Les expressions régulières

Définitions

Introduction

Comment décrire un langage ??
Étant donné un mot, appartient il à un langage donné ??
Nous allons parler de la théorie des langages, en particulier nous décrivons les expressions régulières, et par conséquent les langages réguliers

Les langages

Définitions

On appelle **alphabet** *un ensemble fini non vide A de symboles (lettres de 1 ou plusieurs caractères).*
On appelle **mot** *toute séquence finie d'éléments de A.*
On note ε le **mot vide**.
On note A^* l'ensemble infini contenant tous les mots possibles sur A.
On note A^+ l'ensemble des mots non vides que l'on peut former sur A, c'est à dire $A^+ = A^* - \{\varepsilon\}$
On note $|m|$ la longueur du mot m, c'est à dire le nombre de symboles de A composant le mot.
On note A^n l'ensemble des mots de A^* de longueur n. Remarque : $A^* = \bigcup_{n=0}^{\infty} A^n$

Exemples

Soit l'alphabet $A = \{a, b, c\}$. $aaba$, $bbbacbb$, c et ε sont des mots de A^*, de longueurs respectives 4, 7, 1 et 0.
Soit l'alphabet $A = \{aa, b, c\}$. aba n'est pas un mot de A^*. $baab$, caa, bc, $aaaa$ sont des mots de A^* de longueurs respectives 3, 2, 2 et 2.

Notation

On note . l'opérateur de **concaténation** de deux mots : si $u = u_1 \dots u_n$ (avec $u_i \in A$) et $v = v_1 \dots v_p$ (avec $v_i \in A$), alors la concaténation de u et v est le mot $u.v = u_1 \dots u_n v_1 \dots v_p$
Remarque : un mot de n lettres est en fait la concaténation de n mots d'une seule lettre.

Les langages

Propriété

$|u.v| = |u| + |v|$
$(u.v).w = u.(v.w)$ *(associativité)*
ε *est l'élément neutre pour* . : $u.\varepsilon = \varepsilon.u = u$

Remarque : nous écrirons désormais uv pour $u.v$

Définition

On appelle **langage** *sur un alphabet* A *tout sous-ensemble de* A^*.

Exemples

Soit l'alphabet $A = \{a, b, c\}$
Soit L_1 l'ensemble des mots de A^* ayant autant de a que de b. L_1 est le langage infini $\{\varepsilon, c, ccc, \ldots, ab, ba, \ldots, abccc, acbcc, accbc, \ldots, aabb, abab, abba, baab, \ldots, accbccbccccca, \ldots, bbccccaccbcabcccacac, \ldots\}$
Soit L_2 l'ensemble de tous les mots de A^* ayant exactement 4 a. L_2 est le langage infini $\{aaaa, aaaac, aaaca, \ldots, aabaa, \ldots, caaaba, \ldots, abcabbbaacc, \ldots\}$

Opération sur les langages

Opérations sur les langages

union : $L_1 \bigcup L_2 = \{w \text{ tq } w \in L_1 \text{ ou } w \in L_2\}$
intersection : $L_1 \bigcap L_2 = \{w \text{ tq } w \in L_1 \text{ et } w \in L_2\}$
concaténation : $L_1 L_2 = \{w = w_1 w_2 \text{ tq } w_1 \in L_1 \text{ et } w_2 \in L_2\}$
puissance : $L^n = \{w = w_1 \ldots w_n \text{ tq } w_i \in L \text{ pour tout } i \in \{1, \ldots, n\}\}$
étoile : $L^* = \cup_{n \geq 0} L^n$

Les langages réguliers

Problème

étant donné un langage, comment décrire tous les mots acceptables ? Comment décrire un langage?
Il existe plusieurs types de langage (classification), certains étant plus facile à décrire que d'autres. On s'intéresse ici aux **langages réguliers**.

Définitions

Un **langage régulier** *L sur un alphabet A est défini récursivement de la manière suivante :*

- *$\{\varepsilon\}$ est un langage régulier sur A*
- *Si a est une lettre de A, $\{a\}$ est un langage régulier sur A*
- *Si R est un langage régulier sur A, alors R^n et R^* sont des langages réguliers sur A*
- *Si R_1 et R_2 sont des langages réguliers sur A, alors $R_1 \bigcup R_2$ et $R_1 R_2$ sont des langages réguliers*

Les langages réguliers se décrivent très facilement par une **expression régulière**.

Les langages réguliers

Définitions

Les expressions régulières (E.R.) sur un alphabet A et les langages qu'elles décrivent sont définis récursivement de la manière suivante :

- *ε est une E.R. qui décrit le langage $\{\varepsilon\}$*
- *Si $a \in A$, alors a est une E.R. qui décrit $\{a\}$*
- *Si r est une E.R. qui décrit le langage R, alors $(r)^*$ est une E.R. décrivant R^**
- *Si r est une E.R. qui décrit le langage R, alors $(r)^+$ est une E.R. décrivant R^+*
- *Si r et s sont des E.R. qui décrivent respectivement les langages R et S, alors $(r)|(s)$ est une E.R. décrivant $R \bigcup S$*
- *Si r et s sont des E.R. qui décrivent respectivement les langages R et S, alors $(r)(s)$ est une E.R. décrivant RS*
- *Il n'y a pas d'autres expressions régulières*

Remarques

on conviendra des priorités décroissantes suivantes : $*$, concaténation, $|$ C'est à dire par exemple que $ab^*|c = ((a)((b)^*))|(c)$
En outre, la concaténation est distributive par rapport à $|$: $r(s|t) = rs|rt$ et $(s|t)r = sr|tr$.

Les langages réguliers

Exemples

- $(a|b)^* = (b|a)^*$ dénote l'ensemble de tous les mots formés de a et de b, ou le mot vide.
- $(a)|((b)^*(c)) = a|b^*c$ est soit le mot a, soit les mots formés de 0 ou plusieurs b suivi d'un c. C'est à dire $\{a, c, bc, bbc, bbbc, bbbbc, \ldots\}$
- $(a^*|b^*)^* = (a|b)^* = ((\varepsilon|a)b^*)^*$ décrit tous les mots sur $A = \{a, b\}$ ou encore A^*
- $(a|b)^*abb(a|b)^*$ dénote l'ensemble des mots sur $\{a, b\}$ ayant le facteur abb
- $b^*ab^*ab^*ab^*$ dénote l'ensemble des mots sur $\{a, b\}$ ayant exactement 3 a
- $(abbc|baba)^+aa(cc|bb)* = \{abbcaa, \ldots, babaabbababaaa, \ldots, abbcabbcaaccbbbb, \ldots\}$

Remarques

$(a|b)^*a(a|b)^*$, qui décrit les mots sur $\{a, b\}$ ayant au moins un a est **ambiguë**. Car, par exemple, le mot $abaab$ "colle" à l'expression régulière de plusieurs manières :

$abaab = \varepsilon\ .\ a\ .\ baab$ avec $\varepsilon \in (a|b)^*$, et $baab \in (a|b)^*$

$abaab = ab\ .\ a\ .\ ab$ avec $ab \in (a|b)^*$, et $ab \in (a|b)^*$

$abaab = aba\ .\ a\ .\ b$ avec $aba \in (a|b)^*$, et $b \in (a|b)^*$

Par contre, l'e.r. $b^*a(a|b)^*$ décrit le **même langage** et **n'est pas** ambiguë.

$abaab = \varepsilon\ .\ a\ .\ baab$ avec $\varepsilon \in b^*$, et $baab \in (a|b)^*$

Les automates à états finis

Problématique

Problème

Le problème qui se pose est de pouvoir reconnaître si un mot donné appartient à un langage donné. Un **reconnaisseur** pour un langage est un programme qui prend en entrée une chaîne x et répond oui si x est une phrase (un mot) du langage et non sinon.

Théorème

Les **automates à états finis** *(A.E.F.) sont des reconnaisseurs pour les langages réguliers.*

Automates à états finis

Définitions

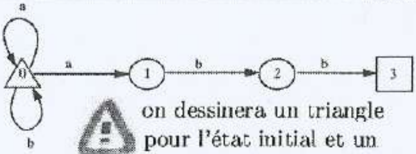
Un **automate à états finis (AEF)** *est défini par*

- *un ensemble* **fini** *E d'états*
- *un état $e_0 \in E$ distingué comme étant l'***état initial**
- *un ensemble fini T (T inclus dans E) d'états distingués comme* **états finaux** *(ou* **états terminaux***)*
- *un alphabet Σ des symboles d'entrée*
- *une* **fonction de transitions** *Δ qui à tout couple formé d'un état et d'un symbole de Σ fait correspondre un ensemble (éventuellement vide) d'états : $\Delta(e_i, a) = \{e_{i_1}, \ldots, e_{i_n}\}$*

Exemple

$\Sigma = \{a, b\}$, $E = \{0, 1, 2, 3\}$, $e_0 = 0$, $T = \{3\}$
$\Delta(0, a) = \{0, 1\}$, $\Delta(0, b) = \{0\}$, $\Delta(1, b) = \{2\}$, $\Delta(2, b) = \{3\}$ (et $\Delta(e, l) = \emptyset$ sinon)

Représentation graphique



on dessinera un triangle pour l'état initial et un

Représentation par une table de transition

état	a	b
0	0,1	0
1	-	2
2	-	3
3	-	-

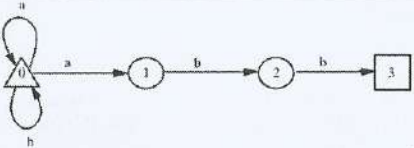
$e_0 = 0$ et $T = \{3\}$

Automates à états finis

Définition

Le **langage reconnu par un automate** *est l'ensemble des chaînes qui permettent de passer de l'état initial à un état terminal.*

Exemple



L'automate de l'exemple précédent accepte le langage régulier (l'expression régulière) ....

Remarque

Un automate peut très facilement être simulé par un algorithme (et donc on peut écrire un programme simulant un AEF). C'est encore plus facile si l'automate est **déterministe**, c'est à dire lorsqu'il n'y a pas à choisir entre 2 transitions). Ce que signifie donc le théorème 3.1 c'est que l'on peut écrire un programme reconnaissant tout mot (toute phrase) de tout langage régulier. Ainsi, si l'on veut faire l'analyse lexicale d'un langage régulier, il suffit d'écrire un programme simulant l'automate qui lui est associé.

4

Définitions

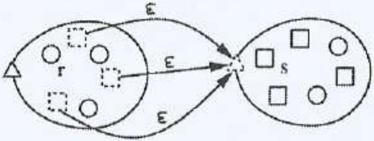
On appelle **ε-transition,** *une transition par le symbole ε entre deux états.*

Pour une expression régulière s, on note $A(s)$ un automate reconnaissant cette expression.

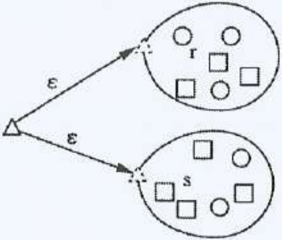
- automate acceptant la chaîne vide 

- automate acceptant la lettre a 

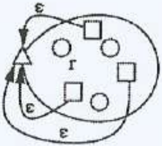
- automate acceptant $(r)(s)$
  1. mettre une ε-transition de chaque état terminal de $A(r)$ vers l'état initial de $A(s)$
  2. les états terminaux de $A(r)$ ne sont plus terminaux
  3. le nouvel état initial est celui de $A(r)$
  4. (l'ancien état initial de $A(s)$ n'est plus état initial)



- automate reconnaissant $r|s$
  1. créer un nouvel état initial q
  2. mettre une ε-transition de q vers les états initiaux de $A(r)$ et $A(s)$
  3. (les états initiaux de $A(r)$ et $A(s)$ ne sont plus états initiaux)



- automate reconnaissant r^+
  mettre des ε-transition de chaque état terminal de $A(r)$ vers son état initial



Construction d'un AFN à partir d'une E.R.

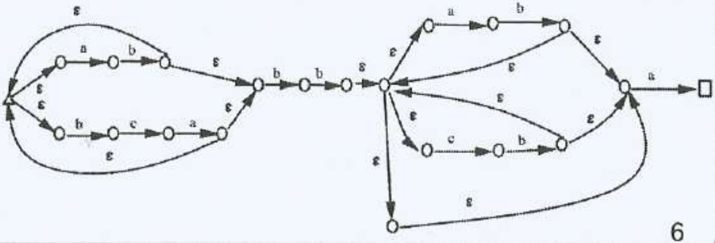
Exemple1

Donner un AFN des expressions régulières suivantes :

- a*
- a+
- a|b
- a*|b
- a*|b+
- a+|b+
- (a|b)+
- (a|b)*

Exemple1

Donner une ER que reconnaît l'automate suivant :



Automates finis déterministes (AFD)

Définitions

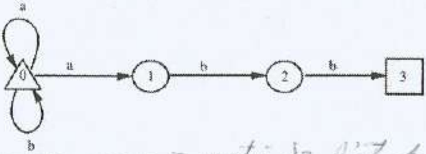
*On appelle ε-***transition***, une transition par le symbole ε entre deux états.*
Un automate fini est dit **déterministe** *lorsqu'il ne possède pas de ε-transition et lorsque pour chaque état e et pour chaque symbole a, il y a* **au plus un** *arc étiqueté a qui quitte e*

Remarques

L'automate donné en exemple précédemment qui reconnait $(a|b)^*abb$ n'est pas déterministe, puisque de l'état 0, avec la lettre a, on peut aller soit dans l'état 0 soit dans l'état 1.

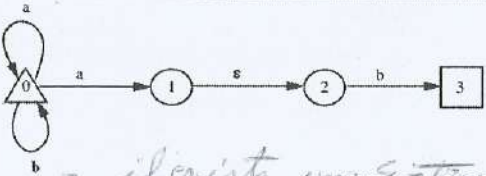
Les AFD sont plus faciles à simuler (pas de choix dans les transitions, donc jamais de retours en arrière à faire). Il existe des algorithmes permettant de **déterminiser** un automate non déterministe (c'est à dire de construire un AFD qui reconnait le même langage que l'AFN[1] donné). L'AFD obtenu comporte en général plus d'états que l'AFN, donc le programme le simulant occupe plus de mémoire.

Exemple1



AF-ND ~~car~~ à partir de l'état 0
il y a deux arcs étiquetés a

Exemple2



car il existe une ε-tran

Déterminisation d'un AFN ne contenant pas de ε_transition

Algorithme

1. Partir de l'état initial : $E = \{e_0\}$
2. Construire $E^{(1)}$ l'ensemble des états obtenus à partir de E par la transition a : $E^{(1)} = \Delta(E, a)$
3. Recommencer 2 pour toutes les transitions possibles et pour chaque nouvel ensemble d'état $E^{(i)}$
4. Tous les ensemble d'états $E^{(i)}$ contenant au moins un état terminal deviennent terminaux
5. Renuméroter alors les ensemble d'états en tant que simples états .

Exemple

état	a	b
0	0,2	1
1	3	0,2
2	3,4	2
3	2	1
4	-	3

$e_0 = 0$ et $T = \{2, 3\}$

Déterminisation d'un AFN ne contenant pas de Ɛ_transition

Solution

Définition

On appelle ε-fermeture de l'ensemble d'états $T = \{e_1, \ldots, e_n\}$ l'ensemble des états accessibles depuis un état e_i de T par des ε-transitions

ε-fermeture$(\{e_1, \ldots, e_n\} = \{e_1, \ldots, e_n\} \cup \{e$ tq $\exists e_i$ avec $i = 1, 2, \ldots, n$ tq $e_i \xrightarrow{\varepsilon}\xrightarrow{\varepsilon}\xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} e\}$

Calcul de ε transition

Mettre tous les états de T dans une pile P
Initialiser ε-fermeture(T) à T
Tant que P est non vide faire
  Soit p l'état en sommet de P
  dépiler P
  Pour chaque état e tel qu'il y a une ε-transition entre p et e faire
    Si e n'est pas déja dans ε-fermeture(T)
      ajouter e à ε-fermeture(T)
      empiler e dans P
    finsi
  finpour
fin tantque

Exemple

état	a	b	c	ε
0	2	-	0	1
1	3	4	-	-
2	-	-	1,4	0
3	-	1	-	-
4	-	-	3	2

$e_0 = 0$ et $T = \{4\}$

Déterminisation d'un AFN contenant de ε_transition

Algorithme

1. Partir de l'ϵ-fermeture de l'état initial
2. Rajouter dans la table de transition toutes les ϵ-fermetures des nouveaux "états" produits, avec leurs transitions
3. Recommencer 2 jusqu'à ce qu'il n'y ait plus de nouvel "état"
4. Tous les "états" contenant au moins un état terminal deviennent terminaux
5. Renuméroter alors les états.

Exemple

état	a	b	c	ε
0	2	-	0	1
1	3	4	-	-
2	-	-	1,4	0
3	-	1	-	-
4	-	-	3	2

$e_0 = 0$ et $T = \{4\}$

Minimisation d'un AFN

Objectif

obtenir un automate ayant le minimum d'états possible.
on définit des classes d'équivalence d'états par raffinements successifs. Chaque classe d'équivalence obtenue forme un seul même état du nouvel automate.

Algorithme

1 - Faire deux classes : A contenant les états terminaux et B contenant les états non terminaux.

2 - S'il existe un symbole a et deux états e_1 et e_2 d'une même classe tels que $\Delta(e_1, a)$ et $\Delta(e_2, a)$ n'appartiennent pas à la même classe, alors créer une nouvelle classe et séparer e_1 et e_2. On laisse dans la même classe tous les états qui donnent un état d'arrivée dans la même classe.

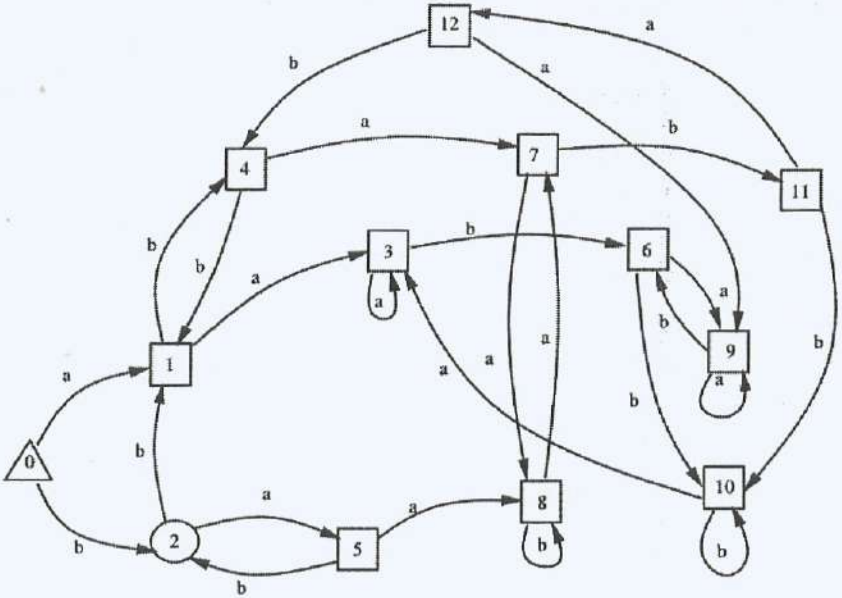
3 - Recommencer 2 jusqu'à ce qu'il n'y ait plus de classes à séparer.

4 - Chaque classe restante forme un état du nouvel automate

Exemple (voir diapo 13)

Minimisation d'un AFN

Exemple



Calcul d'une ER à partir d'un AEF

Définition

On appelle L_i le langage que reconnaîtrait l'automate si e_i était son état initial. On peut alors écrire un système d'équations liant tous les L_i :

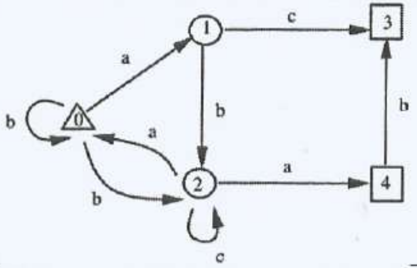
- chaque transition $\Delta(e_i, a) = e_j$ permet d'écrire l'équation $L_i = aL_j$
- pour chaque $e_i \in T$ on a l'équation $L_i = \varepsilon$
- les équations $L_i = \alpha$ et $L_i = \beta$ se regroupent en $L_i = \alpha|\beta$

On résoud ensuite le système (on cherche à calculer L_0) en remarquant juste que

Propriété

si $L = \alpha L|\beta$ *alors* $L = \alpha^*\beta$

Exemple (voir Slide 13)



Déterminisation d'un AFN (cas général)

Définition

On appelle ε-fermeture de l'ensemble d'états $T = \{e_1, \ldots, e_n\}$ l'ensemble des états accessibles depuis un état e_i de T par des ε-transitions

ε-fermeture$(\{e_1, \ldots, e_n\} = \{e_1, \ldots, e_n\} \cup \{e \text{ tq } \exists e_i \text{ avec } i = 1, 2, \ldots, n \text{ tq } e_i \xrightarrow{\varepsilon}\xrightarrow{\varepsilon}\xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} e\}$

Exemple