

TP numéro 1

Sémantique des langages de programmation, ENSIE

Semestre 1, 2019–20

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

Exercice 1 : Expressions arithmétiques

On définit le type des expressions arithmétiques en OCaml :

```
type exp_arith =  
| Ent of int  
| Var of string  
| Plus of exp_arith * exp_arith  
| Moins of exp_arith * exp_arith  
| Fois of exp_arith * exp_arith  
| Div of exp_arith * exp_arith
```

Exercice 1.1 : Sémantique dénotationnelle

On définit le type des valeurs de retour :

```
type valeur = Z of int | Err
```

ainsi que le type des évaluations :

```
type valuation = string -> int
```

1. Écrire une fonction (+~) de type valeur -> valeur -> valeur qui implémente la fonction $\tilde{+}$ vue en cours.
NB : une fois la fonction (+~) préfixe définie, il est possible d'utiliser la notation infixe +~ .
2. Faire de même (-~), (*~)¹ et (/~). On fera attention au cas de la division.

1. Attention de bien mettre un espace pour que ce ne soit pas considéré comme un début de commentaire.

3. Écrire une fonction `eval_arith` de type `exp_arith -> valuation -> valeur` qui implémente la fonction `[[_]]_` vue en cours.
4. Définir l'expression $(z + (x - y)) \times (x + 7)$ et la valuation $\{x \mapsto 1, y \mapsto 3, z \mapsto 2\}$. Quelle est le résultat de l'évaluation de l'expression dans la valuation ?
5. Même question pour l'expression $z + ((x \times y)/(x - 1))$ dans la même valuation.

Exercice 1.2 : Sémantique opérationnelle à petits pas

On définit les configurations comme des enregistrements avec deux champs : une expression arithmétique, et une valuation.

```
type config = { expr : exp_arith; s : valuation }
```

6. Écrire une fonction `etape_sopp_arith` de type `config -> config` applique une étape de la sémantique opérationnelle à petit pas sur la configuration. On lèvera une exception `Terminale v` si la configuration passée en paramètre est terminale, c'est-à-dire que l'expression est la valeur `v`. On lèvera l'exception `Erreur` si aucune étape ne peut être appliquée.
7. Écrire une fonction `eval_sopp_arith` de type `exp_arith -> valuation -> valeur` qui applique itérativement `etape_sopp_arith` sur la configuration initiale jusqu'à obtenir une valeur qui est renvoyée.
8. Tester sur les exemples, comparer avec la partie précédente.

Exercice 2 : Expressions booléennes

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

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

1. Implémenter la sémantique dénotationnelle des expressions booléennes. On utilisera la version qui donne une sémantique coupe-circuit au `and` et au `or`.
2. Définir et implémenter la sémantique opérationnelle à petits pas des expressions booléennes.
3. Tester les deux sémantiques sur l'expression `not (x = 0 and x ≤ (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 3 : IMP

Pour définir la sémantique opérationnelle de IMP, on a besoin de pouvoir manipuler les valuations. Pour cela, on va maintenant utiliser des maps en OCaml. On redéfinit donc le type des valuation de la façon suivante :

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

```
type valuation = int StringMap.t
```

1. Écrire une fonction `affiche_valuation` de type `valuation -> unit` qui affiche une valuation. On pourra utiliser la fonction `StringMap.iter`.
2. Modifier les fonctions écrites dans les parties précédentes pour prendre en compte le nouveau type des valuations. La fonction `StringMap.find` sera utile.

On représente en OCaml les programmes de IMP par le type

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

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