

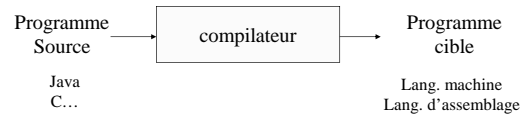
Théorie des langages et Compilation

Claire.Lefevre@info.univ-angers.fr

1

Compilation

- Traduction d'un programme écrit dans un premier langage (langage source) en un programme équivalent écrit dans un autre langage (langage cible)



2

quelques tâches d'un compilateur

- Analyse lexicale : lecture du programme source pour regrouper les caractères en unités lexicales (« catégories de mots »)
- Analyse syntaxique (ou grammaticale) : on regroupe les unités lexicales (« mots ») en structures grammaticales (« phrases ») qui seront généralement représentées par un arbre
- Production de code intermédiaire : construction d'un programme écrit dans un langage de + bas niveau

=> En compilation, on étudie des méthodes, des techniques, des algorithmes efficaces pour réaliser ce type de tâches

3

Les langages : ils sont partout

- Informatique :
 - Langages de programmation « évolués »
 - Langages machine
 - Protocoles de communication
 - Adresses IP, adresses web...
- « naturels » :
 - Français, anglais, chinois...
- Musique :
 - Do ré ré mi do do ré
- Génétique :
 - Codes ADN : ATCTACGTAAG

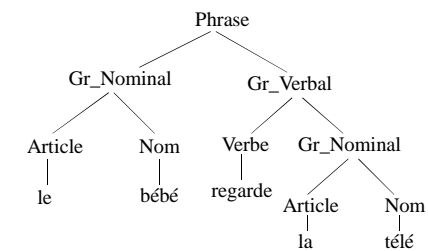
4

que fait-on avec ?

- Les décrire : caractériser les expressions « bien formées » d'un langage
 - INSTR : BLOC
 - | if EXPR_PAR INSTR [else INSTR]
 - | do INSTR while EXPR_PAR
 - ...
 - Phrase → Gr_Nominal Gr_Verbal
 - Gr_Nominal → Article Nom
 - Gr_Nominal → Article Adjectif Nom
 - Gr_Verbal → Verbe Gr_Nominal
 - Gr_Verbal → Verbe
 - ...

5

- Vérifier qu'une expression est bien formée, et construire sa structure
 - Le bébé regarde la télé bien formé
 - Bébé le télé regarde la mal formé



6

- « transformer » une expression
 - Traduire dans un autre langage (compilation par exemple)
 - « calculer » quelque chose (calculatrice, ...)
 - Rechercher des motifs
 - egrep, awk, lex...
- ```
$ cat fich
If (x <= y){
 y = x;
}
else {
 x = y-x;
}
$ egrep -n '= y' fich
1: If (x <= y){
5: x = y-x;
$
```

7

## Théorie des langages Quoi ? Pourquoi ?

- Étude de « machines » abstraites
- Pour décrire, analyser, travailler efficacement sur les langages
- Pour modéliser la notion de calcul fini afin d'étudier
  - quels problèmes on est capable de résoudre
  - avec quelle efficacité

8

## Historique

- Années 1930 : A. Turing => existence de pbs pour lesquels il n'existe pas de calcul fini pouvant fournir un résultat dans tous les cas
- Années 1940-50 : automates finis
- Fin années 1950 : N. Chomsky, grammaires formelles
- Fin années 1960 : S. Cook étend les machines de Turing => étude de la « complexité » de pbs

9

## Deux grands types d'utilisation

- Les automates et les grammaires sont utilisés pour la conception et le développement de logiciels (en particulier en compilation)
- Les machines de Turing nous aident à comprendre quels problèmes on est capable de résoudre en temps fini et, parmi ceux-ci, quels problèmes ne sont pas résolubles en temps « raisonnable »

10

# Langages

## Concepts de base

11

## Alphabets et chaînes

- Un alphabet  $\Sigma$  est un ensemble fini, non vide, de symboles
  - Ex :  $\Sigma = \{a, b, c\}$
- Un mot ou une chaîne  $\omega$  formé(e) sur un alphabet est une suite finie  $s_1 s_2 \dots s_n$  de symboles de cet alphabet
  - Ex :  $\omega = abaa$
- La chaîne vide, notée  $\varepsilon$ , est une chaîne ne contenant aucun symbole
- La longueur d'une chaîne  $\omega$ , notée  $|\omega|$ , est le nombre de symboles composant la chaîne  $\omega$ 
  - $|abaa| = 4$        $|\varepsilon| = 0$

12

## Opérations sur les chaînes

- La concaténation de 2 chaînes  $u$  et  $v$ , notée  $u.v$  ou  $uv$ , est la chaîne obtenue en juxtaposant  $u$  et  $v$

si  $u = a_1 a_2 \dots a_n$  et  $v = b_1 b_2 \dots b_p$   
alors  $uv = a_1 a_2 \dots a_n b_1 b_2 \dots b_p$

- Puissances d'une chaîne  $\omega$

- $\omega^k$  est la chaîne formée par la concaténation de  $k$  occurrences de  $\omega$

$\omega^k = \underbrace{\omega \omega \omega \dots \omega}_k$   
k fois

- $\omega^0 = \varepsilon$

13

- Un préfixe d'une chaîne  $\omega$  est une suite, éventuellement vide, de symboles débutant  $\omega$
- Un suffixe de  $\omega$  est une suite de symboles terminant  $\omega$
- Une sous-chaîne (ou facteur) d'une chaîne  $\omega$  est une suite de symboles apparaissant consécutivement dans  $\omega$

si  $\omega = x.u.y$   
alors  $x$  est un préfixe de  $\omega$   
 $y$  est un suffixe de  $\omega$   
 $u$  est une sous-chaîne

- Notation :  $|\omega|_x$  est le nombre d'occurrences de la chaîne  $x$  dans la chaîne  $\omega$

14

## Langages

- Un langage est un ensemble de chaînes sur un alphabet  $\Sigma$
- Le langage vide, noté  $\emptyset$ , ne contient aucune chaîne
- Attention :  $\emptyset \neq \{\varepsilon\}$
- Le langage « plein », noté  $\Sigma^*$ , contient toutes les chaînes que l'on peut former sur l'alphabet  $\Sigma$
- $\Sigma^+$  contient toutes les chaînes *non vides* sur  $\Sigma$

15

## Opérations sur les langages

- L'union de  $A$  et  $B$  est composée de toutes les chaînes qui apparaissent dans l'un au moins des langages  $A$  ou  $B$  :  
 $A \cup B = \{\omega \mid \omega \in A \text{ ou } \omega \in B\}$
- L'intersection de  $A$  et  $B$  est composée des chaînes apparaissant à la fois dans  $A$  et dans  $B$  :  
 $A \cap B = \{\omega \mid \omega \in A \text{ et } \omega \in B\}$
- La différence de  $A$  et  $B$  est le langage composé des chaînes de  $A$  n'apparaissant pas dans  $B$  :  
 $A \setminus B = \{\omega \mid \omega \in A \text{ et } \omega \notin B\}$
- Le complémentaire de  $A$  sur un alphabet  $\Sigma$  comprend toutes les chaînes de  $\Sigma^*$  n'apparaissant pas dans  $A$  :  
 $\bar{A} = \Sigma^* \setminus A$

16

- La concaténation de 2 langages  $A$  et  $B$  est le langage, noté  $A.B$  ou  $AB$ , composé de toutes les chaînes formées par une chaîne de  $A$  concaténée à une chaîne de  $B$  :

$$A.B = \{u.v \mid u \in A, v \in B\}$$

- Ex :  $A = \{ab, a\}$   $B = \{bc, b\}$   
 $A.B = \{abbc, abb, abc, ab\}$

- Puissances d'un langage  $A$  :

$A^k$  est le langage formé par la concaténation de  $k$  occurrences de  $A$

- $A^0 = \{\varepsilon\}$
- $A^{n+1} = A^n.A$  (ou  $A.A^n$ )

$A^k$  : « mots formés par la concaténation de  $k$  mots de  $A$  »

17

- Étoile de Kleene (fermeture ou clôture par .)

- La fermeture de Kleene d'un langage  $A$  est le langage, noté  $A^*$ , défini par :  $A^* = \bigcup_{n \geq 0} A^n = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \dots$   
« mots formés par la concaténation d'un nbre qcq de mots de  $A$  »

- La fermeture positive de  $A$  est le langage, noté  $A^+$ , défini par :  
 $A^+ = \bigcup_{n \geq 1} A^n = A^1 \cup A^2 \cup A^3 \cup \dots$   
« mots formés par la concaténation de 1 ou plusieurs mots de  $A$  »

Propriété :  $A^+ = A.A^* = A^*.A$

$$A^* = A^+ \cup \{\varepsilon\}$$

18

## Langage ou problème ?

- Un problème de décision : auquel on répond par *oui / non*  
Ex 1 : soit  $x$  un nombre décimal, décider s'il est premier  
Ex 2 : soit un prg écrit en C, est-il syntaxiquement correct ?
- On peut modéliser ces problèmes par la notion de langage  
L = ensemble de données pour lesquelles la réponse est *oui*  
Ex 1 :  $L_p$  = ens. des nbres premiers (notation décimale)  
Ex 2 :  $L_c$  = ens. des prgs C syntaxiquement corrects

19

## Ce que la théorie des langages permet :

- Déterminer si une chaîne donnée appartient à un langage  
=> reconnaissance d'un langage
- Définir exactement quelles chaînes constituent un langage  
=> spécification d'un langage
- Différents modèles permettent de formaliser ces pb
- Ces modèles n'ont pas tous la même « puissance »
- On classe les langages selon le type de modèles qui permettent de les formaliser

20

## Classification de Chomsky

| Classes de langages       | Types de machines  | Types de grammaires        |
|---------------------------|--------------------|----------------------------|
| Réguliers                 | Automates finis    | Type 3 : régulières        |
| Non contextuels           | Automates à pile   | Type 2 : non contextuelles |
| Contextuels               |                    | Type 1 : contextuelles     |
| Récursivement énumérables | Machines de Turing | Type 0 : sans restriction  |

21

## Objectifs du cours

- Théorique : étude des langages réguliers et des langages non contextuels
- Applications à la compilation :
  - Les automates finis et les expressions régulières sont à la base des outils utilisés pour l'analyse lexicale
  - Les grammaires non contextuelles sont à la base des outils utilisés pour l'analyse syntaxique
- Notions de calculabilité et de décidabilité

22

# Expressions régulières et Automates finis

23

## 2 modèles pour les langages réguliers

- Expressions régulières (ER) : manière « algébrique » de décrire un langage régulier  
=> notation simple et précise
- Automates finis (AF) : manière quasi opérationnelle de décrire un langage régulier  
=> peut facilement être implémenté

Dans de nombreux systèmes de recherche de motifs

- les ER servent de langage d'interface avec l'utilisateur
- les AF sont à la base de l'implémentation

24

## Expressions régulières (ER) et langages

- Les ER sur un alphabet  $\Sigma$  et les langages correspondants sont définis récursivement par :

base :

- $\emptyset$  est une ER qui représente le langage  $\emptyset$
- $\epsilon$  est une ER qui représente le langage  $\{\epsilon\}$
- $a$  est une ER (pour tout  $a \in \Sigma$ ) qui représente  $\{a\}$

récur : si  $r$  et  $s$  sont des ER qui représentent les langages  $R$  et  $S$ , alors

- $r \mid s$  (ou  $r + s$ ) qui représente le langage  $R \cup S$
  - $r.s$  (ou  $rs$ ) qui représente le langage  $R.S$
  - $r^*$  qui représente le langage  $R^*$
- sont des ER

25

- Notation : on note  $r^+$  pour  $r.r^*$  (ou  $r^*.r$ )
- Priorité des opérateurs (par ordre décroissant) :  
\* puis . puis |
- Deux ER  $r$  et  $s$  sont équivalentes, noté  $r \equiv s$  ou  $r = s$ , si elles représentent le même langage

Rem : de nombreuses ER peuvent représenter le même langage

Ex :

- $\emptyset^* = \epsilon$
- $a^* = a^+ \mid \epsilon$
- $(a \mid \epsilon)^* = a^*$
- $(a \mid b)^* = (a^*b^*)^*$

26

## Automates finis

- Un modèle pour la reconnaissance d'un langage
- Automate fini pour un langage  $L$  : « machine » qui permet de répondre à la question  $\omega \in L$  ?

27

## Automates finis déterministes (AFD)

- Un AFD est un quintuplet  $(\Sigma, Q, q_0, F, \delta)$  où :
  - $\Sigma$  est un alphabet fini
  - $Q$  est un ensemble fini, non vide, d'états
  - $q_0 \in Q$  est l'état de départ (initial)
  - $F \subseteq Q$  est l'ensemble des états acceptants (ou finals)
  - $\delta$  est une fonction de transition de  $Q \times \Sigma$  dans  $Q$
- $\delta(p, a) = q$  est parfois noté  $p \xrightarrow{a} q$  et signifie :  
« si on est en l'état  $p$  et que l'on lit le symbole  $a$  alors on passe en l'état  $q$  »

28

## Chaînes et langage acceptés par un AFD

Soit un AFD  $M = (\Sigma, Q, q_0, F, \delta)$

- $M$  accepte une chaîne  $\omega = a_1a_2\dots a_n$  si il y a un chemin dans le diagramme de transitions qui
  - début en l'état initial  $q_0$
  - est étiqueté par  $a_1a_2\dots a_n$
  - et se termine en un état acceptant de  $F$
- Le langage accepté par  $M$  est  
 $L(M) = \{\omega \mid \omega \text{ est accepté par } M\}$

29

## Automate complet

- Un AFD est dit complet si la fonction de transition  $\delta$  est totale :  $\delta(p, a)$  est partout définie (pour tous  $p$  et tous  $a$ )
- Un état puits  $P$  est un état qui n'est pas acceptant et tel que toutes les transitions issues de  $P$  mènent à  $P$
- On peut toujours compléter un AFD pour le rendre complet :
  - On ajoute un état puits  $P$
  - On fait en sorte que toutes les transitions non définies mènent à  $P$
- Cela ne change pas le langage accepté par l'AFD

30

### Formellement

- On définit une fonction  $\Delta$  qui étend la fonction de transition  $\delta$  aux chaînes
- $\Delta(q, \omega) = q'$  signifie :  
« si on est en l'état  $q$  et que l'on lit la chaîne  $\omega$  alors on arrive en l'état  $q'$  »
- Définition par récurrence sur la longueur de  $\omega$   
 $\Delta : Q \times \Sigma^* \rightarrow Q$   
 base :  $\Delta(q, \varepsilon) = q$   
 $\Delta(q, a) = \delta(q, a)$  si  $a \in \Sigma$   
 récur :  $\Delta(q, \omega a) = \delta(\Delta(q, \omega), a)$  si  $\omega \in \Sigma^*$  et  $a \in \Sigma$

31

Soit un AFD  $M = (\Sigma, Q, q_0, F, \delta)$ , on a alors :

- Une chaîne  $\omega$  est acceptée par  $M$  ssi  $\Delta(q_0, \omega) \in F$   
« en partant de l'état initial  $q_0$  et en lisant  $\omega$ , on arrive à un état acceptant »
- Le langage accepté par  $M$  est  $L(M) = \{\omega \mid \Delta(q_0, \omega) \in F\}$   
« l'ensemble de toutes les chaînes qui, partant de  $q_0$ , mènent à un état acceptant »
- Les langages pour lesquels il existe un AFD sont appelés les langages réguliers

32

### Propriété de clôture : langage complémentaire

- Soit  $L$  un langage reconnu par un AFD, alors il existe aussi un AFD qui reconnaît le langage complémentaire  $\bar{L}$

Construction :

Si  $M_1 = (\Sigma, Q, q_0, F, \delta)$  est un AFD complet qui reconnaît  $L$ ,

alors  $M_2 = (\Sigma, Q, q_0, Q \setminus F, \delta)$  est un AFD qui reconnaît  $\bar{L}$

33

### Exemple 1

Un homme (H), un loup (L), une bique (B) et un chou (C) sont sur la rive gauche d'une rivière.

Il y a une barque pouvant transporter l'homme et l'un *seulement* des 3 autres.

- But : faire traverser la rivière à tout le monde
- Contraintes :
  - si le loup et la bique sont ensemble sans surveillance, le loup mange la bique
  - De même pour la bique et le chou

Est-il possible de faire traverser la rivière à tout le monde sans perte ?  
Si oui, comment ?

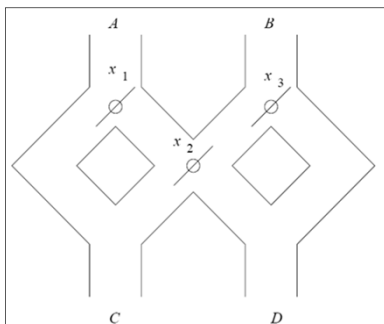
Modéliser à l'aide d'un AFD les situations et transitions possibles

⇒ États : situations (qui est sur chaque rive)

⇒ Transitions : passages possibles d'une situation à une autre

34

### Exemple 2 : jeu de bille



35

### Automates finis non déterministes (AFN)

- Un AFN est un quintuplet  $(\Sigma, Q, q_0, F, \delta)$  où :
  - $\Sigma$  est un alphabet fini
  - $Q$  est un ensemble fini, non vide, d'états
  - $q_0 \in Q$  est l'état de départ (initial)
  - $F \subseteq Q$  est l'ensemble des états acceptants (ou finals)
  - $\delta$  est une fonction de transition de  $Q \times \Sigma$  dans  $2^Q$

$\delta(p, a) = \{q_1, q_2, \dots, q_n\}$  signifie :

« si on est en l'état  $p$  et que l'on lit le symbole  $a$  alors on peut aller en  $q_1$  ou en  $q_2$  ou... en  $q_n$  »

- Rem : un AFD est un cas particulier d'AFN où  $\delta(p, a)$  est réduit à un seul état

36

## Chaînes et langage acceptés par un AFN

- une chaîne  $\omega = a_1 a_2 \dots a_n$  est acceptée par un AFN s'il existe un chemin dans le diagramme de transitions qui
  - débute en l'état initial  $q_0$
  - est étiqueté par  $a_1 a_2 \dots a_n$
  - se termine en un état acceptant de  $F$

37

## Remarques

- Dans un AFN, il peut exister 0, 1 ou plusieurs chemins qui débutent en  $q_0$  et sont étiquetés par  $\omega$ 
  - Pour déterminer qu'une chaîne est acceptée, il faut trouver un chemin qui mène à un état acceptant
  - Pour déterminer qu'une chaîne est refusée, il faut essayer tous les chemins possibles pour s'assurer qu'aucun ne mène à un état acceptant
- En pratique, on peut essayer tous les chemins « en même temps » en considérant qu'on se trouve dans plusieurs états simultanément

38

## Formellement

- On peut définir  $\Delta$  qui étend la fonction  $\delta$  aux chaînes
- $\Delta(q, \omega) = \{q_1, q_2, \dots, q_n\}$  signifie :  
« si on est en l'état  $q$  et que l'on lit la chaîne  $\omega$  alors on arrive en  $q_1$  ou en  $q_2$  ou... en  $q_n$  »

Soit un AFN  $M = (\Sigma, Q, q_0, F, \delta)$ , on a alors :

- Une chaîne  $\omega$  est acceptée par  $M$  ssi
 
$$\Delta(q_0, \omega) \cap F \neq \emptyset$$

« en partant de  $q_0$ , et en lisant  $\omega$ ,  
il y a au moins un état où on peut arriver qui est acceptant »
- Le langage accepté par  $M$  est
 
$$L(M) = \{\omega \mid \Delta(q_0, \omega) \cap F \neq \emptyset\}$$

39

## Propriété de clôture : intersection

- Soit  $L_1$  et  $L_2$  deux langages reconnus par des AFN (resp. AFD), alors il existe aussi un AFN (resp. AFD) qui reconnaît le langage  $L_1 \cap L_2$

Construction :

Si  $M_1 = (\Sigma, Q_1, q_{0_1}, F_1, \delta_1)$  est un AFN qui reconnaît  $L_1$ ,  
et  $M_2 = (\Sigma, Q_2, q_{0_2}, F_2, \delta_2)$  un AFN qui reconnaît  $L_2$   
alors  $M = (\Sigma, Q_1 \times Q_2, (q_{0_1}, q_{0_2}), F_1 \times F_2, \delta)$   
avec  $\delta((q_1, q_2), a) = \delta_1(q_1, a) \times \delta_2(q_2, a)$   
est un AFN qui reconnaît  $L_1 \cap L_2$

40

## Équivalence entre automates

- Deux automates  $M$  et  $M'$  sont équivalents ssi ils reconnaissent le même langage  
ou encore, ssi  $L(M) = L(M')$
- On va montrer que, pour tout AFN, on peut construire un AFD équivalent

41

## Transformation d'un AFN en un AFD équivalent

Soit  $N = (\Sigma, Q, q_0, F, \delta)$  un AFN,  
on construit  $D = (\Sigma, Q_D, q_{0_D}, F_D, \delta_D)$  un AFD  
qui reconnaît le même langage

- L'AFD a le même alphabet que l'AFN
- Les états de l'AFD sont des ensembles d'états de l'AFN
- L'état initial de l'AFD est  $q_{0_D} = \{q_0\}$
- Transitions de l'AFD :  
si  $S = \{p_1, \dots, p_n\}$  est un état de l'AFD (qui regroupe  $n$  états de l'AFN)  
alors  $\delta_D(S, a) = \delta(p_1, a) \cup \dots \cup \delta(p_n, a)$   
= tous les états qu'on peut atteindre dans l'AFN en partant d'un état de  $S$  et en lisant  $a$
- Les états acceptants de l'AFD sont ceux qui contiennent au moins un état acceptant de l'AFN

42

## Équivalence entre AFD et AFN

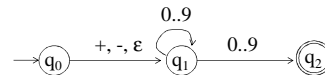
- Un AFD est un cas particulier d'AFN (où il y a au plus une transition pour chaque état et chaque symbole)  
=> les AFN sont au moins aussi puissants que les AFD
- Mais les AFN sont-ils plus puissants que les AFD ?  
Non, puisque tout AFN peut être transformé en un AFD équivalent

=> les AFN et les AFD ont exactement la même capacité de représentation  
ils reconnaissent les mêmes langages :  
les langages réguliers

43

## Automates avec $\epsilon$ -transitions (AFN- $\epsilon$ )

- $\epsilon$ -transition : transition étiquetée par  $\epsilon$  :  $p \xrightarrow{\epsilon} q$   
« on peut passer de l'état  $p$  à l'état  $q$  sans rien lire »
- Exemple : les entiers signés ou non



44

- Un AFN- $\epsilon$  est un quintuplet  $(\Sigma, Q, q_0, F, \delta)$  où  $\delta$  est définie de  $Q \times \Sigma \cup \{\epsilon\}$  dans  $2^Q$

Rem : Un AFN- $\epsilon$  est une généralisation d'un AFN où on autorise des transitions « vides »

- une chaîne  $\omega = a_1 a_2 \dots a_n$  est acceptée par un AF s'il existe un chemin dans le diagramme de transitions qui
  - début en l'état initial  $q_0$
  - est étiqueté par  $a_1 a_2 \dots a_n$
  - se termine en un état acceptant de  $F$

Rem : c'est la même définition que pour un AFN, mais ici le chemin peut inclure des arcs étiquetés  $\epsilon$ , même si  $\epsilon$  n'apparaît pas explicitement dans  $\omega$

45

## Équivalence entre automates finis

- Un AFN est un cas particulier d'un AFN- $\epsilon$  (où il n'y a pas d'  $\epsilon$ -transitions)  
=> les AFN- $\epsilon$  sont au moins aussi puissants que les AFN
- Mais les AFN- $\epsilon$  sont-ils plus puissants que les AFN ?  
Non, car tout AFN- $\epsilon$  peut être transformé en un AFD équivalent (admis)

=> les AFN- $\epsilon$ , les AFN et les AFD ont exactement la même capacité de représentation  
ils reconnaissent les mêmes langages :  
les langages réguliers

46

## Propriété de clôture : langage « renversé »

### Définitions

- si  $u = a_1 a_2 \dots a_n$  alors  $u^R = a_n \dots a_2 a_1$
- $L^R = \{u^R \mid u \in L\}$

### Propriété

- Soit  $L$  un langage reconnu par un AFD, alors il existe aussi un AF qui reconnaît le langage renversé  $L^R$

### Construction :

Si  $M_1 = (\Sigma, Q, q_0, F, \delta)$  est un AFD qui reconnaît  $L$ ,

alors  $M_2 = (\Sigma, Q \cup \{q_R\}, q_R, \{q_0\}, \delta_R)$

avec  $\delta_R(q, a) = \{p \mid \delta(p, a) = q\}$

et  $\delta_R(q_R, \epsilon) = F$

est un AFN- $\epsilon$  qui reconnaît  $L^R$

47

## Transformation d'une ER en un AFN- $\epsilon$

### Intérêt

- Automates : modèle quasi opérationnel pour reconnaître les mots d'un langage
  - AFD : efficace à exécuter
  - AFN : + facile à concevoir (et on peut transformer en AFD)
- Expressions régulières :
  - Encore + facile à concevoir que AFN
  - Notation très pratique
  - Mais ce n'est pas opérationnel

=> l'utilisateur conçoit des ER (facile, pratique)

La machine transforme l'ER en AFD

C'est le principe de fonctionnement des systèmes de recherche de motifs (grep, lex, ...)




48



Soit  $e$  une ER, on construit un AFN- $\epsilon$   $M$  qui reconnaît le même langage :  $L(M) = L(e)$

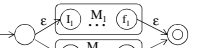
par récurrence sur le nombre d'opérateurs de  $e$

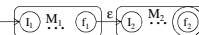
**base :**

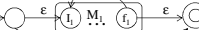
- si  $e = \emptyset$ ,  $M$  :  reconnaît  $\emptyset$
- si  $e = \epsilon$ ,  $M$  :  reconnaît  $\{\epsilon\}$
- si  $e = a$ ,  $M$  :  reconnaît  $\{a\}$

**récur :** si  $e_1$  et  $e_2$  ont au plus  $n$  opérateurs, alors on suppose qu'il existe  $M_1$  et  $M_2$  t.q.  $L(M_1) = L(e_1)$  et  $L(M_2) = L(e_2)$ .

Soit  $e$  ayant  $n+1$  opérateurs

- si  $e = e_1 | e_2$ ,  $M$  :  reconnaît  $L(e_1) \cup L(e_2)$

- si  $e = e_1 . e_2$ ,  $M$  :  reconnaît  $L(e_1) . L(e_2)$

- si  $e = e_1^*$ ,  $M$  :  reconnaît  $L(e_1)^*$

49

## Équivalence entre AF et ER

- On sait que les AFD, AFN et AFN- $\epsilon$  ont tous la même expressivité : les langages réguliers

Qu'en est-il de l'expressivité des ER ?

- On a vu que toute ER peut être transformée en AF  
=> les ER ne sont pas plus puissantes que les AF
- Mais sont-elles aussi puissantes que les AF ?  
=> Oui, car on peut montrer que, pour tout AF, il existe une ER qui représente le même langage

=> les ER et les AF ont exactement la même capacité de représentation

ils reconnaissent les mêmes langages :

les langages réguliers

50

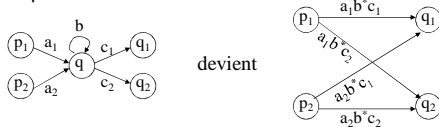
## Construction d'une ER correspondant à un AFD par élimination d'états

Étant donné un AFD,

on va éliminer un à un les états  $q$  qui ne sont pas l'état initial ni un état final

en contrepartie, on ajoute des arcs entre prédécesseurs et successeurs de  $q$ , étiquetés par des ER

Exemple

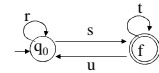


51

## méthode

- Pour chaque état final  $f$ , éliminer tous les états sauf  $q_0$  et  $f$

- Si  $q_0 \neq f$  alors on obtient :



et l'ER correspondante est (p.ex.) :  $r^* s (t | u r^* s)^*$

Si  $q_0 = f$  alors on obtient :



et l'ER correspondante est :  $r^*$

- L'ensemble des chaînes acceptées par l'AF est l'union des chaînes acceptées par chaque état final

52

# Propriétés des langages réguliers

53

- On a vu 4 caractérisations des langages réguliers
  - ER
  - AFD, AFN, AFN- $\epsilon$

=> Pour montrer qu'un langage est régulier,

il « suffit » de trouver une ER ou un AF qui le décrit

On peut utiliser cette démarche pour montrer des propriétés des langages réguliers

=> Pour montrer qu'un langage n'est pas régulier,

il faudrait être sûr qu'il n'existe pas d'ER ni d'AF

On va voir, via le lemme de l'étoile, une façon de faire

54

### Lemme de l'étoile (pumping lemma)

But : montrer qu'un langage L n'est pas régulier

Exemple :

Supposons que  $L = \{a^n b^n \mid n \geq 0\}$  est régulier

- Alors il existe un AFD à N états qui reconnaît L
- Si on lit  $a^N$ , alors on passe au moins 2 fois par le même état de l'automate

Autrement dit, il y a une boucle de lg k (p. ex) sur les a

- Donc, si  $a^N b^N$  est accepté par l'AFD,  $a^{N+k} b^N$  l'est aussi, pourtant, il n'appartient pas à L

Donc L'AFD ne reconnaît pas L, et L n'est pas régulier

55

### Lemme de l'étoile (informel)

Si L est un langage régulier, alors :

- Il existe un AFD à N états qui reconnaît L
- Et tous les mots de L de longueur  $\geq N$  sont tels que
  - il existe une boucle dans les N premiers symboles
  - et donc, si on passe 0 ou plusieurs fois dans la boucle, le mot est toujours accepté par l'AFD (et donc  $\in L$ )

- Utilisation :

- pour montrer qu'un langage n'est pas régulier
- en utilisant une démarche par l'absurde (voir ex. précédent)

56

### Quelques propriétés de clôture

- Soient A et B, deux langages réguliers, alors

- $A \cup B$
- $A.B$
- $A^*$
- $\bar{A}$
- $A \cap B$
- $A|B$
- $A^R = \{\omega^R \mid \omega \in A\}$

sont des langages réguliers

- Tout langage fini est régulier

57

### Utilisation 1 : montrer qu'un langage est régulier

Exemples

- Soit p un entier naturel,  
 $L_p = \{a^n b^n \mid n \leq p\}$
- $L_2 =$  mots sur  $\{a, b\}$  qui ne contiennent pas la sous-chaine aab
- $L_3 =$  mots w sur  $\{a, b\}$  t.q.  $|w|_a$  impair et  $|w|_b$  pair

58

### Utilisation 2 : montrer qu'un langage n'est pas régulier

On utilise le fait déjà démontré (lemme de la pompe)  
que  $\{a^n b^n \mid n \geq 0\}$  n'est pas régulier

Exemples

- $L_1 = \{0^n 1^m 2^n \mid n, m \geq 0\}$
- $L_2 =$  les chaînes de parenthèses bien équilibrées  
par exemple,  $((()()))$  ou  $()(())$
- $L_3 = \{0^n 1^n \mid n \geq 3\}$

59