TD 11

Exercice 1. Pour chaque expression Caml ci-dessous, donnez son type s'il existe.

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")`
4. `let f x = x in let g = ref f in let h = ref f in (!g 3,!h "trois")`

Exercice 2. 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 ?
4. Montrer que la construction `let` s'encode à l'aide des autres constructions de manière bien typée.

On définit la relation $M \rightarrow N$ comme suit :

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

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

5. Soit M, N tels que $M \rightarrow N$. Montrez que si $\Gamma \vdash M : \tau$, alors $\Gamma \vdash N : \tau$.

Exercice 3. 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
$$M ::= \cdots \mid \mathbf{try}\ M\ \mathbf{recovering\ with}\ N \mid \mathbf{abort}$$
Proposez une extension du système de type pour supporter ce mécanisme d'exception.
2. Donnez la dérivation de $\vdash \mathbf{try}\ (\lambda x : \mathbf{bool}.\ \mathbf{if}\ x\ \mathbf{then\ ff\ else\ abort\ fi})\ \mathbf{tt\ recovering\ with\ tt}$
3. 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 $\tau_i \rightarrow$`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.

Exercice 4. Appliquez l'algorithme d'unification «naïf» (exponentiel) vu en cours (voir Figure 1) aux systèmes d'équations suivants. Pouvez vous donner un unificateur autre que le mgu ?

1. $\{y \doteq f(x,z), y \doteq f(\dot{3},\dot{5})\}$
2. $\{f(g(x)) \doteq f(z), g(z) \doteq g(g(\dot{3}))\}$
3. $\{\mathsf{a}(x,x) \doteq \mathsf{a}(\mathbf{int}, \mathsf{a}(\mathbf{int},\mathbf{int}))\}$
4. $\{f(x) \doteq f(f(f(x)))\}$
5. $\{\mathsf{a}(\mathsf{a}(x,\mathbf{int}),\mathbf{int}) \doteq \mathsf{a}(y,z), \mathsf{a}(\mathbf{int},y) \doteq \mathsf{a}(z,t)\}$
6. $\{\alpha \doteq \beta \rightarrow \beta, \beta \doteq \gamma \rightarrow \gamma, \gamma \doteq \delta \rightarrow \delta\}$

Exercice 5. Associez à chaque expression pureML ci-dessous un système d'équations en appliquant l'algorithme de HINDLEY (voir Figure 2). Donnez «de tête» son mgu lorsqu'il existe.

1. $x \dot{+} \dot{3}$
2. $\dot{3}\ \dot{4}$
3. `letrec` $f(x) = x \dot{+} \dot{3}$ `in` $f\ y$
4. `letrec` $f(x) =$ `if` $x = 0$ `then` $\dot{0}$ `else` $f\ (x \dot{+} (\dot{-1}))$ `in` $f\ y$

Donnez un exemple de terme clos de pureML qui ne type pas en monomorphic pureML.

$$
\begin{array}{rlll}
(E \cup \{f(s_1,\ldots,s_m) \doteq f(t_1,\ldots,t_n)\},\theta) & \to (E \cup \{s_1 \doteq t_1,\ldots,s_m \doteq t_m\},\theta) & & \text{(Dec)} \\
(E \cup \{f(s_1,\ldots,s_m) \doteq g(t_1,\ldots,t_n)\},\theta) & \to \mathsf{Fail} & \text{si } f \neq g & \text{(DecFail)} \\
(E \cup \{x \doteq x\},\theta) & \to (E,\theta) & & \text{(Triv)} \\
(E \cup \{x \doteq t\},\theta) & \to (E[x := t],\theta[x := t]) & \text{si } x \notin \mathrm{fv}(t) & \text{(Bind)} \\
(E \cup \{t \doteq x\},\theta) & \to (E[x := t],\theta[x := t]) & \text{si } x \notin \mathrm{fv}(t) & \text{(Bind')} \\
(E \cup \{x \doteq t\},\theta) & \to \mathsf{Fail} & \text{si } t \neq x \in \mathrm{fv}(t) & \text{(Check)} \\
(E \cup \{t \doteq x\},\theta) & \to \mathsf{Fail} & \text{si } t \neq x \in \mathrm{fv}(t) & \text{(Check')}
\end{array}
$$

FIGURE 1 – Algorithme d'unification de ROBINSON.

Pour une expression de monomorphic-pureML rectifiée u_0, c'est-à-dire que chaque variable n'est liée qu'au plus une fois, l'algorithme crée une variable de type fraîche α_t pour chaque sous-terme t de u_0. L'algorithme construit un *système d'équations de types E_{u_0}* en ajoutant inductivement les ensembles d'équations pour chaque sous-terme :

$$
\begin{array}{ll}
\{\alpha_{\dot{n}} \doteq \mathtt{int}\} & \text{pour } \dot{n} \\
\{\alpha_u \doteq \alpha_v \to \alpha_{uv}\} & \text{pour } uv \\
\{\alpha_x \doteq \alpha_u, \alpha_{\mathtt{let}\ x=u \mathtt{in} v} \doteq \alpha_v\} & \text{pour } \mathtt{let}\ x = u\ \mathtt{in}\ v \\
\{\alpha_f \doteq \alpha_x \to \alpha_u, \alpha_{\mathtt{letrec}\ f(x)=u \mathtt{in} v} \doteq \alpha_v\} & \text{pour } \mathtt{letrec}\ f(x) = u\ \mathtt{in}\ v \\
\{\alpha_u \doteq \mathtt{int}, \alpha_v \doteq \mathtt{int}, \alpha_{u \dot{+} v} \doteq \mathtt{int}\} & \text{pour } u \dot{+} v \\
\{\alpha_u \doteq \mathtt{int}, \alpha_{\dot{-} u} \doteq \mathtt{int}\} & \text{pour } \dot{-} u \\
\{\alpha_u \doteq \mathtt{int}, \alpha_{\mathtt{if}\ u=0 \mathtt{then} v \mathtt{else} w} \doteq \alpha_v, \alpha_{\mathtt{if}\ u=0 \mathtt{then} v \mathtt{else} w} \doteq \alpha_w\} & \text{pour } \mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w
\end{array}
$$

On ajoute enfin un contexte de typage Γ_{u_0} formé des hypothèses $x : \alpha_x$ quand x parcourt les variables libres de u_0. On calcule alors le mgu θ de E_{u_0} et on obtient un jugement de typage $\Gamma_{u_0}\theta, \Delta \vdash u_0 : \alpha_{u_0}\theta$ où Δ type les variables non libres de u_0.

FIGURE 2 – Algorithme de typage de HINDLEY.