TD Info 8 : Automates finis

Michael Monerau

9 février 2011

1 Codage des automates finis

On rappelle la :

Définition 1. *Un automate fini $\mathcal{A}$ sur l'alphabet Σ est un quintuplet $\langle Q, \Sigma, E, \mathcal{I}, \mathcal{F} \rangle$ où Q est l'ensemble fini des états, $\mathcal{I} \subseteq Q$ est l'ensemble des états initiaux, $\mathcal{F} \subseteq Q$ les finaux, et où $E \subseteq Q \times A \times Q$ est l'ensemble des transitions.*

L'automate est *déterministe* lorsque E est une relation déterministe (ie. c'est le graphe d'une fonction $\delta : Q \times A \longrightarrow Q$). On prendra nos automates sans ε-transitions.

1.1 Automates déterministes

1.1.1 Définition

Pour implémenter un automate, on va considérer que les états sont étiquetés par des entiers et que l'alphabet est l'ensemble des caractères, noté `char` en Caml. On pose donc le type d'un automate déterministe :

```
type automate_det =
{
    initial: int;
    finaux : int t; (* set *)
    transition : int -> char -> int;
};;
```

où `t` est le type d'un ensemble, qu'on amène dans l'espace de noms au préalable par `#open "set";;`.

Pour définir un ensemble, on donne une fonction de comparaison (la structure sous-jacente est un arbre rouge-noir) du genre

```
let pred_int a b =
  if a < b then -1
  else if a > b then 1
  else 0;;
```

pour donner :

```
let ens_vide = empty_set pred_int.
```

Après, on ajoute des éléments en utilisant `add` et on teste l'appartenance grâce à `mem`. Voir le manuel de référence pour plus d'informations.

Question 1.1. Exemple

À l'aide de ce type, stocker dans une constante l'automate reconnaissant les mots bien parenthésés avec au plus deux niveaux d'imbrication.

Question 1.2. Parcours

Écrire une fonction `accepte : automate_det -> string -> bool` qui renvoie si un automate déterministe passé en argument reconnaît le mot spécifié.

Indication : On pourra utiliser les fonctions `sub_string`, `nth_char` et `string_length`, dont on trouvera une description dans le manuel de référence.

1.1.2 Parenthésage

On pose la :

Définition 2. *On dit qu'un mot est bien parenthésé si à chaque parenthèse ouvrante on peut associer une unique parenthèse fermante plus loin dans le mot.*

Par exemple, "`(a)((b)(c()))`" est bien parenthésé, mais "`((a)`" ou "`)(a)(`" ne le sont pas.

Question 1.3. Mots bien parenthésés

L'ensemble des mots bien parenthésés est-il rationnel ?

Écrire une fonction `automate_parentheses : int -> automate` qui renvoie l'automate déterministe ayant pour langage $\mathcal{L}_n = \{$ mots bien parenthésés avec au plus n niveaux d'imbrication $\}$, où n est passé en argument.

Tester l'automate sur des mots parenthésés et vérifier les résultats.

1.2 Automates non déterministes

Un automate est non-déterministe si en lisant une lettre, plusieurs transitions sont possibles à partir d'un état. C'est-à-dire que la relation E n'est pas un graphe de fonction.

On dit qu'un mot est accepté par l'automate s'il existe un chemin acceptant qui lit le mot.

Question 1.4. Implémentation

Définir un type approprié pour représenter un tel automate et écrire la fonction `accepte`.

Vérifier sur un automate non-déterministe reconnaissant $L = (a + b)^\star.ab$.

2 Automates à pile

2.1 Définition, puissance de calcul, déterminisme

Un automate à pile est un automate fini mais qui possède un autre attribut : une pile. Chaque transition dépend du caractère de haut de pile et en empile un autre. Formellement : on ajoute à la définition un alphabet de pile Z ainsi qu'un symbole de pile initial $z_0 \in Z$. L'ensemble des transitions est maintenant $E \subseteq (\Sigma \cup \{\varepsilon\}) \times Q \times Z \times Q \times Z^\star$. Une transition est : $q_1, z \to^a q_2, h$, et sa sémantique est : si on est dans l'état q_1 et que la pile est $z.w$ (ie. z est le haut de pile) alors si on lit la lettre a on passe dans l'état q_2 et la pile devient $h.w$ (noter que z est une lettre mais h est un mot).

On ne demande pas que l'automate soit déterministe.

Il y a plusieurs modes d'acceptation différents pour un tel automate : par pile vide (le calcul est acceptant s'il finit sur une pile vide), par états finaux (comme un automate fini), par sommet de pile (si le sommet de pile est dans un certain sous-ensemble de Z). Ces trois modes d'acceptation définissent des classes d'automate qui reconnaissent les mêmes langages. On choisira l'acceptation par pile vide.

Question 2.1. Implémentation

Définir un type `automate_pile` pour un automate à pile et implémenter sa fonction d'acceptation (on utilisera le mode par état final).

Les automates à pile reconnaissent strictement plus de langages que les automates finis traditionnels. Ils reconnaissent l'ensemble des langages algébriques (appelés ainsi car ils vérifient un système d'équations), encore décrits comme l'ensemble des langages de grammaires hors-contexte [cf. explications au tableau].

Les langages algébriques ne présentent pas toutes les propriétés agréables des langages rationnels. Par exemple, la classe des langages algébriques n'est close ni par intersection ni par complémentation (donc pas par union non plus).

2.2 Exemple

Question 2.2. Grammaire

Écrire la grammaire hors-contexte du langage des mots bien parenthésés.

Question 2.3. Langage de Dyck

Écrire un automate à pile qui reconnaît le langage des mots bien parenthésés sans borne sur le niveau d'imbrication. Vérifier sur des exemples.

2.3 Déterminisme

De même, on définit la notion d'automate à pile déterministe en demandant qu'une seule transition ne soit possible à chaque instant.

Contrairement aux automates finis classiques, les automates à pile déterministes reconnaissent strictement moins de langages que les automates à pile non-déterministes. En effet, on peut complémenter un automate déterministe à pile (c'est plus technique qu'il n'y paraît), alors que les langages algébriques ne sont pas stables par complémentation.

De plus, les trois modes d'acceptation ne sont plus équivalents.

Question 2.4. Langage non-déterministe

Soit le langage $L = \{a^n b^p a^n \mid n, p > 0\} \cup \{a^n b^p a^p \mid n, p > 0\}$.

On peut montrer que L ne peut être reconnu par un automate à pile déterministe (les langages déterministes vérifient une forme de lemme de l'étoile renforcé que L ne présente pas).

Trouver un automate à pile non-déterministe qui reconnaît L, et comprendre pourquoi on ne peut pas le déterminiser.