Langages et Compilation
Chap. 1 – Introduction

H. Fouchal

Université de Reims Champagne-Ardenne

January 31, 2010

Objectifs et plan du cours

$\Rightarrow$ **Objectif : définir la notion de compilateur et les principes de base.**

1. Introduction à la compilation
   - Environnement d'un compilateur
   - Compilation en deux étapes
2. Analyse lexicale : langages et automates
   - Automates déterministes et non déterministes
   - Construction automatique d'un analyseur lexical
3. Analyse syntaxique
   - Dérivation gauche et droite
   - Analyse syntaxique descendante et remontante.

Plan du cours (suite)

4. Traduction dirigée par la syntaxe
   - Attributs et actions sémantiques
   - Arbres abstraits
5. Tables des symboles
   - Représentation d'une table des symboles
   - Vérification de type
6. Génération du code intermédiaire
   - Code à trois adresses
   - Génération de code pour les expressions booléennes
7. Gestion de la mémoire à l'exécution
8. Introduction à l'optimisation du code intermédiaire.

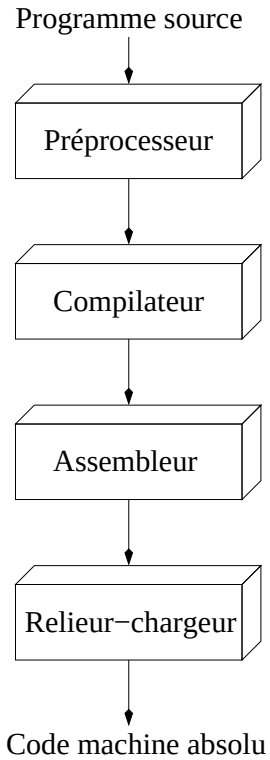
Un compilateur est un programme qui lit un *programme* écrit dans un premier *langage* (programme source) et le traduit en un programme *équivalent* écrit dans un autre langage (cible). Pendant la traduction, le compilateur signale les éventuelles erreurs à l'utilisateur.

La compilation fait appel à :

- la théorie des langages ;
- l'architecture des machines ;
- l'algorithmique et le génie logiciel.

Environnement du compilateur

Au début des années 50, la réalisation d'un compilateur est difficile
Aujourd'hui : utilisation de techniques systématiques.



éventuellement dispersé dans différents fichiers
contient des définitions de macros...

remplace les macros par leur définition
crée un fichier unique

programme cible en langage d'assemblage

code machine translatable

inclut les bibliothèques

Principe de compilation en 2 étapes

2 phases : analyse et synthèse

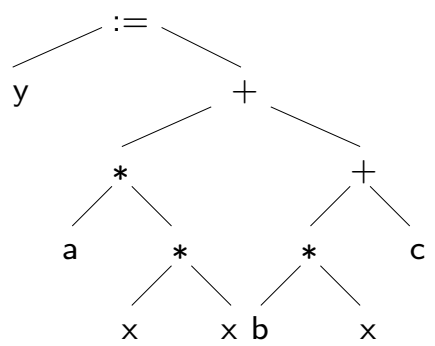
Partie analyse : sépare les $\neq$ constituants du prog. source et produit une *représentation intermédiaire*

Partie synthèse : génère le prog. cible à partir de la rep. intermédiaire

Représentation intermédiaire : souvent un *arbre abstrait*.

- nœuds : opérations du programme source
- feuilles : arguments de ces opérations.

$y := ax^2 + bx + c$



Partie analyse

Exemple d'outils utilisant des techniques d'analyse :

- éditeur structurel : analyse de la structure, fournit les mots clés du langage, vérifie l'adéquation entre début et fin de bloc...
- interpréteurs de commandes.

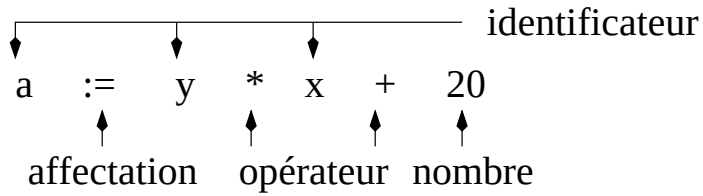
Analyse du programme source en 3 phases :

1. anlayse lexicale ;
2. analyse syntaxique ;
3. analyse sémantique.

Analyse lexicale ou analyse linéaire

Le but de l'analyseur lexical est de reconnaître les unités lexicales ou lexèmes :

- les identificateurs et les mots clés du langage ;
- l'affectation et les opérateurs (+,*,...).



Analyse syntaxique

On peut également définir la notion d'*instruction* :

- Soit *id* un identificateur et *exp* un expression, alors
  id := *exp* est une instruction.
- Soit *exp* une expression et *inst* une instruction, alors
  si *(exp)* **alors** *inst* **fsi** est une instruction.

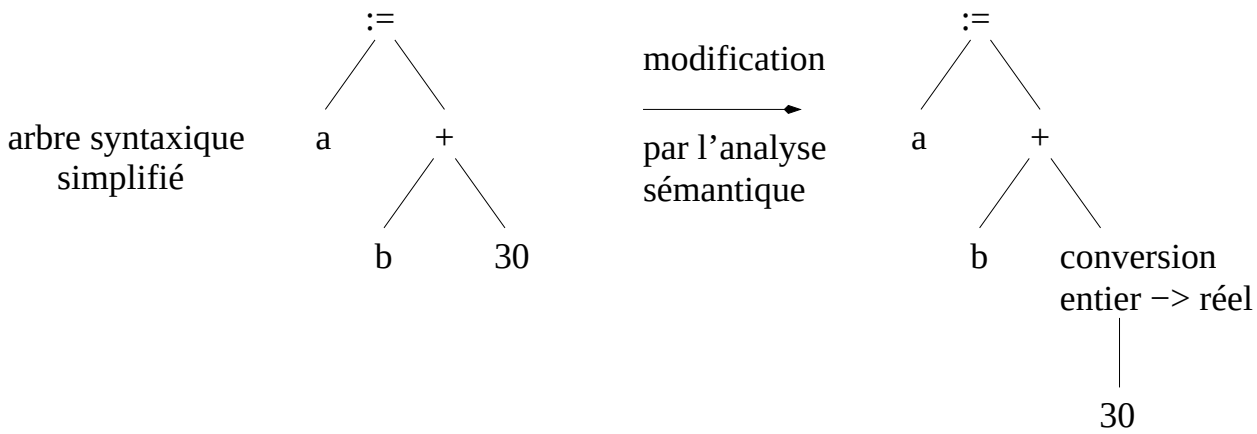
Exemple :

- *a:=5*b+c* est une instruction (arbre syntaxique)
- **si** *5* **alors** *a:=b* **fsi** n'est pas une instruction.

⇒ utilisation de **grammaires**

Analyse sémantique

$\Rightarrow$ vérifie la présence d'erreurs d'ordre sémantique (vérification de type).
Utilisation de l'arbre résultat de l'analyse syntaxique.
Par exemple, si a et b sont des réels, $a := b + 30$ est une instruction.

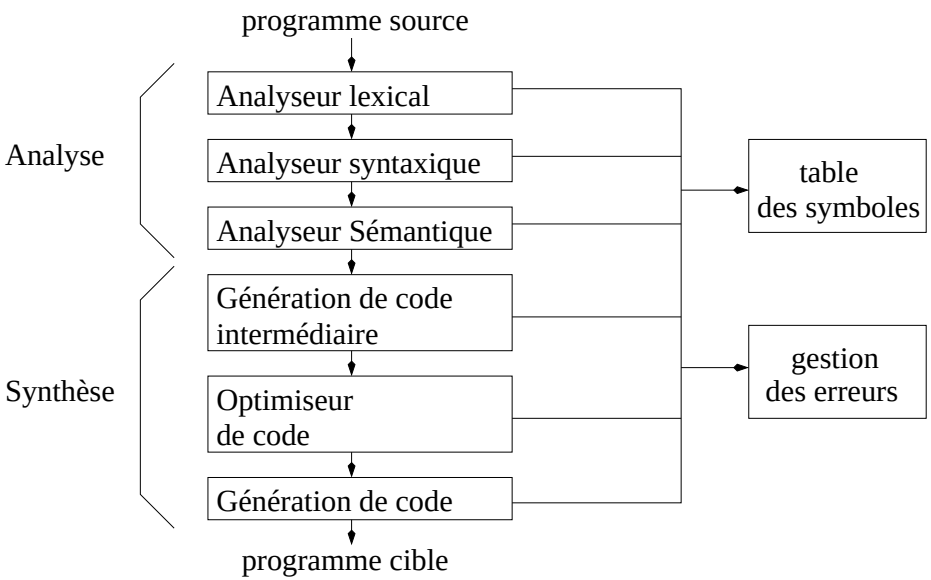


Cette modification est nécessaire car la représentation des entiers est $\neq$ de celle des réels.
Autre exemple d'erreur sémantique : tableau indicé par un réel.

Synthèse du code

3 phases :

1. génération du code intermédiaire
2. optimisation du code
3. génération du code exécutable.



Synthèse

Génération du code intermédiaire :

- ce n'est pas obligatoire mais 2 bonnes propriétés : facile à produire et à traduire en langage machine ;
- utilisation de variables temporaires ;
- choix de l'ordre pour faire un calcul.

Optimisation de code :

- amélioration du code intermédiaire ;
- réduction du nombre de variables et d'instructions.

Production de code :

- choix des emplacements mémoire pour les variables ;
- assignation de variables aux registres.

Table des symboles et détection des erreurs

La table des symboles enregistre les identifiants et les attibuts (emplacement mémoire, type, portée) :

- chaque identifiant (variable) a une entrée dans la table des symboles ;
- l'analyseur lexical crée dans la TS, une entrée à chaque fois qu'il rencontre un nouvel identificateur. Par contre, les attributs seront calculés plus tard.
- L'analyseur sémantique se servira de la table des symboles pour vérifier la concordence des types.

Table des symboles et détection des erreurs

Détection des erreurs à plusieurs niveaux, essentiellement pendant l'analyse :

- erreur lexicale : le flot de caractères n'est pas reconnu ;
- erreur syntaxique : construction non reconnue par le langage ;
- erreur sémantique : pb de typage,...

Exemple de traduction