Chapitre 2
Analyse lexicale
Partie I

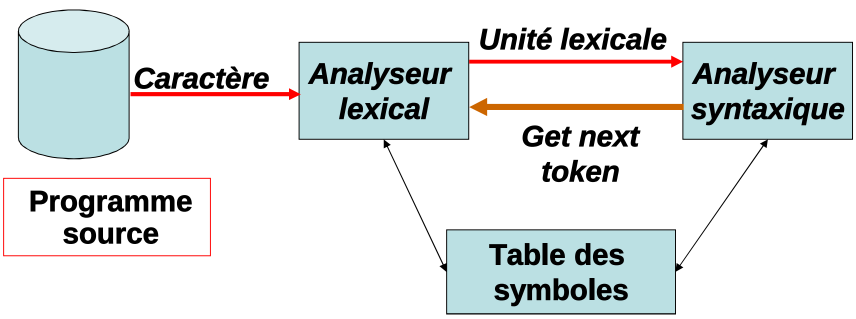
Sommaire

Rôle de l'analyseur lexical

- L'analyse lexicale constitue la première étape d'un compilateur.
- Elle consiste à lire des caractères d'entrée et de produire une suite de mots que l'on appelle des unités lexicales.
- Elle permet également la détection et l'élimination des caractères superflus (commentaires, tabulations, fin de ligne, ...).
- Elle gère aussi les numéros de lignes dans le programme source pour pouvoir associer à chaque erreur son numéro de ligne.

Interaction avec les autres modules

- Interaction entre un analyseur lexical et un analyseur syntaxique.



Terminologie

Trois termes majeurs sont utilisés en analyse lexicale :

- Unité lexicale : un couple constitué d'un nom d'unité lexicale et d'une valeur d'attribut optionnelle.

Exemple : Mot clé, identificateur, opérateur, ...

Les noms d'unité lexicale sont les symboles d'entrée de l'analyseur syntaxique.

- Motif : est une description de la forme que les lexèmes d'une unité lexicale peuvent prendre. Il peut être simple ou complexe.
- Lexème : est une séquence de caractères dans le programme source qui est reconnue par le motif d'une unité lexicale.

Les unités lexicales

- Exemples d'unités lexicales :
  - Mots clés : ce sont des mots réservés au langage de programmation.
    - Exemples (langage C) : if, else, while, do, for, main, char, int, float, double, sizeof, ...etc.
  - Identificateurs : un identificateur est un nom propre donné par le programmeur pour désigner une entité de son programme (une variable, une fonction ou une étiquette). En C, un identificateur est une suite de caractères qui commence par une lettre ([a...z] ou [A...Z] ou « _ ») et ne contient que des lettres et/ou des chiffres [0...9]).

Les unités lexicales

- Exemples d'unités lexicales (suite) :
  - Symboles spéciaux ou délimiteurs : ce sont des séparateurs d'unités lexicales.
    - Exemples (langage C):
      - point virgule « ; »
      - caractère blanc ou espace « »
      - tabulation
      - saut de ligne
      - accolades « { } »
      - guillemets « " »
      - apostrophe « ' »

Les unités lexicales

- Exemples d'unités lexicales (suite) :
  - Opérateurs : des symboles pour des opérations de base offertes par le langage de programmation.
    - Exemples (opérateurs du langage C) : +, -, *, /, %, >, >=, ==, <=, <, !=, &, |, ^, ~, >>, <<, &&, ||, !, ?:, =, +=, -=, *=, /=, %=, &=, |=, ^=, ~=, >>=, <<=, &&=, ||=, [ ], ( ), ., ->, & (adresse), sizeof.
  - Nombres : suite de chiffres avec éventuellement un signe (+ ou -), un point décimal « . ».
  - Commentaires : chaînes de caractères délimitées entre deux symboles spéciaux. Un commentaire est un texte à ignorer.

Attributs des unités lexicales

- L'analyseur lexical regroupe les informations sur les unités lexicales dans des attributs qui leur sont associés.
- Les unités lexicales influent sur les décisions de l'analyseur syntaxique.
- Les attributs influent sur la traduction des unités lexicales.
- En général, une unité lexicale a un seul attribut implanté comme un <u>pointeur</u> <u>vers l'entrée</u> de la <u>table des symboles</u> dans laquelle l'information sur l'unité lexicale est stockée.

Attributs des unités lexicales

- Comme exemples d'attributs, considérons l'instruction du langage Fortran : « E = M * C ** 2 ». Nous avons les unités lexicales et les attributs suivants :
  - <ID, pointeur vers l'entrée de la TS associée à E>
  - <OP_AFFECT>
  - <ID, pointeur vers l'entrée de la TS associée à M>
  - <OP_MULT>
  - <ID, pointeur vers l'entrée de la TS associée à C>
  - <OP_PUISS>
  - <NUMBER, valeur entière 2>

Attributs des unités lexicales

- Remarque :
  - Pour certaines unités lexicales, aucune valeur d'attribut n'est nécessaire : le premier composant suffit pour identifier le lexème en question.
- Exemples :
  - Pour ce qui est des symboles comme >, >=, =, <=, < et <> (langage Pascal), l'analyseur syntaxique a juste besoin de savoir que cela correspond à l'unité lexicale OP_REL (opérateur relationnel).
    - C'est seulement lors de la génération de code que l'on aura besoin de distinguer < de >= (par exemple).

Attributs des unités lexicales

- Exemples (suite) :
  - Pour ce qui est des identificateurs, l'analyseur syntaxique a juste besoin de savoir que c'est l'unité lexicale ID.
    - Mais le générateur de code, lui, aura besoin de l'adresse mémoire de la variable correspondant à cet identificateur.
    - L'analyseur sémantique aura aussi besoin du type de la variable pour vérifier que les expressions sont sémantiquement correctes.

Erreurs lexicales

- Peu d'erreurs sont détectables au seul niveau lexical, car un analyseur lexical a une vision très locale du programme source (il ne voit que les unités lexicales).
- Les erreurs se produisent lorsque l'analyseur est confronté à une suite de caractères qui ne correspond à aucun des motifs d'unités lexicales qu'il a à sa disposition : on dit qu'il y a une erreur lexicale.
- Par exemple, dans un programme C, si l'analyseur rencontre la chaîne de caractères « esle », s'agit t-il du mot clé « else » mal orthographié ou d'un simple identificateur ?
  - Dans ce cas précis, comme « esle » correspond au motif de l'unité lexicale ID, l'analyseur lexical ne détectera pas d'erreur, et transmettra à l'analyseur syntaxique qu'il a reconnu un ID.
  - S'il s'agit du mot clé mal orthographié, c'est l'analyseur syntaxique qui détectera alors une erreur.

Alphabet, mots et langage

- Les lexèmes sont avant tout des chaînes de caractères, c'est-à-dire des mots construits à partir de l'alphabet du langage de programmation.
- Les unités lexicales sont les noms des classes des lexèmes.
- Chaque unité lexicale est définie par une règle (un motif).
- Cette règle est utilisée par l'analyseur lexical pour faire correspondre un lexème à son unité lexicale : c'est le processus de reconnaissance d'unités lexicales.

Alphabet

- On appelle alphabet Σ un ensemble fini non vide. Les éléments d'un alphabet sont appelés lettres (ou symboles)
- Exemple d'alphabets
  - $\Sigma = \{a, b, .., z\}$
  - $\Sigma = \{\alpha, \beta, \delta, .., \varphi, +, *, /, =\}$
  - Σ = {if, then, else, while, begin, end}

Mots

- Une suite de symboles, appartenant à un alphabet Σ, mis bout à bout est appelé un mot (ou une chaîne) sur Σ.

  Par exemple : 01001100, est un mot construit sur l'alphabet $B=\{0,1\}$.
- Le nombre de symboles entrant dans la composition d'un mot α est appelé la longueur de α, notée $|\alpha|$.

  Par exemple : La longueur du mot 01001100 est $|01001100| = 8$.
- On distingue un mot particulier, dont la longueur vaut zéro, que l'on appellera le mot vide. Ce mot est conventionnellement représenté par le symbole ε.

Langages formels

Alphabet p	Mots de p
p = {a, b,.., z}	« abab », « surf »,« cascade », …
p = {~, , ¢, .., ¨, +, *, /, =}	« ~ = *2+¨ », « +~ ¢=* », …
p = {if, then, else, while, begin, end}	(if then else), (begin begin end)

© Compilation / M.MOURCHID, SMI/2021-2022 17

Langages formels

- La concaténation de deux mots ~ et notée ~ est le mot obtenu en juxtaposant les symboles de à la suite de ceux de ~.
- Exemple : Si ~ = abra et = cadabra alors ~ = abracadabra.
- Le mot m^n représente la concaténation de m avec lui même n fois :

$$\overbrace{mm\ldots m}^{n} = m^{n}$$

© Compilation / M.MOURCHID, SMI/2021-2022 18

Langages formels

- Étant donné trois mots ~, et ¥ définis sur un alphabet ġ, on dit que ~ est un préfixe du mot ~ , un suffixe du mot ~ et une sous-chaîne de ~ ¥. £ est un préfixe, un suffixe et une sous-chaîne de tout mot. Si ~ ÷ et ~ est un préfixe (ou suffixe) de , alors on dit que ~ est un préfixe (ou suffixe) propre de .
- L'ensemble de tous les mots que l'on peut construire sur un alphabet ġ, y compris £, est noté ġ*.

Langages formels

- Un langage sur un alphabet ġ est un ensemble de mots construits sur ġ. Tout langage défini sur ġ est donc une partie de ġ*.
- On distingue deux langages particuliers qui sont définis indépendamment de tout alphabet. Il s'agit du langage vide (noté Ą) et du langage composé uniquement du mot vide ({£}). ġ peut être vu comme le langage composé des symboles de ġ.

Exemples de Langages

- Exemple de langages sur p = {a, b,.., z}
  - L_0 = {a}, L_1 = {aa,ab},
  - L_2 = {waw | w Č p*},
  - L_3 = {w Č p* | $|w|_a$ á 10 }

Où $|.|_{x \text{ Č } p}$: p* ì IN   est la fonction qui calcule le nombre d'occurrence de la lettre x de p dans un mot de p*.

Opérations sur les langages formels

Les langages étant des ensembles, on peut leur appliquer les opérations définies sur ces derniers:

- L'union de deux langages L_1 et L_2, est le langage, noté L_1 Ć L_2 constitué des mots appartenant à L_1 ou à L_2.

  L_1 Ć L_2 = {xûx Č L_1 ou x Č L_2}
- L'intersection de L_1 et L_2, est le langage, noté L_1ąL_2 constitué des mots appartenant à L_1 et à L_2.

  L_1ą L_2 = {xûx Č L_1 et x Č L_2}
- La différence de L_1 et L_2 est le langage, noté L_1-L_2, constitué des mots appartenant à L_1 et n'appartenant pas à L_2.

  L_1 - L_2 = {xûx Č L_1 et x č L_2}

Opérations sur les langages formels

- La différence de ģ* et de L, noté $\overline{L}$ est appelé le complément du langage L.

$$\overline{L} = \left\{ x \in \Sigma^* \quad et \quad x \notin L \right\}$$

- On définit de plus l'opération de concaténation de deux langages : la concaténation de deux langages L_1 et L_2 est le langage noté L_1L_2 composé des mots xy tels que x Č L_1 et y Č L_2.

  L_1L_2 = {xyûx Č L_1 et y Č L_2}

- on note L^n la concaténation de L avec lui même n fois :

  L^n={x_1x_2 … x_n | x_i ČL ? 1áián }

Opérations sur les langages formels

- On définit enfin la fermeture de Kleene du langage L, notée L* de la façon suivante :

$$L^* = \bigcup_{k \geq 0} L^k$$

$$L^+ = L^* - \{ \varepsilon \}$$

Opérations sur les langages formels

Proposition

Les identités suivantes sont vraies pour tous les langages X, Y, Z :

- $X(YZ)=(XY)Z$
- X£= £ X=X
- X(YĆZ)=XY Ć XZ et (X Ć Y)Z=XZ Ć YZ
- XX* Ć £ =X*
- (X*)*=X*

Langages réguliers

- Le problème préliminaire qui se pose à propos des langages est:

  comment décrire un langage ?
- La question qui se pose ensuite est:

  étant donné une phrase (mot) est ce qu'elle appartient au langage ?

  Dans cette partie nous définissons les expressions régulières qui permettent de décrire un certain type de langages : les langages réguliers.

Langages réguliers

- Un langage régulier L sur un alphabet ģ est défini récursivement comme suit:
  - ¹£» est un langage régulier sur ģ
  - ?aČģ, ¹a» est un langage régulier sur ģ
  - si R est un langage régulier sur ģ alors R^k et R^* sont des langages réguliers sur ģ
  - si R_1 et R_2 sont des langages réguliers alors R_1ĆR_2 et R_1R_2 sont des langages réguliers sur ģ.

Expressions régulières (ER)

- Une expression régulière est une notation permettant de spécifier un modèle.
- Le modèle reconnait un ensemble de mots (un langage) sur un alphabet.
- L'analyseur lexical est chargé de reconnaître les unités lexicales du texte source.
- Chacun de ces lexèmes est un langage dans la plupart des cas, un langage régulier.

Expressions régulières (ER)

Exemple:

Soit Σ={a,b}

- **a | b** dénote l'ensemble {a,b}
- **(a | b)(a | b)** dénote {aa, ab, bb, ba}
- **a*** dénote l'ensemble des mots formés d'un nombre quelconque (éventuellement nul) de a: {ε, a, aa, aaa, aaa, …}
- **(a | b)*** = **(b | a)*** = **(a* | b*)*** = Σ* dénote l'ensemble des mots formés d'un nombre quelconque (éventuellement nul) de a ou de b
- **a | a*b** dénote l'ensemble contenant le mot a et tous les mots constitués d'un nombre quelconque (éventuellement nul) de a suivi de b.

Expressions régulières (ER)

- Soit p un alphabet, les expressions régulières et les langages qu'elles décrivent sont définis comme suit :
  - ?aČp, a est une expression régulière qui décrit le langage ¹a»,
  - £ (mot vide) est une expression régulière qui décrit le langage ¹£»
  - Ą (ensemble vide) est une expressions régulière qui décrit le langage vide.
  - Si ~ et sont des expressions régulières décrivant respectivement les langages A et B alors
    - (~ |) (ou ~ + est l'expression régulière décrivant le langage AĆB
    - (~) est l'expression régulière décrivant le langage AB,
    - ~* est l'expression régulière décrivant le langage A*

  Nulle autre expression n'est une expression régulière

Exemples d'expressions régulières

- Alphabétique = ('a' | 'b' | … | 'z' | 'A' | 'B' | … | 'Z')
- Numérique = (0 | …| 9)
- Opérateurs = (+ | - | / | * | = | <= | >= | < | >)
- Naturel = Numérique$^+$
- Entier = (+ | - | £) Naturel
- Identificateur = (Alphabétique(Alphabétique|Numérique)*)

Spécification des unités lexicales

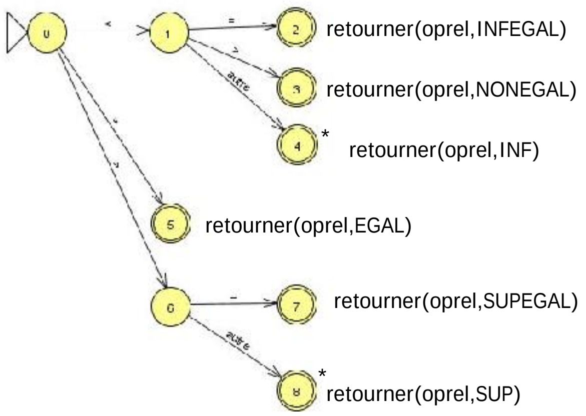
- Les motifs de lexèmes sont représentés par des expressions régulières.
- Les détails sur les expressions régulières se trouvent dans le cours Théorie des langages.
- Exemple : On décrit l'ensemble des identificateurs C par l'expression régulière :
  - lettre_= [A-Za-z_]
  - chiffre = [0-9]
  - id = lettre_(lettre_|chiffre)*

Reconnaissance des unités lexicales

- Grammaire pour les instructions de branchement :
  - instr → if expr then instr
    | if expr then instr else instr
    | ε
  - expr → terme oprel terme
    | terme
  - terme → id
    | nbre
- Motif de l'unité lexicale oprel
  - oprel = (< | > | <= | >= | = | <>)
- On commence par convertir les motifs en diagrammes de transition.
- Il peut y avoir plusieurs diagrammes de transitions, chacun spécifiant une unité lexicale.

Diagramme de transition

Diagramme de transition pour **oprel**



Architecture d'un analyseur lexical

- But : construire un analyseur lexical sur la base d'un diagramme de transition.
- Une variable etat sert à conserver le numéro de l'état courant.
- Une fonction car_suiv() récupère le caractère suivant dans l'entrée.
- Si le caractère suivant n'est pas reconnu, on appelle la fonction echec().
- Si l'état contient * alors il faut reculer le pointeur d'une position (reculer()).

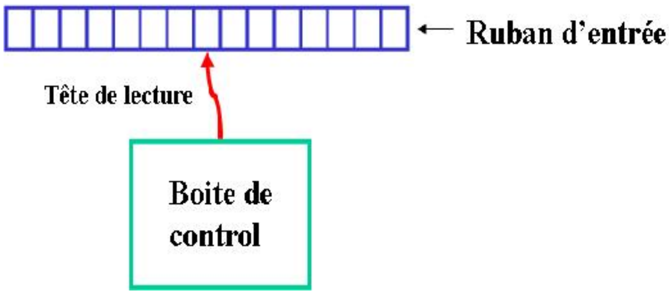
Implémentation de oprel

```
int lire_op_rel() {
   while(1) {
      switch(etat) {
         case 0  : c=car_suiv();
                  if(c=='<') etat=1;
                  else if(c=='=') etat=5;
                  else if(c=='>') etat=6;
                  else echec()
                  break;
         . . . . . . .
         case 8  : reculer();
                  return SUP; }
   }
}
```

Automates finis

Un automate «lit» un mot écrit sur son ruban d’entrée.

Il part d’un état initial et à chaque lettre lue, il change d’état. Si, à la fin du mot, il est dans un état final, on dit qu’il reconnaît le mot lu, ou qu’il l’accepte.



Automates finis déterministes

- Un automate d’états finis A est défini comme un 5-uplet A = <Q, p, ¢, q_0, F > où :
  - Q est l’ensemble des états,
  - p est un alphabet (un ensemble de symboles),
  - ¢ : Q ò p ì Q est la fonction de transition de l’automate,
  - q_0 est l’état initial de l’automate,
  - F ċ Q est l’ensemble des états finaux de l’automate.
- Les automates peuvent être représentés par des graphes où les états sont les noeuds du graphe et les arcs représentent la fonction de transition.

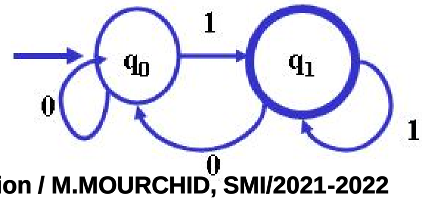
Exemple d'automate fini

Soit A = <Q, p, ¢, q_0, F> avec

- p = {0,1}
- S = {q_0, q_1}
- ¢ = {(q_0, 0, q_0), (q_0, 1, q_1), (q_1, 1, q_1), (q_1, 0, q_0) }
- q_0 est l'état initial
- F = {q_1}

A peut être représenté par le graphe suivant:

	0	1
q_0	q_0	q_1
q_1	q_0	q_1



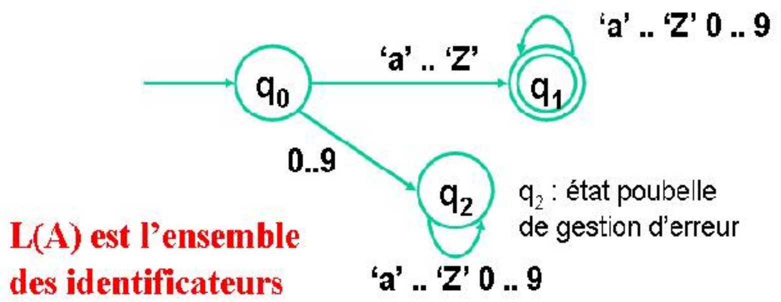
Langage accepté par un AFD

- On dit qu'un automate d'états finis A = <Q, p, ¢, q_0, F > accepte le mot x, composé des symboles $x_1x_2x_3 \ldots x_n$ ssi ¢(¢(… ¢(¢(q_0, x_1), x_2), x_3),…), x_{n-1}), x_n) Č F
- Intuitivement $x_1x_2x_3 \ldots x_n$ correspond aux étiquettes des arcs formant un chemin à travers les transitions de l'automate, commençant par q_0 et se terminant par q_k Č F.
- A chaque étape, x_i correspond à l'étiquette du i^{ème} arc du chemin.
- L'ensemble des mots pouvant être acceptés par un automate A est appelé le langage accepté par A, noté L(A).

Exemple: Langage accepté par un AFD

Soit A l'automate donné par:

- p = { 'a' , … , 'z' , 'A' , … , 'Z', 0 , …, 9 }
- Q = {q_0 , q_1 , q_2 } avec comme état initial q_0
- F = { q_2 }



Automates non-déterministes (AFN)

- Il est difficile d'écrire simplement les automates correspondant à une ER compliquée, l'automate non-déterministe simplifie cette tâche
- Un automate non-déterministe A = <Q, p, ¢, q_0, F > est un automate qui présente la particularité suivante :

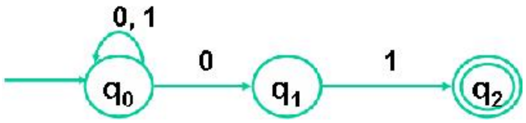
  Il possède une fonction de transition étendue

  ¢ : Q ò p Ć {£} ì o(Q)

  1. plusieurs arcs reconnaissant le même symbole peuvent « sortir » du même état
  2. Il peut y avoir des transitions, appelées (£-transitions), qui ne correspondent à aucune reconnaissance de caractères.

Exemple de AFN (sans ε-transitions)

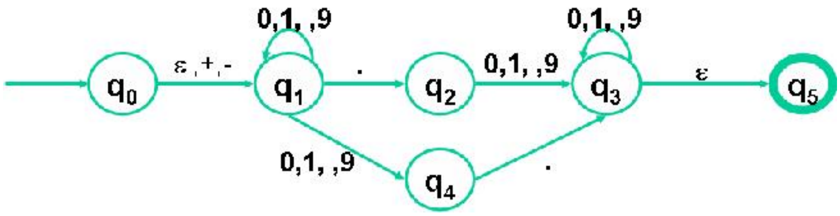
Soit l'automate donné par le graphe:



δ	Q	0	1
	q_0	$\{q_0,q_1\}$	$\{q_0\}$
	q_1	Ą	$\{q_2\}$
	q_2	Ą	Ą

Exemple de AFN (avec ε-transitions)

Soit l'automate donné par le graphe:



δ	Q	£	+,-	.	0,1, ,9
	q_0	$\{q_1\}$	$\{q_1\}$	Ą	Ą
	q_1	Ą	Ą	$\{q_2\}$	$\{q_1, q_4\}$
	q_2	Ą	Ą	Ą	$\{q_3\}$
	q_3	$\{q_5\}$	Ą	Ą	$\{q_3\}$
	q_4	Ą	Ą	$\{q_3\}$	Ą
	q_5	Ą	Ą	Ą	Ą

Transformer un AFN en un AFD

Deux cas sont possibles:

- 1er cas : l'automate non-déterministe est sans £-transition
- 2ème cas : l'automate non-déterministe contient des £-transitions

© Compilation / M.MOURCHID, SMI 45

Transformer un AFN (sans ε-transitions) en un AFD

Soit $A_N = < Q_N, p_N, \text{¢}_N, q_{N0}, F_N >$ un automate non-déterministe sans £-transitions, il existe un automate déterministe $A_D = < Q_D, p_D, \text{¢}_D, q_{D0}, F_D >$ tel que:

$L(A_N) = L(A_D)$

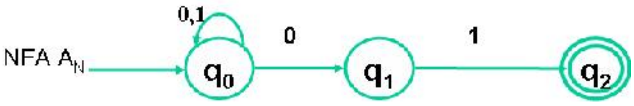
- Q_D ċ $o(Q_N)$

  tous les éléments de $o(Q_N)$ qui sont accessibles depuis l'état initial q_{D0}
- $p_D = p_N$
- $\text{¢}_D(q_D$ Č Q_D, a Č $p_D) =$ Ć$_{q_N \text{ Č } q_D}$ $\text{¢}_N(q_N, a)$
- $q_{D0} = \{q_{N0}\}$
- $F_D = \{q$ Č Q_D | $(q$ ą $F_N)$ ÷ Ą}

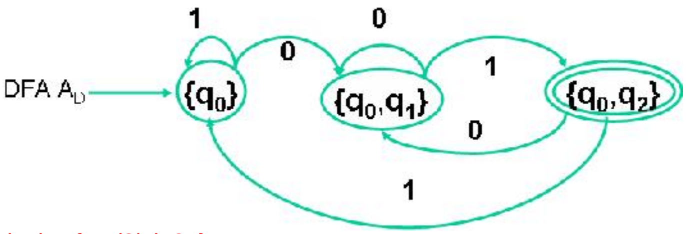
© Compilation / M.MOURCHID, SMI 46

Exemple : Transformer un AFN (sans ε-transitions) en un AFD

L'automate donné par le graphe suivant:

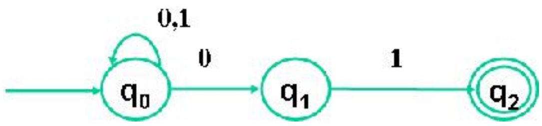


est équivalent à celui donné par le graphe suivant



$L(A_N) = L(A_D) = \{w=(0|1)^*01\}$

Exemple : Transformer un AFN (sans ε-transitions) en un AFD



δ_D

- **état accessible depuis q_{D0}**

ā (Q_N)	0	1
A	∅	∅
$q_{D0} = \{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
$\{q_1\}$	∅	$\{q_2\}$
$\{q_2\}$	∅	∅
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
$F_{D=}\{q_0,q_2\}$	$\{q_0,q_1\}$	$\{q_0\}$
$\{q_1,q_2\}$	∅	$\{q_2\}$
$\{q_0,q_1,q_2\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$

Transformer un AFN (avec ε-transitions) en un AFD

On définit la fonction £-fermeture : Q ì o(Q) comme suit:

1. q Č £-fermeture(q)
2. Si q_1Č ¢(q,£) alors q_1 Č £-fermeture(q)
3. Si q_1 Č £-fermeture(q) ė q_2 Č £-fermeture(q_1) alors q_2 Č £-fermeture(q)

On peut définir par extension

£-fermeture : o(Q) ì o(Q) telle que:

£-fermeture(Q' ċ Q) = Ć$_{q' Č Q'}$ £-fermeture(q')

£-fermeture(q) regroupe tous les états accessibles depuis q sur des chemins constitués seulement de £-transitions

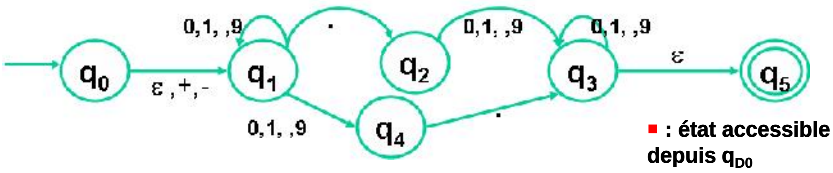
Transformer un AFN (avec ε-transitions) en un AFD

Soit A_E = <Q_E, p_E, ¢$_E$, q_{E0}, F_E > un automate non-déterministe avec £-transitions (£-NFA), il existe un automate déterministe (DFA) A_D=<Q_D, p_D, ¢$_D$, q_{D0}, F_D > tel que L(A_E) = L(A_D)

- Q_D ċ £-fermeture(o (Q_E))

  tous les éléments de £-fermeture(o(Q_E)) qui sont accessibles depuis l'état initial q_{D0}
- p_D = p_E
- ¢$_D$(q_D Č Q_D, a Č p_D) = £-fermeture(Ć$_{q_E \text{ Č } q_D}$ ¢$_E$(q_E, a))
- q_{D0} = {£-fermeture(q_{E0})}
- F_D = {q Č Q_D | (q ą F_E) ÷ Ą}

Exemple : Transformer un ε-AFN en AFD



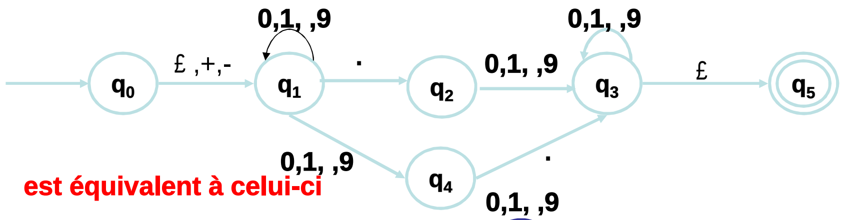
■ : état accessible depuis q_{D0}

δ_D

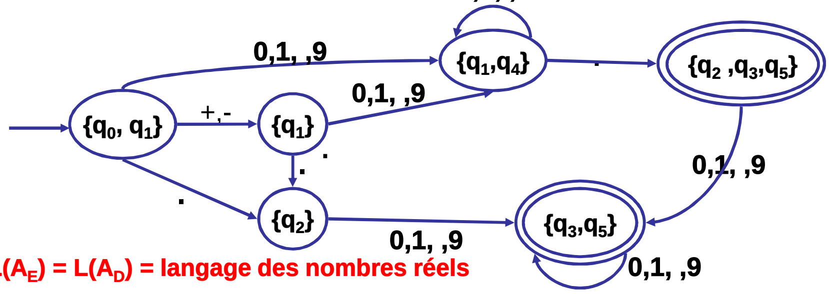
Q_E	£-fermeture	+,-	£-fermeture	.	£-fermeture	0,1, ,9	£-fermeture
$\{q_0\}$	$q_{D0}=\{q_0, q_1\}$	$\{q_1\}$	$\{q_1\}$	$\{q_2\}$	$\{q_2\}$	$\{q_1, q_4\}$	$\{q_1, q_4\}$
$\{q_1\}$	$\{q_1\}$	Ą	Ą	$\{q_2\}$	$\{q_2\}$	$\{q_1, q_4\}$	$\{q_1, q_4\}$
$\{q_2\}$	$\{q_2\}$	Ą	Ą	Ą	Ą	$\{q_3\}$	$\{q_3, q_5\}$
$\{q_3\}$	$\{q_3, q_5\}$	Ą	Ą	Ą	Ą	$\{q_3\}$	$\{q_3, q_5\}$
$\{q_4\}$	$\{q_4\}$	Ą	Ą	$\{q_3\}$	$\{q_3, q_5\}$	Ą	Ą
$\{q_5\}$	$\{q_5\}$	Ą	Ą	Ą	Ą	Ą	Ą
	$F_{D1}=\{s_1, q_4\}$	Ą	Ą	$\{q_2, q_3\}$	$\{q_2, q_3, q_5\}$	$\{q_1, q_4\}$	$\{q_1, q_4\}$
	$F_{D2}=\{q_2, q_3, q_5\}$	Ą	Ą	Ą	Ą	$\{q_3\}$	$\{q_3, q_5\}$

Exemple : Transformer un ε-AFN en AFD

L'automate suivant



est équivalent à celui-ci



$L(A_E) = L(A_D)$ = langage des nombres réels

Minimiser un AFD

- Minimiser un AFD A revient à regrouper ses états équivalents de telle sorte à réduire son nombre d'états
- Deux états p,q de A sont dits équivalents ssi:

  ? wČp* tq ¢(p,w) est un état final ę ¢(q,w) est un état final

  1. w = £, si p Č F et q č F alors p et q sont distinguables (non équivalents)
  2. w ≠ £, (¢(p,w)=r et ¢(q,w)=s) ė (r et s sont distinguables) alors p et q sont distinguables

Minimiser un DFA

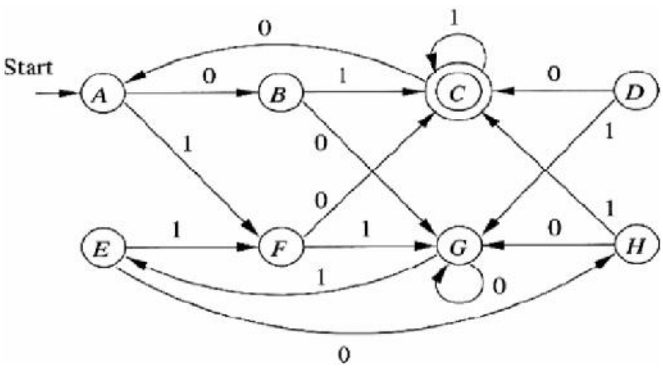
L'algorithme de minimisation construit une partition P de l'ensemble Q des états de l'automate déterministe, telle que : chaque état se trouvant dans le même sous ensemble a le même comportement sur les mêmes entrées.

L'algorithme est le suivant :

Minimiser un DFA

- ➔P ê ¹ F,Q-F » (où F est l'ensemble des états finaux)
  - Tant que (P change)
    - T ê Ą
    - Pour chaque ensemble pČP
      - T ê T Ć partition(p)
    - P ê T
  - Fin tant
- ➔Partition(p)
  - Pour chaque cČġ
    - Si c sépare p en ¹p$_1$, …, p$_k$»
    - Alors return (¹p$_1$, …, p$_k$»)
  - Fin pour
  - Return(p)

Exemple: Minimiser un AFD



- A ø E, car ¢(A,1)=¢(E,1)=F et ¢(A,0)=B; ¢(E,0)=H avec Bø H
- B ø H, car ¢(B,1)=¢(H,1)=C ė ¢(B,0)=¢(H,0)=G
- Ė(A ø B), car ¢(B,1)=C Č F ė ¢(A,1) č F
- etc.

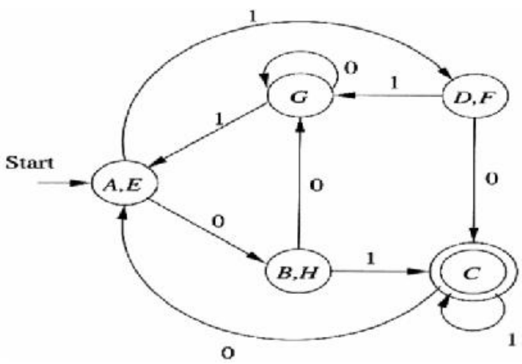
Exemple: Minimiser un DFA

	A	B	C	D	E	F	G
B	X						
C	X	X					
D	X	X	X				
E		X	X	X			
F	X	X	X		X		
G	X	X	X	X	X	X	
H	X		X	X	X	X	X

- [s_i,s_j] = ✗ signifie que Ė (s_i ø s_j) (i.e. s_i et s_j sont distinguables)

Exemple: Minimiser un AFD

L'automate minimal regroupant les états équivalents est

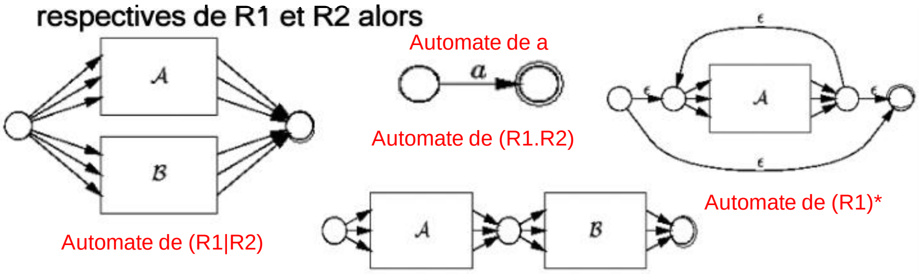


Equivalence des automates finis aux expressions régulières

Théorème

Un langage est régulier si et seulement s'il est généré par un automate fini.

- On suppose que les automates finis n'ont qu'un seul état final q_f avec ¢(q_f,a) = Ø pour tout aČģ. De même, on peut supposer que q_0 č ¢(q,a) pour tout qČQ et aČģ.
- Alors les automates finis reconnaissent les expressions régulières par les constructions suivantes : Si A et B sont des automates respectives de R1 et R2 alors

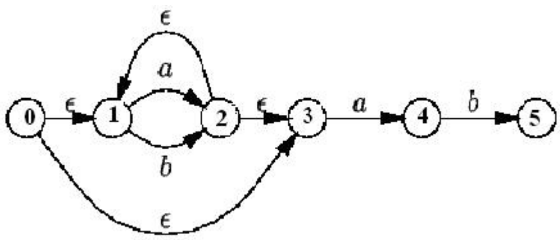


Equivalence des automates finis aux expressions régulières

D'après ce qui précède, toute expression régulière est reconnue par un automate.

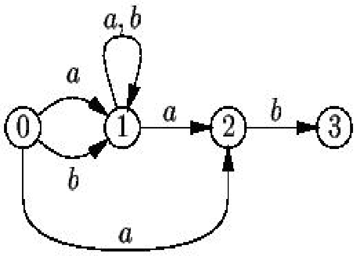
Exemple

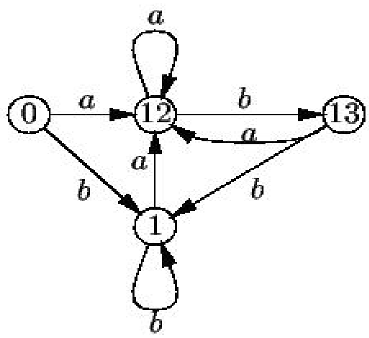
Soit l'expression régulière (a+b)*ab, l'automate associé est:



Equivalence des automates finis aux expressions régulières

L'automate ainsi obtenu est non déterministe et il possède beaucoup de £-transitions. En supprimant les £ -transitions et en déterminisant, on obtient:





Détermination du langage accepté par un automate fini

Réciproquement, pour obtenir le langage reconnu par un automate on utilise l'algorithme suivant :

Algorithme de Mac Naughton et Yamada

Principe : L'ensemble des transitions menant d'un état s à un état t est, pour un état q distinct, l'union de deux ensembles de chemins

- d'une part, les chemins menant de s *à* t sans passer par q;
- d'autre part, les chemins menant de s *à* t en passant par q.

Détermination du langage accepté par un automate fini

ALGORITHME :

Soient Aut = <ġ, Q, q_0, F, ¢>, s Č Q, t Č Q et P ċ Q (un sous-ensemble des états).

On note $W_{s,P,t}$ l'ensemble de tous les mots (chemins) uČġ* tels que u est l'étiquette d'un chemin menant de s à t et n'utilisant pas d'autres états que ceux de P comme intermédiaires.

Donc le langage reconnu par l'automate est :

L(Aut) = Ć$W_{s,Q,t}$ où s ČD et t Č F qui est l'union de tous les $W_{s,Q,t}$ possibles. D étant l'ensemble des états départ.

Détails

On démarre avec P = Q et on cherche, pour tous les couples (s,t) tels que s ČD et t Č F, le langage $W_{s,P,t}$. Ce langage est défini grâce à la formule générale suivante :

$W_{s,P,t} = W_{s,P',t}$ Ć $W_{s,P',q} . (W_{q,P',q})^* . W_{q,P',t}$

où P' est défini par P' = P \ {q} (q étant un état que l'on choisit arbitrairement dans P afin de pouvoir avancer.

Détermination du langage accepté par un automate fini

Méthode de Kleene

Soit A= <ġ, Q, q0, F, ¢> un automate fini non déterministe ne contenant pas d'£-transitions.

Pour tout état q , notons Xq l'ensemble des mots menant de l'état q à un état final. Chaque langage Xq satisfait l'équation linéaire

$$X_q \text{ Z } \underset{\text{où } q' \text{Z¢} (q,a)}{\text{ġ}} aX_{q'} \text{ ẊH £ } \textit{si } q \text{ Č } F \text{Ẓ}$$

Pour obtenir une expression régulière du langage, il suffit de résoudre le système des équations linéaires ci-dessus par élimination de variables : toutes les équations qu'on manipule peuvent être mises sous la forme Xq=KXq+L où K et L sont des expressions linéaires des variables Xq' , q'÷q ;

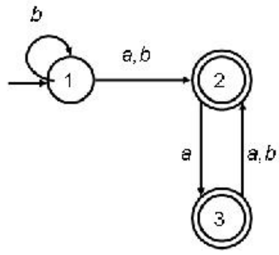
Détermination du langage accepté par un automate fini

K peut être le langage vide (si le membre droit de l'équation ne contient pas la variable Xq), mais il ne contient pas le mot vide (car l'automate ne possède pas d'Č-transitions) ; d'après le lemme d'Arden, l'équation est équivaut à Xq=K*L. En remplaçant Xq par K*L dans les autres équations, on obtient encore un système d'équations linaires, équivalent au système d'origine.

Détermination du langage accepté par un automate fini

Exemple

Soit L le langage reconnu par l'automate suivant :



Le système d'équations linéaires associé à cet automate est :

$$\begin{cases} X_1 = bX_1 + (a+b)X_2 & (1) \\ X_2 = aX_3 + \varepsilon & (2) \\ X_3 = (a+b)X_2 + \varepsilon & (3) \end{cases}$$

Détermination du langage accepté par un automate fini

On peut éliminer X_3 dans l'équation (2), en remplaçant X_3 par $(a+b)X_2+\pounds$: on obtient

$$X_2=a((a+b)X_2+\pounds)+\pounds=a(a+b)X_2+(a+\pounds)$$

Ce qui est équivaut (d'après le lemme d'Arden) à:

$$X_2=(a(a+b))^*(a+\pounds)$$

On peut maintenant éliminer X_2 dans l'équation (3) :

$$X_3=(a+b)(a(a+b))^*(a+\pounds)+\pounds$$

Ainsi que dans l'équation (1) :

$$X_1=bX_1+(a+b)(a(a+b))^*(a+\pounds)$$

D'où d'après le lemme d'Arden $X_1=b^*(a+b)(a(a+b))^*(a+\pounds)$

Le langage reconnu par l'automate est alors :

$$X_1=b^*(a+b)(a(a+b))^*(a+\pounds)$$

Automates finis déterministes

Un AFD est un cas particulier d'AFN :

- Il n'y a aucun arc étiqueté par ε.
- $\forall (s,a) \in S \times \Sigma$, il y a un seul arc sortant de s étiqueté par a.

Implémentation d'un AFD.

```
s=e0;
c=car_suiv();
tant que (c!=eof) {
   s=trans(s,c);
}
si (sČF)
   retourner “oui”
sinon
   retourner “non”
```

Lemme d’Arden

(Résolution d’équation linéaire)

Lemme d’Arden

Soient X, K et L des langages sur un alphabet A. On suppose que £čK. Alors

X=KX Ć L si et seulement si $X=k^*L$