Rapport de projet : StartAutomaton

Nicolas Endredi, Baptiste Oruezabal

27 mars 2013

Table des matières

0.1 Présentation du projet

0.1.1 Automates et expressions rationnelles

Les expressions rationnelles sont une manière très simples de décrire un langage, notamment des langages définis récursivement. Cependant cette représentation n'est pas pratique pour implémenter des programmes analysant le langage. On utilise donc des automates, créés à partir de ces expressions. On sait, d'après le théorème de Kleene, qu'un langage est reconnu par un automate si et seulement si il peut être décrit par une expression rationnelle. Ainsi toutes les expressions rationnelles traitées durant ce projet, ont un automate correspondant.

0.1.2 La bibliothèque Automaton

La bibliothèque Python "Automaton" implémente l'objet "automaton" qui est un automate simple : on peut lui définir son alphabet, ses états, ses états (notamment initiaux et finaux) et ses transitions.
Pour représenter les "epsilon transitions" , on peut définir des caractères qui seront traités comme des "epsilon".

0.1.3 Sujet du projet

Le but du projet est d'ajouter des fonctionnalités à la bibliothèque "Automaton" en utilisant des algorithmes vus dans le cours d'Informatique Théorique. Ces fonctionnalités sont des manipulations d'automate ainsi que la conversion d'une expression rationnelle en automate

0.2 Fonctionnalités de StartAutomaton

Ici sont listées les différentes fonctionnalités implémentées dans StartAutomaton. Celles demandées dans le sujet apparaissent dans la table des matière.

0.2.1 Affichage

Afficher l'alphabet de l'automate

Affiche un "pretty set" contenant l'alphabet de l'automate

Afficher toutes les transitions de l'automate

Affiche un "pretty set" contenant toutes transitions de l'automate

Afficher tous les états de l'automate

Affiche un "pretty set" contenant tous les états de l'automate

Afficher les caractères epsilon

Affiche un "pretty set" contenant les caractères qui représentent epsilon dans l'automate

Afficher les états finaux

Affiche un "pretty set" contenant tous les états finaux de l'automate

Afficher les état initiaux

Affiche un "pretty set" contenant tous les états initiaux de l'automate

0.2.2 Tests

Tester si l'automate est déterministe

On renvoie un booléen signifiant si l'automate est déterministe ou non

Tester si l'automate est complet

On renvoie un booléen signifiant si l'automate est complet ou non

0.2.3 Sélecteurs et éditeurs

Obtenir l'origine d'une transition

On renvoie l'état à l'origine d'une transition donnée

Obtenir la destination d'une transition

On renvoie l'état de destination d'une transition donnée

Obtenir les epsilon transitions de l'automate

On renvoie la liste des transitions avec un caractère codant l'epsilon

Supprimer un état initial donné de l'automate

On enlève l'état de la liste des états initiaux de l'automate (ne supprime pas l'état de la liste d'états et ne supprime pas les transitions dans lesquelles il est présent)

Supprimer un état final donné de l'automate

On enlève l'état de la liste des états finaux de l'automate (ne supprime pas l'état de la liste d'états et ne supprime pas les transitions dans lesquelles il est présent)

Supprimer un transition donnée de l'automate

On enlève la transition de la liste des transitions de l'automate (ne supprime pas les états de la liste d'états)

Supprimer tous les états initiaux de l'automate

On vide la liste des états initiaux de l'automate

Supprimer tous les états finaux de l'automate

On vide la liste des états finaux de l'automate

Supprimer toutes les transitions de l'automate

On vide le dictionnaire stockant les transitions

Supprimer toutes les epsilon transitions (avec réagencement des autres transitions)

On supprime les transitions par un caractère encodant epsilon, sans pour autant changer le langage reconnu par l'automate

0.2.4 Manipulation d'automates

Compléter

On construit un état puits. Ensuite on ajoute pour chaque état les transitions par les lettres de l'alphabet qu'il lui manque. Ces transitions se font vers l'état puits.

Déterminiser

On fusionne des états de l'automate afin que tous les états n'aient pas deux transitions sortantes avec la même et qu'il n'y ait qu'un seul état initial

Miroir

On inverse les état finaux et initiaux, puis on renverse les transitions

Union de deux automates

On fait l'union de deux automates déterminisés (si non déterministes) et complétés (si non complets)

Intersection de deux automates

On fait l'intersection de deux automates déterminisés (si non déterministes) et complétés (si non complets)

Complément

On crée l'automate permettant de compléter l'automate original

Minimiser

On minimise l'automate. On crée donc le plus petit automate reconnaissant le même langage que l'automate d'origine

0.2.5 Conversions et utilitaire

Conversion d'une expression rationnelle en automate

On créé l'automate permettant de reconnaître le langage décrit par l'expression rationnelle donnée

Reconnaitre l'opérateur et l'action à réaliser pour une expression

Si l'expression commence par un opérateur, on créé un sous automate avec une des fonctionnalités ci-dessus.
Sinon, on créé juste une transition depuis l'état donné en paramètre vers un nouvel état, par la lettre lue

Créer un sous automate à partir d'un "+" lu précédemment dans l'expression

On créé un sous automate à partir de l'expression en paramètre et renvoie tous les états finaux construits par ce sous automate

Créer un sous automate à partir d'une "*" lue précédemment dans l'expression

On créé un sous automate à partir de l'expression en paramètre et renvoie l'état initial comme état final

Créer un sous automate à partir d'une "." lu précédemment dans l'expression

On créé un sous automate à partir de l'expression en paramètre et renvoie le dernier état final construit par ce sous automate

Renverser un tuple à trois éléments

On inverse le premier et le dernier élément sans toucher au deuxième

0.3 Problèmes rencontrés

0.3.1 La bibliothèque Automaton

Le temps nécessaire à la compréhension de toutes les méthodes implémentées par la bibliothèque Automaton a été considérable. Notamment pour comprendre les types d'objets utilisés pour stocker les propriétés d'un automate.

De plus la méthode "remove_epsilon_transitions()" était ambiguë : elle supprimait les caractères encodant epsilon, sans modifier les transitions elles-mêmes.
Nous avons donc décidé de la redéfinir, pour qu'elle supprime les epsilon transitions, sans modifier le langage reconnu par l'automate.
Nous avons aussi ajouté la méthode "remove_epsilons()" pour supprimer les caractères représentant epsilon.

0.3.2 Méthodes de manipulation d'automates

Déterminiser

Cette méthode étant la base de nombreux algorithmes de manipulation d'automates, elle nous a demandé beaucoup plus de temps et de réflexion. Elle est aussi celle qui a nécessité le plus de tests.

Minimiser

Une première version de minimiser a été tentée sans utiliser le double renversement, mais cette version de la méthode ne fonctionnait pas pour tous les automates. Nous l'avons donc abandonnée au profit du double renversement, plus clair et plus simple à implémenter

Conversion d'une expression rationnelle vers un automate

Cette méthode a été rapide à coder, mais longue à concevoir.
Nous avons étudié l'algorithme de Thomson et essayé de concevoir un algorithme, qui n'aurait pas besoin d'autant d'epsilon transitions.
Malgré de nombreux essais, la version définitive semble correctement fonctionner pour tous les cas que nous avons testés.

Conversion d'une expression rationnelle en préfixée

Par manque de temps, nous avons préféré ne pas implémenter cette fonctionnalité, pour nous concentrer sur la re-factorisation de notre code et sa documentation.

0.4 Projection

0.4.1 Implémentation de la conversion d'une expression rationnelle en préfixée

L'implémentation de cette fonctionnalité devrait prendre moins d'une semaine selon nos calculs. Ainsi nous essaierons de l'implémenter avant la soutenance de ce projet

0.4.2 Création de classes utilisables par StartAutomaton

Lors de la re-factorisation du code, nous avons remarqué que certaines actions méritent d'avoir une classe propre pour augmenter la lisibilité du code

0.5 Conclusion

Presque toutes les fonctionnalités demandés ont été implémentées avec un code bien lisible et des fonctionnalités utiles ont été ajoutées. De plus la plus part des méthodes on été factorisées afin d'en améliorer la lisibilité. Ainsi nous pensons avoir respecté les demandes du sujet.
Cependant, on a pu constater qu'il y a de nombreuses optimisations à faire dans les algorithmes que ce soit au niveau du temps ou de la mémoire utilisée, mais, le temps ne nous le permettant pas, nous laissons cela au soin d'autres développeurs qui voudraient améliorer notre classe StartAutomaton