Compilation

Mirabelle Nebut

Bureau 223 - extension M3
mirabelle.nebut at lifl.fr

2010-2011

À propos du cours

Organisation : C / TD / TP sur 12 semaines

Évaluation :

- par contrôles continus courts en amphi ;
- des TP rendus ;
- examen de 3h en fin de semestre.

Docs et infos là (mais ne dispensent pas d'assister au cours) :

`http://www.fil.univ-lille1.fr/portail/`

Organisation première semaine

Intensif !

Cours 1 :

- cours d'introduction ;
- analyse lexicale.

Cours 2 : grammaires algébriques

TD : analyse lexicale

Introduction à la compilation

Définitions, généralités
- Un compilateur, c'est quoi ?
- Outils pour la compilation

Structure d'un compilateur
- Structure générale
- Structure conceptuelle détaillée
- Structures de données

Contenu du cours de Compil

Bibliographie

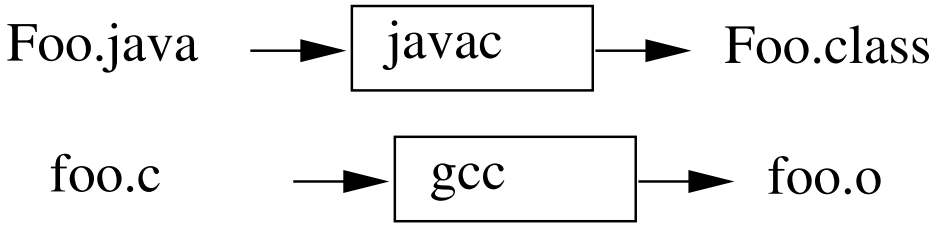
Définitions, généralités
Un compilateur, c'est quoi ?
Outils pour la compilation

Structure d'un compilateur
Structure générale
Structure conceptuelle détaillée
Structures de données

Contenu du cours de Compil

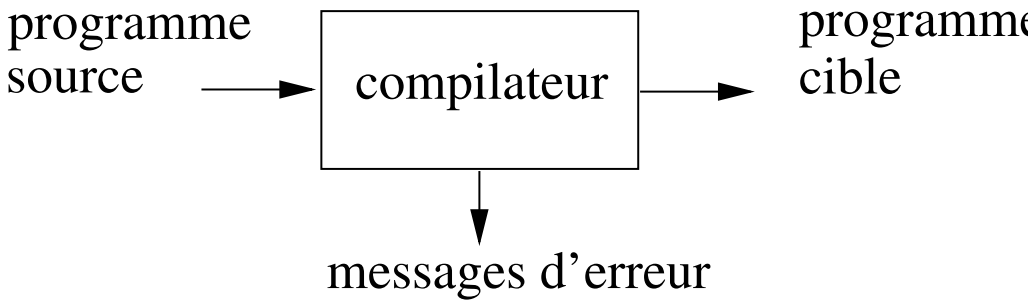
Bibliographie

Un compilateur, c'est quoi ?

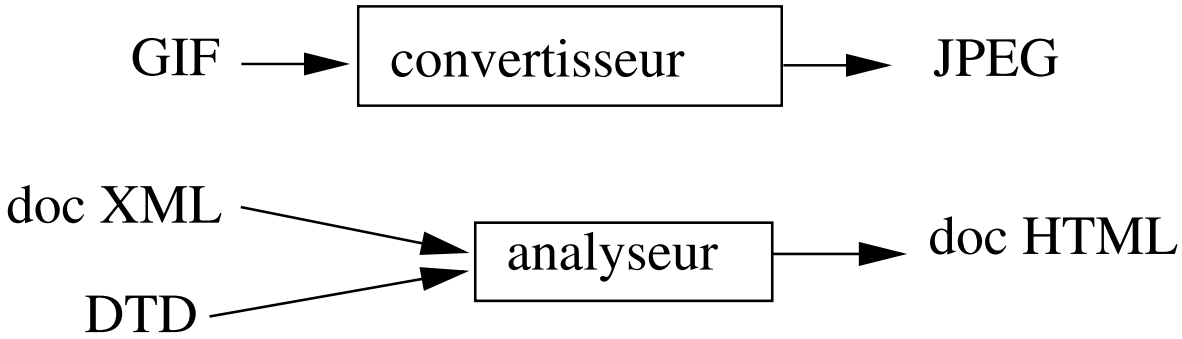


Dans ce cas un logiciel qui produit un exécutable à partir d'un programme :

- entrée : programme dans le langage source ;
- sortie : programme (équivalent) dans le langage cible ;
- ou message(s) d'erreur si entrée non correcte.



Mais encore ?



Dans ce cas un logiciel qui tranforme une entrée textuelle en sortie équivalente :

- source et cible pas nécessairement des programmes ;
- cible pas nécessairement exécutable ;
- conservation de l'information ;
- correction de l'entrée.

Mais aussi ?

Dans ce cas un logiciel qui reconnaît et analyse une entrée textuelle pour produire une sortie :

- source et cible pas nécessairement équivalents ;
- notion de correction du source ;
- application de traitement de données textuelles.

Donc, un compilateur, c'est quoi ?

C'est un logiciel qui :

- prend en entrée une donnée textuelle source (programme, donnée xml, fichier de configuration, etc) ;
- la reconnaît (l'analyse) pour vérifier sa correction ;
- émet éventuellement un message d'erreur ;
- calcule une donnée de sortie (programme, donnée, etc).

Les compilateurs et vous

- Parmi vous, peu seront amenés à travailler sur un compilateur pour Java !
- Par contre, vous utilisez tous les jours un compilateur.
- $\Rightarrow$ important de savoir comment ça marche.

Et surtout :

- les applications de traitement des données textuelles sont le quotidien des informaticiens ;
- $\Rightarrow$ important de savoir les concevoir.

En cours/TD : pas de production de code.

En TP : compilateur pour un mini-langage ($\longrightarrow$ byte-code Java)

Outils pour la compilation

Des critères importants pour faire un "bon" compilateur :

- correction :
  - entrée invalide détectée ?
  - sortie conforme aux attentes ?

  $\Rightarrow$ outil de prédilection = théorie du langage
- efficacité :
  - faut-il attennnnnnndre le résultat de la compilation ?

  $\Rightarrow$ outil de prédilection = algorithmique
- bonne conception du logiciel :
  - logiciel facile à modifier/étendre ?

  $\Rightarrow$ outil de prédilection = génie logiciel

C'est ce qui fait de la compilation un sujet varié et passionnant !

♡ ♡ ♡

À quoi sert la théorie du langage (en compilation)

Permet de définir rigoureusement et reconnaître algorithmiquement (pour les langages source et cible) :

- leur vocabulaire ou lexique : les mots autorisés ;
  ⇒ automates à nombre fini d'états, expressions régulières ;
  ⇒ analyse lexicale ;
- leur syntaxe : la structure des phrases autorisées ;
  ⇒ automates à pile, grammaires algébriques ;
  ⇒ analyse syntaxique ;
- leur sémantique : la signification des phrases autorisées ;
  ⇒ grammaires attribuées ;
  ⇒ analyse sémantique.

À quoi sert le génie logiciel (en compilation)

Notions bien identifiées et couramment admises :

- structuration d'un compilateur en modules ;
- conception objet, structures de données, algorithmes ;
- génération automatique de code pour les analyseurs lexical et syntaxique.

Définitions, généralités
Un compilateur, c'est quoi ?
Outils pour la compilation

Structure d'un compilateur
Structure générale
Structure conceptuelle détaillée
Structures de données

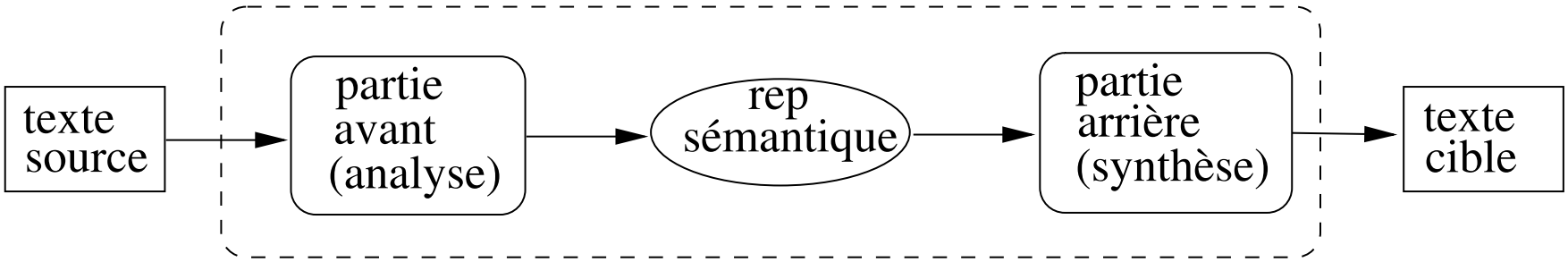
Contenu du cours de Compil

Bibliographie

Structure globale

En deux parties :

- analyse/reconnaissance ;
- synthèse/transformation.



Structure classique d'une application de traitement de données textuelles.

Structure conceptuelle détaillée

- Plusieurs découpages en module possibles ;
- document de cours : tiré de [Wilhem & Maurer] ;
- structure conceptuelle $\neq$ en pratique ;
- illustration par l'exemple :

```
program pgm;
int x,y;
x := 2;
y := x*x+1;
```

Analyse lexicale

- seul module au contact avec le texte source ;
- lit le texte source sous la forme d'une suite de caractères ;
- décompose cette suite en une suite d'unités lexicales appelées symboles ou tokens ;
- en pratique souvent sous-programme appelé par l'analyseur syntaxique ;
- génération automatique de code courante (à partir d'une description formelle des unités lexicales, cf TP).

Crible

- détermine quels symboles sont importants pour la suite de l'analyse et sont envoyés à l'analyseur syntaxique ;
- supprime tous les symboles qui ne sont pas significatifs de la structure du texte ;
- numérotation éventuelle des identificateurs ;
- souvent confondu avec l'analyseur lexical.

Analyse syntaxique

- reçoit les symboles issus de l'analyse lexicale ;
- en pratique appelle l'analyseur lexical ;
- sait comment est structuré un texte correct (expressions, instructions, déclarations, etc) ;
- tente de reconnaître dans le flot des symboles la structure d'un texte correct ;
- structure sous-jacente : arbre syntaxique ;
- différentes approches, génération automatique de code courante (cf TP).

Analyse sémantique

Vérifie certaines propriétés dites statiques (= à la compilation, par opposition à dynamique = à l'exécution).

- vérification de typage ;
- vérification des déclarations ;
- en pratique, parfois intégré à l'analyse syntaxique ;

Produit une représentation interne du source, selon les cas :

- un arbre syntaxique décoré ;
- un arbre abstrait décoré ;
- un code intermédiaire ;
- ...

Optimisations indépendantes de la machine cible

Analyses plus ou moins poussées pour signaler :

- risques d'erreur à l'exécution ;
- opportunités d'optimisation.

Analyse de flot de données :

- propagation de constantes ;
- indication des variables non initialisées/non utilisées ;
- élimination du code mort ;
- élimination de calculs redondants. . .

Allocation mémoire

Début de la phase de synthèse, dépend de la machine cible :

- longueur d'un mot ou d'une adresse ?
- entités directement adressables de la machine ?
- contraintes d'alignement (frontière de mots) ?

Ex : `x` adresse 0, `y` adresse 1.

Génération du programme cible

- utilise les adresses calculées par l'allocation mémoire ;
- accès aux registres plus rapide qu'accès à la mémoire ;
- nombre de registres limité : allocation des registres.

Ex : on suppose une machine cible avec les instructions suivantes :

LOAD *adr*, *reg* ADDI *int*, *reg*
STORE *reg*, *adr* MUL *adr*, *reg*
LOADI *int*, *reg*

et un registre R. Pour `x := 2 ; y := x*x+1 ;`

```
1  LOADI 2,R     2  STORE R,0
3  LOAD 0,R      4  MUL 0,R
5  ADDI 1,R      6  STORE R,1
```

Optimisations dépendantes de la cible

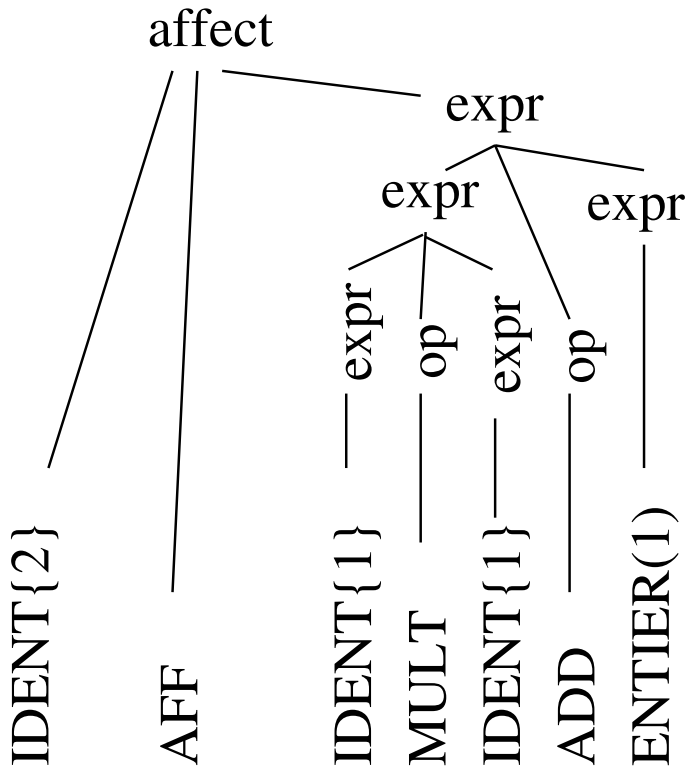
Optimisation à lucarne : déplacement d'une fenêtre sur le programme cible, pour améliorer les parties visibles.

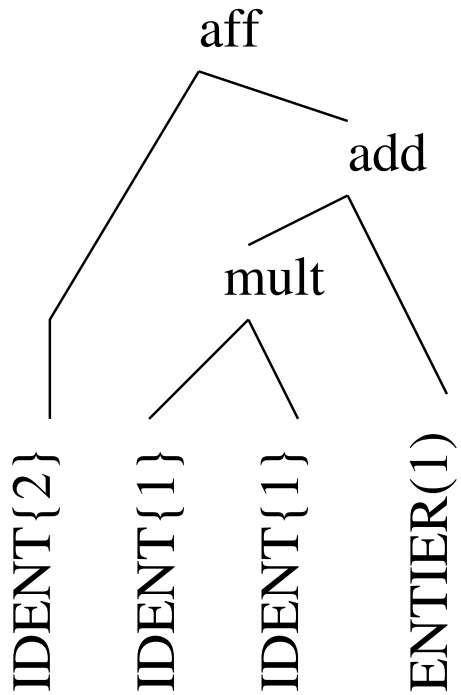
- éliminer les instructions inutiles ;
- remplacer les instructions générales par des instructions plus efficaces.

Ex :

- `STORE R,0  LOAD 0,R` : LOAD inutile ;
- si la machine possède une instruction INC *reg*, remplacer ADDI 1,R par INC R.

Des arbres un peu partout





Et les algorithmes qui vont avec.

Autres structures

- Beaucoup d'automates (finis, à pile) ;
- Piles ;
- Tables (ex : des symboles).

Et les algorithmes qui vont avec.

Définitions, généralités
Un compilateur, c'est quoi ?
Outils pour la compilation

Structure d'un compilateur
Structure générale
Structure conceptuelle détaillée
Structures de données

Contenu du cours de Compil

Bibliographie

Ce qui me tient à cœur

Sensibilisation aux notations formelles

Sensibilisation à la génération automatique de code

Sensibilisation au test du logiciel en TP

Plan du cours

Surtout partie avant des compilateurs :

- analyse lexicale : techniques et outils ;
- analyse syntaxique :
  - grammaires algébriques ;
  - automates à pile, automate des items ;
  - analyse descendante LL(1) ;
  - analyse ascendante ;
- analyse sémantique : grammaires attribuées
- génération de code : TP.

Les TPs

Mini-projet : réalisation d'un mini-compilateur pour le mini-langage impératif Ava v2 :

- génération d'un analyseur lexical (avec JFlex) ;
- génération d'un analyseur syntaxique (avec Cup) ;
- construction d'un arbre abstrait ;
- vérification de type ;
- génération de bytecode Java.

+ un TP sur l'analyse LL(1) descendante.

Bibliographie

Bibliographie complète sur le portail :

- "Le dragon" : Aho, Sethi, Ullman, *Compilateurs : principes, techniques et outils* ;
- un ouvrage très pragmatique : Grune, Bal, Jacobs, Langendoen, *Compilateurs* ;
- un ouvrage plus formel, très rigoureux : Wilhelm, Maurer, *Les compilateurs : théorie, construction, génération* ;
- un ouvrage très pragmatique, orienté Java : Appel, *Modern Compiler Implementation in Java*.

Les intervenants

- Groupe 1 : Cédric Lhoussaine
- Groupe 2 : Mirabelle Nebut
- Groupe 3 : Mirabelle Nebut