Master SESI [M1]
UE CAO – Rattrapage

Jean-Paul Chaput

20 juin 2010

Durée : 2 heures – Tous documents autorisés

Le barème est donné à titre indicatif et peut être modifié par le correcteur

Écrivez lisiblement, un texte difficilement déchiffrable sera toujours considéré comme faux

Représentation d'une expression par un ABL

Un ABL est une structure inspirée du LISP représentant une expression arithmétique et permettant d'en faire une évaluation rapide. Dans la modélisation traditionnelle d'un ABL chaque noeud est un bi-pointeur. Le premier pointeur est le `CAR`, il pointe sur une donnée, le second est le `CDR` il est assimilable à un pointeur *next*. Dans notre implémentation nous ne suivrons pas strictement cette règle : il est clair que les noeuds terminaux (ou atomes) n'ont pas besoin de `CDR` et que le `CAR` pointe sur un type de donnée (ici on prendra des `strings`).

Dans un ABL on distingue trois types de noeuds :

1. Les atomes : ce sont les noeuds terminaux du graphe. Ils n'ont ni membre `CAR` ni `CDR`, mais un membre de type `string`. Ce membre indique soit le nom d'une variable, soit le type d'opération arithmétique à effectuer.
2. Les expressions : ces noeud représentent une expression complète. Leur `CAR` pointe sur un atome qui donne le type de l'opération et leur `CDR` sur la liste des opérandes. Cette liste est construite en utilisant des noeuds "opérandes". Une expression peut apparaître dans l'ABL à la place d'un atome "nom de variable".
3. Les opérandes : ils servent à construire les listes d'opérandes. Ils disposent de `CAR` et `CDR`.

1 Implémentation des ABL

Question 1 **2pt**

En plus des trois types (ou classes `Atom`, `Expression` et `Operand`) de noeuds précédemment décrits on ajoute, pour l'implémentation en C++ une classe de base `ABL`.

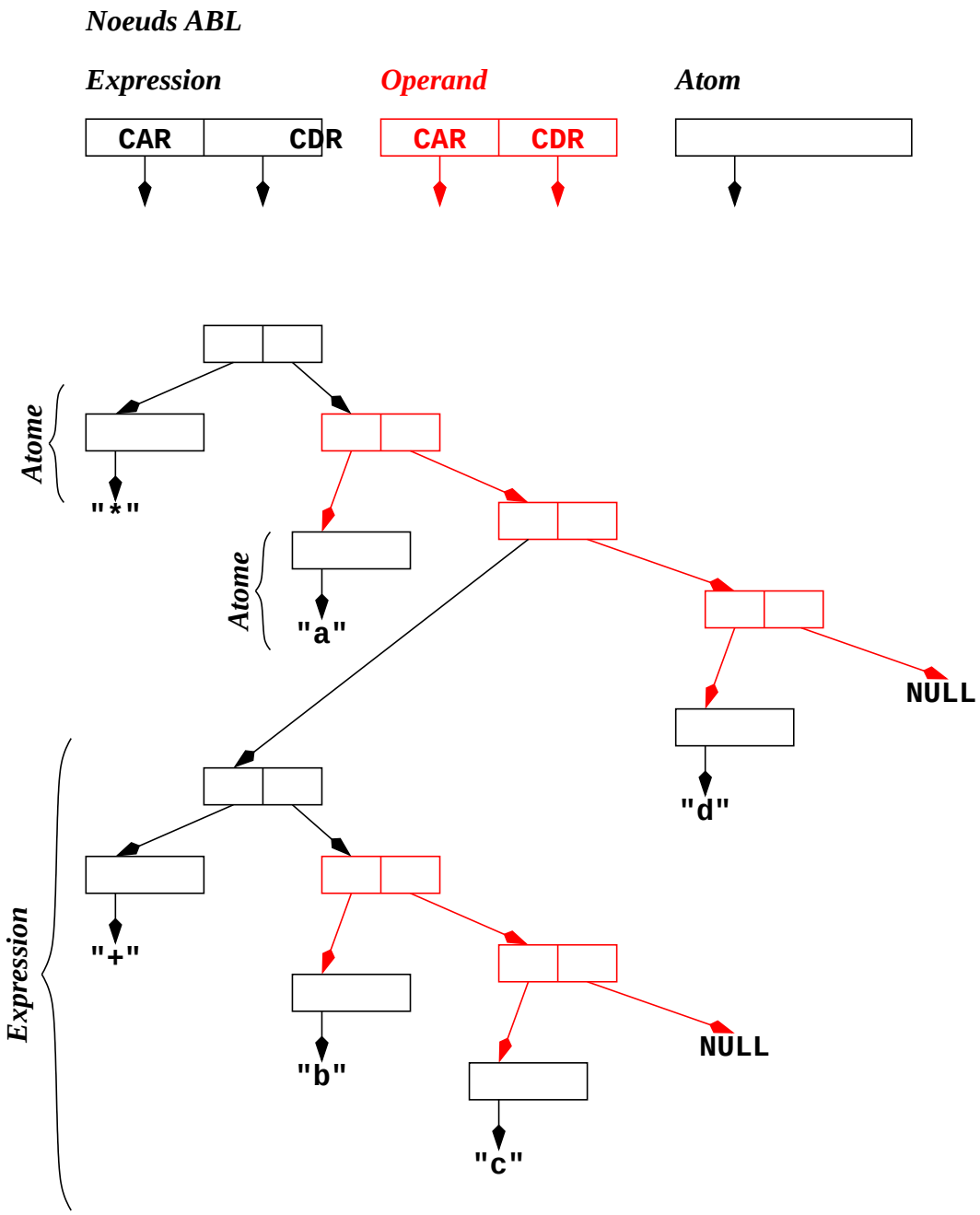

FIGURE 1 – Structure d'un ABL : $(\times\ a\ (+\ b\ c)\ d)$

En vous basant sur la présence (ou l'absence de membres) proposer un arbre d'héritage pour ces différentes classes.

Question 2 **4pt**

Proposer une implémentation de la classe `ABL`, en respectant les contraintes suivantes :

1. On ne devra pas pouvoir instancier d'objets de la classe `ABL`, expliquer votre méthode.
2. Pour chacun des membres potentiel `CAR` et `CDR` fournir une fonction `CAR()` accédant au membre et une seconde permettant de positionner sa valeur. Ces fonctions peuvent-elles avoir le même nom, si oui expliquez pourquoi.
3. Prévoir une fonction `getString()` renvoyant une `string`.

Question 3 **4pt**

Écrire les classes dérivées `Atom` et `Operand` (déclaration et définitions).

Les "valeurs" qui seront stockées dans les atomes sont des chaînes de caractères (`string`). Donc des noms de variables.

Afin de pouvoir par la suite convertir un ABL en BDD, il faut être capable s'associer un index unique (qui servira d'ordre de décomposition pour le BDD) et une variable. On ajoutera donc à la classe `Atom` la mécanique nécessaire pour créer des identifiants uniques et une table (`map`) associant un nom de variable à un index. Cette table sera partagée par tous les objets de la classe `Atom`.

Question 4 **4pt**

Implémentation de la classe `Expression`. Le type d'opération réalisé par l'expression sera fixé une fois pour toute à l'appel du constructeur. Les opérandes seront ajoutées une à une via une fonction membre dédiée.

La destruction d'une expression devra entraîner la destruction de tout l'arbre dont elle est la racine.

On introduit donc la fonction membre suivante :

1. `addOperand()` : ajoute un opérande à l'expression. Cet opérande peut être soit un pointeur sur une expression, soit un nom de variable. Tous les objets intermédiaires nécessaire au chaînage devront être crées automatiquement.

Pour la fonction `getString()`, on adoptera la représentation préfixée suivante :

$$(\times\ a\ b\ (+\ c\ d))$$

Peut-on se contenter du destructeur par défaut ? Justifiez votre réponse.

Question 5 **1pt**

En vous appuyant sur la fonction `getString()`, écrire une fonction générique d'affichage dans les flux pour les différents types de noeuds. Cette fonction a-t-elle besoin d'être `friend` ?

Question 6 **5pt**

Implémentez, dans la class `Expression`, une méthode `toBdd()` transformant une expression ABL complète en BDD. On s'inspirera de ce qui a été présenté en TME pour les EBM.

On rapelle l'interface de la classe BDD :

```
enum OperatorType { UndefinedOperator=0, Not, And, Or, Xor };

class Bdd {
  private:
           unsigned int       _ident;    // Unicity identifier.
           unsigned int       _index;    // Decomposition variable index.
           Bdd*               _high;     // Low cofactor pointer.
           Bdd*               _low;      // High cofactor pointer.
  public:
    static Bdd*               ConstHigh;
    static Bdd*               ConstLow;
  private:
                              Bdd          ( unsigned index, Bdd* high, Bdd* low );
                             ~Bdd          ();
  public :
    static Bdd*               create       ( unsigned index, Bdd* high, Bdd* low );
    static Bdd*               apply        ( OperatorType, Bdd*, Bdd* );
  public:
    inline unsigned           getIdent     ();
    inline unsigned           getIndex     ();
    inline Bdd*               getHigh      ();
    inline Bdd*               getLow       ();
};
```