Chapitre 7

Solutions des exercices

Exercice 82:

On remarque d'abord que, pour toute requête R et structures de Herbrand $\mathcal{H}_1$, $\mathcal{H}_2$ telles que $\mathcal{H}_1 \subseteq \mathcal{H}_2$, si $\mathcal{H}_1 \models R$ alors $\mathcal{H}_2 \models R$. Puis, on a, comme dans l'exercice 80, que

$$\begin{array}{ll} & P \models R \\ \Leftrightarrow & P \cup \{\neg R\} \text{ non satisfaisable} \\ \Leftrightarrow & P \cup \{\neg R\} \text{ ne possède pas de modèle de Herbrand} \\ \Leftrightarrow & \neg R \text{ est faux dans tout modèle de Herbrand de } P \\ \Leftrightarrow & \mathcal{M}_P \not\models \neg R \\ \Leftrightarrow & \mathcal{M}_p \models R \end{array}$$

Exercice 83:

Évidemment, T_P est monotone. Il suffit donc de montrer pour tout ensemble D dirigé vers le haut que $T_P(\bigcup D) \subseteq \bigcup T_P(D)$. Soit $D \subseteq HS(\mathcal{F}, \mathcal{P})$ dirigé vers le haut, et $A' \in T_P(\bigcup D)$. Par la définition de T_P, il y a une clause $A \leftarrow B_1, \ldots, B_n$ du programme P et une substitution close $\sigma \in \Sigma_g$ telle que $A' = A\sigma$, et $B_1\sigma, \ldots, B_n\sigma \in \bigcup D$. Donc, il y a $S_1, \ldots, S_n \in D$ telles que pour tout i, $B_i\sigma \in S_i$. Puisque D est dirigé vers le haut, il y a un majorant $S \in D$ de $S_1, \ldots, S_n$. Par conséquence, $B_1\sigma, \ldots, B_n\sigma \in S$, donc $A' \in T_P(S) \subseteq \bigcup(T_P(D))$.

Exercice 84:

1. Plusieurs solutions sont possibles. On peut modifier la définition du prédicat *IncList* :

$$\begin{array}{rcl} Nat(0) & \leftarrow & \\ Nat(s(x)) & \leftarrow & Nat(x) \\ SortedList(nil) & \leftarrow & \\ SortedList(cons(x, nil)) & \leftarrow & Nat(x) \\ SortedList(cons(s(x), cons(s(y), z))) & \leftarrow & SortedList(cons(x, cons(y, z))) \\ SortedList(cons(0, cons(y, z))) & \leftarrow & SortedList(cons(y, z)) \end{array}$$

Du point du vu du génie logiciel, il est préférable de couper le prédicat en deux, et de définir un prédicat de comparaison :

$$\begin{array}{rcl}
Nat(0) & \leftarrow & \\
Nat(s(x)) & \leftarrow & Nat(x) \\
InfEgal(0,x) & \leftarrow & Nat(x) \\
InfEgal(s(x),s(y)) & \leftarrow & InfEgal(x,y) \\
SortedList(nil) & \leftarrow & \\
SortedList(cons(x,nil)) & \leftarrow & Nat(x) \\
SortedList(cons(x,cons(y,z))) & \leftarrow & InfEgal(x,y), SortedList(cons(y,z))
\end{array}$$

2. On utilise le fait que s est une permutation de t ssi on peut transformer s en t par une suite de transpositions entre voisins :

$$\begin{array}{rcl}
Permutation(x,x) & \leftarrow & List(x) \\
Permutation(x,y) & \leftarrow & ElemPerm(x,z),\ Permutation(z,y) \\
ElemPerm(cons(x,y),cons(x,z)) & \leftarrow & ElemPerm(y,z) \\
ElemPerm(cons(x,cons(y,z)),cons(y,cons(x,z))) & \leftarrow &
\end{array}$$

3. Le programme est $P_3 \cup P_4$, plus

$$Sort(s,t) \ \leftarrow\ Permutation(s,t),\ SortedList(t)$$

Exercice 85:

Soit T une machine de Turing avec
- alphabet Σ
- symbole blanc $\flat$
- un ensemble fini Q d'états
- un état initial $q_0 \in Q$
- un ensemble $Q_a \subseteq Q$ d'états acceptants
- un ensemble fini Δ de productions de la forme $q, c \mapsto q', c', d$
  où $c, c' \in \Sigma \cup \{\flat\}$, $q, q' \in Q$, $d \in \{\leftarrow, \rightarrow, \downarrow\}$.

Une exécution de la machine T est acceptante quand elle atteint un état acceptant. Le programme logique correspondant consiste des clauses suivantes :

Pour chaque état acceptant $q \in Q_A$

$$Accept(x,q,y) \ \leftarrow$$

Pour chaque production $q, c \mapsto q', c', \downarrow$:

$$Accept(x,q,c(y)) \ \leftarrow\ Accept(x,q',c'(y))$$

Pour chaque production $q, c \mapsto q', c', \rightarrow$:

$$\begin{array}{rcl}
Accept(x,q,c(y)) & \leftarrow & Accept(c'(x),q',y) \\
Accept(x,q,c(\epsilon)) & \leftarrow & Accept(c'(x),q',\flat(\epsilon))
\end{array}$$

Pour chaque production $q, c \mapsto q', c', \leftarrow$, et pour chaque symbole $c'' \in \Sigma \cup \{\flat\}$:

$$\begin{array}{rcl} Accept(c''(x), q, c(y)) & \leftarrow & Accept(x, q', c''(c'(y))) \\ Accept(\epsilon, q, c(y)) & \leftarrow & Accept(\epsilon, q', \flat(c'(y))) \end{array}$$

La requête R_m est $\leftarrow (Accept(\epsilon, q_0, m(\flat(\epsilon))))$.

L'idée est de représenter une configuration (v, q, w) de la machine de Turing par un triplet de termes (s, q, t), où s est un terme représentant le ruban v telle que le symbole de racine de s est le symbole à l'extrémité *droite* de v, et où t est un terme représentant le ruban w telle que le symbole de racine de t est le symbole à l'extrémité *gauche* de v. De plus, on voudrait maintenir l'invariant que le terme t ne soit jamais ϵ. Dans la programmation logique pure, on ne peut pas éviter que dans certaines dérivations cet invariant est violé (il n'y a pas de *if-then-else* comme on OCaml par exemple). Par contre, les dérivations qui violent cet « invariant » ne peuvent pas réussir (sauf quand la violation est le dernier pas pour atteindre un état acceptant).

Exercice 86:

On montre d'abord l'implication indiquée dans la preuve. La conclusion est équivalente à

$$\tilde{\forall}\big((\neg L\theta \wedge \neg K\theta) \rightarrow (P(\bar{t})\theta \wedge \neg K\theta)\big)$$

Soit $\mathcal{A}$ un modèle de la clause de programme, et α une valuation telle que $\alpha, \mathcal{A} \models \neg L\theta \wedge \neg K\theta$. Il faut montrer que $\alpha, \mathcal{A} \models P(\bar{t})\theta$. Puisque $\mathcal{A} \models \tilde{\forall}(P(\bar{s}) \vee L)$, on a aussi que $\mathcal{A} \models \tilde{\forall}((P(\bar{s}) \vee L)\theta)$, donc $\alpha, \mathcal{A} \models P(\bar{t})\theta \vee L\theta$. Puisque $\alpha, \mathcal{A} \models \neg L\theta$, il suit que $\alpha, \mathcal{A} \models P(\bar{t})\theta$.

Maintenant, on montre par induction sur n que pour tout $i < n$:

$$P \models \tilde{\forall}\big(\neg L_{i+1} \rightarrow \neg L_1(\theta_1 \circ \ldots \circ \theta_i)\big)$$

Pour $i = 0$, on a évidemment $P \models (\neg L_1 \rightarrow \neg L_1)$.

Pour le pas d'induction : Par la petite proposition montrée au-dessus, on a

$$P \models \tilde{\forall}(\neg L_{i+1} \rightarrow \neg L_i\theta_i)$$

et par hypothèse d'induction :

$$P \models \tilde{\forall}\big(\neg L_i \rightarrow \neg L_1(\theta_1 \circ \ldots \circ \theta_{i-1})\big)$$

Donc, on a aussi

$$P \models \tilde{\forall}\big(\neg L_i\theta_i \rightarrow \neg L_1(\theta_1 \circ \ldots \circ \theta_{i-1} \circ \theta_i)\big)$$

Il suit que

$$P \models \tilde{\forall}\big(\neg L_{i+1} \rightarrow \neg L_1(\theta_1 \circ \ldots \circ \theta_{i-1} \circ \theta_i\big)$$

Dans une LD-réfutation de longueur m on a que $L_m = \square$ et $L_1 = (\neg A_1 \vee \ldots \vee \neg A_n)$, donc

$$P \models \tilde{\forall}\big(\neg\square \rightarrow \neg(\neg A_1 \vee \ldots \vee \neg A_n)(\theta_1 \circ \ldots \circ \theta_{m-1})\big)$$

ce qui est équivalent à

$$P \models \tilde{\forall}\big((A_1 \wedge \ldots \wedge A_n)(\theta_1 \circ \ldots \circ \theta_{m-1})\big)$$

Exercice 87:

Soit $N := (\leftarrow A_1, \ldots, A_n)$. La preuve est par induction sur la longueur de la LD-réfutation de $P \cup \{N\sigma\}$. Si la longueur est 0, alors $N = \square$ et l'énoncé suit toute de suite.

Supposons donnée une LD-réfutation de la longueur $n + 1$ de $P \cup \{N\sigma\}$. Soit $R(\bar{s}) \leftarrow L$ la clause du programme utilisée dans le premier pas de cette réfutation. N est donc de la forme $\leftarrow K_1, R(\bar{t}), K_2$. Comme d'habitude nous pouvons supposer que $\mathrm{Var}(R(\bar{s}) \leftarrow L) \cap Dom(\sigma) = \emptyset$. Donc, il y une substitution θ_1 telle que $\bar{t}\sigma\theta_1 = \bar{s}\theta_1 = \bar{s}\sigma\theta_1$. Donc, il y a aussi un mgu θ'_1 de $\bar{s} \stackrel{?}{=} \bar{t}$, et une substitution τ telle que $\theta'_1 \circ \tau = \sigma \circ \theta_1$.

Il existe une LD-réfutation de $P \cup \{((\leftarrow K_1, L, K_2)\sigma)\theta_1\}$ de longueur n, avec substitution de réponse θ_2. Par hypothèse d'induction, il existe donc aussi une LD-réfutation de $P \cup \{(\leftarrow K_1, L, K_2)\theta'_1\}$ avec substitution de réponse θ'_2, telle que $\theta'_2 \geq \tau \circ \theta_2$.

On obtient une LD-réfutation de $P \cup \{N\}$ avec substitution de réponse $\theta'_1 \circ \theta'_2$. On a donc que

$$\theta'_1 \circ \theta'_2 \geq \theta'_1 \circ \tau \circ \theta_2 = \sigma \circ \theta_1 \circ \theta_2$$

Exercice 88:

Toutes les substitutions de réponse obtenues sont des substitutions closes :

$$\{\{x \to cons(s^n(0), cons(s^{n+1}(0), \ldots, cons(s^{n+m}(0), nil) \ldots))\} \mid n, m \geq 0\}$$

Exercice 89:

On montre que $\mu_1 \circ \nu_2$ est un mgu de $\bar{s}_1 \stackrel{?}{=} \bar{t}_1 \wedge \bar{s}_2 \stackrel{?}{=} \bar{t}_2$. L'énoncé en suit par l'unicité du mgu à renommage près (voir l'exercice 74).

Pour montrer que $\mu_1 \circ \nu_2$ est un unificateur de $\bar{s}_1 \stackrel{?}{=} \bar{t}_1 \wedge \bar{s}_2 \stackrel{?}{=} \bar{t}_2$: On a d'abord que $\bar{s}_1\mu_1 = \bar{t}_1\mu_1$, donc $\bar{s}_1\mu_1\nu_2 = \bar{t}_1\mu_1\nu_2$. De plus, $(\bar{s}_2\mu_1)\nu_2 = \bar{s}_2\nu_2 = (\bar{t}_2\mu_1)\nu_2$.

Pour montrer que $\mu_1 \circ \nu_2 \geq \tau$ pour tout unificateur τ de $\bar{s}_1 \stackrel{?}{=} \bar{t}_1 \wedge \bar{s}_2 \stackrel{?}{=} \bar{t}_2$: Soit τ un unificateur de $\bar{s}_1 \stackrel{?}{=} \bar{t}_1 \wedge \bar{s}_2 \stackrel{?}{=} \bar{t}_2$. τ est donc aussi un unificateur de $\bar{s}_1 \stackrel{?}{=} \bar{t}_1$. Donc, il existe une substitution τ_1 telle que $\tau = \mu_1 \circ \tau_1$. Puisque τ est aussi un unificateur de $\bar{s}_2 \stackrel{?}{=} \bar{t}_2$, on a

$$\bar{s}_2\tau_1 = (\bar{s}_2\mu_1)\tau_1 = \bar{s}_2\tau = \bar{t}_2\tau = (\bar{t}_2\mu_1)\tau_1$$

Donc il y a une substitution τ_2 telle que $\tau_1 = \nu_2 \circ \tau_2$. En conséquence : $\tau = \mu_1 \circ \tau_1 = \mu_1 \circ \nu_2 \circ \tau_2$.

Exercice 90:

Voir Figure 7.1.

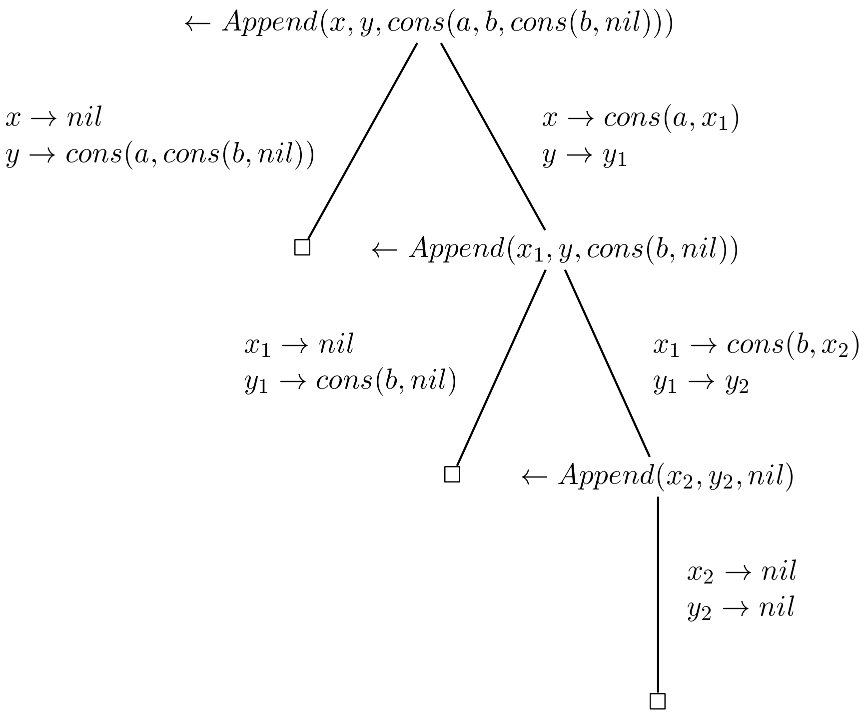

FIG. 7.1 – La solution de l'exercice 90.