Appendix E

Corrigés des exercices du chapitre 6

Exercice 6.1

```
type ('a, 'b, 'c) enreg = { e : 'a; f : 'b; g : 'c }
type 'a pre = Pre of 'a
type abs = Abs

let vide = { e = Abs; f = Abs; g = Abs }

let enreg_e a = { e = Pre a; f = Abs; g = Abs }

let proj_e r = match r.e with Pre v -> v
let proj_f r = match r.f with Pre v -> v
let proj_g r = match r.g with Pre v -> v

let exten_e r x = { e = Pre x; f = r.f; g = r.g }
let exten_f r x = { e = r.e; f = Pre x; g = r.g }
let exten_g r x = { e = r.e; f = r.f; g = Pre x }
```

Exercice 6.2

Hypothèse (H1) pour $\texttt{proj}_e$: supposons $E \vdash \texttt{proj}_e(\{e_1 = v_1;\ \ldots;\ e_n = v_n\}) : \tau$ et $e = e_i$. Une dérivation de ce typage est forcément de la forme:

$$\dfrac{E \vdash \texttt{proj}_e : \{e_i : \texttt{Pre}\ \tau;\ \ldots\} \rightarrow \tau \qquad \dfrac{E \vdash v_1 : \tau_1 \qquad \ldots \qquad E \vdash e_n : \tau_n}{E \vdash \{e_1 = v_1;\ \ldots;\ e_n = v_n\} : \{e_1 : \texttt{Pre}\ \tau_1;\ \ldots;\ e_n : \texttt{Pre}\ \tau_n;\ \emptyset\}}}{E \vdash \texttt{proj}_e(\{e_1 = v_1;\ \ldots;\ e_n = v_n\}) : \tau}$$

D'où $\tau = \tau_i$, et le résultat attendu puisque nous avons une sous-dérivation de $E \vdash v_i : \tau_i$ et que le résultat de la delta-réduction est v_i.

Hypothèse (H1) pour $\texttt{exten}_e$: supposons $E \vdash \texttt{exten}_e(\{e_1 = v_1; \ldots; e_n = v_n\}, v) : \tau$. Si e n'est pas l'une des e_i, la dérivation de ce typage est de la forme:

$$\dfrac{E \vdash \texttt{exten}_e : \tau' \times \tau_v \to \tau \qquad \dfrac{\dfrac{E \vdash v_1 : \tau_1 \quad \ldots \quad E \vdash e_n : \tau_n}{E \vdash \{e_1 = v_1; \ \ldots; \ e_n = v_n\} : \tau'} \qquad E \vdash v : \tau_v}{E \vdash (\{e_1 = v_1; \ \ldots; \ e_n = v_n\}, v) : \tau' \times \tau_v}}{E \vdash \texttt{exten}_e(\{e_1 = v_1; \ \ldots; \ e_n = v_n\}, v) : \tau}$$

avec $\tau' = \{e_1 : \texttt{Pre}\ \tau_1; \ \ldots; \ e_n : \texttt{Pre}\ \tau_n; \ \emptyset\}$ et $\tau = \{e : \texttt{Pre}\ \tau_v; \ e_1 : \texttt{Pre}\ \tau_1; \ \ldots; \ e_n : \texttt{Pre}\ \tau_n; \ \emptyset\}$. D'autre part, le résultat de la delta-réduction est $\{e = v; \ e_1 = v_1; \ \ldots; \ e_n = v_n\}$. Comme e et les e_i sont deux à deux différentes, la règle (record) appliquée aux hypothèses $E \vdash v : \tau_v$ et $E \vdash v_i : \tau_i$ montre que le résultat de la delta-réduction a le type τ, comme désiré.

Si $e = e_j$, on a une dérivation de la même forme mais avec $\tau = \{e_1 : \texttt{Pre}\ \tau_1; \ \ldots; \ e_j : \texttt{Pre}\ \tau_v; \ldots; e_n : \texttt{Pre}\ \tau_n; \emptyset\}$. Le résultat de la delta-réduction est $\{e_1 = v_1; \ldots; e_j = v; \ldots; e_n = v_n\}$. On conclut que ce résultat est de type τ à partir des hypothèses $E \vdash v : \tau_v$ et $E \vdash v_i : \tau_i$, $i \not j$.

Forme des valeurs selon leur type: on examine les règles de typage qui peuvent s'appliquer à des valeurs (const-inst, op-inst, fun, paire, record) et on remplit la matrice suivante des combinaisons (valeur, type) possibles:

	c	op	$\texttt{fun}\ x \to a$	(v_1, v_2)	$\{e_i = v_i\}$
T	X				
$\tau_1 \to \tau_2$		X	X		
$\tau_1 \times \tau_2$				X	
$\{\tau\}$					X

(Rappelons que par l'hypothèse H0 un opérateur a toujours un type flèche). De plus, le type d'une valeur enregistrement $\{\ldots e_i = v_i \ldots\}$ est nécessairement de la forme $\{\ldots e_i : \texttt{Pre}\ \tau_i \ldots; \ \emptyset\}$ avec $\emptyset \vdash v_i : \tau_i$. Donc, toutes les étiquettes marquées présentes dans ce type enregistrement sont bien présentes dans la valeur.

Hypothèse (H2) pour $\texttt{proj}_e$: supposons $\emptyset \vdash \texttt{proj}_e(v) : \tau$. Vu le schéma de type de $\texttt{proj}_e$, nous avons $\emptyset \vdash v : \{e : \texttt{Pre}\ \tau_1; \ \tau_2\}$. Donc, v est un enregistrement et contient un champ étiqueté e. Par conséquent, l'application de $\texttt{proj}_e$ se réduit par la delta-règle pour les projections.

Hypothèse (H2) pour $\texttt{exten}_e$: si $\emptyset \vdash \texttt{exten}_e(v) : \tau$, v a nécessairement un type enregistrement. C'est donc un enregistrement, et l'application $\texttt{exten}_e(v)$ se réduit par la delta-règle pour $\texttt{exten}_e$.

Exercice 6.3
Algorithme de sortage: $S(\tau, \kappa, K) = K'$ si τ est de la sorte κ, ou échec sinon.

- Si $\tau = \alpha$ et $\alpha \notin Dom(K)$:
  prendre $K' = K + \{\alpha \leftarrow \kappa\}$.

- Si $\tau = \alpha$ et $\alpha \in Dom(K)$:
  si $\kappa = K(\alpha)$, prendre $K' = K$, sinon échec.

- Si $\tau = T$:
  si $\kappa = \mathtt{TYPE}$, prendre $K' = K$, sinon échec.

- Si $\tau = \tau_1 \rightarrow \tau_2$ ou $\tau = \tau_1 \times \tau_2$:
  si $\kappa = \mathtt{TYPE}$, prendre $K' = S(\tau_2, \mathtt{TYPE}, S(\tau_1, \mathtt{TYPE}, K))$, sinon échec.

- Si $\tau = \{\tau'\}$:
  si $\kappa = \mathtt{TYPE}$, prendre $K' = S(\tau', \mathtt{R}(\emptyset), \tau')$, sinon échec.

- Si $\tau = \emptyset$:
  si $\kappa = \mathtt{R}(E)$ pour un certain E, prendre $K' = K$, sinon échec.

- Si $\tau = (e : \tau_1;\ \tau_2)$:
  Si $\kappa = \mathtt{TYPE}$ ou $\kappa = \mathtt{PRE}$, échec.
  Si $\kappa = \mathtt{R}(E)$ et $e \in E$, échec.
  Si $\kappa = \mathtt{R}(E)$ et $e \notin E$, prendre $K' = S(\tau_2, \mathtt{R}(E \cup \{e\}), S(\tau_1, \mathtt{TYPE}, K))$.

- Si $\tau = \mathtt{Abs}$:
  si $\kappa = \mathtt{PRE}$, prendre $K' = K$, sinon échec.

- Si $\tau = \mathtt{Pre}\ \tau'$:
  si $\kappa = \mathtt{PRE}$, prendre $K' = S(\tau', \mathtt{TYPE}, K)$, sinon échec.

Pour typer une contrainte $(a : \tau)$ ou une déclaration de constructeur C `of` τ, on appelle $S(\tau, \mathtt{TYPE}, K)$ et l'algorithme S vérifier et propage la contrainte $\tau :: \mathtt{TYPE}$ vers les variables de types de τ, déterminant au passage leurs sortes.

Exercice 6.4
L'idée est d'interpréter la rangée τ dans le type somme $[\tau]$ comme un "ou", et non plus comme un "et" comme dans le cas des enregistrements. Ainsi, le type enregistrement $\{e : \mathtt{Pre\ int};\ f : \mathtt{Pre\ bool};\ \emptyset\}$ signifie "il y a un champ e de type `int` *et* un champ f de type `bool` *et* rien d'autre". Le type somme $[C : \mathtt{Pre\ int};\ D : \mathtt{Pre\ bool};\ \emptyset]$ signifie, lui, "il y a un constructeur C qui porte un argument de type `int` *ou* un constructeur D d'argument `bool` *ou* rien d'autre". Si la rangée se termine par une variable α au lieu de $\emptyset$, cela signifie "...ou d'autres constructeurs" au lieu de "...ou rien d'autre". Avec ces intuitions, on obtient les types suivants pour les opérateurs:

$$\begin{array}{rcl} C & : & \forall \alpha, \beta.\ \alpha \rightarrow [C : \mathtt{Pre}\ \alpha;\ \beta] \\ P_C & : & \forall \alpha.\ [C : \mathtt{Pre}\ \alpha;\ \emptyset] \rightarrow \alpha \\ F_C & : & \forall \alpha, \beta, \gamma.\ [C : \mathtt{Pre}\ \alpha;\ \beta] \times (\alpha \rightarrow \gamma) \times ([\beta] \rightarrow \gamma) \rightarrow \gamma \end{array}$$

Le type de C indique que le résultat contient le constructeur C avec un argument de type α, mais peut aussi être vu comme contenant d'autres constructeurs si le contexte le demande. Ainsi, $C(1)$ a le type $[C : \mathtt{Pre\ int};\ \tau]$ pour toute rangée τ, et `if` *cond* `then` $C(1)$ `else` $D(\mathtt{true})$ a le type $[C : \mathtt{Pre\ int};\ D : \mathtt{Pre\ bool};\ \tau]$, puisque le résultat est soit un C soit un D.

Le type de P_C exprime que l'argument doit posséder le constructeur C et aucun autre, c.à.d. que l'on est sûr statiquement que la projection ne peut pas échouer.

Enfin, le type de F_C indique que le premier argument doit contenir le constructeur C et peut-être d'autres constructeurs. Le second argument doit s'appliquer à l'argument de C. Quant au troisième argument, il doit s'appliquer à tous les constructeurs qui peuvent être dans le premier argument, *sauf* C. En effet, le troisième argument ne sera jamais appliqué si le premier est de constructeur C. On obtient en particulier les types suivants:

$$\begin{array}{rcl}
\texttt{fun}\ x \rightarrow F_C(x, \texttt{fun}\ y \rightarrow y, \texttt{fun}\ z \rightarrow 0) & : & [C : \texttt{Pre int};\ \alpha] \rightarrow \texttt{int} \\
\texttt{fun}\ x \rightarrow F_C(x, \texttt{fun}\ y \rightarrow y, P_D) & : & [C : \texttt{Pre int};\ D : \texttt{Pre int};\ \emptyset] \rightarrow \texttt{int}
\end{array}$$

La première fonction correspond, en ML, à un filtrage "ouvert" (avec un cas par défaut à la fin): elle peut s'appliquer à n'importe quelle somme contenant au moins le constructeur C d'argument `int`. La seconde fonction correspond à un filtrage "fermé" (sans cas attrape-tout à la fin): elle ne peut s'appliquer qu'à des sommes qui contiennent au plus les constructeurs C et D.