Achieving Scalability in Configurable Software System Testing

By Teng Long, Arun Balasubramanian
Department of Computer Science
University of Maryland
December 6, 2010
Motivation

- Many modern software systems are highly configurable
  - To enable/disable functions (disable_ssh)
  - To select a specific mode (one_process_model)
  - To set a value (timeout)
- Testing all possible combinations is infeasible
  - For an application with 40 boolean configuration options, there are about $1.1 \times 10^{12}$ configurations
- From existing study, only a small set of configurations is sufficient to get the maximum coverage
  - Interaction between configuration options are usually very weak
Configurable Software System Testing Process:

- Configuration Option Study
- Full Configuration Set Generation
- Testing on this Configuration Set

Our Focus
Existing Method

- Learn interaction between options and generate a minimum set of configurations incrementally based on the interactions.

  For example:
  - Setting `Option_A = true` guarantees coverage of line 1-10
  - Setting `Option_B = true` guarantees coverage of line 11-20
  - Setting **Both of them** `true` guarantees coverage of line 1 – 30
  Thus, Option_A and Option_B are in an interaction

- To achieve this, a full symbolic execution (of all options) is run on the test suite.
Disadvantages: Not Scalable!

- Full symbolic evaluation needed to get guaranteed coverage ($O(n\log n)$ time in theory)
- It’s not always feasible to run symbolic evaluation on all options.
  - Limitation of symbolic evaluator (multiple thread)
Objective

- The time-cost of configuration option analysis should be **linear** to the total number of configuration options.
- The **coverage** of configurations generated using our method should **not significantly vary** from the maximum possible coverage.
Approach of Configuration Option Study

- Obtain dependency information from configuration documentations.
- Study the coverage feature of options individually.
Configuration Options Feature Study

- Low-strength interaction between options are given in the configuration documents
  e.g. (an item in the configuration document of vsftpd)

```plaintext
allow_anon_ssl
    Only applies if ssl_enable is active. If set to YES, anonymous users will be
    Default: NO
```
We classify configuration options into the following types:

- **Enabler**
  - When running configuration option A symbolically, if the coverage of all paths can be obtained by a **concrete value** of A, then A is called an **enabler**, and the concrete value of A is called the **enabling value**.

- **Switch**
  - When running configuration option A symbolically, if the coverage of all paths can be obtained only by **more than one concrete value** of A, then A is called an **switch**, and each of the concrete value of A is called a **candidate value**.
Configuration Options Classification

- Classify options as **enabler** or **switch**
  - Step 1: Start by setting all options to its default value
  - Step 2: For each independent option, set it symbolic, and all other options concrete. Run it on the test suite, and obtain coverage for all paths.
  - Step 3: For each option that depends on other options, set the options that are required to concrete values, mark them as enablers; set the option depending on others symbolic. Run it, and obtain coverage for all paths.
  - Step 4: For each symbolic option, if one path’s coverage includes the other paths’, it is an enabler; else it is a switch.
Configuration Set Generation

- Construct **full configurations** with enablers and switches
  
  - Step 1: For each option, if it is an enabler, record its enabling value; if it is a switch, record all candidate values.
  
  - Step 2: Generate a set of configurations. In all the configuration, the enablers are all set as the enabling values, and the switches are set as all candidate values.
Experiment Setting

- Subjective application: vsftpd-2.0.7, a widely used secure FTP daemon;
- Coverage criteria: Line coverage
- Symbolic evaluator: Otter, a symbolic evaluator developed in CS department, UMCP.
Test suite:
A test suite with 64 test cases for vsftpd.

Configuration options of vsftpd:
30 options are studied in this research. All other options are eliminated either because their symbolic execution is not supported by Otter, or that they don’t have an effect on the coverage.
Experiment result

![Graph showing the relationship between time cost (seconds) and number of configuration options. The graph is a straight line indicating a linear increase. The x-axis represents the number of configuration options (1 to 18), and the y-axis represents time cost (0 to 14000 seconds).]
Observations

- The final configurations were generated in linear time.
- The time taken to generate all configurations was nearly half the time taken to run symbolic evaluation on all options.
Conclusion

- By using our method, we are able to finish the configuration option analysis part in linear time;
- The coverage of configuration set does not significantly vary from the maximum possible coverage.