Automate des Occurrences

Références :

- *Éléments d'algorithmique*. Beauquier, Berstel, Chrétienne. p.351
- *Introduction à l'Algorithmique*. Cormen et al. p.886

Remarque : La preuve du lemme 1 est inspirée de Beauquier, cependant Beauquier considère le théorème démontré immédiatement à partir de ce lemme, ce qui n'est pas trivial voire carrément faux.

Contexte : On dispose d'un texte $t \in \Sigma^*$ de taille n et d'un mot $x \in \Sigma^*$ de taille m ("petit" devant n). On notera P l'ensemble des préfixes de x.

Définition 1

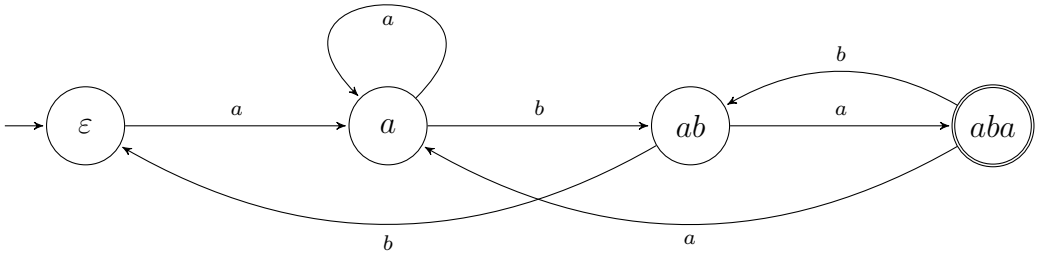
Posons $f_x(u)$ = le plus long suffixe de u qui soit préfixe de x

Développement : On considère l'automate $\mathcal{A}_x = (P, \Sigma, \delta, \{\varepsilon\}, \{x\})$ avec :

$$\delta(p, a) = \text{ le plus long suffixe de } pa \text{ qui soit dans } P = f_x(pa).$$

Cet automate est appelé *Automate des occurrences* de x.

Exemple : Le mot aba a pour automate



Théorème 1

$\mathcal{A}_x$ est l'automate minimal reconnaissant le langage $L = \Sigma^* x$

Démonstration : On sait que les états de l'automate minimal s'identifient aux $u^{-1}L$ pour $u \in \Sigma^*$.

Lemme 1

$\forall u \in \Sigma^*,\ u^{-1}L = f_x(u)^{-1}L.$

En effet soit $u \in \Sigma^*$, alors $f_x(u)$ est un suffixe de u donc on peut écrire $u = v f_x(u)$ et ainsi $u^{-1}L = f_x(u)^{-1}v^{-1}L \supseteq f_x(u)^{-1}L$.

Réciproquement, soit $w \in u^{-1}L$ alors $uw \in \Sigma^* x$ posons donc $uw = vx$. On veut montrer que $f_x(u)w \in \Sigma^* x$. Distinguons alors 2 cas :

- Si w est un suffixe de x posons $x = yw$. Il s'en suit après simplification que $u = vy$ avec y préfixe de x donc y est suffixe de $f_x(u)$; ce qui s'écrit $f_x(u) = zy$ et donc $f_x(u)w = zyw = zx \in \Sigma^* x$.
- Si x est un suffixe de w, alors $w \in \Sigma^* x$ et donc $f_x(u)w \in \Sigma^* x$. □

Le lemme 1 nous donne une application de l'ensemble P des préfixes de x dans l'ensemble des états de l'automate minimal : $f_x(u) \mapsto u^{-1}$ et qui est de plus surjective. Cette application se prolonge en un morphisme d'automates :

$$\delta_{min}(p^{-1}L, a) = a^{-1}p^{-1}L = (pa)^{-1}L = f_x(pa)^{-1}L$$

Enfin le morphisme est injectif, en effet soit u, v tels que $f_x(u)^{-1}L = f_x(v)^{-1}L$ alors $u^{-1}L = v^{-1}L$ ce qui signifie : $\forall w \in \Sigma^*$, $uw \in L$ **ssi** $vw \in L$.

$f_x(u)$ est un préfixe de x posons donc $x = f_x(u)x'$. Pour $w = x'$, on a $ux' = u'f_x(u)x' = u'x \in L$ et donc $vx' \in L = \Sigma^* x$. Posons $vx' = v'x$, en résumé on a après simplification par x' à droite : $v = v'f_x(u)$. Ainsi $f_x(u)$ est un suffixe de v qui est un préfixe de x, c'est donc un bord de $f_x(v)$ et par symétrie $f_x(v)$ est un bord de $f_x(u)$ donc $f_x(u) = f_x(v)$.

Les deux automates sont isomorphes, ils reconnaissent donc le même langage et sont tous deux minimaux, d'où le théorème. □

Algorithme : Trouver une occurrence de x dans t reviens alors à suivre une exécution de l'automate $\mathcal{A}_x$ avec le mot t. Ce calcul est évidemment en $O(|t|)$ et peut être implémenté de manière très simple (bas niveau).

Limites : L'automate utilise cependant une mémoire de $O(|\Sigma| \times |x|)$.

Généralisation On peut généraliser l'automate $\mathcal{A}_x$ à un ensemble $X \subset \Sigma^*$ fini de mots. On a alors $P = \{u \mid \exists x \in X,\ \exists y \in \Sigma^*,\ uy = x\}$.

L'automate reconnaît alors $\Sigma^* X$. On perd la minimalité a priori.

Calcul de f_x : Une partie de la complexité de la recherche du motif x est caché dans le calcul de f_x. Mais celui-ci peut se faire en $O(|x|)$.

En effet :

$$f_x(ua) = \begin{cases} ua & \text{si } ua \text{ préfixe de } x \\ f_x(Bord(u)a) & \text{sinon} \end{cases}$$

Calcul des bords [Beauquier] p.344

Proposition 1

$$Bord(ua) = \begin{cases} Bord(u)a & \textit{si } Bord(u)a \textit{ préfixe de } u \\ Bord(Bord(u)a) & \textit{sinon} \end{cases}$$

Algorithme 1 : Algorithme de calcul des bords maximaux

```
Entrées : x
début
    β[0] ← −1
    pour j = 1 à m faire
        i ← β[j − 1]
        tant que i ≥ 0 et x[j] ≠ x[i + 1] faire
            i ← β[i]
        β[j] ← i + 1
    renvoyer β
fin
```

Pour le calcul de la fonction de transition, on va construire la table $\beta[i, a]$ qui est le bord de $x_1 \cdots x_i a$.

Algorithme 2 : Algorithme de calcul des transitions

```
Entrées : x
début
    β[0, a] ← −1 pour tout a ∈ Σ
    pour j = 1 à m faire
        pour a ∈ Σ faire
            i ← β[j − 1, x[j]]
            tant que i ≥ 0 et x[i + 1] ≠ a faire
                i ← β[i, x[i + 1]]
            β[j, a] ← i + 1
    renvoyer β
fin
```

Le calcul se fait en $O(m|\Sigma|)$.