

Master Ingénierie Informatique.

Année 2004-2005.

Automates avancés — Projet.

## Modalités

Les sources du projet ainsi qu'un court rapport (5 à 10 pages) sont à transmettre par mail pour le 16/5/2005. Le langage de programmation peut être au choix C, Caml ou java.

## 1 Dictionnaires

Ce sujet est à réaliser à deux étudiants. Pour représenter de façon compacte un ensemble de mots (destiné par exemple à être utilisé par un correcteur orthographique), on utilise un automate acyclique. On se place sur l'alphabet [a-z], sans caractères accentués ou non alphabétiques, sans distinguer majuscules et minuscules. Travail demandé :

- Définir un type permettant de représenter les automates.
- Implémenter la minimisation des automates acycliques en temps linéaire (par rapport au nombre de transitions).
- Écrire une fonction permettant l'ajout d'un mot à l'ensemble représenté par un automate. Écrire de même une fonction permettant la suppression.
- Écrire une fonction construisant l'automate minimal correspondant à d'une liste de mots contenus dans un fichier. Comparer l'utilisation mémoire et le temps d'exécution de l'approche construisant d'abord un arbre, puis qui le minimise, et celle minimisant l'automate après chaque ajout de mot.
- Définir un nouveau type d'automate dans lesquels les transitions peuvent être étiquetées par des mots. Écrire une fonction construisant un tel automate à partir d'un automate habituel, en compressant les chemins dont les états intermédiaires ne sont entrée (resp. sortie) que d'une transition.
- Écrire le test d'appartenance d'un mot à un tel automate.
- Écrire un vérificateur d'orthographe prenant en argument un fichier texte et un lexique compressé, et fournissant la liste des mots incorrects du texte.

## 2 Logique temporelle

Ce projet est à réaliser individuellement. On veut pouvoir vérifier des propriétés de logique temporelle linéaire LTL sur des automates de Büchi. On

rappelle la sémantique de cette logique sur les mots infinis. Soit  $u = u_0u_1 \dots$  où les  $u_i$  sont des lettres.

- $u, i \models a$  si  $u_i = a$  ;
- $u, i \models \alpha \vee \beta$  si  $u, i \models \alpha$  ou  $u, i \models \beta$  ;
- $u, i \models \neg\alpha$  si l'on n'a pas  $u, i \models \alpha$  ;
- $u, i \models X\alpha$  si  $u, i + 1 \models \alpha$  ;
- $u, i \models \alpha \cup \beta$  s'il existe un entier  $j$  tel que  $u, j \models \beta$  et pour tout  $k$  tel que  $i \leq k \leq j - 1$ , on a  $u, k \models \alpha$ .

Le travail demandé est le suivant.

- Définir un type pour représenter les automates de Büchi généralisés.
- Écrire une fonction de conversion d'une formule LTL en automate de Büchi.
- Écrire une fonction testant si un automate de Büchi donné satisfait une spécification LTL.

### 3 Bibliothèque d'automates et lex

Ce projet est à réaliser par groupes de 3 étudiants au plus. Il demande de définir les types nécessaires de d'implémenter les algorithmes vus en cours sur les automates et expressions rationnelles.

- Construction d'un automate à partir d'une expression rationnelle par l'algorithme de Glushkov.
- Construction d'une expression rationnelle à partir d'un automate en utilisant les équations.
- Détermination d'automates.
- Minimisation : algorithmes de Moore, Hopcroft et Brzozowski.
- Test d'équivalence d'expressions.
- Implémentation efficace en mémoire de l'automate du langage  $A^*x$  permettant de reconnaître le motif  $x$  dans un texte.

Utiliser ensuite la partie précédente pour réaliser une implémentation partielle de `lex` supportant :

- les constructeurs usuels d'expressions. Les contextes à droite (fin de ligne, opérateur /) ou à gauche (début de ligne) pourront ou non être supportés.
- les conditions de départ `%s` et `%x`.

On pourra utiliser `lex` lui-même pour analyser le fichier contenant les expressions. Dans ce cas, vous pourrez tester l'analyseur écrit en l'utilisant à la place de `lex`.