Bases formelles du TAL
Projet « Automates »
Distribué le 21 février 2006
Retour le 28 avril 2006

Objectifs

L'objectif de ce projet est de permettre une recherche dans des textes en décrivant le motif recherché par le biais d'automates.

Plus précisément, il s'agit de réaliser une application capable
– de lire dans un fichier une ou plusieurs descriptions d'automates
– de manipuler ces automates : complétion, déterminisation, suppression des epsilon-transitions, réunion.
– de trouver dans un texte toutes les occurrences d'un mot reconnu par un automate,
– et d'afficher ces occurrences avec un contexte déterminé (une ligne par exemple)
Par ailleurs, pour tester de façon réaliste ce programme, on demande de choisir un phénomène linguistique qui peut être décrit par une série d'automates, et de mener une petite étude linguistique pour définir la série d'automates appropriée. Parmi les phénomènes que l'on peut proposer (d'autres peuvent être envisagés après accord de l'enseignant) :
– inversion locative du sujet (*devant lui se tenait Jean...*)
– adverbiaux temporels (*la plupart du temps*, *mardi à huit heures...*)
– déterminants complexes (*une majorité de*, *entre trois et cinq...*)
– constructions à verbe support (*faire attention*, *prendre un rendez-vous...*)
– formes composées et surcomposées des temps verbaux (*Quand Panturle a eu labouré son champ, il a déjeuné*)
– les compléments d'agent dans les phrases passives (entouré *d'amis*, assassiné *par Brutus...*)
– ...

Recherche dans un texte

On supposera que l'on dispose de textes étiquetés, qui prennent la forme illustrée par l'exemple suivant :

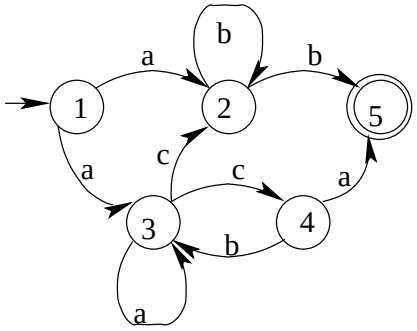
```
On:CL3ms
s':CL3ms
occupe:VP3s
de:P
l':Dfs
entreprise:NCfs
depuis:P
sa:Dfs
naissance:NCfs
jusqu'à:P
sa:Dfs
mort:NCfs
"
,
résume:VP3s
Me:NCms
Jean-Michel:NPms
Lepretre:NPms
,
associé:VKms
de:P
le:Dms
cabinet:NCms
Rambaud-Martel:NPms
.
```

Les motifs recherchés peuvent aussi bien concerner un mot (ou une partie de mot) qu'une étiquette (ou une partie d'étiquette). Il faut donc prévoir une notation permettant de distinguer les deux cas.
Le jeu d'étiquette est décrit à la page
`http://www.linguist.jussieu.fr/~amsili/Ress/jeuEtiquettes.txt`

Format des automates

Le format des fichiers décrivant les automates est libre, mais on peut s'inspirer de l'exemple suivant, où la description donne toutes les informations nécessaires :

```
1 a 2
2 b 2
2 b 5
1 a 3
3 c 2
3 c 4
3 a 3
4 b 3
4 a 5
5
```



Implémentation

Toute liberté est laissée pour l'implémentation des automates en Java. Voici toutefois une proposition, qui peut guider votre développement. Elle repose sur les cinq classes `AutomateFini`, `Etat`, `Transition`, `BandeLecture` et `Configuration` dont certaines sont définies ci-dessous. Il est important de profiter de ce travail pour mettre en œuvre le principe d'encapsulation. Pour cela, on veillera à bien cacher les détails d'implémentation de chaque classe, grâce aux mécanismes de contrôle de l'accessibilité, et à bien spécifier l'interface de chaque classe. Les sections qui suivent décrivent brièvement certaines classes ainsi qu'une partie de leurs méthodes. La dernière section décrit la classe `HashSet` qui permet de manipuler des ensembles en Java. On veillera à définir pour toutes les classes une méthode `toString` qui réalise la représentation d'un objet sous forme d'une chaîne de caractères.

La classe AutomateFini

Un automate fini A est un quintuplet $\langle Q, \Sigma, \delta, q_0, F\rangle$ où : Q est l'ensemble des états, Σ est l'alphabet d'entrée, δ est la fonction de transition ($\delta \subseteq Q \times \Sigma \times Q$), $q_0 \in Q$ est l'état initial et $F \subseteq Q$ est l'ensemble des états d'acceptation.

La classe `AutomateFini` suit de près la définition formelle donnée ci-dessus, elle est composée de deux ensembles d'`Etat` (Q et F), d'un ensemble de `Transition` (δ), et d'un `Etat` initial (q_0). On définit de plus les méthodes suivantes :

- `public Etat nouvelEtat()` crée un nouvel état et l'ajoute aux états de l'automate.
- `public Transition addTrans(Etat o, char sym, Etat d)` crée une transition et l'ajoute aux transitions de l'automate
- `public void setInit(Etat e)` établit `e` comme état initial de l'automate
- `public Etat getInit()` revoie l'état initial de l'automate
- `public void addAccept(Etat e)` ajoute `e` à l'ensemble des états d'acceptation de l'automate
- `public HashSet getAccept()` revoie l'ensemble de états d'acceptation de l'automate
- `public boolean isAccept(Etat e)` renvoie `true` si `e` est un état d'acceptation de l'automtae
- `public HashSet delta(Etat e, char s)` renvoie l'ensemble composé des états x tels que $(e, s, x) \in \delta$

La classe Transition

Une transition est un triplet $\langle e_1, s, e_2 \rangle$ où e_1 et e_2 sont des états et s un symbole de Σ. Un tel triplet signifie que $e_2 \in \delta(e_1, s)$. La classe `Transition` suit exactement cette structure. On définit de plus les trois accesseurs `getSymbol()`, `getOrigine()` et `getDestination()`.

La classe BandeLecture

La classe `BandeLecture` regroupe le bande de lecture proprement dite (la suite de symboles) et la position de la tête de lecture. On définit de plus les méthodes suivantes :

- `public boolean fin()` renvoie `true` si l'on est arrivé en fin de bande.
- `public char getCarCourant()` renvoie le symbole se trouvant sous la tête de lecture.
- `public boolean avance()` déplace la tête de lecture d'un cran vers la droite

La classe Configuration

Une configuration est composée d'un `AutomateFini`, d'un `Etat` de ce dernier et d'une `BandeLecture`. On définit de plus les méthodes suivantes :

- `public int etat()` renvoie l'état de la configuration, on pourra représenter les états par des constantes telles que `BLOQUE`, `ACCEPT` et `ENCOURS`
- `public HashSet suivants()` renvoie un ensemble regroupant toutes les configurations atteignables en un mouvement à partir de la configuration courante.
- `public static boolean accept(AutomateFini a, String mot)` renvoie `true` si `mot` appartient au langage reconnu par `a`.

La classe HashSet

La classe `HashSet` est définie dans le package `java.util` (pensez à écrire `import java.util.*;` au début de vos fichiers de définition de classes). Elle permet de représenter des ensembles d'objets quelconques. Elle implémente de plus les méthodes suivantes :

- `public boolean add(Object o)` ajoute l'objet `o` à l'ensemble
- `public void clear()` enlève tous les éléments de l'ensemble
- `public boolean contains(Object o)` renvoie `true` si l'ensemble contient l'objet `o`
- `public remove(Object o)` enlève `o` de l'ensemble s'il est présent
- `public int size()` renvoie le cardinal de l'ensemble
- `public iterator()` renvoie un iterateur sur l'ensemble. L'itérateur permet de parcourir l'ensemble à l'aide des méthodes `next()` et `hasNext()`. La boucle standard de parcours d'un ensemble peut s'écrire de la façon suivante.

```
for(iterator i = c.iterator();i.hasNext();)
    processObject(i.next());
```