Devoir de Programmation Fonctionnelle

Amine Kheddaoui Gowshigan Selladurai

Avril 2022

Contents

1 Présentation du problème et de sa solution algorithmique

Notre problème ici est d'implémenter un solveur d'équation booléennes dans le langage Ocaml. La première étape consistait donc à définir le type des expressions booléennes que nous allions utiliser. Les éléments booléennes sont définies comme suit :

```
type eb = V of int
   | Vrai
   | Faux
   | AND of eb * eb
   | OR of eb * eb
   |XOR of eb * eb
   | NAND of eb * eb
   | NOT of eb;;
```

La solution algorithmique ici se déroule en troi étapes : Premièrement, la détermination de l'ensemble de variables. Ensuite, la génération des environnements et enfin l'évaluation des expressions.

2 Discussion des choix d'implémentation

2.1 Détermination de l'ensemble de variables

```
let rec ens exp =
  match exp with
   |Vrai -> []
   |Faux -> []
   |V(i)-> [i]
   |NOT(a)->(ens a)
   |AND(a,b)->(ens a)@(ens b)
   |NAND(a,b)->(ens a)@(ens b)
   |OR(a,b)->(ens a)@(ens b)
   |XOR(a,b)->(ens a)@(ens b);;
```

Nous avons d'abord créé une fonction qui vérifie qu'un élément appartient à une liste, utile pour en éliminer les doublons.

```
let rec appartient e l =
  match l with
  |[] -> false
  |head::tail -> (e=head) || appartient e tail;;
```

Nous avons ici la fonction qui élimine tous les éléments qui sont en double dans une liste.

```
let rec elim_doublon l =
  match l with
  |[]->[]
  |head::tail->
  if appartient head tail then
    elim_doublon tail
  else  head::elim_doublon tail ;;
```

2.2 Implémentation des environnements

Tout d'abord, nous avons une fonction append, qui ici nous sert à ajouter un même élément à chaque élément de la liste.

```
let append elem list = List.map ( fun l -> elem::l) list;;
```

Nous avons ensuite une fonction qui génère l'environnement, en se servant de la fonction précédente.

```
let rec generateur_aux l =
 match l with
 |[] -> [[]]
 | hd::tl ->let l_aux = generateur_aux tl in
 (append (a,FALSE) l_aux)@(append (a,TRUE) l_aux);;
```

2.3 Fonctions d'évaluation

```
let rec eval eb=
  match eb with
  |Vrai -> Vrai
  |Faux -> Faux
  |V(i)-> V(i)
  |NOT eb -> NOT (eval eb)
  |AND (eb1,eb2) -> AND(eval eb1,eval eb2)
  |OR (eb1,eb2) -> OR(eval eb1,eval eb2)
  |XOR (eb1,eb2) -> XOR(eval eb1,eval eb2)
  |NAND (eb1,eb2) -> NAND(eval eb1,eval eb2);;
```

2.4 Test des fonctions auxiliaires

2.5 Test des fonctions d'évaluation