▶ **MP – Option Informatique**
Troisième TP Caml
Jeudi 26 novembre 2009

Un peu de logique

1 Expressions logiques

Les expressions logiques que nous manipulons sont formées à partir de :
- Deux constantes vrai et faux.
- Des variables propositionelles : $x_0, x_1, \ldots$.
- Un opérateur unaire : *non*
- Cinq opérateurs binaires : *ou*, *et*, *oux* (ou exclusif), *implique* et *equivaut*.

Cette syntaxe permet de construire des expressions telles que :

$$(x_1 \textit{ ou } x_2) \textit{ et } (x_3 \textit{ implique } 1)$$

$$(x_1 \textit{ ou } x_2) \textit{ et } (x_2 \textit{ ou } x_3) \textit{ et } x_4$$

Nous allons représenter les expressions logiques en *Caml* par des arbres syntaxiques. Pour cela, nous définissons un type récursif expr :

```
type expr =
    Vrai
  | Faux
  | X of int
  | Non of expr
  | Ou of expr * expr
  | Et of expr * expr
  | Oux of expr * expr
  | Implique of expr * expr
  | Equivaut of expr * expr
```

▶ **Question 1** *Écrivez les expressions données ci-dessus en exemple sous forme d'arbres de type* expr.

Un environnement est une fonction de l'ensemble des variables $\{x_0, x_1, \ldots\}$ dans l'espace booléen. Dans la pratique, le support des environnements sera fini. Nous pourrons donc représenter un environnement en *Caml* par un tableau de booléens g. (Le contenu de la case $g.(i)$ correspondant à la valeur booléeene attribuée à la variable x_i.)

▶ **Question 2** *Définissez une fonction* eval *qui prend pour arguments une expression e (de type* expr*) et un environnement g. Cette fonction retournera un booléen correspondant à l'évaluation de e dans l'environnement g.*

value eval : *expr* → *bool vect* → *bool*

2 Arbres de décision

2.1 Définition

Nous allons utiliser une structure de données particulière, les arbres de décision, pour représenter les fonctions booléennes, en espérant que cette structure simplifie les calculs dans les exemples qui nous intéressent. Etant donné un ensemble ordonné de n variables notées $x_0, \ldots, x_{n-1}$, on définit les arbres de décision sur ces variables de la façon suivante :

1. les symboles "vrai" et "faux" sont des arbres de décision ;
2. $Test_{x_i}(v, f)$ est un arbre de décision, si v et f sont des arbres de décision tels que :
   - les arbres v et f sont différents,
   - toutes les variables de v et f sont des x_j avec $j > i$.

Dans un noeud interne $Test_{x_i}(v, f)$, x_i est la variable testée, v est le fils positif et f le fils négatif.

```
type abd =
  | Bool of bool
  | Test of variable * abd * abd
```

Pour formaliser la manière dont un arbre de décision représente une fonction booléenne, on introduit une fonction test composant trois applications g, g' , g" de $B^n \to B$ de la manière suivante : $test(g, g', g'')$ est une fonction de $B^n \to B$ définie par
- $test(g, g', g'')(x_0, \ldots, x_{n-1}) = g'(x_0, ..., x_{n-1})$ si $g(x_0, \ldots, x_{n-1})$ s'évalue en "vrai".
- $test(g, g', g'')(x_0, \ldots, x_{n-1}) = g''(x_0, ..., x_{n-1})$ si $g(x_0, \ldots, x_{n-1})$ s'évalue en "faux".

Finalement :
- Les arbres "vrai" et "faux" représentent respectivement les applications constantes vraie et fausse.
- L'arbre $Test_{x_i}(v, f)$ représente $test(xi, g', g'')$, ou les arbres v et f représentent respectivement les applications g' et g'' .

▶ **Question 3** *Écrivez une fonction evaluation qui prend en argument un arbre de décision a et un tableau x et qui évalue la fonction représentée par a en x*

value evaluation : *abd* → *bool vect* → *bool*

2.2 Manipulations

- **Question 4**
- **Question 5**
- **Question 6**
- **Question 7**
- **Question 8**
- **Question 9**

▶ **MP – Option Informatique**
Troisième TP Caml
Jeudi 26 novembre 2009

Un peu de logique

Un corrigé

▶ Question 2

```
  let rec eval g = function
      Vrai -> true
    | Faux -> false
    | X i -> g.(i)
    | Non e -> not (eval g e)
    | Ou (e1, e2) -> (eval g e1) or (eval g e2)
    | Et (e1, e2) -> (eval g e1) && (eval g e2)
    | Oux (e1, e2) -> (eval g e1) <> (eval g e2)
    | Implique (e1, e2) -> not (eval g e1) || (eval g e2)
    | Equivaut (e1, e2) -> (eval g e1) = (eval g e2)
```

▶ Question 9

```
let et f g = abd_test f g (Bool false)
let ou f g = abd_test f (Bool true) g
let implique f g = abd_test f g (Bool true)
```

▶ Question 4

```
let rec evaluation a vlt = match a with
  | Bool b -> b
  | Test (X i,l,r) -> if vlt.(i) then evaluation l vlt else evaluation r vlt
```

▶ Question 6

```
      let rec abd_neg a = match a with
      | Bool b -> Bool (not b)
      | Test (i,l,r) -> Test(i, abd_neg l, abd_neg r)
```

▶ Question 7

```
let rec abd_partiel v b = function
  | Bool x -> Bool x
  | Test (w,l,r) ->
      if w = v then
if b then l else r
      else
let nl , nr = abd_partiel v b l,abd_partiel v b r
in if nl = nr then nl else Test (w,nl,nr)
```

▶ Question 8

```
let rec min_arbres = function
  | [] -> None
  | Bool x :: s -> min_arbres s
  | Test(i,l,r) :: s -> match min_arbres s with None -> Some i | Some j -> Some (min i j)

let rec abd_test c v f =
  match min_arbres [c;v;f] with
    | None -> (match c with Bool true -> v | Bool false -> f)
    | Some x -> Test (x, abd_test (abd_partiel x false c) (abd_partiel x false v) (abd_partiel x false f),
      abd_test (abd_partiel x true c) (abd_partiel x true v) (abd_partiel x true f)
     )
```