# Compilation - Frédéric Dabrowski

Alexandre Masson

14 Janvier 2013

# 1 Introduction

- langages de haut niveau : langages itératifs , logiques, spécialisés
- il faut trouver un moyen de convertir ces codes en code machine pour les exécuter.
- grandes variété de compilateur
  - distincts dont la façon dont il ont été construits et la fonction censé
  - nombre de passes principalement
- Compilateur et Interpréteur :
  - le compilateur traduit un programme dans une autre représentation
  - les deux programmes sont sémantiquement équivalents
  - le cout supplémentaire lié à la compilation sera amorti au moment de l'exécution (perte de temps à la compilation pour certaines optimisation , mais gain de temps à l'exécution)

**Comparaisons Compilateur et Interprète** Interprète
- + : test immédiats, rapidité de mise au point, convivialité
- - : place en mémoire, temps d'exécution

Compilateur
- + : gestion efficace de la mémoire, tps d'exec, meilleure détection des erreurs
- - : plus lourd à mettre en oeuvre , moins souple.

**Travail du compilateur**
- l'analyse
- la synthèse
- objectifs :
  - faire toute les vérif statiques
  - transformation code source-code source(correct)
  - traduction *production et optimisation du code machine*

**Analyse** lexicale : scanner et cibler.
syntaxique : parser.
Analyse sémantique

**analyse lexicale** suite de caractères : suite de mots
lit une suite de caractère et construit une suite lexicale(token)

**le cribble** Reconnait dans la suite de symbole produite par le scanner

**le parser** Reconnait la structure syntaxique.
L.
Schéma en BNF.

**Analyse sémantique** Optimisation indépendante de la plateforme cible et indications sur les risques d'erreur à l'exécution.

existence possible d'un chemin d'exécution dans lequel une variable est utilisée sans avoir été init.

## 2 Synthèse

**affectation d'adresses** Interviennent des caractéristiques de la machine cible. elles déterminent l'affectation des unités mémoire pour les types élémentaires.

Il y a souvent une contrainte d'adressage au niveau des instructions pour des raisons d'efficacité.

**Génération de code** Il produit les instructions du programme cible.Pour adresser les variables, utilise les adresses calculées lors de l'étape précédente. Garde les valeurs des expressions et des variables dans les registres de la machine autant que possible pour efficacité.//// effectue la sélection de code : meilleur choix possible d'instruction.

**les passes du compilateur** Une passes correspond à un parcours d'un des niveau de représentation ( source, intermédiaire,...). Plusieurs phases peuvent être regroupées en une seule passe , par exemple les analyses.

# 3 les outils : Lex et Yacc

**lex et yacc**

- Lex : générateur d'analyseurs lexicaux
  - prend en entrée la définition des unités lexicales
  - produit un automate fini déterministe minimal permettant de reconnaître ces unités
  - l'automate est produit sous forme d'un prog
- Yacc : générateur d'analyseurs syntaxiques

**Compilation**

- bison -d calc.c
- produit calc.tab.c
- produit calc.h qui contient la définition des codes des unités lexicales, afin qu'elles puissent être partagées

**Structure d'un fichier Lex** %{
Partie 1 : déclarations pour le compilateur C

%}
Partie 2 : définitions régulières

%%
Partie 3 : règles

partie 4 : fonction C supplémentaires

Les parties 1 et 4 sont optionnelles , en effet on est pas obligé de passer par la partie C

Définir des expressions régulières dans la partie 2 évite les recopie d'expression dans la partie 3 lors de l'utilisation fréquente d'une expression, de plus la possibilité de la nommer est pratique et rend la partie 3 beaucoup plus lisible.

une règle est de la forme : expressionRégulière action. l'élément gauche de la première règle est l'axiome.