THÉORIE DES LANGAGES : ANALYSE LEXICALE

L3 INFORMATIQUE – 2020/21

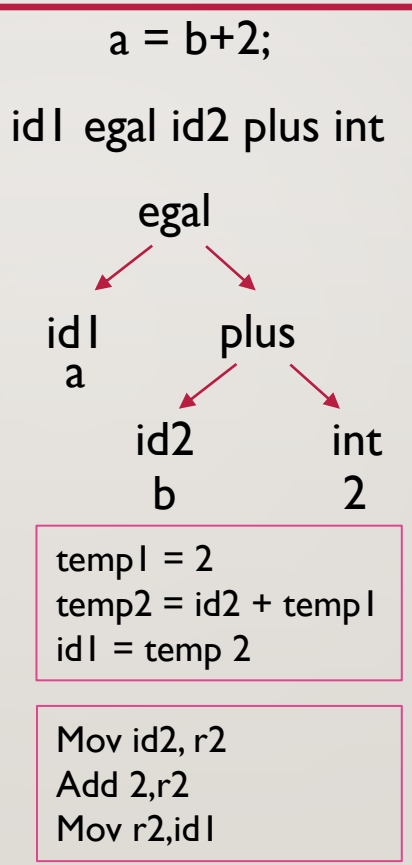
Eric Andres
eric.andres@univ-poitiers.fr

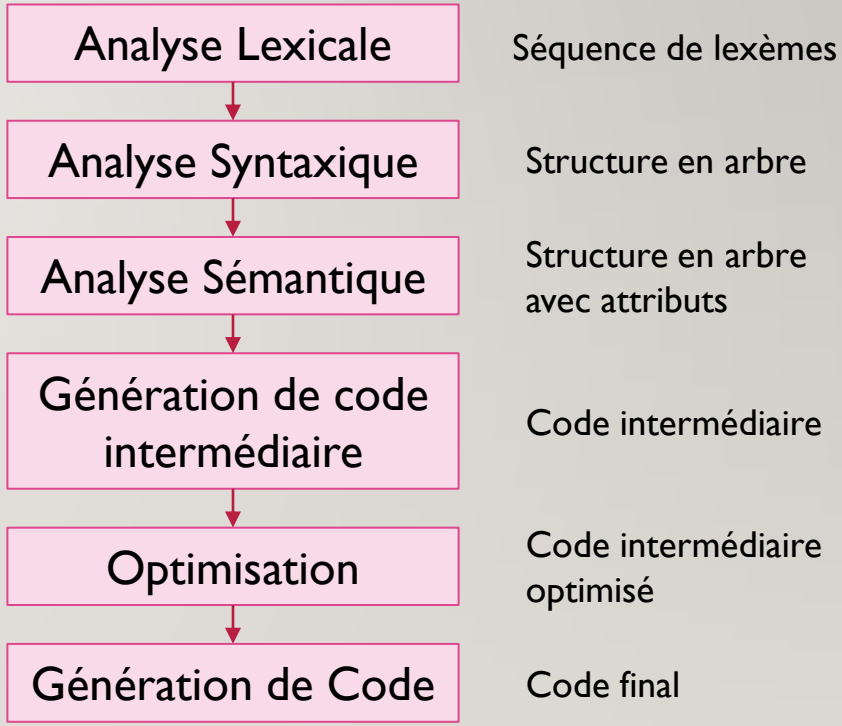
RÔLE D'UN COMPILATEUR

- Le but d'un compilateur est de traduire des programmes écrits dans un langage source en un langage destination
- Souvent, le source est un langage de programmation de haut et la destination est en langage machine ou un code intermédiaire plus ou moins universel.
- Parfois, traduction de source en source : langage à langage ou manipulation de données en XML par exemple.
- Le compilateur doit détecter si le source est conforme aux règles de programmation

ÉTAPES DE COMPILATION

- Un compilateur opère généralement en phases, chacune d'elles transformant le programme source d'une représentation en une autre
- Les phases ne sont pas toujours clairement définies
- Certaines phases peuvent être absentes





ANALYSE LEXICALE

- Parcourt le programme source de gauche à droite, caractère par caractère
- Ignore les caractères superflus (espaces, tab, Retour Chariots, commentaires)
- Convertit les unités lexicales en tokens ({, id, =, ==, abstract, int ...)
- Envoie ces informations à l'analyse syntaxique
- Parfois, on en profite pour construire la table des symboles : structure de données contenant un enregistrement pour chaque identifiant

ANALYSE SYNTAXIQUE

- Vérifie si la séquence d'unités lexicales fournie par l'analyse lexicale respecte les « règles » syntaxiques du langage : vérification de la GRAMMAIRE

ANALYSE SÉMANTIQUE

- Contrôle si le programme est sémantiquement correct (contrôle des types, portée des identificateurs, correspondance entre paramètres formels et paramètres effectifs)
- Collecte des informations pour la production de code (dans la table des symboles)

EN COMPLÉMENT DU COMPILATEUR

- Les préprocesseurs produisent ce qui sera l'entrée d'un compilateur : inclusion de fichiers, macro-expansion

- L'Éditeur de liens (chargement et relieur)
  le chargement consiste à prendre du code machine *translatable* et à modifier les adresses *translatables*
  le relieur constitue un unique programme à partir de plusieurs fichiers contenant du code machine translatable (compilation séparée ou bibliothèque)

EN COMPLÉMENT DU COMPILATEUR

- Les interpréteurs s'arrêtent à la phase de production de code intermédiaire
- L'exécution du programme source consiste alors en une interprétation/exécution du code intermédiaire par un programme appelé interpréteur

- Avantages : portabilité et exécution immédiate
- Désavantage : exécution plus lente qu'avec un compilateur complet

EN COMPLÉMENT DU COMPILATEUR

- Le compilateur ne produit pas de code pour un processeur réel, mais pour un processeur d'une machine virtuelle
- Portabilité : pour une nouvelle architecture, il n'y a pas besoin de modifier le compilateur, il suffit à priori de porter le programme qui implémente la machine virtuelle
- Principe de JAVA : « Compile once, run everywhere »
- Un peu exagéré car dans le cas de Java, l'environnement ne se compose pas seulement d'un processeur, mais aussi de nombreuses librairies

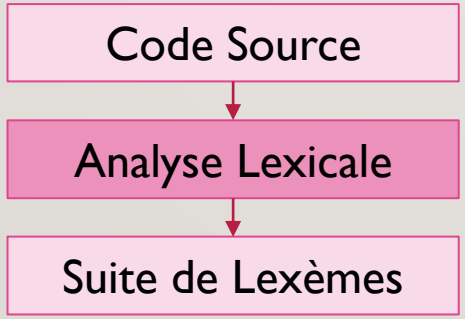
ANALYSE LEXICALE

C'EST DE ÇA QU'ON VA PARLER EN L3 INFORMATIQUE – 2020/21

On parle de théorie des langages != linguistique

INTRODUCTION

- L'analyse lexicale se trouve au tout début de la chaîne de compilation
- Elle collabore avec l'analyse syntaxique pour analyser la structure du langage
- Elle transforme une suite de caractères en une suite de mots, dit aussi lexèmes (tokens)
- L'analyse lexicale utilise essentiellement des automates pour reconnaître des expressions régulières



INTRODUCTION

- L'analyse lexicale peut également effectuer un certain nombre de tâches secondaires :
- Élimination des caractères superflus (espaces, tabulations, Retour Chariots, commentaires, ...)
- Compter le nombre de Retour Chariots afin de repérer le numéro de la ligne si une erreur est détectée
- Peut servir à d'autres tâches comme dans des parseurs ou dans Unix, les fonctions sed et grep par exemple.

NOTIONS DE BASE

- Un alphabet Σ est un ensemble (en général fini) de lettres $\Sigma = \{a,b,c, \ldots z\}$ $\Sigma = \{0,1,2,\ldots,9\}$
- Un mot (sur Σ) est une suite finie de lettres de Σ. On note ε est le mot vide !
- Un langage est un ensemble de mots.

L = {vis, clou, marteau, tournevis, écrou} : ce qu'il y a dans mon atelier;
L = {ε, a, b, aa, ab, ba, bb, aaa, aab, aba, abb,baa, …} : tous les mots formés de a et de b (y compris aucun).

Langage ici != Langage de Programmation

Un langage de Programmation est formé de mots acceptables (mots réservés, identifiants bien formés, entiers, réels, signes mathématiques, ponctuations, séparations, ….) : ça forme la syntaxe
+ une grammaire : qui vérifie que le « if » se termine par un « end if » par exemple

NOTIONS DE BASE

- Notations : on note Σ^* l'ensemble de tous les mots (y compris ε) qui peuvent être écrit sur Σ
- $\Sigma^* = \bigcup_{i=0}^{n} \Sigma^i$: étoile de Σ
- $\Sigma^+ = \bigcup_{i=1}^{n} \Sigma^i$: étoile propre de Σ

Notez que quelque soit le langage L sur Σ, on a $L \subseteq \Sigma^*$

- Def : Longueur d'un mot w = nombre de lettres du mot w : noté |w|

NOTIONS DE BASE : LA CONCATÉNATION

- En langage naturel, on construit certains mots en assemblant deux mots
  - tire, bouchon : tirebouchon
  - super, marché : supermarché
- L'opération de concaténation généralise ce procédé : $\begin{array}{c} \Sigma^* \times \Sigma^* \to \Sigma^* \\ (u, v) \mapsto u \cdot v = uv \end{array}$
- Un mot est une concaténation de lettres $w = w_1 w_2 w_3 \ldots w_n$

NOTIONS DE BASE : LA CONCATÉNATION

- L'opération de concaténation : $\Sigma^* \times \Sigma^* \to \Sigma^*$, $(u, v) \mapsto u \cdot v = uv$
- Propriétés :
  - Opération interne à Σ^*
  - Associative : u(vw) = (uv)w

Opération non associative : ^
(2^3)^4 = 8^4 = 4096
2^(3^4) = 2^81 = beaucoup = 2417851639229258349412352

  - Pas commutatif : en général u.v != v.u

Exemple d'opérations non commutatives : se verser une bière
- Ouvrir la bouteille *suivi de* se verser à boire
  != se verser à boire *suivi de* Ouvrir la bouteille

  - ε le mot vide est neutre pour la concaténation : u . ε = ε . u = u

NOTIONS DE BASE : MONOÏDE

- Def : Une application $\circ : E \times E \rightarrow E$ est une loi de composition interne ou opération sur E si pour tout $x, y \in E, x \circ y \in E$
  - Exemple : addition sur les entiers, concaténation sur les mots.
- Def : Une opération $\circ$ est associative si $x \circ (y \circ z) = (x \circ y) \circ z$, pour tout $x,y,z \in E$
- Def : Une opération $\circ$ un élément neutre 0 si pour tout $x \in E , x \circ 0 = 0 \circ x = x$
- Def : Soit un ensemble E muni d'une opération $\circ$ interne , associative et ayant un élément neutre pour cette opération, alors $< E, \circ >$ est un monoïde.
- Exemple : $< N, + >$ et $< \Sigma^* , \cdot >$ sont des monoïdes.

NOTIONS DE BASE : MONOÏDE

- Def : Toute partie F c E tel que $0 \in F$ et que pour tout $x,y, \in F, x \circ y \in F$,

  alors F est un sous-monoïde de E

- Exemple : $< N, + >$ est un sous-monoïde de $< Z, + >$

- Remarque : l'intersection de deux sous-monoïdes est un sous-monoïde.

NOTIONS DE BASE : MORPHISME

- Def : Soient deux monoïdes <E, +> et <F, *>

  Une application R : E→F est un morphisme si pour tout x,y ∈ E,

  Nous avons R(x+y) = R(x) * R(y)

  - Si de plus $R(0_E) = 0_F$ alors nous avons un morphisme de monoïde.
  - Un morphisme bijectif est appelé isomorphisme.

- Informellement, un morphisme est une transformation des éléments d'un monoïde qui opère « lettre à lettre ».
- Exemple : longueur d'un mot est un morphisme de $< \Sigma^* , \cdot >$ vers < N,+ >
  - Preuve : |u . v| = |u| + |v|…

NOTIONS DE BASE : SOUS-MOT, FACTEUR

- Def : Un **sous-mot** de u est composé d'une sous-suite de lettres de u, non forcément consécutives mais dans le même ordre.
  - Exemple : **aa** est un sous-mot de **abcdab** ; **dc** n'est pas un sous-mot de **abcdab**

- Def : Un **facteur** de u est une sous-suite de lettres consécutives et dans le même ordre.
  - Exemple : **aa** est un sous-mot de **abcdab** mais pas un facteur; **bcd** est un facteur de **abcdab**

- Def : Le **facteur gauche** (resp. droit) est un facteur commençant à la première lettre (resp. se terminant à la dernière lettre).
  - Exemple : **abc** est facteur Gauche de **abcdab**; **dab** est un facteur Droit de **abcdab**

NOTIONS DE BASE : OPÉRATION SUR LES LANGAGES

- **Opérations ensemblistes** : les langages sont des ensembles de mots : $A \cup B, A \cap B, A - B$
- **Produit induit par la concaténation** : $A \bullet B = \{ u \bullet v : u \in A \text{ et } v \in B \}$
  - Exemple : Calculons $A \bullet B$ pour $A=\{\varepsilon,ab\}$ et $B=\{b,bb\}$: $A \bullet B = \{\varepsilon b, \varepsilon bb, abb, abbb\} = \{b, bb, abb, abbb\}$
- **Etoile** : Quelque soit le langage L, on va noter $L^0=\{\varepsilon\}$; $L^1=L$; $L^2=L \bullet L$; $L^n = L^{n-1} \bullet L$ alors $L^* = \bigcup_{i=0}^{\infty} L^i$
- **Quelques autres opérations :**
  - **Def : Résiduel à Gauche : Soit u** un mot et **L** un langage **L** $\in \Sigma^*$, on appelle résiduel à gauche de L par u, noté $u^{-1}L$, le langage $u^{-1}L = \{v : uv \in L\}$
  - exemple : C'est le langage où l'on déjà reconnu une première partie : $L=\{ab, bc, a, abb, bb\}$

$a^{-1}L = \{b, \varepsilon, bb\}$; $(abc)^{-1}L = \varnothing$

NOTIONS DE BASE : ORDRES

- Def : **Ordre alphabétique :**
  - On se définit un ordre sur les lettres : a < b < c … < z
  - On compare lettre à lettre jusqu'à trouver une lettre différente.
  - Si jamais on arrive au bout d'un des mots, le plus court est l'inférieur.
- Exemple : ε < **ami** < **amitié** < **amour**
- Avantage : vous le connaissiez bien et pratique pour chercher un mot.
- Inconvénient : **aba** a une infinité de mots plus petits : par exemple **a, aa, aaa, aaaa, aaaaa**, ….

  Si on veut numéroter des mots ça peut être très embêtant.

NOTIONS DE BASE : ORDRES

- Def : **Ordre hiérarchique :**
  - On se définit un ordre sur les lettres : a < b < c … < z
  - Quand on veut comparer deux mots u et v, si les mots sont de longueur différentes, le mot le plus court est le plus petit sinon on applique l'ordre alphabétique.
- Exemple : Pour Σ = {a,b,c}, on a ε < **a** < **b** < **c** < **aa** < **ab** < **ac** < **ba** < **bb** < **bc** < **aaa** < …
- Remarques :
  - ε reste le mot le plus petit dans tous les ordres.
  - C'est un ordre total
  - On a un nombre fini de mots qui précèdent un mot donné. On peut donc les numéroter et les représenter sous forme d'arbres.

LANGAGES RATIONNELS ET LANGAGES RECONNAISSABLES

- Def : Σ un alphabet, on appelle **classe des langages rationnels** $\mathrm{Rat}(\Sigma)$ l'ensemble des langages vérifiant :
  - Tout sous-ensemble fini de Σ^* est dans $\mathrm{Rat}(\Sigma)$
  - Pour $L1 \in \mathrm{Rat}(\Sigma)$ et $L2 \in \mathrm{Rat}(\Sigma)$ alors
    - $L1 \cup L2 \in \mathrm{Rat}(\Sigma)$
    - $L1 \bullet L2 \in \mathrm{Rat}(\Sigma)$
    - $L1^* \in \mathrm{Rat}(\Sigma)$ (et bien sûr $L2^*$ aussi)
  - $\mathrm{Rat}(\Sigma)$ est le plus petit ensemble de langages possibles vérifiant cela.

LANGAGES RATIONNELS ET LANGAGES RECONNAISSABLES

- Def : Expression régulière

  La définition d'un langage rationnel en

  - Un ensemble fini de mots (y compris ε)
  - Concaténation
  - Union
  - Etoile

  Et donc nécessairement fini. Une telle expression s'appelle **« expression régulière »**

LANGAGES RATIONNELS ET LANGAGES RECONNAISSABLES

Propriété : Un langage est rationnel ssi il existe une expression rationnel le définissant

exemple :

- **ab*** : attention, c'est un abus : on devrait écrire **{a}.{b}*** = {a, ab, abb, abbb, …}
- **(ab)*** = {ε, ab, abab, ababab, … }
- **a+ba*** = {a}u {b}{a}* = {a, b, ba, baa, baaa, …}

aa* = a+ : Deux expressions rationnelles peuvent être différentes et représenter le même langage !

LANGAGES RATIONNELS ET LANGAGES RECONNAISSABLES

Propriété :

- Si $L \in Rat(\Sigma)$ et si $R: \Sigma^* \to \Pi^*$ est un mophisme de monoïde

  alors $R(L) \in Rat(\Pi)$

- L'image miroir d'un langage rationnel est aussi rationnel (on verra la preuve en TD)