Projet d'algorithmique – L3 2008-09

2 décembre 2008

1 Arbre lexical

La première partie du projet consiste à construire l'arbre lexical reconnaissant un ensemble de mots.
Exemple :
l'arbre lexical de la figure 1 reconnait l'ensemble de mots
{`ce,ces,des,le,les lettres,mes,mettre`}. L'arbre sera construit sous la

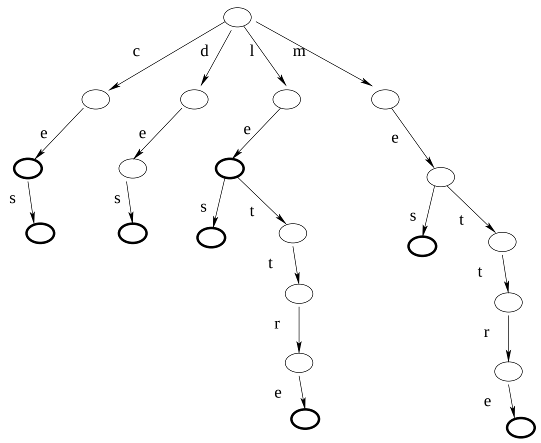

Fig. 1 – Arbre lexical

forme d'un arbre binaire fils gauche, frère droit, voir figure 2.

2 Minimisation d'un arbre lexical

Pour tout automate, il existe un automate ayant un nombre minimal d'états qui reconnait le même langage.

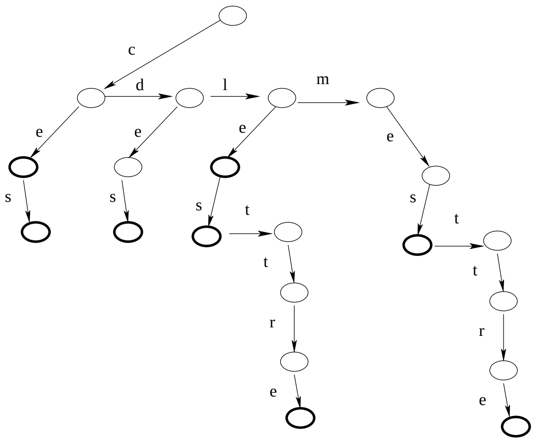

FIG. 2 – Arbre lexical forme fils gauche-frère droit

La base de la minimisation est que si deux états ont le même statut (terminal ou non terminal) et que leurs transitions mènent aux mêmes états, alors ces deux états sont équivalents et seront fusionnés.
Dans le cas particulier d'un arbre lexical, on peut remarquer que les états fusionnables sont situés à la même hauteur.
On fusionne les états ayant les mêmes sorties en les numérotant à l'aide d'une fonction de hachage lors de la phase remontante d'un parcours en profondeur.
La fonction de hachage prend en paramètre :

- le statut de l'état (*Terminal, NonTerminal*)
- la liste des transitions (lettre, Numéro d'état atteint)

Puisqu'elle agit à la remontée, les états atteints ont obtenu un numéro. Les états à fusionner, ayant même statut et même liste de transitions seront associé au même numéro par la fonction de hachage.
Exemple :
pour l'arbre lexical de la figure 1, supposons une fonction de hachage h qui fournit successivement les valeurs :

- h(*Terminal*,$\emptyset$)=α (*Terminal*)
- h(*Terminal*,(s,α))=β (*Terminal*)
- h(*NonTerminal*,(e,β))=γ
- h(*NonTerminal*,(s,α))=δ
- h(*NonTerminal*,(e,δ))=ε
- h(*NonTerminal*,(e,α))=ζ

• h(*NonTerminal*,(e,ζ))=η
• h(*NonTerminal*,(t,η))=θ (*Terminal*)
• h(*Terminal*,(s,α)(t,θ))=ι (*Terminal*)
• h(*NonTerminal*,(e,ι))= κ
• h(*NonTerminal*,(s,α)(t,θ))=λ
• h(*NonTerminal*,(e,λ))= μ
• h(*NonTerminal*,(c,γ),(d,ε),(l,κ),(m,μ))=ν
soit

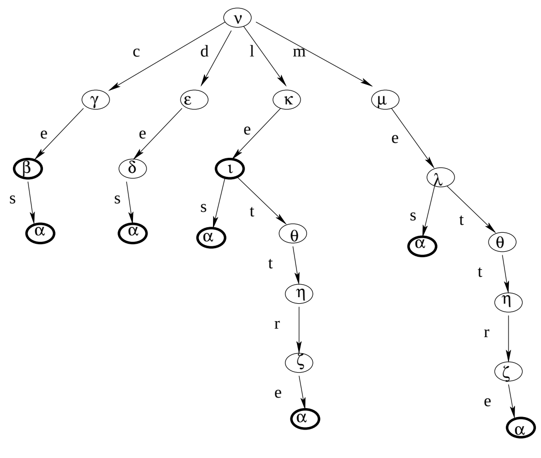

Fig. 3 – Arbre lexicographique numéroté

Un parcours en profondeur de l'arbre obtenu permet la renumérotation des états et la construction finale de l'automate. On obtient :

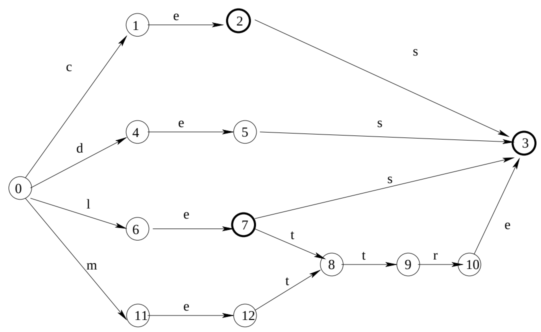

Fig. 4 – Automate

3 Automate de A^*L

On note A l'alphabet. On construit à partir de l'automate de L un automate reconnaissant A^*L en ajoutant, pour tout $a \in A$ une boucle de l'état initial vers lui-même. Ces nouvelles transitions rendent l'automate non-déterministe. On veut pouvoir l'utiliser pour reconnaitre les mots de L dans le texte. Pour celà on utilise une technique qui simule le determinisme : quand le choix se pose entre deux chemins, plutôt que de suivre l'un puis l'autre, on suit les deux en même temps. Ainsi, à chaque instant, plutôt que de mémoriser l'état dans lequel on est, il faut mémoriser *l'ensemble* des états dans lesquels on se trouve en ayant suivi tous les chemins possibles. Un mot est reconnu quand un des états de l'ensemble est final. Vous utiliserez cette technique pour trouver chaque occurence des mots de L dans un texte. A chaque fois qu'un mot u est reconnu, une ligne de la forme :
`u` suivi d'un espace, suivi de la position suivi d'un retour chariot
devra être écrite sur la sortie standard.La position de u est celle de sa première lettre dans le texte (la première lettre du texte est à la position 0). Une fois qu'un chemin de l'état initial (vous noterez qu'il n'y en a qu'un) vers l'état final (unique aussi) a été trouvé, vous devrez pouvoir retrouver l'étiquette du chemin (le mot accepté) en temps constant.

4 Projet

Le projet est à réaliser en binôme (2 étudiants). Une soutenance individuelle pourra être demandée.
Les sources seront placées dans le répertoire

`/home/shares/igm/pub/Algo2/login/Graphes`,

au plus tard le lundi 19 janvier 2009, par l'un des membres du binôme. Le répertoire du second membre du binôme ne contiendra qu'un fichier dont le nom est celui du second membre du binôme. Le rapport est à remettre le lundi 19 janvier 2009 avant 17h au secrétariat.

Dans le **rapport**, on décrira soigneusement les algorithmes, leurs complexités et on motivera le choix des structures de données utilisées. Le rapport contiendra une notice d'utilisation du programme comprenant les commandes de compilation et d'exécution. Il inclura aussi un jeu d'exécutions, c'est-à-dire le traitement d'exemples que vous choisirez avec soin ; le choix des exemples devra être justifié. Il sera tenu compte de la clarté et de la présentation du rapport.

Les formats des fichiers générés sont impératifs. Dans toute la suite, mot désigne une chaine de caractères (de taille majorée par 512 caractères) au sens du format "%s" de `scanf`. Fonctionnalités :
à partir du répertoire de votre projet, la commande : `make MonGrep` devra produire un exécutable de nom `MonGrep` utilisable sur les machines de l'université.
La page de manuel est la suivante :

SYNOPSIS MonGrep [**-a** `fic`] [**-A** `fic`] [**-g** `fic`] [**-r** `fic`] [**-o**] `fichiers`
DESCRIPTION
Une seule des options `-a`, `-A`, `-r` ou `-g` doit être présente. Elle pourra être associée 'a l'option `-o`

Options

-o le résultat doit être placé dans un fichier de même nom que le fichier d'entrée suffixé par *.option.*
S'il y a plusieurs fichiers, plusieurs fichiers résultats devront être créés. Ainsi la commande `MonGrep -a fic1 fic2 -o` produira les fichiers `fic1.a` et `fic2.a` .
En cas d'abscence de cette option les résultats seront affichés sur la sortie standard.

-a fic Construit l'arbre lexical des mots du fichier `fic` et l'affiche les transitions empruntées lors d'un parcours en largeur.
Les lettres d'un même niveau sont sur la même ligne. Chaque lettre est suivie d'un espace, chaque niveau se termine par '\n' L'arbre de la figure 2 est décrit par :

```
c
e d
s e l
s e m
s e
t s
t t
r t
e r
e
```

-A fic Construit l'automate minimal des mots du fichier `fic` et l'affiche sous forme de liste de transitions. Les états sont numérotés par un

parcours en profondeur et parcourus par numéro croissant.
Chaque ligne contient :
le statut de l'état (`N` ou `T`) suivi de la liste des transitions sous la forme `lettre numéro de l'état atteint`, dans l'ordre lexicographique. Chaque élément est suivi d'un espace, toutes les lignes se terminent par '\n'
L'automate de la figure 4 est décrit par :

```
N c 1 d 4 l 6 m 11
N e 2
T s 3
T
N e 5
N s 3
N e 7
T s 3 t 8
N t 9
N r 10
N e 3
N e 12
N t 8
```

-g fic $\texttt{fic}_1, \ldots, \texttt{fic}_n$ Construit l'automate minimal des mots du fichier `fic` et l' utilise pour faire une recherche de mots dans $\texttt{fic}_1, \ldots, \texttt{fic}_n$.

-r fic $\texttt{fic}_1, \ldots, \texttt{fic}_n$ Lit la description de l'automate contenue dans `fic` (le format du fichier est celui de l'option `-A`) et l' utilise pour faire une recherche de mots dans $\texttt{fic}_1, \ldots, \texttt{fic}_n$.

Exemples

La commande `MonGrep -a fic1 -o` construit l'arbre lexical des mots de `fic1`, l'écrit dans `fic1.a`. La commande `MonGrep -r fic1 fic2 fic3 fic4` recupère dans `fic1` un automate, le construit en mémoire, et l'utilise pour faire des recherches de mots dans les fichiers `fic2`, `fic3`, et `fic4`. Le résultat de la recherche est écrit sur la sortie standard.