Maîtrise d'Informatique **J.Bétréma**
Compilation (tronc commun) **Travaux dirigés**

Algorithme de construction d'un automate à partir d'une expression rationnelle

1 Description de l'algorithme

Soit r une expression rationnelle, construite à partir des lettres de l'alphabet, de l'opérateur d'union (noté $|$), de l'opérateur de concaténation, de l'opérateur étoile (noté $*$), et de parenthèses; les opérateurs sont munis des priorités habituelles. A chaque expression r correspond un unique *arbre syntaxique* : chaque noeud interne est étiqueté par un opérateur, et le sous-arbre issu de ce noeud définit une *sous-expression* de r.

On numérote, de 1 à n, les occurrences des *lettres* dans r, on appelle *positions* ces numéros, et on note x_i la lettre en position i; par exemple l'expresssion $r = (a|ab|ba)^*$ possède 5 positions, et :

$$x_1 = a,\ x_2 = a,\ x_3 = b,\ x_4 = b,\ x_5 = a\ .$$

Soit L le langage dénoté par r; un mot w dans L correspond à une suite de positions (en général cette suite n'est pas unique); par exemple, ici, le mot aba correspond aux suites de positions $1, 4, 5$ ou $2, 3, 1$. N'importe quelle suite de positions n'est pas admissible pour un mot de L : par exemple, ici, la position 2 doit toujours être suivie de 3, et la position 3 ne peut être suivie que par $1,\ 2$ ou 4.

Le but de cet algorithme est de construire la fonction *suivant*, qui caractérise, pour chaque position i, la liste des positions qui peuvent suivre i dans un mot de L; on obtient ainsi un automate non déterministe, dont les états sont les positions, et qui reconnaît L : cet automate comporte une transition $i \to j$, étiquetée x_j, pour chaque position j dans *suivant(i)* (la description de cet automate est volontairement incomplète).

Le calcul de *suivant* passe par le calcul de trois autres fonctions, définies pour toute sous-expression de r (et donc pour tout noeud interne de l'arbre syntaxique) :

- *vide*, qui indique si le langage dénoté par une expression, contient le mot vide;
- *premier*, qui donne la liste des positions initiales admissibles pour les mots du langage dénoté par une expression; pour l'exemple introductif, *premier* $(r) = \{1, 2, 4\}$.
- *dernier*, qui donne la liste des positions finales admissibles pour les mots du langage dénoté par une expression; pour l'exemple introductif, *dernier* $(r) = \{1, 3, 5\}$.

1.1 Calcul de *vide, premier* et *dernier*

Ces fonctions peuvent être calculées récursivement, à partir de leurs valeurs évidentes sur les lettres de l'alphabet, comme suit (les formules pour *dernier* sont symétriques de celles pour *premier*) :

- *vide* $(r_1|r_2)$ = *vide* (r_1) OU *vide* (r_2);

  premier $(r_1|r_2)$ = *premier* (r_1) UNION *premier* (r_2);

- *vide* $(r_1 r_2)$ = *vide* (r_1) ET *vide* (r_2);

  premier $(r_1 r_2)$ =

  SI *vide* (r_1) ALORS *premier* (r_1) UNION *premier* (r_2)
  SINON *premier* (r_1) ;

- *vide* (r^*) = VRAI;

  premier (r^*) = *premier* (r) .

1.2 Calcul des transitions

- pour toute sous-expression de la forme $r_1 r_2$, pour toute position i dans *dernier* (r_1), et pour toute position j dans *premier* (r_2), j appartient à *suivant* (i);

- pour toute sous-expression de la forme r_1^*, pour toute position i dans *dernier* (r_1) et pour toute position j dans *premier* (r_1), j appartient à *suivant* (i) .

2 Exercices

1. Exécuter cet algorithme sur l'expression $(b|a^*c)^*a^+b$.

   Note : on complètera l'algorithme ci-dessus pour l'opérateur $+$.

2. Quels sont les états initiaux et terminaux? Cette question comporte un piège : vérifier que votre réponse est correcte, par exemple, pour les langages b et a^+c.

3. Déterminiser et minimiser l'automate obtenu.

4. L'expression $(b|a^*c)^*a^+b$ dénote le langage L des mots w qui se terminent "pour la première fois" par ab (ie. ab est suffixe de w, et il n'y a pas d'autre occurrence du facteur ab dans w). En considérant qu'un mot $w \in L$ s'écrit : $w = uab$, et que dans u, tout caractère a est suivi du caractère a ou c (mais pas de b), on est conduit à proposer l'expression $(b|c|a(a|c))^*ab$ pour L. Examiner si cette proposition est correcte en appliquant à nouveau l'algorithme à cette expression, puis en déterminisant et minimisant l'automate obtenu.