Compilation

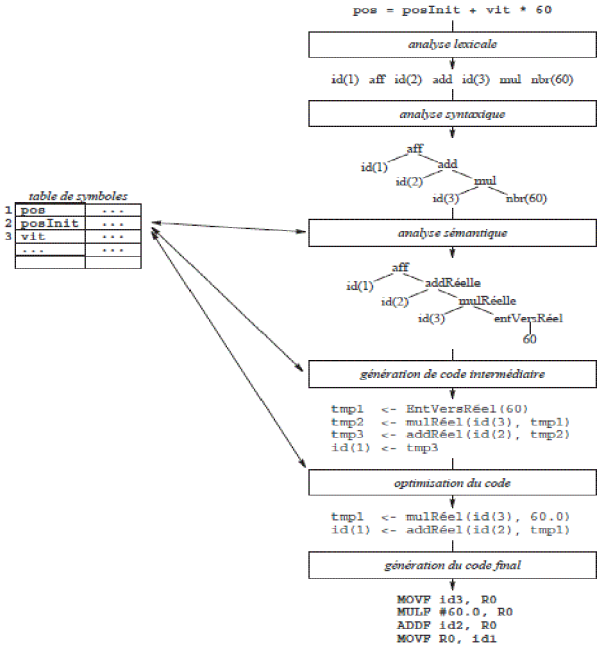
1- Introduction

Introduction

- Les **compilateurs** sont les outils que nous utilisons le plus fréquemment.
- La compilation est une **traduction** : un texte écrit en Pascal, C, Java, etc., exprime un algorithme et il s'agit de produire un autre texte, spécifiant le même algorithme dans le **langage machine**.
- En **généralisant**, on peut dire que compiler c'est lire une suite de caractères obéissant à une certaine syntaxe, et construire une autre représentation de l'information.

Entrée/Sortie

- **Entrée** : toujours la même chose
  - une suite de caractères, appelée le texte source.
- **Sortie** : nature variable
  - un programme exécutable pour un processeur physique
  - un fichier de code pour une machine virtuelle
  - un code abstrait destiné à un autre outil
  - etc.



Les phases

Analyse lexicale

- Les caractères isolés qui constituent le texte source sont regroupés pour former des **unités lexicales**, qui correspondent aux **mots du langage**.
- L'analyse lexicale opère sous le contrôle de l'analyse syntaxique; c'est une fonction de lecture, qui **fournit un mot lors de chaque appel**.

Analyse syntaxique

- Alors que l'analyse lexicale reconnaît les mots du langage, l'analyse syntaxique en reconnaît les **phrases**, la **grammaire**.
- Le rôle principal de cette phase est de dire si le texte source est **conforme à la syntaxe** du langage.

Les phases

Analyse sémantique

Il s'agit ici de vérifier certaines propriétés sémantiques, c'est-à-dire relatives à la signification de la phrase et de ses constituants :

- les identificateurs ont-ils été **declarés** ?
- les opérandes ont-ils les **types** requis ?
- y a-t-il pas des **conversions** à insérer ?
- les **arguments** des appels de fonctions ont-ils les types requis ?
- etc.

Les phases

Optimisation du code

Transformer le code afin que le programme résultant s'exécute **plus rapidement** :

– inutile de recalculer des expressions dont la valeur est **déjà connue**,

– transporter à l'extérieur des boucles des expressions dont les opérandes ont la **même valeur** à toutes les itérations,

– **détecter, et supprimer**, les expressions inutiles,

– etc.

Les phases

Génération du code finale

- Pas forcément la plus difficile à réaliser
- Elle nécessite la **connaissance de la machine cible** (réelle ou abstraite), et notamment ses possibilités en matière de registres, piles, etc.
- Deux grandes catégories **d'architectures** pour la machine cible :
  - Architectures à pile
  - Architectures à registre

Analyse lexicale

- **Entrée** : suite de caractères
- **Sortie** : suite d'unités lexicales (seront utilisées par l'analyse syntaxique)
- **Tâches** :
  - Détection et suppression des séparateurs (blancs, retour fin de ligne, commentaires, etc)
  - Reconnaissance des unités lexicales
    - symboles simples et doubles : +, =, **(**, **)**, … **<=**, **:=**,
    - mots-clés réservés: **if**, **while**, **return**, …
    - nombres : **1000**, **2e-9**, …
    - identificateurs : **i**, **hello, world, toto, ...**

Analyse lexicale

Définitions

- L'unité lexicale :

  représentée par une **constante** conventionnelle :

  INFEGAL, PLUS, IDENTIF, PLUSPLUS, NOMBRE, TANTQUE, ...

- Son modèle :

  spécifie l'unité lexicale en utilisant les **expressions régulières**

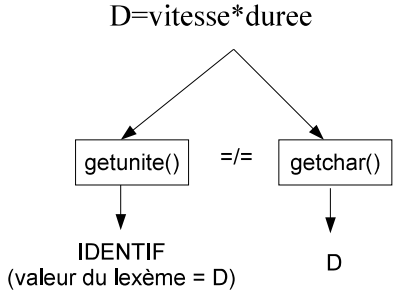
  <=, +, lettre(lettre | chiffre)*,++, ...

- Le lexème reconnu :

  la **suite de caractères reconnue** (elle doit être **maximale**)

  "<=" "helloworld"

Analyse lexicale



- Écrire l'analyseur lexicale c'est écrire la fonction getunite()
- C'est possible d'utiliser des outils pour obtenir un analyseur lexical
  Ex : l'outil lex
- Ou bien en traduisant un automate reconnaissant le modèle des unités

Langages

Définitions

- Un **alphabet** est un ensemble de caractères. Ex : {0, 1}, {A, C, G, T }.
- Une **chaîne** ou un **mot** sur un alphabet Σ est une séquence finie de caractères de Σ. Ex : 00011011, ACCAGTTGAAGTGGACCTTT, helloworld. La *chaîne vide*, ne comportant aucun caractère est notée ε.
- Un **langage** sur un alphabet Σ est un ensemble de chaînes construites sur Σ.

  Ex : l'ensemble des nombres en notation binaire, l'ensemble des mots français, ...

Langages

Opérations sur les langages

- **Union** de L et M notée L U M : $\{ x \mid x \in L \text{ ou } x \in M \}$
- **Concaténation** de L et M notée LM : $\{ xy \mid x \in L \text{ et } y \in M \}$
- **Notations courantes :**
  - $\mathbf{L^n} = LL \ldots L$
  - $\mathbf{L^*}$, fermeture de Kleene : $\{ x1\ x2 \ldots xn \mid xi \in L, n \in N \text{ et } n \geq 0\}$
  - $\mathbf{L^+}$, fermeture positive : $\{ x1\ x2 \ldots xn \mid xi \in L, n \in N \text{ et } n > 0\}$

Exemples

L est le langage {A, B, . . . Z, a, b, . . . z }, C est est le langage {0, 1, . . . 9}.

A quoi correspond :

- L
- C
- $L \cup C$
- L^5
- $L(L \cup C)^*$

Expressions régulières

Une expression régulière r sur un alphabet Σ est **une formule qui définit un langage** L sur Σ, telle que :

1. Si $a \in \Sigma$, alors a est une expression régulière qui définit le langage {a}

2. Soient x et y deux expressions régulières, définissant les langages L et L'. Alors :

- (x)|(y) est une expression régulière définissant le langage $L \cup L'$
- (x)(y) est une expression régulière définissant le langage LL'
- (x)* est une expression régulière définissant le langage L*
- (x) est une expression régulière définissant le langage L

Note 1 : Priorités entre opérateurs : * > concaténation > |

Note 2 : Les parenthèses dans les expressions régulières permettent de modifier les priorités

Définitions régulières

- Les définitions régulières permettent de donner des noms à certaines expressions en vue de leur **réutilisation** par leur noms dans d'autres expressions.
- Soit les n définitions suivantes :

$$d1 \rightarrow r1$$
$$d2 \rightarrow r2$$
$$\ldots$$
$$dn \rightarrow rn$$

Alors, elles sont dites régulières si :

- Les noms di sont des chaînes sur un alphabet disjoint de Σ,
- Les noms di sont distincts
- chaque expression ri est une expression régulière sur $\Sigma \cup \{d1, d2, .., di-1\}$

- Voir exemples du slide suivant

Exemples

1) Donner l'expression régulière des **identificateurs**
2) Donner l'expression régulière des **nombres**
3) Réecrire les expressions précédentes en utilisant les **définitions régulières**.

Exemple

lettre → A | B | . . . | Z | a | b | . . . | z

Ou lettre → [A – Za-z] Ou lettre → [A, B . . . Z, a, b . . . z]

Chiffre → 0 | 1 | . . . | 9

identificateur → lettre (lettre | chiffre)*

chiffres → chiffre chiffre*

partiedecimale → . chiffres | ε

exposant → (E (+ | - | ε) chiffres) | ε

nombre → chiffres partiedecimale exposant

Automate d'états finis

Un automate d'états fini est défini par la donnée de

– un ensemble fini d'états E,

– un ensemble fini de caractères d'entrée Σ (alphabet)

– une fonction de transition, transit : E × Σ → E,

– un état initial,

– un ensemble d'états F, appelés états d'acceptation ou états finaux

Automate d'états finis

- Un automate d'états fini **accepte** une chaîne de caractères c1 c2 . . . ck si et seulement si :

  Il existe un chemin joignant l'état initial à un état final, composé de k transitions étiquetées par les caractères c1 , c2 , . . . ck.

Exemples

- Donner un automate d'états finis qui accepte le langage désigné par l'expression régulière : a(b | c)*
- Donner un automate d'états finis qui accepte le langage sur l'alphabet {a, b, c, d} constitué de toutes les chaînes qui contiennent un nombre paire de caractères "a"
- Donner un automate d'états finis qui accepte le langage sur l'alphabet {a, b, c, d} constitué de toutes les chaînes qui contiennent un nombre paire de caractère "a" et aucun "d"

Exemples