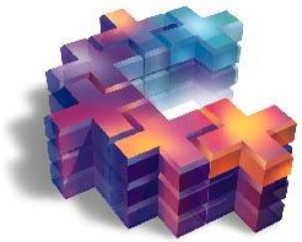
PRINCIPES ET TECHNIQUES DES COMPILATEURS

CH1: ANALYSE LEXICALE

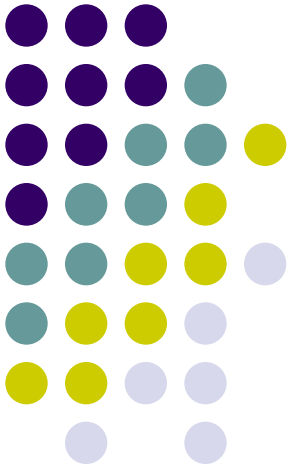


Olfa MOUELHI

olfa.mouelhi@esprit.tn

esprit

Ecole Supérieure Privée
d'Ingénierie et de Technologies

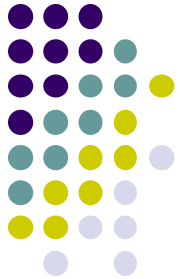


ANNÉE UNIVERSITAIRE 2013 / 2014

PLAN

- Généralités
- Unité lexicale, Lexème et Modèles
- Définition régulière
- Diagramme de transition

ANALYSE LEXICALE

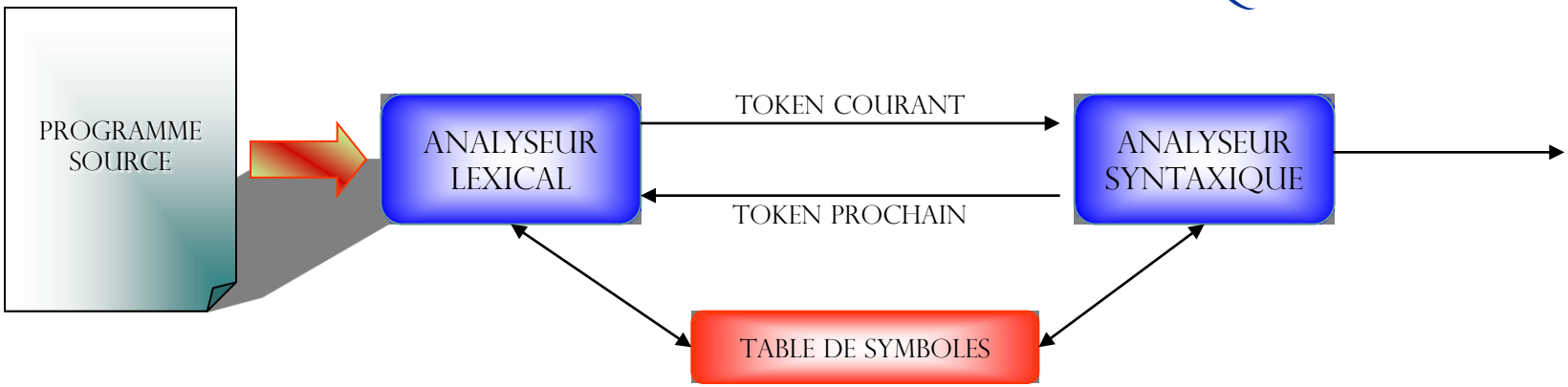


- Première phase de la compilation.
- Reconnaissance d'unité lexicales à partir du code source (flots de caractères).
- Principales unités lexicales :
  - Les caractères spéciaux simples : +, =, etc.
  - Les caractères spéciaux doubles : <=, ++, etc.
  - Les mots-clés : if, while, etc.
  - Les constantes littérales : 123, -5, etc.
  - Les identificateurs : i, vitesse_du_vent, etc.

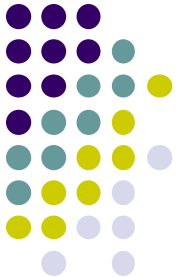
RÔLE DE L'ANALYSEUR LEXICAL

- Lire les caractères du texte d'entrée.
- Supprimer les blancs, les commentaires, etc.
- Former des unités lexicales (tokens).
- Passer des couples <unité lexicales,valeur lexicale> à l'Analyseur Syntaxique.
- Relier les messages d'erreurs issus du compilateur au code source.
- Traitement préprocesseur s'il y en a.

SCHÉMA USUEL D'IMPLÉMENTATION DE L'ANALYSEUR LEXICAL – INTERACTION AVEC L'ANALYSEUR SYNTAXIQUE



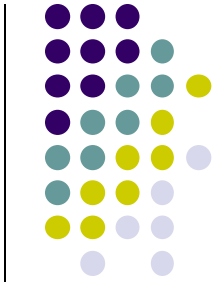
UNITÉS LEXICALES, LEXÈMES ET MODÈLES



A propos d'une unité lexicale reconnue dans le texte source on doit distinguer quatre notions importantes :

- L'unité lexicale.
- le lexème.
- éventuellement, un attribut.
- le modèle.

UNITÉS LEXICALES, LEXÈMES ET MODÈLES

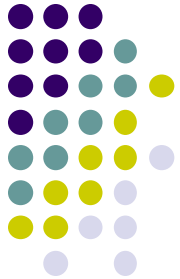


UNITÉ LEXICALE

Pour la plupart des langages de programmation, les constructions suivantes sont traitées comme des unités lexicales :

- Mots clés.
- Opérateurs.
- Identificateurs.
- Constantes.
- Chaînes littérales.
- Symboles de ponctuation (‘(’, ‘)’, ‘,’, ‘:’, etc.)

UNITÉS LEXICALES, LEXÈMES ET MODÈLES



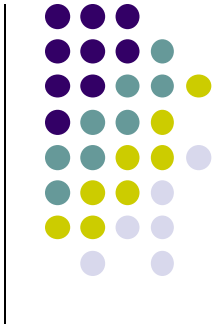
LEXÈME

Un lexème est une suite de caractères du programme source qui concordent avec le modèle de l'unité lexicale. Exemple :

```
const max_length = 256;
```

Dans la déclaration précédente, la chaîne de caractères max_length est un lexème de l'unité lexicale Identifier.

UNITÉS LEXICALES, LEXÈMES ET MODÈLES

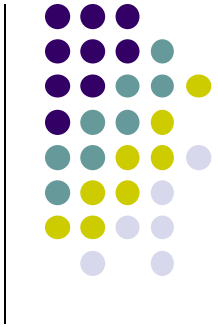


MODÈLE

le modèle sert à spécifier l'unité lexicale.

- Pour les mots réservés tels que **`const`**, **`if`**, **`while`**, etc. le lexème et le modèle coïncident généralement. Le modèle de l'unité lexicale **`const`** est la chaîne **`const`**.
- Pour une unité lexicale **`rel_oper`** qui représente les opérateurs relationnels, le modèle est l'ensemble des opérateurs relationnels : <, < =, ==, >=, >, !=
- Pour décrire précisément les modèles des unités lexicales plus complexes tels que les identificateurs et les nombres, on utilise les expressions régulières .
- Dans la programmations dirigée par les modèles, ces derniers sont exprimés par des expressions régulières.
- Des langages et outils permettent d'engendrer une reconnaissance efficace par automates finis des expressions régulières.

UNITÉS LEXICALES, LEXÈMES ET MODÈLES



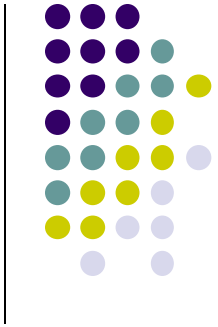
ATTRIBUT

L'attribut, s'il existe, dépend de l'unité lexicale en question, et la complète. Seules les dernières des dix unités précédentes ont un attribut :

- Pour un nombre, il s'agit de sa valeur (123, -5).
- Pour un identificateur, il s'agit d'un renvoi à une table dans laquelle sont placés tous les identificateurs rencontrés

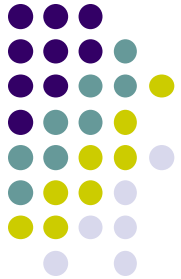
Pour des raisons de diagnostic, on peut mémoriser le numéro de ligne où une unité lexicale apparaît pour la première fois. L'unité lexicale aussi bien que le numéro de ligne peuvent être rangés dans l'entrée de la table de symboles associée.

UNITÉS LEXICALES, LEXÈMES ET MODÈLES



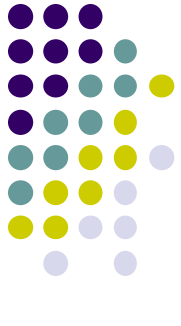
UNITÉ LEXICALE	LEXÈMES	DESCRIPTION INFORMELLE DES MODÈLES
const	const	const
if	if	if
rel_oper	< <= == != >= >	< <= == != >= >
identifier	e pi length	Lettre suivie de lettres ou de chiffres ou le caractère '_'
number	3.141 256 0.196	Constantes numériques.
literal	« stack overflow »	Chaînes de caractères entre guillemets.

SPÉCIFICATION DES UNITÉS LEXICALES



- Les chaînes.
- Les langages.
- Les expressions régulières.
- Les définitions régulières.

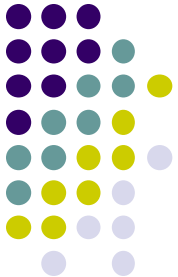
SPÉCIFICATION DES UNITÉS LEXICALES



LES CHAÎNES (OU MOTS ET PHRASES)

- Un alphabet est un ensemble fini de symboles. Exemples : {0; 1}, {A; C; G; T}, l'ensemble de toutes les lettres, l'ensemble des chiffres, le code ASCII, etc.
- Les caractères blancs (c'est-`a-dire les espaces, les tabulations et les marques de fin de ligne) ne font généralement pas partie des alphabets.
- Une chaîne (on dit aussi mot) sur un alphabet $\sum$ est une séquence finie de symboles extraits de $\sum$. Exemples, respectivement relatifs aux alphabets précédents : 00011011, ACCAGTTGAAGTGGACCTTT, Bonjour, 2001. On note ε la chaîne vide, ne comportant aucun caractère.
- La longueur d'une chaîne *s*, noté $|s|$ est le nombre d'occurrences de symboles dans s. la chaîne ε est de longueur 0.
- Un *langage* sur un alphabet $\sum$ est un ensemble de chaînes construites sur $\sum$. Exemples triviaux : $\varnothing$, le langage vide, $\{\varepsilon\}$, le langage réduit à l'unique chaîne vide. Des exemples plus intéressants (relatifs aux alphabets précédents) : l'ensemble des nombres en notation binaire, l'ensemble des chaînes ADN, l'ensemble des mots de la langue française, etc.

LES DÉFINITIONS RÉGULIÈRES



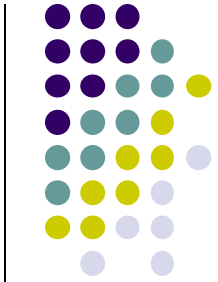
Les définitions régulières permettent de donner des noms à des ER définies sur un alphabet Σ à partir de symboles de base et de les utiliser comme s'ils étaient des symboles de Σ.

- $d_1 \rightarrow r_1$
- $d_2 \rightarrow r_2$
- …
- $d_n \rightarrow r_n$

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

AXIOME	DESCRIPTION
$r \mid s = s \mid r$	| est commutatif
$r \mid (s \mid t) = (r \mid s) \mid t$	| est associatif
$r(st) = (rs)t$	La concaténation est associative
$r(s \mid t) = (rs) \mid rt$ et $(s \mid t)r = sr \mid tr$	La concaténation est distributive pa rapport à |
$\varepsilon r = r$ et $r\varepsilon = r$	ε est l'élément neutre pour la concaténation.
$r^* = (r \mid \varepsilon)^*$	Relation entre * et ε.
$r^{**} = r^*$	* est idempotent.

RAPPEL : LES EXPRESSIONS RÉGULIÈRES



EXEMPLE

letter → A | B | … | Z | a | b | … | z

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

identifier → letter (letter | digit)*

number → digit+

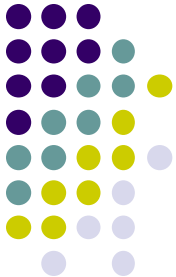
exponent → (E(+|-|ε)number)|ε

float → number exponent

NOTATIONS ABRÉGÉES

- L'opérateur unaire postfixe + signifie « au moins une instance »
  - $r^* = r+ \mid \varepsilon$
  - $r+ = rr^*$
- L'opérateur unaire postfixe ? signifie « zéro ou une instance » ($r? = r \mid \varepsilon$)
- Classe de caractères : la notation [*abc*] où *a*, *b* et *c* sont des symboles d'alphabets signifie *a* | *b* | *c*.
- Une classe de caractères [*a* – *z*] dénote l'expression régulière *a* | *b* | …| *z*.
- Identificateurs : [*A* – *Za* – *z*] [*A* – *Za* – *z0* – *9*]*

RECONNAISSANCE DES UNITÉS LEXICALES



Considérons la grammaire suivante :

if	→	if
then	→	then
else	→	else
relop	→	< | <= | = | <> | >= | >
letter	→	[A – Za – z]
digit	→	[0 – 9]
number	→	[0 – 9]+
id	→	[A – Za – z]([A – Za – z 0 – 9])*
float	→	number+ (.number+)? (E (+ | -)? Number+)?
instruction	→	**if** expression **then** instruction
	|	**if** expression **then** instruction **else** instruction
expression	→	term **relop** term
term	→	**id** | **float**
separator	→	**space | tab | nl**
blank	→	**separator+**

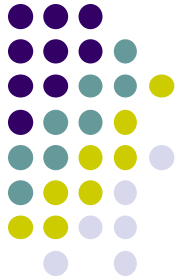
OBJECTIF :

Construire un analyseur lexical qui isole les lexèmes associés à la prochaine unité lexicale et qui produise un couple composé de l'unité lexicale appropriée et d'une valeur d'attribut en utilisant la table décrite dans ce qui suit.

Les blancs définis par l'ER «blank» sont éliminés par l'analyseur lexical.

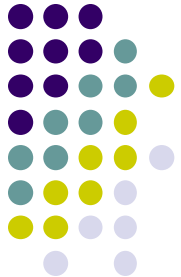
Lorsque l'analyseur lexical rencontre des blancs, il continue à la recherche d'une unité lexicale significative qu'il retourne à l'analyseur syntaxique.

RECONNAISSANCE DES UNITÉS LEXICALES



EXPRESSION RÉGULIÈRE	UNITÉ LEXICALE	VALEUR D'ATTRIBUT
blank		
if	**if**	
then	**then**	
else	**else**	
id	**id**	Pointeur vers une entrée de la table de symboles.
float	**float**	Pointeur vers une entrée de la table de symboles.
<	**relop**	**LT**
<=	**relop**	**LE**
=	**relop**	**EQ**
<>	**relop**	**DIFF**
>=	**relop**	**GE**
>	**relop**	**GT**

DIAGRAMMES DE TRANSITIONS



- Les diagrammes de transitions décrivent les actions qui sont réalisées quand l'analyseur syntaxique appelle un analyseur lexical pour fournir la prochaine unité lexicale.

- Un état initial du diagramme.

- En entrant dans un état on lit le prochain caractère. Si l'étiquette d'un arc sortant de l'état courant concorde avec le caractère d'entrée on passe à l'état pointé par cet arc. Autrement on signale une erreur.

DIAGRAMMES DE TRANSITIONS

EXEMPLES

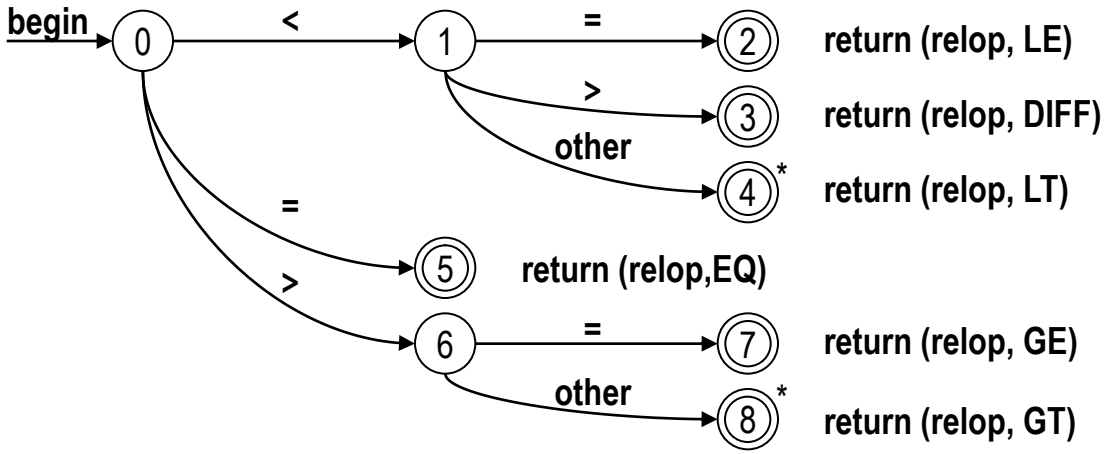


DIAGRAMME DE TRANSITION POUR LES OPÉRATEURS DE RELATION

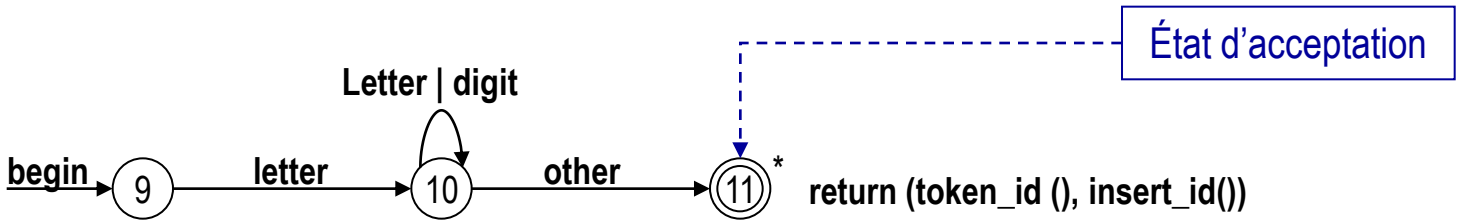
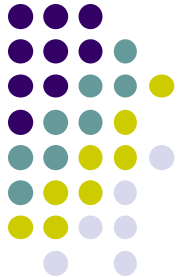


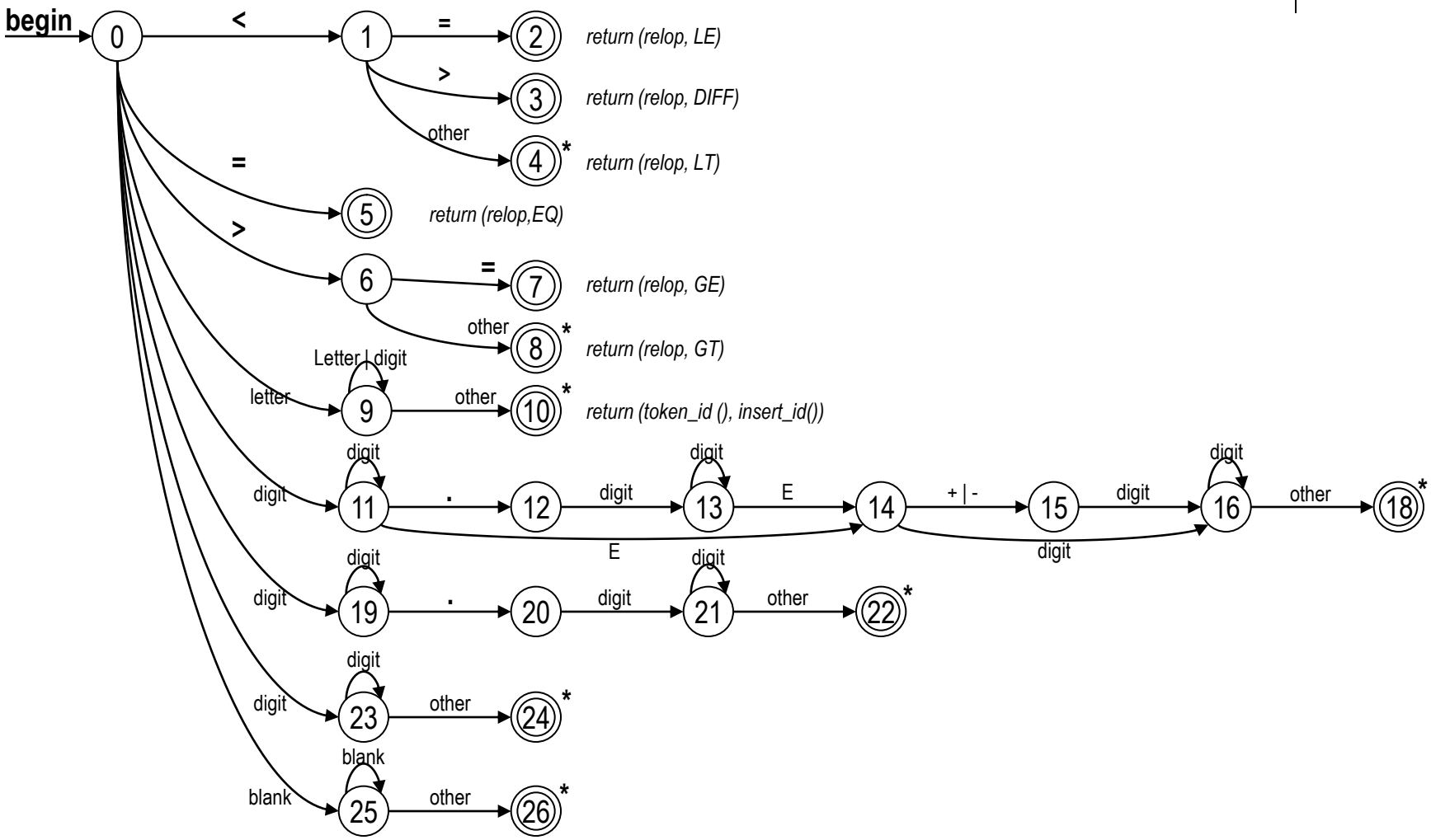
DIAGRAMME DE TRANSITION POUR LES IDENTIFICATEURS ET MOTS CLÉS

DIAGRAMMES DE TRANSITIONS

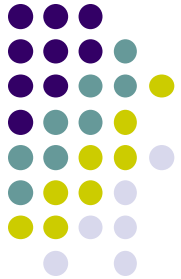


- Bien séparer les mots clés des identificateurs.
- Placer les mots clés (*if*, *then*, *else*) dans la table de symboles avant de commencer l'analyse.
- Noter dans la table de symboles l'unité lexicale à retourner lorsque une de ces chaînes est reconnue.
- La procédure *insert_id* a accès au tampon où l'unité lexicale a été trouvée. Elle fonctionne comme suit:
  - On examine la table de symboles. Si le lexème est trouvé avec l'indication mot clé on retourne 0.
  - Si on trouve le lexème comme variable, on retourne un pointeur sur une entrée de la table de symboles.
  - Si le lexème n'est pas trouvé dans la table de symboles, il y est rangé et un pointeur sur cette nouvelle entrée est retourné.
- La procédure *token_id* recherche le lexème dans la table de symboles, si c'est un mot clé, l'unité lexicale correspondante est retournée sinon un ***id*** est retourné.

RÉALISATION DE DIAGRAMMES DE TRANSITIONS



RÉALISATION DE DIAGRAMMES DE TRANSITIONS

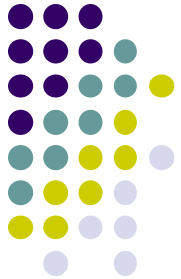


```
token lexical () {
   while (TRUE) {
      switch (state) {
         case  0: c = nextchar ();
                  if (c == SPACE || c == TAB || c == EOL);
                  else if (c == '<') state = 1;
                  else if (c == '=') state = 5;
                  else if (c == '>') state = 6;
                  else state = failure ();
                  break;
         /* ... States  1 to 8 */
         case 9:  c = nextchar (input);
                  if (isalnum (c)) state = 10;
                  else state = failure ();
                  break;
         case 10: back (1);
                  insert_id ();
                  return (token_id());
         /* ... State 12 to 22 */
         case 23: c = nextchar ();
                  if (!isdigit (c)) state = 24;
                  break;
         case 24: back (1);
                  insert_nb ();
                  return NB;
      }
   }
}
```

back (*n*) recule de *n* caractères dans le buffer

failure () : routine de reprise sur erreur

TYPE DES ERREURS



- Identificateur trop long
- Constante caractère trop longue
- Caractère invalide
- Fin de fichier dans un commentaire : erreur fatale
- Fin de fichier dans une chaîne de caractères: erreur fatale
- Erreur dans une constante numérique
- Ligne trop longue
- …