TP numéro 2

Sémantique des langages de programmation, ENSIIE

Semestre 1, 2018–19

Le but de ce TP est d'implémenter en OCaml les sémantiques opérationnelles à petits pas de IMP et de MiniML, ce qui nous fournira des interpréteurs pour ces langages.

Pour ceux qui n'aurait pas assez avancé au dernier TP, on trouvera à l'adresse `http://www.ensiie.fr/~guillaume.burel/download/imp.ml` la sémantique opérationnelle à petit pas sur les expressions arithmétiques.

Exercice 1 : Expressions booléennes

On considère maintenant les expressions booléennes, représentées en OCaml parle type :

```
type 'a exp_bool =
    Bool of bool
  | Inf of 'a exp_arith * 'a exp_arith
  | Egal of 'a exp_arith * 'a exp_arith
  | Not of 'a exp_bool
  | Or of 'a exp_bool * 'a exp_bool
  | And of 'a exp_bool * 'a exp_bool
```

1. Définir et implémenter la sémantique opérationnelle à petits pas des expressions booléennes.
2. Tester les deux sémantiques sur l'expression $\mathsf{not}\ (x = 0\ \mathsf{and}\ x \leq (7/z))$ pour la valuation $\{x \mapsto 1, y \mapsto 3, z \mapsto 2\}$ puis dans la valuation $\{x \mapsto 0, z \mapsto 0\}$.

Exercice 2 : IMP

Pour définir la sémantique opérationnelle de IMP, on a besoin de pouvoir manipuler les valuations. Pour cela, on va utiliser des maps. Pour simplifier les choses, on supposera que le type des variables est `string`. On définira donc le type des valuations par :

```
module SM = Map.Make(String)

type valu = int SM.t
```

On pourra retrouver l'ancienne version des valuations avec la fonction suivante :

```
let valuation_of_valu s : string valuation = fun x -> SM.find x s
```

1. Écrire une fonction `affiche_valu` de type `valu -> unit` qui affiche une valuation. On pourra utiliser la fonction `SM.iter`.

On représente en OCaml les programmes de IMP par le type

```
type imp =
    Skip
  | Aff of string * string exp_arith
  | Seq of imp * imp
  | If of string exp_bool * imp * imp
  | While of string exp_bool * imp
```

2. Implémenter la sémantique opérationnelle à petit pas de IMP. La fonction `SM.add` permettra de faire l'opération $\sigma[x \leftarrow n]$.
3. Tester sur l'exemple
   $x := 24; y := 36;$ while not $x = y$ do if $x \leq y$ then $y := y - x$ else $x := x - y$

Exercice 3 : MiniML

Exercice 3.1 : Encodage profond

On représente en OCaml les expressions fonctionnelles de MiniML par :

```
type miniml =
    Const of int
  | Var of string
  | Fun of string * miniml
  | App of miniml * miniml
  | Plus of miniml * miniml
```

On considérera la fonction `fresh_var : unit -> string` suivante qui retourne un nom de variable frais.

```
let fresh_var =
  let counter = ref 0 in
  fun () -> incr counter; Printf.sprintf "x%i" !counter
```

1. Écrire une fonction `subst` de type `string -> miniml -> miniml -> miniml` telle que `subst x t s` substitue la variable `x` par `t` dans `s`. ($\{t/x\}s$ avec les notations du cours.) Attention aux captures de variables !
2. Écrire une fonction `ss_val_step` qui implémente une étape de la sémantique opérationnelle à petits pas d'appels par valeur. On lèvera une exception si on est appelé sur une valeur.

3. Écrire une fonction `ss_val` qui implémente la sémantique opérationnelle à petits pas d'appels par valeur.
4. Tester sur l'exemple $\Big((\texttt{fun}\ x\ \texttt{->}\ (\texttt{fun}\ y\ \texttt{->}\ x+y))\ 1\Big)\ \Big(((\texttt{fun}\ x\ \texttt{->}\ 3)\ 1)+2\Big)$.
5. Faire de même avec la sémantique d'appel par nom.
6. Tester les deux sémantiques sur $(\texttt{fun}\ x\ \texttt{->}\ 1)((\texttt{fun}\ x\ \texttt{->}\ x\ x)\ (\texttt{fun}\ x\ \texttt{->}\ x\ x))$.

Exercice 3.2 : Syntaxe abstraite d'ordre supérieur

La technique de syntaxe abstraite d'ordre supérieur permet d'utiliser les mécanismes de lieurs du langage dans lequel on implémente la sémantique pour gérer les lieurs du langage dont on définit la sémantique. Cela permet de déléguer au langage hôte la gestion de ces lieurs, ce qui simplifie notamment la gestion de la capture des variables.

Dans notre cas, cela revient à utiliser le type suivant pour MiniML :

```
type miniml =
    Const of int
  | Var of string
  | Fun of (miniml -> miniml)
  | App of miniml * miniml
  | Plus of miniml * miniml
```

L'expression $\texttt{fun}\ x\ \texttt{->}\ x+1$ sera alors représentée par `Fun(fun x -> Plus(x, Const 1))`.

7. Reprendre les questions 2 à 6 avec ce nouveau type. Constater qu'on n'a plus besoin de la fonction `subst`.