PEREZ Thomas
LARBI YOUCEF Mohamed Reda

Conception pratique d'algorithmes
Egrep

I- Introduction

On souhaite implémenter pour ce projet la commande *egrep*. Celle ci permet de rechercher dans un flux de texte, les lignes contenant l'expression régulière **étendue** donnée en paramètre. Celle-ci sera faite en C++ pour l'implémentation des automates, ainsi que les outils *flex* pour l'analyse lexicale des expressions régulières, et *bison* pour l'analyse syntaxique.

II- Pourquoi le C++ ?

On souhaite utiliser pour ce projet un bon langage, adapté pour les programmes nécessitant de bonnes performances (étant donné la complexité de la construction de l'automate), orienté objet pour avoir une bonne organisation du projet, et possédant des outils pratiques (notamment les hash map) pour simplifier le développement. Le C++ répond à toutes ces contraintes.

III- Organisation du projet

Notre projet se compose en trois grandes parties : l'analyse lexicale, l'analyse syntaxique, et l'implémentation des automates.

Concernant l'analyse syntaxique, on utilisera la grammaire de l'*opengroup*, en ne prenant que la partie des expressions régulières **étendues**, et les brackets expressions.

Concernant l'implémentation des automates, on décomposera cela en trois modules : Un module *State*, implémentant les états. Un module principale *Automata* implémentant la structure d'automate, ainsi que plusieurs fonctionnalités comme la déterminisation, la transposion etc, et enfin un module *AutomataBuilder* permettant de construire des automates à l'aide de fonctionnalités comme la concaténation d'automates, l'étoile de

Kleene, l'union etc. A tout cela, on ajoute un module *AutomataPrinter* permettant d'afficher les automates (utiles pour les tests). On associera à chacun de ces module un module de test (par exemple *testAutomata* pour le module *Automata*).

On ajoute à tout ces modules un module *Tools*, possédant plusieurs fonctionnalités utilitaires, utile notamment pour le parser (par exemple, parser une classe d'équivalence en automate).

IV- Comment utiliser notre commande egrep ?

Le dossier du rendu est composé des dossiers suivants :

- bin/ Contenant tout les binaires (tests, ainsi que la commande *egrep*)
- obj/ Contenant tout les fichiers objets
- src/ Contenant tout les fichiers sources
- include/ Contenant les fichiers d'entêtes
- lexer/ Contenant l'analyse lexical
- parser/ Contenant l'analyse syntaxique
- Rapport/ Contenant le rapport

A ce la on ajoute un fichier *Makefile* pour compiler le programme, et un fichier readme.

Pour exécuter le programme, il faut donc le compiler via la commande *make.* Une fois ceci fait, un fichier binaire *egrep* est généré dans le dossier *bin.* On l'utilisera de la façon suivante :

$ egrep <expression regulière> [?fichier1 fichier2 …]

si aucun fichier n'est fourni, le programme lira dans *stdin*.

V- Tests

Nos tests se sont appuyés sur une conversion des automates en fichier .dot . Cette conversion se fait via la fonction *serialize* de la classe Automata. Nos sérialisation se font à trois instants clefs du programme : l'automate initial (après la construction de Thompson), la première suppression des epsilon-transitions, la première déterminisation, et finalement la minimisation.

Nous avons également ajouté dans le Makefile une règle *dot_to_pdf* qui se charge de convertir les fichiers *.dot* générés en *.pdf*. Cela nous permet d'obtenir une vision plus graphique de l'automate, en plus de celle fournie par la classe AutomaPrinter et la surcharge de l'opérateur << .

VI- Conclusions et perspectives d'améliorations

Notre implémentation d'egrep est fonctionnelle à quelques exceptions près :

- Tout d'abord un cas de la forme [<caractère> -] (par exemple : [a-]) n'est pas géré car il semble nécessiter une modification de la grammaire que nous n'avons pas réussis à mettre en place

- Ensuite les bracket expression de type [. .] et [= =] ne sont pas gérées non plus, leur signification dépendant du système sur lequel s'exécute la commande

- Les *bracket expression* contenant une range expression et un (ou plusieurs) caractères normaux ne sont pas gérées (par exemple : [ad-h]). Ceci est du à une mauvaise construction de la range expression dans le parser de notre part.

Mis à part ces trois cas le programme fonctionne correctement.

De plus, divers refractoring de plusieurs fonctions pourrait être fait, notamment dans la fonction determinise() dans laquelle on pourrait utiliser le mot clé *auto* à la place d'itérateurs pour parcourir les conteneurs.