Handout 3

Automates Déterministes

1 Automates Finis Déterministes

Un *automate fini déterministe* (AFD) est un quintuplet $(\Sigma, Q, q_0, F, \delta)$ tel que

- Σ est un alphabet fini ;
- Q est un ensemble fini, appelé l'ensemble des *états* ;
- $q_0 \in Q$, appelé *l'état initial* ;
- $F \subseteq Q$, appelé *ensemble des états acceptants* ;
- $\delta : Q \times \Sigma \rightsquigarrow Q$ est une fonction partielle, appelée la *fonction de transition*.

Certains auteurs disent *état final* à la place de *état acceptant*.

Un AFD est *complet* quand sa fonction de transition est une fonction totale.

Étant donné un automate $(\Sigma, Q, q_0, F, \delta)$, on définit une fonction partielle $\delta^* : Q \times \Sigma^* \rightsquigarrow Q$ par récurrence sur le deuxième argument :

$$\begin{array}{rcl} \delta^*(q, \epsilon) & = & q \\ \delta^*(q, wa) & = & \left\{ \begin{array}{ll} \text{indéfini} & \text{quand } \delta^*(q,w) \text{ indéfini} \\ \delta(\delta^*(q,w), a) & \text{quand } \delta^*(q,w) \in Q \end{array} \right. \end{array}$$

Définition équivalente : Quand $w = w_1 \cdots w_n$ et il existe $p_0, \ldots, p_n$ tel que pour tous $0 \leq i < n$: $\delta(p_i, w_i) = p_{i+1}$, alors $\delta^*(p_0, w) = p_n$.

Un automate $A = (\Sigma, Q, q_0, F, \delta)$ *accepte* un mot $w \in \Sigma^*$ quand $\delta^*(q_0, w) \in F$.

Le *langage reconnu* par l'automate $A = (\Sigma, Q, q_0, F, \delta)$ est $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$.

Un langage L est *reconnaissable* quand il existe un AFD A tel que $L = \mathcal{L}(A)$. Les anglophones disent : *regular langage*. On note *Rec* la classe des langages reconnaissables.

Attention : il y a à priori trois comportements différents possibles d'un automate $(\Sigma, Q, q_0, F, \delta)$ sur un mot $w \in \Sigma^*$:

- il consomme le mot entier et arrive dans un état acceptant : $\delta^*(q_0, w) \in F$;
- il consomme le mot entier et arrive dans un état qui n'est pas acceptant : $\delta^*(q_0, w) \in Q \setminus F$;
- il ne consomme pas le mot entier car il bloque sur une transition manquante : $\delta^*(q_0, w)$ pas défini.

Dans le premier cas l'automate accepte $w \in \Sigma^*$, dans les deux derniers cas il le *refuse*. Quand un automate est complet, le troisième cas ne peut jamais se produire.

Il est facile de montrer, on utilisant un état *puits*, que pour tout AFD A il y a un AFD complet A' tel que $\mathcal{L}(A) = \mathcal{L}(A')$.

Tout langage fini est reconnaissable (construction d'un arbre préfixe).

Il n'est pour l'instant pas évident si la classe des langages reconnaissable est close sous union, intersection, concaténation, ou étoile de Kleene. En revanche, cette classe est close sous complément !

Aussi, on a une méthode évidente pour montrer qu'un langage est reconnaissable (construire l'automate), mais pour l'instant aucune méthode pour montrer qu'un langage n'est *pas* reconnaissable.

2 Complétion d'un automate

Un automate est *complet* quand sa fonction de transition est une fonction totale.

Étant donné un automate $A = (\Sigma, Q, q_0, F, \delta)$ on définit sa *complétion* par $A_c = (\Sigma, Q \cup \{\bot\}, q_0, F, \delta_c)$ avec pour tous $a \in \Sigma$:

$$\begin{array}{rcll} \delta_c(q,a) & = & \delta(q,a) & \text{si } q \in Q,\ \delta(q,a) \in Q \\ \delta_c(q,a) & = & \bot & \text{si } q \in Q,\ \delta(q,a) \text{ indéfini} \\ \delta_c(\bot,a) & = & \bot & \end{array}$$

Il est sous-entendu que $\bot \notin Q$ (sinon il faut choisir un autre symbole). On a que $\mathcal{L}(A) = \mathcal{L}(A_c)$.

3 Complément d'un langage reconnaissable

La classe *Rec* des langages reconnaissables est close sous complément : Soit $A = (\Sigma, Q, q_0, F, \delta)$ un automate déterministe *complet* (il est important qu'il est complet !), alors on peut définir son complément $A^{comp} = (\Sigma, Q, q_0, Q - F, \delta)$. On a que $(\mathcal{L}(A))^{comp} = \mathcal{L}(A^{comp})$.

4 Utiliser les automates pour chercher des motifs dans des mots

La méthode naïve pour décider si un motif est un facteur d'un mot vue en IP1 correspond à ce pseudo-code :

```
function bool is_factor(pattern,text) :
  # est-ce que pattern est un facteur de text ?
  for i from 0 to length(text)-length(pattern) :
    if is_factor_at(pattern,text,i) then return true
  return false

function bool is_factor_at(pattern,text,i) :
  # est-ce que pattern est un prefixe de text[i:] ?
  for j from 0 to length(pattern)-1 :
    if pattern[j] != text[i+j] then return false
  return true
```

Coût d'exécution dans le pire des cas : $|text| * |pattern|$. Les automates permettent de faire mieux !