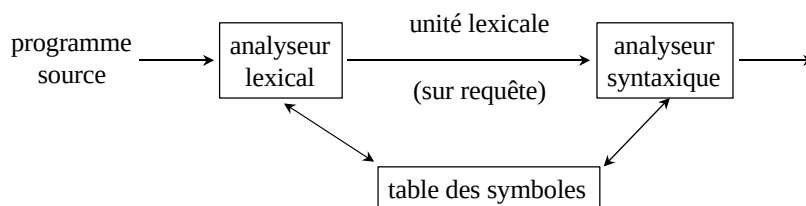


CH.2 L'ANALYSE LEXICALE

- 2.1 L'analyseur lexical
- 2.2 Le texte d'entrée
- 2.3 Les unités lexicales
- 2.4 Les diagrammes de transition
- 2.5 Les automates finis

2.1 L'analyseur lexical



Rôle :

- lire les caractères d'entrée ;
- réaliser un pré-traitement du programme source ;
- transmettre à l'analyseur syntaxique des unités lexicales ;
- initialiser la table des symboles ;
- garder un lien entre compilateur et utilisateur.

Intérêts de l'analyse lexicale :

- Conception modulaire plus simple du compilateur ;
- Simplification de l'écriture de l'analyseur syntaxique ;
- Techniques spécifiques d'entrée du texte ;
- Portabilité accrue (modifications de l'alphabet d'entrée) ;
- Existence de techniques générales d'analyse lexicale ;
- Problèmes voisins (traitement de texte, ...)

Lexèmes, unités lexicales, modèles, attributs :

- Lexème=chaîne de caractères ;
- Unité lexicale=type de lexèmes (pour la syntaxe) ;
- Modèle=règle décrivant quelles chaînes correspondent à un modèle donné ;
- Attribut=informations additionnelle (pour la sémantique).

Exemples (PASCAL) :

lexèmes	unité lexicale	Attribut
vitesse	identificateur	pointeur vers table des symboles
3.1416	nombre_flottant	représentation ANSI du nombre
:=	affectation	aucun
<	op_relation	code pour inférieur
=	op_relation	code pour égal
"bonjour"	littéral	tableau de caractères
begin	début_bloc	aucun

Facteurs modifiant la complexité de l'analyse lexicale :

- Le placement dans la ligne est-il important ?
- Quelle est la signification des blancs ?
- Les mots clés sont-ils réservés ?

Erreurs lexicales : peu (caractères interdits, ...)

2.2 Le texte d'entrée

- Nécessité d'optimiser la phase de lecture.
- Utilisation d'un tampon linéaire ou circulaire.
- Couple de pointeurs pour gérer le problème des "caractères de pré-vision"

Exemple :

```
etiq:vrai:=valeur=.1;
  ↑   ↑↑   ↑↑   ↑↑↑↑
  1   23   4 5   67 8
```

Fonctions appropriées dans certains langages :

get et unget en C

2.3 Les unités lexicales

Alphabet : binaire, ASCII, ... ;

Mot : chaîne de caractères ; mot vide ϵ

Langage : ensemble de mots ;

exemples : $\emptyset$, $\{\epsilon\}$, $\{\text{begin}, \text{end}\}$

Opérations sur les langages :

Union ensembliste : $L \cup M$;

Concaténation : LM ;

Fermeture positive : $L^+ = \bigcup_{i=1}^{\infty} L^i$

Fermeture de Kleene : $L^* = \bigcup_{i=0}^{\infty} L^i$

N.B. Ni la complémentation, ni l'intersection.

Expressions régulières :

- i) L'expression régulière ϵ représente $\{\epsilon\}$;
- ii) Si a est une lettre, alors c'est une expression régulière qui représente $\{a\}$;
- iii) Si r et s sont des expressions régulières qui représentent $L(r)$ et $L(s)$, alors :
 - $r \mid s$ représente $L(r) \cup L(s)$ (ou $r + s$)
 - rs représente $L(r)L(s)$
 - r^* représente $L(r)^*$.

Langage régulier

= langage représenté par expression régulière.

Exemples :

$a \mid b$, $(a \mid b)^*abba$, $a^*b(a^*b)^*$ et $(a \mid b)^*b$
 $(1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)^*$

Définition régulière :

permet de donner des noms à des expressions régulières.

Exemples :

lettre $\rightarrow A \mid B \mid \dots \mid Z \mid a \mid b \mid \dots \mid z$

chiffre $\rightarrow 0 \mid 1 \mid \dots \mid 9$

id $\rightarrow \text{lettre} (\text{lettre} \mid \text{chiffre})^*$

chiffres $\rightarrow \text{chiffre} (\text{chiffre})^*$

fraction_opt $\rightarrow . \text{chiffres} \mid \epsilon$

exposant_opt $\rightarrow E (+ \mid - \mid \epsilon) \text{chiffres} \mid \epsilon$

nb $\rightarrow \text{chiffres fraction_opt exposant_opt}$

Avec cette définition, **id** reconnaît a , $a0b$, $begin$

Et **nb** reconnaît 0 , 1.0 , $2E4$, $1.5E-8$, $0.25E-0$

Mais **nb** ne reconnaît pas $0.$, $.1$, $1E2.0$

Notations abrégées possibles :

$r^+ = r r^*$; $r? = r | \epsilon$; $[abc] = a | b | c$; $[a-z] = [ab \dots z]$

Exemples :

lettre $\rightarrow [A-Za-z]$

chiffre $\rightarrow [0-9]$

id $\rightarrow \text{lettre} (\text{lettre} | \text{chiffre})^*$

chiffres $\rightarrow (\text{chiffre})^+$

fraction_opt $\rightarrow (. \text{chiffres})^+$

exposant_opt $\rightarrow (E(+ | -)? \text{chiffres})?$

nb $\rightarrow \text{chiffres} \text{fraction_opt} \text{exposant_opt}$

Langages réguliers : appropriés aux définitions lexicales,
pas aux définitions syntaxiques :
les systèmes de parenthésages ne sont pas réguliers.

2.4 Les diagrammes de transition

Exemple : fragment de grammaire

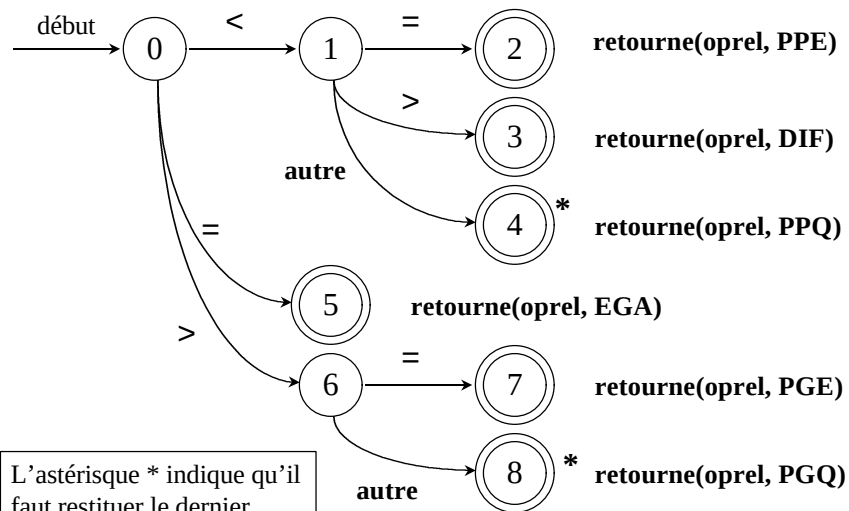
instr $\rightarrow \mathbf{si} \text{ expr } \mathbf{alors} \text{ instr}$
 $\quad \quad \quad | \mathbf{si} \text{ expr } \mathbf{alors} \text{ instr } \mathbf{sinon} \text{ instr} | \epsilon$
expr $\rightarrow \text{terme } \mathbf{oprel} \text{ terme} | \text{terme}$
terme $\rightarrow \mathbf{id} | \mathbf{nb}$

partie
syntaxique

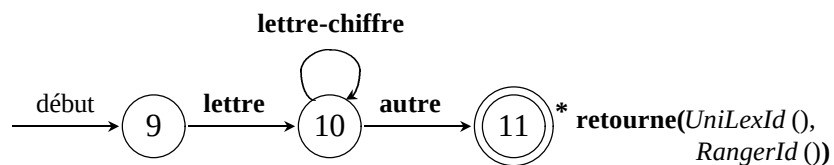
si $\rightarrow \text{si}$
alors $\rightarrow \text{alors}$
sinon $\rightarrow \text{sinon}$
oprel $\rightarrow < | <= | = | <> | > | >=$
id $\rightarrow \text{lettre} (\text{lettre} | \text{chiffre})^*$
nb $\rightarrow \text{chiffre}^+ (.\text{chiffre}^+)? (E(+ | -)? \text{chiffre}^+)?$

partie
lexicale

Diagrammes de transition : unité lexicale **oprel**



Diagrammes de transition : unité lexicale **id** et mots clés



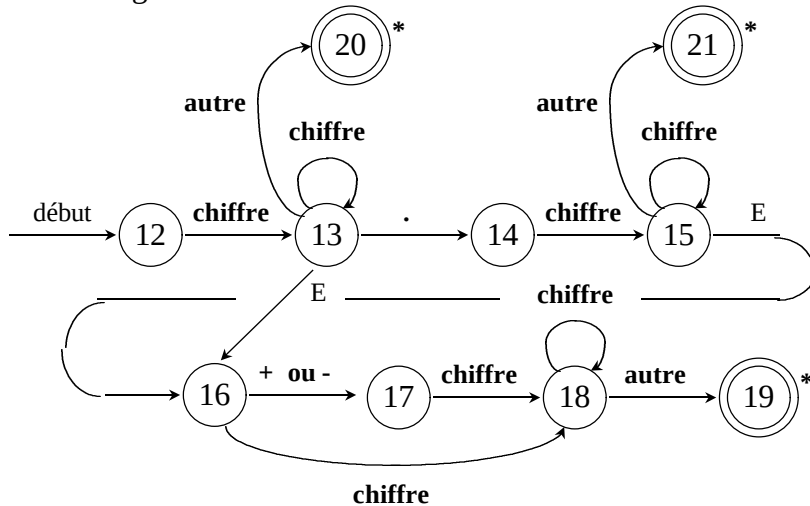
Permet aussi de distinguer les mots clés :

Table des symboles initialisée par les mots clés ;

RangerId examine la table, retourne nouveau pointeur,
ou pointeur existant,
ou 0 si mot clé ;

UniLexId de même retourne **id** ou l'unité lexicale mot clé.

Diagrammes de transition : unité lexicale **nb**



Implantation facile en C ou PASCAL

2.5 Les automates finis

Modèle proche de l'implantation, permettant de répondre "oui" ou "non" selon que la chaîne de caractères en entrée répond ou non à un modèle d'expression régulière donné. Généralisation des diagrammes de transition.

Dans la théorie, pas d'action sémantique associée, mais assez facile de la rajouter.

Utilisations autres que pour l'analyse lexicale liée à un compilateur : recherche de chaînes de caractères, traitement de textes, prétraitement, formatage.

Deux modèles équivalents, déterministe (AFD) et non déterministe (AFN).

Automate fini non déterministe (AFN) ou déterministe (AFD) :

- Ensemble fini d'états E ;
- Alphabet d'entrée fini Σ ;
- Fonction de transition δ ;
- État initial q_0 ;
- Ensemble d'états terminaux F ;

AFD : $\delta : E \times \Sigma \rightarrow E$

- au plus une transition par couple état-lettre ;

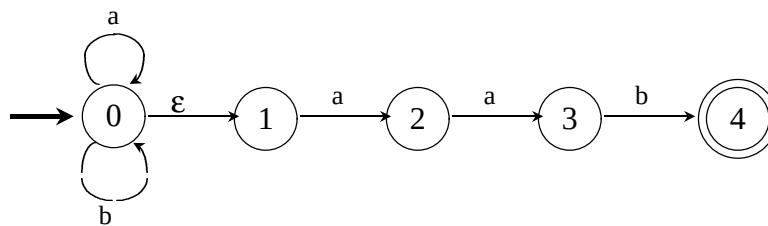
AFN : $\delta : E \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^E$, ensemble des parties de E ;

- plusieurs transitions possibles par couple ;
- possibilités de transitions vides ou ϵ -transitions.

Mots **acceptés** (ou reconnus) formant le **langage accepté**.

Représentation sous forme de graphe de transitions.

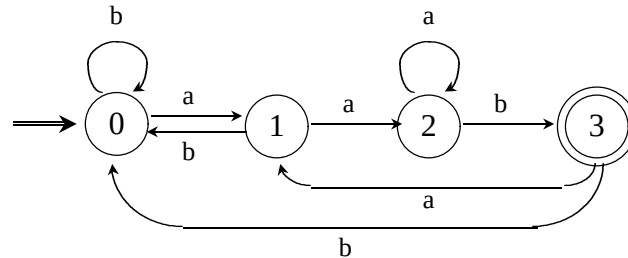
Exemple d'AFN :



Le mot $w = abaab$ est **accepté** (ou reconnu) car **il existe** un chemin de l'état initial à un état terminal tel que la concaténation des symboles apparaissant sur les arcs constituant ce chemin fasse w :

0 a 0 b 0 ϵ 1 a 2 a 3 b 4

Exemple d'AFD :

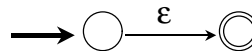


Le mot $w = abaab$ est encore **accepté** : la suite des lettres constituant w est la suite des étiquettes d'un chemin (nécessairement unique) de l'état initial à un état terminal :

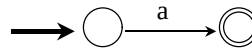
0 a 1 b 0 a 1 a 2 b 3

Construction d'un AFN à partir d'une expression régulière (construction de Thompson).

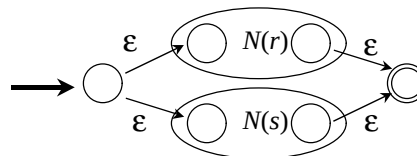
• pour ϵ : $N(\epsilon)$



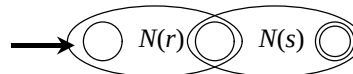
• pour a : $N(a)$



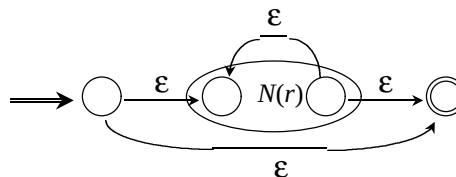
• pour $r \mid s$: $N(r \mid s)$



• pour rs : $N(rs)$

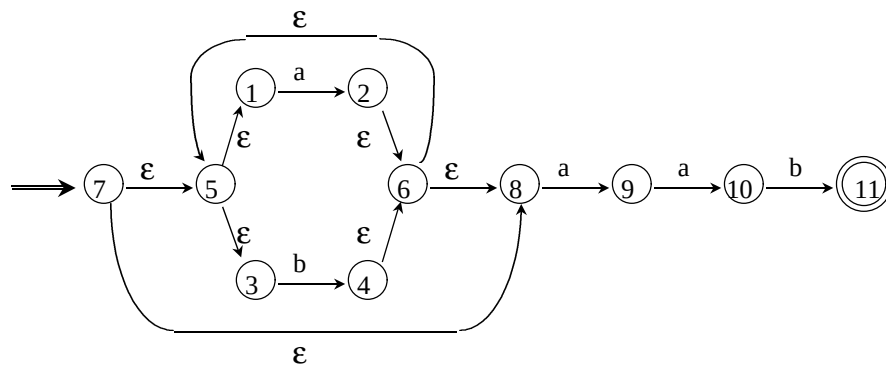


• pour r^* : $N(r^*)$



On réalise d'abord une analyse syntaxique de l'expression régulière, puis construction de Thompson.

Exemple : $r = (a \mid b)^*aab$

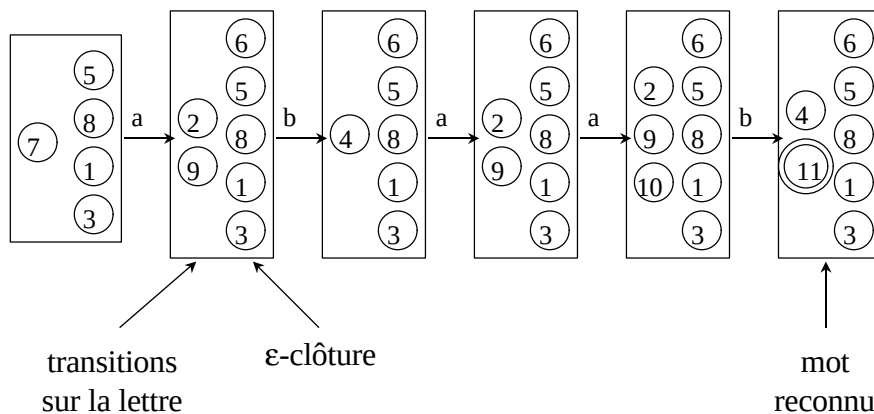


États numérotés suivant l'ordre de construction de l'automate.

Utilisation d'un AFN pour reconnaître un mot :

Construction des chemins correspondants dans l'AFN.

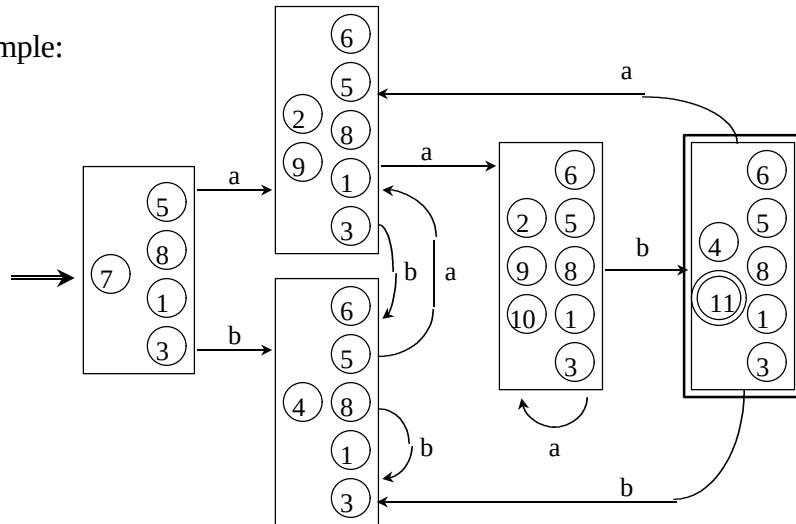
Exemple :



Détermination d'un AFN.

Principe : construire toutes les transitions possibles, complétées par leur ϵ -clôture.

Exemple:



Utilisation des AFN et des AFD :

Langage défini par une expression régulière de longueur k et mot de longueur n .

- 1) Construction de l'AFN en temps $O(k)$ et place $O(k)$;
Construction des chemins dans l'AFN pour tester le mot en temps $O(k \times n)$ et place $O(k)$;
- 2) Construction de l'AFD en temps $O(2^k)$ et place $O(2^k)$;
Test du mot en temps $O(n)$ et place $O(1)$;
- 3) Évaluation paresseuse : ne calculer de l'AFD que la partie effectivement utilisée par le mot à tester.

Approche 1) efficace si peu de mots courts à traiter ;

Approche 2) efficace si de nombreux mots longs à traiter ;

Approche 3) bon compromis général.