

A

1.

- a) **“Location of element in list” fails the disjointness property. Give an example that illustrates this.**

The disjointness property says that blocks must not overlap. Disjointness property states that no element of domain is in more than one block.

Example 1:

List with 3 repeated elements:

ArrayList = [1,1,1]

element = 1

Block 1: First entry in list = 1

Block 2: Last entry in list = 1

Block 3: Some other position in list = 1

The “location of element in list” fails this property because an example list could fall under multiple blocks. For example, if the same element is repeated multiple times in an array list [1, 1, 1] and we were searching for 1, the input would fall in block 1, 2, and 3. There are many ways to demonstrate that this fails the disjointness property.

Example 2:

List with only 1 element:

ArrayList = [6]

element = 6

Block 1: First entry in list = 6

Block 2: Last entry in list = 6

If there is only one element in the array [6] and we’re searching for 6, this is both the first and last element in the array (block 1 and block 2). Therefore, since 6 is in 2 blocks, it does not satisfy the disjointness property.

- b) **“Location of element in list” fails the completeness property. Give an example that illustrates this.**

The completeness property says that the blocks must cover the entire domain. The three blocks don’t describe if there is an input where the element being searched is not in the array. Thus, these three blocks don’t cover all possibilities.

Example 1:

List with 3 elements:

ArrayList = [2, 4]

element = 3

Does not fall/is not included in any block

c) Supply one or more new partitions that capture the intent of “Location of element in list” but do not suffer from completeness or disjointness problems.

- “

To make sure that the new partitions do not suffer from completeness or disjointness, check whether these properties are true or false.

Block 1:

- If element is the first entry in the list → [true | false]

Block 2:

- If element is the last entry in the list → [true | false]

2.

2.1 List all of the input variables, including the state variables

1 textbox, 1 test requirement button (truth table), 5 test buttons (GACC, CACC, RACC, GICC, RICC), 4 other buttons (New Expression, Graph Coverage, Data Flow Coverage, Minimal - MUMCUT coverage), browser features (back, forward, refresh)

2.2 Define characteristics of the input variables. Make sure you cover all input variables

- C1: Textbox (P =) → valid string or not
- C2: Test Requirement button (Truth Table) → click or not
- C3: 5 Test Buttons (GACC, CACC, RACC, GICC, RICC) → which button
- C4: 4 Other buttons (New Expression, Graph Coverage, Data Flow Coverage, Minimal - MUMCUT coverage) → which button
- C5: Browser back → click or not

2.3 Partition the characteristics into blocks

- C1: Input to P textbox is a valid string or not [valid string | invalid string]
- C2: Truth Table is clicked or not [click | not click]
- C3: Which Logic Test Type button is clicked [GACC | CACC | RACC | GICC | RICC | None]
- C4: Which other button is clicked [New Expression | Graph Coverage | Data Flow Coverage | Minimal - MUMCUT coverage | None]
- C5: Browser back [click back | not click]

2.4 Designate one block in each partition as the "Base" block

- C1: Input to P textbox is a valid logical statement or not
 - Base: Input is valid string
- C2: Truth Table is clicked or not
 - Base: Truth Table button is clicked
- C3: Which Logic Test Type button is clicked
 - Base: GACC
- C4: Which other button is clicked
 - Base: "New Expression" is clicked
- C5: Browser back is clicked or not
 - Base: Browser back is not clicked

2.5 Define values for each block

- C1: Input to P textbox is a valid logical statement or not
 - [Valid: " $q \wedge p$ "] [Invalid: "67"]
- C2: Truth Table is clicked
 - [click | not click]
- C3: Which Logic Test Type button is clicked.
 - [GACC | CACC | RACC | GICC | RICC | None]
- C4: Which other button is clicked
 - [New Expression | Graph Coverage | Data Flow Coverage | Minimal - MUMCUT coverage | None]
- C5: Browser back is clicked
 - [click back | not click back]

2.6 Write a test set (a set of test cases) that satisfies Base Choice Coverage (BCC)

Write your tests with the values from the previous step

Be sure to include inputs (test input values) and expected outputs

- TC1 (Base) = [$q \wedge p$, Truth Table is clicked, GACC, None, browser back is not clicked]
 - Expected:

The following result for GACC is based on the truth table on Truth Table:
the right:

Major Clause	Set of possible tests
q	(1,3), (1,4), (2,3), (2,4)
p	(1,2), (1,4), (3,2), (3,4)

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

Number of Test Cases: $1 + (2-1) + (2-1) + (6-1) + (5-1) + (2-1) = 13$

- TC2: [$q \wedge p$, Truth Table is clicked, GACC, None, **browser back is clicked**]

- Expected:

Are you sure you want to send a form again?

To reopen this page Safari must resend a form. This might result in duplicate purchases, comments, or other actions.

Cancel Send

Please enter your logic expression in the text box below, using the following logic operators. The variable names can be any string. Parentheses can be used in the logic expression. (Please be aware that this application gets slow for expressions with more than 5 or 6 variables.)

Not : ! And : & Or : |

Implication : > Exclusive Or : ^ Equivalence : =

P =

Test Requirements:

Tests:

Others:

Share Expression:

The following result for GACC is based on the truth table on the right:

Major Clause	Set of possible tests
q	(1,3), (1,4), (2,3), (2,4)
p	(1,2), (1,4), (3,2), (3,4)

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- TC3: [q ^ p, Truth Table is clicked, GACC, **New Expression clicked**, browser back is not clicked]

- Expected:

Logic Coverage Web Application

Please enter your logic expression in the text box below, using the following logic operators. The variable names can be any string. Parentheses can be used in the logic expression. (Please be aware that this application gets slow for expressions with more than 5 or 6 variables.)

Not : ! And : & Or : |

Implication : > Exclusive Or : ^ Equivalence : =

P =

Test Requirements:

Tests:

Others:

Share Expression:

- There is no truth table or set of possible tests
- TC4: [q ^ p, Truth Table is clicked, GACC, **Graph Coverage clicked**, browser back is not clicked]

- Expected:

Graph Coverage Web Application

Graph Information

Please enter your graph edges in the text box below. Put each edge in one line. Enter edges as pairs of nodes, separated by spaces.(e.g.: 1 3) <div style="border: 1px solid #ccc; height: 40px; margin-top: 5px;"></div>	Enter initial nodes below (can be more than one), separated by spaces. If the text box below is empty, the first node in the left box will be the initial node. <div style="border: 1px solid #ccc; height: 20px; margin-top: 5px;"></div>	Enter final nodes below (can be more than one), separated by spaces. <div style="border: 1px solid #ccc; height: 20px; margin-top: 5px;"></div>
--	---	--

Test Requirements: Nodes Edges Edge-Pair Simple Paths Prime Paths

Test Paths: Algorithm 1: Slower, more test paths, shorter test paths Node Coverage Edge Coverage

Edge-Pair Coverage Prime Path Coverage

Algorithm 2: Faster, fewer test paths, longer test paths Edge Coverage Edge-Pair Coverage

Prime Path Coverage

Algorithm 1 is our original, not particularly clever, algorithm to find test paths from graph coverage test requirements. In our 2012 ICST paper, *"Better Algorithms to Minimize the Cost of Test Paths,"* we described an algorithm that combines test requirements to produce fewer, but longer test paths (algorithm 2). Users can evaluate the tradeoffs between more but shorter test paths and fewer but longer test paths and choose the appropriate algorithm.

Other Tools: New Graph Data Flow Coverage Logic Coverage Minimal-MUMCUT Coverage

- TC5: [q ^ p, Truth Table is clicked, GACC, **Data flow coverage clicked**, browser back is not clicked]
 - Expected:

Data Flow Graph Coverage Web Application

Graph Information

Please enter your graph edges in the text box below. Put each edge in one line. Enter edges as pairs of nodes, separated by spaces.(e.g.: 1 3) <div style="border: 1px solid #ccc; height: 40px; margin-top: 5px;"></div>	Enter initial nodes below (can be more than one). If the text box below is empty, the first node in the left box will be the initial node. <div style="border: 1px solid #ccc; height: 20px; margin-top: 5px;"></div>	Enter final nodes below (can be more than one), separated by spaces. <div style="border: 1px solid #ccc; height: 20px; margin-top: 5px;"></div>
--	--	--

Data Flow Information

Please enter your defs in the text box below. Put one variable and all defs for the variable in one line, separated by spaces. Put the variable name at the beginning of the line.(e.g.: x 1 2) <div style="border: 1px solid #ccc; height: 40px; margin-top: 5px;"></div>	Please enter your uses in the text box below. Put one variable and all uses for the variable in one line, separated by spaces. Put the variable name at the beginning of the line.(e.g.: x 3 4 2,3) <div style="border: 1px solid #ccc; height: 40px; margin-top: 5px;"></div>
---	---

Test Requirements: DU Pairs DU Paths

Test Paths: All Def Coverage All Use Coverage All DU Path Coverage

Others: New Graph New DU Info Graph Coverage Logic Coverage Minimal-MUMCUT Coverage

- TC6: [q ^ p, Truth Table is clicked, GACC, **Minimal - MUMCUT coverage clicked**, browser back is not clicked]
 - Expected:

Minimal MUMCUT Coverage Web Application

Please check that all literals are lower case letters or upper case letters (No combination of lower case letters and upper case letters allowed), preceded by an optional '!' to denote negation. Please check that literals are separated by ' & ' and terms are separated by ' | '. A correct example is : a & b | c & d. In the result, 0 represents FALSE and 1 represents TRUE; the order of 0s and 1s in the test points maps to the order of the variables entered in the predicate. For example, if the predicate is a & b, then 10 means a = 1 and b = 0. If the predicate is entered as b & a, then 10 still means that a = 1 and b = 0. (Please be aware that this application gets slow for expressions that have more than 5 or 6 variables.)

Not : ! And : & Or : |

P =

Please check that infeasible literal combinations are separated by semicolons and that literal assignments to 0 or 1 are separated by commas. For example if a = 0 and b = 0 is infeasible and if c = 0 and d = 0 are infeasible, the format is a = 0, b = 0; c = 0, d = 0.

Enter infeasible literal combination (if any) in format a = 0, b = 0; c = 1, d = 1:

Enter a positive integer for the test set size (only for Fault Detection Maximization):

[Go to the help page](#)

Criteria:

Others:

- TC7: [q ^ p, Truth Table is clicked, **CACC**, None, browser back is not clicked]
 - Expected:

The following result for CACC is based on the truth table on the right:

Major Clause	Set of possible tests
q	(1,3), (2,4)
p	(1,2), (3,4)

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- TC8: [q ^ p, Truth Table is clicked, **RACC**, None, browser back is not clicked]
 - Expected:

The following result for RACC is based on the truth table on the right:

Major Clause	Set of possible tests
q	(1,3), (2,4)
p	(1,2), (3,4)

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- TC9: [q ^ p, Truth Table is clicked, **GICC**, None, browser back is not clicked]
 - Expected:

The following result for GICC is based on the truth table on the right:

Major Clause	Set of possible tests
--------------	-----------------------

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- TC10: [$q \wedge p$, Truth Table is clicked, **RICC**, None, browser back is not clicked]
 - Expected:

The following result for RICC is based on the truth table on the right:

Major Clause	Set of possible tests

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- TC11: [$q \wedge p$, Truth Table is clicked, **None**, None, browser back is not clicked]
 - Expected:

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- TC12: [$q \wedge p$, **Truth Table is not clicked**, GACC, None, browser back is not clicked]
 - Expected:

The following result for GACC is based on the truth table on the right:

Major Clause	Set of possible tests
q	(1,3), (1,4), (2,3), (2,4)
p	(1,2), (1,4), (3,2), (3,4)

Truth Table:

Row#	q	p	P	Pq	Pp
1	T	T		T	T
2	T		T	T	T
3		T	T	T	T
4				T	T

- Although the truth table was not clicked, when we clicked the GACC button, it still appeared
- TC13: [67, Truth Table is clicked, GACC, None, browser back is not clicked]
 - Expected:

unexpected char: '6'

- Since the input was invalid, we received the response above