LANGAGES ET AUTOMATES LOCAUX

1 Langages locaux

1.1 Définition

Définition. Un langage L sur un alphabet A est dit local quand il existe trois ensembles $P \subset A$, $S \subset A$ et $N \subset A^2$ tels que

$$L \setminus \{\varepsilon\} = (PA^* \cap A^*S) \setminus (A^* N A^*)$$

Autrement dit, un langage est local quand il est défini par l'ensemble des premières lettres de ses mots P, l'ensemble des dernières lettres S et l'ensemble N des facteurs de longueur 2 **interdits** dans les mots de L.

Bien sûr, plutôt que de préciser les facteurs de longueur 2 interdits, on peut aussi définir les facteurs de longueur 2 autorisés : $F = A^2 \setminus N$: dans ce cas, $L \setminus \{\varepsilon\}$ peut être défini comme l'ensemble des mots qui commencent par une lettre de P, se terminent par une lettre de S et dont tous les facteurs de longueur 2 sont dans F (plus difficile à écrire en termes de langage...)

Exemples

a) Avec $P = \{a\}$, $S = \{b\}$ et $N = \{aa, bb\}$, on peut avoir $L = (ab)^*$ ou $L = (ab)^+$.

b) Avec $P = \{a\}$, $S = \{b\}$ et $N = \{ab, ba\}$, on a $L = \varnothing$ ou $L = \{\varepsilon\}$.

c) Le langage $(abc)^*$ est local en prenant $P = \{a\}$, $S = \{b\}$ et $F = \{ab, bc, ca\}$.

d) Le langage $a(ba)^*b$ est local en prenant $P = \{a\}$, $S = \{b\}$ et $F = \{ab, ba\}$.

e) Le langage a^*ba n'est pas local (pourquoi ?).

1.2 Opérations sur les langages locaux

Proposition 1

L'intersection de deux langages locaux est un langage local.

L'étoile d'un langage local est un langage local.

En revanche, la réunion ou la concaténation de deux langages locaux n'est pas en général un langage local : soit $L = ab$ et $L' = a^*$ (qui sont locaux), $L \cup L'$ et LL' ne sont pas locaux (pourquoi ?).

Cependant, en ajoutant une condition :

Proposition 2 *Si L et L' sont deux langages locaux construits sur des alphabets disjoints, alors LL' et $L \cup L'$ sont des langages locaux.*

Remarque. Pourquoi les langages locaux sont-ils intéressants ? Parce qu'ils se prêtent bien à l'écriture de l'algorithme de construction d'un automate reconnaissant un langage défini par une expression régulière.

2 Expressions régulières linéaires

Définition. Une expression régulière est dite linéaire quand elle ne contient pas deux fois la même lettre.

On a alors le résultat suivant.

Proposition 3 *Le langage régulier associé à une expression régulière linéaire est local.*

Remarque. La réciproque est fausse : le langage $a(ba)^*b$ est local, mais n'est pas associé à une expression régulière linéaire.

2.1 Linéarisation des expressions régulières

Quand on a une expression régulière quelconque e, on peut la linéariser en modifiant l'alphabet utilisé : pour toute lettre a présente k fois dans l'expression régulière, on associe l'alphabet A' obtenu à partir de A en remplaçant a par les lettres a_1, …, a_k. On associe à l'expression régulière e l'expression e' sur l'alphabet A' en remplaçant chaque lettre a par ses associées numérotées dans l'ordre de leur apparition.

Par exemple, si $e = (ab)*|a(aba*)*$, la linéarisée de e est $e' = (a_1b_1)*|a_2(a_3b_2a_4*)*$.

Si on sait écrire le méta-algorithme pour les langages locaux et si à partir d'une solution pour une expression linéaire, on peut donner une solution pour une expression quelconque, alors en passant par la linéarisée, on va pouvoir répondre au problème posé :

expression $e \to$ expression linéarisée $e' \to$ langage local $L(e') \to$ algorithme pour $e' \to$ alg. pour e

2.2 Calcul des préfixes, suffixes de longueur 1 et facteurs de longueur 2

Si L est un langage, on note $P(L)$ l'ensemble des lettres initiales (des mots) de L, $S(L)$ celui des lettres finales (des mots) de L et $F(L)$ celui des facteurs de longueur 2 des mots de L.

Si e est une expression régulière, pour alléger les notations, on note $P(e)$ plutôt que $P(L(e))$ et de même pour $S(e)$ et $F(e)$.

Les expressions régulières étant définies inductivement, on peut donner des définitions inductives de ces ensembles (associés à des langages réguliers ou des expressions régulières).

Préfixes

- $P(\varnothing) = P(\varepsilon) = \varnothing$
- pour toute lettre a, $P(a) = \{a\}$
- si e et f sont deux expressions régulières, alors
  - $P(e|f) = P(e) \cup P(f)$
  - $P(e*) = P(e)$
  - $P(ef) = \begin{cases} P(e) & \text{si } \varepsilon \notin L(e) \\ P(e) \cup P(f) & \text{si } \varepsilon \in L(e) \end{cases}$

Suffixes

- $S(\varnothing) = S(\varepsilon) = \varnothing$
- pour toute lettre a, $S(a) = \{a\}$
- si e et f sont deux expressions régulières, alors
  - $S(e|f) = S(e) \cup S(f)$
  - $S(e*) = S(e)$
  - $S(ef) = \begin{cases} S(f) & \text{si } \varepsilon \notin L(f) \\ S(e) \cup S(f) & \text{si } \varepsilon \in L(f) \end{cases}$

Facteurs (de longueur 2)

- $F(\varnothing) = F(\varepsilon) = \varnothing$
- pour toute lettre a, $F(a) = \varnothing$
- si e et f sont deux expressions régulières, alors
  - $F(e|f) = F(e) \cup F(f)$
  - $F(e*) = F(e) \cup S(e)P(e)$
  - $F(ef) = F(e) \cup S(e)P(f) \cup F(f)$

Exemples

a) En appliquant ces règles de calculs, on en déduit que $P(a*) = S(a*) = \{a\}$, $P(ab*) = \{a\}$, $S(ab*) = \{a, b\}$ et $F(ab*) = \{ab, bb\}$

b) De même, si e est l'expression régulière $(a_1b_1)*|a_2(a_3b_2a_4*)*$, alors $P(e) = \{a_1, a_2\}$, $S(e) = \{b_1, a_2, b_2, a_4\}$ et $F(e) = \{a_1b_1, b_1a_1, a_2a_3, a_3b_2, b_2a_4, a_4a_4, b_2a_3, a_4a_3\}$

On constate qu'on a besoin d'un autre algorithme, qui permet de savoir si le mot vide appartient au langage décrit par l'expression régulière.

Mot vide ?

- $V(\varnothing) = faux$, $V(\varepsilon) = vrai$
- pour toute lettre a, $V(a) = faux$
- si e et f sont deux expressions régulières, alors
  - $V(e|f) = V(e)$ ou $V(f)$
  - $V(e*) = vrai$
  - $V(ef) = V(e)$ et $V(f)$

2.3 Complexité

On considère une expression régulière linéaire e.

On peut évaluer grossièrement la complexité des fonctions précédentes en fonction du nombre de lettres de e :

- l'algorithme **Mot vide ?** est de complexité linéaire
- donc les algorithmes **Préfixes** et **Suffixes** sont de complexités quadratiques

Mais c'est peu efficace. Si on regarde bien, on peut calculer en même temps les 4 fonctions. Les ensembles P, S et le booléen V se calculent alors en complexité linéaire, les réunions d'ensembles se faisant par simple concaténation.

Le seul point coûteux est celui du calcul des facteurs de longueur 2, à cause des concaténations $S(x)P(y)$. Il est clair que si e contient n lettres, alors $P(e)$ et $S(e)$ ont au plus n éléments, donc on en déduit que la complexité du calcul de F est quadratique.

En pratique, ces complexités maximales sont rarement atteintes, en profitant de l'évaluation paresseuse des booléens et des nombreux cas de bases.

3 Automates locaux

Définition. Soit $\mathscr{A} = (E, i, T, \delta)$ un automate.

On dit que $\mathscr{A}$ est un automate local quand toutes les transitions étiquetées par une même lettre aboutissent au même état :

pour tout $a \in A$, il existe $q \in E$ tel que pour tout $p \in E$, si $\delta(p, a)$ existe, alors $\delta(p, a) = q$

L'intérêt des automates locaux réside dans le théorème suivant.

Théorème 4 *Tout langage local est reconnu par un automate local, qu'on peut supposer standard. Et réciproquement, tout automate local reconnaît un langage local.*