Test Generation from Requirements

Lecture 6

Dr. Iichul Yoon (icyoon@sunykorea.ac.kr)
Sources: (1) Foundations of Software Testing (textbook), (2) slides by Prof. Mathur

Test Generation from Predicates

- Predicates arise from requirements in a variety of applications.

An example
- A boiler needs to be shut down when the following conditions hold:
  - The water level in the boiler is below X lbs. (a)
  - The water level in the boiler is above Y lbs. (b)
  - A water pump has failed. (c)
  - A pump monitor has failed. (d)
  - Steam meter has failed. (e)

The boiler is to be shut down when a or b is true or the boiler is in degraded mode and the steam meter fails. We combine these five conditions to form a compound condition (predicate) for boiler shutdown.

Boolean expression $E$ that when true must force a boiler shutdown:

$$E = a + b + c + d + e$$

where the $+$ sign indicates "OR" and a multiplication indicates "AND."

Test Generation from Predicates

- Goal
  - The goal of predicate-based test generation is to generate tests from a predicate $p$ that guarantee the detection of any fault, that belongs to a class of faults, in the coding of $p$.

Predicate testing

- Another example
  - A condition is represented formally as a predicate, also known as a Boolean expression. For example, consider the requirement
    
    "if the printer is ON and has paper then send document to printer."

    This statement consists of a condition part and an action part. The predicate for the condition part is:

    $$p_1: (\text{printer status = ON}) \land (\text{printer tray != empty})$$

Test Generation from Predicates

- Terms
  - Relational operators ($relop$)
    - $\{<, \leq, >, \geq, =, \neq\}$ ($= \text{ and } ==$ are considered equivalent.)
  - Boolean operators ($bop$)
    - $\{\land, \lor, \lor, \land\}$ also known as $\{\text{not, AND, OR, XOR}\}$.
  - Relational expression
    - $e_1 relop e_2$. (e.g. $a+b < c$)
    - $e_1$ and $e_2$ are expressions whose values can be compared using $relop$.
  - Simple predicate
    - A Boolean variable or a relational expression. $(x<0)$
  - Compound predicate
    - Join one or more simple predicates using $bop$.
    - e.g., $(\text{gender} == \text{"female"} \land \text{age} < 65)$
Test Generation from Predicates

*Terms*
- **Boolean expression**
  - One or more Boolean variables joined by bop. (e.g., a \( \land \) b \( \lor \) c)
  - a, b, and c are also known as *literals*.
  - Negation is also denoted by placing a bar over a Boolean expression
  - a\( \land \)b and a\( \lor \)b for a \( \land \) b and a \( \lor \) b when there is no confusion.
- **Singular Boolean expression**
  - When each literal appears only once in the expr (e.g., a \( \land \) b \( \lor \) c)
- **Mutually Singular**
  - In an expression \( E = e_1 \land e_2 \land \ldots \land e_k \land bop_1 \land bop_2 \land \ldots \land bop_m \land e_{k+1} \land \ldots \land e_n \land e_{k+2} \land k > 1 \) and \( e_i \) are mutually singular when \( e_i \) and \( e_j \) do not share any literal
  - \( e_i \) is considered singular if it is singular and mutually singular with the remaining elements of \( E \).
  - \( e_i \) is considered non-singular if it is non-singular by itself and mutually singular with the remaining elements of \( E \).

Disjunctive normal form (DNF)
- Sum of product terms: e.g., \( (pq) + (rs) + (ac) \).

Conjunctive normal form (CNF)
- Product of sums: e.g., \( [p+q][r+s][a+c] \)
- Any Boolean expression in DNF can be converted to an equivalent CNF and vice versa.
  - Example
    - CNF: \( (p+q)(r+s)(a+c) \)
    - DNF: \( (pq+rs) \)
- A predicate \( p \), can be converted to a Boolean expression by replacing each relational expression in \( P \), by a distinct Boolean variable.

Abstract Syntax Tree Representation

- **AST(\( P_r \))**
- \( (a+b<c) \land \neg p \land (r > s) \)

Fault Model for Predicate Testing

*Fault targets of predicate testing*
- Boolean operator fault
- Relational operator fault
- Arithmetic expression fault

*Boolean operator fault*
- Correct predicate: \( (a<b) \lor (c>d) \land e \)
  - Here a, b, c, and d are integer variables and e is a Boolean variable.
  - \( (a>b) \land (c>d) \land e \) Incorrect Boolean operator
  - \( (a<b) \lor (c>d) \land e \) Incorrect negation operator
  - \( (a>b) \land (c>d) \lor e \) Incorrect Boolean operators
  - \( (a<b) \lor (e>d) \land c \) Incorrect Boolean variable.
Fault Model for Predicate Testing

- Relational operator fault
  - Correct predicate: \((a < b) \lor (c > d) \land e\)
    - \((a == b) \lor (c > d) \land e\) Incorrect relational operator
    - \((a == b) \lor (c \leq d) \land e\) Two relational operator faults
    - \((a == b) \lor (c > d) \lor e\) Incorrect relational and Boolean operators

- Arithmetic expression fault
  - Assume that \(E_c: e_1 \text{ rop }_1 e_2\) is a correct expression, and \(E_i: e_3 \text{ rop}_2 e_4\) is an incorrect expr.
    - Assume that \(E_c\) and \(E_i\) use the same set of variables.
      - off-by-\(1\) fault, if any test making \(e_3 = e_2\) makes \(|e_3 - e_4| = 1\).
      - i.e., \(E_i\) is equivalent to either \((e_3 \text{ rop}_2 e_4 + 1)\) or \((e_3 \text{ rop}_2 e_4 - 1)\).
      - off-by-\(1\)^* fault, if any test making \(e_3 = e_2\) makes \(|e_3 - e_4| \geq 1\).
      - off-by-\(1\)^+ fault, if any test making \(e_3 = e_2\) makes \(|e_3 - e_4| > 1\).

Arithmetic expression faults: example

Suppose that the correct predicate \(E_c: a < b + c\), where \(a\) and \(b\) are integer variables. Assuming that \(c = 1\), three incorrect versions of \(E_i\) follow.

- \(a < b\)
  - Assuming that \(c = 1\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b\).
  - \(a < b + 1\)
    - Assume that \(c = 1\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b + 1\).
    - \(a < b - 1\)
      - Assuming that \(c > 0\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b - 1\).

\[ \text{e.g. } a = 2, b = 1, c = 1 \]

- \(a < b + 1\)
  - Assume that \(c = 1\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b + 1\).
  - \(a < b - 1\)
    - Assuming that \(c > 0\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b - 1\).

\[ \text{e.g. } a = 4, b = 2, c = 2 \]

- \(a < b + 1\)
  - Assume that \(c = 1\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b + 1\).
  - \(a < b - 1\)
    - Assuming that \(c > 0\), there is an off-by-\(1\) fault in \(E_i\) as \(a = b - 1\).

\[ \text{e.g. } a = 3, b = 2, c = 1 \]
Goal of predicate testing

- Given a correct predicate $p_c$, generate a test set $T$ such that
  - there is at least one test case $t \in T$ for which $p_c$ and its faulty version $p_i$ evaluate to different truth values
- Guarantee the detection of any fault of the kind in the fault model under consideration.

Example
- Suppose that $p_c: a < b + c$ and $p_i: a > b + c$.
- Consider a test set $T = \{t_1, t_2\}$ where $t_1: a = 0, b = 0, c = 0$ and $t_2: a = 0, b = 1, c = 1$.
- $t_2$ can reveal the fault

More for Fault Model

- Missing Boolean variable faults
- Extra Boolean variable faults

Correct predicate: $a \lor b$
- Missing Boolean variable fault: $a$
- Extra Boolean variable faults: $a \lor b \lor c$

Predicate Constraints: BR Symbols

- BR: Boolean and Relational
  - $BR = \{t, f, <, >, \leq, \geq, +, -\}$
  - A BR symbol is a constraint on a Boolean variable or a relational expression.
  - For a predicate $E: a < b$ and the constraint associated is "$<$", a test case that satisfies this constraint for $E$ must cause $E$ to evaluate to false.
  - $+$ on $E$: $e_1 \text{ rop } e_2$, A test for $E$ ensure that $0 < e_1 - e_2 \leq \varepsilon$
  - $-$ on $E$: $e_1 \text{ rop } e_2$, A test for $E$ ensures that $-\varepsilon \leq e_1 - e_2 < 0$
  - A constraint $C$ could be infeasible for a predicate $p_r$.
  - e.g., constraint is infeasible for $a > b \land b > d$, if $d > a$ is given.

Constraint and test case
- Consider an expr $E: a < c+d$, and a constraint "$C: (=)" on E.
- A test case satisfying $C$ on $E$ is: $<a=1, c=0, d=1>$

Predicate Constraints

- Let $p_r$ denote a predicate with $n \land$ and $\lor$ operators ($n>0$).
  - A predicate constraint (or simply, constraint) $C$ for $p_r$ is:
    - A sequence of $(n+1)$ BR symbols
    - One BR symbol for each Boolean variable or relational expr in $p_r$.
  - Test case $t$ satisfies $C$ for predicate $p_r$, if each component of $p_r$ satisfies the corresponding constraint in $C$ when evaluated against $t$.
  - That is, constraint $C$ for predicate $p_r$ guides the development of a test for $p_r$. 

**True and False Constraints**

- $p_r(C)$ denotes the value of predicate $p_r$ evaluated using a test case that satisfies $C$.
- $C$ is a true constraint when $p_r(C)$ is true and a false constraint otherwise.

- A set of constraints $S$ is partitioned into subsets $S^t$ and $S^f$, respectively, such that
  - $S = S^t \cup S^f$
  - For each $C$ in $S^t$, $p_r(C) = true$
  - For any $C$ in $S^f$, $p_r(C) = false$

**Predicate Testing Criteria**

- Given a predicate $p_r$, we want to generate a test set $T$ such that
  - $T$ is minimal and
  - $T$ guarantees the detection of any fault in the coding of $p_r$

- Common criteria
  - BOR
  - BRO
  - BRE

**Predicate Testing: BOR Testing Criterion**

- A test set $T$ that satisfies the BOR testing criterion for a compound predicate $p_r$ guarantees the detection of single or multiple Boolean operator faults in the implementation of $p_r$.

- $T$ is referred to as a BOR-adequate test set and sometimes written as $T_{BOR}$.

**Predicate Testing: BRO Testing Criterion**

- A test set $T$ that satisfies the BRO testing criterion for a compound predicate $p_r$ guarantees the detection of single or multiple Boolean operator and relational operator faults in the implementation of $p_r$.

- $T$ is referred to as a BRO-adequate test set and sometimes written as $T_{BRO}$.
Predicate Testing: BRE Testing Criterion

- A test set \( T \) that satisfies the BRE testing criterion for a compound predicate \( p_r \), guarantees the detection of single or multiple Boolean operator, relational expression, and arithmetic expression faults in the implementation of \( p_r \).

- \( T \) is referred to as a BRE-adequate test set and sometimes written as \( T_{\text{BRE}} \).

Guaranteeing Fault Detection: Meaning?

- Let \( T_x \), \( x \in \{ \text{BOR}, \text{BRO}, \text{BRE} \} \), be a test set derived from predicate \( p_r \).

- Let \( p_f \) be another predicate obtained from \( p_r \) by injecting single or multiple faults of one of three kinds:
  - Boolean operator fault, relational operator fault, and arithmetic expression fault.

- \( T_x \) is said to guarantee the detection of faults in \( p_f \) if for some \( t \in T_x \), \( p_r(t) \neq p_f(t) \).

Guaranteeing Fault Detection: Example

- **BOR-adequate test set**
  
  Let \( p_r = a < b \land c > d \)

  Constraint set \( S = \{ (t, t), (t, f), (f, t) \} \)

  Compute \( T_{\text{BOR}} = \{ t_1, t_2, t_3 \} \) is a BOR adequate test set that satisfies \( S \).
  
  - \( t_1: \langle a=1, b=2, c=1, d=0 \rangle \); Satisfies \((t, t)\), i.e. \( a < b \) is true and \( c > d \) is also true.
  
  - \( t_2: \langle a=1, b=2, c=1, d=2 \rangle \); Satisfies \((t, f)\)
  
  - \( t_3: \langle a=1, b=0, c=1, d=0 \rangle \); Satisfies \((f, t)\)

How to evaluate the effectiveness?

- Generate all variants by injecting Boolean operator faults, and evaluate against \( T \)

### Predicate

<table>
<thead>
<tr>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a &lt; b \land c &gt; d )</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

**Single Boolean operator fault**

| 1 | \( a < b \land c > d \) | true | true | true |
| 2 | \( a < b \land c < d \) | false | true | false |
| 3 | \( a < b \land c < d \) | false | false | true |

**Multiple Boolean operator faults**

| \( \neg a < b \land c < d \) | true | true | false |
| \( \neg a < b \land c < d \) | false | false | true |
| \( \neg a < b \land c < d \) | false | false | false |
Generating BOR, BRO, BRE Adequate Tests

- We want to generate minimal test set.

- Review of Cartesian product of two sets A and B
  \[ A \times B = \{ (a, b) \mid a \in A \text{ and } b \in B \} \]

- To generate minimal set, define another set product called onto product
  \[ A \bigodot B = \{ (u, v) \mid u \in A, v \in B \} \]
  where each element of A appears at least once and each element of B appears once as v.
  - Note that \( A \bigodot B \) is a minimal set.

Cartesian vs. Onto Product

- Let \( A = \{t, =, >\} \) and \( B = \{f, <\} \)

- \( A \times B = \{ (t, f), (t, <), (=, f), (=, <), (>, f), (>, <) \} \)

- \( A \bigodot B = \{ (t, f), (=, <), (>, <) \} \)

- Any other possibilities for \( A \bigodot B ? \)

Generation of BOR Constraint Set

- Use AST(\( P_r \))
  - Leaf nodes: Boolean variable or relation expression
  - Internal nodes: Boolean operators

- Procedure BOR_CSET
  - Label each leaf node with the initial constraint set
  - Compute BOR-constraint set for internal nodes
    - Visit in a bottom-up order
    - Rules are different depending on internal node type
  - The constraint set for the root is the BOR-adequate test set
### Generation of BOR Constraint Set

- **Compute BOR-constraint set for internal nodes**

  \( N \) is an OR-node:
  \[
  S_1' = S_{N_1}' \otimes S_{N_2}' \\
  S_2' = (S_{N_1}' \times \{ t \}) \cup (\xi \times S_{N_2}') \\
  \text{where } \xi \in S_{N_1}' \text{ and } \xi \in S_{N_2}'
  \]

  \( N \) is an AND-node:
  \[
  S_1' = S_{N_1}' \otimes S_{N_2}' \\
  S_2' = (S_{N_1}' \times \{ t \}) \cup (\xi \times S_{N_2}') \\
  \text{where } \xi \in S_{N_1}' \text{ and } \xi \in S_{N_2}'
  \]

  \( N \) is NOT-node:
  \[
  S_1'' = S_{N_1}' \\
  S_2'' = S_{N_2}'
  \]

### Generation of BOR Adequate Test Set

- **A test set \( T \) that satisfies the BOR testing criterion for a compound predicate \( p_r \), guarantees the detection of single or multiple Boolean operator faults in the implementation of \( p_r \).**

- The BOR constraint set \( S \) for relational expression \( e_1 \) relope \( e_2 \): 
  \[
  S = \{ (>, (=), (<) \}
  \]

- Separation of \( S \) into its true \( (S^t) \) and false \( (S^f) \) components:
  - relop: \( > \)
    \[
    S^t = \{ (>), (=), (<) \} \\
    S^f = \{ (>, (=), (<) \}
    \]
  - relop: \( \geq \)
    \[
    S^t = \{ (>, (=), (<) \} \\
    S^f = \{ (>, (=), (<) \}
    \]
  - relop: \( = \)
    \[
    S^t = \{ (=) \} \\
    S^f = \{ (=), (>), (<) \}
    \]
  - relop: \( < \)
    \[
    S^t = \{ (<) \} \\
    S^f = \{ (=), (>), (<) \}
    \]
  - relop: \( \leq \)
    \[
    S^t = \{ (<) \} \\
    S^f = \{ (>), (=), (<) \}
    \]
Generation of BRO Constraint Set

- Procedure BRO_CSET
  - Label each leaf node with the initial constraint set
    - For Boolean variable node, \{t, f\}
    - For relational expression node, \{<, =, >\}
  - Compute BOR-constraint set for internal nodes
  - Visit in a bottom-up order
  - Use BOR_CSET procedure
  - The constraint set for the root is the BRO-adequate test set

BRO Constraint Set: Example (cont.)

- Traverse AST in a bottom up order and compute constraint set for internal nodes

\[ S'_{N0} = S_{N2} \cap \{t\} \]
\[ S'_{N3} = S_{N4} \cap \{t\} \]
\[ S'_{N4} = (S_{N0} \times \{<\}) \cup (\{<\} \times S_{N3}) \]
\[ = \{(>, f, >, f, <, f, <\}\} \]
\[ S'_{N5} = \{(>, f, >, f, <, f, <\}\} \]
\[ S'_{N6} = (S'_{N4} \times \{<\}) \cup (\{<\} \times S'_{N5}) \]
\[ = \{(>, f, >, f, <, f, <\}\} \]

Generation of BRO Constraint Set: Example

- Compute BRO-adequate test set for \(p_{2'}: (a+b<c) \land \lnot p \lor (r>s)\)
  - Label each leaf node with its constraint set \(S\)

\[ S_{N0} = \{<\} \]
\[ S_{N1} = \{(>, f, >, f, <, f, <\}\} \]
\[ S_{N2} = \{(>, f, >, f, <, f, <\}\} \]
\[ N_{0} \]
\[ \lnot p \]
\[ S_{N3} = \{<\} \]
\[ p \]
\[ N_{2} \]
\[ \{(>, f, >, f, <, f, <\}\} \]

BRO Constraint Set: Example (cont.)

- Next, compute the constraint set for the root node.
  \[ N_{6} \]
  \[ N_{5} \]
  \[ N_{4} \]
  \[ N_{3} \]

\[ N_{0} \]

\[ \{(>, f, >, f, <, f, <\}\} \]
\[ \{(>, f, >, f, <, f, <\}\} \]
\[ \{(>, f, >, f, <, f, <\}\} \]
\[ \{(>, f, >, f, <, f, <\}\} \]
Generation of BRE Constraint Set

- A test set $T$ that satisfies the BRE testing criterion for a compound predicate $p$, guarantees the detection of single or multiple Boolean operator, relational expression, and arithmetic expression faults in the implementation of $p$.

- BRE-constraint set for a relational expression is
  $\{(+\varepsilon), (=), (-\varepsilon)\}$, $\varepsilon > 0$
  - For expression $e_1$ rop $e_2$

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Satisfying condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>$0 &lt; e_1 - e_2 \leq +\varepsilon$</td>
</tr>
<tr>
<td>$=$</td>
<td>$e_1 = e_2$</td>
</tr>
<tr>
<td>$-$</td>
<td>$-\varepsilon \leq e_1 - e_2 &lt; 0$</td>
</tr>
</tbody>
</table>

Generation of BRE Constraint Set

- Procedure BRE_CSET
  - Label each leaf node with the initial constraint set
    - For Boolean variable node, $\{t,f\}$
    - For relational expression node, $\{+\varepsilon, =, -\varepsilon\}$
  - Compute BOR-constraint set for internal nodes
    - Visit it in a bottom-up order
    - Use BOR_CSET procedure
  - The constraint set for the root is the BRE-adequate test set