

# Programmation 1

## TD n°13

15 décembre 2020

### 1 Unification et typage

#### Arbres et termes

On note  $\Sigma$  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  $\Sigma$  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  $\Sigma$  et  $\mathbb{X}$ .

En particulier, si  $(A, \Sigma)$  est une  $\Sigma$ -algèbre, et  $f : \mathbb{X} \rightarrow A$  est une évaluation des variables alors il existe une unique fonction  $f^\dagger : T_\Sigma(\mathbb{X}) \rightarrow A$  qui est un morphisme de  $\Sigma$ -algèbres et qui coïncide avec  $f$  sur les variables.

#### Substitutions

Une substitution  $\sigma$  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  $\sigma$  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  $\sigma$  et  $\tau$  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  $\theta$  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  $\sigma$  telle que

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

#### Exercice 1 :

The relation  $\leq$  on the substitutions is not antisymmetric.

1. Show that  $\sigma \leq \tau \wedge \tau \leq \sigma$  if and only if  $\sigma$  and  $\tau$  differ only by a renaming.
2. Show that if there is a solution to a unification problem, there is only one most general (except renaming).

#### Exercice 2 :

Apply the “naive” (exponential) unification algorithm seen below (see Figure 1) to the following systems of equations. Can you find unifiers other than the mgu? **Note :** The symbols  $\dot{3}$ ,  $\dot{5}$ , etc. are to be seen as nullary functions.

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.  $\{a(x, x) \doteq a(\mathbf{int}, a(\mathbf{int}, \mathbf{int}))\}$
4.  $\{f(x) \doteq f(f(f(x)))\}$
5.  $\{\alpha \doteq \beta \rightarrow \beta, \beta \doteq \gamma \rightarrow \gamma, \gamma \doteq \delta \rightarrow \delta\}$

**Exercise 3 :**

1. Show that the algorithm seen before (c.f. Figure 1) is necessarily exponential.
2. Propose a data structure for the mgu which circumvents the problem mentioned in the previous question.
3. Propose a modification of the rules of the naive algorithm adapted to this new structure.
4. What is the complexity of the algorithm obtained ?

**Exercise 4 :**

1. Give an example of a closed term from pureML that does not type into monomorphic pureML.
2. Give an example of a closed term which does not type in pureML but which does not reduce to Wrong.

**Exercise 5 :**

Imagining the natural generalization of the pureML typing rules, type the given program :

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

Is it well-typed?

**Exercise 6 :**

Write a function `length` in OCaml for the following type :

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

Explain.

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

FIGURE 1 – Algorithme d'unification de ROBINSON.