rayalas@oregonstate.edu

CS 562: Applied Software Engineering

Final Report

Swathi Sri Vishnu Priya Rayala – 932-696-872

The library I have tested is the pyparsing. This module provides a library of classes that client code uses to construct the grammar directly in Python code. It is an alternative approach to create and execute simple grammars vs. the traditional lex/yacc approach, or the use of regular expressions. The pyparsing module provides a library of classes that client code uses to construct the grammar directly in Python code. As a part of class project I tested few grammars for strings, arithmetic operations and SQL Queries.

Calculator Module

This module evaluates expressions that contain numerals, variables and equations. I have created a function to generate random expressions with numerals, variables and mathematical constants using the operators - +, *, -, /, ^. All the expressions with the first four operators worked fine with a reasonable performance but for the expressions that included ^, there is a performance degradation while evaluating the expression. This is because of not having a restriction on the numbers to which a base number is raised to.

Following are the test cases implemented using TSTL –

Random expressions are generated and tested correctly

Lengthy expressions are evaluated correctly

The five operators (+, -, *, /, ^) are used for testing

Expressions with mathematical constants (e and pi) are tested

Assignment operations like “area = 3 * 4” are verified and evaluated correctly.

Few Bugs found –

For example, expressions like ((3 + pi) * e) - 12 are evaluated correctly. However, expressions like (a + 3) * 4 are not evaluated correctly. The grammar interprets the variables a, b, etc as zero. But, the variables like e and pi are taken as 2.71828182846 and 3.14159265359 respectively. This is something weird as the grammar should give an error rather than taking the value as zero.

Also, the expressions with only 2 mathematical constants (e and pi) are interpreted and evaluated correctly. The mathematical constant i or i2 is not accepted by the grammar. It is however evaluated as zero.

Below is the screenshot of the wrongly evaluated expressions –

Code Coverage –

By trying with various inputs, I tried to cover all functions of the code. Test cases that would raise exceptions are also tested. Also verified whether the exceptions raised are handled successfully by the system under testing. Inputs that include integers, floating point numbers and string variables are tested and verified if they are accepted the system under test or not.

74.7663551402 PERCENT COVERED

84 BRANCHES COVERED

66 STATEMENTS COVERED

Simple SQL Module

The module defined grammar that accepts only the SELECT queries of SQL. This module outputs the column names, table names and the WHERE conditions of a given SQL Query.

Following are the test cases implemented using TSTL –

SQL queries with different column names are tested

SQL queries with WHERE conditions are verified

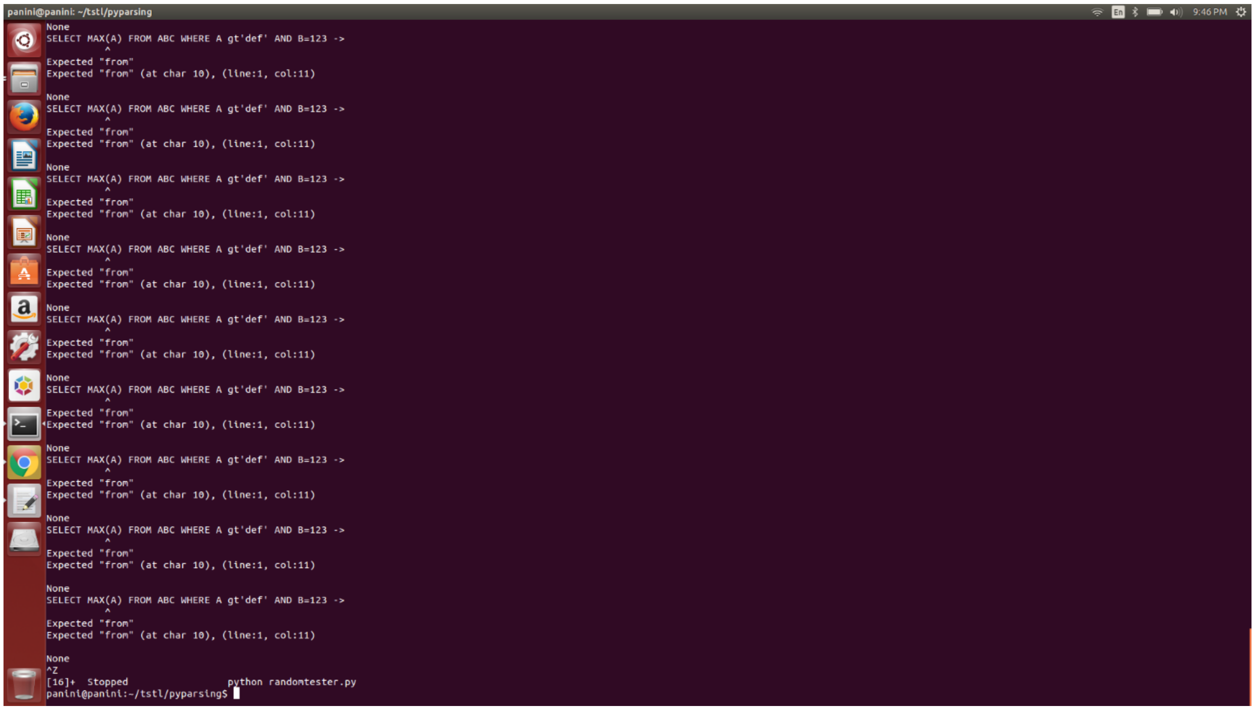
Few bugs found –

For queries like, *SELECT A, B, C from ABC_Table* are successfully executed by the grammar by displaying the column names as [A, B, C] and Table name as ABC_Table

But for queries like *SELECT MAX(A) from ABC_Table*, the grammar failed to display the column names. However, it is successful to display the Table Name.

rayalas@oregonstate.edu

Below is the screenshot of the error which cannot display the column tokens correctly –



Code Coverage –

By trying with various inputs, I tried to cover all functions of the code. Test cases that would raise exceptions are also tested. Also verified whether the exceptions raised are handled successfully by the system under testing. Column names that include * is verified. The WHERE conditions are tested with multiple number of conditions. Also, the operators used in the WHERE conditions are verified. For example, columnA > column is verified against columnA gt column.

82.9268292683 PERCENT COVERED

43 BRANCHES COVERED

41 STATEMENTS COVERED

How well does the tester work?

The random tester of TSTL was designed well. The errors thrown by the random tester for the system under test were self-explanatory. The sequence of the operations that it outputs for initializations and assignment of variables is very helpful. So whenever an error is thrown by the code we test, the record of the flow of operations would let us know why the test case was failed.

It is with the help of random tester, we can test a particular test case with a number of randomly generated inputs. Thus, we need not concentrate on whether all the inputs are included in the test or not. By just mentioning the character set, all the possible inputs can be tested across the test case defined.

What is wrong with TSTL?

As of now TSTL supports only for Python. It would be great if TSTL supports for other languages too like Java, C and C++. Also, it would be very helpful if we could find some documentation for TSTL regarding it's usage with strings and numerals, the features available in TSTL, the syntax on how a TSTL file should be, documents explaining guards, actions, pools, properties, reference variables etc.

What is good about TSTL?

It is a good start for automated testing with TSTL for a naïve programmer who hasn't had any idea of automated testing. Being compatible with python is an advantage as it is an easy to learn language and any one can easily accustom to TSTL. Apart from these, TSTL provides the facility to log the errors that we encountered while testing. Also, the delta-debugging feature was very useful as it would reduce the number the steps that are required to reproduce an error. The

other good thing about TSTL is the “guards” – to limit the conditions in which an action should be enabled. This feature is necessary as the functions need not be executed every time which would unnecessarily increase the number the steps while testing for an error.

In summary, I really enjoyed testing with TSTL. As it is a new tool for me to work on, there is a lot of scope to explore the tool in order to know what it can do.