Théorie des Langages et Compilation

Cours 5 : La Compilation

Plan du cours

I. Qu'est ce qu'un compilateur

II. Les phases de la compilation

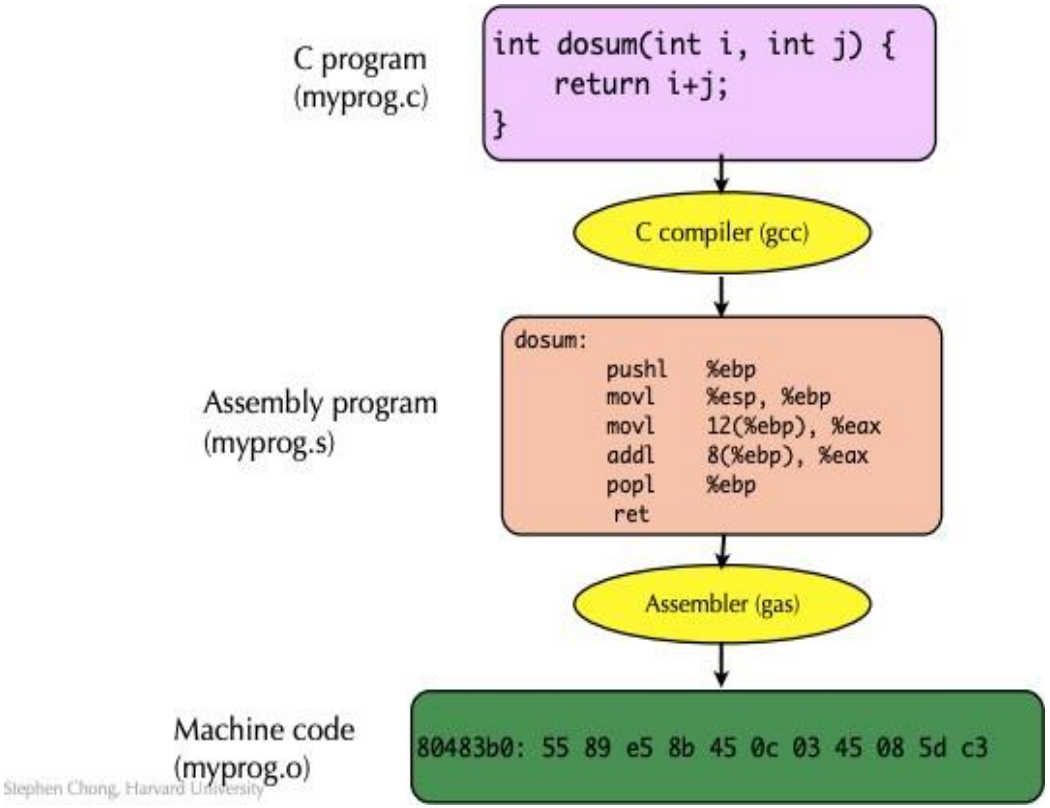
1. L'analyse

2. La génération du code

III. Activité pratique

Introduction

Les langages crées par l'homme pour communiquer avec les ordinateurs sont des langages artificiels compréhensible par l'être humain . Ils doivent être traduits pour être compréhensible par une machine. D'ou le rôle des compilateurs.



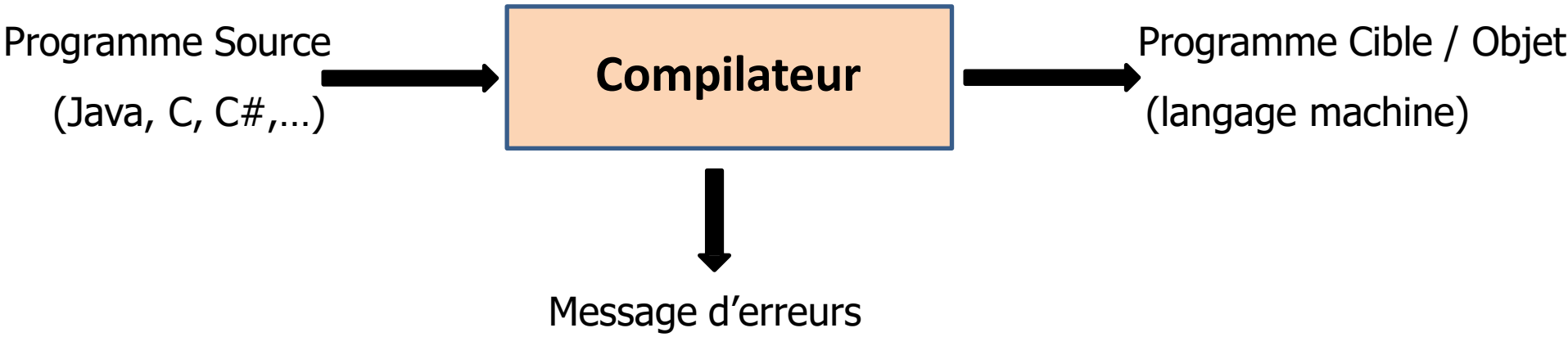
Source: Stephen Chong Harvard University

Compilation

Définition

Un compilateur est un programme qui traduit un programme source de haut niveau en un programme cible.

Le plus souvent, le langage cible est le langage machine, ou un langage proche du langage machine.



Compilation

Compilation & Théorie des langages

Pour espérer avoir des programmes qui « font la même chose » il est important que les langages source et objet soient rigoureusement définis. La définition d'un langage comporte deux volets:

- La syntaxe qui décrit la forme des constructions autorisées;
- La sémantique qui précise le sens des constructions syntaxiquement correctes.

Compilation

Compilation & Théorie des langages

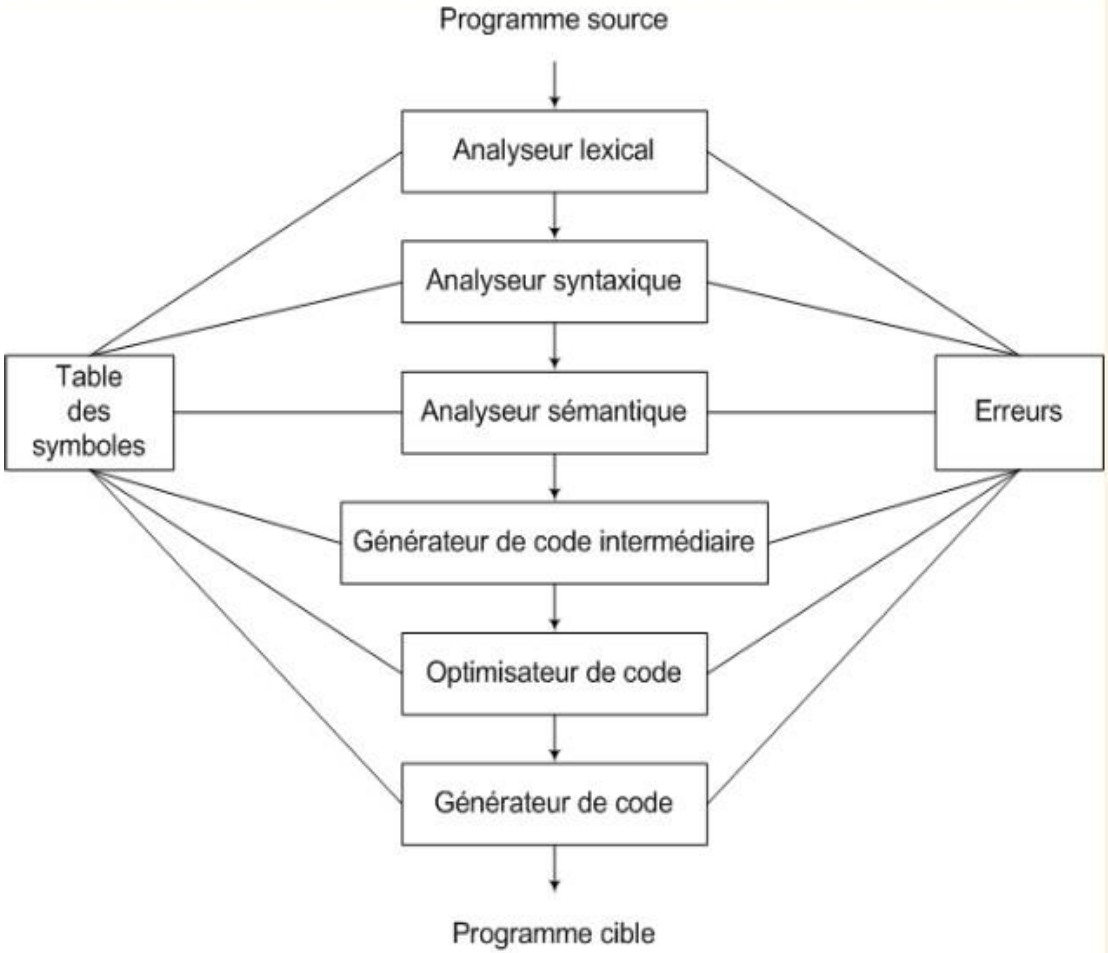
- Pour définir syntaxiquement un langage on dispose d'outils conceptuels fournis par un domaine des mathématiques appelée théorie des langages;
- Pour concevoir et réaliser un compilateur on utilise des concepts, des techniques et des outils d'un domaine de l'informatique appelé compilation. La compilation utilise évidemment la théorie des langages.

Compilation

Phases de la compilation

Les Étapes de la Compilation

- La compilation se décompose en 6 phases : 3 phases d'analyse et 3 phases de génération de code.



Analyse

L'étape de l'analyse se décompose en trois phases:

1. Analyse lexicale
2. Analyse syntaxique
3. Analyse sémantique

Analyse

Analyse lexicale (Scanning)

- La première phase de la compilation est appelée **analyse lexicale ou scanning**. Comme son nom l'indique, elle tente d'isoler les mots d'un programme d'entrée en un ensemble d'entités lexicales appelé **lexèmes**, **élément lexicaux** ou **jetons lexicaux (tokens en anglais).**

Outils théoriques: *Les expressions régulières , les automates à états finis*

❑ **Des exemples de lexèmes:**

- Mots clés: while, void, if, for,…
- Identificateurs: déclarés par le programmeur.
- Opérateurs: +, -, *, /, =, ==, ...
- Constante numérique: 124, 12,35, 0,09E-23

Analyse

Analyse lexicale (Scanning)

- La sortie de la phase lexicale est un flux de jetons. De plus, cette phase crée des tables qui sont utilisées par les phases suivantes du compilateur. Une de ces tables, appelée table des symboles, stocke tous les identifiants utilisés dans le programme source, y compris les informations pertinentes et les attributs des identifiants.

❑ **Exemple:** Nous empruntons l'exemple proposé dans Le Dragon[1]

position := initiale + vitesse * 60

On suppose que les variables position, initiale et vitesse ont été déclarées et sont de type réel

Analyse

Analyse lexicale (Scanning)

- La sortie de la phase lexicale est un flux de jetons. De plus, cette phase crée des tables qui sont utilisées par les phases suivantes du compilateur. Une de ces tables, appelée table des symboles, stocke tous les identifiants utilisés dans le programme source, y compris les informations pertinentes et les attributs des identifiants.

❑ **Exemple:** Nous empruntons l'exemple proposé dans Le Dragon[1]

position := initiale + vitesse * 60

On suppose que les variables position, initiale et vitesse ont été déclarées et sont de type réel

Analyse

Analyse lexicale (Scanning)

position := initiale + vitesse * 60

position	:=	initiale	+	vitesse	*	60

<position, identificateur>
<:=, symb-affectation>
<initiale,identificateur>
<+, symb-plus>
<vitesse,identificateur>
<*, symb-mult>
<60, constante-entière>

Analyse

Analyse lexicale (Scanning)

position := initiale + vitesse * 60

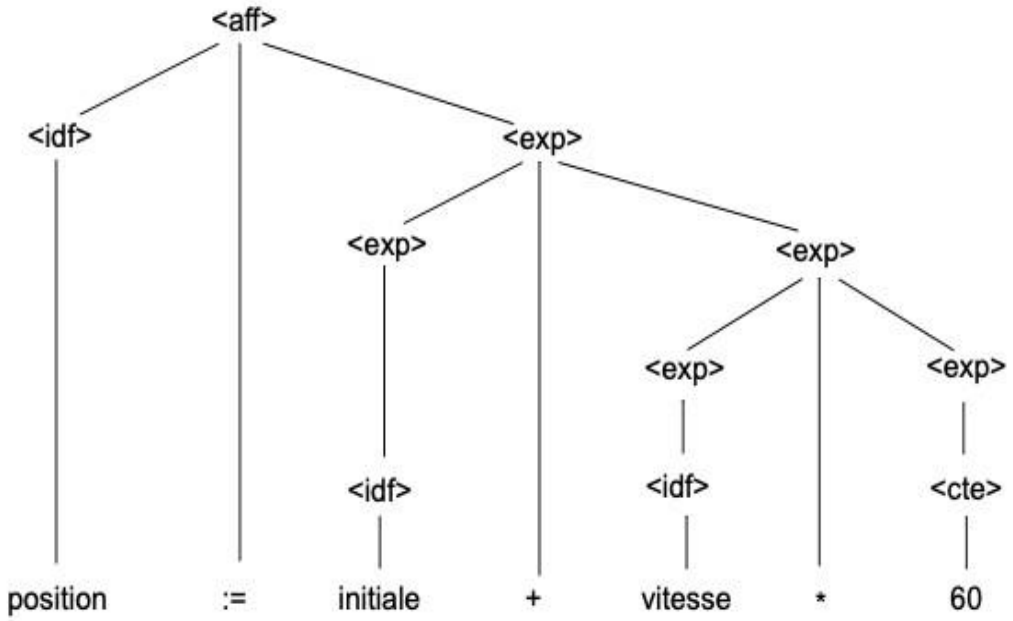
position	:=	initiale	+	vitesse	*	60

<position, identificateur>
<:=, symb-affectation>
<initiale,identificateur>
<+, symb-plus>
<vitesse,identificateur>
<*, symb-mult>
<60, constante-entière>

Analyse

Analyse Syntaxique (Parsing)

L'analyse syntaxique regroupe les unités lexicales en structures grammaticales en suivant les règles figurant dans une grammaire. Le résultat de cette analyse est souvent représenté par un arbre syntaxique.



Outils Théoriques: *Les grammaires*

Analyse

Analyse Sémantique

- L'analyse sémantique vérifie la cohérence sémantique du code.

- Il utilise l'arbre de syntaxe de la phase précédente avec la table des symboles pour vérifier que le code source donné est sémantiquement cohérent.

- Il vérifie également si le code transmet une signification appropriée.

Syntaxe Forme [d'un programme donné]
Sémantique Signification [de ce programme]

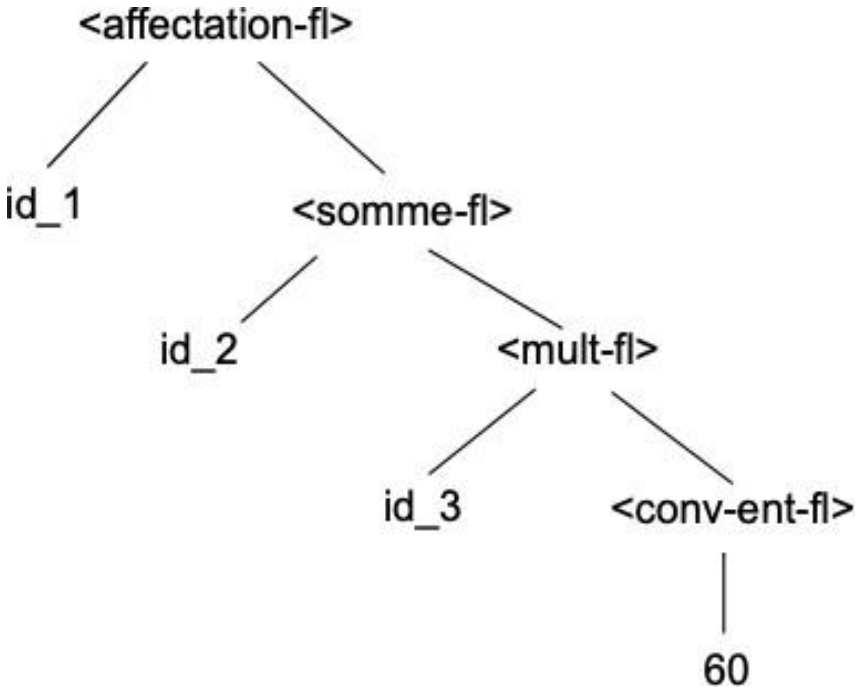
- L'analyseur sémantique vérifiera les incompatibilités de type, les opérandes incompatibles, une fonction appelée avec des arguments incorrects, une variable non déclarée, etc.

Analyse

Analyse Sémantique

- **Dans l'exemple:**

Du point de vue des types, le fragment analysé est correct; il faut noter toute fois la conversion de l'entier 60 vers le réel 60.0 et le fait que les opérations étiquetant l'arbre abstrait sont des opérations flottantes.



Génération de code

La génération de code se fait en 3 étapes:

1. Génération de code intermédiaire
2. Optimisation de code
3. Génération effective du code final

Génération de code

Code intermédiaire

❑ **Génération de code intermédiaire:**

Le générateur de code intermédiaire est une traduction de l'arbre de syntaxe abstraite en code intermédiaire. Il se situe entre le langage de haut niveau et le langage machine. Ce code intermédiaire doit être généré de manière à faciliter sa traduction dans le code machine cible.

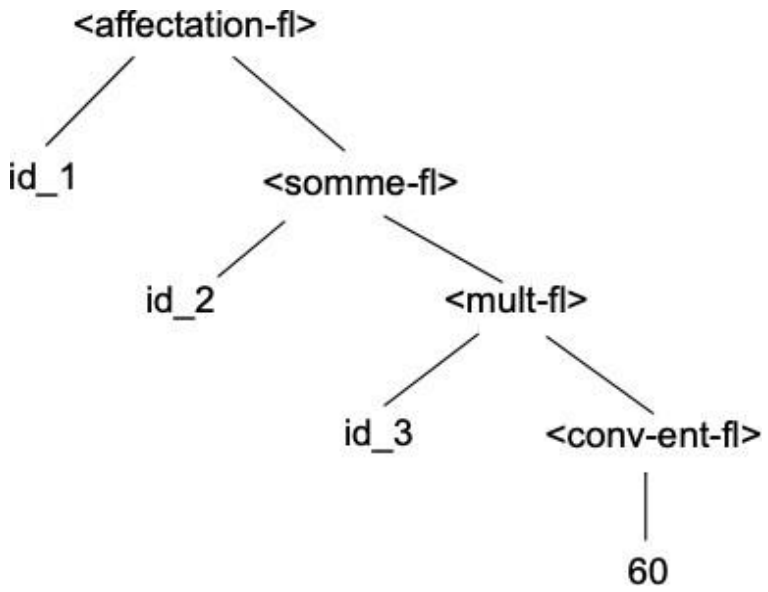
- **Code intermédiaire**

temp_1 := Ent-Vers-Fl (60)

temp_2 := id_3 * temp_1

temp_3 := id_2 + temp_2

id_1 := temp_3



Génération de code

Optimisation de code

- **Optimisation de code:**

L'optimiseur tente d'améliorer le code généré par le générateur de code intermédiaire. L'objectif est généralement de rendre le code plus rapide, mais l'optimiseur peut également essayer de réduire la taille du code.

- **Code amélioré:**

temp_2 := id_3 * 60.0

id_1 := id_2 + temp_2

Génération de code

Code final

- **Génération du code final:**

  Le générateur de code génère un code objet à partir d'un code intermédiaire (optimisé)

  Par exemple, le code suivant peut être généré pour notre exemple.

Code cible:

```
MOVF id_3, R2
MULF #60.0, R2
MOVF id_2, R1
ADDF R2, R1
MOVF R1, id_1
```

Activité Pratique