TP n° 7 : Reconnaissance non déterministe

Vous trouverez sur ma page Web (`http://www.pps.jussieu.fr/~delatail/AF3.html`) une correction des premiers TPs sur les automates. Le fichier « `correctionLE.java` » donne donne la fonction de reconnaissance et les algorithmes de lecture et écriture dans un fichier. *Vous pouvez récupérer ce code, mais il est indispensable que vous ayez compris le fonctionnement de la fonction de reconnaissance et la construction des automates donnés en exemple, sans quoi la suite des TPs ne sera qu'un long et douloureux calvaire.* Il est préférable d'avoir aussi compris l'idée générale des fonctions de lecture et d'écriture dans un fichier, mais il n'est pas indispensable d'en comprendre tous les détails.

NOTATION : A la fin du TD, vous m'enverrez vos fichiers dans un mail à l'adresse `delatail@pps.jussieu.fr`. N'oubliez pas les fichiers auxiliaires, ainsi que les fichiers codant les automates pris en exemple. Les fonctions demandées sont celles que je vous présente ci-dessous, mais la présence de fonctions codées dans les précédents TPs (standardisation, complété, automate produit, complémentaire, miroir...) sera bien évidemment appréciée. Si vous n'avez pas le temps de terminer le code d'une fontion, vous pouvez la laisser en la mettant en commentaires.

1 Reconnaissance non déterministe

La structure de données que nous utilisons pour représenter un automate permet d'implémenter aussi bien des automates non déterministes (avec plusieurs états initiaux et plusieurs transitions portant la même lettre et partant d'un même état) que déterministes ; par contre, notre fonction de reconnaissance ne marche que pour les automates déterministes. L'objectif de cette section est d'implémenter la reconnaissance d'un mot par un automate non déterministe.

Exercice 1 :

Dans votre classe `Automate`, ajoutez une méthode « `boolean reconnAux(Etat q, String s)` » qui reconnaît *de manière récursive* si un mot `s` peut être lu à partir de l'état `q`.

Exercice 2 :

Écrivez une méthode « `boolean reconnaissanceND (String s)` » qui implémente la reconnaissance dans un automate non déterministe.

Exercice 3 :

Testez vos méthodes sur les exemples suivants :

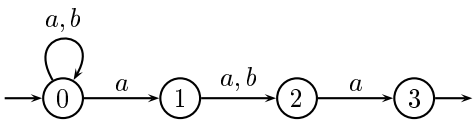

FIG. 1 – Automate $\mathcal{A}_1$

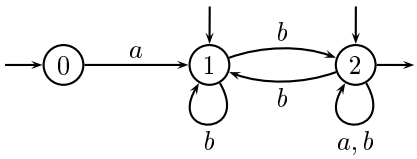

FIG. 2 – Automate $\mathcal{A}_2$

On implémentera le premier exemple "à la main", et le deuxième exemple à partir d'un fichier.

2 Déterminisation

Notre but dans cette section est d'implémenter l'algorithme de déterminisation.

Afin d'y arriver en douceur, on propose d'abord un autre algorithme de reconnaissance non déterministe, similaire au cas déterministe sauf que l'on navigue sur une liste d'états au lieu d'un seul état.

Autrement dit, on part de la liste des états initiaux, on en déduit la liste des états accessibles en lisant la première lettre du mot, puis on en déduit la liste des états accessibles en lisant la deuxième lettre du mot, etc. Une fois le mot lu en entier, il est reconnu si la liste des états obtenue au final contient au moins un état terminal.

Exercice 4 :

Définissez une nouvelle classe « `ListeEtats` » qui implémente une liste d'états. Implémentez la fonction « `static ListeEtats concat(ListeEtats l1,ListeEtats l2)` » qui concatène deux listes d'états.

Exercice 5 :

Dans votre classe `Automate`, ajoutez une méthode qui retourne la liste des états initiaux.

Exercice 6 :

Dans votre classe `Etat`, ajoutez une méthode « `ListeEtats cibleND(char c)` » qui retourne la liste des états accessibles par une transition portant la lettre `c`.

Exercice 7 :

Dans votre classe `Automate`, ajoutez une méthode « `boolean reconnaissanceNDbis (String s)` » qui implémente le nouvel algorithme de reconnaissance non déterministe.

L'algorithme de déterminisation fonctionne sur la même idée, mais cette fois-ci il faut être capable de dire si une liste d'états a déjà été atteinte. On construira donc une fonction « `static boolean compare(ListeEtats l1, ListeEtats l2)` » qui dit si les deux listes `l1` et `l2` contiennent les mêmes états. On pourra aussi ajouter à la classe `Automate` une fonction « `Alphabet alph()` » qui génère l'alphabet des lettres utilisées par l'automate.

Exercice 8 :

Dans votre classe `Automate`, ajoutez une méthode « `Automate determinise()` » qui retourne l'automate déterminisé. On pourra procéder en deux passes : une pour déterminer le nombre d'états du nouvel automate, une autre pour générer les transitions proprement dites.