Flashback post correction de DM

Exercice 1 :
Considérons les expressions booléennes suivantes :

$$b ::= \texttt{True} \mid \texttt{False} \mid e \dot{<} e \mid \dot{\neg} b \mid b \dot{\vee} b$$

où les expressions arithmétiques e sont les mêmes que celles de IMP mais sans variables (il n'y a donc pas besoin d'environnement), équipées de leur sémantique dénotationelle $[\![e]\!]$.

1. Donner une sémantique opérationnelle à petits pas $\to$ pour les expressions booléennes, qui vérifie les propriétés des questions suivantes suivantes et faire les preuves avec **tous** les cas.
2. On rappelle que $\to^*$ est la clôture réflexive et transitive de $\to$. Montrer les point suivants.
   - $\dot{2} \dot{<} \dot{1} \to^* \texttt{False}$.
   - $\dot{\neg}((\texttt{False} \mathbin{\dot{\vee}} \texttt{False}) \mathbin{\dot{\vee}} \texttt{True}) \to^* \texttt{False}$.
   - $\texttt{False} \mathbin{\dot{\vee}} (\texttt{False} \mathbin{\dot{\vee}} (\texttt{False} \mathbin{\dot{\vee}} \texttt{True})) \to^* \texttt{True}$.
3. Montrer que pour toute expression booléenne b, il existe au plus une expression booléenne b' telle que $b \to b'$.
4. (Progrès) Montrer que pour tout expression bouléenne b, soit b est l'expression `True`, soit b est l'expression `False`, soit il existe b' telle que $b \to b'$.
5. Proposer une sémantique opérationelle à grand pas cette fois, dont les jugement pourront être notés $\vdash b \Rightarrow x$ où b est une expression booléenne et $x \in \{0, 1\}$, avec $\vdash b \Rightarrow 1$ (resp. 0) signifiant que b finit son calcule sur `True` (resp. `False`).

Maintenant, on type

Ne surtout pas hésiter à utiliser le cours sans modération.

Exercice 2 :
(Un exercice nécessaire) Considérons l'expression PCF u suivante :

```
letrec f x = if x=0 then 0 else x + f (x-1) in f 6
```

Calculer la sémantique dénotationnelle de u.

Exercice 3 :
(La sémantique n'est pas adéquate) Considérons l'expression PCF u suivante :

```
letrec f (x) = 3 in
letrec g (x) = g (x) in
f (g 0)
```

1. Ce n'est pas une expression valide, car il manque les annotations de type. Les ajouter.

2. Calculer la sémantique dénotationnelle de u.
3. Montrer qu'il n'y a aucune dérivation $E \vdash u \Rightarrow v$ où E est un P-environnement, et v une valeur.

Exercice 4 :
(Boolean Function Calculus) On considère le langage suivant

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

1. Proposer un système de typage adapté.
2. Donner une dérivation de $\vdash (\lambda x.\textbf{if } x \textbf{ then ff else } x)\textbf{tt} : \textbf{bool}$
3. Montrer que le système de type proposé est déterministe.
4. Quel élément de la syntaxe du langage de programmation est crucial pour garantir le déterminisme du typage ? Expliciter avec un exemple.
5. Montrer que la construction `let` s'encode à l'aide des autres constructions de manière bien typée.

Unification et typage

Arbres et termes

On note Σ une signature algébrique et $\mathbb{X}$ un ensemble infini dénombrable de variables. L'ensemble $T_\Sigma(\mathbb{X})$ est l'ensemble des arbres *finis* dont les nœuds sont des éléments de Σ ou des variables dans $\mathbb{X}$, qui sont alors nécessairement des feuilles.
Plus formellement, on écrit $T_\Sigma(\mathbb{X})$ comme l'algèbre initiale engendrée par Σ et $\mathbb{X}$.
En particulier, si (A, Σ) est une Σ-algèbre, et $f : \mathbb{X} \to A$ est une évaluation des variables alors il existe une unique fonction $f^\dagger : T_\Sigma(\mathbb{X}) \to A$ qui est un morphisme de Σ-algèbres et qui coïncide avec f sur les variables.

Substitutions

Une substitution σ est une fonction de $\mathbb{X}$ vers $T_\Sigma(\mathbb{X})$ qui diffère de l'identité seulement sur un ensemble fini de variables.
On note $t\sigma$ le terme obtenu via $\sigma^\dagger(t)$ lorsque σ est une substitution et t un terme.
On dit qu'une substitution est *plate* lorsque chaque variable est envoyée sur une variable. On dit qu'une substitution est un *renommage* lorsqu'elle est plate et est une bijection.
Lorsque σ et τ sont deux substitutions, on note $\sigma\tau$ la substitution $\tau^\dagger \circ \sigma$, ce qui se traduit par $t(\sigma\tau) = (t\sigma)\tau$.

Ordre sur les substitutions

On écrit $\sigma \leq \tau$ lorsqu'il existe une substitution θ telle que $\sigma\theta = \tau$. Cet ordre est l'ordre de *généralisation*.

Problème d'unification

Un problème d'unification est un ensemble E fini de contraintes de la forme $t \doteq t'$ où t et t' sont des termes. Une solution à un problème d'unification E est une substitution σ telle que

$$\forall t \doteq t' \in E, t\sigma = t'\sigma$$

Exercice 5 :

(Définitions de base) La relation $\leq$ sur les substitutions n'est pas antisymétrique.

1. Montrer que $\sigma \leq \tau \wedge \tau \leq \sigma$ si et seulement si σ et τ ne diffèrent que par un renommage.
2. Montrer que s'il existe une solution à un problème d'unification, il en existe une unique plus générale (à renommage près). *Hint : quel algorithme permet de le calculer ?*

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

FIGURE 1 – Algorithme d'unification de ROBINSON.

Exercice 6 :

(Algorithme naïf d'unification) Appliquer l'algorithme d'unification « naïf » (exponentiel) vu en cours (voir Figure 1) aux systèmes d'équations suivants. Pouver 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. $\{x \doteq \mathsf{a}(x,\mathbf{int})\}$
7. $\{\alpha \doteq \beta \to \beta, \beta \doteq \gamma \to \gamma, \gamma \doteq \delta \to \delta\}$

Exercice 7 :

(Unification en temps polynomial)

1. Montrer que l'algorithme classique vu en cours (c.f. Figure 1) peut nécessiter un temps exponentiel.
2. Proposer une structure de donnée pour les mgu qui contourne le problème évoqué à la question précédente.

3. Proposer une modification des règles de l'algorithme naïf adapté à cette nouvelle structure.
4. Quelle est la complexité de l'algorithme obtenu ?

Exercice 8 :
(Monomorphisme et typage)

1. Donner un exemple de terme clos de pureML qui ne type pas en monomorphic pureML.
2. Donner un exemple de terme clos qui ne type pas en pureML mais qui ne se réduit pas à Wrong.

Exercice 9 :
(Effets de bords, part II) En imaginant la généralisation naturelle des règles de typage de pureML, typer le programme suivant :

```
let r = ref (fun x -> x)
in
  r := (fun n -> n+1);
    !r "abc" ;;
```

Exercice 10 :
(Bushes) Écrire en OCaml une fonction `length` pour le type suivant :

```
type 'a mycroft =
  | Nil
  | Cont of 'a * ('a list) mycroft
```

Discuter.