Théorie des langages et Compilation

Pr. Nora El AMRANI
EMSI 2024/2025

October 18, 2024

Langues et langages ?

Introduction

Langues et langages ?:
Supports de communication.

Introduction

Langues et langages ?:
Supports de communication.

- **Langues:**

Introduction

Langues et langages ?:
Supports de communication.

- **Langues:** supports de communication permettant la communication entres les humains.
- **Langages:**

Introduction

Langues et langages ?:
Supports de communication.

- **Langues:** supports de communication permettant la communication entres les humains.
- **Langages:** supports de communication permettant la communication homme - machine.

Introduction

Langues et langages ?:
Supports de communication.

- **Langues:** supports de communication permettant la communication entres les humains.
- **Langages:** supports de communication permettant la communication homme - machine.

L'idéal c'est d'utiliser un langage naturel (une langue) pour communiquer avec la machine, mais l'interprétation d'une phrase dépend du contexte : ambiguïté
Alors qu'un langage de programmation devra être non ambigu pour une exécution correcte des ordres.
D'où la création des langages Formels $\Longrightarrow$ Théorie des langages.

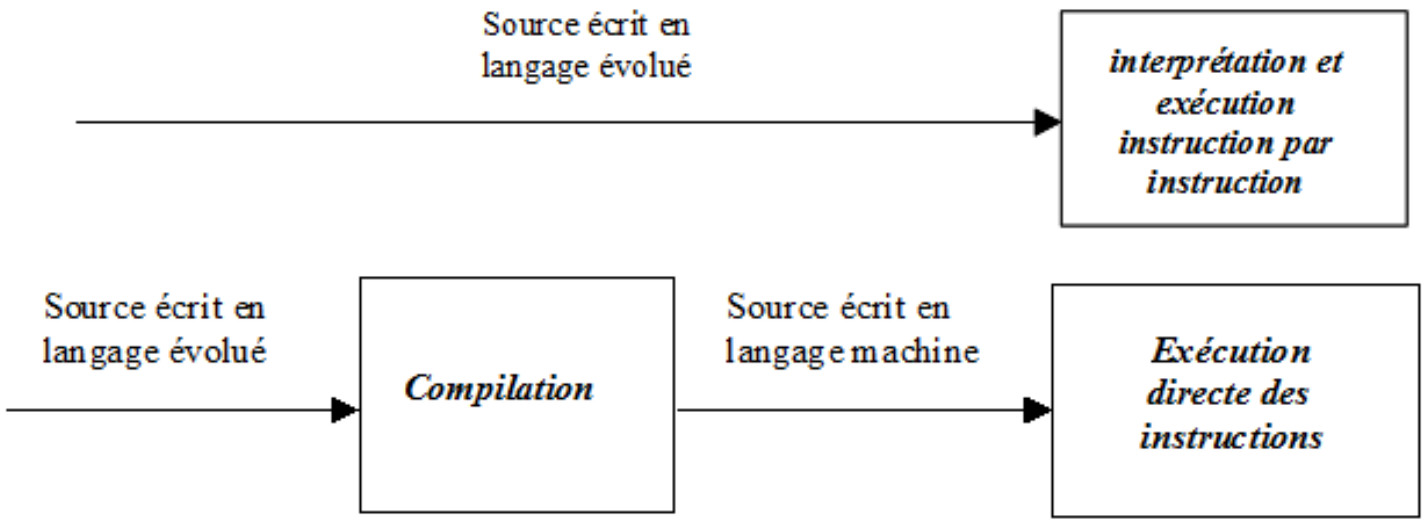
Introduction (suite)

Les langages(comme pour les langues naturelles) ont :

- Un alphabet.
- Des mots.
- Une grammaire.

Introduction(suite)

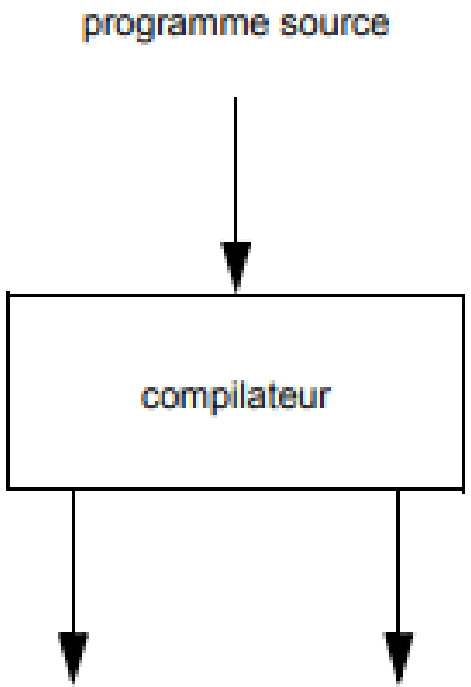
Un ordinateur ne comprend que son langage machine.
Il est nécessaire d'utiliser un interprète ou bien un compilateur.



Définition d'un compilateur et utilité

Définition:

C'est un programme particulier qui traduit un programme écrit en programme de haut niveau en instructions exécutables par machine. Autrement dit c'est un programme qui transforme un code source en un code objet.



Les taches d'analyse d'un compilateur

- Le premier Compilateur a été crée en 1954 (celui de Fortran).

Les taches d'analyse d'un compilateur

- L'analyse lexicale : reconnaitre les éléments constitutifs de la chaine entrée, et en dresser la liste.
- L'analyse syntaxique :Vérifier la conformité avec les règles de constitution du code.
- L'analyse sémantique : analyser le sens et fixer une interprétation.

Les taches d'analyse d'un compilateur

L'analyseur lexical

- Repérer les éléments significatifs appelés unités lexicales (mots-clés, constantes, identificateurs ...).
- Éliminer les superflus (commentaires, les espaces, les retours à la ligne)

Exemples d'unités lexicales :

- Identifiants : noms de variables ou de fonctions.
- Mot-clefs : mots réservés du langage.
- Opérateurs
- Ponctuations : points, virgules, parenthèse,....
- ...

Les taches d'analyse d'un compilateur

L'analyseur lexical

Exemple : à partir d'un morceau de C suivant :

$$\text{if } (j < i + 1) \text{ // test sur } j$$
$$a = a + b;$$

L'analyseur lexical déterminera la suite suivante des Token :
$< if, mot - cle >$; $< (, separateur >$; $< j, ident >$;
$<<, op - rel >$; $< i, ident >$; $< +, op - arith >$; $< 1, cste >$;
$<), separateur >$; $< a, ident >$; $<=, op - affect >$; $< a, ident >$;
$< b, ident >$; $<;, seprateur >$;

Les taches d'analyse d'un compilateur

L'analyse syntaxique:
La structuration de la suite des Token en catégorie $<< grammaticales >>$. Normalement on présente le résultat de cette structuration à l'aide d'un arbre appelé **arbre abstrait**, dont le but est de vérifier que les unités lexicales sont dans le bon ordre définis par le langage. L'analyseur lexical sait comment doivent être construite les expressions, les instructions, les déclaration de variables, ...
Exemple: Si l'analyseur de C reçoit ce morceau de code:
$< if, mc - if >$; $< j, ident >$; $<<, op - rel >$; $< i, ident >$;
$< +, op - arith >$;

Les taches d'analyse d'un compilateur

L'analyse syntaxique:
La structuration de la suite des Token en catégorie $<< grammaticales >>$. Normalement on présente le résultat de cette structuration à l'aide d'un arbre appelé **arbre abstrait**, dont le but est de vérifier que les unités lexicales sont dans le bon ordre définis par le langage. L'analyseur lexical sait comment doivent être construite les expressions, les instructions, les déclaration de variables, ...
Exemple: Si l'analyseur de C reçoit ce morceau de code:
$< if, mc - if >$; $< j, ident >$; $<<, op - rel >$; $< i, ident >$; $< +, op - arith >$; Il va directement détecter une erreur, parce que il sait qu'il doit y avoir une (juste après le if.

L'analyse sémantique

C'est une phase de contrôles, par exemple on vérifie que l'assemblage de constituants du programme a un sens.

Exemple:

On ne peut pas additionner un réel avec une chaine de caractère, ou affecter une variable à un nombre ....

Les codes source et objet doivent faire la même chose, ce qui nécessite une bonne et rigoureuse définition du langage objet et source

La définition d'un langage comporte :

1. **La syntaxe** : la forme des constructions autorisées.
2. **La sémantique** : le sens des instructions syntaxiquement correctes.

$\Downarrow$

Théorie des Langages

(Domaine des Mathématiques)

Langages

Définition:

- *Un alphabet : un ensemble fini $\sum$ de symboles.*
- *Un mot ou chaine : une suite v fini de symboles d'un alphabet $\sum$.*
- *Un langage : un ensemble L de mots construites sur un alphabet $\sum$*

Notations : Soient $\sum$ un alphabet, et $v \in \sum^*$.
On note par :

- $|\sum|$ le cardinal d'un alphabet $\sum$.
- $|v|$ la langueur de v,et par $|v|_a$ le nombre de symbole a dans v.
- ϵ le mot de langueur 0.
- $\sum^*$ l'ensemble de tous les mots que l'on peut construire sur un alphabet $\sum$.

- ϵ le mot vide, $|\epsilon| = 0$.
- 0^n: le mot composé de n occurrences de 0.
- Soit $x \in \sum$, $x^0 = \epsilon$.
- Soit $\sum$ un alphabet alors :
  - $\sum^1 = \sum$
  - $\sum^0 = \{\epsilon\}$
  - $\sum^* = \cup \sum^i \; i \geqslant 0$, si $\sum = \{a\}$ alors $\sum^* = \{\epsilon, a, aa, aaa, \ldots\}$.
  - $\sum^+ = \cup \sum^i \; i > 0$, $\sum^+ = \sum^* / \{\epsilon\}$, si $\sum = \{a\}$ alors $\sum^+ = \{a, aa, aaa, \ldots\}$.

Exemples :

- $\sum = \{a, b, z, !, 8, 1\}$ est un alphabet de taille 5.
- $L1 = \{8ab!, aba1z\}$ $L2 = \{ab!, abab, zabbb, aa\}$ sont deux langages finis sur $\sum$.
- $L3 = \{ab, abab, ababab, \ldots\}$ est un langage infini de $\sum$.
- $|L1| = 2$.
- $|\epsilon| = 0$
- $|8ab| = 3$.

Les opérations sur les mots

Opérations sur les mots :

- La concaténation : la concaténation de x et y de $\sum^*$, donne xy ou yx.
  On note que la concaténation est:
  - non commutative : $\forall x, y \in \sum^*$ si $x \neq y$ alors : $xy \neq yx$.
  - associative : $\forall x, y, z \in \sum^*$: $x(yz) = (xy)z$.
  - ϵ est l'élément neutre $\forall x \in \sum^*$: $x\epsilon = \epsilon x = x$..

Les opérations sur les mots

Préfixes et suffixes: Soit

$$v = v_1 v_2 \dots v_l$$

un mot sur un alphabet $\sum$, alors

$$\epsilon, v_1, v_1 v_2, \dots, v_1 \dots v_{l-1}, v_1 \dots v_l$$

sont les préfixes de v.
Un préfixe de v est dit propre si il est différent de ϵ et de v.
Aussi

$$\epsilon, v_l, v_{l-1} v_l, \dots, v_2 \dots v_l, v_1 \dots v_l$$

Sont les suffixes de v.
Un suffixe de v est dit propre si il est différent de ϵ et de v.

Opérations sur les mots

- On appelle occurence d'une lettre a dans un mot v son nombre d'apparition dans le mot v. Et on la note $|v|_a$.
  Exemple: Soit $\sum = \{1, 9, 6, i\}$, et le mot 96.
  - $|96| = 2$, $|96|_6 = 1$, $|96|_9 = 1$, $|96|_5 = 0$.
- *Le miroir* du mot $x = x_1 x_2 \dots x_n$ est défini par : $x^R = x_n x_{n-1} \dots x_2 x_1$.
  - $x = 69i$ alors $x^R = i96$
  - Propriété : $(xy)^R = y^R x^R$.
- x est un *facteur* du mot y s'il existe des mots u et v tels que : $y = uxv$.

Opérations sur les mots

Conjugaison des mots: on dit que deux mots x et y sont conjugués s'ils existent deux mots u et v de $\sum^*$ tel que $x = uv$ et $y = vu$.

Remarque:

x et y sont conjugués propre s'il existe deux mots u et v de $\sum^+$ tel que : $x = uv$ et $y = vu$.

Exemple:

Les conjugués propres du mot $x = 96i1$:

Opérations sur les mots

Conjugaison des mots: on dit que deux mots x et y sont conjugués s'ils existent deux mots u et v de $\sum^*$ tel que $x = uv$ et $y = vu$.

Remarque:

x et y sont conjugués propre s'il existe deux mots u et v de $\sum^+$ tel que : $x = uv$ et $y = vu$.

Exemple:

Les conjugués propres du mot $x = 96i1$:

- $y1 = 6i19$ avec $u = 9$ et $v = 6i1$
- $y2 = i196$ avec $u = 96$ et $v = i1$
- $y3 = 196i$ avec $u = 96i$ et $v = 1$

- Un mot v est primitif (non décomposable) si $v = y^n$ alors $n = 1$.
- Un mot v est périodique (décomposable) si $k > 1$ et y primitif tel que : $w = y^k$, y est dit le motif ou la période.

L'analyse lexicale

L'analyseur lexical constitue la première étape d'un compilateur. Sa tache principale est de lire les caractères d'entrée et de produire comme résultat une suite d'unités lexicales que l'analyseur syntaxique aura à traiter. En plus l'analyseur lexical doit :

- Éliminer les caractères superflus.
- gérer les numéro de lignes dans le programme source, pour pouvoir associer à chaque erreur rencontrée par la suite la ligne dans laquelle elle intervient.

L'analyse lexicale

Définition:

Une unité lexicale est une suite de caractères qui a une signification. collective.

Exemples:

- Les chaines $\leq$, $\geq$, $<$, $>$ sont des opérateurs rationnels, l'unité lexicale est OPREL.
- Les chaines *if* , *else*, *while* sont des mots clefs.

Définition:

Un modèle est une règle associée à une unité lexical qui décrit l'ensemble des chaines du programmes qui peuvent correspondre à cette unité lexicale.

L'analyse lexicale

Définition:

On appelle lexème toute suite de caractères du programme source qui concorde avec le modèle d'une unité lexicale.

Exemple:

- L'unité lexicale IDENT (identificateurs) en *C* à pour modèle : toute suite non vide de caractères composée de chiffres, lettres ou du symbole " $-$ " et qui ne commence pas par un chiffre.

Remarques:

- Pour décrire le modèle d'un unité lexicale, on utilisera des expressions régulières.
- On conviendra des priorités décroissantes suivantes : $*$, concaténation, $|$.

La concaténation est distributive par rapport à $|$. :$a(b|c) = ab|ac$ et $(a|b)c = ac|bc$.

Exercices : décrivez les ensembles des mots générés par les expressions régulières suivantes

- $(a|b)^* = (b|a)^*$:
- $(a)|((b)^*(c))$.
- $(a^*|b^*)^*$.
- $(a|b)^*abb(a|b)^*$.
- $b^*ab^*ab^*ab^*$
- $(abbc|baba)^+aa(cc|bb)^*$.

- $(a|b)^* = (b|a)^*$ dénote l'ensemble de tous les mots formés de a et de b, ou le mot vide.
- $(a)|((b)^*(c)) = a|b^*c$ est soit le mot a, soit les mots formés de 0 ou plusieurs b suivi d'un c. C'est à dire $\{a, c, bc, bbc, bbbc, bbbbc, \ldots\}$
- $(a^*|b^*)^* = (a|b)^* = ((\varepsilon|a)b^*)^*$
- $(a|b)^*abb(a|b)^*$ dénote l'ensemble des mots sur $\{a, b\}$ ayant le facteur abb
- $b^*ab^*ab^*ab^*$ dénote l'ensemble des mots sur $\{a, b\}$ ayant exactement 3 a
- $(abbc|baba)^+aa(cc|bb)*$

L'analyse lexicale

Nous disposant avec les E.R (Les expressions régulières), d'un mécanisme de base pour décrire les unités lexicales. Par exemple :

ident = (a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v |w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q |R|S|T|U |V|W|X|Y|Z|_) (a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r| s|t|u|v|w|x|y|z|A|B |C|D|E|F|G|H|I|J|K|L|M |N| O|P|Q| R|S|T|U|V|W|X|Y|Z|0|1|2|3|4|5|6|7|8|9|_)*

C'est un peu chiant et illisible . . .

Alors on s'autorise des définitions régulières.

Exemple:

L'unité lexicale *IDENT* (identificateurs) en *C* devient :

- $lettre = A - Z|a - z$
- $chiffre = 0 - 9$
- $sep = \_$
- $IDENT = (lettre|sep)(lettre|chiffre|sep)^*$

Les attributs

- Pour ce qui est des symboles $\leq$, $\geq$, $<$, $>$ l'analyseur syntaxique a juste besoin de savoir que cela correspond à l'unité lexicale *OPREL*. C'est seulement lors de la génération de code que l'on aura besoin de distinguer $<$ de $>$ (par exemple).
- Pour ce qui est des identificateurs, l'analyseur syntaxique a juste besoin de savoir que c'est l'unité lexical IDENT, mais le générateur de code lui aura besoin de l'adresse de la variable correspondant à cet identificateur. L'analyseur sémantique aura aussi besoin de type de la variable pour vérifier que les expressions sont sémantique-ment correctes.

Cet information supplémentaire, utile pour l'analyseur syntaxique mais utile pour les autres phase du compilateur, est appelé attribut.

L'analyse lexicale

Peu d'erreurs sont détectables au seul niveau lexical, car l'analyseur lexical a une vision très locale du programme source. Les erreurs se produisent lorsque l'analyseur est confronté à une suite de caractères qui ne correspond à aucun des modèles d'unité lexicale qu'il a à sa disposition. Alors plusieurs stratégies sont possibles:

- Mode panique: ignore les caractères qui posent problème et on continue.
- Transformation du texte source: insérer un caractère, remplacer, échanger, ...

La stratégie la plus simple est le mode panique. En fait, en général, cette technique se contente de refiler le problème à l'analyseur syntaxique. Mais c'est efficace puisque c'est lui, la plupart du temps, le plus apte à le résoudre.

Les Opérations sur les langages

Soient $L1$ et $L2$ deux langages respectivement définis sur les alphabets $\sum_1$ et $\sum_2$.

- Le complément $\overline{L1} = \{m/m \notin L1\} = \sum^* - L1$
- L'union $L1 \sqcup L2 = L1 + L2 = \{m \quad /m \in L1 \quad ou \quad m \in L2\}$.
- l'intersection $L1 \sqcap L2 = \{m \quad /m \in L1 \quad et \quad m \in L2\}$.
- La différence $L1 - L2 = \{m/m \in L1 \quad et \quad m \notin L2\}$.
- Concaténation $L1L2$. Exp :Pour $L1 = \{a, b\}, L2 = \{c, ef\}$ alors $L1L2 = \{ac, aef, ac, bef\}$
- Auto concaténation $\underbrace{L1 \ldots L1}_{m} = L1^m$
- Fermeture de Kleene $L1^* = \sqcup_{k \geq 0} L1^k$.

Description d'un langage

- Énumération des mots qui le composent:
  $L = \{\epsilon, 011, 00111, 1110\}$.(finis)
- L'application d'opérations à des langages plus simple(Description littéraire) : Ensemble des mots construits sur l'alphabet $\{0, 1\}$, commençant par une suite de zero et se terminant par une suite de 1, tel que si le nombre de zero et $n \in \mathbb{N}$ alors le nombre de 1 est $n + 1$. (infinis)
- Certain langages infinis peuvent être décrits par un ensemble de règles appelé grammaire (Grammaire de réécriture).
- Certains langages infinis ne peuvent pas être décrits, ni par l'application d'opérations, ni par un ensemble de règles. On parle alors de langage indécidable.

Ici on s'intéresse aux langages réguliers.

Langages réguliers

Définition:

Un langage L sur l'alphabet $\sum$ est régulier si on peut l'obtenir par récursivité :

- *à partir des seuls langages "de base" :*
  - *langage vide : $\emptyset$*
  - *langage $\{\epsilon\}$*
  - *langages de la forme $\{a\}$, avec $a \in \sum$*
- *à l'aide des seules opérations :*
  - *réunion*
  - *concaténation*
  - *étoile de Kleene.*

N.B: "Par récursivité" veut dire qu'on applique un nombre fini de fois une telles opérations, à partir des langages "de base".

Langages réguliers

Quelques langages réguliers:

- Pour tout mot $u \in \sum^*$, le langage $\{u\}$ est régulier :
  - Si u s'écrit $a_1 \dots a_n$ sur $\sum$, alors le langage $\{u\}$ s'écrit comme la concaténation $\{u\} = \{a_1\}\{a_2\} \dots \{a_n\}$.
  - $\{u\}$ est régulier car chaque $\{a_i\}$ est régulier et l'ensemble des langages réguliers est clos pour la concaténation.
- Tout langage fini est régulier.
  - Un ensemble fini de mots $L = \{u_1, \dots, u_k\}$ s'écrit :
    $$L = \{u_1\} \cup \{u_2\} \cup \dots \cup \{u_k\}.$$
  - L est régulier car chaque $\{u_i\}$ l'est et ensemble des langages régulier est clos pour l'union.

Expressions régulières

Les expressions régulières sur un alphabet permettent de décrire les langages réguliers, de façon plus simple qu'en utilisant des opérations ensemblistes.

Les expressions régulières

Définition:

Une expression E est une expression régulière sur un alphabet $\sum$ si et seulement si :

- $E = \emptyset$ *ou*
- $E = \epsilon$ *ou*
- $E = a$ *avec* $a \in \sum$ *ou*
- $E = E_1|E_2$. E_1 *et* E_2 *sont deux expressions régulières sur* $\sum$. *ou*
- $E = E_1.E_2$ *et* E_1 *et* E_2*sont deux expressions régulières sur* $\sum$. *ou*
- $E = E_1^*$. E_1 *est une expression régulière sur* $\sum$.

Langage décrit par une expression régulière

Définition:

Le langage $L(E)$ décrit par une expression régulière E définie sur un alphabet $\sum$ est défini par :

- $L(E) = \emptyset$ *si* $E = \emptyset$.
- $L(E) = \{\epsilon\}$ *si* $E = \epsilon$.
- $L(E) = \{a\}$ *si* $E = a$.
- $L(E) = L(E_1) \cup L(E_2)$ *si* $E = E_1|E_2$.
- $L(E) = L(E_1).L(E_2)$ *si* $E = E_1.E_2$.
- $L(E) = L(E_1)^*$ *si* $E = E_1^*$ *ou* E_1 *est une expression régulière sur* $\sum$

Les expressions régulières

Exemple:
Sur l'alphabet $\sum = \{a, b, c\}$:

- $E1 = a^*bbc^*$ décrit le langage
  $L(E1) = \{a^n bbc^p / n \geq 0, etp \geq 0\}$.
- $E2 = (a|b|c)^*(bb|cc)a^*$ décrit le langage
  $L(E1) = \{wbba^n, wcca^n / w \in \sum^*, n \geq 0\}$

Les expressions régulières

Theorem

Toute expression régulière décrit un langage régulier.

Theorem

Tout langage régulier peut être décrit par une expression régulière.

Donnez une expression régulière qui accepte chacun des langages suivants (définis sur l'alphabet $\Sigma = \{0, 1\}$) :

1. Toutes les chaînes qui se terminent par 00.
2. Toutes les chaînes dont le 10ème symbole, compté à partir de la fin de la chaîne, est un 1.
3. Tous les nombres binaires divisibles par 4.