An overview of techniques aimed at automatically generating oracles from tests

Luke Chircop

Christian Colombo

Adrian Francalanza

Mark Micallef

Gordon J. Pace

Introduction

- Having software that is robust and error free is important
- In fact, more than 50% of development time is dedicated to testing software
- Although such tests are performed, software will still contain a number of unresolved errors.
- Runtime verification can be introduced but:
  - Applying this technology can:
    - Be time consuming
    - Increase development costs

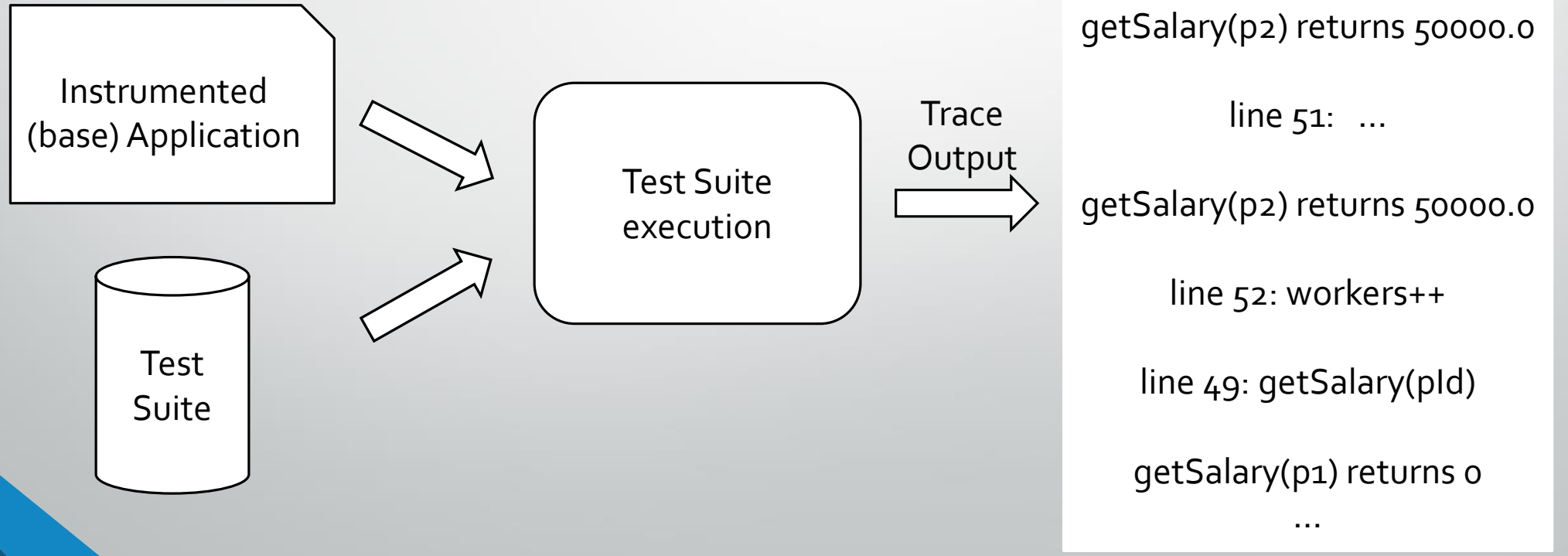
What we have observed

- Tests would have already been written to test software functionality
  - These contain a lot of information such as:
    - The sequence of events expected
    - Number of assertions to carry out after particular behaviour
- Therefore, we are exploring possibility of
  - Extracting such information
  - And automatically generating monitors.

Current work

- We identified some existing work that has been carried out
  - Direct mapping from tests to runtime monitors is not always ideal
    - Due to very specific tests
- Other possible approaches are currently being evaluated:
  - Work carried out by Leonardo Mariani caught our attention
    - Revolves around aiding developers identify the cause of regression test failures by
      - Identifying points of interest
      - Generating models of expected good behaviour
      - Getting trace of failed test and comparing it with models to identify cause of failure

Logging execution trace of tests



Generating expected behavioural models

...
line 49: getSalary(pId)

getSalary(p2) returns 50000.0

line 51: totalSalary addition

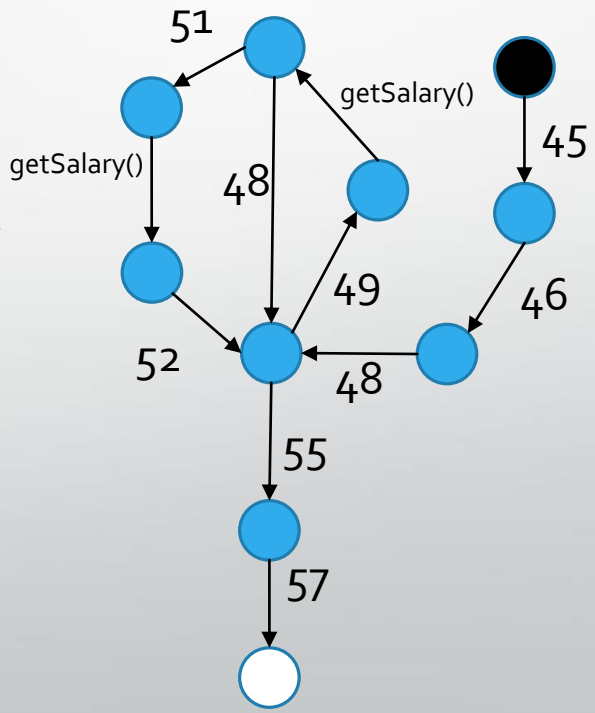
getSalary(p2) returns 50000.0

line 52: workers++

line 49: getSalary(pId)

getSalary(p1) returns 0
...

FSA Model



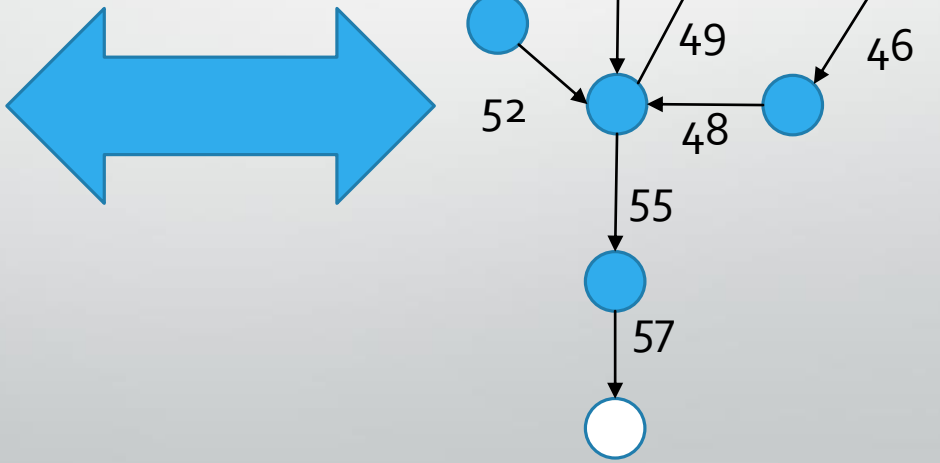
Variable Assertion model

...
MODELS THAT HOLD AT LINE 46
totalSalary == 0
workers !- 0
...

MODELS THAT HOLD AT LINE 49
totalSalary >=0
Workers >= 0
getSalary() return >= 0
...

Identification of failure

...
line 51: totalSalary addition ✓
getSalary(p2) returns 50000.0 ✓
line 52: workers++ ✓
line 49: getSalary(p1) ✓
getSalary(p1) returns -1 ✗
...



...
MODELS THAT HOLD AT LINE 46
totalSalary == 0
workers !- 0
...

MODELS THAT HOLD AT LINE 49
totalSalary >=0
Workers >= 0
getSalary() return >= 0
...

What we are interested in

- This process is of great interest to us especially the:
  - Technique used to extract traces of execution
  - And process of inferring models from the traces gathered

Discussion

- A 5 step process can be considered:
  1) Identify points of interest on software to be monitored by analysing tests
  2) Instrument points of interest with loggers
  3) Run software with tests and gather traces of execution
  4) Process traces to generate models of expected good behaviour i.e.
     a) Execution flow
     b) State of software at points of interest
  5) Use models to generate runtime monitors able to verify behaviour at runtime
- A limitation that we are currently foreseeing is the possibility of having specific tests limiting what the runtime monitor is able to verify

Conclusion

- Introduced the need for software testing and runtime verification
- Identified the problem:
  - Runtime verification may increase development costs which is undesirable
- Discussed a state of the art technique that is able to
  - Extract information and
  - Infer models
  - Can be used for our approach to automatically generate runtime monitors
- Proposed idea:
  - To automatically generate runtime monitors from existing tests
- Evaluated potential limitations of approach

Questions?