TP n° 9 : Déterminisation

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 :

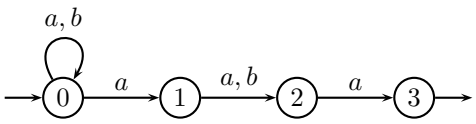

FIG. 1 – Automate $\mathcal{A}_1$

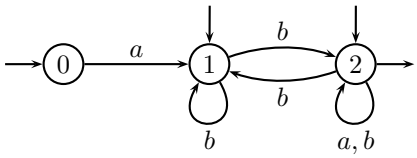

FIG. 2 – Automate $\mathcal{A}_2$

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 à partir de l'état par une transition portant la lettre `c`.

Exercice 7 :

Dans votre classe `ListeEtat`, ajoutez une méthode « `ListeEtats cibleND(char c)` » qui retourne la liste des états accessibles à partir d'un des états de la liste par une transition portant la lettre `c`.

Exercice 8 :

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.

Exercice 9 :

Construisez une fonction « `static boolean compare(ListeEtats l1, ListeEtats l2)` » qui dit si les deux listes `l1` et `l2` contiennent les mêmes états.

Exercice 10 :

Ajoutez à la classe `Automate` une fonction « `Alphabet alph()` » (ou « `char[] alph()` ») qui génère l'alphabet des lettres utilisées par l'automate.

Pour construire le déterminisé d'un automate à n états, on procédera de la façon suivante :
- on construit un tableau de liste d'états, de taille 2^n, et on place dans la première case du tableau la liste des états initiaux
- on génère successivement chaque liste d'états accessible à partir d'une liste d'états déjà obtenue et d'une lettre de l'alphabet : si cette nouvelle liste d'états n'a pas déjà été obtenue auparavant on l'ajoute dans le tableau, sinon on ne fait rien
- on construit un nouveau tableau, cette fois de la bonne taille et contenant des états plutôt que des listes d'états, qui correspondent chacun à une liste d'états du premier tableau
- on construit les transitions adéquates entre états.

Exercice 11 :

Dans votre classe `Automate`, ajoutez une méthode « `Automate determinise()` » qui retourne l'automate déterminisé.