Option Informatique
X 99 : Arbres de décision

Corrigé

11 mars 2007

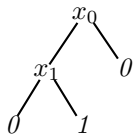
1 Arbres de décision

Question 4

a

- *pour x_0 :* 

- *pour x_1 :* 

- *Les trois dernières formules ont le même arbre :* 

b

Par l'absurde, supposons qu'il existe des arbres représentant une tautologie sans être réduits à la feuille 1*. On peut donc prendre l'arbre a de hauteur minimale parmi ceux-ci. Celui-ci s'écrit test(x_i, a_1, a_0), où a_1 et a_0 sont ses sous-arbres gauche et droit. Ceux-ci sont réduits à la feuile 1, par minimalité de a. Les deux fils de la racine de a sont alors égaux, absurde.*

c

Par récurrence sur le nombre n d'arguments d'une fonction booléenne. Si $n = 1$, voir la question précédente. Soit maintenant g une fonction booléenne à $n \geq 2$ arguments. On a $g \equiv (x_0 \wedge g_{0\leftarrow 1}) \vee ((\neg x_0) \wedge g_{0\leftarrow 0})$ puisque par définition de $g_{i\leftarrow 0/1}$, cette fonction coïncide avec g pour toutes les valuations dex_0. Mais par hypothèse de récurrence, $g_{0\leftarrow 1}$ et $g_{0\leftarrow 0}$ sont réprésentables par des arbres v et f respectivement. Il suffit donc de considérer test(x_0, v, f).

Question 6

a

En s'inspirant de la question 4 :

$$g = (x_i \wedge g_{i\leftarrow 1}) \vee ((\neg x_i) \wedge g_{i\leftarrow 0})$$

$$\neg g = (x_i \wedge (\neg g_{i\leftarrow 1})) \vee ((\neg x_i) \wedge (\neg g_{i\leftarrow 0}))$$

b

Cette question donne en fait la signification des arbres de décision représentant une fonction booléenne, introduite par des exemples dans la partie précédente. Pour représenter la valeur de vérité d'une fonction booléenne f, pour une assignation quelconque de ses variables, on considère les variables d'entrée dans un ordre prédéfini. Chaque variable x_i représente un noeud, son sous-arbre gauche représente l'assignation partielle sur la formule f lorsque cette variable est assignée à vrai, et son sous-arbre droit l'assignation partielle sur f lorsque x_i est faux.

Cette représentation est minimale, au sens où elle donne une forme normale disjonctive de la formule f. On comprend alors pourquoi les deux sous-arbres gauche et droit doivent être différents : si ce n'est pas le cas, la valuation de f est la même quelque soit la valeur de la variable associée au noeud que l'on considère.

$$\begin{aligned} test(g,g',g'') &= (g \wedge g') \vee ((\neg g) \vee g'') \\ &= (x_i \bigwedge((g_{i\leftarrow 1} \wedge g'_{i\leftarrow 1}) \vee ((\neg g_{i\leftarrow 1}) \wedge g''_{i\leftarrow 1}))) \\ &\quad \bigvee((\neg x_i) \bigwedge((g_{i\leftarrow 0} \wedge g'_{i\leftarrow 0}) \vee ((\neg g_{i\leftarrow 0}) \wedge g''_{i\leftarrow 0}))) \end{aligned}$$

2 Programmation des arbres de décision

Question 7

a

```
let abd_proj i=Test(i,Bool(true),Bool(false)) ;;
```

b

Par récurrence sur la hauteur h de l'arbre. Si $h = 1$, on a un arbre réduit à une feuille, et sa négation est trivialement bien formée. Si $h \geq 1$, on doit donner la négation de $test(x_i, v, f)$ bien formé. Pour cela, on exécute le même test (l'assignation des variables est identique), mais on donne la négation des conclusions. L'arbre reste bien formé car par hypothèse, les variables de v et f sont toutes d'indice supérieur à i, et les négations de v et f sont distinctes sans quoi v et f seraient identiques.

```
let rec abd_neg =function
| Bool(b)→Bool(not(b))
| Test(v,g,d)→Test(v,abd_neg g,abd_neg d) ;;
```

Question 8

a

```
let rec abd_egal = fun
| Bool(b1) Bool(b2) → b1=b2
| Test(v1,g1,d1) Test(v2,g2,d2) →
(v1=v2)&&(abd_egal g1 g2)&&(abd_egal d1 d2)
| _ _→false ;;
```

b

```
let rec abd_partiel i b=function
| Bool(bb) → Bool(bb)
| Test(v,g,d)→match (v>i,v=i) with
| (true,_)→Test(v,g,d)
| (_,true)→if b then g else d
| _→let (g1,d1)=(abd_partiel i b g,abd_partiel i b d)
   in if abd_egal g1 d1 then g1 else Test(v,g1,d1) ;;
```

c

```
let rac=function Bool(_)→max_int|Test(i,_,_)→i ; ;

let rec abd_test c v f=match c with
| Bool(b)→if b then v else f
| Test(i,gg,dd)→let m=min i (min (rac gg) (rac dd)) in
let c0=abd_partiel m false c
and c1=abd_partiel m true c
and v0=abd_partiel m false v
and v1=abd_partiel m true v
and f0=abd_partiel m false f
and f1=abd_partiel m true f
in let g1=abd_test c1 v1 f1
   and d1=abd_test c0 v0 f0
   in if abd_egal g1 d1 then g1 else Test(m,g1,d1) ; ;
```

d

```
let rec abd_et f g=match (f,g) with
| (Bool(true),_)→g
| (Bool(false),_) | (_,Bool(false))→Bool(false)
| (_,Bool(true))→f
| (Test(i,g1,d1),Test(j,g2,d2))→let m=min i j in
let g3=abd_et (abd_partiel m true f) (abd_partiel m true g)
and d3=abd_et (abd_partiel m false f) (abd_partiel m false g)
in if abd_egal g3 d3 then g3 else Test(m,g3,d3) ; ;

let abd_ou f g= let (gg,dd)=(abd_neg f,abd_neg g)
        in abd_neg(abd_et gg dd) ; ;

let abd_implique f g=abd_ou (abd_neg f) g ; ;
```