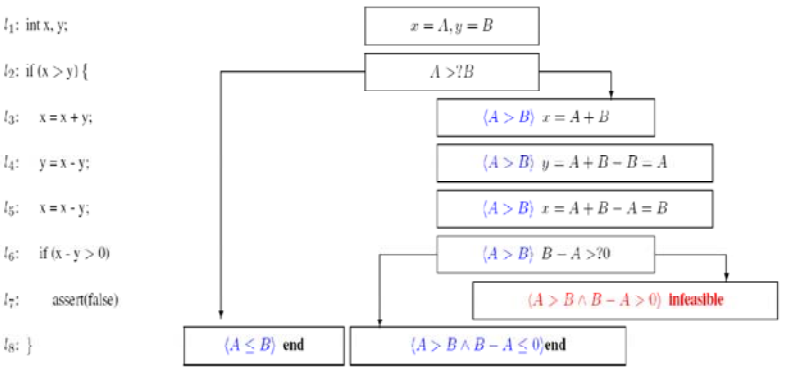
APPENDIX A: *Symbolic Execution by Example*



Example of symbolic execution and path conditions

The diagram in the above figure illustrates how symbolic execution works. The execution keeps a symbolic value for each variable in the program and also keeps a path condition that must be true for the execution to proceed. The execution starts with the variables x and y assuming the symbolic values of "A" and "B" respectively. Two branches are possible at the first if condition on line l_2; the first moves to line l_8 at the end of the program with a path condition of "A ≤ B" and the second goes to line l_3 inside the body of the if condition with a path condition of "A > B". The symbolic execution updates the values of x and y with each statement. It uses a symbolic solver such as an SMT solver to resolve and simplify the values and the path conditions. For example, this enables simplifying the path condition expression "A+B-A" to "A" on line l_4.

Lines l_7 and l_8 deserve the most attention in this example. Note how an SMT solver can easily decide that line l_7 is infeasible since the path condition corresponding to it is not valid. And that line l_8 can execute under the disjunction of the two conditions corresponding to it.

APPENDIX B: *PBCOV Analysis of the Motivation Example*

This appendix illustrates the usage of PBCOV by walking through the selection sort example presented in Section 2. The property P claimed in the sort function states that "*forall k, 0 ≤ k < n-1 → a[k] ≤ a[k+1]*". This property makes sure that the entries in the resulting array are all in order.

To enable the user to annotate code with properties, PBCOV provides the macro PBCOV_ASSERT. We annotated the sort function and added properties to claim order after sort finishes execution. The annotated function is presented in the figure. Note how the PBCOV_ASSERT macro is used on lines 19-21: a) a loop is needed to simulate the *forall* quantifier, and b) the implication operator was substituted by its C language equivalent, i.e., (a → b) is substituted by (!a || b).

```
0 #include "pbcov_assert.h"
1 void sort ( int a[], int n) {
            … lines 2 to 18 from Figure 1, Section 2
19      for (int k= 0; k < n-1; k++) {
20          PBCOV_ASSERT ( !(k >= 0 && k < n) || (a[k+1] >= a[k]) );
21      }
22 }
```

Annotated selection sort algorithm

Recall that property P has three atomic predicate terms p_1 = (0 ≤ k), p_2 = (k < n-1) and p_3 = (a[k] ≤ a[k+1]), where P = !(p_1 && p_2) || p_3. The state space of P comprises eight states, seven of which satisfy P, and one {110} does not. In this failing state, k is in the correct range and satisfies (p_1 && p_2), but at least two of the entries are not in order, thus violating p_3.

PBCOV instruments the program and runs the instrumented program against the test suite T presented in Section 2. Test cases t_1 through t_6 from T cover state {111} and test cases t_7 and t_8 do not exercise the property and thus cover no states. PBCOV expresses the states covered by the test suite in a truth table format and consequently computes P_{pass} = p_1 && p_2 && p_3 and P_{fail} = false. Then it computes P_{sym} and P_{cover} as described in Section 3.2 and builds the equivalence check circuit and passes it to a model checker. The model checker returns the missing states where P_{cover} and P_{sym} differ which are: {000}, {001}, {010}, {011}, {100}, {101}, and {110} where P is true for all the states except the last.

The missing states reported by the model checker might not all be feasible. For example, looking only at lines 19 and 20, it is impossible for p_1 and p_2 to be both false at the same time and thus {000} and {001} are not feasible.

PBCOV expresses each missing state in a formula s_{miss} which is the conjunction of the atomic terms representing the state. For example, for states {000} and {001} s_{miss} has the forms $(!p_1 \;\&\&\; !p_2 \;\&\&\; !p_3)$ and $(!p_1 \;\&\&\; !p_2 \;\&\&\; p_3)$, respectively. PBCOV passes the formula ($F = s_{miss} \;\&\&\; S_{sym}$) to an SMT solver and checks whether F is satisfiable. If the formula is satisfiable, then the state s_{miss} is feasible, otherwise, it is not.

For our example, it turns out that six out of the seven missing states are infeasible, since the array index k of the assert statement iterates from 0 to $n-2$. Consequently, p_1 and p_2 are always true regardless of any test case. Thus all the missing states where p_1 and p_2 evaluate to false are infeasible. As a result, the reachable state space includes {111} where the property passes and {110} where the property fails. It is worth noting that the SMT solver returned a conclusive result in this example, however, in more complex situations, this might not be the case and PBCOV may need to compute an over-approximation of the reachable state space as discussed in the previous section. PBCOV computes n_{cov}^{true}= 1, n_{cov}^{false}= 0, n_{feas}^{true}= 1, n_{cov}^{false}= 1, and $m_{pbcov} = \log(1 + n_{cov}^{true} + n_{cov}^{false}) / \log(1 + n_{feas}^{true} + n_{feas}^{false}) = 0.63$, which suggests weak property coverage given that the reachable state space was not over-approximated.

Finally, the SMT solver generated the input vector e = <1, 0, -1, 0> as a satisfying assignment for the {110} state. This test case reveals the bug as it results in the erroneous output=<0, -1, 0, 1> and should be included in the test suite. Applying PBCOV again with the augmented tests suite we obtain n_{cov}^{false}= 1, and consequently m_{pbcov} = 1.

APPENDIX C: *Subject Programs and Properties*

GZIP

GZIP is a GNU utility that compresses files using a variation of the Limpel-Ziv algorithm [Ziv and Limpel 1978]. The SIR repository provides the source code of five releases of GZIP; releases 1.1.2, 1.2.2, 1.2.3, 1.2.4, and 1.3. We applied our study to release 1.1.2. SIR provides the source code of GZIP with 16 seeded bugs and a test suite encompassing 214 test cases. Each test case comprises an input file, a set of configuration options, and a script to perform cleanup after each GZIP invocation. The SIR documentation states that the provided test suite can detect all the bugs, and that none of the bugs is detected by more than 25% of the test cases. Surprisingly, we observed that it achieved moderate structural coverage and failed to detect 9 out of the 16 seeded bugs in all of the five releases; consequently, only 7 versions of release 1.1.2 with seeded bugs were relevant to our study. Moreover, one bug is detected by 198 out of 214 test cases, and 3 bugs are detected by less than 3 test cases.

A number of the GZIP functions are well documented with semi-formal English pre-conditions and post-conditions, which we formalized and embedded in the code. We did this for all functions that are called two levels deep from the main function of GZIP, and ended up with 16 assert statements. We annotated GZIP with the 16 properties and applied PBCOV to the resulting code a property at a time.

Memory Manager

The MMan program is an illustrative example from the reference manual of the ANSI C Specification Language (ACSL) [Baudin et al. 2009]. ACSL is designed to specify behavioral properties of C programs. MMan emulates memory allocation using a hierarchical memory structure. MMan takes as input arguments an initialized memory pool and the size of the memory to be allocated; it returns a pointer to the newly allocated memory chunk. Consequently, the ACSL property that describes this function states that the input memory pool must be valid as a pre-condition. As a post-condition, the function must return a NULL chunk if the input size is zero. Also if the size is larger than zero, the function must return NULL if the allocation failed, otherwise it must return a valid pointer and must flag the corresponding memory chunk as used.

We used Crest to automatically generate a test suite for MMan with 1,124 test cases. Each test case consists of a vector of five numbers. Each number corresponds to an argument for one invocation of the function. This test suite achieved near perfect

structural coverage as shown in Table 2. In addition, we seeded MMan with six bugs which included operand and operator mutation and code deletion. The generated test suite was able to detect all of the seeded bugs.

RBBST

A Red Black Binary Search Tree (RBBST) [Cormen et al. 2009] is a self balanced binary search tree that uses node colors to maintain a logarithmic balanced depth. Any valid RBBST obeys the following properties: 1) the color of each node is either red or black; 2) the color of the root of the tree is always black; 3) the color of all the leave nodes is black; 4) the color of the children of red colored nodes must be black; 5) every simple path from a node to any of its descendent leaves must contain the same number of black colored nodes.

Given an element and an RBBST, the Insert (Remove) function first places (removes) the element from the RBBST as if the RBBST were an ordinary Binary Search Tree. Second, the Insert and Remove functions rotate and re-color the nodes of the RBBST to ensure the that the properties above hold for the RBBST. RBBST is used as a benchmark in the context of software testing and verification [Griesmayer et al. 2007] because of its interesting properties, its wide application in map structures, and its considerable code complexity (e.g., loops and recursion).

We downloaded a C implementation of RBBST from [Martinian 2010] and automatically generated a test suite using UDITA. The generated test suite contained 616 test cases. Each test case consisted of an existing RBBST with up to eight-elements and one element to test either the Insert or the Remove functions. We seeded the implementation with twelve bugs. The test suite we generated achieved near perfect structural coverage (see Table 2), and was able to detect all of the seeded bugs.

In addition to the five coloring properties of the RBBST, we included several more properties to claim the size consistency as well as the acyclicity of the tree. We describe all the claimed properties in Table C.1. The variable "current_node" denotes the node currently visited in the post-order traversal, and "bdepth" specifies the number of black colored nodes from the node to its leaf following the path always to the left. It is worth noting that it could be as well the number of black colored nodes in the right-most path, this value is the same as that of the leftmost path in a consistent tree; otherwise, "bdepth" will show discrepancy at some point while traversing the tree which indicates an invalid RBBST.

Finally, we built a conjunction of all the nine claims presented in Table C.1 into one property P_{rbbst}. We asserted P_{rbbst} into one PBCOV_ASSERT and embedded that in a post-order traversal of the tree.

Table C.1. Properties used in the RBBST subject program

	Sub-property	Explanation
1	current_node = root ➔ current_node→color = black	Root is black colored
2	current_node→left ≠ NULL ➔ current_node→key ≥ current_node→ left→key	BST Property
3	current_node→right ≠ NULL ➔ current_node→key ≤ current_node→ right→key	
4	current_node→color = red ➔ current_node→right→color = current_node→left→color = black	Both children of a red colored node are black colored
5	current_node = root ➔ number of counted nodes = theoretical size	Size consistency, and connected tree property
6	node→left→color = node→right→color ➔ node→right→bdepth = node→left→bdepth	Every simple path from a node to any of its descendents must contain the same number of black colored nodes
7	node→left→color ≠ node→right→color ➔ |node→right→bdepth - node→left→bdepth| = 1	
8	current_node→visited = false	Acyclic tree property
9	current_node = root ➔ input key was visited in tree	Input key was inserted in the tree (in case insert function was called)

Sorted Linked List (SLL)

A sorted linked list is an ordinary linked list data structure that stores all items in increasing order of item keys. The Insert function ensures this property by inserting the new element in the correct place. Moreover, the Remove function is the same as the ordinary list removal function. We used the implementation of both the insertion and deletion functions provided in the "Find the bug" book [Barr 2002]. Function Insert takes a pointer to the head of the list and the new element to insert as arguments and returns the new head of the list after insertion. Similarly, function Remove takes a pointer to the head of the list and the element to remove. It returns the new head of the list after removal and a pointer to the removed node if the element was found; NULL otherwise.

In addition to the seeded bugs described in the book, we added our own to have a total of three seeded bugs in the Insert function and another three in the Remove function. We automatically generated a test suite that consisted of 1,751 test cases, where each test case corresponds to a vector of seven elements. The first four are inserted and the last three are

removed from the list. Also in this case, the test suite achieved near perfect structural coverage and was able to reveal all the relevant bugs.

We used two properties to describe the required functionality. For Insert, the property indicated that the list contains no cycles, the number of elements in the list has been incremented, the list contains the newly inserted element, and the elements are sorted. The property describing Remove indicated that the list is still sorted, has no cycles, the number of occurrences of the element to be removed is decremented if it was larger than zero, and the number of elements in the list was decremented and the removed element is returned if the key was found.

TCAS

TCAS is a safety critical embedded application that aims at preventing midair collisions in an airplane. It is a two dimensional alert system that alerts the pilot in the case of an imminent collision. TCAS calculates the time it takes for two airplanes to be in close contact. It also identifies two zones in the vicinity of an airplane. The first zone is the larger zone in which TCAS issues a Traffic Advisory (TA) alert to announce an incoming airplane. In case the threat of collision persists and the airplane is about to enter the inner and smaller danger zone, TCAS issues a Resolution Advisory (RA), with instructions to maneuver either Upward by climbing, or Downward by descending. TCAS is widely used as sample program to apply verification techniques as it is a safety critical program that needs to be properly tested and verified.

In theory, any implementation of TCAS must comply with the TCAS II manual. The Siemens benchmark provides a C implementation of TCAS that encompasses 173 lines of code contained in 9 functions and that complies with the TCAS II manual. Siemens researchers provide 41 variants of TCAS, where each variant encloses one seeded bug. The 41 versions cover bugs with different types such as mutations of constants, operands, and operators, logic changes, as well as missing code. TCAS comes with a test suite that contains 1608 test cases, detects the bugs in all of the 41 variants, and achieves high structural coverage as shown in Table 3.

The function that evaluates the collision status is called alt_sep_test, it takes as input 14 parameters and outputs one parameter. The output parameter instructs the plane to climb (UPWARD_RA), descend (DOWNWARD_RA), or do nothing (UNRESOLVED). The relevant input parameters are: Own_Tracked_Alt, Other_Tracked_Alt, Positive_RA_Alt_Thresh, Up_Separation, and Down_Separation.

The work in [Ammann and Black 2001] defines five separate formal properties for TCAS. These properties specify the relation of the expected output to the input values. Each property is the conjunction of two sub-properties, where each sub-property has the form of a pre-condition implying a condition on the result value. The first property, shown below, makes sure that the result is not an order to descend if the upward separation is bigger than an altitude threshold and the down separation is smaller than the same altitude threshold. It also makes sure that the result is not an order to climb if the upward separation is smaller than the altitude threshold, and the down separation is bigger than the same altitude threshold. The reader can find more details about the rest of the properties in [Ammann and Black 2001] .

P1-A: *Up_Separation ≥ Positive_RA_Alt_Thresh □ Down_Separation < Positive_RA_Alt_Thresh → result ≠ DOWNWARD_RA*

P1-B: *Up_Separation < Positive_RA_Alt_Thresh □ Down_Separation ≥ Positive_RA_Alt_Thresh→ result ≠ UPWARD_RA*

APPENDIX D: *PBCOV metric detailed results*

Table D.1. PBCOV metric results for GZIP properties 13, 14, 15, and 16

Property	Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
13	original	T_{full}	5	0	6	0	92.08	92.08
	1,2,3,4	T_{full}	5	0	6	0	92.08	92.08
		T_{pass}	5	0	6	0	92.08	92.08
	5	T_{full}	5	0	6	0	92.08	92.08
		T_{pass}	4	0	6	0	82.71	82.71
	6	T_{full}	4	0	6	0	82.71	82.71
		T_{pass}	2	0	6	0	56.46	56.46
	7	T_{full}	4	0	6	0	82.71	82.71
		T_{pass}	3	0	6	0	71.25	71.25
14	original	T_{full}	2	0	3	1	68.27	79.25
	1,6,7	T_{full}	2	0	3	1	68.27	79.25
		T_{pass}	1	0	3	1	43.07	50
	2,4,5	T_{full}	2	0	3	1	68.27	79.25
		T_{pass}	2	0	3	1	68.27	79.25
	3	T_{full}	2	0	3	1	68.27	79.25
		T_{pass}	0	0	3	1	0	0
15	original	T_{full}	1	1	6	2	50	N/A
	1,3,4, 5,6,7	T_{full}	1	1	6	2	50	N/A
		T_{pass}	1	1	6	2	50	N/A
	2	T_{full}	0	2	6	2	50	N/A
		T_{pass}	0	2	6	2	50	N/A
16	original	T_{full}	1	0	2	0	63.1	63.1
	1,2,4,5, 6,7	T_{full}	1	0	2	0	63.1	63.1
		T_{pass}	1	0	2	0	63.1	63.1
	3	T_{full}	0	0	2	0	0	0
		T_{pass}	0	0	2	0	0	0

Table D.2. Results for PBCOV metrics for the RBBST Insert

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	T_{full}	18	0	181	6,219	33.6	56.59
1	T_{full}	18	12	181	6,219	39.19	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59
2	T_{full}	18	14	181	6,219	39.9	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59
3,4,6,7	T_{full}	18	0	181	6,219	33.6	56.59
	T_{pass}	18	0	181	6,219	33.6	56.59
5	T_{full}	19	3	181	6,219	35.78	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59
8	T_{full}	18	10	181	6,219	38.43	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59
9	T_{full}	24	26	181	6,219	44.87	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59
10,11	T_{full}	18	4	181	6,219	35.78	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59
12	T_{full}	18	5	181	6,219	36.27	N/A
	T_{pass}	18	0	181	6,219	33.6	56.59

Table D.3. Results for PBCOV metrics for the RBBST Remove

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	T_{full}	18	0	180	3,404	35.98	56.65
1	T_{full}	19	0	180	3,404	36.61	57.63
	T_{pass}	19	0	180	3,404	36.61	57.63
2	T_{full}	18	7	180	3,404	39.81	N/A
	T_{pass}	18	0	180	3,404	35.98	56.65
3,5	T_{full}	18	0	180	3,404	35.98	56.65
	T_{pass}	18	0	180	3,404	35.98	56.65
4	T_{full}	20	42	180	3,404	50.63	N/A
	T_{pass}	18	0	180	3,404	35.98	56.65
6	T_{full}	18	16	180	3,404	43.44	N/A
	T_{pass}	18	0	180	3,404	35.98	56.65
7	T_{full}	18	1	180	3,404	36.61	N/A
	T_{pass}	18	0	180	3,404	35.98	56.65
8	T_{full}	18	4	180	3,404	38.32	N/A
	T_{pass}	18	0	180	3,404	35.98	56.65

Table D.4. PBCOV results for the memory manager program

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	T_{full}	2	0	37	11	28.23	30.21
1	T_{full}	2	1	37	11	35.63	N/A
	T_{pass}	2	0	37	11	28.23	30.21
2,4,5,6	T_{full}	5	1	37	11	50	N/A
	T_{pass}	2	0	37	11	28.23	30.21
3	T_{full}	4	1	37	11	46.04	N/A
	T_{pass}	2	0	37	11	28.23	30.21

Table D.5. PBCOV results for Sorted Linked List Insert program

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	T_{full}	3	0	3	0	100	100
1	T_{full}	2	1	2	1	100	N/A
	T_{pass}	2	0	2	1	79.25	100
2,3,6	T_{full}	3	0	3	0	100	100
	T_{pass}	3	0	3	0	100	100
4	T_{full}	3	2	3	2	100	N/A
	T_{pass}	3	0	3	2	77.38	100
5	T_{full}	3	2	3	2	100	N/A
	T_{pass}	2	0	3	2	61.32	79.25

Table D.6. PBCOV results for the Sorter Linked List Remove program

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	T_{full}	4	0	6	3	69.9	82.71
1	T_{full}	5	1	6	3	84.51	N/A
	T_{pass}	4	0	6	3	69.9	82.71
2	T_{full}	5	1	6	2	88.57	N/A
	T_{pass}	4	0	6	2	73.25	82.71
3	T_{full}	5	1	5	3	88.57	N/A
	T_{pass}	4	0	5	3	73.25	89.83
4	T_{full}	4	2	5	5	81.16	N/A
	T_{pass}	4	0	5	5	67.12	89.83
5	T_{full}	4	0	5	2	77.4	89.83
	T_{pass}	4	0	5	2	77.4	89.83
6	T_{full}	5	2	5	2	100	N/A
	T_{pass}	5	2	5	2	100	N/A

Table D.7. PBCOV results for Property 1 of TCAS

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	full	8	0	10	2	85.67	91.64
1	full	8	1	8	2	96.03	N/A
	passing	7	0	8	2	86.72	94.64
	passing-assert	7	0	8	2	86.72	94.64
2,3,5-9	full	7	0	7	1	94.64	100
	passing	7	0	7	1	94.64	100
	passing-assert	7	0	7	1	94.64	100
4,41	full	8	0	8	1	95.43	100
	passing	7	0	8	1	90.31	94.64
	passing-assert	7	0	8	1	90.31	94.64
10	full	7	0	8	1	90.31	94.64
	passing	7	0	8	1	90.31	94.64
	passing-assert	7	0	8	1	90.31	94.64
11	full	7	0	9	1	86.72	90.31
	passing	7	0	9	1	86.72	90.31
	passing-assert	7	0	9	1	86.72	90.31
12-23,25-28,30,33-35,38-39	full	7	0	7	1	94.64	100
	passing	7	0	7	1	94.64	100
	passing-assert	7	0	7	1	94.64	100
24,29	full	7	0	7	0	100	100
	passing	7	0	7	0	100	100
	passing-assert	7	0	7	0	100	100
31	full	8	0	8	2	91.64	100
	passing	7	0	8	2	86.72	94.64
	passing-assert	7	0	8	2	86.72	94.64
32	full	9	0	9	1	96.03	100
	passing	7	0	9	1	86.72	90.31
	passing-assert	7	0	9	1	86.72	90.31
36	full	6	1	7	0	100	N/A
	passing	5	1	7	0	93.58	N/A
	passing-assert	5	0	7	0	86.17	86.17
37	full	10	0	10	2	93.49	100
	passing	7	0	10	2	81.08	86.72
	Passing-assert	7	0	10	2	81.08	86.72
40	full	6	0	6	0	100	100
	passing	6	0	6	0	100	100
	passing-assert	6	0	6	0	100	100

Table D.8. PBCOV results for Property 2 of TCAS

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	full	14	1	18	1	92.65	N/A
1	full	16	1	18	1	96.49	N/A
	passing	13	1	18	1	90.4	N/A
	Passing-assert	13	0	18	1	88.1	89.63
2,3,5-8,12-21,25-27,33-34,38-39	full	13	1	15	1	95.59	N/A
	passing	13	1	15	1	95.59	N/A
	Passing-assert	13	0	15	1	93.15	95.19
4,41	full	14	1	18	1	92.56	N/A
	passing	13	1	18	1	90.4	N/A
	Passing-assert	13	0	18	1	88.1	89.63
9,30	full	12	1	15	1	93.15	N/A
	passing	12	1	15	1	93.15	N/A
	Passing-assert	12	0	15	1	90.54	92.52
10	full	14	1	16	1	95.93	N/A
	passing	13	1	16	1	93.7	N/A
	Passing-assert	13	0	16	1	91.31	93.15
11,31	full	14	1	19	1	91.07	N/A
	passing	13	1	19	1	88.95	N/A
	Passing-assert	13	0	19	1	86.69	88.1
22	full	12	0	14	0	94.72	94.72
	passing	12	0	14	0	94.72	94.72
	Passing-assert	12	0	14	0	94.72	94.72
23	full	12	1	15	1	93.15	N/A
	passing	12	1	15	1	93.15	N/A
	Passing-assert	12	0	15	1	90.54	92.52
24	full	13	1	13	1	100	N/A
	passing	13	1	13	1	100	N/A
	Passing-assert	13	0	13	1	97.46	100
28,35	full	13	1	15	1	95.59	N/A
	passing	11	0	15	1	87.71	89.63
	Passing-assert	11	0	15	1	87.71	89.63
29	full	12	0	12	0	100	100
	passing	12	0	12	0	100	100
	Passing-assert	12	0	12	0	100	100
32	full	15	1	18	2	93.06	N/A
	passing	13	1	18	2	88.95	N/A
	Passing-assert	13	0	18	2	86.69	89.63
36	full	12	2	14	1	97.68	N/A
	passing	10	2	14	1	92.52	N/A
	Passing-assert	10	0	14	1	86.49	88.55
37	full	17	1	22	2	91.48	N/A

	passing	13	1	22	2	84.14	N/A
	Passing-assert	13	0	22	2	81.99	84.17
40	full	11	1	11	1	100	N/A
	passing	11	1	11	1	100	N/A
	Passing-assert	11	0	11	1	96.88	100

Table D.9. PBCOV of property 3 for TCAS

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
Original	T_{full}	14	2	15	2	98.03	N/A
1	T_{full}	15	2	16	2	98.17	N/A
	T_{pass}	14	1	16	2	94.17	N/A
	$T_{pass\text{-}assert}$	14	0	16	2	91.98	95.59
2-3,5,7-9,12-23,25-30,33-35,38-39	T_{full}	14	1	15	1	97.87	N/A
	T_{pass}	14	1	15	1	97.87	N/A
	$T_{pass\text{-}assert}$	14	0	15	1	95.59	97.68
4, 41	T_{full}	14	2	15	2	98.03	N/A
	T_{pass}	14	1	15	2	95.93	N/A
	$T_{pass\text{-}assert}$	14	0	15	2	93.7	97.68
6	T_{full}	16	1	17	1	98.17	N/A
	T_{pass}	14	1	17	1	94.17	N/A
	$T_{pass\text{-}assert}$	14	0	17	1	91.98	93.7
10	T_{full}	17	1	19	1	96.72	N/A
	T_{pass}	14	1	19	1	91.07	N/A
	$T_{pass\text{-}assert}$	14	0	19	1	88.95	90.4
11	T_{full}	17	1	20	1	95.26	N/A
	T_{pass}	14	1	20	1	89.7	N/A
	$T_{pass\text{-}assert}$	14	0	20	1	87.61	88.95
24	T_{full}	14	1	14	1	100	N/A
	T_{pass}	14	1	14	1	100	N/A
	$T_{pass\text{-}assert}$	14	0	14	1	97.68	100
31	T_{full}	17	1	19	1	96.72	N/A
	T_{pass}	14	1	19	1	91.07	N/A
	$T_{pass\text{-}assert}$	14	0	19	1	88.95	90.4
32	T_{full}	15	1	19	1	93.06	N/A
	T_{pass}	13	1	19	1	88.95	N/A
	$T_{pass\text{-}assert}$	13	0	19	1	86.69	88.1
36	T_{full}	14	1	16	2	94.17	N/A
	T_{pass}	14	0	16	2	91.98	95.59
	$T_{pass\text{-}assert}$	14	0	16	2	91.98	95.59
37	T_{full}	16	2	18	2	96.72	N/A
	T_{pass}	14	1	18	2	91.07	N/A
	$T_{pass\text{-}assert}$	14	0	18	2	88.95	91.98
40	T_{full}	14	0	14	0	100	100
	T_{pass}	14	0	14	0	100	100
	$T_{pass\text{-}assert}$	14	0	14	0	100	100

Table D.10. PBCOV results for Property 4 of TCAS

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	Full	3	2	4	2	92.08	N/A
1-5,7-9,12-30,33-35,37-39,41	Full	3	2	3	2	100	100
	passing	3	2	3	2	100	100
	passing-assert	3	0	3	2	77.38	N/A
6,31-32	Full	4	2	4	2	100	N/A
	passing	3	2	4	2	92.08	N/A
	passing-assert	3	0	4	2	71.25	86.13
10,11	Full	5	2	5	2	100	N/A
	passing	3	2	5	2	86.17	N/A
	passing-assert	3	0	5	2	66.67	77.37
36	Full	3	2	4	1	100	N/A
	passing	3	1	4	1	89.83	N/A
	passing-assert	3	0	4	1	77.38	86.13
40	Full	3	1	3	1	100	N/A
	passing	3	1	3	1	100	N/A
	passing-assert	3	0	3	1	86.14	100

Table D.11. PBCOV results for Property 5 of TCAS

Version	Tests	n_{cov}^{true}	n_{cov}^{false}	n_{feas}^{true}	n_{feas}^{false}	m_{pbcov}	m_{pbcov}^{opt}
original	full	7	1	7	2	95.43	N/A
1-8,12-21,25-27,31-34,37-39,41	full	7	1	7	2	95.43	N/A
	passing	7	1	7	2	95.43	N/A
	Passing-assert	7	0	7	2	90.31	100
9,23,30	full	6	1	7	2	90.31	N/A
	passing	6	1	7	2	90.31	N/A
	Passing-assert	6	0	7	2	84.51	93.58
10,11	full	7	2	7	2	100	N/A
	passing	7	1	7	2	95.43	N/A
	Passing-assert	7	0	7	2	90.31	100
22	full	6	0	6	1	93.58	100
	passing	6	0	6	1	93.58	100
	Passing-assert	6	0	6	1	93.58	100
24	full	7	1	7	1	100	N/A
	passing	7	1	7	1	100	N/A
	Passing-assert	7	0	7	1	94.64	100
28	full	7	1	7	2	95.43	N/A
	passing	5	0	7	2	77.82	86.17
	Passing-assert	5	0	7	2	77.82	86.17
29	full	6	0	6	0	100	100
	passing	6	0	6	0	100	100
	Passing-assert	6	0	6	0	100	100
35	full	7	1	7	2	95.43	N/A
	passing	5	0	7	2	77.82	86.17
	Passing-assert	5	0	7	2	77.82	86.17
36	full	4	2	5	1	100	N/A
	passing	4	2	5	1	100	N/A
	Passing-assert	4	0	5	1	82.71	89.83
40	full	5	1	5	1	100	N/A
	passing	5	1	5	1	100	N/A
	Passing-assert	5	0	5	1	92.08	100

APPENDIX E: ***Structural Coverage Results***

Table E.12. Structural coverage results for RBBST Insert

Version	test suite	Test-suite length	line	branch	branch once	call	block	decision	C-use	P-use
1	T_{full}	616	100	100	92	100	100	96	76	69
	T_{pass}	556	98	100	89	90	98	94	74	66
2	T_{full}	616	100	100	92	100	100	96	85	80
	T_{pass}	512	98	100	87	90	98	92	79	71
3	T_{full}	616	100	100	92	100	100	96	76	69
	T_{pass}	616	100	100	92	100	100	96	76	69
4	T_{full}	616	100	100	92	100	100	96	76	69
	T_{pass}	616	100	100	92	100	100	96	76	69
5	T_{full}	616	100	100	92	100	100	96	68	56
	T_{pass}	528	96	100	89	100	98	94	66	56
6	T_{full}	616	100	100	92	100	100	96	76	69
	T_{pass}	616	100	100	92	100	100	96	76	69
7	T_{full}	616	100	100	92	100	100	96	76	69
	T_{pass}	616	100	100	92	100	100	96	76	69
8	T_{full}	616	100	100	92	100	100	96	76	69
	T_{pass}	540	98	100	91	100	98	94	72	62
9	T_{full}	616	100	100	92	100	100	96	85	80
	T_{pass}	512	96	100	87	90	97	92	78	71
10	T_{full}	616	90	100	75	92	91	75	56	35
	T_{pass}	448	90	100	75	92	91	75	56	35
11	T_{full}	616	96	100	86	92	93	86	59	44
	T_{pass}	448	96	100	86	92	93	86	59	44
12	T_{full}	616	100	100	92	100	100	97	77	70
	T_{pass}	460	100	100	92	100	100	97	77	70

Table E.13. RBBST Remove structural coverage results

Version	test suite	Test-suite length	line	branch	branch once	call	block	decision	C-use	P-use
1	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
2	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	582	**89**	**93**	**80**	**93**	**89**	**85**	**63**	**51**
3	T_{full}	616	**92**	**93**	**82**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	556	**92**	**93**	**82**	**100**	**91**	**88**	**67**	**53**
4	T_{full}	616	**46**	**68**	**45**	**36**	**63**	**60**	**37**	**40**
	T_{pass}	168	**27**	**40**	**27**	**22**	**37**	**35**	**21**	**23**
5	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
6	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	554	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
7	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**52**
	T_{pass}	612	**91**	**93**	**82**	**95**	**90**	**88**	**65**	**52**
8	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	592	**90**	**93**	**82**	**95**	**90**	**87**	**64**	**51**
9	T_{full}	616	**92**	**93**	**84**	**100**	**91**	**89**	**67**	**53**
	T_{pass}	582	**88**	**93**	**80**	**93**	**89**	**85**	**63**	**51**

Table E.14. Memory manager structural coverage results

Version	test suite	Test-suite length	Line	branch	branch once	call	block	decision	C-use	P-use
1	T_{full}	1124	100	100	100	100	100	100	100	88
	T_{pass}	337	100	100	100	100	100	100	100	88
2	T_{full}	1124	100	100	100	100	100	100	100	88
	T_{pass}	933	94	100	92	100	97	93	94	81
3	T_{full}	1124	100	100	100	100	100	100	100	88
	T_{pass}	388	100	100	100	100	100	100	100	88
4	T_{full}	1124	100	100	100	100	100	100	100	88
	T_{pass}	48	100	100	92	100	97	86	94	69
5	T_{full}	1124	100	100	100	100	100	100	100	88
	T_{pass}	32	71	100	83	75	79	79	75	63
6	T_{full}	1124	100	100	100	100	100	100	100	88
	T_{pass}	32	74	100	83	75	79	79	75	63

Table E.**15. SLLInsert structural coverage results**

Version	test suite	Test-suite length	Line	branch	branch once	call	block	decision	C-use	P-use
1	T_{full}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
	T_{pass}	73	**47**	**50**	**38**	**N.A.**	**45**	**38**	**36**	**50**
2	T_{full}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
	T_{pass}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
3	T_{full}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
	T_{pass}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
4	T_{full}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
	T_{pass}	754	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
5	T_{full}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
	T_{pass}	73	**50**	**50**	**38**	**N.A.**	**45**	**38**	**33**	**50**
6	T_{full}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**
	T_{pass}	1751	**100**	**100**	**100**	**N.A.**	**100**	**100**	**100**	**100**

Table E.16. SLL Remove structural coverage results

Version	test suite	Test-suite length	Line	branch	branch once	call	block	decision	C-use	P-use
1	T_{full}	1751	**100**	**100**	**100**		**100**	**100**	**100**	**100**
	T_{pass}	73	**81**	**87**	**75**		**84**	**81**	**84**	**85**
2	T_{full}	1751	**90**	**100**	**87**		**95**	**93**	**97**	**95**
	T_{pass}	1319	**90**	**100**	**87**		**95**	**93**	**97**	**92**
3	T_{full}	1751	**90**	**100**	**87**		**95**	**93**	**91**	**90**
	T_{pass}	647	**90**	**100**	**87**		**95**	**93**	**91**	**90**
4	T_{full}	1751	**90**	**100**	**87**		**95**	**93**	**97**	**90**
	T_{pass}	982	**90**	**100**	**87**		**95**	**93**	**97**	**90**
5	T_{full}	1751	**100**	**100**	**100**		**100**	**100**	**100**	**100**
	T_{pass}	151	**95**	**100**	**93**		**97**	**96**	**98**	**95**
6	T_{full}	1751	**90**	**100**	**87**		**95**	**93**	**96**	**90**
	T_{pass}	1751	**90**	**100**	**87**		**95**	**93**	**96**	**90**

Table E.17. GZIP structural coverage results

Version	suite	length	Line	branch	branch once	call	block	decision	C-use	P-use
1	T_{full}	214	74.63	75.27	58.53	49.3	58	50	56	42
	T_{pass}	213	74.4	74.87	58.2	49.1	58	49	56	42
2	T_{full}	214	74.69	75.81	59.01	50.1	58	50	56	42
	T_{pass}	198	74.69	75.81	59.01	50.1	58	50	56	42
3	T_{full}	214	72.99	73.79	57.66	46.91	58	50	56	42
	T_{pass}	197	69.24	70.03	54.91	40.12	53	46	53	40
4	T_{full}	214	74.52	75.67	58.87	49.7	58	50	56	42
	T_{pass}	211	73.64	74.87	58.2	47.7	57	49	55	42
5	T_{full}	214	74.69	75.81	59.01	50.1	58	50	56	42
	T_{pass}	213	71.29	72.58	56.32	49.3	56	47	53	40
6	T_{full}	214	74.52	74.33	57.86	50.7	58	50	56	42
	T_{pass}	16	52.2	51.61	37.63	40.52	40	33	35	27
7	T_{full}	214	71.68	71.01	54.97	48.21	58	50	56	42
	T_{pass}	201	70.16	69.53	53.56	46.61	53	45	52	39

Table E.18. TCAS structural coverage results without $T_{passing\text{-}assert}$

		GCOV				ATAC			
version	**test-suite**	**Line**	**branch**	**branch once**	**call**	**block**	**decision**	**C-use**	**P-use**
1-3,6-10,13-14,16-25,28,33,35,37-39	Full	98.46	100	92.42	100	99	90	98	91
	passing	98.46	100	92.42	100	99	90	98	91
4	Full	98.46	100	92.42	100	99	90	98	91
	passing	98.46	100	90.91	100	99	90	98	91
5,15,27	Full	98.46	100	95.31	100	99	94	98	97
	passing	98.46	100	95.31	100	99	94	98	97
11	Full	100	100	93.55	100	100	91	100	93
	passing	100	100	93.55	100	100	91	100	93
12,34	Full	98.46	100	95.45	100	99	94	98	97
	passing	98.46	100	95.45	100	99	94	98	97
26,29,30,41	Full	98.46	100	92.19	100	99	90	98	91
	passing	98.46	100	92.19	100	99	90	98	91
31,32	Full	98.51	100	91.43	100	99	90	98	92
	passing	98.51	100	91.43	100	99	90	98	92
36	Full	98.46	100	92.42	100	99	90	98	91
	passing	96.92	100	86.36	100	98	88	95	88
40	Full	98.46	100	93.33	100	99	91	98	91
	passing	96.92	100	86.67	100	98	89	95	88

Table E.19. TCAS structural coverage results for $T_{pass\text{-}assert}$ *Property 1*

	GCOV				ATAC			
Version	**line**	**branch**	**branch once**	**call**	**block**	**Decision**	**C-use**	**P-use**
1-3,6-10,13-14,16-25,28,33,35,37-39	98.46	100	92.42	100	99	90	98	91
4	98.46	100	90.91	100	99	90	98	91
5	98.46	100	95.31	100	99	94	98	97
11	100	100	93.55	100	100	91	100	93
12,34	98.46	100	95.45	100	99	94	98	97
15,27	98.46	100	95.31	100	99	94	98	97
26,29,30,41	98.46	100	92.19	100	99	90	98	91
31,32	98.51	100	91.43	100	99	90	98	92
36,40	96.92	100	86.36	100	98	88	95	88

37-39	98.46	100	92.42	100	99	90	98	91

Table E.20. TCAS structural coverage results for $T_{pass\text{-}assert}$ *Property 2*

	GCOV				ATAC			
version	line	branch	branch once	call	block	decision	C-use	P-use
1-3,6-10,13,14,16-25,28,33,35,37-39	98.46	100	92.42	100	99	90	98	91
4	98.46	100	90.91	100	99	90	98	91
5,15,27	98.46	100	95.31	100	99	94	98	97
11	100	100	93.55	100	100	91	100	93
12,34	98.46	100	95.45	100	99	94	98	97
26	98.46	100	92.19	100	99	90	98	91
29,30,41	98.46	100	92.19	100	99	90	98	91
31,32	98.51	100	91.43	100	99	90	98	92
36	96.92	100	86.36	100	98	88	95	88
40	96.92	100	86.67	100	98	89	95	88

Table E.21. TCAS structural coverage results for $T_{pass\text{-}assert}$ *Property 3*

	GCOV				ATAC			
version	line	branch	branch once	call	block	decision	C-use	P-use
1-4,6-10,13,14,16-21,23-25,28,33,35-39	96.92	100	86.36	100	98	88	95	88
5,15,27	96.92	100	89.06	100	98	92	95	94
11	98.41	100	87.1	100	99	89	98	90
12,34	96.92	100	89.39	100	98	92	95	94
22	96.92	100	87.88	100	98	88	95	88
26,29,30,41	96.92	100	85.94	100	98	88	95	88
31	97.01	97.14	82.86	100	97	87	92	87
32	97.01	100	85.71	100	98	88	96	89
40	96.92	100	86.67	100	98	89	95	88

Table E.22. TCAS structural coverage results for $T_{pass\text{-}assert}$ *Property 4*

property 4, T_{passing-Assert}	GCOV				ATAC			
Version	line	branch	branch once	call	block	decision	C-use	P-use
1-4,6-10,13,14,16-21,24,25,28,33,35-39	95.38	96.97	77.27	100	96	82	93	79
5,15,27	95.38	96.88	79.69	100	96	85	93	84
9,22,23	95.38	96.97	78.79	100	96	82	93	79
11	96.83	100	80.65	100	98	87	95	87
12,34	95.38	96.97	80.3	100	96	86	93	85
26,29,30,41	95.38	96.88	76.56	100	96	81	93	78
31	95.52	94.29	74.29	100	95	81	90	79
32	95.52	97.14	77.14	100	96	83	94	82
40	95.38	96.67	78.33	100	96	83	93	79

Table E.23. TCAS structural coverage results for $T_{pass\text{-}assert}$ *Property 5*

	GCOV				ATAC			
Version	line	branch	branch once	call	block	decision	C-use	P-use
1-3,6-10,13,14,16-25,28,33,35,37-39	98.46	100	92.42	100	99	90	98	91
4	98.46	100	90.91	100	99	90	98	91
5,15,27	98.46	100	95.31	100	99	94	98	97
11	100	100	93.55	100	100	91	100	93
12,34	98.46	100	95.45	100	99	94	98	97
26,29,30,41	98.46	100	92.19	100	99	90	98	91
31,32	98.51	100	91.43	100	99	90	98	92
36	96.92	100	86.36	100	98	88	95	88
40	96.92	100	86.67	100	98	89	95	88