

Optimization / Code

Analyse Sémantique

Analyse Syntaxique

Analyse Lexicale

Intro



université
virtuelle
Burkina ★ Faso



Sciences
du Numérique

Théorie des Langages et Compilation

MSI-02

Généralités



Dr. Tegawendé F. BISSYANDE

tegawende.bissyande@gmail.com

Semestre 1 / Semaine 5
“Sécurité des Réseaux et des Logiciels”

Disclaimer

- **Source d'Infos**

Cours est basé sur les cours de

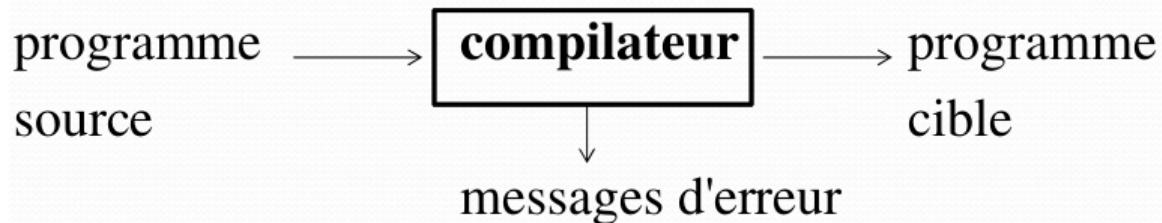
- C. Paulin de Université Paris Sud
- M.D. Rahmani de Université Mohammed V
- compilation du MIT
- ainsi que des ouvrages suivants
- 1. A. Aho, M. Lam, R. Sethi et J. Ullman, "Compilateurs : principe, techniques et outils", Pearson Education.
- 1. R. Wilhelm et D. Maurer, "Les compilateurs: théorie, construction, génération", Masson

Introduction à la Compilation

Définition

Un compilateur est un programme qui lit un programme écrit dans un premier langage (le langage source) et le traduit en un programme équivalent dans un autre langage (le langage cible).

Le compilateur doit aussi vérifier que le programme a un certain sens et signaler les erreurs qu'il détecte.



Etapes de la Compilation

Il y'a deux parties dans la compilation: l'analyse et la synthèse.

- **La partie analyse** partitionne le programme source en ses constituants et en crée une représentation intermédiaire.

- **La partie synthèse** construit le programme cible à partir de cette représentation intermédiaire.

Abstract Tree (Arbre Abstrait)

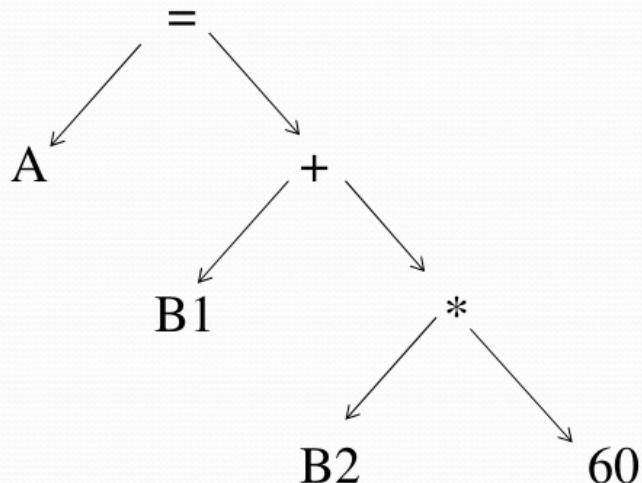
Pendant l'analyse, les opérations spécifiées par le programme source sont déterminées et conservées dans une structure hiérarchique appelée **arbre abstrait**.

*Un arbre abstrait est constitué de **nœuds** qui représentent les **opérations** et les **fils des nœuds** qui représentent les **arguments des opérations***

Exemple d'Arbre Abstrait

Soit l'instruction: $A = B1 + B2 * 60$

Son **arbre abstrait** est:



Analyse du Code Source

Phases d'analyse

1. **L'analyse lexicale** (linéaire): le flot de caractères formant le programme source est **lu** de gauche à droite et **groupé** en *lexèmes* (mots), qui sont des suites de caractères ayant une signification collective.
2. **L'analyse syntaxique** (grammaticale): les unités lexicales sont regroupés hiérarchiquement dans des collections imbriquées (phrases) ayant une signification collective.
3. **L'analyse sémantique**: contrôle pour s'assurer que l'assemblage des constituants du programme a un sens.

Exemple d'Analyse Lexicale

Soit l'instruction: **A = B1 + B2 * 60**

2.1- L'analyse lexicale:

1. L'identificateur "**A**"
2. Le symbole d'affectation "**=**"
3. L'identificateur "**B1**"
4. Le signe "**+**"
5. L'identificateur "**B2**"
6. Le signe de multiplication "*****"
7. Le nombre "**60**"

Remarque: les blancs qui séparent les mots sont éliminés.

Exemple d'Analyse Syntaxique

Les structures grammaticales sont représentées par un arbre syntaxique et généralement exprimées par des règles récursives.

Les règles de grammaires:

➤ règles de base non récursives:

1. *Tout identificateur est une expression.*
2. *Tout nombre est une expression.*

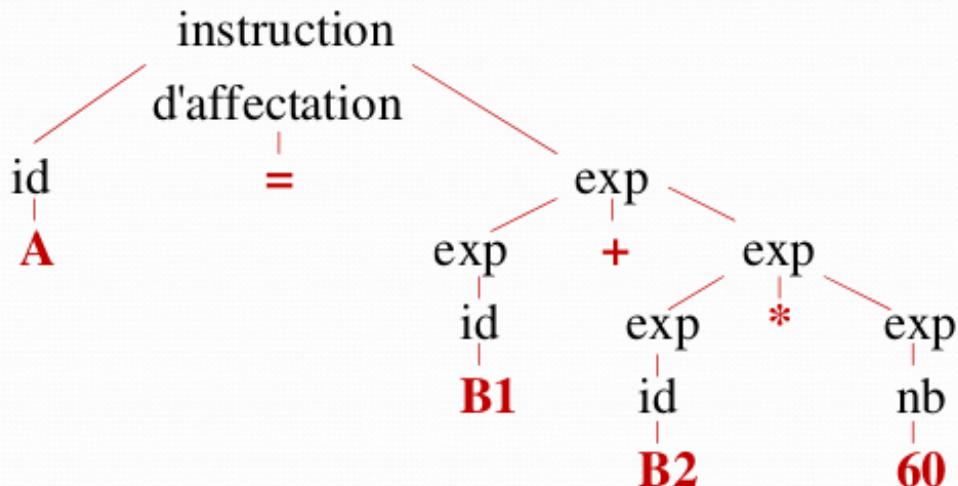
➤ règle récursive:

3. *Si $e1$ et $e2$ sont des expressions, $e1+e2$, $e1*e2$ et $(e1)$ sont aussi des expressions.*

Exemple d'Analyse Syntaxique (suite)

4. Une instruction peut être de la forme: $id = exp$

L'arbre syntaxique correspondant:

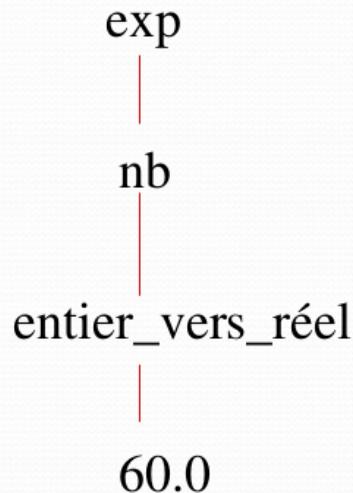


Remarque: L'arbre abstrait est une forme compacte de l'arbre syntaxique

Exemple d'Analyse Sémantique

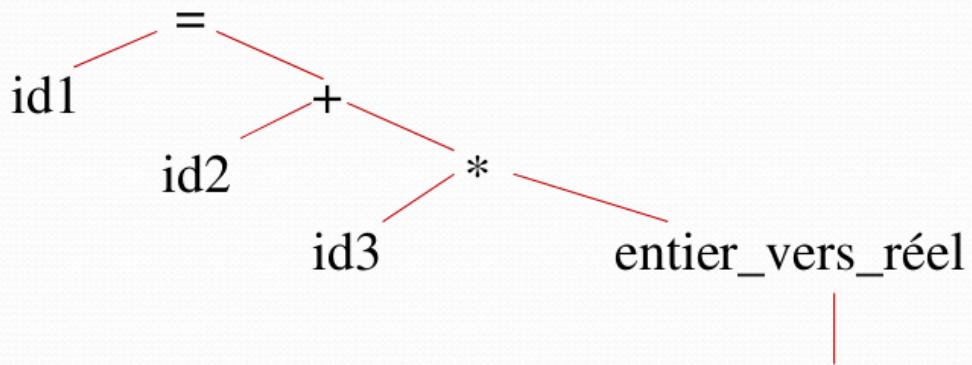
Un constituant important est le contrôle su type.

Si nous déclarons A, B1 et B2 des réels, alors nous devons convertir l'entier 60 en réel 60.0



Exemple de Code Intermédiaire

L'arbre abstrait:



```
temp1 := entier_vers_réel(60);      60.0
temp2 := id3 * temp1;
temp3 := id2 + temp2;
id1   := temp3;
```

Synthèse du Code Cible

Exemple d'Optimisation de Code

Amélioration du code intermédiaire pour que le code final s'exécute plus rapidement et utilise le minimum de mémoire.

```
temp1 := id3 * 60.0;  
id1 := id2 + temp1;
```

Génération du Code

Production du code cible en langage d'assemblage.

L'aspect crucial est l'assignation des variables aux registres du processeur.

Si nous utilisons 2 registres $R1$ et $R2$, la traduction pourrait être :

MOVF R2, id3

F: nombre en virgule flottante

MULF R2, #60.0

MOVF R1, id2

ADDF R1, R2

#: 60.0 doit être traité comme

MOVF id1, R1

une constante

Implémentation de Compilateurs

Frontend (Partie frontale)

L'étude précédente traite de l'organisation logique d'un compilateur. Souvent on regroupe les phases en 2 parties:

La partie frontale: constituée des phases qui dépendent principalement du langage source:

- l'analyse lexicale,
- l'analyse syntaxique,
- l'analyse sémantique,
- la création de la table des symboles,
- la production du code intermédiaire
- elle inclut le traitement des erreurs associées à chacune de ces phases.

Backend (Partie finale)

Elle est constituée des phases qui dépendent de la machine cible.

- Optimisation du code intermédiaire,
- production du code final,
- gestion de la table des symboles,
- traitement des erreurs.

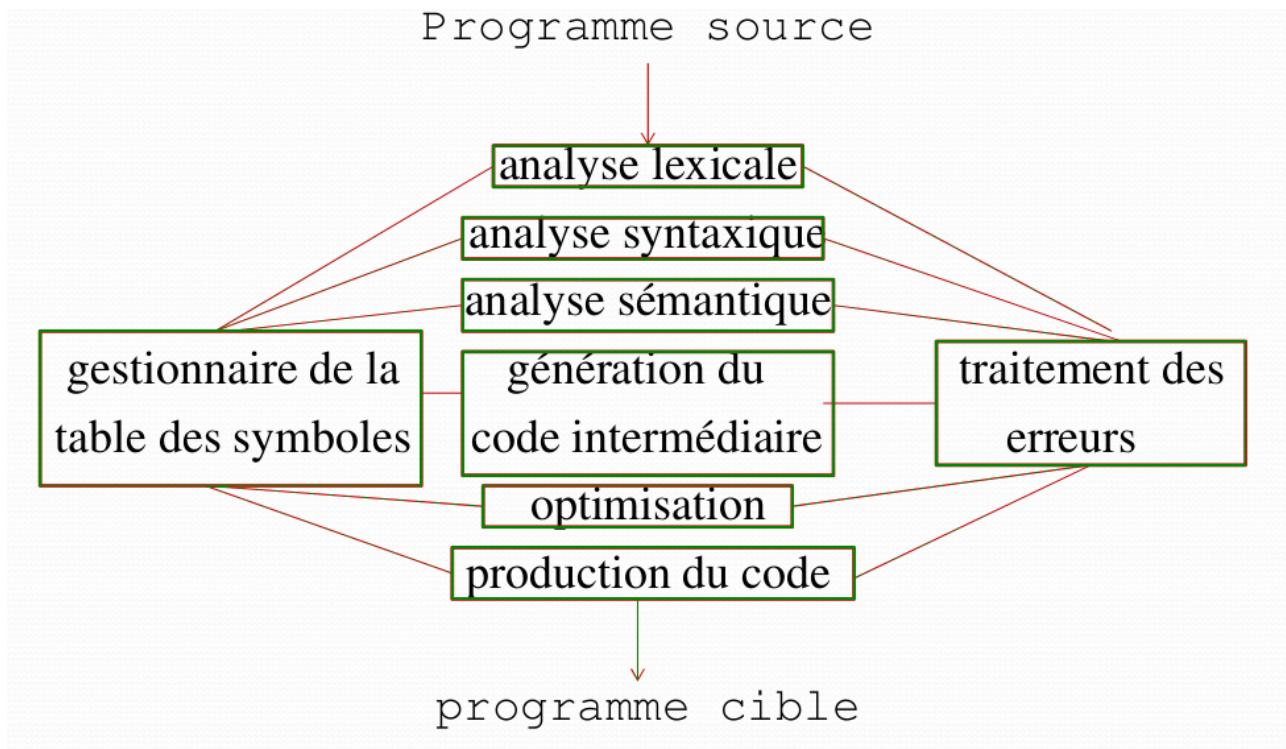
Six phases

Conceptuellement un compilateur opère en 6 phases,

Chacune transforme le programme source d'une représentation en une autre.

En outre 2 phases interagissent avec les 6 phases

Schema de déroulement



Outils utilisés à chaque étape

Analyse

- analyse lexicale
 - expressions régulières
 - automates à états finis
- analyse syntaxique
 - grammaires
 - automates à piles
- analyse sémantique

Synthèse

- code intermédiaire - traduction dirigée par la syntaxe
- optimisation
- production du code

Traitements Parallèles

- table des symboles
- traitement des erreurs

Génération de la Table de Symboles

Une table des symboles est une structure de données contenant un enregistrement pour chaque identificateur, muni de champs pour ses attributs (*emplacement mémoire, son type, sa protéction..*).

Pour les fonctions, la table des symboles garde le *nombre et les types de ses arguments, le mode de passage de chacun d'eux et la valeur de retour.*

Traitements d'Erreurs

Chaque phase peut rencontrer des erreurs.

Après avoir détecté une erreur, une phase doit la traiter de telle façon que la compilation puisse continuer et que d'autres erreurs dans le programme puissent être détectées.

Edition de Liens (chargeurs et relieurs)

Charger les fichiers et les fonctions des bibliothèques
et les relier pour la production du code exécutable.

Interprètes

Au lieu de produire un programme cible, un interprète effectue lui même les opérations spécifiées par le programme source.

On utilise souvent des interprètes pour exécuter les langages de commande.

exemples:

- Le *shell* d'Unix
- Le *command.com* du DOS

**Vous savez (presque) tout de la
compilation!!!**

