M2 P7 Prolog-TAL – Travaux Dirigés 1

Quelques gammes autour des expressions régulières, automates et transducteurs

Eric de la Clergerie et André Bittar

13 Novembre 2008

Introduction

Le but de ce TD est d'examiner comment Prolog peut être utilisé pour traiter la reconnaissance de langages réguliers, que ce soit au travers d'expressions régulières, d'automates à états finis (FSA) ou de transducteurs. Comme exemple concret, nous nous intéressons à la reconnaissance et transduction de nombres, vu comme un exemple simple de reconnaissance et normalisation d'entités nommées. Une indication de transducteur possible est donné par la figure 1.

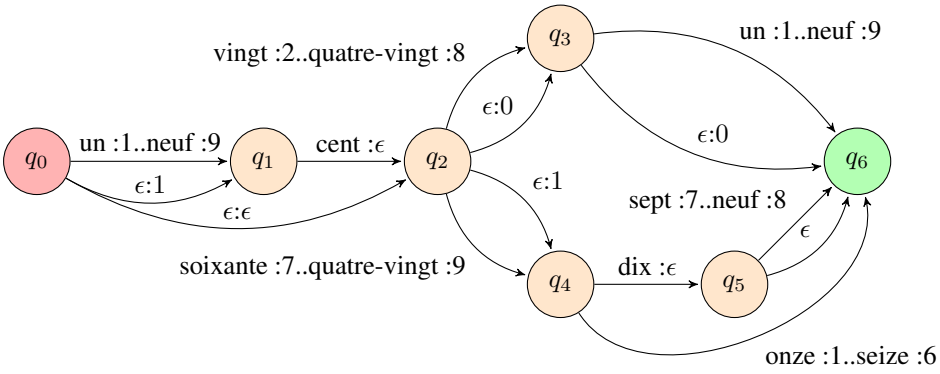

FIG. 1 – Transducteur simplifié pour les nombres de 0 à 999

Le matériel pour ce TD peut être récupéré sous forme d'archive `.tgz` à `http://alpage.inria.fr/~clerger/PrologTAL08/td1.tgz`. Il comprend :

- un moteur FSA `fsa.pl`
- un moteur RegExp `regexp.pl`
- un « compilateur » `regexp2fsa.pl`
- une RegExp `numbers.rx` pour la reconnaissance des nombres de 0 à 999 écrits en lettres
- un FSA `numbers.fsa` pour la reconnaissance des nombres de 0 à 999 écrits en lettres

Exercice 1 : Reconnaître des nombres

Compiler avec `dyacc` le moteur RegExp et vérifier que la RegExp `numbers.rx` permet bien de reconnaître des nombres écrits en lettres.

De même, compiler avec `dyacc` le moteur FSA et vérifier que le FSA `numbers.fsa` permet bien de reconnaître des nombres écrits en lettres.

Vérifier que les résultats sont identiques avec les deux moteurs.
Éventuellement, corriger la description (RegExp ou FSA) pour une meilleure reconnaissance des nombres.

Exercice 2 : Émettre des chiffres

Modifier le moteur FSA en un moteur FST `fst.pl` pouvant traiter des transducteurs à deux niveaux.
Modifier le FSA `numbers.fsa` en un FST `numbers.fst` permettant d'émettre la liste Prolog des chiffres constituant le nombre reconnu. Éventuellement, rechercher des notations compactes pour des actions synchronisées dans des ranges.
En résultat, pour *deux cent quatre-vingt treize*, le programme doit retourner la liste `[2,9,3]`.
Vérifier sur quelques exemples que le programme fonctionne correctement.

Exercice 3 : Générer des nombres

En modifiant la requête dans le moteur FST `fst.pl` (par inversion des rôles des bandes), vérifier que le transducteur `numbers.fst` permet d'engendrer des nombres en lettres.
Éventuellement, en modifiant encore la requête, vérifier qu'il est possible de spécifier partiellement le nombre en lettre et la liste des chiffres et d'utiliser le transducteur pour combler les trous.

Exercice 4 : Compiler vers des FSA

Compiler avec `dyacc` le « compilateur » Regexp2fsa (`regexp2fsa.pl`) et vérifier que l'automate produit à partir de `numbers.rx` est fonctionnel et équivalent à `numbers.fsa`.

Exercice 5 : Étendre les RegExp en transducteurs

Concevoir une extension immédiate des Expressions Régulières permettant de travailler sur 2 bandes. Modifier en conséquence le moteur `regexp.pl` en un moteur `regexp2.pl`.
Modifier l'expression régulière `numbers.rx` en une expression régulières sur 2 bandes permettant l'émission d'une liste de chiffres.
Vérifier que le résultat est équivalent à la version FST.

Exercice 6 : Compiler vers des FST

Modifier le compilateur `regexp2fsa.pl` en un compilateur `regexp2fst.pl` permettant de compiler des expressions régulières 2 bandes vers des transducteurs.

Exercice 7 : Compiler vers Prolog

Explorer comment modifier le compilateur `regexp2fsa.pl` en un compilateur `regexp2dyalogfsa.pl` pour produire directement des clauses Prolog plutôt que des transitions FSA. Essentiellement, il s'agit de modifier le prédicat emit_trans/4 et d'émettre la requête.
Appliquer des modifications similaires pour le compilateur `regexp2fst.pl`.

Exercice 8 : Étendre à 3 bandes

Cet exercice est optionnel ! L'objectif est d'étendre les transducteurs pour qu'ils travaillent sur trois bandes et d'utiliser de tels transducteurs sur les nombres avec la bande 1 pour une représentation en français, la bande 2 pour une représentation en chiffres et la bande 3 pour une représentation en anglais.

Cette extension peut prendre diverses formes, au choix. Soit au niveau d'un moteur `regexp3.pl`, d'un moteur `fst3.pl` ou d'un compilateur vers fst3 ou vers DyALog.

Exercice 9 : Reconnaître des entrelacements

Cet exercice est totalement optionnel !

L'objectif est d'implémenter une extension des expressions régulières permettant de reconnaître l'entrelacement des langages reconnus par deux expressions. Ainsi, l'entrelacement `(a,b) ## (1,2)` des séquences (a, b) et $(1, 2)$ reconnaît les séquences $ab12$, $a1b2$, $a12b$, $1a2b$ et $12ab$.

L'opérateur d'entrelacement ne sort pas du cadre des expressions régulières et peut être implanté en le substituant par des opérateurs plus habituels (disjonctions et séquences).

Néanmoins une implantation plus directe est possible en considérant qu'une transition pour un FSA va d'une liste d'états $\mathcal{S}_1$ vers une autre liste d'états $\mathcal{S}_2$. Ces listes collectent les états associés à chaque sous-automate en train d'être traversé. Une transition de l'entrelacement consiste à avancer dans un des sous-automates de l'entrelacement.

Les opérations de base sur ces listes d'états sont :

– démarrer la traversée d'un nouvel sous-automate en ajoutant son état initial à la liste des états en cours ;
– enlever de la liste l'état final d'un sous-automate après une traversée complète ;
– avancer un état dans la liste suite à la traversée d'une transition.