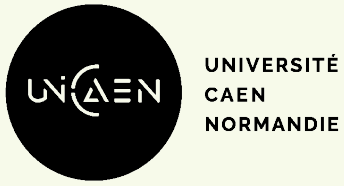
Compilation : présentation

[L3 Informatique] Théorie des langages et compilation

Gaétan Richard

2021-2022



La compilation

Définition

Compilateur :

En informatique, un compilateur est un programme qui **transforme un code source en un code objet**. (source : *Wikipedia*)

Exemples d'usage :

- Compilation d'un fichier `C` ;
- Transformation d'un fichier `python` en code `pyc` ;
- Analyse d'une requête `SQL` ;
- Colorisation syntaxique de code ;
- Analyse d'un fichier de configuration dans un format évolué ;
- ...

Langage de très haut niveau

Pour écrire un compilateur, on utilise souvent un langage particulier au-dessus d'un autre langage. On utilise alors des outils qui convertissent le code écrit en code de haut niveau.

Exemples :

- En `Java` : anltr ;
- En `C` : lex / yacc ;
- En `python` : antlr, ou autres ;
- ...

Vision de très haut niveau

La compilation est également l'occasion de regarder l'ensemble de la chaîne depuis le code source jusqu'à l'exécutable.

On en profite pour comprendre et être capable d'utiliser :

- des expressions régulières
- des grammaires
- les outils de linkages et les bibliothèques
- les failles binaires (comme le *buffer overflow*)

Le cours de compilation de L3 Informatique

Les choix

Outil : Antlr en `java`

Source : langage calculette avec fonctions (langage simplifié inspiré en parti de `C`)

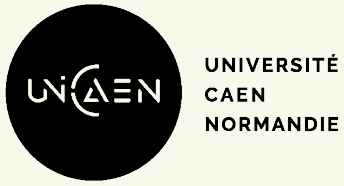
Destination : langage MVàP (langage assembleur simplifié mélangeant les inspirations du `i386`, du bytecode JAVA et d'autres)

Compilation : du source à l'éxécutable

[L3 Informatique] Théorie des langages et compilation

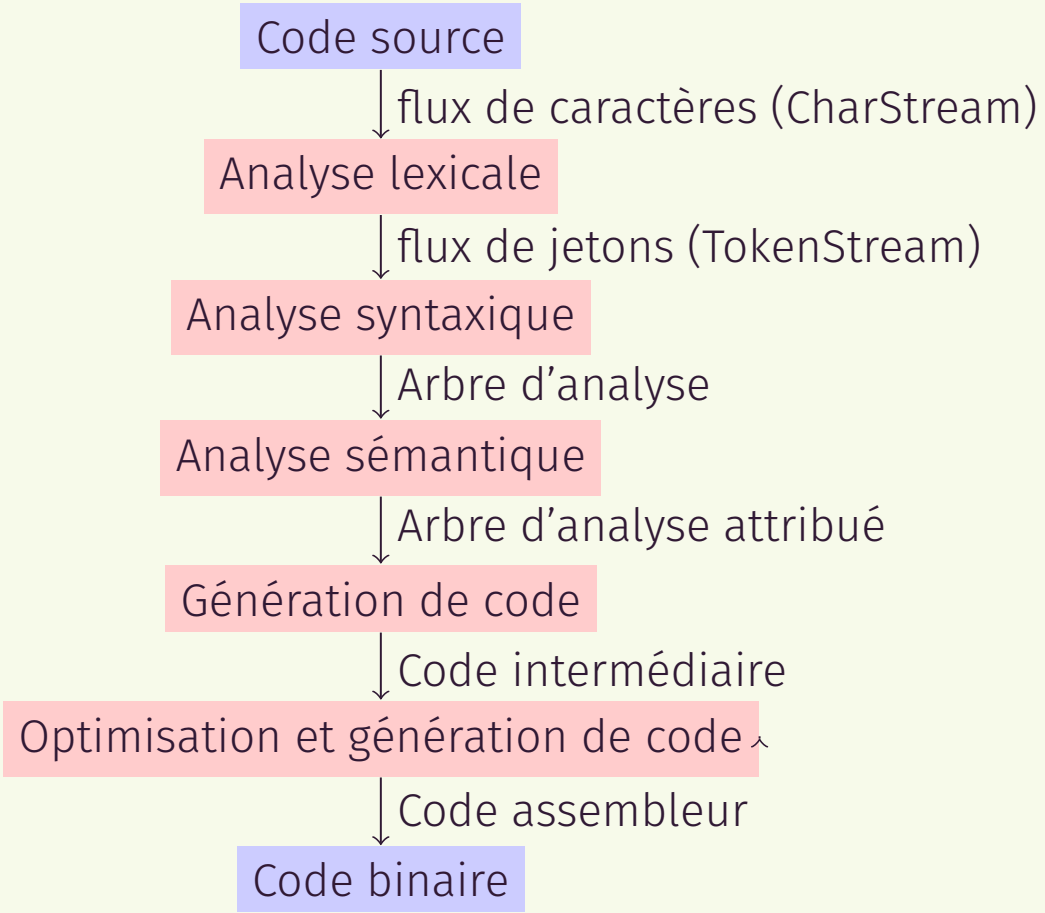
Gaétan Richard

2021-2022

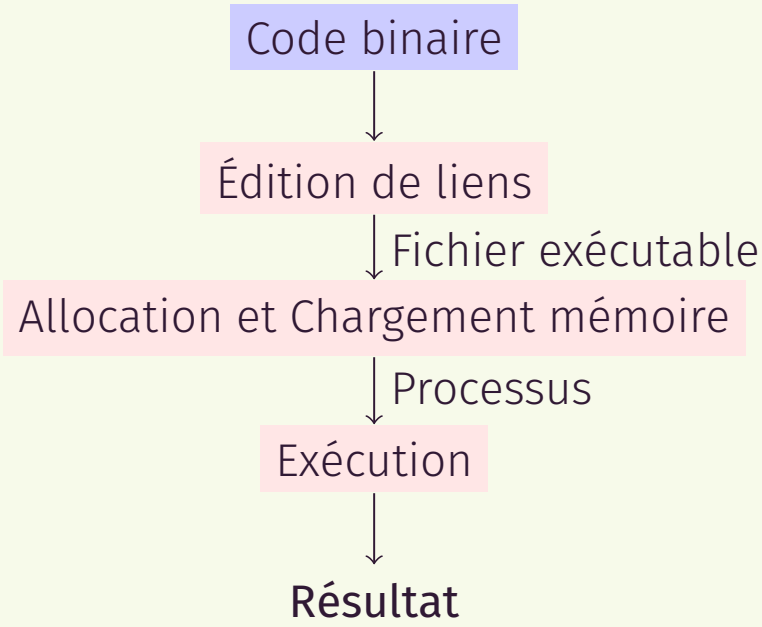


Vue d’ensemble

Les grandes étapes de la compilation ...

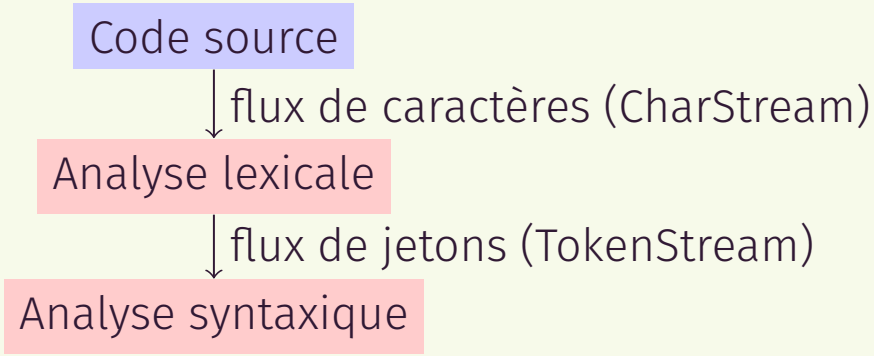


Jusqu’à l’exécution



Analyse lexicale

Présentation



Analyse lexicale : vocabulaire et définition

Lexème : chaîne de caractères correspondant à une unité élémentaire du texte.

Unité lexicale : classe ou type de lexèmes. *Exemples* : mot-clé, identifiant, nombre, opérateur arithmétique, ...

Jeton (token) : objet ayant :

- une unité lexicale ;
- un lexème ;
- un numéro de ligne (et de caractère) dans le source ;
- une valeur pour un nombre, adresse dans la table des symboles pour un identificateur ;
- ...

L'analyse lexicale converti un fichier d'entrée en un flux de jetons.

Fonctionnement de l'analyse lexicale

La définition des jetons se fait en général par des **expressions régulières**.

L'analyseur lexical est un **automate fini** avec des actions qui permet de découper les jetons.

Gestion des ambiguïtés

- 1 lexème satisfait 2 expressions
- 1 lexème est le préfixe d'un autre

Gestion des ambiguïtés

1 lexème pour 2 expressions

Par exemple, si un mot est à la fois identifié comme un `mot clé` et comme un `identifiant`.

Gestion via des priorités (en général, ordre d'apparition dans le code) du lexeur.

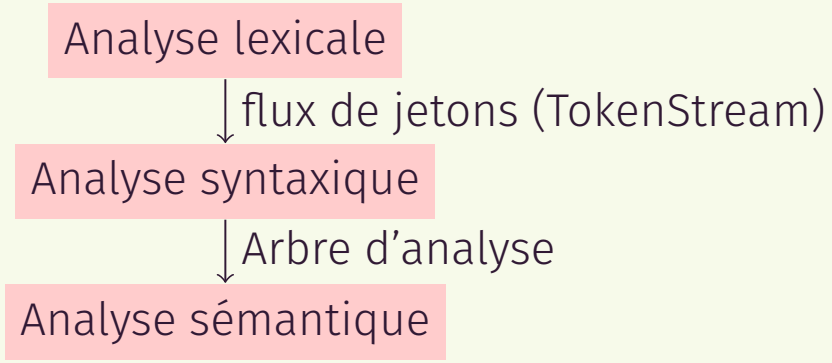
1 lexème est le préfixe d'1 autre

Par exemple, si un préfixe d'un identifiant est un mot clef.

Par défaut une **approche gloutonne** (*greedy*), on conserve la plus longue correspondance. Il existe aussi souvent une notation pour prendre la plus petite correspondance (`*?`). Ceci est très utile pour les commentaires et les chaînes de caractères.

Analyse syntaxique

Présentation



Analyse syntaxique : vocabulaire et définition

Une grammaire donne la syntaxe des mots admissibles.

L'arbre d'analyse a pour nœuds des non-terminaux et pour feuilles des terminaux (ce sont les jetons du lexeur).

L'arbre de syntaxe abstraite (AST) a pour nœuds des opérations et pour feuilles les opérandes.

Principe : l'analyse syntaxique transforme un flux de jetons en arbre d'analyse ou en AST.

Fonctionnement de l'analyse syntaxique

Cas général : il est **impossible** de faire un analyseur syntaxique qui transforme **efficacement** un flux de jetons en arbre d'analyse pour n'importe quelle grammaire.

Mais pour des **grammaires particulières**, on peut **garantir d'être efficace**. `Antlr` offre cette garantie pour les **grammaires LL*** .

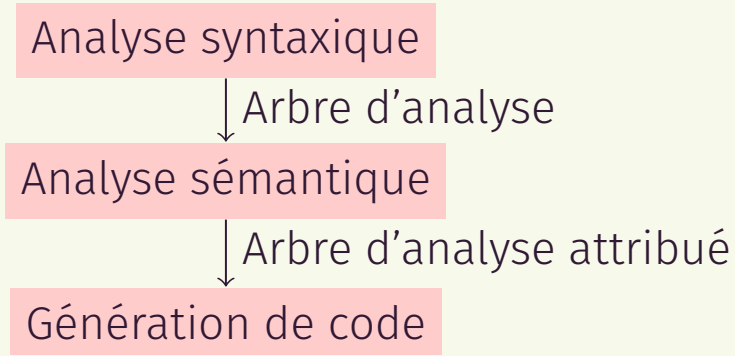
grosso modo, on ne fait pas de backtrack lorsqu'on cherche une règle : il suffit de regarder à distance k vers l'avant pour être fixé sur la règle qu'on doit applique

Travail dans l'analyse syntaxique

- gérer les ambiguïtés : s'il existe deux arbres de dérivations correspondant à une expression, on utilise des informations de **priorité** ou d'**associativité**.
- Gérer l'incorrect : si la grammaire n'est pas dans la bonne forme, il existe un certain nombre de mécanismes.

Analyse sémantique

Présentation



Fonctionnement

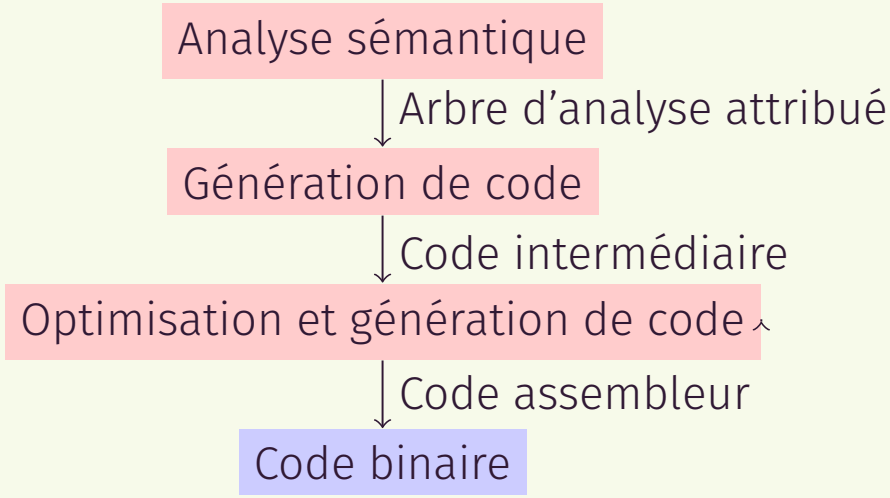
Objectif : Donner un sens à l'arbre de dérivation.

Méthode : ajouts d'**annotations** dans l'arbre calculées avec des règles locales.

Informations :

- vérification des types ;
- résolution des noms (via une **table des symboles**) ;
- affectation ;
- ...

Génération du code


Analyse sémantique
Arbre d'analyse attribué
Génération de code
Code intermédiaire
Optimisation et génération de code
Code assembleur
Code binaire

Fonctionnement

Objectif : passer de l'AST à du code machine.

Représentation intermédiaire : on utilise une représentation intermédiaire avant le code machine binaire

Avantages :

- Indépendance de la machine physique ;
- Phase d'optimisation plus facile.

Exemple : code 3 adresses.

Mise en place

Si on suppose que l'AST reflète la priorité et l'associativité des opérateurs :

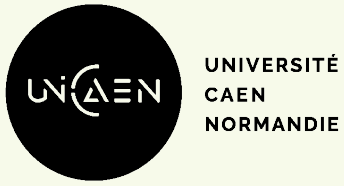
- il suffit de parcourir l'arbre ;
- affecter un nouveau registre pour chaque résultat d'opération ;
- à chaque nœud, on récupère le code machine et le registre contenant le résultat ;
- on obtient le code complet par un parcours postfixe (Gauche - Droite - Racine).

Compilation : MVàP

[L3 Informatique] Théorie des langages et compilation

Gaétan Richard

2021-2022



Le code à Pile

Code à Pile

Instructions typiques :

Action	Instructions
Ajouter sur la pile	`PUSH a`
Addition	`ADD`
Stockage	`STORE x`

Avantages :

- Forme compacte ;
- Pas de registre à nommer ;
- Simple à produire, simple à exécuter

Inconvénients :

- Les processeurs opèrent sur des registres, pas des piles ;
- Il est difficile de réutiliser les valeurs stockées dans la pile.

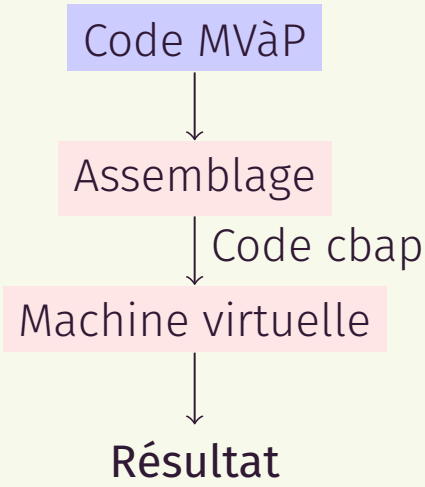
Exemple de code

Exemple : $x \leftarrow (a - b) * (c + d)$

```
PUSH a
PUSH b
SUB
PUSH c
PUSH d
ADD
MULT
STORE x
```

La machine virtuelle à Pile

Principe


Code MVàP
Assemblage
Code cbap
Machine virtuelle
Résultat

Code MVàP

Code source : `test.mvap`

```
PUSHI 5
PUSHI 8
MUL
PUSHI 2
PUSHI 1
MUL
ADD
WRITE
HALT
```

Code assemblé : `test.mvap.cbap`

```
 Adr |  Instruction
 ----+--------------
   0 | PUSHI 5
   2 | PUSHI 8
   4 | MUL
   5 | PUSHI 2
   7 | PUSHI 1
   9 | MUL
  10 | ADD
  11 | WRITE
  12 | HALT
```

```
java -cp "MVaP.jar" MVaPAssembler test.mvap
```

Exécution

```
pc |                |    fp   pile
 0 | PUSHI        5 |     0 [ ] 0
 2 | PUSHI        8 |     0 [ 5 ] 1
 4 | MUL            |     0 [ 5 8 ] 2
 5 | PUSHI        2 |     0 [ 40 ] 1
 7 | PUSHI        1 |     0 [ 40 2 ] 2
 9 | MUL            |     0 [ 40 2 1 ] 3
10 | ADD            |     0 [ 40 2 ] 2
11 | WRITE          |     0 [ 42 ] 1
   42
12 | HALT           |     0 [ 42 ] 1
```

Fonctionnement

Contenu :

- Trois registres spéciaux pc , sp et fp ;
- Un segment de code ;
- Une pile.

Généralités :

- La mémoire est organisée en mots, on y accède par une adresse qui est représentée par un entier.
- Les valeurs simples sont stockées dans une unité de la mémoire.
- Une partie de la mémoire est réservée aux instructions du programme
- Un registre stocke l'adresse de l'instruction en cours d'exécution pc (*program Counter*)
- Le code s'exécute de manière séquentielle sauf instruction explicite de saut
- Un registre stocke l'adresse de la première cellule libre de la pile (sommet de pile) sp (*Stack Pointer*)
- Les variables locales sont stockées dans la pile P

Restrictions

Spécificités :

- Pas de tas ;
- pas de registres ;
- une pile contenant déjà les opérations de bases ;
- un espace dédié et séparé pour le code.

Limitations :

- Très difficile de faire l'allocation de taille inconnue à la compilation ;
- impossible de modifier le code à la volée.