TME 6 : Représentation des fonctions Boolèennes : ROBDD

Objectif

L'objectif principal de ce TME est de vous familiariser avec la représentation des fonctions Booléennes sous forme de ROBDD. Les ROBDD (Reduced Ordered Binary Decision Diagram) sont utilisés pour représenter de façon compacte une ou plusieurs fonctions Booléennes, partageant le même support (c?est à dire dépendant d?un même ensemble de variables Booléennes). Pour un ordre donné des variables constituant le support, cette représentation est canonique : A chaque fonction Booléenne est associé un unique graphe orienté acyclique (DAG). Le graphe étant acyclique, chaque noeud BDD définit un sous-graphe dont il est la racine. Par conséquent, chaque noeud BDD correspond à une fonction Booléenne particulière. On utilise le fait que les variables Booléennes constituant le support sont ordonnées pour identifier ces variables par leur index.

Créez un répertoire TME6, et recopiez dans ce répertoire les fichiers qui se trouvent dans le répertoire

```
/users/enseig/encadr/cao/tme6
```

A) Structures de données et fonctions de base

La structure C permettant de représenter un noeud du graphe ROBDD est définie de la façon suivante :

```
typedef struct bdd_t {
        unsigned int            INDEX;
        struct bdd_t            *HIGH;
        struct bdd_t            *LOW;
} bdd_t
```

Pour construire et visualiser le graphe ROBDD, on disposes des fonctions suivantes

```
extern bdd_t  *create_bdd(unsigned int index, bdd_t *high, bdd_t *low)
extern bdd_t  *apply_bdd(unsigned oper, bdd_t *p1, bdd_t *p2)
extern bdd_t  *not_bdd(bdd_t *p)
extern void   display_bdd(bdd_t *p)
```

- La fonction **create_bdd()** gère un dictionnaire de tous les noeuds BDD déjà créés et fournit la garantie qu?il n?existera jamais deux noeuds BDD possédant les mêmes valeurs pour les champs INDEX, HIGH et LOW. La fonction create_bdd() recherche le noeud BDD possédant les valeurs définies par les arguments, et renvoie un pointeur sur le noeud BDD correspondant. Si ce noeud n?existe pas, cette fonction crée un nouveau noeud dans le dictionnaire, en affectant la valeur index au champs INDEX, la valeur high au champs HIGH, et la valeur low au champs LOW.
- La fonction **apply_bdd()** prend pour arguments un entier définissant un opérateur Booléen (les valeurs possibles sont OR, AND, et XOR), deux pointeurs p1 et p2 sur deux noeuds BDD représentant deux

fonctions Booléennes F1 et F2 (notons que les deux fonctions F1 et F2 ne dépendent pas forcément de toutes les variables Booléennes définies dans le support global). La fonction apply_bdd() construit le graphe ROBDD représentant la fonction F = F1 oper F2, et renvoie un pointeur sur le noeud ROBDD racine de ce graphe.

- La fonction **not_bdd()** prend pour unique argument un pointeur p sur un noeud BDD représentant une fonction Booléenne F, et renvoie un pointeur sur le noeud BDD représentant la fonction Booléenne NOT(F).
- La fonction **display_bdd()** prend pour unique argument un pointeur p sur un noeud BDD représentant une fonction Booléenne F, et affiche sur le terminal standard l?ensemble des noeuds BDD qui interviennent dans le graphe ROBDD représentant F. Il y a un noeud BDD par ligne, et chaque noeud BDD est décrit par quatre entiers :
    - ♦ N représente un numéro identifiant le noeud BDD
    - ♦ H représente le numéro du noeud BDD représentant le cofacteur HIGH.
    - ♦ L représente le numéro du noeud BDD représentant le cofacteur LOW.
    - ♦ X représente l?INDEX de la variable associée au noeud BDD

Pour la représentation des variables, vous utiliserez les mêmes fonctions que celles utilisées lors du TME5.

```
typedef struct var_t{
                char *NAME;
                unsigned INDEX;
                unsigned VALUE;
} var_t;

extern var_t *cons_var (char *name, unsigned index, unsigned value);
extern var_t *get_var_index (unsigned index);
extern var_t *get_var_name (char *name);
```

B) Représentation Graphique

Soit la fonction Booléenne F(a,b,c,d,e), définie par l?expression suivante

E0 = (a? . (b + c + d?) . e) + c

La notation a? signifie NOT(a).

E0 peut se re-écrire sous forme préfixée de la façon suivante :

E0 = OR(AND (NOT (a) OR (b c NOT (d) e) c)

B.1 En utilisant la décomposition de Shannon, représentez graphiquement le graphe ROBDD associé à la fonction F(a,b,c,d,e) pour l?ordre a > b > c > d > e , puis pour l?ordre e > d > c > b > a.

B.2 Précisez, pour chaque noeud du ROBDD ainsi construit, quelle est la fonction Booléenne représentée par ce noeud.

C) Fonctions not_bdd() et apply_bdd()

Les fonctions apply_bdd() et not_bdd() ont été présentées en cours. Vous trouverez le code de ces fonctions dans le fichier bdd.c.

C.1 Soient F1 et F2 deux fonctions Booléennes, et les fonctions F1H, F1L, F2H, F2L définies par la décomposition de Shannon suivant la variable x :

- F1 = x . F1H + x? . F1L
- F2 = x . F2H + x? . F2L

La relation de recurrence (F1 op F2) = x . (F1H op F2H) + x?. (F1L op F2L) a été démontrée en cours dans le cas des opérateurs OR et AND. Démontrez que cette relation est vraie dans le cas d?un opérateur XOR. Analysez le cas général, ainsi que les cas particuliers où l?une des deux fonctions F1 ou F2 est égale à une des deux fonctions constantes 0 ou 1.

C.2 Soit la fonction F, définie par l?expression E1 = a . (b + c) Représentez graphiquement le ROBDD représentant la fonction F, pour l?ordre des variables a > b > c. On appelle p0, p1, p2, p3, p4 les pointeurs sur les 5 noeuds BDD contenus dans ce graphe :

- p0 représente la fonction F0 = 0
- p1 représente la fonction F1 = 1
- p2 représente la fonction F2 = c
- p3 représente la fonction F3 = b + c
- p4 représente la fonction F4 = a . (b + c)

Représentez graphiquement l?arbre d?appels des fonctions lorsqu?on appelle la fonction not_bdd() avec pour argument le pointeur p4. Quel est le nombre maximum d?appels de fonctions empilés sur la pile d?exécution? Combien de noeuds BDD vont être créés par la fonction create_bdd(), si on suppose que la structure de données ne contient initialement que les 5 noeuds pointés par p0, p1, p2, p3, et p4 au moment de l?appel de la fonction not_bdd(p4)?

D) Fonction abl2bdd()

On cherche à écrire la fonction abl2bdd(), qui construit automatiquement le graphe ROBDD représentant une fonction Booléenne F, à partir d?un arbre ABL représentant une expression Booléenne particulière de cette fonction F.

```
bdd_t *abl2bdd(bip_t *p)
```

Cette fonction prend comme unique argument un pointeur sur la racine d?un arbre ABL, et renvoie un pointeur sur le noeud BDD représentant la fonction. Puisque les arbres ABL ne contiennent que des opérateurs OR, AND, XOR et NOT, et qu?on dispose des fonctions apply_bdd() et not_bdd(), il est possible de construire le graphe ROBDD par application récursive de ces deux fonctions.

D.1 Décrire en français, l?algorithme récursif de cette fonction abl2bdd() dans le cas particulier où tous les opérandes AND, OR ou XOR présents dans l?arbre ABL n?ont que deux opérandes,

D.2 Ecrire en langage C la fonction abl2bdd() dans le cas particulier de la question précédente. Pour valider cette fonction, on écrira un programme main(), qui effectue successivement les opérations suivantes :

- déclaration et indexation de toutes les variables Booléennes,
- construction de l?arbre ABL représentant l?expression E1, avec la fonction parse_abl(),
- affichage de cette expression Booléenne, avec la fonction display_abl(), pour vérifier que l?arbre ABL est correctement construit,
- construction du graphe ROBDD représentant la fonction F, avec la fonction abl2bdd(),
- affichage du graphe ROBDD ainsi construit, en utilisant la fonction display_bdd().

D.3 Comment faut-il modifier la fonction abl2bdd(), pour traiter le cas général, où les opérateurs OR, AND et XOR peuvent avoir une arité quelconque? Modifier le code de la fonction, et validez ces modifications sur l?exemple de l?expression Booléenne E0, ou sur des expressions encore plus complexes.

E) Fonction satisfybdd()

Soit une fonction Booléenne quelconque F(x1, x2, x3, ... xn), dépendant de n variables. Soit la fonction Booléenne G, définie comme un produit (opérateur AND) d?un nombre queconque de variables xi (directes ou complémentées). On dit que G ?satisfait? F sion a la relation G => F. (Autrement dit, si G = 1, alors F = 1). Remarquez que la condition G = 1 impose la valeur de toutes les variables appartenant au support de G (les variables directes doivent prendre la valeur 1, et les variables complémentées doivent prendre la valeur 0). Pour une fonction F donnée, il existe évidemment plusieurs fonction G qui satisfont F...

Exemple : F = (a . b) + c

On peut avoir G = c ou G = a . b ou encore G = a . b . c?

Dans la suite de cet exercice, on cherche à écrire une fonction C qui construit automatiquement le ROBDD représentant une fonction G satisfaisant une fronction F quelconque. Comme il existe plusieurs solutions, on choisira la systématiquement la solution qui minimise le nombre de variable xi dans le support de G.

```
bdd_t *satisfy_bdd(bdd_t *p)
```

La fonction satisfy_bdd() prend pour argument un pointeur sur le noeud BDD représentant la fonction F et renvoie un un pointeur sur le noeud BDD représentant la fonction G.

E.1 La décomposition de Shannon de la fonction F suivant la variable x d?index le plus élévé définit les cofacteurs FH et FL : F = x . FH + x? . FL Pour alléger les notations, on note sat(F) la fonction Booléenne construite par la fonction satisfy_bdd() pour la fonction F. Donner la relation de récurrence entre les fonctions sat(F), sat (FH), et sat(FL). On étudiera successivement le cas général où ni FH et FL ne sont constantes, et les quatre cas particuliers où l?une des deux fonctions FH ou FL sont égales à 0 ou 1.

E.2 Ecrire, en langage C, le code de la fonction satisfy_bdd(), et appliquez-la sur la fonction Booléenne F définie dans la partie, en modifiant le programme main de la fonction précédente.

F) Fonction bdd2abl()

On souhaite pour finir écrire la fonction bdd2abl(), qui construit automatiquement un arbre ABL représentant une expression Booléenne multi-niveaux, à partir d'un ROBDD représentant une fonction Booléenne.

```
bip_t *bdd2abl(bdd_t *p)
```

Cette fonction prend comme unique argument un pointeur sur un noeud BDD, et renvoie un pointeur sur le bipointeur représentant la racine de l?arbre ABL représentant l?expression Booléenne.

Il existe évidemment un grand nombre d?expressions Booléennes équivalentes pour une même fonction Booléenne. Dans un premier temps, nous nous contenterons d?utiliser des opérateurs OR, et AND à deux opérandes, ainsi que l?opérateur NOT. Soit un noeud BDD représentant une fonction Booléenne F. Soient x la variable associée à ce noeud BDD, FH et FL les fonctions Booléennes associées aux noeuds BDD pointés par les pointeurs HIGH et LOW (cofacteurs de Shannon).

F.1 Quelle expression Booléenne dépendant de x, FH et FL peut-on associer à la fonction F dans le cas général ou FH et FL sont quelconques ? (aucune des deux fonctions FH ou FL n?est égale à 0 ou à 1)

F.2 Comment cette expression Booléenne se simplifie-t-elle lorsque l?une des deux fonctions FH ou FL est égale à 0 ou à 1. Etudier un par un les 4 cas particuliers.

F.3 En utilisant les résultats des questions précédentes, proposez un algorithme de construction de l?arbre ABL

F.4 Ecrire en langage C la fonction bdd2abl(), et validez-la sur différents exemples, en utilisant la fonction display_abl().

Compte-Rendu

Il ne vous est pas demandé de compte-rendu écrit pour ce TME, mais vous devrez faire une démonstration de votre code au début du prochain TME.