Théorie des Langages

Introduction

Claude Moulin

Université de Technologie de Compiègne

Printemps 2013

Sommaire

Sommaire

Site NF11

- Site Moodle NF11
- Site NF11 : `http://www4.utc.fr/~nf11/`
  - Planning Cours, TD, TP.
  - Supports de cours
  - sujets TD
  - sujets TP
  - sujets d'examens.
  - Documentation et liens

NF11 - Questions

- Comment analyser un programme écrit dans un langage particulier ?
- Comment écrire un analyseur (parser) pour un langage ?
- Quels sont les concepts théoriques ?
- Comment les mettre en oeuvre en pratique ?

Phases d'analyse

Bases de la théorie des langages de programmation

- Analyse lexicale : grouper les symboles d'un texte en unités lexicales
- Analyse syntaxique : définir les suites d'unités qui sont valides
- Analyse sémantique : donner un sens aux instructions d'un langage

Outils théoriques

- Définir un langage
  - Expressions régulières
  - Grammaires
- Analyser des chaînes
  - Automates
  - Automates à pile
- Représenter en mémoire un programme
  - Structure intermédiaire : Arbre syntaxique

Réalisation pratique

- Familiarisation avec les expressions régulières
- Générateur de parser de programmes
  - Framework ANTLR - v. 4.0
- Langage de programmation
  - Langage Java
    - Héritage
    - Polymorphisme

REALISATION

- Parser d'un langage graphique particulier à partir d'une grammaire
- Application : interpréteur
  - Interpréteur - Visite d'arbres.
  - Génération de code.

Sommaire

Historique - 1

- Avant 1950
  - Langage machine : suite de nombres écrits dans le système hexadécimal. Certains nombres représentent des instructions, d'autres des valeurs, etc. (0A43)
  - Langage d'assemblage : il contient des instructions faisant intervenir les registres de l'unité centrale de l'ordinateur, les données.
    (Ex : ADD #2, R1)
  - Introduction des étiquettes dans le langage d'assemblage dans le but de faciliter les branchements (sauts, boucles).
  - Introduction des variables.
    (Ex : MOV R1, b)

Historique - 2

- 1950 : Définition des Langages de haut niveau
  - expressions
  - fonctions
  - structures de données
- 1970 : Définition des langages à objets
  - Classes, objets, méthodes, variables d'instance
  - Héritage
  - Polymorphisme

Sommaire

Compilateur

- fichier source 1 $\longrightarrow$ compilateur $\longrightarrow$ fichier objet 1
  fichier source 2 $\longrightarrow$ compilateur $\longrightarrow$ fichier objet 2
- fichier objet 1, fichier objet 2
  $\longrightarrow$ éditeurs de liens $\longrightarrow$ programme exécutable
- L'exécutable ne nécessite pas la présence du compilateur.
- Langages compilés : Pascal, C, C++, ADA, Fortran, Cobol

Interpréteur

- Un interpréteur travaille simultanément sur les données et sur le programme source. Il analyse une à une les instructions du programme, les “compile” et les exécute.
  - L’interpréteur doit être présent sur le système où le programme s’exécute
  - L’exécution est plus lente
  - On peut modifier le source pendant l’exécution
- Langages interprétés : Basic, Lisp, Scheme, Tcl, Perl, Prolog, Smaltalk

Langage intermédiaire

- Le code source est d'abord traduit dans une forme binaire intermédiaire (pseudo-code).
- Pour l'exécution :
  - ce pseudo-code est chargé dans un environnement, et interprété.
  - Le code objet est portable d'une plate-forme à une autre. C'est l'interpréteur qui est adapté à la plate-forme.
- Exemples de langages intermédiaires : Java, Python.

Cas de Java

- Compilation :
  - fichier. source (.java) $\rightarrow$ compilateur (javac) $\rightarrow$ fichier objet (.class)
  - Les fichiers objet (et les ressources) peuvent être regroupés dans des fichiers archives (.jar).
- Exécution :
  - Les fichiers objet $\rightarrow$ interpréteur (JVM) $\rightarrow$ résultat.

Avantages de Java

- Portabilité d'une plate-forme à une autre
- Il est possible de charger du code objet dans la JVM (sur PC), localement, ou à partir d'un site distant.
- Il est possible à partir d'une application java, de créer du code source, de le compiler dynamiquement, de le charger et de l'exécuter.

Sommaire

Phases théorique de la compilation

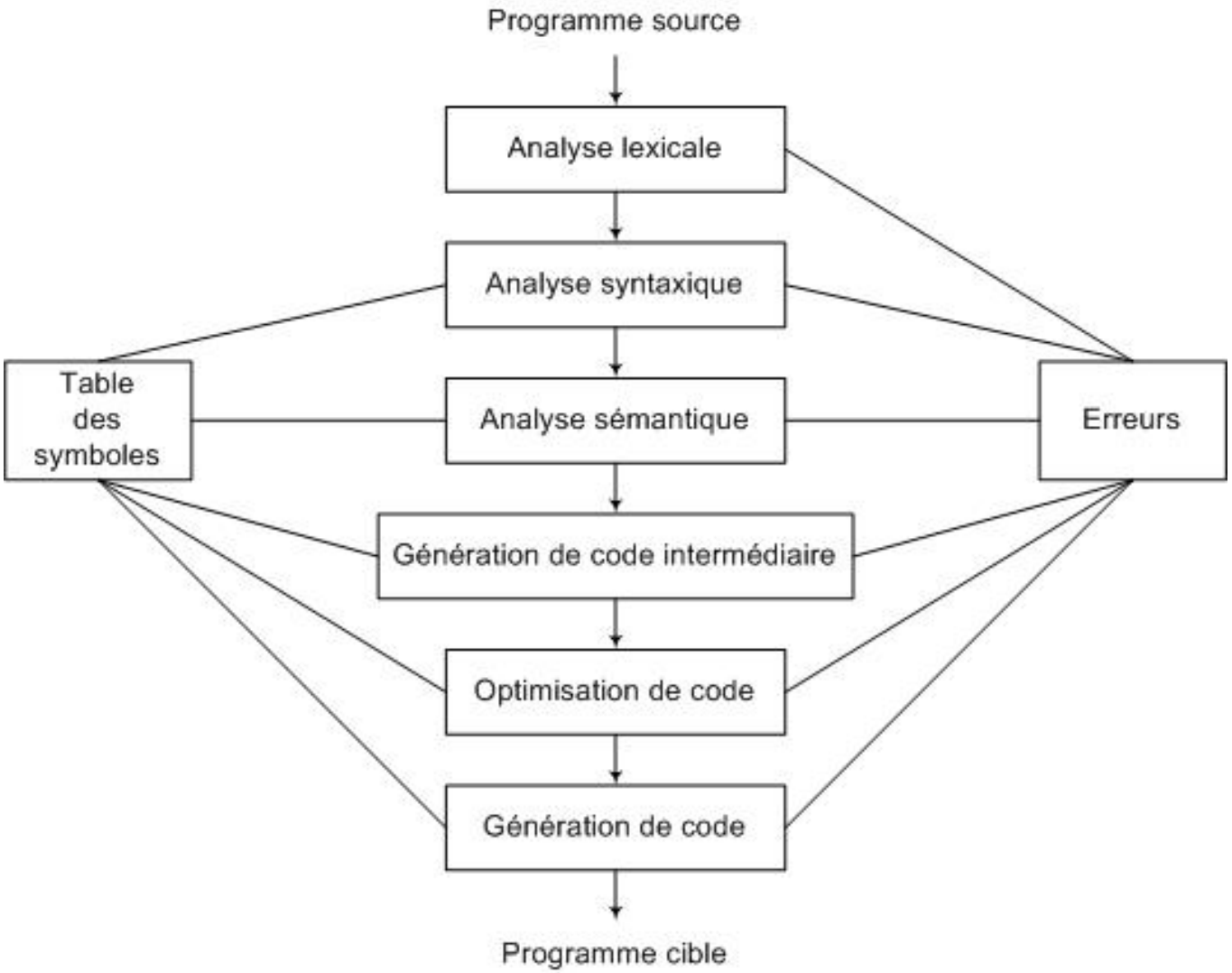

Schéma général

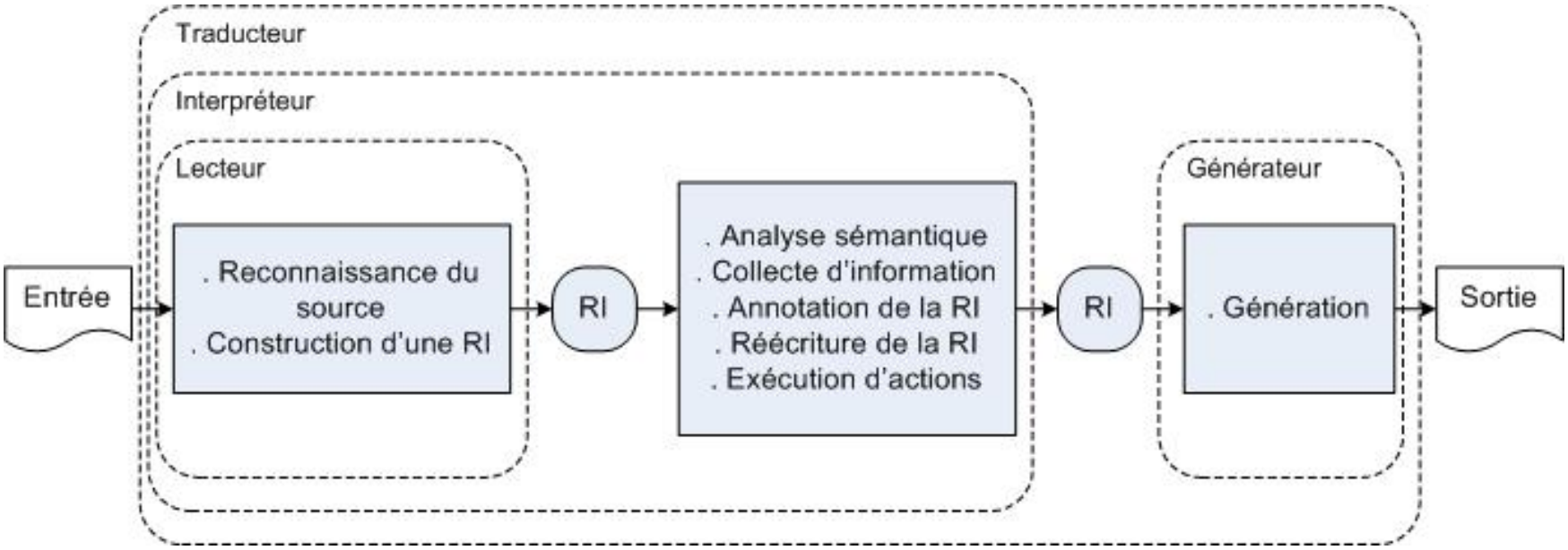

Schéma général

- On considère 4 types de composants qui peuvent intervenir dans une application se rapportant au traitement d'un langage.
  - Lecteur
  - Générateur
  - Traducteur
  - Interpréteur

Lecteur

- Un lecteur prend en entrée un ou plusieurs flux (généralement du texte mais aussi des données binaires).
- Un lecteur généralement construit une représentation intermédiaire (RI) du flux d'entrée. C'est une structure de données classique (arbre, liste, etc.).
- Exemple : un couplage d'un analyseur lexical et d'un analyseur syntaxique crée un arbre représentant le programme source.

Générateur

- Un générateur parcourt une structure de données et génère un flux de sortie.
- Exemples :
  - Sérialiseurs d'objets.
  - Générateurs de code source.
  - Générateurs de page Web.

Traducteur

- Un traducteur lit (il inclut un lecteur) un flux d'entrée (text, code binaire) et émet un flot de sortie conforme à un certain formalisme.
- Exemples :
  - Traducteur d'un programme écrit dans un certain langage en un programme conforme à un autre langage.
  - Pré-processeur de macro.
  - Assembleur.
  - Compilateur.

Interpréteur

- Un interpréteur lit un flux d'entrée, le décode et exécute des instructions.
- Exemples :
  - Calculateur.
  - Implémentations pour Java, Ruby, Python.

Types d'activités

- Analyser des phrases d'entrée.
- Construire des arbres.
- Parcourir des arbres.
- Comprendre ce que signifient les éléments en entrée.
- Créer une interprétation des phrases d'entrée.
- Traduire d'un langage dans un autre.