TAPY

Type Analysis for Python

Jesper Lindstrøm Nielsen - *jesper.jln@gmail.com*
Christoffer Quist Adamsen - *christofferqa@gmail.com*
Troels Leth Jensen - *troelslethjensen@gmail.com*

Aarhus University
May 3, 2013

Abstract

TODO

1 Introduction

Python is a dynamically typed, general purpose programming language that supports both object-oriented, imperative and functional programming styles. As opposed to most other programming languages Python is an indented language, with the intention to allow programmers to write more concise code.
Because of its dynamic nature and little tool support it can be difficult to develop and maintain larger programs. In this report we present our work towards developing a conservative type analysis for Python in Scala.

2 Dynamic features in Python

As mentioned Python is a dynamically typed language, and therefore has a lot in common with e.g. JavaScript. In this section we present some of its interesting dynamic features, together with a bunch of common runtime errors.

Classes are declared using the `class` keyword, supports multiple inheritance and can be modified further after creation. The code in Figure 1 1 declares an empty `Student` class that inherits from `Person`, which in turn inherits from `object`. In line 14 a function `addGrade` is added to the `Student` class and in line 16 the attribute grades is set to an empty dictionary on the $s1$ object. Therefore we can call the `addGrade` function on the `s1` object without getting a runtime error. However, since we forgot to set the grades attribute on the $s2$ object, we get the following runtime error from line 19: `AttributeError: 'Student' object has no attribute 'grades'`.

Notice that the receiver object is given implicitly as a first argument to the addGrade function. In case we had forgot to supply the extra formal parameter `self`, the following runtime error would result from line 18: `TypeError: addGrade() takes exactly 2 arguments (3 given)`.

Another interesting aspect with regards to parameter passing is that Python supports unpacking of argument lists. For instance we could have provided the arguments to the addGrade function in line 18 by means of a dictionary instead: `s1.addGrade(**{ 'course': 'math', grade: 10})`.

```
1          class Person(object):
2            def __init__(self, name):
3              self.name = name
4          class Student(Person):
5            pass
6          s1 = Student('Foo')
7          s2 = Student('Bar')
8          def addGrade(self, course, grade):
9            self.grades[course] = grade
10         Student.addGrade = addGrade
11         s1.grades = {}
12         s1.addGrade('math', 10)
13         s2.addGrade('math', 7)
```

Listing 1: Figure 1

Unlike as in JavaScript, we wouldn't be able to change line 16 into `s1['grades'] = {}`. This would result in the following error: `TypeError: 'Student' object does not support item assignment`, while trying to access `s1['grades'] would result in the following error: TypeError: 'Student' object has no attribute '__getitem__'`. Instead it is possible to call the built-in functions `getattr(obj,attr)` and `setattr(obj,attr,val)`.

But Python also allows the programmer to customize the behavior when indexing into an object by supplying special functions `__setitem__` and `__getitem__`, giving the programmer much more control. As an example consider the new Student class in 2. With this implementation we could set the grades attribute as in JavaScript: `s1['grades'] = {}`, and e.g. get the grade of `s1` in the math course by calling `s1.math` or `s1['math']`.

```
1          class Student(Person):
2            def __getitem__(self, name):
3              return self.grades[name]
4            def __setitem__(self, name, val):
5              setattr(self, name, val)
6            def __getattr__(self, name):
7              if name in self.grades:
8                return self.grades[name]
9              else:
10               return "<no such grade>"
```

Listing 2: figure 2

Bibliography

[1] Elkan, Charles, http://cseweb.ucsd.edu/~elkan/250B/learningmeaning.pdf, February 27, 2013