{with} \ N \mid \mathbf{abort}$$

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, \dots, C_n et on associe à C_i le type $\tau_i \rightarrow \mathbf{exn}$, où \mathbf{exn} est un nouveau type, que l'on peut voir comme le type somme dont les constructeurs sont C_1, \dots, 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 6. 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

$$\tau ::= \mathbf{bool} \mid \sigma \rightarrow \tau$$

et les termes par

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

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

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

La sémantique *small-step* est définie par

$$\begin{array}{l} \perp \rightarrow \perp \\ (\lambda x : \tau. M)N \rightarrow M[N/x] \\ \mathbf{if} \mathbf{tt} \mathbf{then} M \mathbf{else} N \mathbf{fi} \rightarrow M \\ \mathbf{if} \mathbf{ff} \mathbf{then} M \mathbf{else} N \mathbf{fi} \rightarrow N \end{array}$$

avec la règle supplémentaire $C[M] \rightarrow C[M']$ si $M \rightarrow M'$ et si $C[\cdot]$ 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 *termine et converge* (noté $M \downarrow$) si il existe un unique terme N tel que $M \rightarrow^* N \not\rightarrow$. Donnez deux exemples de termes M de type \mathbf{bool} tels que $M \not\downarrow$.
3. Proposez une sémantique dénotationnelle de PCF finitaire. Montrez que votre sémantique est correcte.
4. On appelle λ -terme un terme de PCF qui ne contient pas \perp . On admet que tout λ -terme simplement typé termine et converge. Dédisez-en que votre sémantique est adéquate.
5. On définit la notion d'équivalence observationnelle suivante : $M \simeq N$ si pour tout contexte $C[\cdot]$ (contexte d'évaluation toujours), $C[M] \rightarrow^* \mathbf{tt}$ ssi $C[N] \rightarrow^* \mathbf{tt}$. Montrer que $M \simeq N$ ssi pour tout contexte C ,

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

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.