STAR Laboratory of Advanced Research on Software Technology

Requirements-based Test Generation for Predicate Testing

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
http://www.utdallas.edu/~ewong

Speaker Biographical Sketch

- Professor & Director of International Outreach
  Department of Computer Science
  University of Texas at Dallas
- Guest Researcher
  Computer Security Division
  National Institute of Standards and Technology (NIST)
- Vice President, IEEE Reliability Society
- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)
- Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project
  - *Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses*
- Founder & Steering Committee co-Chair for the SERE conference (*IEEE International Conference on Software Security and Reliability*) (http://paris.utdallas.edu/sere13)

Learning Objectives

- Equivalence Class partitioning
- Boundary value analysis

} Essential *black-box* techniques for generating tests for *functional testing*

- Test generation from predicates

Three Techniques

- BOR
- BRO
- BRE

Test Generation from Predicates (1)

- We will now examine three techniques
  - BOR (Boolean Operator)
  - BRO (Boolean and Relational Operator), and
  - BRE (Boolean Relational Expression)

  for generating test cases that are guaranteed to *detect certain faults in the coding of conditions*

- The conditions from which test cases are generated might arise from requirements or might be embedded in the program to be tested.

- *Conditions guard actions*
  - For example,

    if condition then action

    is a typical format of many functional requirements

Test Generation from Predicates (2)

- 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 following predicate represents the condition part of the statement

  p_r: (*printerstatus* = *ON*) $\wedge$ (*printertray* != *empty*)

Predicates

- Relational operators (relop): {<, ≤, >, ≥, =, ≠}
  = and = = are equivalent
- Boolean operators (bop): {!, ∧, ∨, xor} also known as
  {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 (e.g., $x < 0$)
- Compound predicate: Join one or more simple predicates using bop
  (e.g., *gender* = = "female" ∧ *age* > 65)

Boolean Expressions (1)

- Boolean expression: one or more *Boolean variables* joined by bop.
  - Example: ($a \wedge b \vee !c$) where *a*, *b*, and *c* are also known as literals.
- Negation is also denoted by placing a bar over a Boolean expression. such as in ($a \wedge \overline{b}$)
- We also write ab for $a \wedge b$ and $a + b$ for $a \vee b$ when there is no confusion.
- Singular Boolean expression: When each literal appears **only once**.
  - Example: ($a \wedge b \vee !c$)

Boolean Expressions (2)

- Disjunctive normal form (DNF): Sum of product terms:

  e.g., $(pq) + (rs) + (ac)$.

- Conjunctive normal form (CNF): Product of sums:

  e.g., $(p + q)(r + s)(a + c)$.

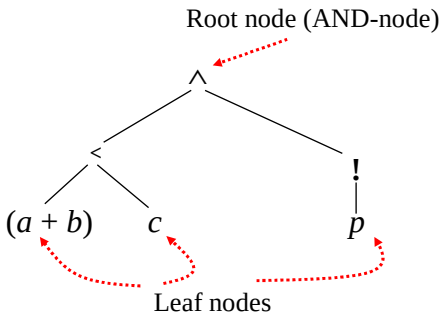
- Any Boolean expression in DNF can be converted to an equivalent CNF and vice versa.
  - e.g., CNF: $(p + !r)(p + s)(q + !r)(q + s)$ is equivalent to DNF: $(pq + !rs)$

Boolean Expressions (3)

- Mutually singular: Boolean expressions e_1 and e_2 are mutually singular when they *do not share any literal*

Boolean Expressions: Syntax Tree Representation

- Abstract syntax tree (AST) for $(a + b) < c \wedge !p$
- Internal nodes are labeled by boolean and relational operators



Fault Model for Predicate Testing

- *What kind of faults are we targeting when testing for the correct implementation of predicates?*
- Suppose that the specification of a software module requires that an action be performed when the condition $(a < b) \vee (c > d) \wedge e$ is true.
- Here a, b, c, and d are integer variables and e is a Boolean variable.

Boolean Operator Faults

- Correct predicate: $(a < b) \vee (c > d) \wedge e$

$(a < b) \wedge (c > d) \wedge e$	Incorrect Boolean operator
$(a < b) \vee !(c > d) \wedge e$	Incorrect negation operator
$(a < b) \wedge (c > d) \vee e$	Incorrect Boolean operators
$(a < b) \vee (c > d) \wedge x$	Incorrect Boolean variable

Relational Operator Faults

- Correct predicate: $(a < b) \vee (c > d) \wedge e$

$(a == b) \vee (c > d) \wedge e$	Incorrect relational operator
$(a == b) \vee (c \leq d) \wedge e$	Two relational operator faults
$(a == b) \vee (c > d) \vee e$	Incorrect relational and Boolean operators

Missing or Extra Boolean Variable Faults

- Correct predicate: $a \vee b$
- Missing Boolean variable fault: a
- Extra Boolean variable fault: $a \vee b \wedge c$

Goal of Predicate Testing (1)

- Given a correct predicate p_c, the goal of predicate testing is to 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 (i.e., p_c = true and p_i = false or vice versa)

Goal of Predicate Testing (2)

- As an 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: <$a = 0$, $b = 1$, $c = 1$>

- The fault in p_i is not revealed by t_1 as both p_c and p_i evaluate to false when evaluated against t_1

- However, the fault is revealed by t_2 as p_c evaluates to true and p_i to false when evaluated against t_2

Predicate Constraints: BR symbols

- Consider the following Boolean-Relational set of BR-symbols:
  - BR={***t***, ***f***, <, =, >}

- A BR symbol is *a constraint* on a Boolean variable or a relational expression

- For example, consider the predicate E: $a < b$ and the constraint " > ".
  - A test case that satisfies this constraint for E must cause E to evaluate to false.

Infeasible Constraints

- A constraint C is considered *infeasible* for predicate p_r if there exists no input values for the variables in p_r that satisfy C.
- For example, the constraint t (*true*) is infeasible for the predicate $(a > b) \wedge (b > d)$ if it is known that $d > a$

Predicate Constraints

- Let p_r denote a predicate with n, $n > 0$, $\vee$ and $\wedge$ operators
- A *predicate constraint* C for predicate p_r is a sequence of $(n + 1)$ BR symbols, one for each Boolean variable or relational expression in p_r. When clear from context, we refer to "predicate constraint" as simply constraint.
- 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.*
  - Constraint C for predicate p_r guides the development of a test case for p_r (i.e., it offers *hints on what the values of the variables should be for p_r to satisfy C*)

Predicate Constraints: Example

- Consider the predicate p_r: $b \wedge (r < s) \vee (u \geq v)$ and a constraint C: (t, =, >)
- The following test case satisfies C for p_r

  <b = *true*, r = 1, s = 1, u = 1, v = 0>
- The following test case does not satisfy C for p_r

  <b = *true*, r = 1, s = 2, u = 1, v = 2>

True and False Constraints

- $p_r(C)$ denotes the value of predicate p_r evaluated using a test case that satisfies C
- C is referred to as 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 for each C in S^t, $p_r(C)$ = *true*, and for any C in S^f, $p_r(C)$ = *false*.
- $S = S^t \cup S^f$

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 the faults (correspond to *some fault model*) in the implementation of p_r
- We will discuss three such criteria named
  - BOR (Boolean Operator),
  - BRO (Boolean and Relational Operator), and
  - BRE (Boolean Relational Expression)

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 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 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_{BRE}

Predicate Testing: Guaranteeing Fault Detection

- Let T_x, $x \in$ {BOR, BRO, 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

- Let $p_r = a < b \wedge c > d$
- Constraint set S = {(t, t), (t, f), (f, t)} ← Given to you at this moment
- Let T_{BOR} = {t_1, t_2, t_3} is a BOR adequate test set that satisfies S

  t_1: < a = 1, b = 2, c = 1, d = 0 > satisfies (t, t)
  (i.e., $a < b$ is true and $c < d$ is also true)
  t_2: < a = 1, b = 2, c = 1, d = 2 > satisfies (t, f)
  t_3: < a = 1, b = 0, c = 1, d = 0 > satisfies (f, t)

Guaranteeing Fault Detection: In Class Exercise (1)

- Inject single or multiple Boolean operator faults in

$$p_r: a < b \wedge c > d$$

and *show that T guarantees the detection of each fault.*

Guaranteeing Fault Detection: In Class Exercise (2)

- The following table lists p_r and a total of 7 faulty predicates obtained by inserting single and multiple Boolean operator faults in p_r
- Each predicate is evaluated against the three test cases in T
  - Each faulty predicate evaluates to a value different from that of p_r for at least one test case in T

	Predicate	t_1	t_2	t_3
	$a < b \wedge c > d$	true	false	false
Single Boolean operator fault				
1	$a < b \vee c > d$	true	true	true
2	$a < b \wedge \neg c > d$	false	true	false
3	$\neg a < b \wedge c > d$	false	false	true
Multiple Boolean operator faults				
4	$a < b \vee \neg c > d$	true	true	false
5	$\neg a < b \vee c > d$	true	false	true
6	$\neg a < b \wedge \neg c > d$	false	false	false
7	$\neg a < b \vee \neg c > d$	true	true	true

Guaranteeing Fault Detection: In Class Exercise (3)

- Can we delete any of these three test cases in T and still guarantee the detection of all the Boolean operator faults?

Guaranteeing Fault Detection: In Class Exercise (4)

- Suppose we remove t_2, then *the faulty predicate* 4 in the previous table cannot be distinguished from p_r by tests t_1 and t_3 BRO
- In fact, if we remove t_2 from T, then T is no longer BOR adequate because the constraint (t, f) is not satisfied
- We can verify that if any column in the previous table is removed, at least one of the faulty predicates will be left indistinguishable by the tests in the remaining two columns

Cross & Onto Product

- The *cross product* of two sets A and B is defined as

  $A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$

- The *onto product* of two sets A and B is defined as

  for finite sets A and B, $A \otimes B$ is a minimal set of pairs (u, v) such that $\{(u, v) \mid u \in A, v \in B$, and each element of A appears at least once as u and each element of B appears once as $v\}$

Set Products: Example (1)

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

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

  $A \otimes B = \{(t, f), (=, <), (>, <)\}$

- Any other possibilities for $A \otimes B$?

Set Products: Example (2)

- Let $A = \{t, =, >\}$ and $B = \{f, <\}$
- Any other possibilities for $A \otimes B$?

$A \otimes B = \{(t, <), (=, f), (>, <)\}$ ← second possibility
$A \otimes B = \{(t, f), (=, <), (>, f)\}$ ← third possibility
$A \otimes B = \{(t, <), (=, <), (>, f)\}$ ← fourth possibility

Algorithm for Generation of BOR Constraint Sets (1)

- We will use the following notation:

p_r is a predicate
AST (p_r) is its abstract syntax tree
N_1, N_2, …. refer to various nodes in the AST (p_r)
S_N is the constraint set for node N in the syntax tree for p_r
S_N^t is the true constraint set for node N in the syntax tree for p_r
S_N^f is the false constraint set for node N in the syntax tree for p_r
$S_N = S_N^t \cup S_N^f$

Algorithm for Generation of BOR Constraint Sets (2)

Procedure for generating a minimal BOR-constraint set from an abstract syntax tree of a predicate p_r

Input: An abstract syntax tree for predicate p_r, denoted by $AST(p_r)$. p_r contains only singular expressions.

Output: BOR-constraint set for p_r attached to the root node of $AST(p_r)$.

Procedure: BOR-CSET

Step 1 Label each leaf node N of $AST(p_r)$ with its constraint set $S(N)$. For each leaf $S_N = \{t, f\}$

Step 2 Visit each nonleaf node in $AST(p_r)$ in a bottom up manner. Let N_1 and N_2 denote the direct descendants of node N, if N is an AND or an OR-node. If N is a NOT-node, then N_1 is its direct descendant. S_{N_1} and S_{N_2} are the BOR-constraint sets for nodes N_1 and N_2, respectively. For each nonleaf node N, compute S_N as follows:

Algorithm for Generation of BOR Constraint Sets (3)

2.1 *N is an OR-node:*

$$S_N^f = S_{N_1}^f \otimes S_{N_2}^f$$
$$S_N^t = (S_{N_1}^t \times \{f_2\}) \cup (\{f_1\} \times S_{N_2}^t)$$
where $f_1 \in S_{N_1}^f$ and $f_2 \in S_{N_2}^f$

2.2 *N is an AND-node:*

$$S_N^t = S_{N_1}^t \otimes S_{N_2}^t$$
$$S_N^f = (S_{N_1}^f \times \{t_2\}) \cup (\{t_1\} \times S_{N_2}^f)$$
where $t_1 \in S_{N_1}^t$ and $t_2 \in S_{N_2}^t$

2.3 *N is NOT-node:*

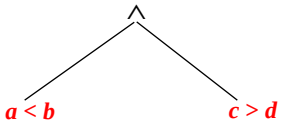
$$S_N^t = S_{N_1}^f$$
$$S_N^f = S_{N_1}^t$$

Step 3 The constraint set for the root of $AST(p_r)$ is the desired BOR-constraint set for p_r.

End of Procedure BOR-CSET

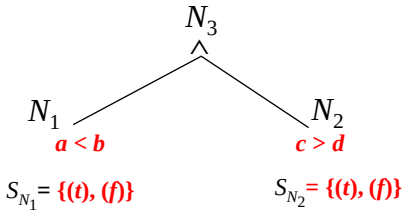
Generation of BOR Constraint Set (1)

- We want to generate T_{BOR} for p_r: $a < b \wedge c > d$
- First, generate syntax tree of p_r



Generation of the BOR Constraint Set (2)

- Second, label each leaf node with the constraint set {(*t*), (*f*)}
- We label the nodes as N_1, N_2, and so on for convenience.



- Notice that N_1 and N_2 are direct descendents of N_3 which is an AND-node

Generation of the BOR Constraint Set (3)

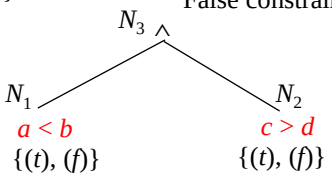
- Third, compute the constraint set for the next higher node in the syntax tree, in this case N_3
- For an AND node, the formulae used are the following.

$$S_{N_3}{}^t = S_{N_1}{}^t \otimes S_{N_2}{}^t = \{(t)\} \otimes \{(t)\} = \{(t, t)\}$$

$$S_{N_3}{}^f = (S_{N_1}{}^f \times \{t_2\}) \cup (\{t_1\} \times S_{N_2}{}^f)$$
$$= (\{(f)\} \times \{(t)\}) \cup (\{(t)\} \times \{(f)\})$$
$$= \{(f, t)\} \cup \{(t, f)\}$$
$$= \{(f, t), (t, f)\}$$

$S_{N_3} = \{(t, t), (f, t), (t, f)\}$

False constraint



Generation of T_{BOR}

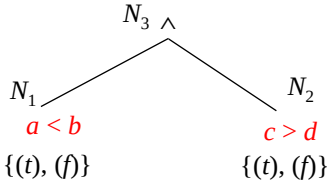
- As per our objective, we have computed the BOR constraint set for the root node of the AST(p_r). We can now generate a test set using the BOR constraint set associated with the root node.

S_{N_3} contains a sequence of three constraints and hence we get a minimal test set consisting of three test cases. Here is one possible test set.

$T_{BOR} = \{t_1, t_2, t_3\}$

$t_1 = < a = 1, b = 2, c = 4, d = 5 >$
$t_2 = < a = 1, b = 0, c = 4, d = 5 >$
$t_3 = < a = 1, b = 2, c = 3, d = 2 >$

$S_{N_3} = \{(t, t), (f, t), (t, f)\}$

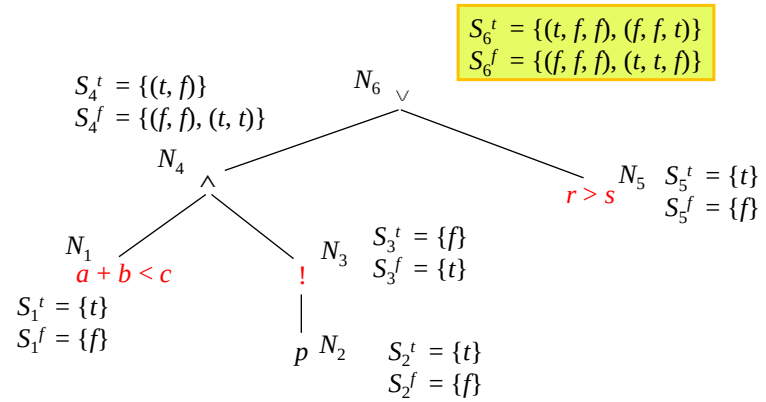


Another Example for T_{BOR} (1)

- Generate the BOR-constraint sets for the predicate $(a + b) < c \wedge !p \vee (r > s)$

Another Example for T_{BOR} (2)

- The abstract syntax tree for $(a + b) < c \wedge !p \vee (r > s)$



Generation of BRO Constraint Set

- Recall that a test set adequate with respect to a BRO constraint set for predicate p_r guarantees the detection of *all combinations of single Boolean operator and relational operator faults*.

BRO Constraint Set

- The BRO constraint set S for relational expression e_1 *relop* e_2
  $S = \{(>), (=), (<)\}$
- The separation of S into its true (S^t) and false (S^f) components depends in *relop*

relop: $>$	$S^t = \{(>)\}$	$S^f = \{(=), (<)\}$
relop: $\geq$	$S^t = \{(>), (=)\}$	$S^f = \{(<)\}$
relop: $=$	$S^t = \{(=)\}$	$S^f = \{(<), (>)\}$
relop: $<$	$S^t = \{(<)\}$	$S^f = \{(=), (>)\}$
relop: $\leq$	$S^t = \{(<), (=)\}$	$S^f = \{(>)\}$

- Note: t_N denotes an element of $S^t{}_N$
  f_N denotes an element of $S^f{}_N$

Algorithm for Generation of BRO Constraint Sets

Procedure for generating a minimal BRO-constraint set from abstract syntax tree of a predicate p_r.

Input : An abstract syntax tree for predicate p_r denoted by $AST(p_r)$. p_r contains only singular expressions.

Output : BRO-constraint set for p_r attached to the root node of $AST(p_r)$.

Procedure: BRO-CSET

Step 1 Label each leaf node N of $AST(p_r)$ with its constraint set $S(N)$. For each leaf node that represents a Boolean variable, $S_N = \{t, f\}$. For each leaf node that is a relational expression, $S_N = \{(>), (=), (<)\}$.

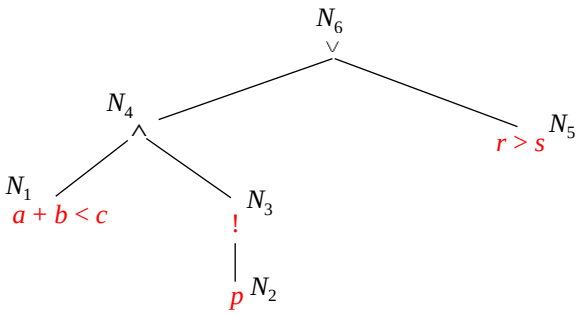
Step 2 Visit each nonleaf node in $AST(p_r)$ in a bottom up manner. Let N_1 and N_2 denote the direct descendants of node N, if N is an AND- or an OR-node. If N is a NOT-node, then N_1 is its direct descendant. S_{N_1} and S_{N_2} are the BRO-constraint sets for nodes N_1 and N_2, respectively. For each nonleaf node N, compute S_N as per Steps 2.1, 2.2, and 2.3 in Procedure BRO-CSET.

Step 3 The constraint set for the root of $AST(p_r)$ is the desired BRO-constraint set for p_r.

End of Procedure BRO-CSET

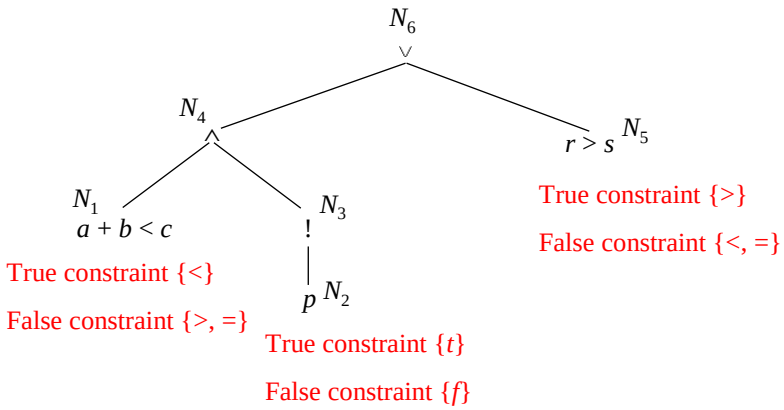
BRO Constraint Set: Example (1)

- p_r: $(a + b < c) \wedge !p \vee (r > s)$
- Step 1: Construct the AST for the given predicate



BRO Constraint Set: Example (2)

- Step 2: Label each leaf node with its constraint set S



BRO Constraint Set: Example (3)

- Step 2: Traverse the tree and compute constraint set for each internal node

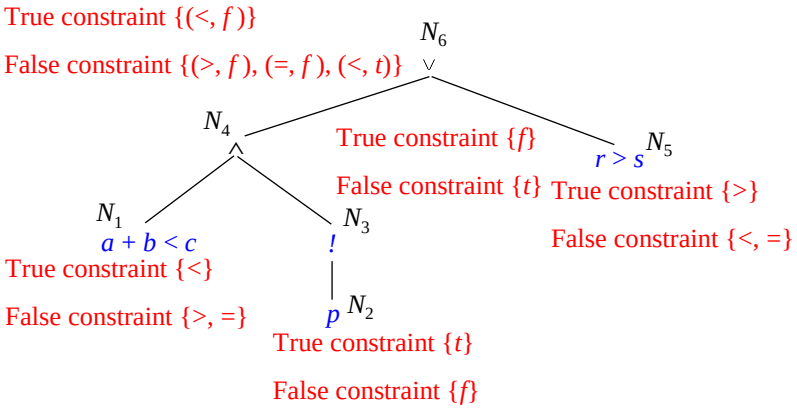
$S^t_{N_3} = S_{N_2}{}^f = \{(f)\}$

$S^f_{N_3} = S_{N_2}{}^t = \{(t)\}$

$S^t_{N_4} = S_{N_1}{}^t \otimes S_{N_3}{}^t = \{(<)\} \otimes \{(f)\} = \{(<, f)\}$

$S^f_{N_4} = (S^f_{N_1} \times \{(t_{N_3})\}) \cup (\{(t_{N_1})\} \times S^f_{N_3})$
$= (\{(>, =)\} \times \{(f)\}) \cup (\{(<)\} \times \{(t)\})$
$= \{(>, f), (=, f)\} \cup \{(<, t)\}$
$= \{(>, f), (=, f), (<, t)\}$

BRO Constraint Set: Example (4)



BRO Constraint Set: Example (5)

- Next compute the constraint set for the root node (this is an OR-node)

$$S^f_{N_6} = S^f_{N_4} \otimes S^f_{N_5}$$
$$= \{(>, f), (=, f), (<, t)\} \otimes \{(=), (<)\}$$
$$= \{(>, f, =), (=, f, <), (<, t, =)\}$$

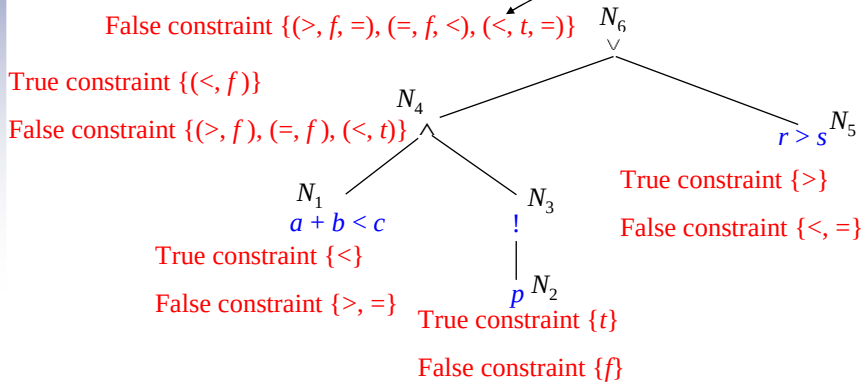
$$S^t_{N_6} = (S^t_{N_4} \times \{(f_{N_5})\}) \cup (\{(f_{N_4})\} \times S^t_{N_5})$$
$$= (\{(<, f)\} \times \{(=)\}) \cup (\{(>, f)\} \times \{(>)\})$$
$$= \{(<, f, =)\} \cup \{(>, f, >)\}$$
$$= \{(<, f, =), (>, f, >)\}$$

BRO Constraint Set: Example (6)

Constraint set for p_r: $(a + b < c) \wedge !p \vee (r > s)$

True constraint {(<, *f*, =), (>, *f*, >)}

False constraint {(>, *f*, =), (=, *f*, <), (<, *t*, =)}



True constraint {<, *f*)}

False constraint {(>, *f*), (=, *f*), (<, *t*)}

True constraint {>}

False constraint {<, =}

True constraint {<}

False constraint {>, =}

True constraint {*t*}

False constraint {*f*}

BRO Constraint Set: In-Class Exercise

- Given the constraint set for p_r: $(a + b < c) \wedge !p \vee (r > s)$, construct T_{BRO}

{(>, *f*, =), (=, *f*, <), (<, *t*, =), (<, *f*, =), (>, *f*, >)}

	$a + b < c$	p	$r > s$	Test case
t_1	>	f	=	$< a = 1, b = 1, c = 1, p = \texttt{false}, r = 1, s = 1 >$
t_2	=	f	<	$< a = 1, b = 0, c = 1, p = \texttt{false}, r = 1, s = 2 >$
t_3	<	t	=	$< a = 1, b = 1, c = 3, p = \texttt{true}, r = 1, s = 1 >$
t_4	<	f	=	$< a = 0, b = 2, c = 3, p = \texttt{false}, r = 0, s = 0 >$
t_5	>	f	>	$< a = 1, b = 1, c = 0, p = \texttt{false}, r = 2, s = 0 >$

Generating the BRE Constraint Set (1)

- We now show how to generate BRE constraints that lead to test cases which guarantee the detection of a single occurrence of *Boolean operator*, *relation operator*, *arithmetic expression*, or *combination* fault in a predicate
- The BRE constraint set for a *Boolean variable* remains $\{t, f\}$ as with the BOR and BRO constraint sets
- The BRE constraint set for a *relational expression* is $\{(+\varepsilon), (=), (-\varepsilon)\}$ where $\varepsilon > 0$

Generating the BRE Constraint Set (2)

- The BRE constraint set S for a relational expression e_1 *relop* e_2 is separated into subsets S^t and S^f based on the following relations

Constraint	Satisfying condition
$+\epsilon$	$0 < e_1 - e_2 \leq +\epsilon$
$-\epsilon$	$-\epsilon \leq e_1 - e_2 < 0$

- Based on the conditions listed above, we can now separate the BRE constraint S into its true and false components as follows

relop : >	$S^t = \{(+\epsilon)\}$	$S^f = \{(=), (-\epsilon)\}$
relop : ≥	$S^t = \{(+\epsilon), (=)\}$	$S^f = \{(-\epsilon)\}$
relop : =	$S^t = \{(=)\}$	$S^f = \{(+\epsilon), (-\epsilon)\}$
relop : <	$S^t = \{(-\epsilon)\}$	$S^f = \{(=), (+\epsilon)\}$
relop : ≤	$S^t = \{(-\epsilon), (=)\}$	$S^f = \{(+\epsilon)\}$

Algorithm for Generation of BRE Constraint Sets (1)

- The procedure to generate a minimal BRE-constraint set is similar to that for BRO and BOR. *The only difference lies in the construction of the constraint sets for the leaves.*

Algorithm for Generation of BRE Constraint Sets (2)

Procedure for generating a minimal BRE-constraint set from an abstract syntax tree of a predicate p_r.

Input: An abstract syntax tree for predicate p_r, denoted by $AST(p_r)$. p_r contains only singular expressions.

Output: BRE-constraint set for p_r attached to the root node of $AST(p_r)$.

Procedure: BRE-CSET

Step 1 Label each leaf node N of $AST(p_r)$ with its constraint set $S(N)$. For each leaf node that represents a Boolean variable, $S_N = \{t, f\}$. For each leaf node that is a relational expression, $S_N = \{(+\epsilon), (=), (-\epsilon)\}$.

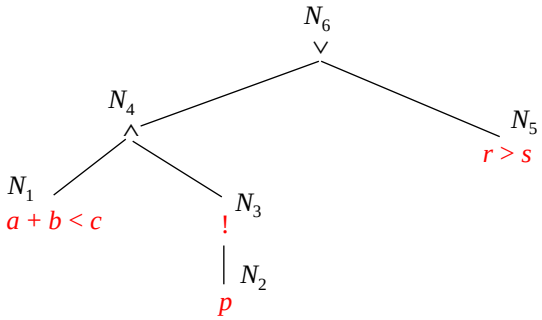
Step 2 Visit each nonleaf node in $AST(p_r)$ in a bottom up manner. Let N_1 and N_2 denote the direct descendants of node N, if N is an AND-or an OR-node. If N is a NOT-node, then N_1 is its direct descendant. S_{N_1} and S_{N_2} are the BRE-constraint sets for nodes N_1 and N_2, respectively. For each nonleaf node N, compute S_N as in Steps 2.1, 2.2, and 2.3 in Procedure BOR-CSET.

Step 3 The constraint set for the root of $AST(p_r)$ is the desired BRE-constraint set for p_r.

End of Procedure BRE-CSET

BRE Constraint Set: Example (1)

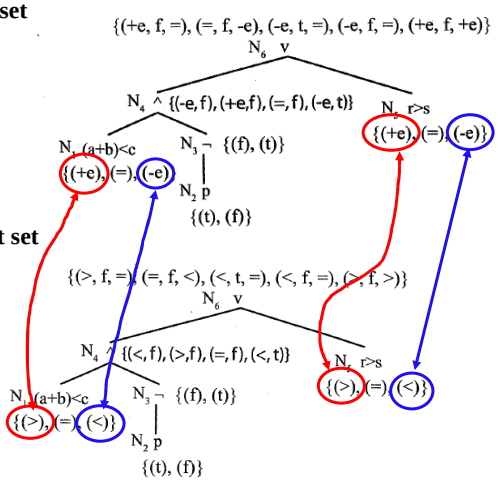
- Generate the constraint set for the predicate p_r: $(a + b < c) \wedge !p \vee (r > s)$



BRE Constraint Set: Example (2)

BRE constraint set

BRO constraint set



BRE Constraint Set: Example (3)

- A sample test set (T_{BRE}) that satisfies the BRE constraints **(ϵ = 1)**

	$a+b<c$	p	$r>s$	Test case
t_1	$+\epsilon$	f	=	$< a=1, b=1, c=1, p=$ false$, r=1, s=1 >$
t_2	=	f	$-\epsilon$	$< a=1, b=0, c=1, p=$ false$, r=1, s=2 >$
t_3	$-\epsilon$	t	=	$< a=1, b=1, c=3, p=$ true$, r=1, s=1 >$
t_4	$-\epsilon$	f	=	$< a=0, b=2, c=3, p=$ false$, r=0, s=0 >$
t_5	$+\epsilon$	f	$+\epsilon$	$< a=1, b=1, c=1, p=$ false$, r=1, s=0 >$

- A sample test set (T_{BRO}) that satisfies the BRO constraints

	$a+b<c$	p	$r>s$	Test case
t_1	>	f	=	$< a=1, b=1, c=1, p=$ false$, r=1, s=1 >$
t_2	=	f	<	$< a=1, b=0, c=1, p=$ false$, r=1, s=2 >$
t_3	<	t	=	$< a=1, b=1, c=3, p=$ true$, r=1, s=1 >$
t_4	<	f	=	$< a=0, b=2, c=3, p=$ false$, r=0, s=0 >$
t_5	>	f	>	$< a=1, b=1, c=0, p=$ false$, r=2, s=0 >$

Singular Boolean Expressions

- Boolean expression: one or more Boolean variables joined by bop
  - Example ($a \wedge b \vee !c$), where a, b, and c are also known as literals
- Singular Boolean expression: When each literal appears only once
  - Example ($a \wedge b \vee !c$)

Mutually Singular Boolean Expressions

- Mutually singular: Boolean expressions e_1 and e_2 are mutually singular when they *do not share any literal*
- Expression e_i is considered *a singular component of* E if and only if e_i is *singular* and is *mutually singular* with each of the other elements of E
- Expression e_i is considered *a non-singular component of* E if and only if it is *non-singular* and is *mutually singular* with each of the remaining elements of E

BOR Constraints for Non-Singular Expressions

- Test generation procedures described so far are for singular predicates. Recall that a singular predicate contains only one occurrence of each variable
- We will now learn how to *generate BOR constraints for non-singular predicates*
- First, let us look at some non-singular expressions, their respective disjunctive normal forms (DNF), and their mutually singular components

Non-Singular Expressions and DNF: Examples

Predicate (p_r)	DNF	Mutually singular components in p_r
$ab(b + c)$	$abb + abc$	a; $b(b + c)$
$a(bc + bd)$	$abc + abd$	a; $(bc + bd)$
$a(bc + !b + de)$	$abc + a!b + ade$	a; $bc + !b + de$

Generating BOR Constraints for Non-Singular Expressions

- We proceed in two steps
  - First we will examine the Meaning Impact (MI) procedure for generating a minimal set of constraints from *a possibly non-singular predicate*
  - Next, we will examine the procedure to generate BOR constraint set for a non-singular predicate

Meaning Impact (MI) Procedure (1)

Input: A Boolean expression $E = e_1 + e_2 + \ldots e_n$ in minimal disjunctive normal form containing n terms. Term e_i, $1 \leq i \leq n$ contains $l_i > 0$ literals.

Output: A set of constraints S_E that guarantees the detection of missing or extra NOT operator fault in a faulty version of E.

Procedure: `MI-CSET`

Step 1 For each term e_i, $1 \leq i \leq n$, construct T_{e_i} as the set of constraints that make e_i true.

Step 2 Let $TS_{e_i} = T_{e_i} - \cup_{j=1, i \neq j}^{n} T_{e_j}$. Note that for $i \neq j$, $TS_{e_i} \cap TS_{e_j} = \emptyset$.

Step 3 Construct S_E^t by including one constraint from each TS_{e_i}, $1 \leq i \leq n$. Note that for each constraint $c \in S_E^t$, $p(c) =$ `true`.

Meaning Impact (MI) Procedure (2)

Step 4 Let e_i^j denote the term obtained by complementing the j^{th} literal in term e_i, for $1 \leq i \leq n$ and $1 \leq j < l_j$. We count the literals in a term from left to right, the leftmost literal being the first. Construct $F_{e_i^j}$ as the set of constraints that make e_i^j true.

Step 5 Let $FS_{e_i^j} = F_{e_i^j} - \cup_{k=1}^{n} T_{ek}$. Thus, for any constraint $c \in FS_{e_i^j}$, $p(c) =$ `false`.

Step 6 Construct S_E^f that is minimal and covers each $FS_{e_i^j}$ at least once.

Step 7 Construct the desired constraint set for E as $S_E = S_E^t \cup S_E^f$.

`End of Procedure MI-CSET`

MI Procedure: Example (1)

- Consider the non-singular predicate: $a(bc + !bd)$
- Its DNF equivalent is $E = abc + a!bd$
- Note that a, b, c, and d are Boolean variables and also referred to as literals
  - Each literal represents a condition
  - For example, a could represent $r < s$
- Recall that + is the Boolean OR operator, ! is the Boolean NOT operator, and as per common convention we have omitted the Boolean AND operator. For example bc is the same as $b \wedge c$

MI Procedure: Example (2)

- Step 0: Identify the DNF equivalent of E as $e_1 + e_2$, where $e_1 = abc$ and $e_2 = a!bd$
- Step 1: Construct a constraint set T_{e_1} for e_1 that makes e_1 true.

  Similarly construct T_{e_2} for e_2 that makes e_2 true

  $T_{e_1} = \{(t, t, t, t), (t, t, t, f)\}$

  $T_{e_2} = \{(t, f, t, t), (t, f, f, t)\}$

  – Note that the four t's in the first element of T_{e_1} denote the values of the Boolean variables a, b, c, and d, respectively. The second element, and others, are to be interpreted similarly.

MI Procedure: Example (3)

- Step 2: From each T_{e_i}, remove the constraints that are in any other T_{e_j}

  This gives us TS_{e_i} and TS_{e_j}

  Note that this step will lead $TS_{e_i} \cap TS_{e_j} = \varnothing$

  – There are no common constraints between T_{e_1} and T_{e_2} in our example.

  Hence we get

  $TS_{e_1} = \{(t, t, t, t), (t, t, t, f)\}$

  $TS_{e_2} = \{(t, f, t, t), (t, f, f, t)\}$

MI Procedure: Example (4)

- Step 3: Construct S^t_E by *selecting one element from each TS*
  – In our case, selecting one test each from TS_{e_1} and TS_{e_2}, we obtain *a minimal set of tests* that *make E true* and cover each term of E as follows

$$S^t_E = \{(t, t, t, f), (t, f, f, t)\}$$

  – Note that
  - There exist four possible S^t_E
  - For each constraint x in S^t_E we get $E(x)$ = true

MI Procedure: Example (5)

- Step 4: For each term in E, obtain terms by complementing each literal, one at a time

$e^1_1 = !abc$ $\quad e^2_1 = a!bc$ $\quad e^3_1 = ab!c$

$e^1_2 = !a!bd$ $\quad e^2_2 = abd$ $\quad e^3_2 = a!b!d$

- From each term e above, derive constraints F_e that make e true. We get the following six sets

MI Procedure: Example (6)

- $F_{e^1_1} = \{(f, t, t, t), (f, t, t, f)\}$
- $F_{e^2_1} = \{(t, f, t, t), (t, f, t, f)\}$
- $F_{e^3_1} = \{(t, t, f, t), (t, t, f, f)\}$
- $F_{e^1_2} = \{(f, f, t, t), (f, f, f, t)\}$
- $F_{e^2_2} = \{(t, t, t, t), (t, t, f, t)\}$
- $F_{e^3_2} = \{(t, f, t, f), (t, f, f, f)\}$

MI Procedure: Example (7)

- Step 5: Now construct FS_e by removing from F_e any constraint that appeared in any of the sets T_e constructed earlier.

$FS_{e^1_1} = F_{e^1_1}$
$FS_{e^2_1} = \{(t, f, t, f)\}$
$FS_{e^3_1} = F_{e^3_1}$

Constraints common with T_{e_1} and T_{e_2} are removed.

$FS_{e^1_2} = F_{e^1_2}$
$FS_{e^2_2} = \{(t, t, f, t)\}$
$FS_{e^3_2} = F_{e^3_2}$

MI Procedure: Example (8)

- Step 6: Now construct S^f_E by selecting one constraint from each FS_e

$$S^f_E = \{(f, t, t, f), (t, f, t, f), (t, t, f, t), (f, f, t, t)\}$$

- Step 7: Now construct $S_E = S^t_E \cup S^f_E$

$$S_E = \{(t, t, t, f), (t, f, f, t), (f, t, t, f), (t, f, t, f), (t, t, f, t), (f, f, t, t)\}$$

- Note: Each constraint in S^t_E makes E true and each constraint in S^f_E makes E false

BOR-MI-CSET Procedure (1)

- The BOR-MI-CSET procedure takes *a possibly non-singular expression E* as input and generates a constraint set that guarantees the detection of Boolean operator faults in the implementation of *E*
- The BOR-MI-CSET procedure using the MI procedure described earlier

BOR-MI-CSET Procedure (2)

Procedure for generating a minimal constraint set for a predicate possibly containing nonsingular expressions.

Input: A Boolean expression E.
Output: A set of constraints S_E that guarantees the detection of Boolean operator faults in E.

Procedure: BOR-MI-CSET

Step 1 Partition E into a set of n mutually singular components, $E = \{E_1, E_2, \ldots E_n\}$.
Step 2 Generate the BOR-constraint set for each singular component in E using the BOR-CSET procedure.
Step 3 **Generate the BOR constraint set for each non-singular component in E using the MI-CSET procedure**
Step 4 Combine the constraints generated in the previous two steps using Step 2 from the BOR-CSET procedure to obtain the constraint set for E.

End of Procedure BOR-MI-CSET

BOR-MI-CSET: Example (1)

- Consider a non-singular Boolean expression: $E = a(bc + !bd)$
- Mutually singular components of E

$e_1 = a$ ← singular
$e_2 = bc + !bd$ ← non-singular

- We use the BOR-CSET procedure to generate the constraint set for e_1 (singular component) and MI-CSET procedure for e_2 (non-singular component)

BOR-MI-CSET: Example (2)

- For component e_1 we get

$$S^t_{e_1} = \{t\}. S^f_{e_1} = \{f\}$$

- Recall that $S^t_{e_1}$ is true constraint set for e_1 and $S^f_{e_1}$ is false constraint set for e_1

BOR-MI-CSET: Example (3)

- Component e_2 is a DNF expression. We can write $e_2 = u + v$ where $u = bc$ and $v = !bd$
- Let us now apply the MI-CSET procedure to obtain the BOR constraint set for e_2
- As per Step 1 of the MI-CSET procedure we obtain
  $T_u = \{(t, t, t), (t, t, f)\}$
  $T_v = \{(f, t, t), (f, f, t)\}$

BOR-MI-CSET: Example (4)

- Applying Steps 2 and 3 to T_u and T_v we obtain

  $TS_u = T_u = \{(t, t, t), (t, t, f)\}$
  $TS_v = T_v = \{(f, t, t), (f, f, t)\}$

  $S^t_{e_2} = \{(t, t, f), (f, t, t)\}$

  One possible choice. Can you think of other alternatives?

- Next we apply Step 4 to u and v. We obtain the following complemented expressions from u and v.
  Note that $u = bc$ and $v = !bd$
  $u_1 = !bc$ $\quad u_2 = b!c$
  $v_1 = bd$ $\quad v_2 = !b!d$

BOR-MI-CSET: Example (5)

- Continuing with Step 4 we obtain

$F_{u_1} = \{(f, t, t), (f, t, f)\}$ $F_{u_2} = \{(t, f, t), (t, f, f)\}$

$F_{v_1} = \{(t, t, t), (t, f, t)\}$ $F_{v_2} = \{(f, t, f), (f, f, f)\}$

- Next we apply Step 5 to the *F* constraint sets to obtain

$FS_{u_1} = \{(f, t, f)\}$ $FS_{u_2} = \{(t, f, t), (t, f, f)\}$

$FS_{v_1} = \{(t, f, t)\}$ $FS_{v_2} = \{(f, t, f), (f, f, f)\}$

BOR-MI-CSET: Example (6)

- Applying Step 6 to the *FS* sets leads to the following

$S^f_{e_2} = \{(f, t, f), (t, f, t)\}$

- Combing the true and false constraint sets for e_2 we get

$S_{e_2} = \{(t, t, f), (f, t, t), (f, t, f), (t, f, t)\}$

BOR-MI-CSET: Example (7)

- Summary

$S^t_{e_1} = \{(t)\}$	$S^f_{e_1} = \{(f)\}$	from BOR-CSET procedure
$S^t_{e_2} = \{(t, t, f), (f, t, t)\}$	$S^f_{e_2} = \{(f, t, f), (t, f, t)\}$	from MI-CSET procedure

- We now apply Step 2 of the BOR-CSET procedure to obtain the constraint set for the entire expression *E*

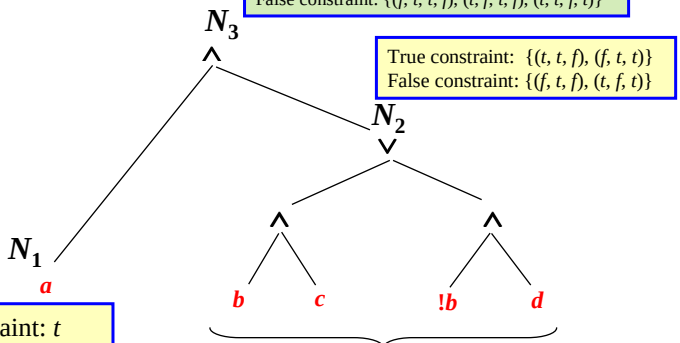
BOR-MI-CSET: Example (8)

Obtained by applying Step 2 of BOR-CSET to an AND node

$S^t_{N_3} = S^t_{N_1} \otimes S^t_{N_2}$

$S^f_{N_3} = (S^f_{N_1} \times \{t_2\}) \cup (\{t_1\} \times S^f_{N_2})$

True constraint: {(*t*, *t*, *t*, *f*), (*t*, *f*, *t*, *t*)}
False constraint: {(*f*, *t*, *t*, *f*), (*t*, *f*, *t*, *f*), (*t*, *t*, *f*, *t*)}



True constraint: {(*t*, *t*, *f*), (*f*, *t*, *t*)}
False constraint: {(*f*, *t*, *f*), (*t*, *f*, *t*)}

True constraint: *t*
False constraint: *f*

Summary (1)

- Most requirements *contain conditions under which functions are to be executed*. Predicate testing procedures covered are excellent means to generate tests to ensure that each condition is tested adequately.

Summary (2)

- Usually one would combine equivalence partitioning, boundary value analysis, and predicate testing procedures to generate tests for a requirement of the following type: