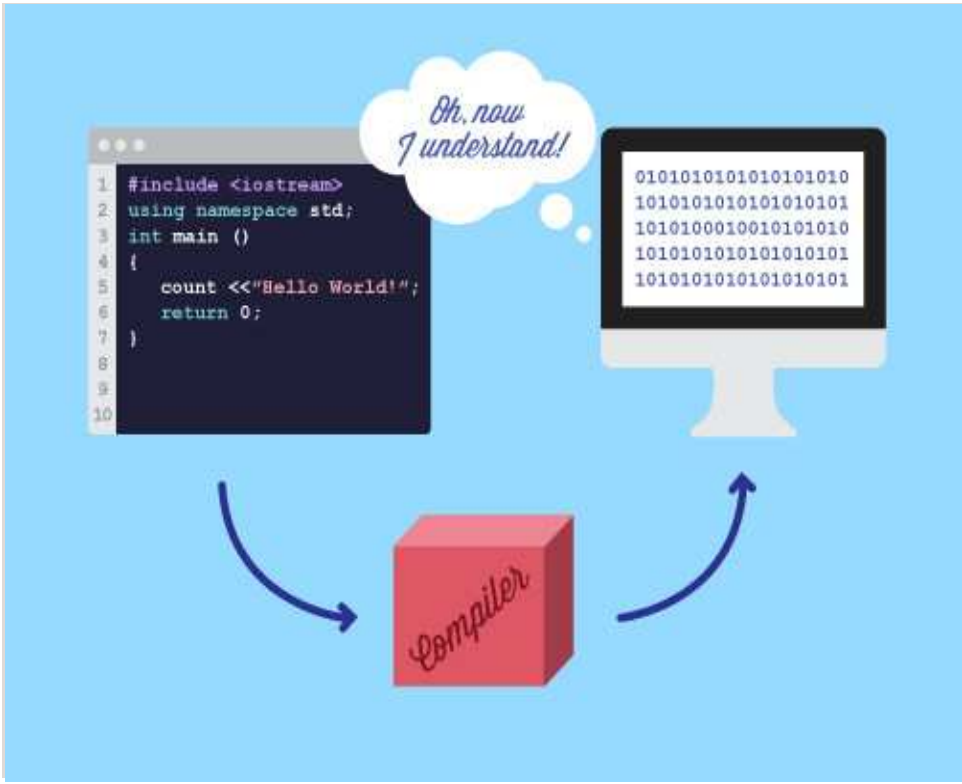
TECHNIQUES DE COMPILATION

2ème SI
Houda Benali

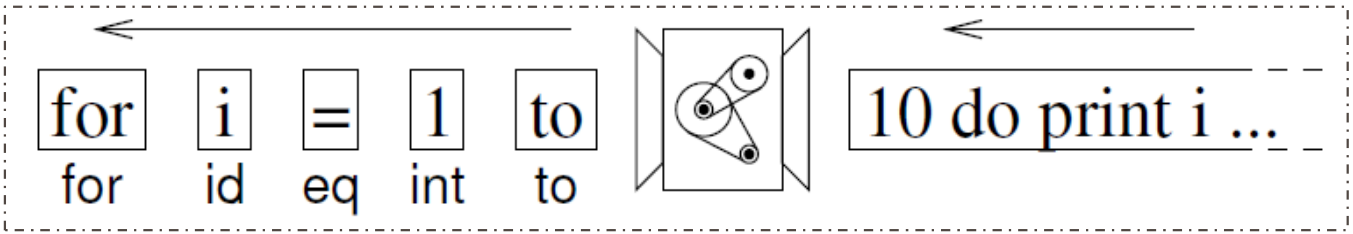
ISSAT Mateur 2020/2021



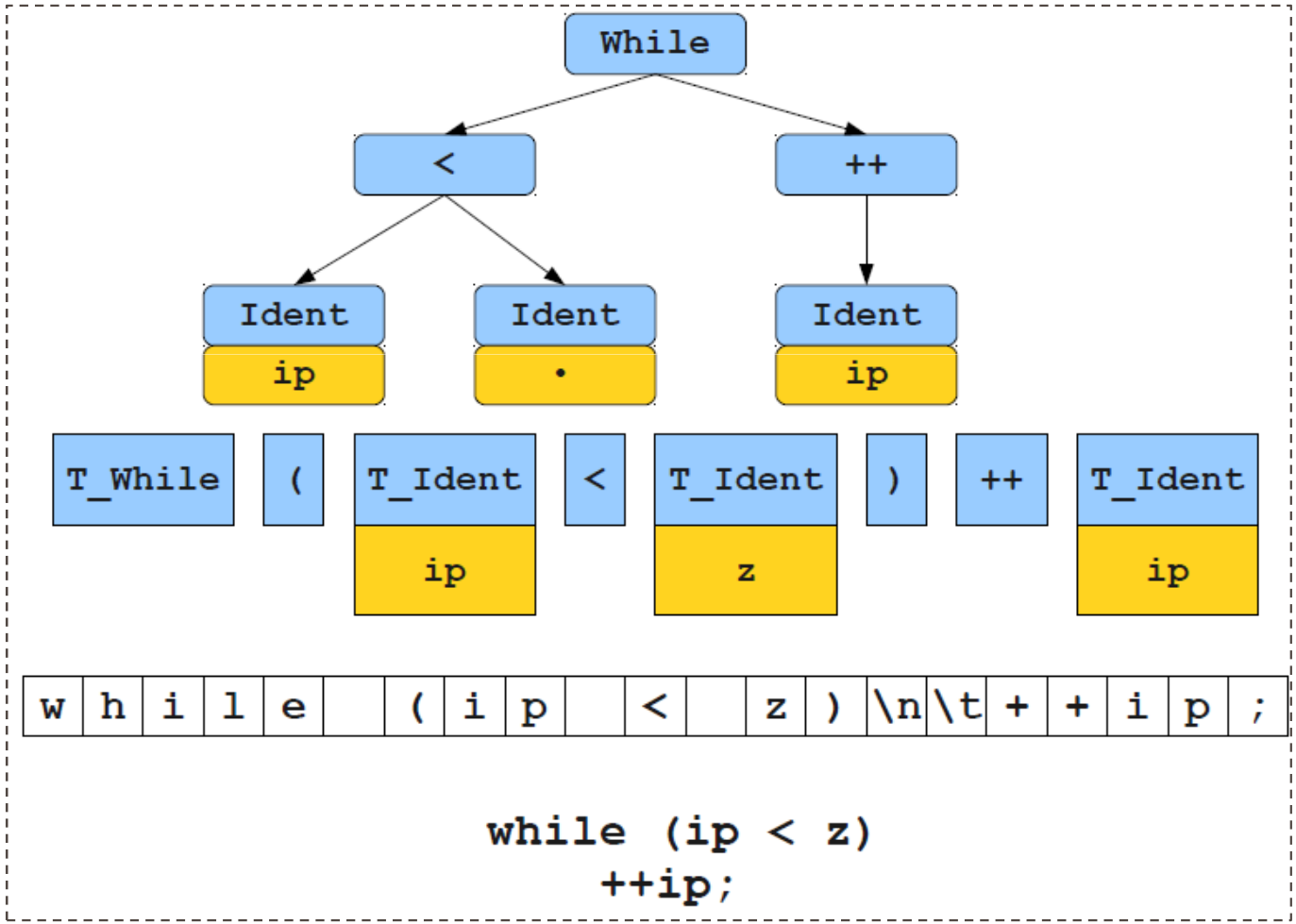
3 - ANALYSE LEXICALE

Objectifs de l’analyse lexicale

- transformer une suite de caractères en une suite d’entités de haut niveau appelées **unités lexicales** « **tokens** ».
  - Il s’agit de reconnaître les « types » des « mots » lus. Pour cela les caractères sont regroupés en **unités lexicales** « **tokens** ».
- associer à chaque **unité lexicale** un **lexème** (étiquette + suite de caractères du code source+une position (ligne, colonne))
- éliminer les caractères superflus (commentaires, espaces, passages à la ligne,…) ;



Objectifs de l’analyse lexicale



Objectifs de l’analyse lexicale

- gérer aussi les numéros de ligne dans le programme source pour pouvoir associer à chaque erreur rencontrée par la suite la ligne dans laquelle elle intervient ;
- identifier et traiter les parties du texte qui ne font pas partie à proprement parler du programme mais sont des directives pour le compilateur.

Les blancs (espaces)

- les blancs (espace, retour chariot, tabulation, etc.) jouent un rôle dans l'analyse lexicale ;
  - ils permettent de séparer deux lexèmes ainsi intx est compris comme un seul lexème (l'identificateur intx) et int x est compris comme deux lexèmes (le mot clé int et l'identificateur x)
- de nombreux blancs sont néanmoins inutiles (comme dans x + 1) et simplement ignorés
- les blancs n'apparaissent pas dans le flot de lexèmes renvoyé

Défis de l’analyse lexicale

- Comment partitionner le programme en une suite de lexèmes?
- Comment associer correctement à chaque lexème une unité lexicale?

Spécification d'une unité lexicale

- Une unité lexicale (UL) est une suite de caractères qui a une signification collective ;
- *Exemples :*
  - *les chaines <, >, <=, >= sont des opérateurs relationnels. L'unité lexicale est OPREL (par exemple).*
  - *Les chaines toto, ind, tab, ajouter sont des identificateurs (de variables, ou de fonctions). L'unité lexicale est IDENT (par exemple).*
  - *Les chaines if, else, while sont des mots clefs.*
  - *Les symboles , . ; ( ) sont des séparateurs. L'unité lexicale est SEP (par exemple).*

Spécification d'une unité lexicale

- **Modèle d'une unité lexicale**
  - Un **modèle** est une règle associée à une unité lexicale qui décrit l'ensemble des chaines du programme qui peuvent correspondre à cette unité lexicale.
  - Pour décrire le modèle d'une unité lexicale, on utilisera des **expressions régulières**.
- **Lexème**
  - suite de caractères du programme source qui concorde avec le modèle d'une unité lexicale.

Spécification d'une unité lexicale

- **Exemples :**
  - *L'unité lexicale IDENT (identificateurs) en C a pour modèle : toute suite non vide de caractères composée de chiffres, lettres ou du symbole '_' et qui ne commencent pas par un chiffre.*
    - *Lexèmes possibles : truc, i, a, ajouter, valeur …*
  - *L'unité lexicale NOMBRE (entier signé) a pour modèle : toute suite non vide de chiffres précédée éventuellement d'un seul caractère parmi {+,-}.*
    - *Lexèmes possibles : +0, -12, 444 …*

Spécification d'une unité lexicale

- **Attributs :**
  - Un attribut est toute information **supplémentaire inutile pour l'analyseur syntaxique** mais **utile pour les autres phases de la compilation**
  - Exemples :
    - Un analyseur syntaxique n'a pas besoin de distinguer entre < et <= (***attribut*** utile pour la génération de code) mais a besoin seulement de savoir qu'ils sont des opérateurs relationnels appartenant à l'UL OPREL
    - L'analyseur syntaxique a besoin seulement de savoir qu'un lexème (var, x, ajouter …) appartient à l'UL IDENT. L'adresse mémoire est un ***attribut*** nécessaire lors de la génération de code.
    - Le type des variables représente aussi un ***attribut*** important pour l'analyse sémantique mais sans aucune importance pour l'analyse syntaxique

Erreurs détectées par l'analyseur lexical

- **Erreur lexicale** : se produit lorsque l'analyseur est confronté à une suite de caractères qui ne correspond à aucun modèle parmi les modèles (i.e. ER implémentées par des automates) qu'il a à sa disposition
- Exemple :
  - Dans un code source, l'analyseur rencontre une suite de caractère '**2var**' ➔ l'analyseur signale que cette suite ne correspond à aucune UL
- Possibilité de proposer une correction
  - Approche à base d'apprentissage à partir de l'historique des corrections effectuées par le développeur (technique de l'IA)

Outils existants

- Des générateurs automatiques d'analyseurs lexicaux
  - Approche plus performant que le développement à partir de zéro !
  - Exemple : lex, flex, ocamllex, sableCC4 …
  - Entrée : fichier contenant les expressions régulières des unités lexicales
  - Sortie: fichier (analyseur lexical) dans un langage cible associé (C pour lex et flex, ocaml pour ocamllex, java pour sableCC4) à compiler par le compilateur du langage

Remarque 1

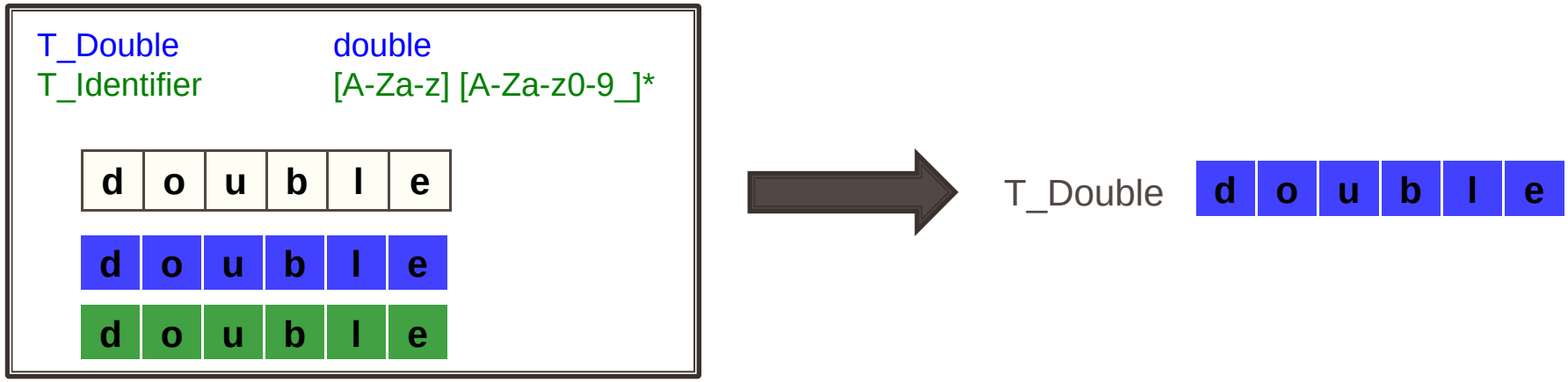
- Dans un code source, l'analyseur rencontre une suite de caractère '**elsef**'
  - Est-ce que c'est le mot clé ***else*** ou un identificateur ***elsef*** ➔ **Ambigüité**
- ***Pas de place à l'ambigüité ➔ il faut définir des règles***
  - ***Par convention :*** il faut chercher la suite correspondant à la plus longue unité lexicale
  - **Le lexème le plus long gagne toujours !**
- **Exercice**
  - **Traitez l'exemple suivant:**
    - **« var , , i »**

Remarque 1

- Dans un code source, l'analyseur rencontre une suite de caractère '**elsef**'
  - Est-ce que c'est le mot clé ***else*** ou un identificateur ***elsef*** ➔ **Ambigüité**
- ***Pas de place à l'ambigüité ➔ il faut définir des règles***
  - ***Par convention :*** il faut chercher la suite correspondant à la plus longue unité lexicale
  - **Le lexème le plus long gagne toujours !**
- **Exercice**
  - **Traitez l'exemple suivant:**
    - **« var , , i »**
      - **quatre lexèmes : IDENT 'var' puis SEP ',' puis SEP ',' puis IDENT 'i'**

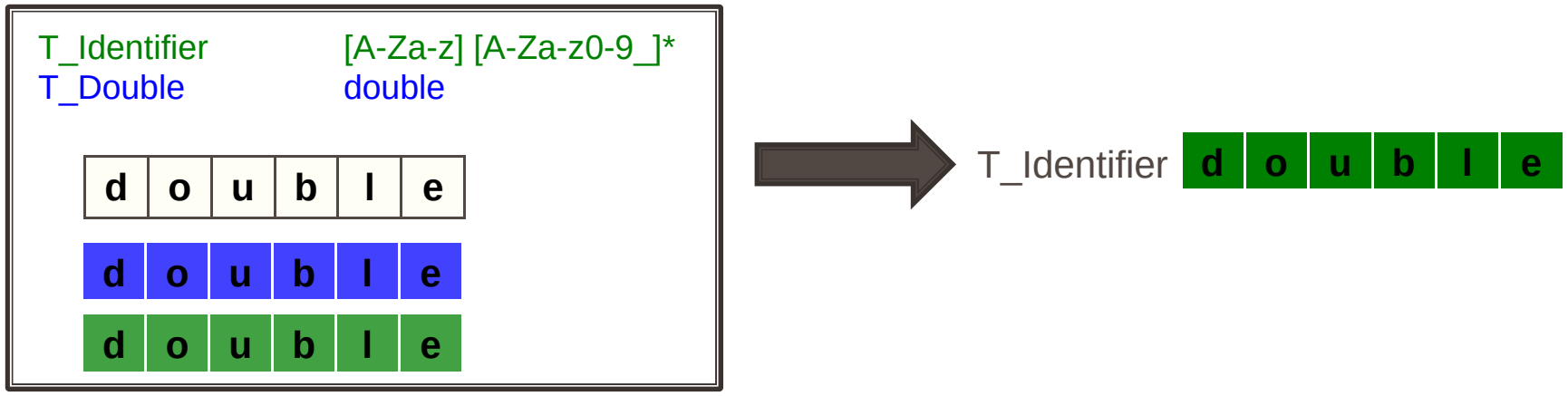
Remarque 2

- Un mot qui correspond à deux unités lexicale !
  - **Choisir la règle la plus prioritaire**
    - **Solution 1** : ordre de priorité = ordre de déclaration (la règle définie en premier est la plus prioritaire) (comme dans Lex)



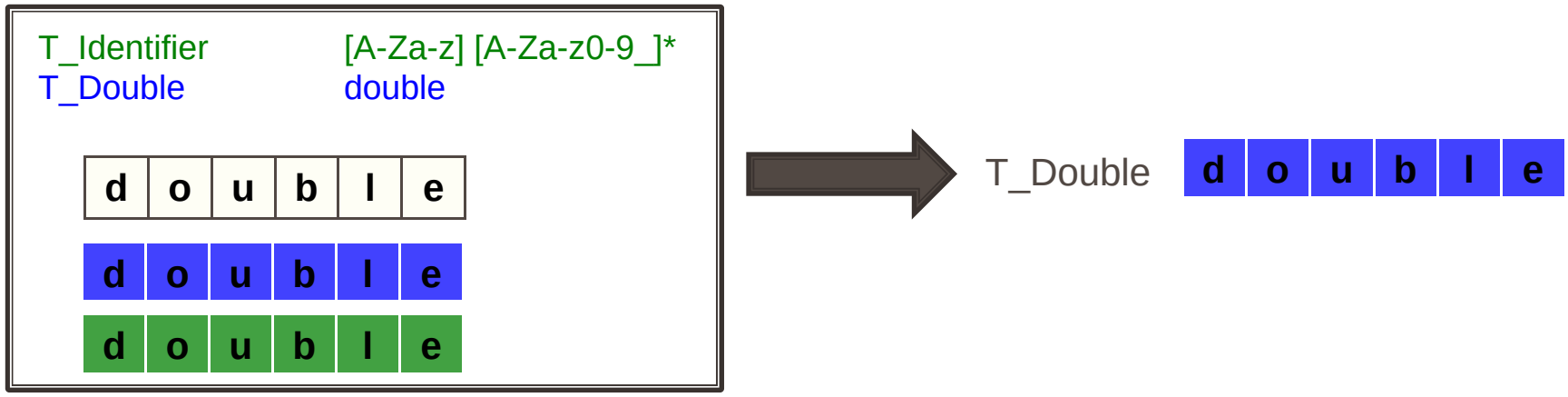
Remarque 2

- Un mot qui correspond à deux unités lexicale !
  - **Choisir la règle la plus prioritaire**
    - **Solution 1** : ordre de priorité = ordre de déclaration (la règle définie en premier est la plus prioritaire) (comme dans Lex)



Remarque 2

- Un mot qui correspond à deux unités lexicale !
  - **Choisir la règle la plus prioritaire**
    - **Solution 2** : priorité d'inclusion (comme dans SableCC4) : Une expression régulière strictement incluse dans une autre gagne la priorité

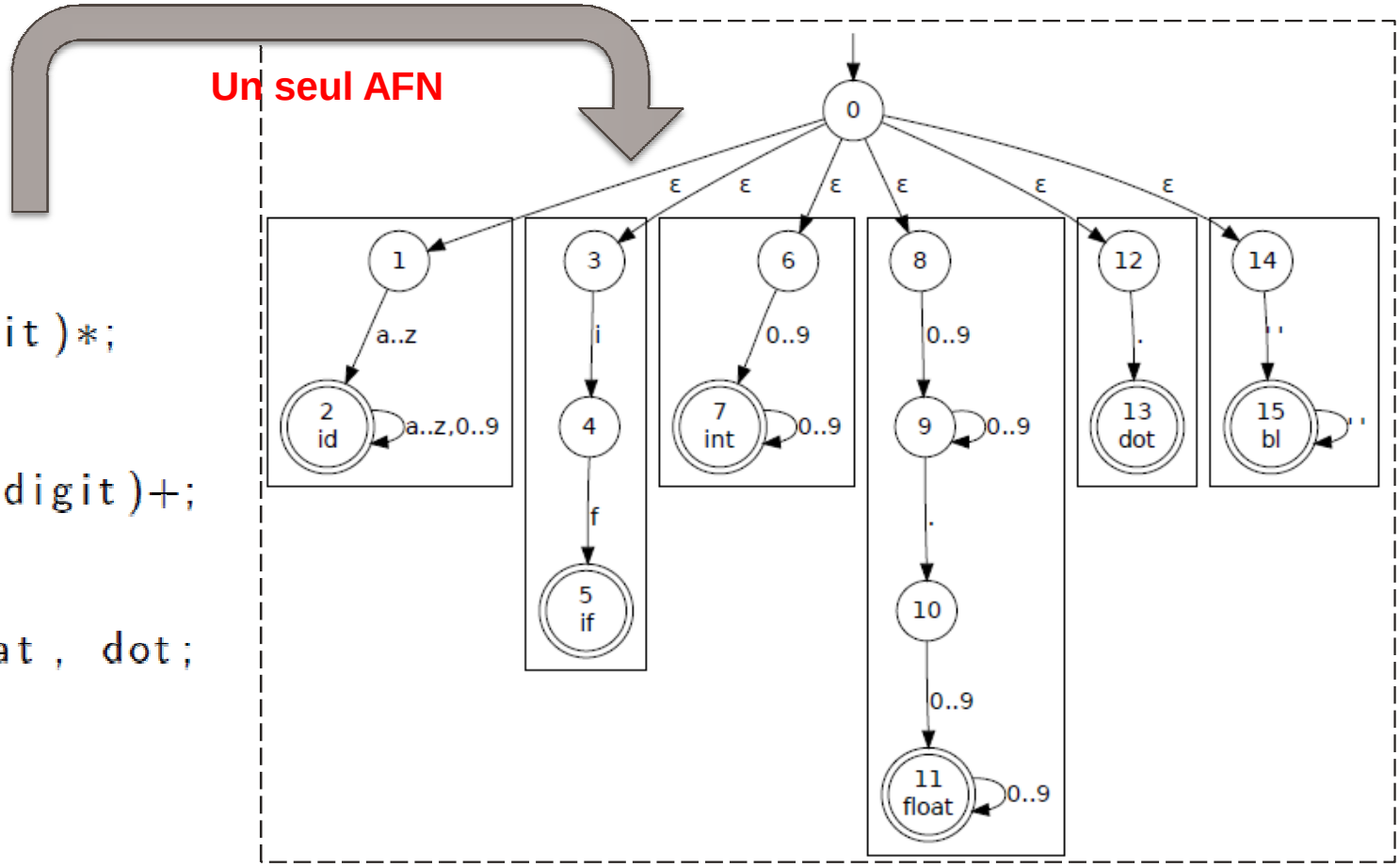


Implémentation des analyseurs lexicaux

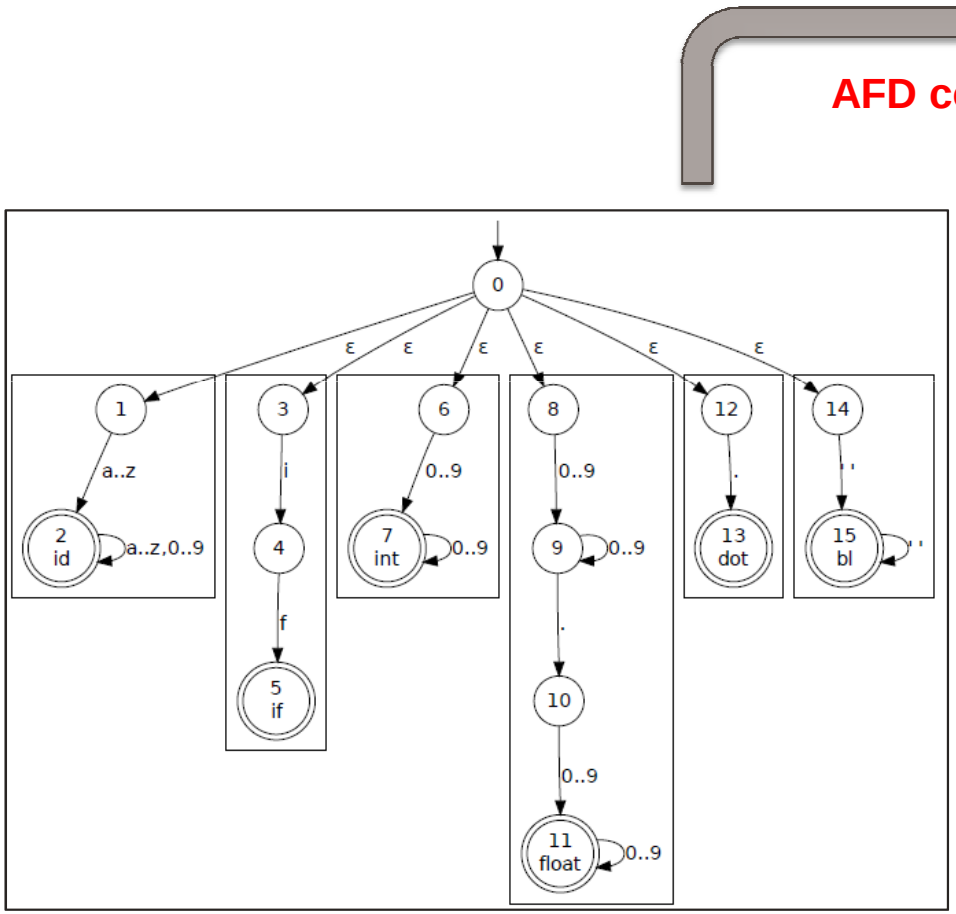
- Une procédure à suivre pour implémenter un analyseur lexical pour un langage donné pourrait se résumer comme suit :
  1. Spécifier chaque type d'entité lexicale à l'aide d'une expression régulière
  2. Convertir chaque expression régulière obtenue en 1 en un automate fini
  3. Construire l'automate union de tous les automates de l'étape 2
  4. Rendre l'automate de l'étape 3 déterministe
  5. Réduire au minimum, le nombre des états de l'automate de l'étape 4.
  6. Implémenter l'automate obtenu à l'étape 5

Exemple avec SableCC4

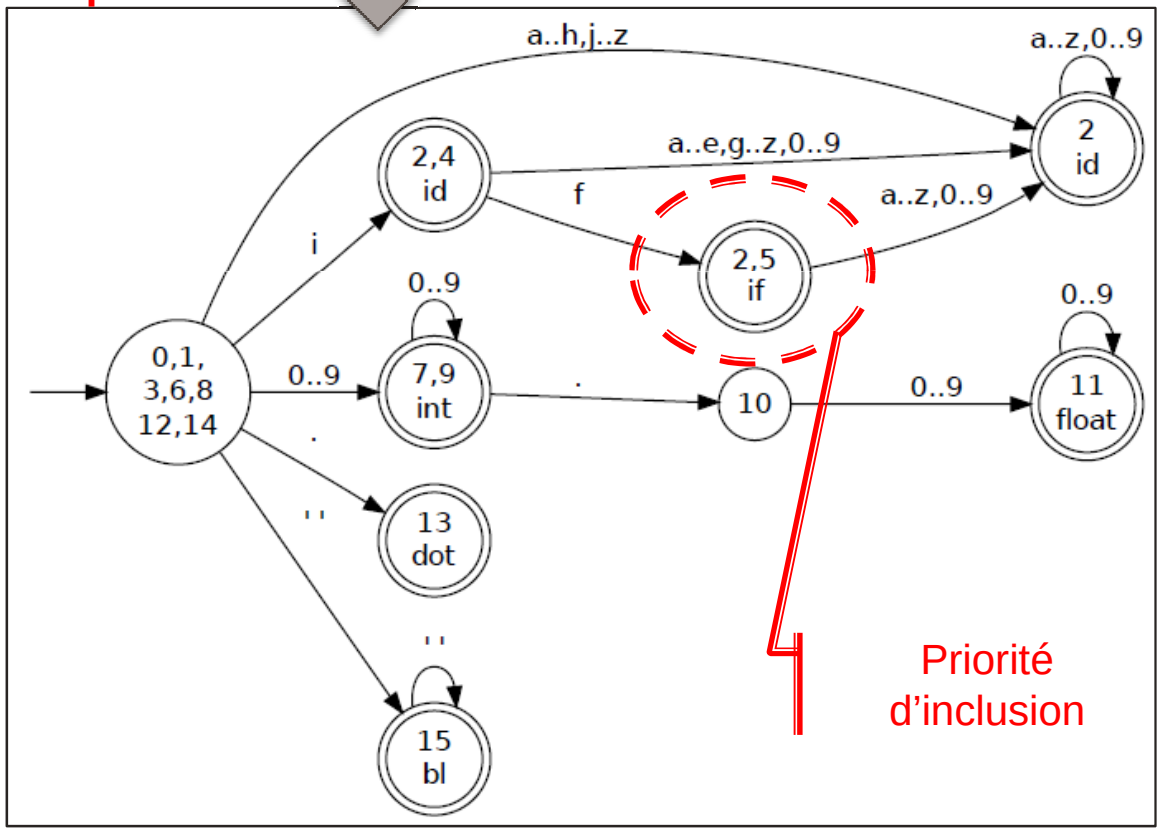
```
Grammar automate:
Lexer
   letter = 'a'..'z';
   digit = '0'..'9';
   id = letter(letter|digit)*;
   if = 'if';
   int = (digit)+;
   float = (digit)+ '.' (digit)+;
   dot = '.';
   bl = '␣';
   Token id, if, int, float, dot;
   Ignored bl;
```



Exemple avec SableCC4

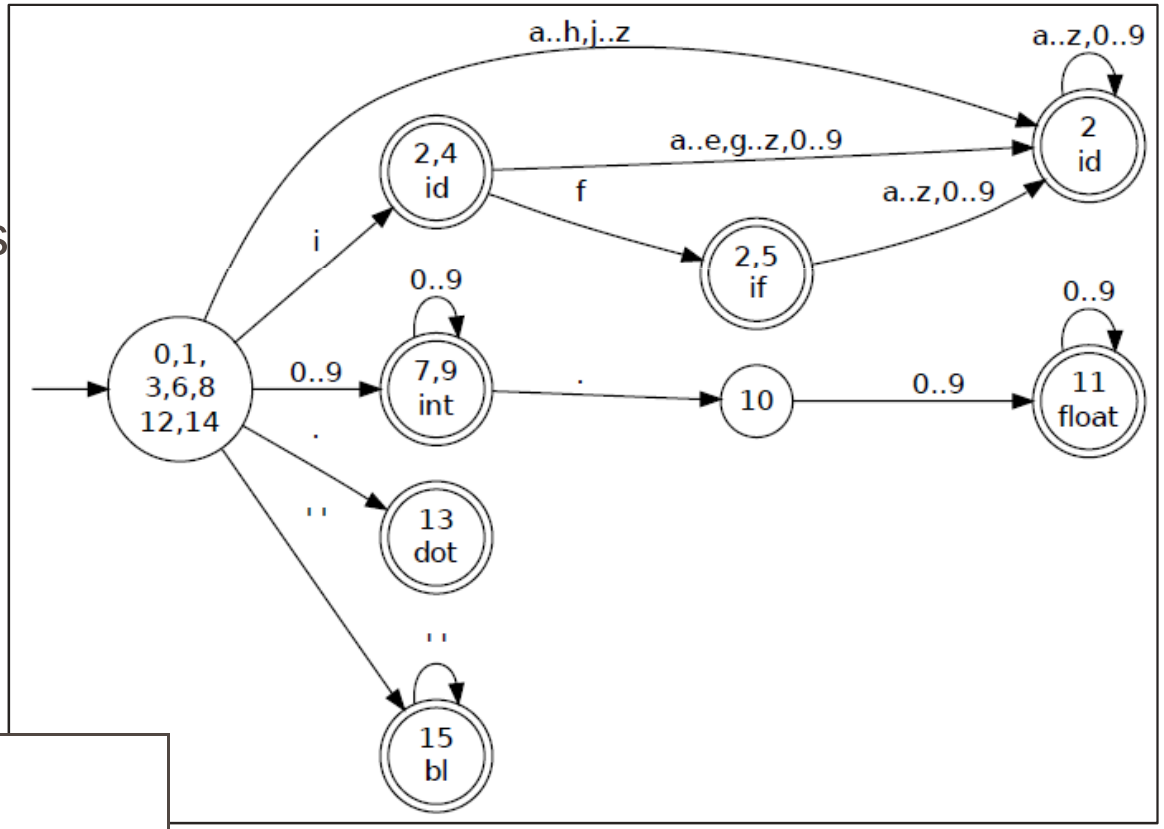


AFD correspondant



Principe d’extraction des lexèmes

- Avancer
  - Le plus loin possible dans l'automate
  - En mémorisant le dernier état d'acceptation rencontré
- Si avancer est impossible, alors
  - Retourner le lexème du dernier état d'acceptation rencontre
  - Repartir de l‘état de départ de l'automate
  - Commencer au caractère qui suit le lexème retourné
- Exemple : pour if02.42



Mémoire d’états	
Lexèmes	

Principe d'extraction des lexèmes : algorithme

```
Données : Un DFA D, une séquence de caractères S
Résultat : Une séquence de jetons J
debut = pos = 0 ; candidat = null ; E = départ(D);
Boucler
    c = caractère numéro pos de S (ou EOF sinon);
    pos++;
    E = successeur de E par la transition c (ou null sinon);
    si E == null alors
        si candidat == null alors retourner erreur lexicale;
        si candidat n'est pas ignoré alors ajouter candidat à J;
        si c == EOF alors retourner J;
        E = départ(D) ; pos = debut = caractère après candidat;
        candidat = null;
    sinon si E accepte jeton j alors
        candidat = new Jeton(j, debut, pos-1);
    fin
fin
```

lexèmes

Tant que pas d'erreur
et c <> de EOF