Gmail by Google

Lee Wei Yeong <evandrix@gmail.com>

Project meeting 9 Mar 12pm

Lee Wei Yeong <lwy08@doc.ic.ac.uk> **Fri, Mar 9, 2012 at 1:53 PM**
To: Tristan Allwood <tristan.allwood@imperial.ac.uk>, Prof Susan Eisenbach <s.eisenbach@imperial.ac.uk>
Cc: Jerome de Lafargue <jerome.de-lafargue08@imperial.ac.uk>

Hi,

I'm sorry I missed project meeting today, was up hacking on FYP and a coursework due later today, alarm didn't go off and wasn't meant to sleep that long.

I've just some updates concerning the progress of my project:
- streamlined the codebase into one main class as the single entry point to generate the test cases, incorporated the other modules into this, for example,
  tracing capabilities to measure code coverage, works now as the unit tests are being produced
- output the test cases both as Python unit tests, as well as serialise to JSON to translate to other languages in future
- decided on the next set of test programs, and eventually the target benchmarks for the report - beautiful soup, dexml, numpy, scipy, among others
- quickly fingerprinted them with a tool to measure sloc count
- next set of programs will largely fall into each of these 4 categories:
  - Loops / Python generators (infinite lists): like maybe checking primality
    - I intend to have like a context stack to mark at which level I am, to try and detect for very long/infinite looping
    - can use bytecode to find the loop conditional and test to see if it will ever pass (ie like whether I'm getting closer to terminating at each loop iteration)
  - Conditionals: like the Fizzbuzz program, or Jerome's triangle program
  - Recursive functions: like factorial
  - Classes & objects: my current class_*/BankAccount examples
- at the moment, my present code performing the lazy instantiation measures code coverage incorrectly for the loops (as it counts all the bytecode traversed across all loop iterations),
  and produces meaningless test cases, since these programs happen to not take any parameters, and only print out stuff
- also on these parameters, my current program generates a 0 when requested for an int value, I need to make this better by hunting for "critical" values in bytecode,
  and testing more over the range of ints
- there's a challenge at the moment for code in Python, to spot one-line errors in well known algorithms implemented in Python, I might use that to improve my test generator
- another related idea is to generate my unit tests, make sure the target program passes all of them, then modify the bytecode of the target program, and make sure my unit tests now fail,
  for example modifying the if-else branch of a conditional which is reported to be "covered" as the unit tests are being generated (ie. mutation) - jumble/jester is a related project for java
- also many of these possible benchmark libraries all ship with test cases written (assumed manually), I can look at those to get a hint of useful test cases I should be generating from code,
  and also compare the quality of my unit tests generated with these

So mainly this week was consolidating stuff from the demo, figuring out on paper first viable strategies for improving the test generator before actually implementing them and reading up on
Python's data model (from the official documentation)

As always, the updated code is available at my GitHub repository @ https://github.com/evandrix/Pyrulan

That should be about all for this week.

Many thanks.

Lee Wei
@ http://bit.ly/t4KM7x