Python Semantics in K 5.0

Qianyang Peng, Wenhua Lin

December 17, 2018

Abstract

This is the report of the final project of CS522: Programming Language Semantics. We implemented a simple semantics for the Python Language, based on the Python Semantics implemented by Dwight Guth on K 3.2.1, and the untyped KOOL semantics for K 5.0.

1 Introduction

For the course project of CS522, we implemented a simple python semantics for K framework 5.0. The structure of the whole project is based on the *Python 3.3 Semantics* implemented by Dwight Guth on K 3.2.1. That is, we have a Makefile for compilation, a python script for source code preprocessing, and a bash file for script execution. For the implementation detail, including the K configuration, basic infrastructure and the implementation of some functionalities, our code is based on the *KOOL* project for K 5.0, which is a small object oriented programming language syntax implemented by K team.

We implemented the following python functionalities:

1. Python code parser;
2. Python arithmic operator syntax and logical operator syntax;
3. Pythonic variable declaration;
4. Pythonic class;
5. Pythonic class instantiation;
6. Pythonic for loop and range;
7. Pythonic list and nested list;
8. Python print;
9. A test module that compares the output of our python framework with real python.

2 K Framework Syntax Overview

This section is a summary to the K tutorial.

2.1 Syntax

2.1.1 Attributes

left : To tell the parser that the syntax production is left associative.

bracket : To tell the parser that the parenthesis is associated to the program.

binder : To tell the parser that "lambda" or "mu" binds to variables.

strict : To tell the parser that to tell k the corresponding construct has a strict version strategy. That is, all arguments should be evaluated before the semantic rule is applied.

strict(i) : Selective strictness. To tell the parser that which argument of syntax production is strict.

seqstrict : To tell the parser that the evaluation order is from left to right.

2.1.2 Predefined constant identifiers

Predefined constant identifiers are variables starts with $. The program to execute cannot contain identifiers with these names and these names are specifically reserved for the semantics.

2.2 Require and Imports

"Require" takes a .k file containing the language features needed for the definition. The path is relevant to the k/include folder.

"imports" keyword can import another module.

2.3 Rules

Rules are introduced by the keyword "rule".

2.3.1 Variables

By convention, variables in the rule should start with a upper case letter. Variables could be tagged with the type associated to them. If tagged, a type checking at run time will be made. Tagging is formally saver but may make the executing speed slower.

2.3.2 Symbols

- "=>": Used in rules to separate the left hand side and right hand side of a rule.
- "~>": This is a construct for K used in a computation structure, or computation. Read as "followed by" or "then".

2.3.3 Attributes

structural : To tell the parser that a rule is desugaring but not a computational step. It is a light rule similar to "macros".

heat : To tell the parser that a rule is a heating rule.

cool : To tell the parser that a rule is a cooling rule.

strict : Same as in the Syntax section.

seqstrict : Same as in the Syntax section.

2.3.4 Side conditions

Side conditions can only be bool expressions in k. It is declared after a K rule using the "when" keyword.

2.3.5 Anonymous variables

Anonymous variable is a "_" when you want to show there is a variable but you do not care about what it is.

2.3.6 Cell Inside Rules

- Cell $< k > ... < /k >$ holds computations, which is a list of " ¿" connected sequences of tasks.
- Cell $< state > ... < /state >$ holds map, which is a set of bindings defined by " Variable —-¿ Target.

2.3.7 Closures

Closures are like a lambda expression but also holds the environment corresponding to the environment.

2.4 Modules

Using different modules can help to grouping language features. Usually we put the language syntax and semantics in separate modules.

2.5 Results

Result identifies the result of computation and stops K from continuing evaluating. One way to do it is to use the built in category "KResult" to tell the parser which are results.

2.6 Builtins

Some builtins, for example, Int and Bool are accessible by default in earlier versions of K. However in K5.0, they are not accessible by default and we need to import the corresponding modules in order to make use of them.

2.7 Configurations and Cells

Configuration are constructed with cells. It starts from a "configuration" keyword and being followed by some XML-style labels which indicates the start and return point of cells.

Cells:

- $< T >< /T >$ The top cell.
- $< k >< /k >$ The program.(maybe?)
- $< env >< /env >$ The environment binds variables to locations.
- $< store >< /store >$ The store binds locations to values.

Some identifiers could be put in the cell. For example:

- $PGM is the program initially passed to the k to run.
- Dot . stands for nothing. .Map stands for an empty map.