

CHAPITRE 1 : EXPRESSIONS REGULIERES ET AUTOMATES

1. Les notions d'expression régulière et d'automate

1.1 La commande egrep d'Unix

La commande Unix egrep permet de chercher un motif dans un texte. Chaque ligne tapée est

- soit recopiée par egrep, si le motif s'y trouve,
- soit ignorée.

Le motif peut être défini par la suite de ses caractères, mais il peut être défini aussi par une *expression rationnelle*. Exemples :

```
chemillier$ egrep 'abc'
abababababababa = ignoree
abbcabbcabbc = ignoree
aaaaabbbbbc = ignoree
aaaaabccccc
aaaaabccccc = ligne recopiee car elle contient 'abc'
```

```
chemillier$ egrep 'aaaaa'
aaaaa = ignoree
dkkhaafkjhdjkhaaaaajhkjh = ignoree
dkjdkjdhkdhjaaaaaakjfhjkfh
dkjdkjdhkdhjaaaaaakjfhjkfh = recopiee car plus de 6
occurrences
```

Plutôt que d'indiquer les répétitions en recopiant les caractères, on peut utiliser un symbole de répétition. Ainsi a* signifie zéro, une ou plusieurs occurrences successives de a (attention au « zéro ») :

```
chemillier$ egrep 'ba*ab'
dkjdkhdbbbbaadhjhd
dlkdjldkjdlkjbaaaabflkjflkfj
dlkdjldkjdlkjbaaaabflkjflkfj = au moins 1 a entre 2 b
```

```
chemillier$ egrep 'a*'
dlkdjlkjdkldj
dlkdjlkjdkldj = recopiee, meme si 0 a
```

```
chemillier$ egrep 'ba*b'
bbbbbbccccabbb
bbbbbbccccabbb
```

```
chemillier$ egrep 'tra(la)*lere'
tralalere
tralalere = 1 occurrence
tralalalalalalere
tralalalalalalere = 5 occurrences
tralala = ne marche pas
tralere
```

```
tralere = OK
```

On peut utiliser egrep en traitant le texte contenu dans un fichier ex1. Les lignes contenant le motif sont recopiées, et on peut aussi afficher leur numéro (option -n) :

```
chemillier$ cat ex1
toto
titi
tata
tutu
tete
```

```
chemillier$ egrep -n 'toto' ex1
1:toto
```

```
chemillier$ egrep -n 'titi' ex1
2:titi
```

Le symbole | (Maj-Alt-L sous Mac) permet d'indiquer un choix entre plusieurs caractères ou suites de caractères :

```
chemillier$ egrep -n 't(a|e|u)t(a|e|u)' ex1
3:tata
4:tutu
5:tete
```

```
chemillier$ egrep -n 't(a|e|i)t(u|o|e)' ex1
5:tete
```

```
chemillier$ egrep 't((a|e)t)*a'
tatatatata
tatatatata = OK
tetetetete = non
tatetatetateta
tatetatetateta = OK
teteteteteteteta
teteteteteteteta = OK
taaaaaaaaaaaaaaaaa
taaaaaaaaaaaaaaaaa = OK
```

Les expressions régulières sous Unix (pour egrep, ou pour la commande lex qu'on verra plus tard) comportent également un certain nombre d'abréviations :

[abc] pour ablc

[a-z] pour abl...lz

[a-zA-Z0-9] pour toutes les lettres et chiffres

x+ pour au moins une occurrence de x, c'est-à-dire xx*

x? pour un x optionnel

Exemple : [A-Z][a-z]* pour un nom propre, c'est-à-dire commençant par une majuscule.

1.2 Expressions régulières et automates finis

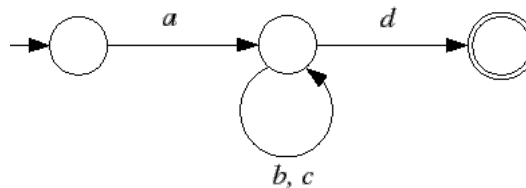
Les expressions régulières sont un moyen de définir et de traiter des ensembles de suites de caractères.

Ces ensembles peuvent contenir :

- des suites de caractères définies explicitement,
- des *répétitions* de suites de caractères,
- des *choix* entre plusieurs suites de caractères.

Exemple : $a(b|c)^*d$

Une autre manière de représenter une expression régulière est de dessiner un graphe appelé « automate fini » :



Combien de mots contient l'ensemble ainsi défini ?

une infinité

=> donc il s'agit d'une représentation finie d'un ensemble infini

Un automate peut être considéré comme une machine avec :

- des états,
- des transitions qui vont d'un état à un autre quand on lit un caractère,
- un état initial (petite flèche entrante) et des états finals (double cercle).

Le « comportement » d'un automate est l'ensemble des suites de caractères qui le font passer de l'état initial à un état final.

On distingue deux types d'automates fini :

- AFD : automate fini déterministe
- AFN : automate fini non déterministe

Les principaux automates finis étudiés dans ce cours sont les AFD.

2. La hiérarchie de Chomsky

2.1 Langages et machines

Un programme informatique est une suite de caractères (aussi appelée « mot ») qu'une machine doit traiter.

Un « langage » est un ensemble de suites de caractères (ou de mots), une machine au sens abstrait est un calculateur sur des mots.

Le domaine des langages formels a été développé par :

- les informaticiens : élaborer des langages de programmation,
- les linguistes : décrire des langues naturelles avec l'idée de les traiter automatiquement.

Le thème central est l'étude des modèles qui permettent une spécification ou une représentation finie des langages formels.

Plusieurs approches :

- spécifier les propriétés des mots du langage : définitions en français ou expression régulières (Kleene) egrep
- spécifier les propriétés des mots du langage : les grammaires formelles (Chomsky)
- reconnaître qu'un mot donné est un mot du langage avec des mécanismes automatiques réalisés par différents types de machines (Turing, McCulloch et Pitt)
- spécification logique, algébrique, etc.

Les automates finis sont les machines les plus simples. Ils interviennent dans de nombreux domaines :

- traitement de texte, compilation, codage, modélisation des circuits, vérification, compression de données, formatage de texte, calcul formel.
- biologie (pour certains algorithmes liés au génome),
- dans l'étude des événements discrets (automatique, musique)

2.2 Classification de Chomsky

Classes de langages	Types de machines	Types de grammaires
réguliers (ou rationnels)	AFD ou AFN	type 3 (régulière)
algébriques	automate à pile déterministe	type 2 (algébrique)
contextuels	machine de Turing non déterministe à espace linéairement borné	type 1 (contextuelle)
rékursivement énumérables	machine de Turing, automates cellulaires, etc.	type 0

Objectifs du cours :

- 1) Éléments *théoriques* pour les deux derniers niveaux : langages rationnels et algébriques
- 2) Applications à la *compilation* :
 - Automates finis (AF) = base de l'analyse lexicale,
 - Grammaires algébriques (voir cours de Patrice Enjalbert) = base de l'analyse syntaxique.

3. Vocabulaire

3.1 Définitions générales

Un *alphabet* Σ un ensemble fini non vide de symboles appelés aussi lettres ou caractères. Exemple : le langage machine est basé sur l'alphabet $\Sigma = \{0, 1\}$.

Un *mot* est une suite finie de symboles.

La *longueur* d'un mot u notée $|u|$ est le nombre de symboles de ce mot. Exemple : le mot $u = 011$ est de longueur $|u| = 3$.

Le *mot vide* noté ε est le seul mot de longueur nulle.

Σ^* représente l'ensemble des mots sur l'alphabet Σ .

Σ^+ représente l'ensemble des mots sur l'alphabet Σ de longueur ≥ 1 .

1.2 Concaténation

La *concaténation* de deux mots u et v , notée uv , est la juxtaposition de u et de v . Pour $u = a_1a_2\dots a_p$ et $v = b_1b_2\dots b_q$, on a $uv = a_1\dots a_pb_1\dots b_q$.

Cette opération est associative, non commutative, et ε est l'élément neutre.

La *puissance* n -ième d'un mot est définie inductivement :

$$u^0 = \varepsilon, u^{n+1} = u^n u.$$

Exemple : $u = aba, u^3 = abaabaaba$.

Un mot $u \in \Sigma^*$ est *facteur* du mot $w \in \Sigma^*$ s'il existe $v, v' \in \Sigma^*$ tels que $w = vuv'$. Si $v = \varepsilon$, on dit que u est *préfixe*. Si $v' = \varepsilon$, on dit que u est *suffixe*. Exemple : abb est facteur de $babba$, et ba est à la fois préfixe et suffixe, mais aa n'est pas facteur.

1.3 Opérations sur les langages

• opérations classiques sur les ensembles :

union \cup , intersection \cap , différence \setminus , complémentaire,

• opérations héritées de la concaténation :

La concaténation de deux langages est définie par :

$$L_1 L_2 = \{uv, u \in L_1 \text{ et } v \in L_2\}.$$

Exemple : $L_1 = \{a, ab\}, L_2 = \{c, bc\}, L_1 L_2 = \{ac, abc, abbc\}$.

La *puissance* d'un langage L est définie inductivement :

$$L^0 = \{\varepsilon\}, L^{n+1} = L^n L.$$

L'*étoile* d'un langage L , notée L^* , est :

$$L^* = \{\varepsilon\} \cup L \cup L^2 \dots \cup L^n \cup \dots$$

Ce langage contient un nombre infini de mots, qui sont les répétitions indéfinies de mots de L .

Exemple :

$L = \{ab, b\}$, L^* est l'ensemble de tous les mots tels que aa n'est pas facteur, et a n'est pas suffixe.

On note également L^+ le langage :

$$L^+ = L \cup L^2 \dots \cup L^n \cup \dots$$

4. Définition des expressions rationnelles (ou régulières)

Définition. Soit Σ un alphabet. Les expressions rationnelles (ou régulières) sur Σ et les langages correspondants sont définis récursivement :

- 1) $\bullet \emptyset$ est une expression rationnelle, et représente l'ensemble vide,
 - $\bullet \epsilon$ est une expression rationnelle, et représente l'ensemble $\{\epsilon\}$,
 - \bullet pour toute lettre a de Σ , a est une expression rationnelle, et représente l'ensemble $\{a\}$.
- 2) Si r et s sont des expressions rationnelles, qui représentent les langages R et S , alors $r + s$, rs , et r^* sont des expressions rationnelles qui représentent les langages $R \cup S$, RS et R^* .

Le terme « rationnel » correspond à l'usage français, « régulier » à l'usage anglais (*regular*).

Ordre de priorité sur les opérations : étoile > concaténation > union.

Exemple : $01^* + 1$ correspond à $(0(1)^*) + 1$.

Quelques expressions rationnelles et leurs langages, pour l'alphabet $\Sigma = \{a, b\}$:

Description en français	Expression rationnelle	Langage correspondant
se terminant par a	$(a + b)^*a$	$\Sigma^* \{a\}$
contenant le facteur bb	$(a + b)^*bb(a + b)^*$	$\Sigma^* \{bb\} \Sigma^*$
ne contenant pas le facteur bb	$(\epsilon + b)(a + ab)^*$	$\{\epsilon, b\} \{a, ab\}^*$

Ces définitions permettent de donner un cadre théorique à l'étude des expressions rationnelles telles qu'on les utilise dans la pratique avec *egrep* ou *lex*.

Il y a une correspondance directe entre les opérations ci-dessus et la notation d'Unix : la notation $r + s$ est équivalente à la barre de disjonction d'Unix $r|s$.

Expression rationnelle	Unix (<i>egrep</i> , <i>lex</i>)
$(a + b)^*a$	$(a b)^*a$
$(a + b)^*bb(a + b)^*$	$(a b)^*bb(a b)^*$
$(\epsilon + b)(a + ab)^*$	$([] b)(a ab)^*$

Le but principal de cette approche théorique est de montrer :

Expressions rationnelles = Langages définis par les AFD