

Testing Boolean Expressions

September 14, 2005

Cem Kaner

Several concepts freely pilfered from

Brian Marick

<http://www.testing.com/writings/half-day-programmer.pdf>

David Gelperin

Booleans?

- An expression that evaluates to `True` (1) or `False` (0)
 - For example:
 - `True` is `True`
 - `(True and False)` is `False`
 - `(5 < 2)` is 0
 - `((2 < 5) OR 1) AND 0` is 0
-

We can lay this out in truth tables

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

We can lay this out in truth tables

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

We can lay this out in truth tables

A	B	Not (A and B)
0	0	1
0	1	1
1	0	1
1	1	0

But sometimes they get a bit unwieldy

A	B	C	D	(A and B) or (C and D)
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1
1	1	0	0	1
1	0	1	1	1
1	0	1	0	0
1	0	0	1	0
1	0	0	0	0
0	1	1	1	1
0	1	1	0	0
0	1	0	1	0
0	1	0	0	0
0	0	1	1	1
0	0	1	0	0
0	0	0	1	0
0	0	0	0	0

Sometimes a wee bit confusing

A	B	C	D	Not(A and B) or (C and Not-D)
1	1	1	1	0
1	1	1	0	1
1	1	0	1	0
1	1	0	0	0
1	0	1	1	1
1	0	1	0	1
1	0	0	1	1
1	0	0	0	1
0	1	1	1	1
0	1	1	0	1
0	1	0	1	1
0	1	0	0	1
0	0	1	1	1
0	0	1	0	1
0	0	0	1	1
0	0	0	0	1

Boolean Challenges

- If (boolean)
 - True → Do the True thing
 - False → Do the False thing
 - Soooooooo ...
 - Every line in the truth table is a different test
 - Combinatorial explosion = 2^N
 - Boolean expressions are abstractions
 - Consider:
IF ($5 < x$)
 - If x is an Int, how many tests are possible?
-

Boolean Strategy – Step 1

- Reduce the number of tests with a domain analysis
 - Consider ($5 < x$)
 - Test with what values for x ?
 - What about the other values?
 - Is it possible to have failures that are triggered by the other values?
-

Boolean Strategy – Step 1

- Ultimately, you reduce the number of tests for a given comparison (or other result) to a minimum
- In analyzing expression, a simple comparison, like $5 < x$, becomes a single True/False column

5	X	$5 < x$
5	5	0
5	6	1
5	4 ?	0
5	0 ?	0
5	MinInt?	0
5	MaxInt?	1
5		

Booleanisms – Step 2

- Assess the complexity of the boolean
 - Is a programmer likely to misunderstand whether $(5 < x)$ is True or False (given that she knows the value of x)?
 - How many programmers are likely to get every decision right for:
(NOT((a OR b) AND NOT-c) OR NOT-d AND
(b OR (NOT-d AND c))) AND A
- For a complex expression, you should test several combinations

Step 3 -- Sampling

- In an expression involving N logical variables, there are 2^N combinations and thus 2^N possible tests
 - Any of these combinations could (in theory) yield a unique failure
 - *Don't forget Hoffman's MASPAR experience*
 - But there's not enough time, so we have to sample
 - The question is, what is our sampling theory?
-

Sampling (2)

- Good testing based on risk
 - Domain testing, for example, focuses on risk of classification error, either by
 - mishandling an entire category
 - so we test every equivalence class
 - or by misclassifying members of a category
 - so we test boundaries as the most possible tests of sub-range mis-classification
 - For binary variables, we have to use a different subsetting strategy
-

Sampling (3)

- In an expression, we have:
 - Variables, $V = 0$ or 1
 - Operators = NOT, OR, AND
 - Parentheses
 - Result, $R = 0$ or 1
 - Let's pretend that the programmer has perfect empirical knowledge:
 - Given values for V_1, V_2, \dots, V_n , the programmer *knows the correct R* , for all combinations of V_i 's.
-

Sampling (4)

- Given perfect empirical knowledge, the error is in the creation of the model (the boolean expression that is supposed to capture the intended relation).
 - If the programmer makes only one mistake:
 - Parenthesis misplaced
 - Evaluation order wrong
 - Scope of an operator wrong
 - Wrong operator (A or B)
 - Missing, added, or misplaced NOT
 - Wrong interpretation of the scope and effect of a NOT
-

Sampling (5)

- We can't do all 2^N cases
 - Suppose we adopt
 - 0-error model
 - 1-error model
 - 2-error model
 - ... N error model
-

Sampling 6: Zero-Error Model

- The expression is coded correctly
 - We test two branches:
 - Expression result TRUE
 - Expression result FALSE
 - We *might* also test special cases that involve side-effects of the calculations (e.g. if we know there is extensive memory cost of a given calculation), but these are an orthogonal dimension to the logic-combination so I'll skip them
-

Sampling 7: One-Error Model

- Consider expression

a AND b AND c

- Make exactly one mistake:

■ (a OR b) AND c	a AND b
■ a OR (b AND c)	a AND c
■ a AND (b OR c)	b AND c
■ (a AND b) OR c	a
■ (a OR c) AND b	b
■ NOT-a AND b AND c	c
■ a AND NOT-b AND c	a AND b AND NOT-c

- The sampling strategy is to pick the smallest set of tests that will cover these mistakes

a	b	c	a&b&c	(a b)&c	a (b&c)	a&(b c)	(a&b) c	(a c)&b	a b c	a
1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	1	1	1	1	1
1	0	1	0	1	1	1	0	0	1	1
1	0	0	0	0	1	0	0	0	1	1
0	1	1	0	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	0	0	0
a	b	c	b	c	a&b	a&c	b&c	~a&b&c	a&~b&c	a&b&~c
1	1	1	1	1	1	1	1	0	0	0
1	1	0	1	0	1	0	0	0	0	1
1	0	1	0	1	0	1	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	0	0
0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

White, red, orange → 0; blue → 1

a	b	c	$a \parallel (b \& c)$	$(a \parallel b) \& c$	$a \& b \& c$	$a \parallel b \parallel c$	$\sim a \parallel (b \& c)$	$a \parallel \sim (b \& c)$
1	1	1	1	1	1	1	1	1
1	1	0	1	0	0	1	0	1
1	0	1	1	1	0	1	0	1
1	0	0	1	0	0	1	0	1
0	1	1	1	1	0	1	1	0
0	1	0	0	0	0	1	1	1
0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	1	1
a	b	c	$a \parallel (\sim b \& c)$	$a \parallel (b \& \sim c)$	a	$(b \& c)$	$a \parallel b$	$a \parallel c$
1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	0	1	1
1	0	1	1	1	1	0	1	1
1	0	0	1	1	1	0	1	1
0	1	1	0	0	0	1	1	1
0	1	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0

Another One-Error Example --- Red, orange \rightarrow 1, grey, purple \rightarrow 0