Predicate-based Testing

- Predicates are conditions
  - Divides the input domain into partitions
  - Define the paths of the program

- Program P
  - Input X; Predicate C
  - If outcome of C is incorrect,
    - Either C is incorrect,
    - Or statement(s) executed before C
  - Most likely, P’s output is incorrect
    - Low probability of “coincidental correctness”

- Predicate-based testing
  - Require certain types of tests for each predicate in the program

Importance of Predicate-based Testing

- Thorough testing of C used to
  - Detect faults in C,
  - Statements executed before C
  - Statements executed after C

Terms Defined

- Predicate
  - Simple or compound predicate

- Simple predicate
  - Boolean variable, or
  - Relational expression,
  - May have one or more NOT (¬) operators

- Relational expression
  - E1 <rop> E2
    - E1 and E2 are arithmetic expressions
    - rop ∈ {<, <=, >, >=, /=, =}

Terms Defined (2)

- Compound predicate
  - At least one “binary Boolean operator”
  - Two or more operands
  - Maybe NOT operators
  - Maybe parenthesis

- Binary Boolean operators
  - OR (|) and AND (∧)

- Simple operand
  - Operand without binary Boolean operators

- Compound operand
  - Operand with at least one binary Boolean operator
Terms Defined (3)

- Boolean expression
  - Predicate with no relational expressions
- Bi = Boolean expression
- Ei = Arithmetic expression
- <rop> or <ropᵢ> = relational operator
- <bop> or <bopᵢ> = binary Boolean operator

Assumptions

- Predicate has no syntactic faults

Types of Faults

- An “incorrect” predicate may have one or more of the following faults
  - Boolean operator fault
    - Incorrect AND/OR or missing/extra NOT
  - Boolean variable fault
    - Incorrect Boolean variable
  - Parenthesis fault
    - Incorrect location
  - Relational operator fault
    - Incorrect relational operator
  - Arithmetic expression fault
    - Various types

Yet More Terms

- Existence of one/more faults is “detected by a test” T if an execution of C with T produces an incorrect outcome of C
- Test set T for C “guarantees the detection” of certain type of faults F in C if the existence of F in C can be detected by at least one element in T, provided C doesn’t contain faults of other types
Yet More Terms (2)

- Assume that \( C^* \) has the same set of variables as \( C \) and is not equivalent to \( C \). Test set \( T \) “distinguishes” \( C \) from \( C^* \) if \( C \) and \( C^* \) produce different outcomes for \( T \).
- Assume that \( C \) contains faults and \( C'' \) is the correct version of \( C \). Test set \( T \) is “insensitive” to the faults in \( C \) if this test cannot distinguish \( C \) from \( C'' \).

Testing Simple Predicates

- Branch testing
  - TRUE and FALSE branches be executed at least once
- Relational Operator Testing
  - Given \( E_1 \ <\text{op}\ > E_2 \)
  - Need 3 tests
  - \( E_1 > E_2; E_1 < E_2; E_1 = E_2 \)
  - If only \( \text{op} \) is incorrect and \( E_1 \) and \( E_2 \) are correct, then detection is guaranteed

Testing Compound Predicates

- Complete branch testing
  - All TRUE and FALSE branches of each simple/compound operand in compound predicate \( C \) be executed at least once
- Exhaustive branch testing
  - All combinations of TRUE and FALSE branches of simple operands in \( C \) be executed at least once
  - \( C \) has \( N \) Boolean Operators, then \( N+1 \) simple operands. Requires \( 2^{(n+1)} \) test cases

Testing Compound Predicates (2)

- Complete relational operator testing
  - Relational operator testing for each relational expression in \( C \)
  - Let \( C# \) be \( (E_1 = E_2) \& (E_3 /= E_4) \)
  - Assume \( T1 \) contains 3 tests
    - \( T11 \) makes \( E_1 = E_2 \) and \( E_3 = E_4 \)
    - \( T12 \) makes \( E_1 > E_2 \) and \( E_3 > E_4 \)
    - \( T13 \) makes \( E_1 < E_2 \) and \( E_3 < E_4 \)
  - \( T1 \) satisfies relational operator testing for each simple operand of \( C# \)
- If \( E_1, E_2, E_3, \) and \( E_4 \) are correct, what can we say about the correctness of operators?
Complete Relational Operator Testing

- Can the test cases T11, T12, and T13 distinguish between C# and
  - (E1 = E2) & (E3 < E4)
  - (E1 /= E2) & (E3 = E4)

BR-constraints

- Given a predicate
  - \(<\text{opd}_1>\ <\text{bop}_1>\ <\text{opd}_2>\ <\text{bop}_2>\ ...\ <\text{opd}_n>\ <\text{bop}_n>\)
  - \(<\text{opd}_i>\) is the ith simple operand

- BR-constraint
  - \((\text{D}_1, \text{D}_2, ..., \text{D}_n)\)
    - Each \(\text{D}_i\) is a symbol specifying a constraint on the Boolean variable or relational expression in \(<\text{opd}_i>\)

BR-constraints (2)

- Constraints for a Boolean variable B
  - The value of B is TRUE
  - The value of B is FALSE
  - No constraint

- Symbols
  - \(\dagger\)
  - \(\times\)
  - \(\ast\)

BR-constraints (2)

- Constraints for a relational expression \((E_1 <\text{rop}> E_2)\)
  - Value is TRUE \(\dagger\)
  - Value is FALSE \(\times\)
  - \((E_1 - E_2) > 0\) \(\dagger\)
  - \((E_1 - E_2) = 0\) \(\dagger\)
  - \((E_1 - E_2) < 0\) \(\times\)
  - No constraint \(\ast\)
Constraint Satisfaction

- **Definition**
  - Constraint D on predicate C is covered (or satisfied) by a test if during the execution of C with this test, the value of each Boolean variable or relational expression in C satisfies the corresponding constraint in D.

- **Examples**
  - $\{=, <\}$
  - For $((E1 \geq E2) \lor \neg (E3 > E4))$

- **Coverage** requires that $(E1 = E2)$ and $(E3 < E4)$

Terms Redefined

- **In terms of BR-constraints**
  - Branch testing $(E1 < \text{rop} > E2)$
    - $\{(t), (f)\}$
  - Relational operator testing $(E1 < \text{rop} > E2)$
    - $\{(>, =), (<, \text{rop}2)\}$
  - Complete branch testing $((E1 < \text{rop1} > E2) < \text{bop} > (E3 < \text{rop2} > E4))$
    - $\{(t, *), (f, *), (*, t), (*, f)\}$
  - Complete relational operator testing $((E1 < \text{rop1} > E2) < \text{bop} > (E3 < \text{rop2} > E4))$
    - $\{(>, *), (=, *), (<, *), (*, >), (*, =), (*, <)\}$

Terms Defined

- **Concatenation**
  - Let $u = (u_1, u_2, ..., u_m)$ and $v = (v_1, v_2, ..., v_n)$ be two sequences.
  - $(u,v) = (u_1, u_2, ..., u_m, v_1, v_2, ..., v_n)$

- **Other terms**
  - Let $A$ and $B$ be two sets.
  - $A \cup B$ denotes the union of $A$ and $B$.
  - $A \times B$ is the product of $A$ and $B$.
  - $|A|$ is the size of $A$.
  - $A \rightarrow B$ is called the onto from $A$ to $B$.
    - Minimal set of $(u,v)$ such that $u \subseteq A$ and every element in $A$ appears in $u$ at least once, $v \subseteq B$ and every element in $B$ appears in $v$ at least once.
Terms Defined

- **Observations**
  - $|A\%B| = \max(|A|, |B|)$
  - $A\%B$ may have several possible values
    - If $C = \{(a), (b)\}$ and $D = \{(c), (d)\}$
    - Then what is $C\%D$
      - $(a,c),(b,d))$
      - $(a,d),(b,c))$
    - How about if $E = \{(a), (b)\}$ and $F = \{(c), (d), (e)\}$

Expected Outcome

- Let $X$ be a constraint that contains “$t$”, “$f$”, “$>$”, “$<$”, and “$=$” for a predicate $C$
- Value produced by $C$ on any input covering $X$: $C(X)$
- $X$ covers the TRUE branch of $C$ if $C(X) = \text{TRUE}$, and
- $X$ covers the FALSE branch of $C$ if $C(X) = \text{FALSE}$
- Let $S$ be a set of constraints for $C$
- Partition $S$ into $S_t$ and $S_f$
  - $S_t(C) = \{X \in S \mid C(X) = t\}$
  - $S_f(C) = \{X \in S \mid C(X) = f\}$

Let’s Try Them Out

- $E1 < E2$
  - $S_1 = \{(<), (>), (=)\}$
  - $S_1_t = \{(<)\}$
  - $S_1_f = \{ (>), (=)\}$
- $E3 \geq E4$
  - $S_2 = \{ (>), (=), (<)\}$
  - $S_2_t = \{ (>), (=)\}$
  - $S_2_f = \{ (<)\}$
- $E5 = E6$
  - $S_3 = \{ (=), (<), (>)\}$
  - $S_3_t = \{ (=)\}$
  - $S_3_f = \{ (<), (>)\}$

|&

- More complex predicates
  - $(E3 \geq E4) \mid (E5 = E6)$
    - $S_4_f = \{ (<), (<), (=)\}$
    - $(E3 \geq E4) \land (E5 = E6)$
      - $S_9_t = \{ (>), (=), (=)\}$
  - How about $S_4_t$ and $S_9_f$?
Surprise Quiz

- How About S9_f?

What Next?

- Once all the constraints have been obtained, test cases may be generated.