Programmation 1

TD n°13

Aliaume Lopez

15 janvier 2020

♻ : reprise d'un exercice ❗ : exercice de compréhension
👍 : exercice fondamental de cours 🏆 : une solution complète par mail = un gâteau

Exercise 1 : Quelques aides

1. Sortir une feuille pour noter la correction.
2. Ne pas attendre la correction pour réfléchir sur une feuille.
3. Ne pas hésiter à demander à son voisin, ou mieux, au chargé de TD.
4. Rédiger et ne pas se contenter d'avoir une idée.

1 Probably Correct Functions

Pour l'exercice suivant, se munir du poly de cours ou bien des transparents pour avoir accès aux sémantiques dénotationnelles et opérationnelles.

❗ **Exercise 2 : Les véritables exceptions**

On ajoute des constructeurs d'exception que l'on note $C_1, \dots, C_n$. Ce sont par exemple des exceptions comme `KeyboardInterrupt`. Pour chaque C_i, on considère un type τ_i d'argument fixé et on ajoute les règles de déductions

$$\frac{}{C_i : \tau_i \to \mathbf{exn}}$$

1. Adapter la syntaxe. Quelles sont les valeurs ? Quels sont les contextes ?
2. Adapter la sémantique à petit pas.
3. L'utiliser pour réduire le terme suivant en supposant que $M \to^* V$.

$$\mathbf{try}\ (\lambda x.\lambda y.y)(\mathbf{abort}\ M)\ \mathbf{with}\ C_i(x) \mapsto x$$

4. Le langage OCaml interdit la construction d'exceptions possédant un type polymorphe. Expliquer.

2 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

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

Exercise 3 : 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 ?*

Exercise 4 : Algorithme naïf d'unification

Appliquez l'algorithme d'unification « naïf » (exponentiel) vu en cours (voir Figure 1) aux

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

FIGURE 1 – Algorithme d'unification de ROBINSON.

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. $\{x \doteq \mathsf{a}(x, \mathbf{int})\}$

7. $\{\alpha \doteq \beta \to \beta, \beta \doteq \gamma \to \gamma, \gamma \doteq \delta \to \delta\}$

! Exercise 5 : Unification en temps polynomial

1. Montrez que l'algorithme classique vu en cours (c.f. Figure 1) peut nécessiter un temps exponentiel.
2. Proposez une structure de donnée pour les mgu qui contourne le problème évoqué à la question précédente.
3. Proposez une modification des règles de l'algorithme naïf adapté à cette nouvelle structure.
4. Quelle est la complexité de l'algorithme obtenu ?

! Exercise 6 : Monomorphisme et typage

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

! Exercise 7 : Effets de bords, part II

En imaginant la généralisation naturelle des règles de typage de pureML, typez le programme suivant :

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

! Exercise 8 : Bushes

Écrire en OCaml une fonction `length` pour le type suivant :

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

Discutez.