Automatic Specification-Based Program Testing

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University, Tokyo, Japan
Email: sliu@hosei.ac.jp
HP: http://cis.k.hosei.ac.jp/~sliu/
Overview

1. Goals of Automatic Specification-Based Testing

2. A decompositional approach to automatic test case generation based on formal specifications

3. Test oracle for test result analysis

4. Conclusion and future research
1. Goals of Automatic Specification-Based Testing

S \rightarrow \text{Transformation} \rightarrow P

Testing
Ideal goal of automatic specification-based testing (ASBT)

Press a Button

Adequate test cases:

```
Method(int x, int y, int z)
int w;
if(x < y)
{
    w = y/x;
    while(w < z)
    {
        ...
    }
} else
{
    ...
}
```

Next
Practical goals of ASBT

1. Every independent function defined in the specification is tested (at least once) (User’s view).

2. Every representative program path is traversed or some required coverage criteria (e.g., MCDC - Modified Condition/Decision Coverage) are satisfied. (Program’s view)
2. A decompositional approach to automatic test case generation based on formal specifications

(1) Strategy and criteria for test case generation

(2) Algorithms for test set generation

(3) “Vibration” method (V-Method) for test set generation from atomic predicates
let denote an operation specification. A set of functional scenarios can be derived from the specification, each defining an independent function in terms of input-output relation.
Scenario-based testing: a strategy for "divide and conquer"

**Specification**

process A(x: int) y: int
pre x > 0
post (x > 10 => y = x + 1) and (x <= 10 => y = x – 1)
end_process

**Program**

int A(int x) {
  if (x > 0) {
    if (x > 10) y := x * 1;
    else y := x – 1;
    return y;
  }
  else System.out.println("the pre is violated")
}

Functional scenario:
~A_pre \land C_i \land D_i
(i=1,...,n)

Functional scenarios

Program paths

M
Definition 1.1 (FSF) Let
\[ S_{\text{post}} \equiv (C_1 \land D_1) \lor (C_2 \land D_2) \lor \cdots \lor (C_n \land D_n), \]
where \( C_i \) is a guard condition and \( D_i \) is a defining condition, \( i = 1, \ldots, n \).

Then, a functional scenario form (FSF) of \( S \) is:
\[ (\neg S_{\text{pre}} \land C_1 \land D_1) \lor (\neg S_{\text{pre}} \land C_2 \land D_2) \lor \cdots \lor (\neg S_{\text{pre}} \land C_n \land D_n) \]
where
\[ f_i = \neg S_{\text{pre}} \land C_i \land D_i \]
is called a functional scenario) and
\[ \neg S_{\text{pre}} \land C_i \]
is called a test condition.
Example:

Test case generation

Specification

```
process A(x: int) y: int
pre x > 0
post (x > 10 => y = x + 1) and
    (x <= 10 => y = x - 1)
```

Functional scenarios:
(1) \( x > 0 \land x > 10 \land y = x + 1 \)
(2) \( x > 0 \land x <= 10 \land y = x - 1 \)

Program

```
y := x * 1
```

Test result analysis

```
y := x - 1
```

```
System.out.println("the precondition is violated")
```
Test strategy:
Let operation $S$ have an FSF
$(\sim S_{\text{pre}} \land C_1 \land D_1) \lor (\sim S_{\text{pre}} \land C_2 \land D_2) \lor \cdots \lor (\sim S_{\text{pre}} \land C_n \land D_n)$, where $(n \geq 1)$.

Let $T$ be a test set for $S$. Then, $T$ must satisfy the condition

$$(\forall i \in \{1, \ldots, n\} \exists t \in T \cdot \sim S_{\text{pre}}(t) \land C_i(t))$$
and
$$\exists t \in T \cdot \neg \sim S_{\text{pre}}(t)$$

where $\neg \sim S_{\text{pre}}(t)$ describes an exceptional situation.
Notation:  \( G: LE \rightarrow Ts \)
where
\( LE \) is the set of logical expressions involved.
\( Ts \) is a set of test sets

**Criterion 1:**

\[
G( (\neg S_{pre} \land C_1 \land D_1) \lor (\neg S_{pre} \land C_2 \land D_2) \lor \cdots \lor (\neg S_{pre} \land C_n \land D_n) ) = \\
G(\neg S_{pre} \land C_1 \land D_1) \cup G(\neg S_{pre} \land C_2 \land D_2) \cup \cdots \cup G(\neg S_{pre} \land C_n \land D_n).
\]
Criterion 2:

Let $\neg S_{pre} \land C_i \land D_i \ (i = 1, \ldots, n)$ be a functional scenario of specification $S$. Then,

$$G(\neg S_{pre} \land C_i \land D_i) = G(\neg S_{pre} \land C_i)$$
Criterion 3:

Let \( P_1 \lor P_2 \lor \cdots \lor P_m \) be a DNF of the test condition \( \sim S_{\text{pre}} \land C_i \). Then, we define

\[
G(\sim S_{\text{pre}} \land C_i) = \\
G(P_1 \lor P_2 \lor \cdots \lor P_m) = \\
G(P_1) \cup G(P_2) \cup \cdots \cup G(P_m)
\]
Criterion 4:
Let $Si_v = \{x_1, x_2, \ldots, x_r\}$ and $Q(x_1, x_2, \ldots, x_q)$ ($q \leq r$) be a relation involving variables $x_1, x_2, \ldots, x_q$. Then,

$$G(Q(x_1, x_2, \ldots, x_q)) = \{ T_c | (\forall x \in \{x_1, x_2, \ldots, x_q\} \cdot Q(T_c(x_1), T_c(x_2), \ldots, T_c(x_q))) \land \ldots \land \ldots \}$$

where $T_c: S_{iv} \rightarrow \text{Values}$

Example: $G(x > y) = \{ (x, 5), (y, 3), (z, 8) \}, \{(x, 8), (y, 2), (z, 300) \}$
Criterion 5:

Let $Q_1 \land Q_2 \land \cdots \land Q_w$ be a conjunction of $w$ atomic predicates in the test condition of a functional scenario of $S$. Then, we have

$$G(Q_1 \land Q_2 \land \cdots \land Q_w) = G(Q_1) \cap G(Q_2) \cap \cdots \cap G(Q_w)$$
(2) Algorithms for Test Set Generation

(2.1) For Atomic Predicates
Let \( Q(x_1, x_2, \ldots, xm) \) be an atomic predicate. It may have three formats:

(1) \( x_1 \theta E \), where \( \theta \in \{=, >, <, \geq, \leq, <>\} \), \( x_1 \) is a single variable, \( E \) a constant.

(2) \( E_1 \theta E_2 \), where \( E_1 \) and \( E_2 \) are both arithmetic expressions that involve only variable \( x_1 \).

(3) \( E_1 \theta E_2 \), where \( E_1 \) and \( E_2 \) are both arithmetic expressions that may involve variables \( x_1, x_2, \ldots, xm \).
An algorithm of test case generation for $x_1, x_2, \ldots, x_m$ in format (3) $E_1 \ominus E_2$:

**Step 1** Randomly choose values $v_2, v_3, \ldots, v_m$ from the corresponding types of variables in $E_1 \ominus E_2$;

**Step 2** Substitute $v_2, v_3, \ldots, v_m$ for variables $x_2, x_3, \ldots, x_m$ in $E_1 \ominus E_2$;

**Step 3** Convert $E_1 \ominus E_2$ to the format $x_1 \ominus E$ by applying appropriate algorithms depending on $\ominus$. 
(2.2) For conjunctions
Let $Q_1 \land Q_2 \land \cdots \land Q_w$ be a conjunction of atomic predicates. Let $x_1, x_2, \ldots, x_r$ be all input variables of the operation specification. Then, an algorithm for generating a test case from the conjunction is:

First apply the corresponding algorithm to generate a test case for $Q_1$, and then use it to evaluate $Q_2, \ldots, Q_w$. If all of them are true, a test case is generated; otherwise, repeat the same procedure until a test case is generated or a stopping generation condition is met.
(2.3) For disjunctions

A simple algorithm for test case generation from the disjunction $P_1 \lor P_2 \lor \cdots \lor P_m$:

The essential part of the algorithm is a while-loop, which produces one test case from each disjunctive clause until all the disjunctive clauses are covered, and then form a test set that contains all of the test cases produced.
(3) “Vibration” method (V-Method) for test set generation from atomic predicates

Let $E_1(x_1, x_2, ..., x_n) \mathbin{R} E_2(x_1, x_2, ..., x_n)$ denote that expressions $E_1$ and $E_2$ have relation $R$, where $x_1, x_2, ..., x_n$ are all input variables involved in these expressions.

**Question:** If the test case generated from this relation is not adequate to meet the required coverage criterion or standard, how to generate other test cases so that the goal of satisfying the required criterion can be quickly approached?
V-Method:

We first produce values for \(x_1, x_2, \ldots, x_n\) such that the relation \(E_1(x_1, x_2, \ldots, x_n) \mathbin{R} E_2(x_1, x_2, \ldots, x_n)\) holds with any "distance" between \(E_1\) and \(E_2\), and then repeatedly create more values for the variables such that the relation still holds but the "distance" between \(E_1\) and \(E_2\) "vibrates" (changes repeatedly) between the initial "distance" and the "maximum" "distance".
Example: E1 > E2
Examples of Distance definition

Distance function definition for nat0, nat, int, real types:

Distance(E₁, E₂, "R") ≡ abs(E₁ - E₂)

where R inset {>, >=, <, <=, <> , =}
Distance function definition for set types:

Distance($E_1$, $E_2$, "subset") $\equiv$
\[
\text{card}(E_2) - \text{card}(E_1)
\]

Distance($E_1$, $E_2$, "psubset") $\equiv$
\[
\text{card}(E_2) - \text{card}(E_1)
\]

Distance($E_1$, $E_2$, "inset") $\equiv$
\[
\text{card}(E_2) - \text{index}(E_1,E_2)
\]

Distance($E_1$, $E_2$, "notin") $\equiv$ \text{card}(E_2)

Distance($E_1$, $E_2$, ")$\equiv$ 0
Distance($E_1$, $E_2$, ")$\equiv$ abs(card($E_1$)-card($E_2$))
3. Test oracle for test result analysis

Definition 3.1: Let $\neg S_{pre} \land C \land D$ be a functional scenario and $T$ be a test set generated from its test condition $\neg S_{pre} \land C$. If the condition

$$\exists t \in T \cdot \neg S_{pre}(t) \land C(t) \land \neg D(t, P(t))$$

holds, it indicates that a bug in program $P$ is found by $t$ (also by $T$).
process A(x: int) y: int
pre x > 0
post (x > 10 => y = x + 1) and (x <= 10 => y = x - 1)

Functional scenarios:
(1) x > 0 \land x > 10 \land y = x + 1
(2) x > 0 \land x \leq 10 \land y = x - 1

Test result analysis

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>\neg \text{A}_{\text{pre}} \land C1 \land \neg D1 \land \neg D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>15</td>
<td>true</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>false</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>true</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>true</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>false</td>
</tr>
</tbody>
</table>

System.out.println ("the precondition is violated")
4. Conclusion and future research

4.1 Conclusion

Automatic specification-based testing can significantly benefit from formal specifications, but still face many challenges.

Test result analysis based on formal specifications can be automatically performed, but it would be extremely difficult for informal specification-based testing.
4.2 Future research

- Establish a theory to explain the relation among specification, test case generation method, coverage criteria, and bug detection effectiveness.

- Develop more efficient algorithms for test case generation from a conjunction.

- Explore automatic debugging techniques to deal with realistic software systems.
Selection of our recent related publications


Thank You!
課題

(1) Briefly describe what is the ideal goal of automatic specification-based testing.

(2) Briefly explain what is specification-based program testing and what activities are involved for such a testing.

(3) Describe the strategy for formal specification-based program testing?