Automated Test Data Generation for Dynamically-Typed Programming Languages

Supervisor: Shin Yoo
COINSE Lab, KAIST, South-Korea
Monday 15th May, 2017 -- Friday 4th August, 2017

Simon Bihel `simon.bihel@ens-rennes.fr`

Wednesday 6th September, 2017

University of Rennes I
École Normale Supérieure de Rennes

Automated Test Data Generation

Motivation of Automated Test Generation

Tests are essentials.

But writing them by hand is:

- time-consuming,
- error-prone.

Test Data Generation

Generate a set of inputs to satisfy some metrics for the System Under Test (e.g. a function, a class...).

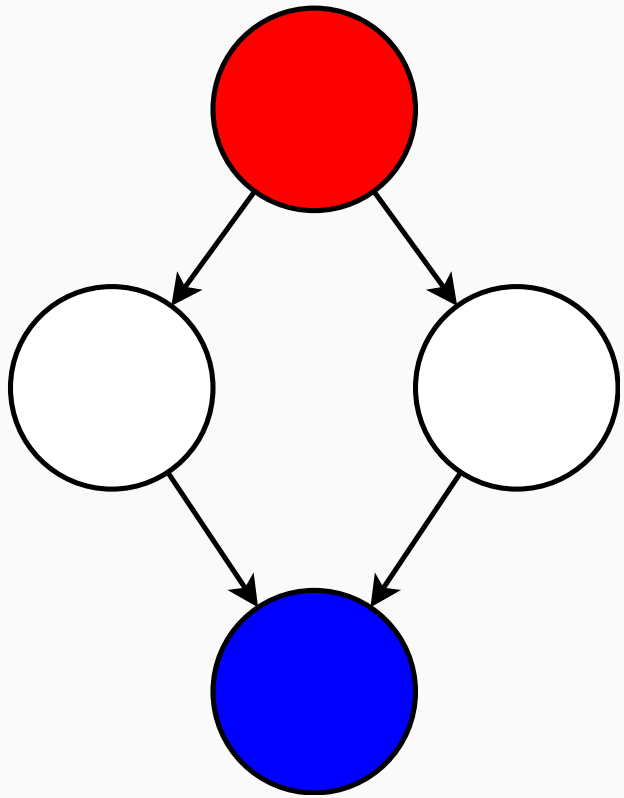
Code coverage reduce the likelihood of crashes. Examples of criteria:

- statement coverage,
- function coverage,
- branch coverage...

Execution Branches

```
int test(char x, char y) {
  if (x==y)
    printf("Equal");
  else
    printf("Not Equal");
  return 1;
}
```

Control flow graph



Search-Based Testing for Branch Coverage

1. Inject code (instrumentation) in the SUT to measure metrics.
2. Execute the SUT and get feedback.
3. Modify inputs and re-execute the SUT.

```
int test(char x, char y) {
  if (branchid(0, Eq(x, y)
    printf("Equal");
  else
    printf("Not Equal");
  return 1;
}
```

Search algorithms include: hill climbing, genetic algorithms...

A Tool for Dynamically-Typed Languages

Background

- Python tool
- Started as a fork of CAVM (a tool for C)
- Alternating Variable Method to modify inputs values

Targets:

- find appropriate types;
- handle Python's lists (multi-types, unbounded).

Big points of the tool

No information on statement raising error.

- When a type error occurs, change type of a random parameter.
- Additional instrumentation on element access to know which list should be expanded.

Another automated tool for dynamic languages without impact because of the narrow test cases

Challenges

Dynamic types problems

Many problems

- Detect type errors bugs
- Relations between parameters
- Working but unfit types (e.g. float instead of int)
- Difficulty to reduce domain of possible types

Often buried in other features of languages

Bigger picture problems

Dynamic languages usually have other paradigms that complexify input space

- Dynamic data structures
- Dynamic insertion of code
- Objects state
- Infinite search spaces
- Understanding the results
- Controlling the environment

Future work

Experiment on libraries with simple inputs like math libraries.

Need of additional methods like:

- first round of static analysis (e.g. gather information on possible types),
- use of existing test suites,
- ...

Conclusion

Conclusion

- Wrote a generic prototype with minimal use of dynamic magic.
- Identified challenges toward a generic tool.
- Automation alone seems unfeasible.