Formal Approaches for Testing against Non-deterministic Specifications & Related Complexity Issues

Natalia Kushik

04, December, 2018
Motivation

- Every information system has software and hardware components (often from different providers)
- The quality of the corresponding components needs to be verified thoroughly
- Tests with guaranteed fault coverage can be derived only when addressing to formal models
- Guaranteed fault coverage requires lots of assumptions
- Testing should be scalable enough

Novel testing approaches should be developed
Complexity should be estimated / decreased
Model based testing

For reactive systems, Finite State Machines are widely used as Formal Models.

Testing steps:
↓ deriving test sequences
↓ applying the test sequences against a given IUT
↓ drawing a conclusion about the correctness of the IUT

A set of software requirements

Test suite
Test sequences
Output response

Implementation under Test

For reactive systems, Finite State Machines are widely used as Formal Models.
Finite State Machine (FSM)

\[ S = \langle S, I, O, h_S, S' \rangle \] is an FSM

- \( S \) is a finite set of states
  \( S' \) is a subset of initial states
- \( I \) and \( O \) are finite input and output alphabets
- \( h_S \subseteq S \times I \times O \times S \) is a behavior relation

Can be
- **Complete or partial**
- **Deterministic** or not
- **Observable** or not
- ... 

Telecommunication protocols can be highly non-deterministic, due to the optionality

Test suite derivation is based on the effective initial / final state identification
Homing sequence (HS)

- \( \alpha \) allows to detect the final state of the machine after \( \alpha \) is applied.

- After applying \( \alpha \) at state \( s_i \) and observing an output response \( \beta_i \), the final state \( s_i' \) becomes known.

Applying \( \alpha \) + observing \( \beta_i \) \( \rightarrow \) drawing a conclusion about \( s_i' \)

- We proposed methods for final state identification for partial and/or non-deterministic specifications [1].
- We also estimated the related complexity [2].
Distinguishing sequence (DS)

- \( \alpha \) allows to detect the initial state of the machine.
- After applying \( \alpha \) at state \( s_i \) and observing an output response \( \beta_i \), the initial state \( s_i \) becomes known.

\[
\text{out}(s_i, \alpha) \cap \text{out}(s_j, \alpha) = \emptyset
\]

- We proposed methods for initial state identification for partial and non-observable specifications.
- We also estimated the related complexity [3].
Bad… very bad ‘news’

Most of the problems in the area of model based testing are NP/PSPACE-complete

- Checking the existence of a distinguishing sequence for complete deterministic FSMs is PSPACE-complete
- Learning a deterministic FSM under a given set of input/output sequences is NP-complete

- Checking the existence of a homing sequence for partial deterministic FSMs is PSPACE-complete [2]
- The length of a distinguishing sequence for a complete non-deterministic FSM with $n$ states is in the worst case $O(2^n)$ [3]
- …
How to decrease the complexity?

Utilizing scalable representations allows to ‘hide’ the complexity
- Logic circuits
- Test Cases as FSMs

Considering specific types of bugs in the software, i.e., specific fault models
- White Box Testing
- User Defined Faults

Switching from preset to adaptive test derivation strategy
- DS for non-deterministic FSMs
- HS for non-deterministic FSMs

Providing effective heuristics
Specific truncating rules for the successor tree + effective traversal

All of them are good for well-defined types of specifications
One of the test generation approaches with the reduced complexity

We propose how to derive tests with the guaranteed fault coverage w.r.t. the fault model \(<S, \approx, FD>\) [4]

\(S\) can be **partial and non-deterministic** (non-observable even) but initialized

\[\downarrow\]

- How to distinguish between \(S\) and a mutant \(M \in FD\)?
  
  \(FD\) is explicitly enumerated \(\rightarrow\) complexity reduction trick

- A DS for the direct sum \(S \oplus M\) can be derived

- \(S \oplus M\) contains all the transitions that are included into FSM \(S\) and FSM \(M\)
DS derivation for a partial non-deterministic FSM

$S' = \{s_1, s_2\}$

- Derive a truncated successor tree (TST)

$\exists \sigma_1 ((s_1, i, o_1, s_1', ) \in h_S \& (s_2, i, o_1, s_2') \in h_S \& s_1' \neq s_2')$

- Truncating rules

**Rule 1** $P$ is the empty set

**Rule 2** Set $P$ contains a subset that labels another node of the path from the root to the node labeled by the set $P$

**Rule 3** $P$ contains singleton

$\alpha$ is a distinguishing sequence iff it labels the path truncated by Rule 1
Deriving a test suite $TS$ w.r.t. $<S, \approx, FD>$

**Input:** FSM $S$ which is initialized

**Output:** A test $TS$ for $S$ or a corresponding message

**Step 1** $i = 0$

**Step 2**

Derive a current mutant $M_i$ for the FSM $S$

Derive a distinguishing sequence $\alpha$ for the pair of initial states of FSM $M_i \oplus S$

If there is no distinguishing sequence for the pair of initial states of the FSM $M_i \oplus S$, then

**Return** a corresponding message

Otherwise,

If $\alpha \notin TS$ then add $\alpha$ into the test suite $TS$

$i++$, and go to **Step 2**
The test suite length can be polynomial

Partial non-observable specification $\xi$

Deterministic/observable FSM $\xi^o$

Maybe, the specification is ‘good’, i.e., allows an observable projection [5]?

Adaptive sequence (of a polynomial length) can be derived for the FSM $M^o \oplus \xi^o$

The number of mutants $|FD|$ can also be polynomial

There can be an exhaustive test suite of a polynomial length w.r.t. $<\xi, \sim, FD>$ [4]
One of the possible applications

- An SDN-enabled switch is an SUT
- How to assure the reliable packet delivery???
- We consider a Fault Model $<\xi, =, FD>$
- The specification $\xi$ is modeled via a logic circuit [6]
- User-Defined faults identify the mutants
- Distinguishing patterns for each mutant form the test suite

Our test suite detects the output and parameter faults in the action / matching part of the rules
Conclusions

- Theoretically: almost all the problems in (software) testing with guaranteed fault coverage have terrible complexity

- Practically: methods and tools for decreasing the complexity seem to be promising

↓

New models need to appear and new methods and tools need to be provided to decrease the complexity

↓

We do have something to tackle as a future work 😊
(Selected) references


7. …
THANK YOU!