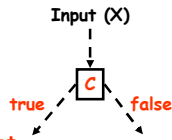


## Predicate-based Testing

- **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 <rop<sub>i</sub>> = relational operator
- <bop> or <bop<sub>i</sub>> = 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  $E1 <rop> E2$
  - Need 3 tests
  - $E1 > E2$ ;  $E1 < E2$ ;  $E1 = E2$
  - If only  $<rop>$  is incorrect and  $E1$  and  $E2$  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  $(E1 = E2) \& (E3 \neq E4)$
  - Assume  $T1$  contains 3 tests
    - $T11$  makes  $E1 = E2$  and  $E3 = E4$
    - $T12$  makes  $E1 > E2$  and  $E3 > E4$
    - $T13$  makes  $E1 < E2$  and  $E3 < E4$
  - $T1$  satisfies relational operator testing for each simple operand of  $C\#$
- If  $E1$ ,  $E2$ ,  $E3$ , and  $E4$  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
  - $(E1 = E2) \& (E3 < E4)$
  - $(E1 \neq E2) \& (E3 = E4)$

## BR-constraints

- Given a predicate
  - $\langle \text{opd}_1 \rangle \langle \text{bop}_1 \rangle \langle \text{opd}_2 \rangle \langle \text{bop}_2 \rangle \dots \langle \text{opd}_n \rangle \langle \text{bop}_n \rangle$
  - $\langle \text{opd}_i \rangle$  is the  $i$ th simple operand
- BR-constraint
  - $(D1, D2, \dots, Dn)$
  - Each  $D_i$  is a symbol specifying a constraint on the Boolean variable or relational expression in  $\langle \text{opd}_i \rangle$

## BR-constraints (2)

- Constraints for a Boolean variable B
  - The value of B is TRUE
  - The value of B is FALSE
  - No constraint
- Symbols
  - t
  - f
  - \*

## BR-constraints (2)

- Constraints for a relational expression  $(E1 \langle \text{rop} \rangle E2)$ 

• Value is TRUE	t
• Value is FALSE	f
• $(E1 - E2) > 0$	>
• $(E1 - E2) = 0$	=
• $(E1 - E2) < 0$	<
• No constraint	*

## 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  $D$
- **E.g.,**
  - $(=, <)$
  - for  $((E1 >= E2) \mid \neg(E3 > E4))$
- **Coverage requires that  $(E1 = E2)$  and  $(E3 < E4)$**

## Constraint Satisfaction (2)

- **Definition**
  - Set  $S$  of BR-constraints on predicate  $C$  is covered (or satisfied) by a test set  $T$  if each constraint in  $S$  is covered for  $C$  by at least one test in  $T$

## Terms Redefined

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

## Terms Defined

- **Concatenation**
  - Let  $u = (u_1, u_2, \dots, u_m)$  and  $v = (v_1, v_2, \dots, v_n)$  be two sequences
  - $(u, v) = (u_1, u_2, \dots, u_m, v_1, v_2, \dots, v_n)$
- **Other terms**
  - Let  $A$  and  $B$  be two sets
  - $A \cup B$  denotes the **union** of  $A$  and  $B$
  - $A * B$  is the **product** of  $A$  and  $B$
  - $|A|$  is the **size** of  $A$
  - $A \% B$  is called the **onto** from  $A$  to  $B$ 
    - Minimal set of  $(u, v)$  such that  $u \in A$  and every element in  $A$  appears in  $u$  at least once;  $v \in B$  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

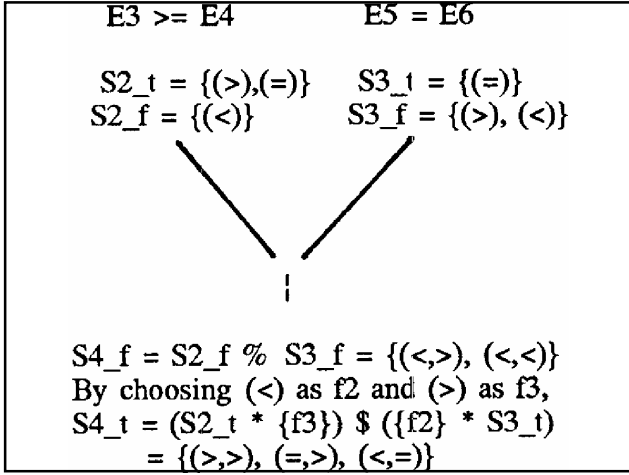
- Let  $X$  be a constraint that contains "t", "f", ">", "<", and "=" for a predicate  $C$
- Value produced by  $C$  on any input covering  $X$ :  $C(X)$
- $X$  covers the TRUE branch of  $C$  if  $C(X)=TRUE$ , and
- $X$  covers the FALSE branch of  $C$  if  $C(X)=FALSE$
- Let  $S$  be a set of constraints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

## Lets Try Them Out

- $E1 < E2$ 
  - $S1 = \{(<), (>), (=)\}$
  - $S1\_t = \{(<)\}$
  - $S1\_f = \{(>), (=)\}$
- $E3 >= E4$ 
  - $S2 = \{(>), (=), (<)\}$
  - $S2\_t = \{(>), (=)\}$
  - $S2\_f = \{(<)\}$
- $E5 = E6$ 
  - $S3 = \{(=), (<), (>)\}$
  - $S3\_t = \{(=)\}$
  - $S3\_f = \{(<), (>)\}$

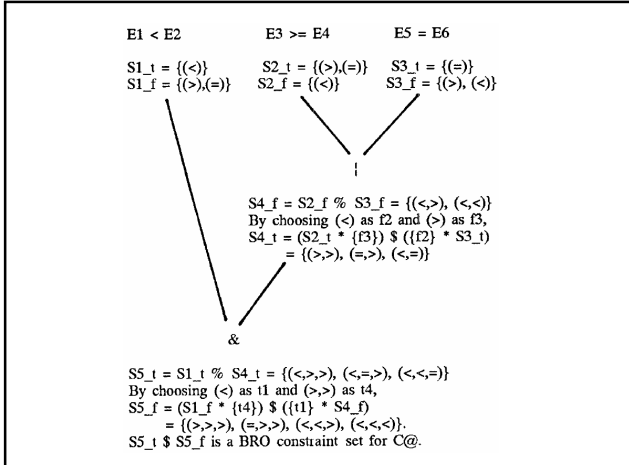
## |&

- **More complex predicates**
  - $(E3 >= E4) \mid (E5 = E6)$ 
    - $S4\_f = \{(<, <), (<, >)\}$
  - $(E3 >= E4) \& (E5 = E6)$ 
    - $S9\_t = \{(>, =), (=, =)\}$
- How about  $S4\_t$  and  $S9\_f$ ?



### Surprise Quiz

- How About S9\_f?



### What Next?

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