Theorie des Langages

Fiche

1 Introduction

- **tab titre**

...	...	...

2 Langages

Un langage est un ensemble de mots, qui peut etre defini :

Un langage est un ensemble de mots, qui peut etre defini :	
- En extension	liste exhaustive de tous les mots du langage. Exemple : un dictionnaire
- En comprehension	on commence une enumeration. Exemple : L={ab, aabb, aaabbb, ...}
- En intension	on se donne des 'regles' Exemple : Tous les mots formes de 'a' et de 'b' qui comportent autant d'occurences de 'a' que d'occurences de 'b' et dont tous les 'a' sont en debut de mot.
- Inductivement	

(Outils de definitions compacts des langages.)

3 Definition d'un langage

Les definitions inductives (en anglais : recursive definitions) Idee : 3 etapes :

1 une base d'objets appartenant a l'ensemble que l'on veut definir	2 des regles pour construire d'autres objets de l'ensemble a partir d'objets de la base ou d'objets deja construits.	3 declaration que les seuls objets de l'ensemble sont ceux construits en appliquant un nombre fini de fois les regles.

(base pas forcement minimale; on donne souvent une definition inductive sous la forme base+regles)
Exemple : Definissons l'ensemble PAIR des entiers pairs positifs : 1) Base : 2 appartient a PAIR , 2) Regle : Si x est dans PAIR, alors x+2 est dans PAIR
Pour montrer qu'un nombre est dans PAIR, on exhibera une suite d'application des regles.

4 Definition inductives

Definition inductive d'un ensemble pas unique : Exemple On peut definir PAIR par: 1)Base : 2 est dans PAIR , 2)Regle : si x et y dans PAIR, alors x+y dans PAIR Avantage de la 2e definition : les preuves qu'un nombre appartient a PAIR sont plus courtes.
Definition formelle de la fermeture inductive: Definition Soit U un univers et B$\subset$U une base, soit Ω une famille d'operations sur U. On appelle fermeture inductive de B par Ω la partie E de U definie par : - initialisation : B$\subset$E - construction : $\forall$f $\in \Omega$ et $\forall$x1,x2,...xn $\in$ E, si n est l'arite de f, si x=f(x1,x2,...xn) est defini, alors x$\in$E - fermeture : E est la plus petite partie de U qui contienne B et qui soit stable par Ω.
Principe pour decrire des langages par une grammaire (procede formel de construction inductive du langage) sous la forme d'un axiome (la base) et d'un ensemble de regles de production.

5 Definition sur les langages

Enoncé Definition	Exemple
Un alphabet Σ est un ensemble fini de caracteres.	Σ = {a,b}
Un mot (appele aussi une chaıne) ω sur Σ est une suite finie de symboles de Σ juxtaposes. Sa longueur (= nombre de caracteres) est notee —ω—. $\|\omega\|^a$ denote le nombre d'occurences de la lettre 'a'$\in \Sigma$ dans le mot ω.	Σ = {a,b}, ω = aabaa est un mot sur Σ; —ω— = 5, —ω—a = 4.
Le mot vide, ne contenant aucun symbole, est note ϵ. —ϵ—=0. ϵ peut etre un mot d'un langage, mais n'est pas une lettre de l'alphabet.	
Definition Si α et β sont 2 mots sur Σ, on appelle concatenation de α et β le mot $\alpha\beta$, note $\alpha \circ \beta$, $\alpha \bullet \beta$, $\alpha.\beta$, $\alpha\beta$	α = ab , β = cd $\alpha \bullet \beta$ = abcd
Concatenation avec le mot vide : $\alpha \bullet \epsilon = \epsilon \bullet \alpha = \alpha$. On notera an la concatenation de n occurences de a (n un nombre fini). a^0 denotera le mot vide ϵ.	
On appelle facteur ou sous-mot d'un mot ω un mot α tel qu'il existe 2 mots β et γ, avec $\omega = \beta\alpha\gamma$.	ω = abccdx cc est un facteur de ω.
Si $\omega = \alpha.\beta$ on dira que α est un prefixe de ω, et que β est un suffixe de ω.	ω =abccdx abc est un prefixe (propre). abccdx est prefixe (mais pas propre). dx est suffixe.
Soit Σ un alphabet, on appelle fermeture de Kleene de Σ, note $\Sigma *$, l'ensemble defini inductivement de la facon suivante -base : tous les caracteres de Σ ainsi que le mot vide ϵ sont dans $\Sigma *$. -regle : si x et y sont dans $\Sigma *$, alors xy est dans $\Sigma *$. $\Sigma *$ est l'ensemble des mots sur Σ, de longueur finie, plus le mot vide ϵ.	
Soit Σ un alphabet, $\Sigma +$ est l'ensemble defini inductivement de la facon suivante : -base : tous les caracteres de Σ sont dans $\Sigma +$. -regle : si x et y sont dans $\Sigma +$, alors xy est dans $\Sigma +$. $\Sigma +$ est l'ensemble des mots sur Σ, de longueur finie.	
On appelle langage (souvent note L) sur un alphabet Σ un sous-ensemble de $\Sigma *$.	
$\overline{L1}$ est l'ensemble des mots de $\Sigma *$ qui ne sont pas dans L1. $\overline{L1}$ s'appelle le complementaire de L1.	

6 Operations sur les langages

Operations:

$\cup$,$\cap$: Soient L1 et L2 2 langages sur l'alphabet Σ : $\rightarrow$L1 $\cup$L2 = {ω — $\omega \in$ L1 ou $\omega \in$ L2} $\rightarrow$L1 $\cap$L2 = {ω — $\omega \in$ L1 et $\omega \in$ L2}	Produit de 2 langages Soient L1 un langage sur l'alphabet Σ1 et L2 un langage sur l'alphabet Σ2 $\rightarrow$L1$\bullet$L2={ω1ω2 $\in$ (Σ1 $\cup$ Σ2)$*$, ω1 $\in$ L1 et ω2 $\in$ L2}	Fermeture de Kleene d'un langage : $\rightarrow$1 - L^0=ϵ $\rightarrow$2 - L^n=LL^{n-1}, $\forall$n$\geq$1 $\rightarrow$3 - $L*$= $\cup$ L^n, n$\geq$0 $\rightarrow$4 - L+=$\cup$ L^n, n¿0

7 Autres

L'ambiguïté Definition Un phrase ambiguë est une phrase a laquelle on peut attribuer plusieurs sens. En informatique : conflit. Definition : Une interpretation d'une phrase ambigue est un sens que l'on attribue a cette phrase. Exemple 1 : C'est la voiture de l'etudiant qui a coule une bielle. Exemple 2 : L'expression 2 + 3 $*$ 4 est ambigue : Interpretation 1 : (2+3)$*$4 (le resultat est 20) Interpretation 2 : 2+(3$*$4) (le resultat est 14)
Une hierarchie de langages Chomsky a defini une hierarchie des langages (la hierarchie de Chomsky) en 4 grandes classes. Type 0: Langages recursivement enumerables. | 1: Langages contextuels. | 2: Langages algebriques(context-free). | 3: Langages rationnels(reguliers). Propriete : On a Type 3 subsetneq Type 2 subsetneq Type 1 subsetneq Type 0 Ici étudde des langages de type 3 (les plus simples).

8 Les grammaires

Au depart :
Backus et Naur introduisent Backus-Naur Form : Metalangage introduit pour ALGOL6
Basee sur la definition inductive.
Moyen simple et elegant de d'ecrire toutes les phrases permises d'un langage (de programmation)

9 Backus-Naur Form

On va utiliser les regles d'ecrites par cette 'grammaire' pour construire la phrase.
On apppliquera a chaque etape une regle : c'est une etape de derivation.
S → E
E → E + E
E → (E)
E → 0E
E → 1E
E → 0
E → 1
= Factorise :
S → E
E → E + E |(E)|0E| 1E | 0 | 1
ou arbre syntaxique

10 Definition formelle des grammaires

Definition
Une grammaire est un quadruplet G = (N,T,P,S) ou :
•N est l'ensemble des symboles non terminaux
•T est l'ensemble des symboles terminaux : caracteres de l'alphabet
•P est un ensemble de regles de production, de la forme $\alpha \to \beta$, avec $\alpha \in (N \cup T)^+$,$\beta \in (N \cup T)^*$
•S = symbole de depart appele l'axiome

•Pour caracteres de N : on utilisera (habituellement) des majuscules.
•Pour caracteres de T : on utilisera (habituellement) des minuscules.
•Pour les regles de P, nos regles seront de la forme $X \to \beta$, avec $X \in N$ et $\beta \in (N \cup T)^*$.
•L'axiome, note S (habituellement), est la base de la definition inductive, et c'est la racine de tout arbre de derivation valide

Exemple :
S → E
E → E + E |(E)|0E| 1E | 0 | 1
N={ S,E }
T= {0,1,(,),+}
P= {S → E, E → E + E |(E)|0E| 1E | 0 | 1 }
On peut avoir des regles de production dont la partie droite est reduite a ϵ; on appelera ces regles des ϵ-productions.

Exemple :
S → E
E → E + E |(E)|0E| 1E | ϵ
Une autre grammaire pour le langage des expressions arithmetiques binaires simplissimes.

Definition
Pour une grammaire G, on note L(G) le langage engendre par G : c'est l'ensemble des mots que l'on peut definir a partir de l'axiome de G en appliquant un nombre fini de fois des regles de G.

Exemple : Grammaire G1 : S → aS S → bS S → a S → b S → ϵ
Exemple 2 :
Quel langage d'ecrit cette grammaire?
Comment pourrait-on simplifier cette grammaire?
Que se passe-t-il si on retire l'ϵ-production?

On veut traduire le langage sur $\Sigma = \{a,b\}$ ou tous les mots sont de la forme $\omega = \alpha$ aa β
Pour le mot : ω = abbaabb
S → AaaB
A → aA
A → bA
A → ϵ
B → aB
B → bB
B → ϵ
devient :
S → AaaB $_{(1)}$
A → aA $_{(2)}$ |bA $_{(3)}$ | ϵ $_{(4)}$
B → aB $_{(5)}$ |bB $_{(6)}$ | ϵ $_{(7)}$
(Les regles de production ont une numerotation implicite (de 1 a 7 ici).)

Arbre syntaxique
Definition
Un arbre syntaxique est un arbre dont la racine est l'axiome (S), dont les noeuds internes sont etiquetes par des symboles de N, et dont les feuilles sont etiquetees par des symboles de T ou par le mot vide ϵ. Chaque noeud interne correspond a une regle de production.
EXEMPLE

Pour un langage donne, il n'y a pas de grammaire unique !
Exemple :
les deux grammaires suivantes d'ecrivent le meme langage :
G1 : S → aS|bS|a|b| ϵ
G2 : S → aS|bS| ϵ
Definition On dit que deux grammaires G1 et G2 sont equivalentes, note G1 ∼ G2, si elles engendrent le meme langage, i.e. si L(G1) = L(G2).

11 Derivations

Si $\alpha \to \beta$ est une production de P, on note $\gamma 1 \alpha \gamma 2 \Rightarrow \gamma 1 \beta \gamma 2$. On dit qu'on a procede a une derivation. On dit que $\gamma 1 \beta \gamma 2$ se derive de $\gamma 1 \alpha \gamma 2$.
Exemple : S ⇒ AaaA et AaaA ⇒ aAaaA sont des derivations pour la grammaire G2 (S → AaaA; A → aA|bA| ϵ).
On peut etiqueter la derivation par le numero de la regle de production utilisee.

Pour un nombre fini de derivations successives $\gamma 1\alpha\gamma 2 \Rightarrow \gamma 1\beta\gamma 2 \Rightarrow \omega$, on ecrit : $\gamma 1\alpha\gamma 2 * \Rightarrow \omega$.
Exemples :
S $\Rightarrow$ AaaA $\Rightarrow$ aAaaA ou S * $\Rightarrow$ aAaaA
S * $\Rightarrow$ abbaabb

Definition
On appelle derivation gauche une suite de derivations obtenues en choisissant a chaque etape le symbole non terminal le plus a gauche. On definit de facon similaire la derivation droite.
Exemple de derivation gauche avec G2 : S $\rightarrow$ AaaB ;
A $\rightarrow$ aA|bA—ϵ; B $\rightarrow$ aB|bB| ϵ :
S (1) $\Rightarrow$ AaaB (2) $\Rightarrow$ aAaaB (4) $\Rightarrow$ aaaB (6) $\Rightarrow$ aaabB (7) $\Rightarrow$ aaab

Definition
On appelle langage engendre par une grammaire G = (N,T,P,S) l'ensemble des mots ω de T* tels que S * $\Rightarrow$ ω.
On le note L(G).On dit qu'un mot ω est engendre par une grammaire G si $\omega \in$ L(G)

12 Grammaires ambiguës

Definition
Une grammaire G est dite ambigue s'il existe un mot ω de L(G) qui admet au moins deux arbres syntaxiques differents.
Exemple :
G1 : G1 : S $\rightarrow$ aS|bS|a|b| ϵ
Le mot ω = ab admet deux arbres syntaxiques differents :

ARBRE SYNTAXIQUE
Definition
Un langage est dit ambigu si toutes les grammaires qui l'engendrent sont ambigues.
Exemple : G1 : S $\rightarrow$ aS|bS|a|b| ϵ
G2 : S $\rightarrow$ aS|bS| ϵ
L(G1) = L(G2); G1 est ambigue mais G2 n'est pas ambigue, donc L(G1) = L(G2) n'est pas un langage ambigu.

13 Grammaires regulieres

Definition
Une grammaire G est dite reguliere si toutes ses regles de production sont de la forme :
A $\rightarrow$ αB, avec : A $\in$ N,$\alpha \in$ T*,B $\in$ N ou B = ϵ
Exemple : G1 : S $\rightarrow$ aS|bS|a|b| ϵ

14 Decidabilite

Definition
Un probleme est dit indecidable si il n'existe pas (et il ne peut pas exister) d'algorithme generique pour le resoudre.

Les problemes suivants sur les grammaires sont indecidables : Deux grammaires G1 et G2 sont-elles equivalentes? Deux grammaires engendrent-elles des langages ayant un mot en commun? Y a-t-il des mots qu'une grammaire n'engendre pas?

15 Hierarchie de Chomsky sur les grammaires

tab titre

Type	Nom	Type de production
0	Langages recursivement enumerables	X $\rightarrow$ Y X $\in$N+ , Y $\in$N (N $\cup$T)
1	Langages contextuels	X$\rightarrow$Y X $\in$NN+ , Y$\in$N(N$\cup$T)* , |Y|$\geq$|X|
2	Langages context-free ou algebriques	X$\rightarrow$Y X $\in$NN , Y$\in$N(N$\cup$T)*
3	Langages rationnels (reguliers)	X$\rightarrow$Y A $\rightarrow$ αB, avec : A $\in$ N,$\alpha \in$ T*,B $\in$ N ou B = ϵ

16 Langages rationnels

17 Introduction aux langages rationnels

Hierarchie de Chomsky :
Classe 3
subsetneq Classe 2 deterministes subsetneq Classe 2 non deterministes
subsetneq Classe 1
subsetneq Classe 0

Classe 3 de la hierarchie de Chomsky : Langages rationnels ('regular languages')
langages les plus simples, les plus rapides, et aussi les moins puissants.
Ils servent : - en compilation a assurer l'analyse lexicale (segmentation d'un flot de caracteres en 'mots') - pour la recherche de motifs - pour le traitement de texte - etc.

Ces langages sont caracterises de plusieurs facons : ils sont :
1 engendres par une grammaire reguliere.
2 d'ecrits par une expression reguliere.
3 engendres par un automate d'etats finis.
Rappel :
Definition
Une grammaire G est dite reguliere si toutes ses regles de production sont de la forme : A $\rightarrow$ αB, avec : A $\in$ N,$\alpha \in$ T*,B $\in$ N ou B = ϵ
Exemple : G1 : S $\rightarrow$ aS|bS|a|b| ϵ
Theoreme Un langage est rationnel ssi il existe une grammaire reguliere qui l'engendre.
Exemple de grammaire regliere : langage des mots sur Σ = {a,b} qui contiennent le facteur aa :
S $\rightarrow$ aS|bS|aA
A $\rightarrow$ aB
B $\rightarrow$ aB|bB| ϵ

18 Les expressions regulieres

Une expression reguliere dèecrit un langage rationnel avec une syntaxe particuliere, et correspond a une grammaire reguliere. Exemples d'expressions regulieres sur Σ = {a,b} : 1 (a + b)* est l'ensemble des mots sur {a,b} 2 (a + b)*aa(a + b)* langage des mots sur Σ = {a,b} qui contiennent le facteur aa 3 b+(a + b)* : mots sur l'alphabet Σ = {a,b} qui commencent par un ou plusieurs b

Une expression reguliere est une expression algebrique qui permet de decrire un langage rationnel.
Definition
Definition inductive des expressions regulieres :
- Base : $\emptyset$, ϵ et les caracteres de Σ sont des expressions regulieres, representant respectivement les langages $\emptyset$,ϵ, {x} si x $\in \Sigma$.
- Regles : si r et s sont des expressions regulieres representant les langages R et S, alors (r + s), r.s, r* et r+ sont des expressions regulieres representant

respectivement les langages R ∪S, R.S, R* et R+.
En anglais : 'regular expression'

1 (a + b)* est l'ensemble des mots sur {a,b}
2 (a + b)*aa(a + b)* langage des mots sur Σ = {a,b} qui contiennent le facteur aa
3 b+(a + b)* : mots sur l'alphabet Σ = {a,b} qui commencent par un ou plusieurs b

On notera L(r) le langage (rationnel) d'ecrit par l'expression reguliere r, et on dira que r engendre le langage L(r).
Souvent, par abus de langage, on confond expression reguliere et langage engendre!

Remarques :
- r + s se note aussi r|s
- r.s se note aussi rs
- * a precedence sur + : a + b* s'interprete comme (a + (b*))

Proprietes :
1 (r*)* = r*
2 r(r*) = (r*)r = r+
3 (a*b*)* = (a + b)*

Un meme langage rationnel peut etre d'ecrit par plusieurs expressions regulieres differentes.
Exemple : langage des mots sur Σ = {a,b} contenant le facteur 'aa' :
r = (a + b)*aa(a + b)* s = b*aa(a + b)*
On a L(r) = L(s)

Definition
Si L(r)=L(s), on dira que r et s sont des expressions regulieres equivalentes, note r ∼s.

Theoreme Un langage est rationnel si et seulement si il existe une expression reguliere le reconnaissant.

19 Les automates d'etats finis

Les automates d'etats finis (AEF) (en anglais : "Finite State Automata" ou FA)
Ce sont les 'machines' reconnaissant les langages rationnels.

Un systeme a etats finis est un modele mathematique "discret". Il est compose d'un nombre fini de configurations, appelees des etats, et d'actions permettant de passer d'un etat a un autre. Les automates d'etats finis sont des systemes a etats finis particuliers.

Un automate d'etats finis est un graphe oriente fini dont les arcs sont etiquetes. Il est compose :
1 d'un nombre fini de configurations (les etats), qui sont les sommets du graphe
2 d'actions permettant de passer d'un etat a un autre (ces actions etiquettent les arcs du graphe)

SCHEMA AUTOMATE

De plus, on a un (unique) etat initial
(START) et 0, 1 ou plusieurs etats finaux (STOP)

Etiquetage de l'etat initial et des etats finaux :
1 l'etat initial est note par une fleche
2 les etats finaux ont un double cerclage

SCHEMA AUTOMATE

Principe (informel) : On part de l'etat initial (q0) et on parcourt le graphe jusqu'a ce qu'on decide de s'arreter sur un etat final (ici q2 ou q4).

SCHEMA AUTOMATE

Ce parcours definit un mot du langage reconnu par l'automate.

SCHEMA AUTOMATE

Le parcours definit le mot 'aabba'.
Definition On peut etiqueter un arc d'un automate d'etats finis par le mot vide ϵ (souvent note par l'absence d'un caractere), il correspond a une ϵ-transition.

On peut etiqueter une transition par plusieurs caracteres : on en choisit un seul. On peut avoir des circuits, des boucles de reflexivite. On peut avoir plusieurs arcs sortants etiquetes par un meme caractere.

Exemple :
SCHEMA AUTOMATE
langage reconnu : L(A) = ab,ac

Exemple :

SCHEMA AUTOMATE

Un automate d'etats finis est donc defini par :
- un nombre fini d'etats Q (les sommets du graphe)
- un alphabet Σ.
- un ensemble fini δ de transitions (les arcs du graphe), etiquetes chacune par une (ou plusieurs) lettre(s) de Σ ou par ϵ

SCHEMA AUTOMATE

Parmi les etats de Q, on distingue :
l'etat initial $q0 \in Q$ (il y a exactement un etat initial)
les etats finaux, qui constituent l'ensemble $F \subset Q$ (il peut y avoir plusieurs ou meme aucun etat final)

Formellement : Definition Un automate d'etats finis est un quintuplet A = (Q,Σ,δ,q0,F), ou :
1 Q est un ensemble d'etats (de cardinal fini)
2 Σ est un alphabet (de cardinal fini)
3 δ est une fonction de transition (qui permet de passer d'un etat a un autre)

4 q0 $\in$ Q est l'etat initial
5 F $\subseteq$ Q est l'ensemble des etats finaux

Exemple classique d'un automate (extrait du livre de Hopcroft and Ullman : Introduction to Automata Theory, Languages and Computation)
Le probleme du passeur, du loup, de la chevre et du chou.

Question :
Quel est l'ensemble des solutions qui permettent au passeur d'emmener de la rive droite a la rive gauche le chou, la chevre et le loup, avec une barque ne pouvant contenir que l'un des trois, sans laisser seuls ensemble ni le loup et la chevre, ni la chevre et le chou?

SCHEMA AUTOMATE

(M - Man), le loup (W - Wolf), la chevre (G - Goat) et le chou (C - Cabbage). Cet automate modelise toutes les solutions possibles.

On peut deduire de cette modelisation par automate d'etats finis :
1 qu'il y a une solution au probleme.
2 qu'il y a deux plus courtes solutions etiquetees GMWGCMG et GMCGWMG.
3 qu'il existe une infinite de solutions (le langage engendre par l'automate est infini).

SCHEMA AUTOMATE

Definition On dit qu'un automate d'etats finis A = (Q,Σ,δ,q0,F)
accepte un mot ω de Σ* si et seulement si il existe (au moins) un chemin dans A allant de q0 a un etat final, etiquete par les lettres successives de ω, entre lesquelles on a eventuellement intercale des occurences de ϵ.
Le langage L(A) reconnu par A est l'ensemble des mots que A accepte.
Exemple 2 :

SCHEMA AUTOMATE
langage reconnu : aab*a + b+ On peut avoir 2 transitions possibles avec la meme lettre : ici sur q2, 2 transitions avec 'b'.

On definit la table de transitions d'un automate d'etats finis A = (Q,Σ,δ,q0,F), qui d'ecrit la fonction de transition δ.
Sur l'exemple 2 :

	a	b
q0	{q1}	{q2}
q1	{q3}	{$\emptyset$}
q2	{$\emptyset$}	{q2,q4}
q3	{q4}	{q3}
q4	{$\emptyset$}	{$\emptyset$}

On va voir qu'il existe plusieurs sortes d'AEF. Il existe des automates plus complexes (automates a pile, machine de Turing).
SCHEMA AUTOMATE

20 Equivalences d'automate

il existe plusieurs types d'Automates d'Etats Finis
ils sont equivalents

21 Le probleme du determinisme

La definition d'un automate d'etats finis n'interdit pas les "conflits".
L = aab*a + ab*b

SCHEMA AUTOMATE

Comment doit-on interpreter δ(q0,a)?

Le choix est non-deterministe.

Definition Un automates d'etats finis deterministe (AEFD) (en anglais : Deterministic Finite Automaton (DFA)) est un automate d'etats finis tel que, de chaque etat q$\in$Q, il part —Σ— transitions, une pour chacune des lettres de l'alphabet Σ.
Remarque : Pas d'ϵ-transition!

Dans un automate deterministe, on a un 'etat poubelle', vers lequel on envoie toutes les transitions non definies.
Souvent, cet etat poubelle est implicite.

Exemple d'automate deterministe sur Σ = a,b pour L = aab*a + ab*b :

SCHEMA AUTOMATE

22 Differentes sortes d'AEF

On definit 3 sortes d'automates d'etats finis :
1 les automates d'etats finis non-deterministes sans ϵ-transition (NFA-W, W pour"Without")
2 les automates d'etats finis non-deterministes avec ϵ-transition (NFA-ϵ)
3 les automates d'etats finis deterministes

Exemple : L = a,ab,ba *

L = a,ab,ba * non deterministe avec ϵ -transition
SCHEMA AUTOMATE

L = a,ab,ba * non deterministe sans ϵ -transition

SCHEMA AUTOMATE

L = a,ab,ba * deterministe

SCHEMA AUTOMATE

L = a,ab,ba *

SCHEMA AUTOMATE

Theoreme
La classe des langages reconnus par :
- les automates d'etats finis deterministes
- les automates d'etats finis non-deterministes sans ϵ-transition
- les automates d'etats finis non-deterministes avec ϵ-transition est la meme : celle des langages rationnels.
Preuve : constructive (algorithmes de passage d'un type d'AEF a un autre)

23 Determinisation d'un AEF avec ϵ-transitions

- On part d'un AEF non deterministe A1 = (Q,Σ,δ,q0,F) sans ϵ-transition.
- On calcule un automate d'etats finis deterministe A2 = (Q0,Σ,δ0,Q0 0,F0), avec Q0 0 = {q0}

Principe : On construit les etats et la table de transition de δ0 :
1 On construit la table de transition de δ (qui comporte des ensembles d'etats)

2 Initialisation de δ0
- on commence par Q'0 ={q0}
- on applique chaque caractere x de Σ a Q'0
- on obtient un ensemble d'etats qui est sera etat de 2'X

3 Construction de δ'
- on choisit un etat Q0 de 2'X non encore traite
- on applique chaque caractere x de Σ chaque etat de Q0 avec δ
- on obtient un ensemble d'etats 1 si cet ensemble ne correspond pas a un ete deja defini de 2'X, on cree un nouvel etat de 2'X

4 les etats finaux de A2 sont ceux qui contiennent au moins un etat final de F

Algo DETERMINISATION Donnee : un automate A1 = (Q,Σ,δ,q0,F) Resultat : un automate deterministe A2 = (Q0,Σ,δ0,Q0 0,F0). Initialisation : Q0 0 $\leftarrow$q0;ATRAITER $\leftarrow$ Q0 0 = q0; Q0 $\leftarrow$Q0 0; tant que ATRAITER 6= $\emptyset$ faire CHOISIR Q0 dans ATRAITER; ATRAITER $\leftarrow$ ATRAITER - Q0; pour chaque caractere x de Σ faire pour chaque etat Q de Q0 faire δ0(Q0,x) $\leftarrow$ δ0(Q0,x)$\cup\delta$(Q,x); si δ0(Q0,x) n'est pas un etat de Q0 alors Q00 $\leftarrow$ δ0(Q0,x); Q0 $\leftarrow$ Q0 +Q00; ATRAITER $\leftarrow$ ATRAITER + Q00; pour chaque etat Q0 de Q0 contenant un etat de F faire AJOUTER Q0 a F0; Retourner((Q0,Σ,δ0,Q0 0,F0)).

Exemple : L = {ab,ac}
δ a b c q0 {q1,q3} {$\emptyset$} {$\emptyset$} q1 {$\emptyset$} {q2} {$\emptyset$} q2 {$\emptyset$} {$\emptyset$} {$\emptyset$} q3 {$\emptyset$} {$\emptyset$} {q4} q4 {$\emptyset$} {$\emptyset$} {$\emptyset$}

Initialisation : Q0 0 = {q0}
δ0(Q0 0,a) = {q1,q3} : on cree un nouvel etat Q0 1 = q1,q3 δ0(Q0 0,b) = {$\emptyset$}; δ0(Q0 0,c) = {$\emptyset$}

On a un etat Q'1 = {q1,q3} qui n'est pas traite :
δ0(Q'1,a) = $\emptyset$ δ0(Q'0,b) = {q2} : on cree un nouvel etat Q'2 = {q2} δ0(Q'0,c) = {q4} : on cree un nouvel etat Q'3 = {q4}

On traite l'etat Q'2 = {q2} : δ0(Q'2,a) = δ0(Q'2,b) = δ0(Q'2,c) = $\emptyset$ On traite l'etat Q'3 = {q4} : δ0(Q'3,a) = δ0(Q'3,b) = δ0(Q'3,c) = $\emptyset$ Etats finaux : Q'2 parce qu'il contient q2, et Q'3 parce qu'il contient q4.

A la fin, on obtient :
δ a b c q0 {q1,q3} {$\emptyset$} {$\emptyset$} q1 {$\emptyset$} {q2} {$\emptyset$} q2 {$\emptyset$} {$\emptyset$} {$\emptyset$} q3 {$\emptyset$} {$\emptyset$} {q4} q4 {$\emptyset$} {$\emptyset$} {$\emptyset$}

δ0 a b c
Q0 0={q0} {q1,q3}=Q0 1 {$\emptyset$} {$\emptyset$} Q0 1={q1,q3} {$\emptyset$} {q2}=Q0 2 {q4}=Q0 3 Q0 2=q2 {$\emptyset$} {$\emptyset$} {$\emptyset$} Q0 3=q4 {$\emptyset$} {$\emptyset$} {$\emptyset$}

Remarque : Etant donne un AEF non deterministe a k etats, l'AEF deterministe correspondant peut avoir 2k etats.

On etend la technique de determinisation en etendant la fonction de transition δ0(Qi,x) a une fonction donnee par les mots ϵ*xϵ*.

Definition
On appelle ϵ-fermeture d'un etat q l'ensemble des etats qi atteignables a partir de q par un chemin etiquete uniquement par le mot vide ϵ.
Definition
On appelle ϵ-fermeture d'un ensemble Q d'etats l'union des ϵ-fermetures des etats appartenant a Q.

Exemple : a*b*c*

SCHEMA AUTOMATE

ϵ-fermeture(q0) = {q0,q1,q2}
ϵ-fermeture(q1) = {q1,q2}
ϵ-fermeture(q2) = {q2}

Principe de determinisation :
Pour un etat Q0 de l'AEF deterministe en cours de calcul :
1 On part de l'ϵ-fermeture de Q0.
2 On calcule δ(Q0)
3 On calcule l'ϵ-fermeture de δ(Q0)
4 On obtient un etat du nouvel automate
Les etats finaux du nouvel automate sont ceux qui contiennent au moins un etat final de l'automate de depart.

Exemple : a*b*c* :

SCHEMA AUTOMATE

δ a b c
ϵ q0 {q0} {$\emptyset$} {$\emptyset$} {q0,q1,q2} q1 {$\emptyset$} {q1} {$\emptyset$} {q1,q2} q2 {$\emptyset$} {$\emptyset$} {q2} {q2}
Construction de δ0, Q0 0=ϵ-fermeture(q0)={q0,q1,q2}

a*b*c*

SCHEMA AUTOMATE

$\delta 0$ a b c
Q0 0 = {q0,q1,q2} {q0}→ Q0 0 {q1}→{q1,q2}=Q0 1 {q2}→{q2}= Q0 1 = {q1,q2} {∅} {Q0 1} {Q0 2} Q0 2 = {q2} {∅} {∅} {Q0 2}

a*b*c*

SCHEMA AUTOMATE

24 Minimisation d'un AEF deterministe

Theoreme
(de Nerode - Myhill) : Pour un langage rationnel donne L, il existe un automate d'etats finis deterministe canonique (uniquement defini), et qui comporte un nombre minimum d'etats (parmi tous les automates deterministes), reconnaissant L.
Il existe un algorithme tres efficace de minimisation.

Principe de minimisation d'un automate d'etats finis deterministe : utilise le principe algorithmique d'eclatement de partitions.

Rappel : une partition d'un ensemble est la definition d'un ensemble de classes, tel que l'union de toutes les classes est l'ensemble de depart et l'intersection de deux classes est vide (une partition correspond a une relation d''equivalence)
Principe algorithmique d'eclatement de partitions (ou d'affinement de partitions)
1 on part d'une (ou plusieurs) (grandes) classes
2 on a un critere qui permet de partitionner un classe en plusieurs classes plus petites
3 on arrete quand chaque classe obtenue est non-partitionnable

Pour minimiser un AEF deterministe :
1 on retire les etats non atteignables;
2 on partitionne l'ensemble des etats en deux classes : 1 les etats finaux 2 les etats non finaux (y compris l'etat poubelle ∅)

Etape d'eclatement d'une classe Ci :
1 appliquer a Ci une transition par un caractere x de $\in \Sigma$;
2 separer les elements de Ci qui n'aboutissent pas a la meme classe

On repete jusqu'a ce qu'il n'y ait plus d'eclatement possible.
A la fin, on a pour toute classe Ci obtenue : $\forall$x $\in \Sigma$, $\forall$q,q0 $\in$ Ci, δ(q,x) = δ(q0,x)
On obtient la description d'un nouvel AEF deterministe, dont l'etat initial est l'etat contenant q0 et dont les etats finaux sont les etats contenant un etat final de l'automate de depart.

Exemple : L = {ab,ac}
SCHEMA AUTOMATE

On part de C1 = {q3,q4} et C2 = {q0,q2,∅}
C2 = {q0,q2,∅} avec b se partitionne en : {q0,∅}|{q2}
{q0,∅} avec a se partitionne en : {q0}|{∅}
{q3,q4} ne se partitionne ni avec a ni avec b.
On obtient finalement la partition :
∅ |{q0}|{q2}|{q3,q4}
On obtient un automate d'etats finis deterministe a trois etats :

SCHEMA AUTOMATE

Algo MINIMISATION Donnee : un automate deterministe A1 = (Q,Σ,δ,q0,F) Resultat : l'automate deterministe minimum A2 = (Q0,Σ,δ0,Q0 0,F0). Initialisation : C ←{Q -F,F}; b ← 1; SUPPRIMER de A1 les etats non atteignables; tant que b=1 faire b ← 0; pour chaque classe C de C faire pour chaque caractere x de Σ faire si par δ on n'aboutit pas dans une meme classe de C alors REMPLACER C dans C par les classes obtenues; b ← 1; δ0 ← fonction de passage d'une classe de C a une autre; F0 ← ensemble des classes de C classes contenant au moins un etat de F ; Retourner(C,Σ,δ0,q0,F0).

Theoreme
Pour un langage rationnel L donne, il existe un unique automate d'etats fini deterministe minimum engendrant L.
Consequence fondamentale : Les langages rationnels sont non ambigus.

25 Langages rationnels : passages d'une representation a une autre

Rappel :
Theoreme Un langage L est rationnel (classe 3) ssi il existe une grammaire reguliere G telle que L(G) = L ssi il existe un automate d'etats finis reconnaissant L ssi il existe une expression reguliere engendrant L
La preuve est constructive, sous forme d'algorithmes de passage :
Automate d'etats finis – Grammaire reguliere
– Expression reguliere

26 Automate d'etats finis vers Grammaire reguliere

Automate d'etats finis → Grammaire reguliere :
Exemple :

SCHEMA AUTOMATE

On represente l'etat initial q0 par l'axiome S On represente chaque autre etat Qi par un non-terminal Ai Si Qi est un etat final, on ajoute une ϵ -transition Ai → ϵ

Automate d'etats finis → Grammaire reguliere :
Exemple :
SCHEMA AUTOMATE

S → bS|aA1 A1 → bS|aA2 A2 → aA2—bA2—ϵ

Cas particuliers : S → bS|aA A → bB B → aB| ϵ
peut se reecrire : S → bS|abB B → aB| ϵ

$S \rightarrow bS|aA$ $A \rightarrow aA|bB$ $B \rightarrow \epsilon$
peut se reecrire : S → bS—aA A → aA—b

27 Grammaire reguliere vers Automate d'etats finis

Grammaire reguliere → Automate d'etats finis :
On reecrit la grammaire reguliere pour obtenir des regles de production de la forme :
$X \rightarrow aY$ ou $X \rightarrow \epsilon$, avec $X,Y \in N$ et $a \in T$

Exemple 1 : $S \rightarrow aaA$ $A \rightarrow bA|\ \epsilon$
se reecrit : $S \rightarrow aA1$ $A1 \rightarrow aA2$ $A2 \rightarrow bA2|\ \epsilon$

Exemple 2 :
$S \rightarrow aS|a$
se reecrit : $S \rightarrow aS|aA1$ $A1 \rightarrow \epsilon$

On represente ensuite l'axiome S par l'etat q0 et le non-terminal Ai par l'etat qi
Les etats finaux correspondent aux regles de la forme; $Ai \rightarrow \epsilon$.

Exemple : $S \rightarrow aaA$ $A \rightarrow bA|\ \epsilon$
se reecrit : $S \rightarrow aA1$ $A1 \rightarrow aA2$ $A2 \rightarrow bA2|\ \epsilon$

SCHEMA AUTOMATE

28 Expression reguliere vers Automate d'etats finis

Rappels :
Definition inductive des expressions regulieres :
Definition
Definition inductive des expressions regulieres :
- Base : $\emptyset$, ϵ et les caracteres de Σ sont des expressions regulieres, representant respectivement les langages $\emptyset$,$\{\epsilon\}$, $\{x\}$ si $x \in \Sigma$.
- Regles : si r et s sont des expressions regulieres representant les langages R et S, alors (r + s), r.s, r* et r+ sont des expressions regulieres representant respectivement les langages $R \cup S$, R.S, R* et R+.

Pour construire un AEF, on applique cette definition inductive : base :

SCHEMA AUTOMATE

regles de construction de l'automate :
r1 + r2 r1r2 r*
SCHEMA AUTOMATE

Pour construire r+, on fera rr*
Exemple : (a + b)*aba
SCHEMA AUTOMATE

29 Automate d'etats finis vers Expression reguliere

Algorithme de Mac Naughton et Yamada : 1. Transformation de l'automate en automate generalise 2. Transformation de l'automate generalise en expression reguliere
Bibliographie et figures : Cours d'Alexis Nasr

Definition Automate generalise : les transitions sont etiquetees par des expressions regulieres (ou par $\emptyset$).

Transformation d'un automate en automate generalise :
1. ajouter un nouvel etat initial avec une ϵ- transition vers q0
2. ajouter un nouvel etat final vers lequel les anciens etats finaux sont envoyes par une ϵ- transition
3. des transitions etiquetees par $\emptyset$ sont ajoutees entre les etats qui ne sont relies par aucune transition, mais entre lesquels il existe un chemin dans l'automate de depart (ces transitions ne peuvent pas etre franchies)

Exemple d'initialisation a un automate generalise
SCHEMA AUTOMATE

A chaque iteration, on supprime un etat. A la fin, il reste une seule transition de l'etat initial a l'etat final, etiquetee par l'expression reguliere recherchee.

Exemple : on supprime l'etat 1 :

SCHEMA AUTOMATE

Exemple (suite) : on supprime l'etat 2 :

SCHEMA AUTOMATE

On obtient l'expression reguliere b*a(a|bb*a)*

30 Applications

Montrer qu'un langage est rationnel Montrer que deux expressions regulieres sont equivalentes Trouver une grammaire reguliere Ameliorer une grammaire reguliere ...

31 Hierarchie de Chomsky