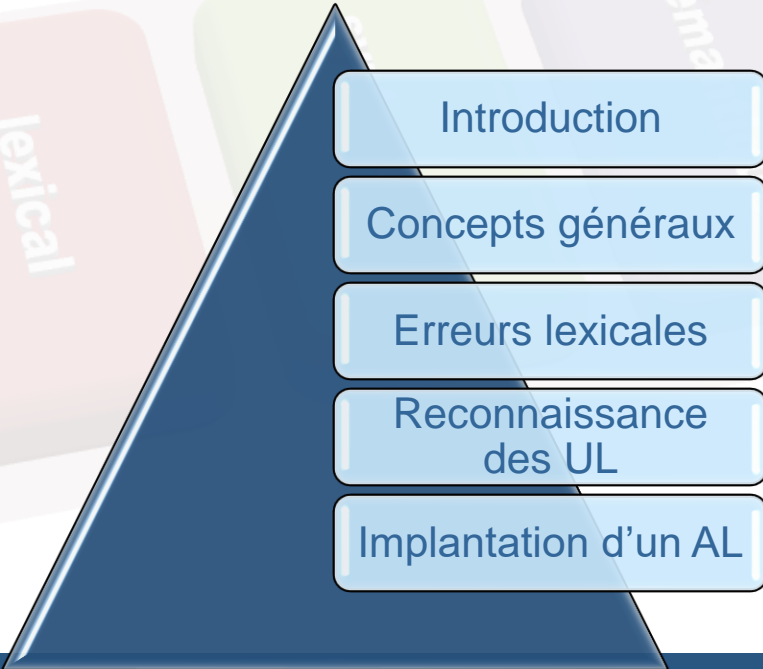


Chapitre 2

Analyse lexicale

- 
- Introduction
 - Concepts généraux
 - Erreurs lexicales
 - Reconnaissance des UL
 - Implantation d'un AL

©Myiam Fourati Cherif 2020-2021

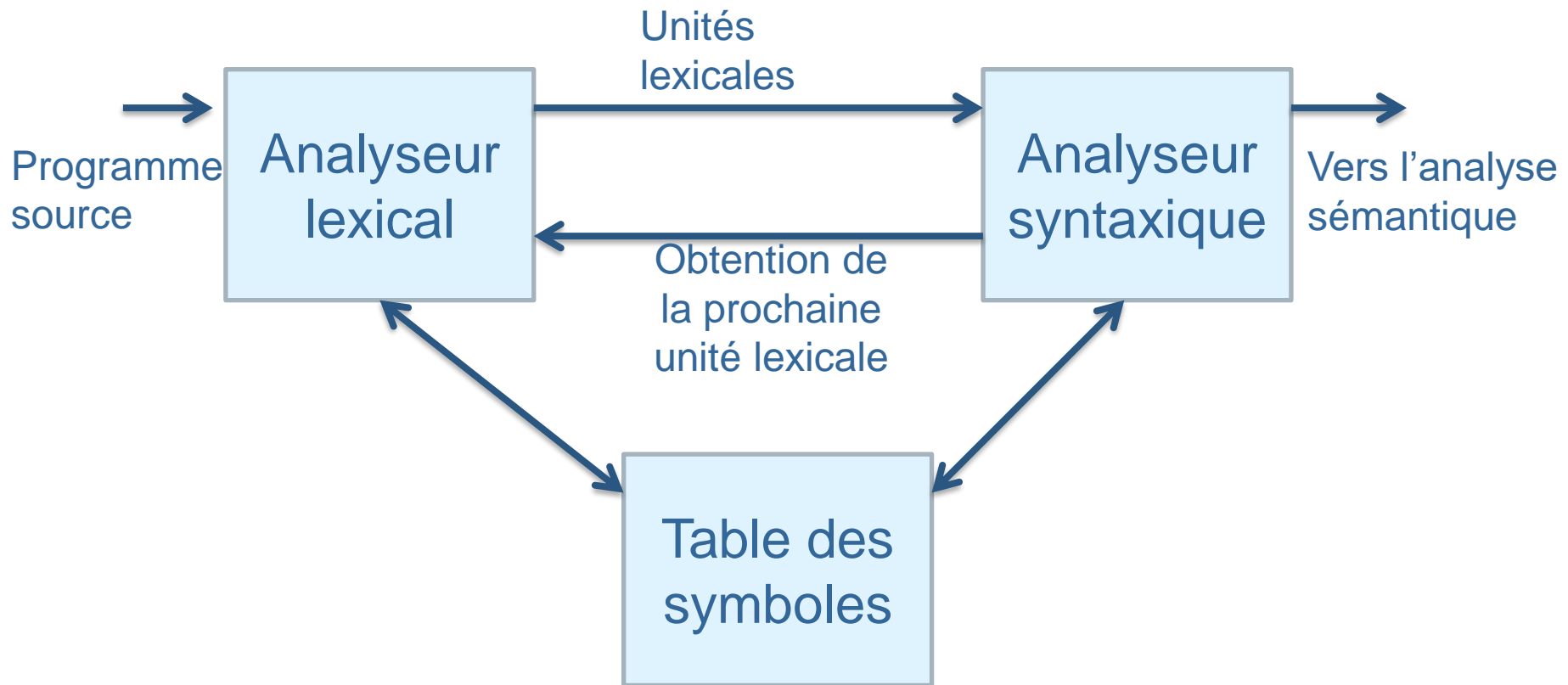
Pourquoi a-t-on besoin d'un analyseur lexical (scanner) ?

- ❑ Le texte source comprend des données inutiles comme les espaces, les fins de ligne, les commentaires, etc.
- ❑ Certains mots du code source possèdent des formes équivalentes (les identificateurs, les nombres littéraux, etc.)
- ❑ L'analyse lexicale permet de traiter ces cas pour simplifier l'écriture de l'analyseur syntaxique.
- ❑ Augmente l'efficacité du compilateur.
- ❑ Augmente la portabilité du compilateur : si le langage ciblé change, seul l'analyseur lexicale doit être modifié

Rôle de l'analyseur lexical

- ❑ Première phase d'analyse du compilateur.
- ❑ Lit les caractères en entrée (code source), les groupe en mots (**lexèmes**) et produit une suite de symboles (**unités lexicales**) correspondant à chaque mot du programme source. Ces unités sont plus facilement manipulables par l'analyseur syntaxique.
- ❑ Reçoit les requêtes d'indentification des mot de l'analyseur syntaxique.
- ❑ Lit le code source, donc il peut procéder à des tâches secondaires :
 - ❑ Elimine les commentaires, les caractères blancs, de tabulation et de fin de lignes, etc.
 - ❑ Relie les messages d'erreurs issus du compilateur au code source (fait correspondre chaque erreur à la ligne correspondante).

Interaction avec l'analyseur syntaxique



Unités lexicales, motifs et lexèmes

- ❑ **Unité lexicale** : suite de caractères ayant une signification collective.
- ❑ **Motif (modèle)** : règle qui décrit la forme commune d'un ensemble de chaînes du programme appartenant à une même unité lexicale. Il est exprimé par une expression régulière :
 - ❑ chiffre $\rightarrow 0|1|\dots|9$
 - ❑ Lettre $\rightarrow A|B|\dots|Z|a|b|\dots|z$
 - ❑ id $\rightarrow \text{lettre}(\text{lettre}|\text{chiffre})^*$: unité lexicale pour les identificateurs.
- ❑ **Lexème** : séquence de caractères dans le code source qui concorde avec le motif d'une unité lexicale (instance).

Exemple

Unité lexicale	Lexèmes	Description du modèle (informelle)
const	Const	Constante
if	If	Le caractère i suivi de f
comp	<, <=, =, ==, >, !=	< ou > ou <= ou >= ou == ou !=
id	pi, count, x1, res	Lettre suivie de lettres ou de chiffres
nbre	3.14459, 0.36, 1,	Toutes les constantes numériques
chaîne	"Ceci est une chaîne"	Tous les caractères entre " et " sauf ".

- En C :
 - `const float pi = 3.1416,`
 - `pi` est un lexème de l'unité lexicale **id**.
 - `3.1416` est un lexème de l'unité lexicale **nbre**.

Attributs des unités lexicales

- ❑ Si plusieurs lexèmes concordent avec un motif d'une unité lexicale, l'analyseur lexical doit fournir aux phases suivantes des informations additionnelles sur le lexème reconnu.
- ❑ Par exemple le motif **nbre** (nombre) correspond à la fois au lexème 0 et 1, ceci n'est pas important pour l'AS, mais il est essentiel pour le générateur de code de savoir quel lexème a été lu.

Attributs des unités lexicales

- ❑ L'analyseur lexical réunit les informations sur les unités lexicales dans les attributs qui leur sont associés :
 - ❑ Le nom de l'unité lexicale est utilisé par la phase analyse syntaxique,
 - ❑ La valeur de l'attribut est utilisée par la phase de traduction.
- ❑ Une unité lexicale a en général un seul attribut : un pointeur vers l'entrée de la table de symboles qui conserve les informations liées à l'unité lexicale (le lexème, son type, l'endroit où il a été rencontré la première fois, etc.). Le pointeur devient l'attribut de l'unité lexicale.

Exemple d'attributs

□ Programme Fortran :

□ $E = M * C ** 2$

□ Unités lexicales et attributs associés :

- **<id**, pointeur vers l'entrée de E dans la table des symboles>
- **<op_assignment>**
- **<id**, pointeur vers l'entrée de M dans la table des symboles>
- **<op_multiplication>**
- **<id**, pointeur vers l'entrée de C dans la table des symboles>
- **<op_exponentiation>**
- **<nbre**, valeur entière 2>

Erreurs lexicales

- ❑ Peu d'erreurs sont détectées dans cette phase.
- ❑ L'analyseur lexicale a une vision locale du code source.
- ❑ Exemple C :
 - ❑ `fi (x ==3) ...`
 - ❑ Comment détecter que le lexème `fi` est le mot clé `if` mal saisi ?
 - ❑ Il peut s'agir d'un identificateur puisqu'il correspond au motif de l'unité lexicale `id`.
- ❑ L'analyseur lexicale détecte les suites de caractères qui ne correspondent à aucun motif d'unités lexicales, comme `2toto`.

Erreurs lexicales

- ❑ Si l'analyseur lexical est bloqué faute de correspondance d'un lexème avec les motifs existants, il peut utiliser une stratégie de rattrapage :
 - ❑ Mode panique : ignore les caractères qui causent problème et il continu, en le signalant à l'utilisateur.
 - ❑ Modifier le code source par une seule transformation :
 - ❑ suppression d'un caractère,
 - ❑ Insertion d'un caractère,
 - ❑ Remplacement d'un caractère par un autre,
 - ❑ Permutation de caractères.
 - ❑ Calculer le nombre minimum de transformations à apporter pour corriger le problème : peu utilisée car coûteuse.

Des expressions régulières pour des unités lexicales

- ❑ Les expressions régulières sont adéquates pour décrire les motifs des lexèmes (unités lexicales).
- ❑ Définition régulière des identificateurs :
 - ❑ $lettre \rightarrow A|B|\dots|Z|a|b|\dots|z$
 - ❑ $chiffre \rightarrow 0|1|\dots|9$
 - ❑ $id \rightarrow lettre(lettre|chiffre)^*$
- ❑ Extensions aux ER :
 - ❑ Zéro ou une occurrence : $r?$ est équivalent à $r|\epsilon$.
 - ❑ Classes de caractères :
 - ❑ $a_1|a_2|\dots|a_n$ est remplacé par $[a_1a_2\dots a_n]$.
 - ❑ $a_1|a_2|\dots|a_n$ est remplacé par $[a_1-a_n]$ si les a_i est une suite logique.
 - ❑ $[\wedge a-z]$: n'importe quel caractère sauf $a\dots z$.
 - ❑ $r+=rr^*$.

Des expressions régulières pour des unités lexicales

□ Définition régulière des identificateurs devient :

□ *lettre* \rightarrow [A-Za-z]

□ *chiffre* \rightarrow [0-9]

□ *id* \rightarrow *lettre*(*lettre*|*chiffre*)*

Dispositif de reconnaissance des unités lexicales

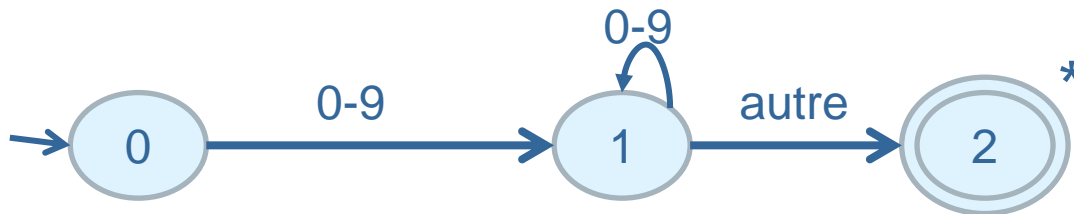
- ❑ Il doit accepter ou rejeter une séquence, exactement comme un AF.
- ❑ Mais, ce n'est pas suffisant :
 - ❑ Il lit les caractères en entrée jusqu'à ce qu'il identifie une unité lexicale,
 - ❑ Retourne l'UL lue à l'analyseur syntaxique et
 - ❑ Prépare le tampon pour le prochain appel.
- ❑ Il doit donc retourner une action à l'analyseur syntaxique (unité lexicale et valeur d'attribut correspondant).
- ❑ Il doit pouvoir reculer d'une position si le symbole rencontré ne fait pas partie du lexème accepté.
- ❑ On l'appellera **Diagramme de transition**.

Exemple

□ Reconnaissance de l'unité lexicale : *entier*.



Automate fini



retourner(nbre, valeur)

Diagramme de
transition pour les
nombres entiers

Exercice

- ❑ Tracer le diagramme de transition pour l'unité lexicale *comp* (opérateurs de comparaison) : $<$, $<=$, $<>$, $=$, $>=$, $>$.
- ❑ Tracer le diagramme de transition pour l'unité lexicale *Id* (Identificateurs).

Exemple 1

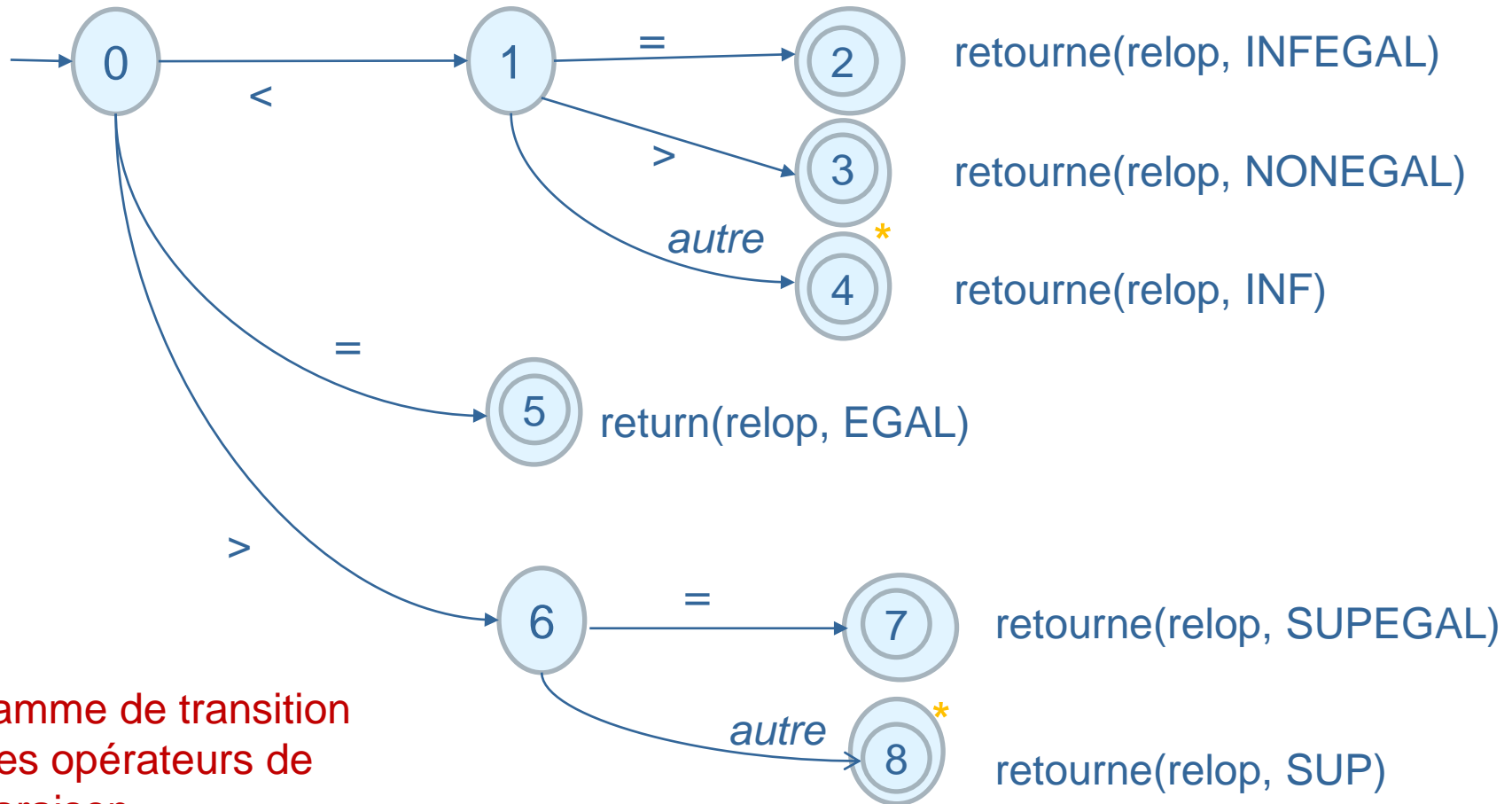


Diagramme de transition
pour les opérateurs de
comparaison

Exemple 2

- On supposera que les mots clés sont préalablement stockés dans la table des symboles.
- Donc ce DT reconnaît également les mots clés.



Diagramme de transition
pour les Id et des mots clés

Reconnaissance des mots réservés et des identificateurs

- ❑ Les **mots réservés** ont le même motif que les identificateurs, mais ne correspondent pas au motif *id*.
- ❑ Deux solutions :
 - ❑ Les considérer comme des identificateurs :
 - ❑ Les charger dès le départ dans la table des symboles,
 - ❑ Un champ de la table indiquera qu'il s'agit d'un mot réservé en indiquant l'UL correspondante,
 - ❑ Dès l'identification d'un *id*, un appel spécial le placera dans la table des symboles s'il n'y est pas, sinon il s'agit d'un mot réservé.
 - ❑ Créer un DT pour chaque mot réservé ensuite les intégrer dans un DT global.

Mots réservés considérés comme des identificateurs

Table des
symboles

if	keyword
while	keyword
then	keyword
do	keyword
...	
var	id

- Cette méthode est préférable (voir obligatoire) si l'analyseur lexical est codé à la main. Pourquoi ?

Des expressions régulières pour tous les mots réservés

- ❑ Les mots réservés peuvent être des préfixes pour des identificateurs (*do* et *done*).
- ❑ L'analyseur lexicale utilisant cette technique est beaucoup plus complexe, mais essentiel pour l'utilisation de générateurs d'analyseurs lexicaux à partir de spécification donnée (ER).
- ❑ Exercice : Tracer le DT reconnaissant les identificateurs, les entiers et le mot réservé *do*.

Que faire si le lexème est reconnu par deux motifs ?

- ❑ Le lexème le plus long est retenu (on dit que l'AL est *glouton*, il fonctionne toujours de cette façon) :
 - ❑ do et dot : dot est pris comme *id*.
 - ❑ > et >= : >= est pris comme *comp*.
- ❑ Si 2 lexèmes correspondent (il peut être un mot clé ou un id), prendre le premier qui correspond au motif :
 - ❑ Si la spécification de l'AL décrit les ER dans cet ordre :
 - ❑ w.h.i.l.e : mot réservé **while**.
 - ❑ letter.(letter | digit)* : **id**
 - ❑ Si le mot while est lu.
 - ❑ Alors le mot réservé **while** est retenu.
 - ❑ Si l'ordre des spécifications change, le mot clé ne sera pas détecté et while sera considéré comme un *id*.
 - ❑ On privilégie en principe les mots clés.

Implantation d'un DT

- Pour écrire un analyseur lexical, nous disposons de 3 alternatives :
 - Analyse lexicale prédictive : une façon rudimentaire, consiste à écrire l'AL « en dur ». Le programme peut se baser sur les diagrammes de transition (voir exemple p21). Il sera **plus efficace** et **compact**, mais compliqué à coder.
 - Utiliser un générateur automatique d'analyseur lexicaux (ex. Flex) : plus facile, rapide, maintenable, mais moins efficace.
 - Ecrire un programme qui implante l'automate à états finis à partir de sa table de transition (plus efficace). Cette technique est utilisée par les générateurs d'AL.

Fragment du code pour l'UL opComp inspiré du DT

```
int uniteSuiivante(void) {  
    char c;  
    c = lireCar(); /* etat = 0 */  
    while (estBlanc(c))  
        c = lireCar();  
    if (c == '<') {  
        c = lireCar(); /* etat = 1 */  
        if (c == '=')  
            return INFEG; /* etat = 2 */  
        else  
            if (c == '>')  
                return DIFF; /* etat = 3 */  
            else {  
                delireCar(c); /* etat = 4 */  
                return INF;  
            }  
    }  
    else if (c == '=')  
        return EGAL; /* etat = 5 */  
    else if (c == '>') {  
        c = lireCar(); /* etat = 6 */  
        if (c == '=')  
            return SUPEG; /* etat = 7 */  
        else {  
            delireCar(c); /* etat = 8 */  
            return SUP;  
        }  
    }  
}
```

Code proposé
par Pascal
MIETLICKI

FIG. 1: Début du programme pour l'analyse lexicale

Utilisation d'un outil (Flex)

- ❑ Définit un analyseur lexical en spécifiant des expressions régulières pour les motifs des unités lexicales (lex.l).
- ❑ Le compilateur Flex transforme les motifs d'entrées en un diagramme de transition.

