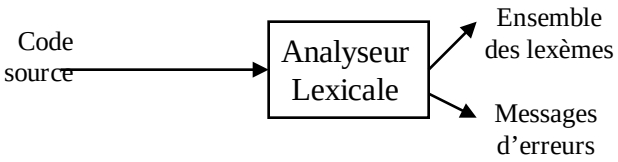
Chapitre 01:
Analyse Lexicale

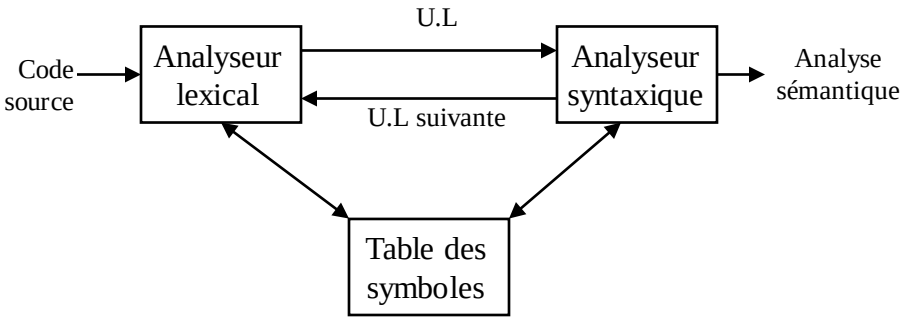
Plan du cours

1. Introduction
2. Définitions
3. Erreurs Lexicales
4. Expressions Régulières
5. Automates
6. Mise en œuvre d'un analyseur lexical

Introduction

- L'analyseur lexical constitue la première étape d'un compilateur. Sa tâche principale est de lire les caractères d'entrée et de produire comme résultat une suite d'unités lexicales que l'analyseur syntaxique aura à traiter.





- L'analyseur lexical réalise certaines tâches secondaires comme:

1. l'élimination de caractères superflus (commentaires, tabulations, fin de lignes,...)
2. La gestion des numéros des lignes dans le programme source.

Définitions

1. **Unité lexicale:**

   Une suite de caractères qui a une signification collective.

2. **Lexème**

   Toute suite de caractère du programme source qui concorde avec **le modèle** d'une unité lexicale.

3. **modèle (Règle lexicale) :**

   Une règle associée à une unité lexicale qui décrit l'ensemble des chaînes du programme qui peuvent correspondre à cette unité lexicale.

4. **Attributs :**

   informations concernant le lexème (champs dans la table des symboles) ;

Erreurs Lexicales

- Peu d'erreurs sont détectables au niveau lexical :
- Plusieurs stratégies sont possibles :

1. Mode panique : on ignore les caractères qui posent problème et on continue. Cette technique transfère le problème à l'analyseur syntaxique

2. Transformations du texte source : insérer un caractère, remplacer, échanger, etc. Elle se fait en calculant le nombre minimum de transformations à apporter au mot qui pose problème pour en obtenir un qui ne pose plus de problèmes.

➔ Cette technique de récupération d'erreur est très peu utilisée en pratique car elle est trop coûteuse à implanter.

Expressions régulières (ER)

- Une expression régulière est une notation pour décrire un langage régulier.
- Soit A un alphabet, une expression régulière est donc:
- Les éléments de A, ε et ϕ sont des expressions régulières.
- Si α et β sont des ERs, alors (α | β), (αβ) et α* sont des ERs.

L'ordre de priorité

Les opérateurs $*$, concaténation et | sont **associatifs à gauche**, et vérifient :

1. $*$ (la répétition).
2. Concaténation.
3. | (l'union).

Remarque

- ε est l'élément neutre par rapport à la concaténation.
- ϕ (l'ensemble vide de caractère), est l'élément neutre par rapport à l'union.

Définitions régulières

La nomination des expressions régulières est dite une définition régulière. Ces noms seront utilisés pour construire d'autres expressions régulières. On écrit donc

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

. . .

$d_n \rightarrow r_n$

où chaque d_i est un nom distinct et chaque r_i est une ER sur $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$

Exemple

lettre → A | B | . . . | Z | a | b | . . . | z

chiffre → 0 | 1 | . . . | 9

id → lettre (lettre | chiffre)*

chiffres → chiffre chiffre *

frac → . chiffres | ε

Exp → (E (+ | - | ε) chiffres) | ε

nb → chiffres frac exp

Notations abrégées

Pour alléger certaines écritures, on complète la définition des ER en ajoutant les notations suivantes :

- Soit x une ER définissant L(x) : $(x)^+$ est une ER définissant $L(x))^+$
- Soit x une ER définissant L(x) : (x)? est une ER définissant $L(x) \cup \{\varepsilon\}$
- Si c_1 , c_2,. . c_k sont des caractères, ER c1 |c2 |... |ck peut se noter $[c_1c_2 \ldots c_k]$,
- [c1–c2] désigne la séquence de tous les caractères c tels que $c_1 \leq c \leq c_2$.

Exemple Les définitions de lettre et chiffre peuvent :

lettre : [A–Za–z]

chiffre : [0–9]

Les mots clés: "for", "if"

Les variables: ['a'-'z']$^+$ ['0'-'9']*

Les entiers: ['0'-'9']$^+$

Les symboles: '(', ')', '+', '*', '='

Le lexème vide: (' ' | '\n')

Automates

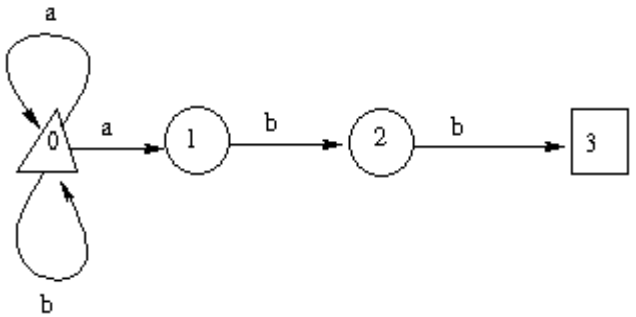
Un automate à états finis (AEF) est défini par:

- Un ensemble fini E d'états
- Un état e_0 distingué comme étant l'état initial
- Un ensemble fini T d'états distingués comme états finaux (ou états terminaux)
- Un alphabet Σ des symboles d'entrée
- Une fonction de transition $\delta: \Sigma \times E \rightarrow E$ qui à tout couple formé d'un état et d'un symbole de fait correspondre un ensemble (éventuellement vide) d'états: $\delta(e_i,a)=\{e_{i1},\ldots,e_{in}\}$
- Les automates sont souvent donnés sous la forme d'un graphe: les états sont les nœuds du graphe et les arcs correspondent à la fonction de transition.

Exemple

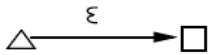
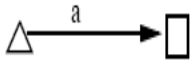
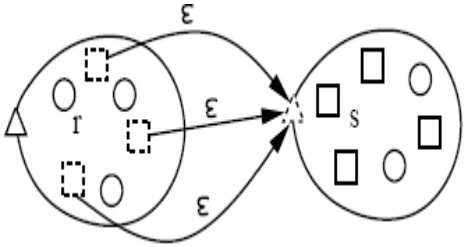
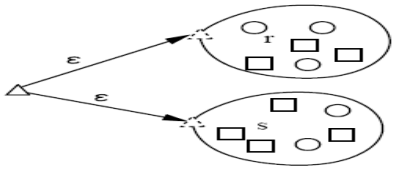
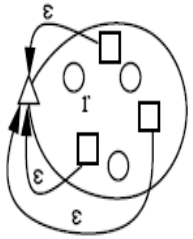
$\Sigma=\{a,b\}$, $E=\{0,1,2,3\}$, $e_0=0$, $T=\{3\}$

$\delta(0,a)=\{0,1\}$, $\delta(0,b)=\{0\}$, $\delta(1,b)=\{2\}$, $\delta(2,b)=\{3\}$,



Construction d'un AFN à partir d'E.Rs

On note A(s) un automate reconnaissant une expression régulière s

automate acceptant la chaîne vide	
automate acceptant la lettre a	
automate acceptant (r)(s) : 1. mettre une ε -transition de chaque état terminal de A(r) vers l'état initial de A(s) 2. les états terminaux de A(r) ne sont plus terminaux 3. le nouvel état initial est celui de A(r) 4. (l'ancien état initial de A(s) n'est plus état initial)	
automate reconnaissant r|s 1. créer un nouvel état initial q 2. mettre une ε-transition de q vers les états initiaux de A(r) et R(s)	
automate reconnaissant r+	

Mise en œuvre d'un analyseur lexical

Manuellement

Un automate déterministe peut très **facilement** être simulé par un algorithme. Alors, on peut écrire un programme reconnaissant tout mot de tout langage régulier. Ainsi, si l'on veut faire l'analyse lexicale d'un langage régulier, il suffit d'écrire un programme simulant l'automate qui lui est associé.

Automatiquement

Il existe des outils pour écrire des programmes simulant des automates à partir de simples définitions régulières. Par exemple : flex

Exemple …