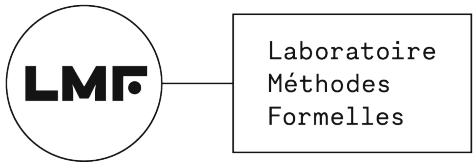

Introduction à la compilation

Polytech'Paris-Saclay– 4ème année –

Analyse Lexique

Burkhart Wolff (& Frederic Voisin)

Rôle d'un Analyseur Lexical

- Découper le texte du programme en une séquence d'unités significatives (« token ») :

  Identificateurs

  Mots-clef

  Constantes littérales (plusieurs sous catégories)

  Opérateurs (plusieurs sous-catégories)

  Symboles de « ponctuation »: `(` `)` `{` `}` `:` `,` `;` *etc…*

  Délimiteurs: espaces, tabulations, passage à la ligne

  Commentaires

  …

- **Le choix des tokens se fait quand on écrit la grammaire**

Rappel sur les tokens

Il existe deux catégories de tokens:

- Tokens « fixes », sans valeur associée: mots-clef, opérateurs arithmétiques, symbole de ponctuation
- Token « génériques » dont la « valeur » est liée au texte du programme correspondant à cette occurrence du token
  - Identificateur: s'agit-il de `x`, de `y`, de `fact` ?
  - Constante littérale entière: quelle valeur ?
  - `OpComp`: quel opérateur de comparaison ?

L'analyseur lexical fournit le « type » et la « valeur » du token.
Dans la grammaire seule la partie « type » apparaît

- Certains tokens ne sont pas transmis à l'analyseur syntaxique

Les autres rôles d'un analyseur lexical…

- Suivi de la position dans le texte, pour les messages d'erreur
  Syntax error on line **22**, colum **30**
- Fourniture d'un listing
- Traitement des « commentaires structurés » ou des « annotations/ attributs » en C/C++/C#
  Exemple : Javadoc
```
/**
 * @param
 * @return
 */
```
- Insertion des identificateurs dans une table des symboles, construction de « références croisées »
- … tout traitement qu'on peut faire le traitement au niveau des caractères ou au moins sans « contexte »

PRINCIPES DE REALISATION D'UN ANALYSEUR LEXICAL

1. On décrit par des expressions régulières les unités lexicales
Des abréviations permettent de simplifier l'écriture de ces expressions

+ une unité particulière pour la reconnaissance de la fin du texte (« `eof` »)
+ une unité pour les symboles non valides (« `illegal symbol at line` ... »)

2. On construit l'AFN qui reconnaît **l'union des langages** dénotés par les expressions régulières.
3. On déterminise et minimise l'automate
4. On associe à chaque expression régulière un fragment de code à exécuter
quand une instance de l'expression est reconnue
5. **Le moteur de l'analyseur lexical est une fonction « générique »**

- prend en entrée la position courante dans le texte,
- cherche une instance d'une expression régulière à partir de la position courante
- exécute le fragment de code associé à l'expression régulière reconnue. Ce fragment de code peut notamment renvoyer le token reconnu

La fonction d'analyse lexicale est appelée à la demande par l'analyseur syntaxique

Là où ça diffère des automates finis (AFD) :

1. Les états de satisfaction ne sont pas tous équivalents:

→ pour la minimisation, on distingue les états de satisfaction qui ne reconnaissent pas la **même** unité lexicale.

2. Un **AFD** reconnait **tout** le mot et décide s'il appartient au langage

Un **analyseur lexical partitionne le mot en sous-mots** dont chacun appartient au langage défini par une des expressions régulières.

Il y a en général **plusieurs** découpages possibles

Résolution des ambiguïtés de découpage :

- On privilégie la reconnaissance par **l'expression qui reconnaît le mot le plus long possible**
- **À longueurs égales**, on reconnaît l'**expression placée en premier** dans le fichier de description des expressions régulières.

Principes de la fonction d'analyse lexicale

À chaque nouvel appel de l'analyseur lexical, à partir de la position courante dans le texte:

- On se place dans l'état initial de l'automate
- On transite dans l'automate jusqu'à blocage, **en mémorisant le dernier état de satisfaction rencontré et la position associée dans le texte**.

- *Une fois bloqué:*

  - Si on n'est jamais passé par un état de satisfaction: `error` ou `eof`

  - Sinon on renvoie l'unité associée à l'état de satisfaction mémorisé et on se repositionne à la position associée dans le texte. Commentaire et délimiteurs : on ne le renvoie pas mais on recherche le prochain token

On peut parfois être allé au-delà de ce qui sera l'unité renvoyée !

Les difficultés de l'analyse lexicale

- Si l'analyseur lexical est incorrect, la suite de la compilation n'a plus de sens
- C'est le seul composant qui traite **tous** les caractères
- Si vous en écrivez un « à la main »
  - Problème d'efficacité: lectures efficaces dans un fichier, gestion de buffers, de débordement, de retours en arrière
  - Représentation efficace de (très) gros automates

En général on utilise un « générateur d'analyseur lexical »

- Pas de « contexte » disponible =>
  - Peu d'erreurs détectables à l'analyse lexicale
  - Pas de distinction entre différentes catégories d'identificateurs
- Avoir un traitement correct de erreurs lexicales demande un effort particulier