Langage Vyking

Lexing et parsing

Compilateurs

INFO0085

PR. PIERRE GEURTS

CYRIL SOLDANI

27 mars 2013

ROBIN KEUNEN s093137
PIERRE VYNCKE s091918
1ère Master Ingénieur Civil

1 Introduction

2 Trois versions de Vyking

Pour séparer les difficultés dans la compilation du langage, nous travaillerons sur trois versions successives du langage. La première, *basic_vyking* n'implémente que les fonctions de bases, le noyau du langage. La deuxième, *listed_vyking* rajoute les listes et les opérateurs de liste. La troisième et dernière version *full_vyking* implémentera les closures.

3 Choix des outils

Nous avons isolé quelques lexer/parser dans la multitudes des outils disponibles.

ANTLR ANTLR est capable de générer du code Python pour le Parser, il dispose de beaucoup de documentation. L'analyse est top-down, ce qui rend la génération d'erreur plus facile. Cependant il vise les grammaires LL, nous ne voulions passer trop de temps à modifier la grammaire. De plus, en lisant sur le web, il apparait que le code généré cause beaucoup d'appels de fonctions[1]. Les appels sont bon marché en Java, le langage de base de ANTLR mais les appels sont chers en Python. D'autres outils sont plus efficaces et orientés Python.

Plex Plex est assez populaire mais cet outil ne fait que l'analyse lexicale. Nous préférons une solution intégrée pour ne pas devoir apprendre deux outils.

PLY PLY est une implémentation de lex et yacc pour Python. PLY est construit en Python pur, il n'est pas construit sur un noyau C. Cette approche le rend plus lent que les outils construit en C mais il est plus efficace (en temps et espace)[2] que tous les autres outils Python. Il existe beaucoup de ressources en ligne et l'outil fournit des outils de diagnostique. Le report d'erreur était un des objectifs de développement pour PLY. D'un point de vue pédagogique, nous avons retenu ce choix car il nous permet de nous familiariser avec Lex et Yacc tout en conservant une approche pythonesque.

Nous avons retenu PLY, si nous avons assez de temps, nous essaierons de développer nos propres outils de lexing et/ou de parsing

4 Lexing de l'indentation

Dans une syntaxe à la python, les espaces et l'indentation n'est significative qu'en début de ligne. Les transitions sont illustrées à la figure ? ? ? (CF EVERNOTE bol state transition). L'état `bol` est enclenché quand le lexer lit un retour à la ligne. Une fois dans l'état `bol`, 3 actions sont possibles :

1. Lecture d'espace ou de tabulation : le token `INDENT` est émis, sa valeur est le nombre d'espaces. Une tabulation vaut 4 espaces par défaut.
2. Lecture d'un retour à la ligne, le lexer reste dans l'état `bol`.
3. Lecture d'un autre caractère, retour à l'état précédent.

5 Filtre indentation

6 Remarques

Les 3 étapes lexing, filtre et parsing sont découplées. Mais grâce à l'utilisation de générateur/itérateurs pour construire les classes `Lexer` et `IndentFilter`, les performances sont améliorées. En effet, au lieu de passer tout le programme dans le `Lexer` et de conserver tous les tokens en mémoire, puis de faire un passe complet à travers le filtre, le `Lexer`génère les tokens un à un et le filtre ajoute les indent et dedent au vol.

10ème règle de greespun

Références

[1] more antlr - java, and comparisons to ply and pyparsing. http://www.dalkescientific.com/writings/diary/archive/2007/11/03/antlr_java.html, 2007.

[2] David Beazley. Writing parsers and compilers with ply. http://www.dabeaz.com/ply/PLYTalk.pdf, February 2007.