Appendix A

Corrigés des exercices du chapitre 1

Exercice 1.1 La construction ML `let rec` $f\ x = a_1$ `in` a_2 peut être vue comme du "sucre syntaxique" pour

$$\texttt{let}\ f = \texttt{fix}(\texttt{fun}\ f \rightarrow \texttt{fun}\ x \rightarrow a_1)\ \texttt{in}\ a_2$$

Exercice 1.2 Le `let rec` multiple peut toujours se ramener à un `let rec` simple en paramétrisant l'une des définitions par-rapport à l'autre. Autrement dit, `let rec` $f\ x = a_1$ `and` $g\ y = a_2$ `in` a_3 peut se transformer en

```
let rec f g x = a1 in
let rec g y = let f = f g in a2 in
let f = f g in
a3
```

Ensuite, on encode ces deux `let rec` simples en termes de `fix` comme dans l'exercice 1.1.

Exercice 1.3 On montre facilement que si deux prédicats P et Q satisfont les règles, alors leur conjonction $P \wedge Q$ les satisfait aussi. En effet, si P et Q satisfont les axiomes, alors $P(a_i(x))$ est vrai pour tout x, ainsi que $Q(a_i(x))$, et donc $(P \wedge Q)(a_i(x))$ est vrai. Pour ce qui est des règles, supposons que $(P \wedge Q)(b_j^k(x))$ est vrai pour tout $k = 1 \ldots n$. Alors, $P(b_j^k(x))$ est vrai pour tout k, et donc $P(c_j(x))$ est vrai puisque P satisfait la règle j. De même, $Q(b_j^k(x))$ est vrai pour tout k, et donc $Q(c_j(x))$ est vrai puisque Q satisfait la règle j. Il s'ensuit que l'implication

$$(P \wedge Q)(b_j^1(x)) \wedge \ldots \wedge (P \wedge Q)(b_j^n(x)) \Rightarrow (P \wedge Q)(c_j(x))$$

est vraie, et donc $P \wedge Q$ satisfait la règle j.

Le résultat précédent s'étend trivialement à des conjonctions sur un nombre arbitraire de prédicats satisfaisant les règles: si on a une famille de prédicats $(P_a)_{a \in A}$ qui satisfont les règles, alors leur conjonction $\bigwedge_{a \in A} P_a$ les satisfait aussi.

On considère alors $P_{min} = \bigwedge\{P \mid P \text{ satisfait les règles}\}$ (c'est-à-dire, la conjonction de tous les prédicats satisfaisant les règles). Par le résultat précédent, P_{min} satisfait les règles, et par construction il est plus petit que tout autre prédicat P satisfaisant les règles.

Exercice 1.4 On note $D(x)$ le prédicat "il existe une dérivation de l'énoncé $P(x)$ dans le système de règles". La preuve que D est le plus petit prédicat satisfaisant les règles est en deux temps: (1) on montre que D satisfait les règles, et (2) on montre que pour tout prédicat Q satisfaisant les règles, $D(x)$ implique $Q(x)$.

Pour (1), il est vrai que $\forall x. D(a_i(x))$, puisque pour tout x donné, $P(a_i(x))$ est une dérivation valide réduite à une feuille. De même, si pour un certain x, nous avons $D(b_j^1(x)) \wedge \ldots \wedge D(b_j^n(x))$, cela signifie que nous avons des dérivations de $P(b_j^1(x)) \ldots P(b_j^n(x))$; on peut donc construire une dérivation de noeud racine $P(c_j(x))$ et de fils les dérivations de $P(b_j^1(x)) \ldots P(b_j^n(x))$, et c'est une dérivation valide. Donc, $D((c_j(x))$ est vrai. Il s'ensuit que D satisfait les règles.

Pour (2), on se donne un prédicat Q satisfaisant les règles, et on montre par récurrence structurelle sur la dérivation que pour tout x, si l'on a une dérivation de l'énoncé $P(x)$, alors $Q(x)$ est vrai. Il s'ensuit $D(x)$ implique $Q(x)$ pour tout x, et donc D est plus petit que Q. Prenant $Q = P_{min}$, par minimalité de P_{min}, il s'ensuit $D = P_{min}$ comme annoncé.

Exercice 1.5 Pour $a = 1\ 2$, une dérivation de $a \xrightarrow{v} v$ devrait se terminer par une des règles 4, 7, 8 ou 9. Mais pour que ces règles s'appliquent, il faudrait que 1 s'évalue en une fonction (pour la règle 4) ou en une opération (pour les autres règles). Cependant, la seule valeur en laquelle 1 peut s'évaluer est 1, qui n'est rien de tout cela. Donc, il n'y a pas de dérivation de $a \xrightarrow{v} v$ pour tout v.

Pour $a' = (\texttt{fun}\ f \to f\ f)\ (\texttt{fun}\ f \to f\ f)$, notons $b = (\texttt{fun}\ f \to f\ f)$. Une dérivation de $a' \xrightarrow{v} v$ doit nécessairement se terminer ainsi:

$$\frac{b \xrightarrow{v} b \qquad b \xrightarrow{v} b \qquad (f\ f)[f \leftarrow b] \xrightarrow{v} v}{b\ b \xrightarrow{v} v}$$

Mais $(f\ f)[f \leftarrow b] = b\ b = a'$, donc toute dérivation de $a' \xrightarrow{v} v$ doit contenir une sous-dérivation de $a' \xrightarrow{v} v$; il n'existe bien sûr pas de dérivation finie satisfaisant cette propriété.

La différence entre a et a' est que a est un terme essentiellement mal formé, alors que a' est un terme bien formé mais dont l'évaluation ne termine pas.

Exercice 1.6 **Cas** $a = \texttt{let}\ x = a_1\ \texttt{in}\ a_2$. La seule règle qui s'applique est 6, et donc D est de la forme

$$\frac{\begin{array}{c}(D_1)\\ \vdots \\ a_1 \xrightarrow{v} v_1\end{array} \qquad \begin{array}{c}(D_2)\\ \vdots \\ a_2[x \leftarrow v_1] \xrightarrow{v} v_2\end{array}}{a_1\ a_2 \xrightarrow{v} v_2}$$

Vu la forme de a, D' se termine nécessairement par la règle 6 elle aussi. Donc, D' contient des sous-dérivations $D'_1 : a_1 \xrightarrow{v} v'_1$ et $D'_2 : a_2[x \leftarrow v'_1] \xrightarrow{v} v'_2$ pour certaines valeurs v'_1 et v'_2. Comme D_1 est une sous-dérivation de D et D'_1 une sous-dérivation de D', nous pouvons appliquer l'hypothèse de récurrence à D_1 et D'_1. Il vient $v_1 = v'_1$. Par conséquent, $a_2[x \leftarrow v_1] = a_2[x \leftarrow v'_1]$ et nous pouvons appliquer l'hypothèse de récurrence à D_2 et D'_2. Il vient $v_2 = v'_2$, ce qui entraîne le résultat attendu $v = v'$.

Exercice 1.7 Pour typer 1 2, il faudrait pouvoir attribuer à 1 un type flèche $\tau_1 \to \tau_2$, ce qui est bien sûr impossible car 1 a le type `int` dans tous les environnements de typage.

Pour typer $\texttt{fun}\ f \to f\ f$, il faudrait construire une dérivation de la forme suivante:

$$\dfrac{\dfrac{E + \{f : \tau_1\} \vdash f : \tau_1 \to \tau_2 \qquad E + \{f : \tau_1\} \vdash f : \tau_2}{E + \{f : \tau_1\} \vdash f\ f : \tau_2}}{E \vdash \texttt{fun}\ f \to f\ f : \tau_1 \to \tau_2}$$

Pour que les feuilles de la dérivation soient justifiées par l'axiome (var), il faudrait que $\tau_1 = \tau_1 \to \tau_2$ et $\tau_1 = \tau_2$. La première de ces égalités est impossible, car τ_1 serait alors un sous-terme strict de lui-même, ce qui est impossible pour tout terme τ_1 fini.

Enfin, dans le cas de $\texttt{let}\ f = \texttt{fun}\ x \to x\ \texttt{in}\ (f\ 1, f\ \texttt{true})$, nous pouvons attribuer à $\texttt{fun}\ x \to x$ le type $\tau \to \tau$ pour n'importe quel τ. Mais pour que $f\ 1$ soit bien typé, il faudrait prendre $\tau = \texttt{int}$, et pour que $f\ \texttt{true}$ soit bien typé, il faudrait prendre $\tau = \texttt{bool}$, et il est impossible de satisfaire ces deux contraintes simultanément.

Exercice 1.8 Notons φ la substitution $[\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n]$. On montre $\mathcal{L}(\varphi(\tau)) = \varphi(\mathcal{L}(\tau))$ par récurrence structurelle sur τ. Le cas de base est τ est une variable γ. Si $\gamma = \alpha_i$ pour un certain i, on a

$$\mathcal{L}(\varphi(\alpha_i)) = \mathcal{L}(\beta_i) = \{\beta_i\} = \varphi(\{\alpha_i\}) = \varphi(\mathcal{L}(\alpha_i)).$$

Si $\gamma \notin \{\alpha_1, \ldots, \alpha_n\}$, on a

$$\mathcal{L}(\varphi(\gamma)) = \mathcal{L}(\gamma) = \{\gamma\} = \varphi(\{\gamma\}) = \varphi(\mathcal{L}(\gamma)).$$

Les cas inductifs sont immédiats par application de l'hypothèse de récurrence.

Soient maintenant deux schémas $\sigma = \forall \alpha_1, \ldots, \alpha_n. \tau$ et $\sigma' = \forall \beta_1, \ldots, \beta_n. \tau'$ égaux à alpha-conversion près. On a donc $\tau' = \tau[\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n]$, et de plus $\beta_i \notin \mathcal{L}(\sigma)$ pour tout i. Par ce qui précède,

$$\mathcal{L}(\tau') = \mathcal{L}(\tau)[\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n]$$

Écrivons $\mathcal{L}(\tau) = \{\alpha_{i_1}, \ldots, \alpha_{i_p}, \gamma_1, \ldots, \gamma_q\}$ où les α_{i_j} sont les α_i libres dans τ, et les γ_j les variables libres dans τ qui ne sont pas les α_i. Calculons maintenant les variables libres de σ et σ'.

$$\begin{array}{rcl}
\mathcal{L}(\sigma) & = & \mathcal{L}(\tau) \setminus \{\alpha_1, \ldots, \alpha_n\} \\
 & = & \{\gamma_1, \ldots, \gamma_q\} \\
\mathcal{L}(\sigma') & = & \mathcal{L}(\tau') \setminus \{\beta_1, \ldots, \beta_n\} \\
 & = & (\mathcal{L}(\tau)[\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n]) \setminus \{\beta_1, \ldots, \beta_n\} \\
 & = & (\{\alpha_{i_1}, \ldots, \alpha_{i_p}, \gamma_1, \ldots, \gamma_q\}[\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n]) \setminus \{\beta_1, \ldots, \beta_n\} \\
 & = & \{\beta_{i_1}, \ldots, \beta_{i_p}, \gamma_1, \ldots, \gamma_q\} \setminus \{\beta_1, \ldots, \beta_n\} \\
 & = & \{\gamma_1, \ldots, \gamma_q\}
\end{array}$$

Pour la dernière égalité, on s'appuie sur le fait que $\{\gamma_1 \ldots \gamma_q\} \cap \{\beta_1 \ldots \beta_n\} = \emptyset$ car sinon l'une des β_i serait libre dans σ. D'où $\mathcal{L}(\sigma) = \mathcal{L}(\sigma')$. CQFD.

Exercice 1.9 Pour le terme `let` $f =$ `fun` $x \to x$ `in` $f\ f$, on donne à f le schéma $\forall\alpha.\, \alpha \to \alpha$, puis on donne à la première occurrence de f un type $(\tau \to \tau) \to (\tau \to \tau)$, et à la seconde occurrence $\tau \to \tau$. Le terme entier a donc le type $\tau \to \tau$ pour τ arbitraire.

Pour `fun` $f \to f\ f$, ce terme n'est toujours pas typable. En effet, une dérivation de typage de ce terme devrait se terminer par:

$$\dfrac{\dfrac{\dfrac{\tau \to \tau_2 \leq \tau_1}{\{f : \tau_1\} \vdash f : \tau \to \tau_2} \qquad \dfrac{\tau \leq \tau_1}{\{f : \tau_1\} \vdash f : \tau}}{\{f : \tau_1\} \vdash f\ f : \tau_2}}{\emptyset \vdash \texttt{fun}\ f \to f\ f : \tau_1 \to \tau_2}$$

pour des types τ, τ_1, τ_2 bien choisis. Cependant, $\tau \to \tau_2 \leq \tau_1$ implique $\tau \to \tau_2 = \tau_1$, et de même $\tau \leq \tau_1$ implique $\tau = \tau_1$. Donc, il faudrait que $\tau \to \tau_2 = \tau$, et ceci est impossible pour tout τ (un terme fini ne peut pas se contenir comme sous-terme).

Exercice 1.10 Considérons $a =$ `fun` $x \to$ `let` $y = x$ `in` y. Sous l'hypothèse $x : \alpha$, avec la définition plus simple de *Gen*, on obtiendrait $y : \forall\alpha.\, \alpha$, et donc l'occurrence de y après le `in` pourrait recevoir le type β. a aurait donc le type $\alpha \to \beta$ où α et β sont deux variables de types distinctes. Ce n'est bien sûr pas un type correct pour la fonction identité.

Exercice 1.11 Commençons par définir l'image $\varphi(\sigma)$ d'un schéma $\sigma = \forall\alpha_1 \ldots \alpha_n.\ \tau$ par une substitution φ. Ensuite, $\varphi(E)$ sera simplement l'application point à point de φ sur les schémas contenus dans E, c'est-à-dire $\varphi(E) = E'$ est tel que $E'(x) = \varphi(E(x))$ pour tout $x \in \mathrm{Dom}(E)$. Naïvement, on prendrait

$$\varphi(\forall\alpha_1 \ldots \alpha_n.\ \tau) = \forall\alpha_1 \ldots \alpha_n.\ \varphi(\tau)$$

mais cela pose des problèmes de capture de variables liées par le quantificateur $\forall$. Par exemple, si $\varphi = [\alpha \leftarrow \beta]$, cela donnerait:

$$\varphi(\forall\alpha.\alpha) = \forall\alpha.\beta \qquad \varphi(\forall\beta.\alpha) = \forall\beta.\beta$$

et on voit que ces deux résultats sont incorrects en se rappelant que les variables liées α et β dans les deux schémas de types peuvent être renommées en toute autre variable γ sans changer la signification du schéma de types. Or,

$$\varphi(\forall\gamma.\gamma) = \forall\gamma.\gamma \qquad \varphi(\forall\gamma.\alpha) = \forall\gamma.\beta.$$

Autrement dit, la définition naïve ci-dessus ne passe pas au quotient par alpha-conversion (renommage des variables liées): suivant les rennomages que l'on effectue sur les variables liées du schéma argument, on obtient des schémas résultats différents, même à alpha-conversion près.

L'idée est de forcer le rennomage "qui va bien" dans le schéma argument afin d'éviter toute interférence entre les variables liées et la substitution. On prend donc:

$$\varphi(\forall\alpha_1 \ldots \alpha_n.\ \tau) = \forall\alpha_1 \ldots \alpha_n.\ \varphi(\tau) \text{ si } \alpha_1, \ldots, \alpha_n \text{ sont hors de portée de } \varphi.$$

Dans cette définition, on dit qu'une variable α est hors de portée d'une substitution φ si

1. $\varphi(\alpha) = \alpha$ (c'est-à-dire, φ ne modifie pas α)

2. si α n'est pas libre dans τ, alors α n'est pas libre dans $\varphi(\tau)$ (c'est-à-dire, φ n'introduit pas α dans son résultat).

Par exemple, prenant $\varphi = [\alpha \leftarrow \beta]$, on voit que α n'est pas hors de portée de φ (car $\varphi(\alpha) = \beta$, donc la condition 1 n'est pas vraie), et β n'est pas hors de portée de φ non plus (car β n'est pas libre dans α, mais β est libre dans $\varphi(\alpha) = \beta$, donc la condition 2 n'est pas vraie). En revanche, γ est hors de portée de φ, car $\varphi(\gamma) = \gamma$, et de plus si γ n'est pas libre dans τ, alors τ ne contient que des variables $\alpha, \beta, \delta, \ldots$, que φ transforme en $\beta, \beta, \delta, \ldots$ respectivement, donc γ n'est pas non plus libre dans $\varphi(\gamma)$.

Notez que dans la définition de $\varphi(\forall \alpha_1 \ldots \alpha_n.\ \tau)$, la condition "$\alpha_1, \ldots, \alpha_n$ sont hors de portée de φ" peut toujours être satisfaite en renommant préalablement les $\alpha_1 \ldots \alpha_n$. (Il y a un nombre infini de variables hors de portée d'une substitution donnée.) On a donc défini $\varphi(\sigma)$ pour tout schéma σ.

Passons à la preuve de la proposition 1.2. La preuve est par récurrence sur la dérivation de typage de $E \vdash a : \tau$ et par cas sur la dernière règle de typage utilisée.

Cas règle (var-inst). Nous avons $E \vdash x : \tau$ avec $\tau \leq E(x)$. Écrivons $E(x) = \forall \alpha_1 \ldots \alpha_n . \tau_x$. Nous avons donc $\tau = \tau_x[\alpha_1 \leftarrow \tau_1, \ldots, \alpha_n \leftarrow \tau_n]$. Quitte à renommer les α_i, nous pouvons de plus supposer les α_i hors de portée de φ. Donc:

$$(\varphi(E))(x) = \varphi(E(x)) = \forall \alpha_1 \ldots \alpha_n . \varphi(\tau_x)$$

d'une part, et d'autre part

$$\varphi(\tau) = \varphi(\tau_x[\alpha_1 \leftarrow \tau_1, \ldots, \alpha_n \leftarrow \tau_n]) = \varphi(\tau_x)[\alpha_1 \leftarrow \varphi(\tau_1), \ldots, \alpha_n \leftarrow \varphi(\tau_n)].$$

Ceci montre que $\varphi(\tau) \leq (\varphi(E))(x)$. Par conséquent, la règle (var-inst) nous permet de conclure que $\varphi(E) \vdash x : \varphi(\tau)$, ce qui est le résultat désiré.

Cas règle (const-inst) ou (op-inst). Comme les schémas $TC(c)$ et $TC(op)$ sont clos (sans variables libres) pour tous c et op, on a $\varphi(TC(c)) = TC(c)$ et de même $\varphi(TC(op)) = TC(op)$. On conclut alors par le même raisonnement que pour la règle (var-inst).

Cas règle (fun). Nous avons $E \vdash (\texttt{fun}\ x \rightarrow a) : \tau_1 \rightarrow \tau_2$ en conséquence de la prémisse $E + \{x : \tau_1\} \vdash a : \tau_2$. Appliquant l'hypothèse de récurrence à cette prémisse, nous obtenons une dérivation de $\varphi(E + \{x : \tau_1\}) \vdash a : \varphi(\tau_2)$, c'est-à-dire $\varphi(E) + \{x : \varphi(\tau_1)\} \vdash a : \varphi(\tau_2)$. Par application de la règle (fun), nous concluons $\varphi(E) \vdash (\texttt{fun}\ x \rightarrow a) : \varphi(\tau_1) \rightarrow \varphi(\tau_2)$, ce qui est le résultat désiré.

Cas règle (app) ou (paire). Même raisonnement que pour la règle (fun).

Cas règle (let). C'est là que les choses se compliquent. Nous avons donc $E \vdash (\texttt{let}\ x = a_1\ \texttt{in}\ a_2) : \tau_2$ en conséquence des prémisses $E \vdash a_1 : \tau_1$ et $E + \{x : \mathit{Gen}(\tau_1, E)\} \vdash a_2 : \tau_2$. Le problème est que les variables généralisées par Gen, c'est-à-dire $\{\alpha_1, \ldots, \alpha_n\} = \mathcal{L}(\tau_1) \setminus \mathcal{L}(E)$, ne sont pas forcément hors de portée de φ, et donc on n'a pas, en général,

$$\varphi(\mathit{Gen}(\tau_1, E)) = \mathit{Gen}(\varphi(\tau_1), \varphi(E)).$$

(Exemple: on peut avoir $\{y : \alpha\} \vdash a_1 : \beta \rightarrow \beta$, et β est généralisable, mais après application de la substitution $[\beta \leftarrow \alpha]$, on se retrouve avec $\{y : \alpha\} \vdash a_1 : \alpha \rightarrow \alpha$, dans lequel α n'est pas généralisable!)

L'astuce est d'arriver à renommer les variables généralisées α_i afin qu'elles soient hors de portée de φ. Pour ce faire, on va appliquer l'hypothèse de récurrence à $E \vdash a_1 : \tau_1$ et non pas à la substitution φ, mais à une substitution ψ "proche" de φ mais contournant les problèmes de capture.

Plus précisément, on se donne des variables $\beta_1, \ldots, \beta_n$ non libres dans E et hors de portée de φ, et on considère la substitution

$$\psi = \varphi \circ [\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n]$$

Par application de l'hypothèse de récurrence, on a $\psi(E) \vdash a_1 : \psi(\tau_1)$. Comme les β_i ne sont pas libres dans E, on a $[\alpha_1 \leftarrow \beta_1, \ldots, \alpha_n \leftarrow \beta_n](E) = E$ et donc $\psi(E) = \varphi(E)$. La dérivation obtenue par récurrence prouve donc

$$\varphi(E) \vdash a_1 : \psi(\tau_1)$$

On applique également l'hypothèse de récurrence à la seconde prémisse, avec la substitution φ cette fois-ci. On obtient une dérivation de

$$\varphi(E) + \{x : \varphi(Gen(\tau_1, E))\} \vdash a_2 : \varphi(\tau_2)$$

Pour conclure le résultat attendu par application de la règle (let-gen), il reste donc à montrer que

$$Gen(\psi(\tau_1), \varphi(E)) = \varphi(Gen(\tau_1, E)).$$

Les variables libres dans $\psi(\tau_1)$ mais pas dans $\varphi(E)$ sont exactement $\{\beta_1, \ldots, \beta_n\}$. En effet, les variables libres dans τ_1 sont

$$\alpha_1, \ldots, \alpha_n, \text{et des variables libres dans } E$$

Lorsque l'on applique le renommage $[\alpha_i \leftarrow \beta_i]$, ces variables libres deviennent

$$\beta_1, \ldots, \beta_n, \text{et des variables libres dans } E$$

Enfin, lorsqu'on applique φ, ces variables libres deviennent

$$\beta_1, \ldots, \beta_n, \text{et des variables libres dans } \varphi(E)$$

Les β_i ne sont pas libres dans $\varphi(E)$, car par hypothèse β_i hors de portée de φ, cela impliquerait β_i libre dans E, contredisant l'hypothèse β_i non libre dans E. Par conséquent,

$$Gen(\psi(\tau_1), \varphi(E)) = \forall \beta_1, \ldots, \beta_n . \psi(\tau_1)$$

Or, $Gen(\tau_1, E) = \forall \alpha_1, \ldots, \alpha_n . \tau_1 = \forall \beta_1, \ldots, \beta_n . \tau_1[\alpha_i \leftarrow \beta_i]$ à alpha-conversion près. De plus, les β_i sont hors de portée de φ, donc:

$$\varphi(Gen(\tau_1, E)) = \forall \beta_1, \ldots, \beta_n . \varphi(\tau_1[\alpha_i \leftarrow \beta_i]) = \forall \beta_1, \ldots, \beta_n . \psi(\tau_1)$$

Nous avons donc montré $\varphi(Gen(\tau_1, E)) = Gen(\psi(\tau_1), \varphi(E))$, et le résultat attendu en découle.