Compilateur Mini Python vers Mips en Racket

Diana Heddadji
Licence 3 - Université Paris VIII Saint-Denis Vincennes
UFR : MISTIC
30 décembre 2019
Numéro étudiant : 16712076

SOMMAIRE

ORGANISATION DU PROJET

Présentation

Ce projet consiste à créer un compilateur, autrement-dit, un programme capable de transformer le code source qu'on lui donne en un code objet qui pourra être exécuté par la machine.

Pour cela, on aura besoin, de Spim, (un simulateur MIPS), de Racket, et de Python pour exécuter les fichiers tests portant l'extension *.py.

SPIM

« SPIM est un simulateur de processeur MIPS, conçu pour exécuter du code en langage assembleur pour cette architecture. » -Wikipedia

On trouvera dans notre code, plusieurs lignes de texte écrites dans ce langage.

Racket

« Racket est un langage de programmation de la famille Lisp. Il fait partie du projet Racket (autrefois PLT Scheme), qui regroupe plusieurs variantes du langage Scheme ainsi qu'une série d'outils pour les utiliser. L'un de ses objectifs est de servir de plate-forme pour la création, la conception et l'implémentation. » -Wikipedia

Python

« Python est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl. »

Le rôle des dossiers/fichiers

Racket

Dans notre dossier Racket, nous allons trouver plusieurs fichiers ayant chacun un rôle précis dans la construction du projet.

- **ast.rkt**

  Ce fichier est une sorte de « header », puisqu'il contient tous les prototypes des structures de l'arbre de syntaxes abstraites.

- **compiler.rkt**

  Ce fichier contient les règles pour traduire le code source en instructions Mips. Par soucis de temps et de manque d'organisation, il contient également une grande partie d'affichage.

- **lexer.rkt**

  Ce fichier permettra de faire notre analyse lexicale en décomposant le code en un ensemble de lexèmes (dit tokens).

- **parser.rkt**

  Ce fichier déclare les modules utiles pour réaliser notre analyse syntaxique.

- **liec.rkt**

  Le fichier principal, c'est celui-ci qu'on appellera sur nos fichiers tests. Il se chargera ainsi de générer un programme mips, portant l'extension *.s, exécutable sur le simulateur SPIM.

Test-Python

Ce dossier contient tous les fichiers tests que notre programme est capable d'interpréter ainsi q'une gestion d'erreurs.

README.md

Ce fichier regroupe des instructions de compilation et d'exécution afin que l'utilisateur sache comment fonctionne le projet.

UNE APPROCHE VERS LE CODE

Les fonctions pertinentes

Certaines fonctions comportant plusieurs instructions ou peu mais qui restent complexes dans leur définition, méritent une petite explication.

Étant donné que l'implémentation des fonctions habituelles pour le parser ou pour le lexer ne changent pas, on s'intéressera davantage à quelques unes du fichier compiler.rkt.

- **compile-function**

  Construit les fonctions (des fichiers tests), de manière à ce qu'elles soient placées avant le main.

- **estimation**

  Cette fonction permet de savoir si le programme comporte une seule ou plusieurs instructions

- **data-expr**

  Définie les data expressions

- **comp-op-expr**

  Compile les opérations et les expressions.

- **print-instr**

  Permet d'afficher chaque instruction Mips.

- **label-location**

  Gère les labels et les emplacements dans la pile.

- **match-prog**

  Cette fonction match les programmes contenant un if ou un while, elle gère également la définition des opérations, des retours de fonctions ou de leur définition...

- **data-match-expr**

  Affiche les .data correspondant aux instructions du programme.

- **register-argument**

  Affecte à chaque module d'une fonction, le registre $a qui lui correspond.

- **compile-expr**

  Cette fonction traduit les expressions du langage Python en Mips.

DÉMONSTRATION

Compilation

Pour compiler notre programme, il suffit d'exécuter ceci :

- `$ racket liec.rkt test-Python/*.py`
- `(Avec « * » remplacé par le nom du fichier test)`

Ceci générera un fichier portant l'extension .s compilable et exécutable sur le simulateur SPIM.

Exécution

Pour exécuter le fichier généré il suffit de lancer SPIM à l'endroit où se situe notre fichier et lancer les commandes suivantes :

- `$ spim`
- `$ load "arithmetic.py.s"`
- `$ run`

Ce projet est capable de reconnaître les opérations arithmétiques de base, les comparaisons, les opérations logiques, les boucles (for-while), les décalages de bits, les booléens, les déclarations de variables et de fonctions, les fonctions d'affichage, les appels de fonctions, les fonctions min et max et une petite gestion d'erreurs.

Ce qu'il manque

- Reconnaître plus de modules du langage python, les pointeurs, les déclarations de classe (gérer les instances de classe, getter, setter, constructeur...), les exceptions...
- Gérer plus de cas d'erreurs
- Programmer de manière plus modulaire (notamment pour le fichier compiler où il aurait fallu créer un autre fichier mips-printer.rkt pour l'affichage et analyzer.rkt pour l'analyse sémantique)