Demo: TSTL: a Language and Tool for Testing

Alex Groce, Jervis Pinto, Pooria Azimi, Pranjal Mittal
Oregon State University
agroce@gmail

ABSTRACT

Writing a test harness is a difficult and repetitive programming task, and the lack of tool support for customized automated testing is an obstacle to the adoption of more sophisticated testing in industry. This paper presents TSTL, the Template Scripting Testing Langauge, which allows users to specify the general form of valid tests for a system in a simple (but expressive) language, and tools to support testing based on a TSTL definition. TSTL is a minimal template-based domain-specific language, using the source language of the Software Under Test (SUT) to support most operations, but adding declaratve idioms for testing. TSTL compiles to a common testing interface that hides that details of the SUT and provides support for logging, code coverage, delta-debugging, and other core testing functionality, making it easy to write universal testing tools (such as random testers or model checkers) that apply to all TSTL-defined harnesses. TSTL is currently available for Python, but easily adapted to other languages as well.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability

Keywords

Domain-specific languages, testing tools

1. INTRODUCTION

Automated testing often requires a user to write a *test harness* — essentially a program that defines and generates the set of valid tests for the Software Under Test (SUT). Such harnesses are common to random testing, many kinds of model checking, and various machine-learning influenced approaches [1]. Unfortunately, these harnesses themselves are complex software artifacts and it is all too easy to spend valuable testing time hunting down bugs in the test harness and not the SUT. Harness code is often highly repetitive (choosing between a set of available API calls to make, and assigning values to parameters in those calls, for example) and is almost always tightly coupled to one particular test generation method. TSTL [2] is a language (and tool) intended to make these difficulties less onerous.

First, TSTL allows a harness to be defined in a declarative style, and provides support for many common testing idioms. Second, the declarative approach allows TSTL to output a generalized interface for testing: a class that allows a testing tool to determine available test actions, check for success or failure of tests as they progress, and even automatically produce logging, collect and analyze code coverage, and delta-debug failed tests.

ISSTA '15 Baltimore, MA USA

```
@import avl
@import math

@<
def heightOk(tree):
     h = tree.tree_height()
     l = len(tree.inorder_traverse())
     if (l == 0):
        return True
     m = math.log(l,2)
     return h <= (m + 1)
 def it(s):
     l = []
     for i in s:
        l.append(i)
     return sorted(l)
@>

source: avl.py

pool: %INT% 4
pool: %AVL% 2 REF
pool: %LIST% 1

log: 3 %AVL%.inorder_traverse()

property: heightOk(%AVL%)
property: %AVL%.check_balanced()

%LIST%:=[]
~%LIST%.append(%INT%)
%INT%:=%[1..20]%
%AVL%:=avl.AVLTree()
%AVL%:=avl.AVLTree(%LIST%)
~%AVL%.insert(%INT%)
~%AVL%.delete(%INT%)
~%AVL%.find(%INT%)
%AVL%.inorder_traverse()

reference: avl.AVLTree ==> set
reference: insert ==> add
reference: delete ==> discard
reference: find ==> __contains__
reference: (\S+)\.inorder_traverse\(\) ==> it(\1)

compare: find
compare: inorder_traverse
```

2. REFERENCES

[1] A. Groce and M. Erwig. Finding common ground: choose, assert, and assume. In *Workshop on Dynamic Analysis*, pages 12–17, 2012.

[2] A. Groce and J. Pinto. In *NASA Formal Methods Symposium*, 2015. To appear.