Tool Operational Requirements

Release 1.1

AdaCore for Adiru

June 19, 2013

This page is intentionally left blank.

CONTENTS

1 Operational Environment **3**

2 Source coverage TORs and Testcases for Ada **5**
2.1 Statement Coverage (SC) TORs and Testcases for Ada 5
2.1.1 Core expectations for Statement Coverage (SC) assessments. 5
2.1.2 SC expectations regarding the combination of multiple 16
2.1.3 SC expectations focused on statement sequences and control-flow transfers. 17
2.1.4 SC expectations regarding exemption regions. 17
2.1.5 SC expectations for mixes of statement constructs, representative of real 18
2.1.6 SC expectations regarding potentially confusing constructs, e.g. multiple 18
2.2 Decision Coverage (DC) TORs and Testcases for Ada 19
2.2.1 Core expectations for Decision Coverage 19
2.2.2 DC expectations regarding the combination of multiple 20
2.2.3 DC expectations regarding exemption regions. 20
2.2.4 DC expectations for mixes of statement and decision constructs representative 21
2.2.5 DC expectations with respect to potentially confusing constructs, ... 21
2.3 Modified Condition/Decision Coverage (MCDC) TORs and Testcases for Ada 22
2.3.1 Core expectations for MCDC assessments. 22
2.3.2 MCDC expectations regarding the combination of multiple 23
2.3.3 MCDC expectations regarding exemption regions. 24
2.3.4 MCDC expectations for mixes of various statement and decision constructs 24
2.3.5 MCDC expectations with respect to potentially confusing constructs, ... 24

3 Language agnostic TORs and Testcases **27**
3.1 Describe the tool output report format and check compliance to it 27
3.1.1 Requirement(s) 27
3.1.2 Testing Strategy 28

4 Appendix **29**

5 Test Cases **31**
5.1 Check the correctness of DC assessments on decisions that are not entirely evaluated (don't yield 31
5.1.1 DC testcases for a decision of the form `A and then B` 31
5.1.2 DC testcases for a decision of the form `not A`, negated simple expression. 31
5.1.3 DC testcases for a decision of the form `A or else B` 31
5.1.4 DC testcases for a decision of the form `A`, simple expression without 32
5.2 Check the correctness of DC assessments on decisions featuring various possible forms 32
5.2.1 Check DC on expressions with one or two operands involving Ada attributes 32
5.2.2 Check DC on expressions with one or two function call conditions. 32
5.2.3 Check DC on expressions with conditions of a Boolean subtype or derived type. 32
5.2.4 Check DC on expressions with conditions involving explicit tests, ... 32
5.3 Check the correctness of DC assessments on expressions of various topologies everywhere they . 32
5.3.1 Check DC on a decision of the form `A and then B` 34

5.3.2 Check DC on a decision of the form `not X` . . . 34
5.3.3 Check DC on a decision of the form `A or else B` . . . 34
5.3.4 Check DC on a decision of the simplest . . . 34
5.4 Exercise consolidation over `A and then B`. . . . 34
5.4.1 ada sources . . . 34
5.4.2 c sources . . . 34
5.4.3 h sources . . . 34
5.5 Exercise consolidation over `A or else B`. . . . 34
5.5.1 ada sources . . . 34
5.5.2 c sources . . . 35
5.5.3 h sources . . . 35
5.6 Exercise the use of an exemption region to exempt a functional precondition . . . 35
5.6.1 ada sources . . . 35
5.6.2 c sources . . . 35
5.6.3 h sources . . . 35
5.7 Exercise an If/Elsif construct involving complex decisions with computations . . . 35
5.7.1 ada sources . . . 35
5.7.2 c sources . . . 36
5.7.3 h sources . . . 36
5.8 Exercise a toplevel If statement with an else sequence embedding a For loop, . . . 36
5.8.1 ada sources . . . 36
5.8.2 c sources . . . 37
5.8.3 h sources . . . 37
5.9 Exercise a simple If statement within a While loop. . . . 37
5.9.1 ada sources . . . 37
5.9.2 c sources . . . 37
5.9.3 h sources . . . 37
5.10 Exercise a three way If statement within a While loop. . . . 37
5.10.1 ada sources . . . 37
5.10.2 c sources . . . 38
5.10.3 h sources . . . 38
5.11 Exercise a simple If statement nested in one of the possible selections of a . . . 38
5.11.1 ada sources . . . 38
5.11.2 c sources . . . 39
5.11.3 h sources . . . 39
5.12 Check that no stmt coverage violations are reported for explicitly deactivated . . . 39
5.12.1 ada sources . . . 39
5.12.2 c sources . . . 39
5.12.3 h sources . . . 39
5.13 Check that Boolean expressions spanning multiple lines are handled correctly. . . . 39
5.14 Check that the tool properly distinguishes independant simple nested . . . 40
5.14.1 ada sources . . . 40
5.14.2 c sources . . . 40
5.14.3 h sources . . . 40
5.15 Check the correctness of MCDC assessments on decisions that are not entirely evaluated (don't . 40
5.15.1 Check MCDC on `A and then Call (B or else C)`, where the nested decision may be . . . 40
5.16 Check the correctness of MCDC assessments on decisions featuring all the possible forms . . . 40
5.16.1 Check MCDC on a simple expression with two operands involving comparison with . . 41
5.16.2 Check MCDC on simple expressions with one or two operands involving Ada attributes . 41
5.16.3 Check MCDC on simple decisions over two operands featuring record component . . . 41
5.16.4 Check MCDC on simple expressions with one or two operands involving Ada string . . 41
5.16.5 TITLE . . . 42
5.16.6 Check MCDC on simple expressions with one or two operands involving Ada string . . 42
5.16.7 Check MCDC on a simple expression with two operands involving a range test. 42
5.16.8 Check MCDC on a simple decision over two operands featuring array slices 42
5.16.9 Check MCDC on a simple decision where conditions are of a Boolean subtype or 42
5.17 Check the correctness of MCDC assessments on expressions of arbitrary topology . . . 42

5.17.1 Check MCDC on a decision of the form . . . 46
5.17.2 Check MCDC on a decision of the form . . . 46
5.17.3 Check MCDC on a decision of the form . . . 46
5.17.4 Check MCDC on a decision of the form . . . 46
5.17.5 Check MCDC on a decision of the form . . . 46
5.17.6 Check MCDC on a decision of the form . . . 46
5.17.7 Check MCDC on a decision of the form . . . 46
5.17.8 Check MCDC on a decision of the form . . . 46
5.17.9 Check MCDC on a decision of the form . . . 46
5.17.10 Check proper behavior in . . . 46
5.17.11 Check MCDC on decisions with diamonds in . . . 46
5.17.12 Check MCDC on a decision of the form . . . 47
5.17.13 Check MCDC on a decision of the form . . . 47
5.17.14 Check MCDC on a decision of the form . . . 47
5.17.15 Check MCDC on a decision of the form . . . 47
5.17.16 Check MCDC on a decision of the form . . . 47
5.17.17 Check MCDC on a decision of the form . . . 47
5.17.18 Check MCDC on a decision of the form . . . 47
5.17.19 Check MCDC on a decision of the form . . . 47
5.17.20 Check MCDC on a decision of the form . . . 47
5.17.21 Check MCDC on a decision of the form . . . 47
5.17.22 Check MCDC on a decision of the form . . . 47
5.18 Exercise consolidation over `A and then B`. . . . 47
5.18.1 ada sources . . . 48
5.18.2 c sources . . . 48
5.18.3 h sources . . . 48
5.19 Exercise consolidation over `Andthen (X => A and then B, Y => C or else D),`. . . . 48
5.20 Exercise consolidation over `Andthen (X => A and then B, Y => C or else D),`. . . . 48
5.20.1 ada sources . . . 48
5.20.2 c sources . . . 48
5.20.3 h sources . . . 48
5.21 Exercise consolidation over `A or else B`. . . . 48
5.21.1 ada sources . . . 49
5.21.2 c sources . . . 49
5.21.3 h sources . . . 49
5.22 Exercise consolidation over `(A or else B) and then C`. . . . 49
5.22.1 ada sources . . . 49
5.22.2 c sources . . . 49
5.22.3 h sources . . . 49
5.23 Exercise the use of an exemption region to exempt a functional precondition . . . 49
5.23.1 ada sources . . . 50
5.23.2 c sources . . . 50
5.23.3 h sources . . . 50
5.24 Exercise an If/Elsif construct involving complex decisions with computations . . . 50
5.24.1 ada sources . . . 50
5.24.2 c sources . . . 51
5.24.3 h sources . . . 51
5.25 Exercise a toplevel If statement with an else sequence embedding a For loop, . . . 51
5.25.1 ada sources . . . 51
5.25.2 c sources . . . 52
5.25.3 h sources . . . 52
5.26 Exercise a three way If statement within a While loop. . . . 52
5.26.1 ada sources . . . 52
5.26.2 c sources . . . 53
5.26.3 h sources . . . 53
5.27 Exercise a simple If statement nested in one of the possible selections of a . . . 53
5.27.1 ada sources . . . 53
5.27.2 c sources . . . 54

5.27.3 h sources . . . 54
5.28 Check that no stmt coverage violations are reported for explicitly deactivated . . . 54
5.28.1 ada sources . . . 54
5.28.2 c sources . . . 54
5.28.3 h sources . . . 54
5.29 Check that when multiple decisions occur on the same source line, individual . . . 54
5.30 Check that decisions spanning multiple lines are recognized and processed . . . 54
5.31 Check that recursive decision evaluations are processed correctly. . . . 54
5.32 Check SC of active pragmas . . . 54
5.32.1 ada sources . . . 55
5.32.2 c sources . . . 55
5.32.3 h sources . . . 55
5.33 Check SC of inactive pragmas . . . 55
5.33.1 ada sources . . . 55
5.33.2 c sources . . . 55
5.33.3 h sources . . . 55
5.34 Check SC of Discrete Subtype Definitions . . . 55
5.34.1 ada sources . . . 55
5.34.2 c sources . . . 56
5.34.3 h sources . . . 56
5.35 Check SC of Object Declarations involving heap or stack dynamic allocation . . . 56
5.35.1 ada sources . . . 56
5.35.2 c sources . . . 56
5.35.3 h sources . . . 56
5.36 Check SC of Object Declarations involving static or fixed-sized stack . . . 56
5.36.1 ada sources . . . 57
5.36.2 c sources . . . 57
5.36.3 h sources . . . 57
5.37 Check SC of Subtype Indications . . . 57
5.37.1 ada sources . . . 57
5.37.2 c sources . . . 58
5.37.3 h sources . . . 58
5.38 Check SC related to Variant Parts in record type declarations . . . 58
5.38.1 ada sources . . . 58
5.38.2 c sources . . . 58
5.38.3 h sources . . . 58
5.39 Check SC of Assignment statements (ARM 5.1) . . . 58
5.39.1 ada sources . . . 58
5.39.2 c sources . . . 59
5.39.3 h sources . . . 59
5.40 Check SC of Null statements (ARM 5.2) . . . 59
5.40.1 ada sources . . . 59
5.40.2 c sources . . . 59
5.40.3 h sources . . . 59
5.41 Check SC of loop Exit statements (ARM 5.7) . . . 59
5.41.1 ada sources . . . 59
5.41.2 c sources . . . 60
5.41.3 h sources . . . 60
5.42 Check SC with Goto statements (ARM 5.8) . . . 60
5.42.1 ada sources . . . 60
5.42.2 c sources . . . 61
5.42.3 h sources . . . 61
5.43 Check SC with IF statements taking [implicit] ELSE branches only. . . . 61
5.43.1 ada sources . . . 61
5.43.2 c sources . . . 61
5.43.3 h sources . . . 61
5.44 Check SC with IF statements taking ELSIF branches only. . . . 61
5.44.1 ada sources . . . 61

5.44.2 c sources . . . 62
5.44.3 h sources . . . 62
5.45 Check SC with IF statements taking IF branches only. . . . 62
5.45.1 ada sources . . . 62
5.45.2 c sources . . . 62
5.45.3 h sources . . . 62
5.46 Check SC with IF statements taking all their branches. . . . 62
5.46.1 ada sources . . . 62
5.46.2 c sources . . . 62
5.46.3 h sources . . . 62
5.47 Check SC with IF statements not executed at all. . . . 62
5.47.1 ada sources . . . 62
5.47.2 c sources . . . 62
5.47.3 h sources . . . 62
5.48 Check SC with CASE statements executed several times, selecting . . . 62
5.48.1 ada sources . . . 63
5.48.2 c sources . . . 63
5.48.3 h sources . . . 63
5.49 Check SC with CASE statements not executed at all . . . 63
5.49.1 ada sources . . . 63
5.49.2 c sources . . . 63
5.49.3 h sources . . . 63
5.50 Check SC with CASE statements executed only once, selecting a single . . . 63
5.50.1 ada sources . . . 63
5.50.2 c sources . . . 63
5.50.3 h sources . . . 63
5.51 Check SC with LOOP statements that execute at least one full . . . 63
5.51.1 ada sources . . . 63
5.51.2 c sources . . . 64
5.51.3 h sources . . . 64
5.52 Check SC with LOOP statements that execute only a partial iteration . . . 64
5.52.1 ada sources . . . 64
5.52.2 c sources . . . 64
5.52.3 h sources . . . 64
5.53 Check SC with LOOP statements not executed at all . . . 64
5.53.1 ada sources . . . 64
5.53.2 c sources . . . 64
5.53.3 h sources . . . 64
5.54 Check SC with LOOP statements that don’t even start an iteration over . . . 64
5.54.1 ada sources . . . 64
5.54.2 c sources . . . 65
5.54.3 h sources . . . 65
5.55 Check SC with nested block statements . . . 65
5.55.1 ada sources . . . 65
5.55.2 c sources . . . 65
5.55.3 h sources . . . 65
5.56 Check SC with block statements with/without declarative part or exception . . . 65
5.56.1 ada sources . . . 65
5.56.2 c sources . . . 66
5.56.3 h sources . . . 66
5.57 Check SC on a simple subprogram called from a wide range of source contexts . . . 66
5.57.1 Check SC on a simple subprogram called from a subprogram actual parameter . . . 66
5.57.2 Check SC on a simple subprogram called from a record default component initialization 66
5.57.3 Check SC on a simple subprogram called from a local object initialization . . . 66
5.57.4 Check SC on a simple subprogram called as part of a subprogram formal default . . . 66
5.57.5 Check SC on a simple subprogram called from a local package elaboration body . . . 66
5.57.6 Check SC on a simple subprogram called from a procedure body . . . 66
5.58 Check SC within a panel of library level subprograms reached through . . . 67

5.58.1 ada sources . . . 67
5.58.2 c sources . . . 67
5.58.3 h sources . . . 67
5.59 Check SC within a panel of nested subprograms reached through a variety of . . . 67
5.59.1 ada sources . . . 68
5.59.2 c sources . . . 68
5.59.3 h sources . . . 68
5.60 Check SC with simple subprogram overridings . . . 68
5.60.1 ada sources . . . 68
5.60.2 c sources . . . 69
5.60.3 h sources . . . 69
5.61 Check SC with complex overridings for tagged types . . . 69
5.61.1 ada sources . . . 69
5.61.2 c sources . . . 69
5.61.3 h sources . . . 69
5.62 Check SC with multi-level Inlining, where subprogram S3 is inlined in S2, in . . . 69
5.62.1 ada sources . . . 69
5.62.2 c sources . . . 70
5.62.3 h sources . . . 70
5.63 Check SC with a subprogram inlined in two other ones, each exercising . . . 70
5.63.1 ada sources . . . 70
5.63.2 c sources . . . 70
5.63.3 h sources . . . 70
5.64 Check SC with return statements in Function . . . 70
5.64.1 ada sources . . . 70
5.64.2 c sources . . . 70
5.64.3 h sources . . . 70
5.65 Check SC with return statements in Procedures . . . 70
5.65.1 ada sources . . . 71
5.65.2 c sources . . . 71
5.65.3 h sources . . . 71
5.66 Check the correctness of SC assessments for subprograms declared in various package regions . . . 71
5.66.1 ada sources . . . 71
5.66.2 c sources . . . 71
5.66.3 h sources . . . 71
5.67 Check SC on package elaboration statements . . . 71
5.67.1 ada sources . . . 72
5.67.2 c sources . . . 72
5.67.3 h sources . . . 72
5.68 Check that object renamings are handled correctly . . . 72
5.68.1 ada sources . . . 72
5.68.2 c sources . . . 72
5.68.3 h sources . . . 72
5.69 Check that package or subprogram renamings are handled correctly . . . 72
5.69.1 ada sources . . . 73
5.69.2 c sources . . . 73
5.69.3 h sources . . . 73
5.70 Exercise a mix of subprogram and package subunits, with multiple levels . . . 73
5.70.1 ada sources . . . 73
5.70.2 c sources . . . 73
5.70.3 h sources . . . 73
5.71 Exercise a package subunits declared in various source contexts. . . . 73
5.71.1 ada sources . . . 74
5.71.2 c sources . . . 74
5.71.3 h sources . . . 74
5.72 Exercise subprogram subunits declared in various source contexts. . . . 74
5.72.1 ada sources . . . 74
5.72.2 c sources . . . 74

5.72.3 h sources . . . 74
5.73 Check SC with expressions that can fail to evaluate because of exceptions . . . 74
5.73.1 TITLE . . . 75
5.73.2 TITLE . . . 75
5.73.3 TITLE . . . 75
5.73.4 TITLE . . . 75
5.74 Check SC correct recognition of alternate exception handlers for user . . . 75
5.74.1 ada sources . . . 76
5.74.2 c sources . . . 76
5.74.3 h sources . . . 76
5.75 Check SC of sequences potentially skipped by exceptions that propagate through . . . 76
5.75.1 ada sources . . . 76
5.75.2 c sources . . . 77
5.75.3 h sources . . . 77
5.76 Check SC with multiple kinds of raise operations, implicit or explicit, . . . 77
5.76.1 ada sources . . . 77
5.76.2 c sources . . . 77
5.76.3 h sources . . . 77
5.77 Check that in case of a library-level instantiation the code of the generic . . . 77
5.77.1 ada sources . . . 77
5.77.2 c sources . . . 78
5.77.3 h sources . . . 78
5.78 Check that in case of a local instantiation the code of the generic unit is . . . 78
5.78.1 ada sources . . . 78
5.78.2 c sources . . . 78
5.78.3 h sources . . . 78
5.79 Exercise a three way if statement with three different . . . 78
5.79.1 ada sources . . . 78
5.79.2 c sources . . . 79
5.79.3 h sources . . . 79
5.80 Check that basic blocks of statements are recognized in various possible . . . 79
5.80.1 Check that statements are recognized in package elaboration bodies and . . . 79
5.80.2 Check that statements are recognized in subprogram bodies and nested blocks . . . 79
5.81 Check that various forms of control-flow transfers are handled properly. . . . 79
5.81.1 Exercise a function where flow control is achieved with goto . . . 79
5.81.2 Exercise a function where flow control is achieved with . . . 79
5.81.3 Exercise a function where flow control is achieved with loop . . . 80
5.81.4 Exercise a function where flow control is achieved with goto . . . 80
5.82 Check processing of exemptions applying to a mix of statements and . . . 80
5.82.1 ada sources . . . 80
5.82.2 c sources . . . 80
5.82.3 h sources . . . 80
5.83 Check processing of exemptions applying to groups of statements within the . . . 80
5.83.1 ada sources . . . 80
5.83.2 c sources . . . 81
5.83.3 h sources . . . 81
5.84 Check processing of exemptions applying to groups of statements within a . . . 81
5.84.1 ada sources . . . 81
5.84.2 c sources . . . 81
5.84.3 h sources . . . 81
5.85 Check processing of exemptions applying to groups of statements within local . . . 81
5.85.1 ada sources . . . 82
5.85.2 c sources . . . 82
5.85.3 h sources . . . 82
5.86 Check processing of exemptions applying to a mix of statements and . . . 82
5.86.1 ada sources . . . 82
5.86.2 c sources . . . 83
5.86.3 h sources . . . 83

5.87 Check processing of exemptions applying to groups of entire subprograms. . . . 83
5.87.1 ada sources . . . 83
5.87.2 c sources . . . 83
5.87.3 h sources . . . 83
5.88 Check processing of exemptions applying to whole subprogram bodies. . . . 83
5.88.1 ada sources . . . 83
5.88.2 c sources . . . 84
5.88.3 h sources . . . 84
5.89 Exercise a toplevel If statement with an else sequence embedding a For loop, . . . 84
5.89.1 ada sources . . . 84
5.89.2 c sources . . . 84
5.89.3 h sources . . . 84
5.90 Exercise a simple If statement within a While loop. . . . 84
5.90.1 ada sources . . . 85
5.90.2 c sources . . . 85
5.90.3 h sources . . . 85
5.91 Exercise a three way If statement within a While loop. . . . 85
5.91.1 ada sources . . . 85
5.91.2 c sources . . . 86
5.91.3 h sources . . . 86
5.92 Exercise a simple If statement nested in one of the possible selections of a . . . 86
5.92.1 ada sources . . . 86
5.92.2 c sources . . . 86
5.92.3 h sources . . . 86
5.93 Check that coverage is not affected by the presence of subprograms oveloading. . . . 86
5.93.1 ada sources . . . 86
5.93.2 c sources . . . 87
5.93.3 h sources . . . 87
5.94 Check that SC assessments remain correct despite misleading identations. . . . 87
5.94.1 ada sources . . . 87
5.94.2 c sources . . . 87
5.94.3 h sources . . . 87
5.95 Check that no stmt coverage violations are reported for explicitly deactivated . . . 87
5.95.1 ada sources . . . 87
5.95.2 c sources . . . 88
5.95.3 h sources . . . 88
5.96 Check that all/none of the straightline statements in a big subprogram are . . . 88
5.96.1 ada sources . . . 88
5.96.2 c sources . . . 88
5.96.3 h sources . . . 88
5.97 Check that statements spanning multiple lines are handled properly. . . . 88
5.97.1 ada sources . . . 88
5.97.2 c sources . . . 89
5.97.3 h sources . . . 89
5.98 Check that multiple statements located on the same source line are . . . 89
5.98.1 ada sources . . . 89
5.98.2 c sources . . . 89
5.98.3 h sources . . . 89
5.99 Check that sequences of multiple nop statements (e.g. null or pragma) are . . . 89
5.99.1 ada sources . . . 89
5.99.2 c sources . . . 90
5.99.3 h sources . . . 90
5.100 Check that the tool isn't fooled into thinking that a statement is not covered . . . 90
5.100.1 ada sources . . . 90
5.100.2 c sources . . . 90
5.100.3 h sources . . . 90
5.101 Check output report compliance to expectations on a case involving a single . . . 90
5.101.1 ada sources . . . 90

5.101.2 c sources . . . 90
5.101.3 h sources . . . 90
5.102 Check output report compliance to expectations on a case where no violation or . . . 90
5.102.1 ada sources . . . 91
5.102.2 c sources . . . 91
5.102.3 h sources . . . 91
5.103 Check output report compliance to expectations on a case with . . . 91
5.103.1 ada sources . . . 91
5.103.2 c sources . . . 91
5.103.3 h sources . . . 91
5.104 Check output report compliance to expectations on a case involving a multiple . . . 91
5.104.1 ada sources . . . 91
5.104.2 c sources . . . 92
5.104.3 h sources . . . 92
5.105 Check output report compliance to expectations on a case involving a single . . . 92
5.105.1 ada sources . . . 92
5.105.2 c sources . . . 92
5.105.3 h sources . . . 92

Index **93**

This page is intentionally left blank.

This is the root of the “Tool Operational Requirements” and “Testcases” hierarchy that constitutes part of the GNATcoverage qualification material.

This is decomposed in several toplevel chapters, summarized in the table below. The language specific chapters that apply depend on your qualification objectives.

(*)	Chapter	Description
intro	**OpEnviron**	*Operational Environment*
rqg	**Ada**	*Source coverage TORs and Testcases for Ada*
rqg	. . decision	*Decision Coverage (DC) TORs and Testcases for Ada*
rqg	. . mcdc	*Modified Condition/Decision Coverage (MCDC) TORs and Testcases for Ada*
rqg	. . stmt	*Statement Coverage (SC) TORs and Testcases for Ada*
rqg	**Common**	*Language agnostic TORs and Testcases*
rq	. . Report	*Describe the tool output report format and check compliance to it*
app	**Appendix**	*Appendix*
app	. . Harness	`/TOR_Doc/Appendix/Harness`
app	. . Reqindex	`/TOR_Doc/Appendix/Reqindex`

This page is intentionally left blank.

CHAPTER

ONE

OPERATIONAL ENVIRONMENT

The correctness of source coverage assessments performed by the tool depends on a few external rules that shall be obeyed. First comes a set of general rules that always apply:

Rule #	
1 (Compiler version)	The tool may only be used with a GNAT/GCC compiler version identified as suitable by the tool provider.
2 (Base compilation flags)	All the applicative code shall be compiled with the `-g -fpreserve-control-flow` command-line options, together with `-gnateS` for Ada sources.
3 (Optimization flags)	Up to GNAT Pro 6.4.2, -O0 is the only supported level. Later releases will support -O1, with or without inlining. -O2 or individual optimization flags are not supported. For Ada, suppression of run-time checks with `-gnatp` is allowed, however not mandatory.
4 (Coding standard)	For criteria involving decisions or conditions in the DO-178B sense, binary Boolean operators shall be restricted to those with short-circuit semantics. These are `and then` and `or else` in Ada, with the rule enforced by the `No_Direct_Boolean_Operator` Restriction pragma in the GNAT Pro series of compilers.

Extra options are allowed when they are known not to influence code generation, as, for example, warning control options.

In any case, the tool behavior correctness for a particular combination of versions and command-line options shall be verified by a complete testsuite run configured for the target qualification level, producing a *Software Test Results* report clear of any test failure.

This page is intentionally left blank.

CHAPTER

TWO

SOURCE COVERAGE TORS AND TESTCASES FOR ADA

Source coverage TORs and Testcases for Ada

This is split in several parts, one per target coverage criterion of relevance:

	Requirement Group	Description
rqg	stmt	*Statement Coverage (SC) TORs and Testcases for Ada*
rqg	decision	*Decision Coverage (DC) TORs and Testcases for Ada*
rqg	mcdc	*Modified Condition/Decision Coverage (MCDC) TORs and Testcases for Ada*

2.1 Statement Coverage (SC) TORs and Testcases for Ada

Statement Coverage (SC) TORs and Testcases for Ada

This is split in several parts:

	Requirement Group	Description
rqg	Core	*Core expectations for Statement Coverage (SC) assessments.*
rq	Consolidation	*SC expectations regarding the combination of multiple*
rq	ControlFlow	*SC expectations focused on statement sequences and control-flow transfers.*
rq	Exemptions	*SC expectations regarding exemption regions.*
rq	MixedConstructs	*SC expectations for mixes of statement constructs, representative of real*
rq	Robustness	*SC expectations regarding potentially confusing constructs, e.g. multiple*

2.1.1 Core expectations for Statement Coverage (SC) assessments.

Core expectations for Statement Coverage (SC) assessments. All the other SC related sections rely on this one.

To ensure coverage of all the relevant language constructs, we decompose the material further in accordance with the Ada Reference Manual (ARM):

	Requirement Group	Description
rqg	LexicalElements	*SC expectations regarding ARM chap. 2 : Lexical Elements*
rq	DeclsAndTypes	*SC expectations regarding ARM chap. 3 : Declarations and Types*
rq	SimpleStatements	*SC expectations regarding ARM chap. 5 : Simple Statements (that do not contain other statements)*
rqg	CompoundStatements	*SC expectations regarding ARM chap. 5 : Compound Statements (that contain*
rqg	Subprograms	*SC expectations regarding ARM chap. 6 : Subprograms*
rq	Packages	*SC expectations regarding ARM chap. 7 : Packages*
rqg	VisibilityRules	*SC expectations regarding ARM chap. 8 : Visibility Rules*
rqg	Structure	*SC expectations regarding ARM chap. 10 : Program Structure*
rq	Exceptions	*SC expectations regarding ARM chap. 11 : Exceptions*
rq	GenericUnits	*SC expectations regarding ARM chap. 12 : Generic Units*

SC expectations regarding ARM chap. 2 : Lexical Elements

SC expectations regarding ARM chap. 2 : Lexical Elements

There are only very few chapter 2 items relevant to SC assessements, for which we provide subsidiary requirements:

	Requirement Group	Description
rq	Pragmas	*SC expectations regarding Pragmas (ARM 2.8)*

SC expectations regarding Pragmas (ARM 2.8)

SC expectations regarding Pragmas (ARM 2.8)

Requirement(s) Different kinds of pragmas exist, some always active, some possibly ignored depending on configuration parameters. They may be located in various contexts, some possibly never elaborated such as as subprogram declarative parts.

- Inactive pragmas shall be ignored;
- Active pragmas that were never elaborated as part of the program execution shall be reported uncovered.

Testing Strategy We check this requirement for different kinds of pragmas in various possible source regions, with the following set of testcases:

(*)	Chapter	Description
TC	**Active**	*Check SC of active pragmas*
TC	**Inactive**	*Check SC of inactive pragmas*

SC expectations regarding ARM chap. 3 : Declarations and Types

SC expectations regarding ARM chap. 3 : Declarations and Types

Requirement(s)

Declarations that were never elaborated as part of the program execution shall be reported as uncovered.

Testing Strategy

We exercise test programs for all the ARM chapter 3 elements of relevance from a structural coverage analysis perspective:

(*)	Chapter	Description
TC	**DiscreteSubtype-Defs**	`/TOR_Doc/Ada/stmt/1_Core/03_DeclsAndTypes/DiscreteSubtypeDefs`
TC	**DynObjects**	`/TOR_Doc/Ada/stmt/1_Core/03_DeclsAndTypes/DynObjects`
TC	**ObjectDecls**	`/TOR_Doc/Ada/stmt/1_Core/03_DeclsAndTypes/ObjectDecls`
TC	**SubtypeIndica-tions**	`/TOR_Doc/Ada/stmt/1_Core/03_DeclsAndTypes/SubtypeIndications`
TC	**VariantParts**	*Check SC related to Variant Parts in record type declarations*

For object declarations, we distinguish cases that might involve dynamic stack or heap allocations as these require run-time library support that some execution environments don't provide.

SC expectations regarding ARM chap. 5 : Simple Statements (that do not contain other statements)

SC expectations regarding ARM chap. 5 : Simple Statements (that do not contain other statements)

Requirement(s)

A simple statement shall be reported as uncovered if it is not executed.

Testing Strategy

We exercise every possible kind of simple statement, as defined by the ARM, in a variety of situations by way of the following set of testcases:

(*)	Chapter	Description
TC	**51_Assign**	*Check SC of Assignment statements (ARM 5.1)*
TC	**52_Null**	*Check SC of Null statements (ARM 5.2)*
TC	**57_Exit**	*Check SC of loop Exit statements (ARM 5.7)*
TC	**58_Goto**	*Check SC with Goto statements (ARM 5.8)*

SC expectations regarding ARM chap. 5 : Compound Statements (that contain

SC expectations regarding ARM chap. 5 : Compound Statements (that contain other statements)

By construction all the SC requirements apply to all the statements nested in compound ones. Specific requirements apply to each possible kind of compound statement, as identified by the ARM standard:

	Requirement Group	Description
rq	If	*SC expectations regarding If statements (ARM 5.3)*
rq	Case	*SC expectations regarding Case statements (ARM 5.4)*
rq	Loop	*SC expectations regarding Loop statements (ARM 5.5)*
rq	Block	*SC expectations regarding Block statements (ARM 5.6)*

SC expectations regarding If statements (ARM 5.3)

SC expectations regarding If statements (ARM 5.3)

Requirement(s) An IF statement comprises an IF *branch*, zero or more ELSIF branches and zero or one ELSE branch. IF and ELSIF branches feature a control expression and all the branches contain one or more statements. The nested statements get to execute when the flow reaches the branch and the control expression, if any, evaluates True.

In addition to the common requirements that apply to the nested statements, IF and ELSIF branches that are never reached shall be reported uncovered.

Testing Strategy We verify all the aspects of this requirement over

- Various forms of IF statements (with/without ELSIF branches, with/without ELSE branches),
- In various source contexts (regular functions or procedures, generic instances, package elaboration body),

All through a panel of branch selection schemes:

(*)	Chapter	Description
TC	**ELSE_Path**	*Check SC with IF statements taking [implicit] ELSE branches only.*
TC	**ELSIF_Path**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/53_If/ELSIF_Path`
TC	**IF_Path**	*Check SC with IF statements taking IF branches only.*
TC	**Multi-ple_Paths**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/53_If/Multiple_Paths`
TC	**No_Execution**	*Check SC with IF statements not executed at all.*

SC expectations regarding Case statements (ARM 5.4)

SC expectations regarding Case statements (ARM 5.4)

Requirement(s) A CASE statement starts with a *header* that introduces the controling expression, followed by zero or more CASE alternatives and zero or one OTHERS alternative. Each alternative contains other statements.

In addition to the common requirements that apply to the nested statements, CASE statement headers that are never reached shall be reported uncovered.

Testing Strategy We verify all the aspects of this requirement over

- A variety of CASE statement forms (with/without OTHERS alternative, with expression or range alternatives, with single/multi-value alternatives).
- In a variety of source contexts (regular functions or procedures, generic instances, package elaboration body),

All through a panel of alternative selection schemes:

(*)	Chapter	Description
TC	**Multi_Alt**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/54_Case/Multi_Alt`
TC	**No_Execution**	*Check SC with CASE statements not executed at all*
TC	**Single_Alt**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/54_Case/Single_Alt`

SC expectations regarding Loop statements (ARM 5.5)

SC expectations regarding Loop statements (ARM 5.5)

Requirement(s) LOOP statements feature a mandatory loop *header* that introduces the control-flow iteration scheme, and an optional set of exit statements.

In addition to the common requirements that apply to the nested statements, For or While loop headers that are never reached shall be reported uncovered.

Testing Strategy We verify all the aspects of this requirement over

- A variety of loop constructs (for, while, reverse, unconditional),
- With and without exit statements for all variants, located at various places in the nested sequence (at the beginning, at the end, somewhere in between),
- Within a variety of source contexts (regular function, procedure or package elaboration bodies, generic instances)

All through a panel of iteration schemes:

(*)	Chapter	Description
TC	**Full_Iteration**	*Check SC with LOOP statements that execute at least one full*
TC	**Incomplete_Iteration**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/55_Loop/Incomplete_It`
TC	**No_Execution**	*Check SC with LOOP statements not executed at all*
TC	**No_Iteration**	*Check SC with LOOP statements that don't even start an iteration over*

SC expectations regarding Block statements (ARM 5.6)

SC expectations regarding Block statements (ARM 5.6)

Requirement(s) A block statement contains others statements and, optionally, declarations and exception handlers as its components. The common SC requirements apply to all those nested statements. No extra requirement applies to the block specific syntactic elements.

Testing Strategy We check proper handling of block statements by way of the following set of testcases:

(*)	Chapter	Description
TC	**Nested**	*Check SC with nested block statements*
TC	**Plain**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/56_Block/Plain`

Testing Strategy

Each subsidiary requirement features its own testing strategy. Here is a direct index of the corresponding testcases:

(*)	Chapter	Description
rq	**53_If**	*SC expectations regarding If statements (ARM 5.3)*
TC	. . ELSE_Path	*Check SC with IF statements taking [implicit] ELSE branches only.*
TC	. . ELSIF_Path	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/53_If/ELSIF_Path
TC	. . IF_Path	*Check SC with IF statements taking IF branches only.*
TC	. . Multiple_Paths	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/53_If/Multiple_Paths
TC	. . No_Execution	*Check SC with IF statements not executed at all.*
rq	**54_Case**	*SC expectations regarding Case statements (ARM 5.4)*
TC	. . Multi_Alt	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/54_Case/Multi_Alt
TC	. . No_Execution	*Check SC with CASE statements not executed at all*
TC	. . Single_Alt	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/54_Case/Single_Alt
rq	**55_Loop**	*SC expectations regarding Loop statements (ARM 5.5)*
TC	. . Full_Iteration	*Check SC with LOOP statements that execute at least one full*
TC	. . Incomplete_Iteration	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/55_Loop/Incomplete_It
TC	. . No_Execution	*Check SC with LOOP statements not executed at all*
TC	. . No_Iteration	*Check SC with LOOP statements that don't even start an iteration over*
rq	**56_Block**	*SC expectations regarding Block statements (ARM 5.6)*
TC	. . Nested	*Check SC with nested block statements*
TC	. . Plain	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/56_Block/Plain

SC expectations regarding If statements (ARM 5.3) SC expectations regarding If statements (ARM 5.3)

Requirement(s) An IF statement comprises an IF *branch*, zero or more ELSIF branches and zero or one ELSE branch. IF and ELSIF branches feature a control expression and all the branches contain one or more statements. The nested statements get to execute when the flow reaches the branch and the control expression, if any, evaluates True.

In addition to the common requirements that apply to the nested statements, IF and ELSIF branches that are never reached shall be reported uncovered.

Testing Strategy We verify all the aspects of this requirement over

- Various forms of IF statements (with/without ELSIF branches, with/without ELSE branches),
- In various source contexts (regular functions or procedures, generic instances, package elaboration body),

All through a panel of branch selection schemes:

(*)	Chapter	Description
TC	**ELSE_Path**	*Check SC with IF statements taking [implicit] ELSE branches only.*
TC	**ELSIF_Path**	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/53_If/ELSIF_Path
TC	**IF_Path**	*Check SC with IF statements taking IF branches only.*
TC	**Multiple_Paths**	/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/53_If/Multiple_Paths
TC	**No_Execution**	*Check SC with IF statements not executed at all.*

SC expectations regarding Case statements (ARM 5.4) SC expectations regarding Case statements (ARM 5.4)

Requirement(s) A CASE statement starts with a *header* that introduces the controling expression, followed by zero or more CASE alternatives and zero or one OTHERS alternative. Each alternative contains other statements.

In addition to the common requirements that apply to the nested statements, CASE statement headers that are never reached shall be reported uncovered.

Testing Strategy We verify all the aspects of this requirement over

- A variety of CASE statement forms (with/without OTHERS alternative, with expression or range alternatives, with single/multi-value alternatives).
- In a variety of source contexts (regular functions or procedures, generic instances, package elaboration body),

All through a panel of alternative selection schemes:

(*)	Chapter	Description
TC	**Multi_Alt**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/54_Case/Multi_Alt`
TC	**No_Execution**	*Check SC with CASE statements not executed at all*
TC	**Single_Alt**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/54_Case/Single_Alt`

SC expectations regarding Loop statements (ARM 5.5) SC expectations regarding Loop statements (ARM 5.5)

Requirement(s) LOOP statements feature a mandatory loop *header* that introduces the control-flow iteration scheme, and an optional set of exit statements.

In addition to the common requirements that apply to the nested statements, For or While loop headers that are never reached shall be reported uncovered.

Testing Strategy We verify all the aspects of this requirement over

- A variety of loop constructs (for, while, reverse, unconditional),
- With and without exit statements for all variants, located at various places in the nested sequence (at the beginning, at the end, somewhere in between),
- Within a variety of source contexts (regular function, procedure or package elaboration bodies, generic instances)

All through a panel of iteration schemes:

(*)	Chapter	Description
TC	**Full_Iteration**	*Check SC with LOOP statements that execute at least one full*
TC	**Incom-plete_Iteration**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/55_Loop/Incomplete_It`
TC	**No_Execution**	*Check SC with LOOP statements not executed at all*
TC	**No_Iteration**	*Check SC with LOOP statements that don't even start an iteration over*

SC expectations regarding Block statements (ARM 5.6) SC expectations regarding Block statements (ARM 5.6)

Requirement(s) A block statement contains others statements and, optionally, declarations and exception handlers as its components. The common SC requirements apply to all those nested statements. No extra requirement applies to the block specific syntactic elements.

Testing Strategy We check proper handling of block statements by way of the following set of testcases:

(*)	Chapter	Description
TC	**Nested**	*Check SC with nested block statements*
TC	**Plain**	`/TOR_Doc/Ada/stmt/1_Core/052_CompoundStatements/56_Block/Plain`

SC expectations regarding ARM chap. 6 : Subprograms

SC expectations regarding ARM chap. 6 : Subprograms

This is split as a set of requirements:

	Requirement Group	Description
rq	DeclsAndCalls	*SC expectations regarding subprogram declarations and calls*
rq	Derived	*SC expectations regarding subprogram overridings*
rq	Inline	*SC expectations regarding Inlining effects*
rq	Return	*SC expectations regarding Return statements*

SC expectations regarding subprogram declarations and calls

SC expectations regarding subprogram declarations and calls

Requirement(s) SC shall be assessed correctly within all possible kinds of subprograms, everywhere they might be declared and anyhow they might be reached.

Testing Strategy We verify all the aspects of this requirement with the following set of testcases:

(*)	Chapter	Description
TC_Set	**CallSites**	*Check SC on a simple subprogram called from a wide range of source contexts*
TC	. . Actual	*Check SC on a simple subprogram called from a subprogram actual parameter*
TC	. . CompInit	*Check SC on a simple subprogram called from a record default component initialization*
TC	. . DeclInit	*Check SC on a simple subprogram called from a local object initialization*
TC	. . FormalDef	*Check SC on a simple subprogram called as part of a subprogram formal default*
TC	. . LocalElab	*Check SC on a simple subprogram called from a local package elaboration body*
TC	. . ProcBody	*Check SC on a simple subprogram called from a procedure body*
TC	**LibLevel**	`/TOR_Doc/Ada/stmt/1_Core/06_Subprograms/DeclsAndCalls/LibLevel`
TC	**Nested**	*Check SC within a panel of nested subprograms reached through a variety of*

SC expectations regarding subprogram overridings

SC expectations regarding subprogram overridings

Requirement(s) Overriding subprograms shall be recognized as distinct from the one they redefine. Exercising either one shall not influence coverage results for the other.

Testing Strategy This requirement is validated by the following set of testcases:

(*)	Chapter	Description
TC	**Simple**	*Check SC with simple subprogram overridings*
TC	**Tagged**	*Check SC with complex overridings for tagged types*

SC expectations regarding Inlining effects

SC expectations regarding Inlining effects

Requirement(s) SC assessements for simple statements shall not be influenced by subprogram Inlining.

Testing Strategy We exercise cases where inlining comes into play in various ways:

(*)	Chapter	Description
TC	**Multilevel**	`/TOR_Doc/Ada/stmt/1_Core/06_Subprograms/Inline/Multilevel`
TC	**Split**	`/TOR_Doc/Ada/stmt/1_Core/06_Subprograms/Inline/Split`

SC expectations regarding Return statements

SC expectations regarding Return statements

Requirement(s) A RETURN statement is both a statement per se and a flow-control operator.

- A RETURN statement shall be reported as uncovered when it is not executed,
- When a RETURN statement is executed, all the statements in the subprogram body that have not been executed by that moment shall be reported uncovered.

Testing Strategy We exercise subprograms that feature one or several return statements in various contexts, conditional or not, arranging to get into combinations of the possible execution flow variants.

(*)	Chapter	Description
TC	**Functions**	`/TOR_Doc/Ada/stmt/1_Core/06_Subprograms/Return/Functions`
TC	**Procedures**	*Check SC with return statements in Procedures*

While each requirement features a distinct testing strategy, a common scheme governs the organization; all the testcases arrange to have a mix of situations where from some subprograms:

- Some but not all of the code executes,
- All the code executes,
- None of the code executes (subprogram is not called).

SC expectations regarding ARM chap. 7 : Packages

SC expectations regarding ARM chap. 7 : Packages

Requirement(s)

The core SC requirements on statements and declarations shall be obeyed in all the possible context variations allowed by package related constructs.

Testing Strategy

We exercise the following set of testcases:

(*)	Chapter	Description
TC	**DeclRegions**	*Check the correctness of SC assessments for subprograms declared in various package regions*
TC	**ElabBody**	`/TOR_Doc/Ada/stmt/1_Core/07_Packages/ElabBody`

SC expectations regarding ARM chap. 8 : Visibility Rules

SC expectations regarding ARM chap. 8 : Visibility Rules

There are only very few chapter 8 items relevant to SC assessements, for which we provide subsidiary requirements:

	Requirement Group	Description
rq	Renamings	*SC expectations regarding Renaming declarations (ARM 8.5)*

SC expectations regarding Renaming declarations (ARM 8.5)

SC expectations regarding Renaming declarations (ARM 8.5)

Requirement(s)

- Renaming declarations shall be recognized and processed as regular object declarations,
- Coverage achieved through package or subprogram renamings shall be as if achieved through the renamed entity.

Testing Strategy We exercise the following set of testcases:

(*)	Chapter	Description
TC	**Objects**	`/TOR_Doc/Ada/stmt/1_Core/08_VisibilityRules/Renamings/Objects`
TC	**Other**	*Check that package or subprogram renamings are handled correctly*

SC expectations regarding ARM chap. 10 : Program Structure

SC expectations regarding ARM chap. 10 : Program Structure

There are only very few chapter 10 items relevant to SC assessements, for which we provide subsidiary requirements:

	Requirement Group	Description
rq	Subunits	*SC expectations regarding Subunits (ARM 10.1.3)*

SC expectations regarding Subunits (ARM 10.1.3)

SC expectations regarding Subunits (ARM 10.1.3)

Requirement(s) SC assessments shall operate in subunits as in regular units.

Testing Strategy We perform a panel of basic statement coverage assessments within several kinds of subunits (subprograms or packages), declared in various possible source contexts (visible or private part of package specs, package body, or subprogram declarative parts):

(*)	Chapter	Description
TC	**Mix**	*Exercise a mix of subprogram and package subunits, with multiple levels*
TC	**Packages**	`/TOR_Doc/Ada/stmt/1_Core/10_Structure/Subunits/Packages`
TC	**Subprograms**	`/TOR_Doc/Ada/stmt/1_Core/10_Structure/Subunits/Subprograms`

SC expectations regarding ARM chap. 11 : Exceptions

SC expectations regarding ARM chap. 11 : Exceptions

Requirement(s)

Statement Coverage shall be assessed correctly for Ada exceptions mechanism as described in Chapter 11 "Exceptions" of the Ada Reference Manual. In particular:

- `raise` statements shall be reported uncovered when unreached,
- The flow-control effects of implicit and explicit exception raises shall be handled correctly:
  - statements that don't execute because of a raise shall be reported uncovered,
  - statements that only execute partially because of an expression evaluation interrupted shall *not* be reported uncovered.
- The tool shall support user defined exceptions as well as language predefined ones,
- The full set of core SC requirements apply to all the statements within exception handlers.

Testing Strategy

We validate all those requirements through a set of testcases that resort to implicit or explicit exceptions for flow-control transfer purposes. All these tescases obey a common testing variation pattern all along; with checks that involve:

- Explicit `raise` statements executed or not, followed by other statements or not
- Variations of these in function, subprogram, or package elaboration bodies, directly within the toplevel sequence of statements, within nested block, conditional or loop statements,
- With one or more candidate handlers at different levels of nesting, always within a single body.

(*)	Chapter	Description
TC_Set	**CutEvals**	*Check SC with expressions that can fail to evaluate because of exceptions*
TC_Set	.. And	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If/RemoteChec
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return/IndexC
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return/RangeC
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return/Remote
TC_Set	.. Flip	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If/IndexChec
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If/RangeChec
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If/RemoteChe
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return/Index
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return/Range
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return/Remot
TC_Set	.. Or	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If/RemoteCheck
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return/IndexCh
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return/RangeCh
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return/RemoteC

Continued on

Table 2.1 – continued from previous page

TC_Set	.. Value	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If/IndexChe
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If/RangeChe
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If/RemoteCh
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return/Inde
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return/Rang
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return/Remo
TC	**Handlers**	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/Handlers
TC	**Propagation**	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/Propagation
TC	**Raise**	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/Raise

SC expectations regarding ARM chap. 12 : Generic Units

SC expectations regarding ARM chap. 12 : Generic Units

Requirement(s)

The code of a generic unit shall be reported as covered only if the generic is instantiated, and the instantiation is either called (in case of a subprogram instantiation or a subprogram declared in generic package) or elaborated (in case of a declaration or elaboration code in a generic package). In all the other cases the code from a generic unit shall be reported as uncovered.

Testing Strategy

We check libray-level and local generic declarations and instantiations by way of the following set of testcases:

(*)	Chapter	Description
TC	**LibrayLevelInstantiations**	*Check that in case of a library-level instantiation the code of the generic*
TC	**LocalInstantiations**	/TOR_Doc/Ada/stmt/1_Core/12_GenericUnits/LocalInstantiations

A few chapters are not included for the following reasons:

Chapter	Not included because ...
ARM chap. 1 : General	No language construct described
ARM chap. 4 : Names and Expressions	The described constructs are not considered on their own for coverage analysis purposes. The coverage information is computed for enclosing statement or declaration constructs.
ARM chap. 9 : Tasks and Synchronization	The execution profile being qualified is based on Zero Foot Print run-time, which does not support any construct described in this chapter
ARM chap. 13 : Representation Issues	Constructs described in this chapter do not result in executable code so they are of no interest for coverage analysis.

2.1.2 SC expectations regarding the combination of multiple

SC expectations regarding the combination of multiple execution traces together.

Requirement(s)

When the coverage achieved by multiple execution traces is evaluated, a statement coverage violation shall only be reported when it would have been for all the traces individually.

Testing Strategy

(*)	Chapter	Description
TC	**MultipleForOne**	*Exercise a three way if statement with three different*

2.1.3 SC expectations focused on statement sequences and control-flow transfers.

SC expectations focused on statement sequences and control-flow transfers.

Requirement(s)

Statement coverage is assessed correctly for straightline sequences (basic blocks) of statements and combinations of such constructed with jump-like control flow transfers (goto, return, raise, exit).

If execution reaches the top of a basic block, no statement in the block is reported uncovered. Conversely, if execution doesn't reach the top of a basic block, all the statements in the block shall be reported uncovered.

Testing Strategy

All the aspects of this requirement are validated by a the following set of testcases :

(*)	Chapter	Description
TC_Set	**Contexts**	*Check that basic blocks of statements are recognized in various possible*
TC	. . ElabBody	*Check that statements are recognized in package elaboration bodies and*
TC	. . Subprograms	*Check that statements are recognized in subprogram bodies and nested blocks*
TC_Set	**Controls**	*Check that various forms of control-flow transfers are handled properly.*
TC	. . Goto	*Exercise a function where flow control is achieved with goto*
TC	. . LocalRaise	*Exercise a function where flow control is achieved with*
TC	. . LoopExit	*Exercise a function where flow control is achieved with loop*
TC	. . Return	*Exercise a function where flow control is achieved with goto*

2.1.4 SC expectations regarding exemption regions.

SC expectations regarding exemption regions.

Requirement(s)

Proper behavior of the Coverage Exemptions facility translates into the following set of rules:

Rule #	Description
1	Exempted regions shall all be synthesized in a distinct section of the output report, with a single message per region.
2	Each exemption message shall expose the range of source locations covered by the region declaration, and indicate whether 0 or more coverage violations were actually exempted within this range.
3	Exemption regions shall not influence the diagnostics emitted on constructs outside of them.
4	The report section dedicated to exemption regions shall not contain any other kind of information.
5	Exemption regions may be declared to exempt various levels of syntactic constructs such as statements or subprograms, and groups of such in all the contexts where they may appear.

Testing Strategy

We validate all the aspects of this requirement with a set of testcases, subdivided in sections according to point #5 - demonstrate ability to exempt various groups of syntactic constructs:

(*)	Chapter	Description
TC	**MixInUnits**	`/TOR_Doc/Ada/stmt/Exemptions/MixInUnits`
TC	**StmtsInBody**	`/TOR_Doc/Ada/stmt/Exemptions/StmtsInBody`
TC	**StmtsInElab**	*Check processing of exemptions applying to groups of statements within a*
TC	**StmtsInHandler**	*Check processing of exemptions applying to groups of statements within local*
TC	**StmtsInSubprog**	`/TOR_Doc/Ada/stmt/Exemptions/StmtsInSubprog`
TC	**SubprogsInUnit**	*Check processing of exemptions applying to groups of entire subprograms.*
TC	**WholeBody**	*Check processing of exemptions applying to whole subprogram bodies.*

Each test features some mix of exempted regions and regular code, exercised in multiple manners to cover well identified sections of the program. All the other aspects of the requirement (rules #1 to #4) are validated by the fact that all the tests run as expected.

For every single stated expectation, exempted region or non-exempted violation, the testsuite driver checks if it appears in the report section expected for it's kind (in addition to the regular expected/reported match checks).

2.1.5 SC expectations for mixes of statement constructs, representative of real

SC expectations for mixes of statement constructs, representative of real application code.

Requirement(s)

The Core requirements are honored on programs mixing arbitrary Ada constructs together, with arbitrary levels of syntactic nesting (such as loops within tests within subprograms etc).

Testing Strategy

We exercise multiple cases of functional code featuring a variety of constructs nested within each other (For, While, Case, If), and for every case check that the behavior matches expectations in a wide range of possible situations:

- Loops entered or not,
- Possible case selections taken alone or combined with others,
- If controls evaluated True only, False only or both,
- Combinations of these allowed by the nesting structure.

(*)	Chapter	Description
TC	**If_For_If_Case**	`/TOR_Doc/Ada/stmt/MixedConstructs/If_For_If_Case`
TC	**While_If1**	*Exercise a simple If statement within a While loop.*
TC	**While_If4**	*Exercise a three way If statement within a While loop.*
TC	**While_If_Case_If**	`/TOR_Doc/Ada/stmt/MixedConstructs/While_If_Case_If`

2.1.6 SC expectations regarding potentially confusing constructs, e.g. multiple

SC expectations regarding potentially confusing constructs, e.g. multiple statements sharing a line.

Requirement(s)

Statement Coverage assessment remains well defined in the presence of code constructs that could fool simple minded analysis engines.

Testing Strategy

Check a panel of cases where code construct particularities could concievably cause inaccuracies or errors in coverage diagnostics if the tool were to implement too simple analysis schemes:

(*)	Chapter	Description
TC	**Homonyms**	*Check that coverage is not affected by the presence of subprograms oveloading.*
TC	**Indentation**	`/TOR_Doc/Ada/stmt/Robustness/Indentation`
TC	**InhibitedCode**	`/TOR_Doc/Ada/stmt/Robustness/InhibitedCode`
TC	**LongSequence**	`/TOR_Doc/Ada/stmt/Robustness/LongSequence`
TC	**MultiLineStatements**	*Check that statements spanning multiple lines are handled properly.*
TC	**MultiStatementLines**	*Check that multiple statements located on the same source line are*
TC	**Multinops**	*Check that sequences of multiple nop statements (e.g. null or pragma) are*
TC	**PartialEval**	`/TOR_Doc/Ada/stmt/Robustness/PartialEval`

2.2 Decision Coverage (DC) TORs and Testcases for Ada

Decision Coverage (DC) TORs and Testcases for Ada

This is split in several parts:

	Requirement Group	Description
rq	Core	*Core expectations for Decision Coverage*
rq	Consolidation	*DC expectations regarding the combination of multiple*
rq	Exemptions	*DC expectations regarding exemption regions.*
rq	MixedConstructs	*DC expectations for mixes of statement and decision constructs representative*
rq	Robustness	*DC expectations with respect to potentially confusing constructs,*

2.2.1 Core expectations for Decision Coverage

Core expectations for Decision Coverage (DC) assessments. All the other DC related sections rely on this one.

Requirement(s)

A *decision* is defined to be any Boolean expression that directly controls the behavior of IF, WHILE and EXIT-WHEN control-flow constructs. Only the expression as a whole is considered a decision; subexpressions are not decisions on their own. The types involved need not be restricted to the standard Boolean type; they may subtypes or types derived from the Ada fundamental type.

In this context, and in addition to the rules governing Statement Coverage, the following set of extra rules shall be obeyed for Decision Coverage (DC) assessements:

Rule #	Description
1	When the control-flow statement influenced by a decision has not been executed, that statement shall be reported as not covered and nothing shall be reported about the decision.
2	When a decision is evaluated only True or False, it shall be reported as only partially covered. In this case as in the previous one, the tool shall designate the decision with an unambiguous file-name:line#:col# reference.
3	When a decision is evaluated both True and False, no decision coverage violation shall be reported for it.
4	When a decision is never evaluated even though the enclosing statement has been executed (e.g. because of exceptions preventing the computation of an outcome), the decision shall be reported as never evaluated.
5	The tool shall be able to handle arbitrarily complex decisions in any context where they might appear.

Testing Strategy

We validate all the DC rules thanks to three main subsets of testcases:

	Requirement Group	Description
TC_Set	NoEval	*Check the correctness of DC assessments on decisions that are not entirely evaluated (don't yield*
TC_Set	Operands	*Check the correctness of DC assessments on decisions featuring various possible forms*
TC_Set	Topologies	*Check the correctness of DC assessments on expressions of various topologies everywhere they*

Rules #1 to 3 are validated by variations exercised in every individual testcase, where we consistently check each decision of interest in multiple manners, always including:

- a situation where the statements exposing the decision aren't executed at all (*rule #1*).
- a set of vectors where the decision evaluates only True (*rule #2*),
- a set of vectors where the decision evaluates only False (*rule #2*),
- a set of vectors where the decision evaluates both True and False (*rule #3*),

Rule #4 and #5 are addressed by the organization of the set of testcase groups presented in the table above.

2.2.2 DC expectations regarding the combination of multiple

DC expectations regarding the combination of multiple execution traces together.

Requirement(s)

When the coverage achieved by multiple execution traces is evaluated, a decision coverage violation shall only be reported when it would have been for all the traces individually.

Testing Strategy

We exercise consolidation of traces obtained for single vector invocations over a range of basic decisions.

For each decision, first run with every possible relevant input vector independently, then check all the possible combinations of those input vectors without repetitions.

(*)	Chapter	Description
TC	**And**	`/TOR_Doc/Ada/decision/Consolidation/And`
TC	**Or**	*Exercise consolidation over A or else B.*

2.2.3 DC expectations regarding exemption regions.

DC expectations regarding exemption regions.

Requirement(s)

Exemption regions shall operate over decision coverage violations as they do for statement coverage violations.

Testing Strategy

Check proper behavior on decisions placed in a restricted set of possible contexts, to validate that the exemption facility works for decision coverage violations specifically. Proper behavior over context variations is deemed validated by the statement coverage testcases.

(*)	Chapter	Description
TC	**PrecondInBody**	`/TOR_Doc/Ada/decision/Exemptions/PrecondInBody`

2.2.4 DC expectations for mixes of statement and decision constructs representative

DC expectations for mixes of statement and decision constructs representative of real application code.

Requirement(s)

The Core requirements are honored on programs mixing arbitrary Ada constructs together, with arbitrary levels of syntactic nesting (such as loops within tests within subprograms etc).

Testing Strategy

We exercise multiple cases of functional code featuring a variety of constructs nested within each other (For, While, Case, If), and for every case check that the behavior matches expectations in a wide range of possible situations:

- Loops entered or not,
- Possible case selections taken alone or combined with others,
- If controls evaluated True only, False only or both,
- Combinations of these allowed by the nesting structure.

(*)	Chapter	Description
TC	**If_ElsAbs**	*Exercise an If/Elsif construct involving complex decisions with computations*
TC	**If_For_If_Case**	*Exercise a toplevel If statement with an else sequence embedding a For loop,*
TC	**While_If1**	`/TOR_Doc/Ada/decision/MixedConstructs/While_If1`
TC	**While_If4**	`/TOR_Doc/Ada/decision/MixedConstructs/While_If4`
TC	**While_If_Case_If**	`/TOR_Doc/Ada/decision/MixedConstructs/While_If_Case_If`

2.2.5 DC expectations with respect to potentially confusing constructs,

DC expectations with respect to potentially confusing constructs, e.g. decisions spanning multiple lines.

Requirement(s)

The Core Decision Coverage requirements shall remain satisfied in presence of constructs that could fool simple minded analyzers.

Testing Strategy

Check a panel of cases where code construct particularities could concievably cause inaccuracies or errors in coverage diagnostics if the tool were to implement too simple analysis schemes:

(*)	Chapter	Description
TC	**InhibitedCode**	`/TOR_Doc/Ada/decision/Robustness/InhibitedCode`
TC_Set	**MultiLineDecisions**	*Check that Boolean expressions spanning multiple lines are handled correctly.*
TC_Set	.. And	`/TOR_Doc/Ada/decision/Robustness/MultiLineDecisions/And`
TC	.. If	`/TOR_Doc/Ada/decision/Robustness/MultiLineDecisions/And/If`
TC	.. While	`/TOR_Doc/Ada/decision/Robustness/MultiLineDecisions/And/While`
TC_Set	.. Or	`/TOR_Doc/Ada/decision/Robustness/MultiLineDecisions/Or`
TC	.. If	`/TOR_Doc/Ada/decision/Robustness/MultiLineDecisions/Or/If`
TC	.. While	`/TOR_Doc/Ada/decision/Robustness/MultiLineDecisions/Or/While`
TC	**NestedIf**	`/TOR_Doc/Ada/decision/Robustness/NestedIf`

2.3 Modified Condition/Decision Coverage (MCDC) TORs and Testcases for Ada

Modified Condition/Decision Coverage (MCDC) TORs and Testcases for Ada

This is split in several parts:

	Requirement Group	Description
rq	Core	*Core expectations for MCDC assessments.*
rq	Consolidation	*MCDC expectations regarding the combination of multiple*
rq	Exemptions	*MCDC expectations regarding exemption regions.*
rq	MixedConstructs	*MCDC expectations for mixes of various statement and decision constructs*
rq	Robustness	*MCDC expectations with respect to potentially confusing constructs,*

2.3.1 Core expectations for MCDC assessments.

Core expectations for MCDC assessments. All the other sub-sections rely on this one.

Requirement(s)

Compared to Decision Coverage, MCDC assessements enlarges the set of expressions that shall be processed as decisions and introduces rules regarding the operands that consistute the expressions.

We distinguish two categories of Boolean expressions for MCDC:

- *Complex* expressions, that feature at least two Boolean operands combined with AND-THEN or OR-ELSE short-circuit operators.
- *Simple* expressions, that are not complex per the preceding definition.

The types involved need not be restricted to the standard Boolean type; they may subtypes or types derived from the Ada fundamental type.

In addition to any expression that directly influence control-flow constructs, the tool shall process any *complex* Boolean expression as a decision, regardless of the context where it apppears, for example on the right-hand side of an assignment, as part of an object declaration initializer, as a subprogram actual or within an active assertion pragma.

The Boolean operands of the short-circuit operators in a decision are called *conditions*. Sub-decisions nesting becomes possible from the variety of contexts where an expression needs to be treated as a decision. For example, `if A and then Op (B or else C)` has two decisions, one with two conditions (B and C) used as an actual in a function call to Op, and an outer one with two conditions as well (A and the function call itself).

All the DC rules apply unchanged to the full set of decisions as defined here. Rule #3, about decisions evaluated both True and False, is complemented by an additional rule:

Rule #	Description
3c	For expressions evaluated both True and False, the tool shall report every condition for which the independent effect was not demonstrated. Such condition specific diagnotics shall designate the particular condition source location.

Testing Strategy

The testing strategy is similar to the one chosen for the DC core requirement, with the following set of testcases:

	Requirement Group	Description
TC_Set	NoEval	*Check the correctness of MCDC assessments on decisions that are not entirely evaluated (don't*
TC_Set	Operands	*Check the correctness of MCDC assessments on decisions featuring all the possible forms*
TC_Set	Topologies	*Check the correctness of MCDC assessments on expressions of arbitrary topology*

Rules 1 to 3c are validated by variations exercised in every individual testcase, where we consistenly check each decision of interest in multiple manners, always including:

- a situation where the statements exposing the decision aren't executed at all (*rule #1*),
- a set of vectors where the decision evaluates only True (*rule #2*),
- a set of vectors where the decision evaluates only False (*rule #2*),
- sets of vectors where the decision evaluates both True and False, with at least
  - one set not demonstrating the independent effect of any condition (*rules #3 and 3c*),
  - one set demonstrating the independent effect of each condition alone (*rules #3 and 3c*),
  - one set demonstrating the independent effect of all the conditions (*rule #3 and 3c*).

2.3.2 MCDC expectations regarding the combination of multiple

MCDC expectations regarding the combination of multiple execution traces together.

Requirement(s)

When the coverage achieved by multiple execution traces is evaluated, an mcdc violation shall only be reported when it would have been for all the traces individually.

Testing Strategy

We exercise consolidation of traces obtained for single vector invocations over a range of basic decisions.

For decisions with two operands, first run with every possible input vector independently, then check all the possible combinations of those input vectors without repetitions.

We operate similarily for decisions with three operands, except we aggregate inputs that differ only on non-evaluated conditions.

(*)	Chapter	Description
TC	**And**	`/TOR_Doc/Ada/mcdc/Consolidation/And`
TC	**AndAB_OrCD**	*Exercise consolidation over Andthen (X => A and then B, Y => C or else D),*
TC	**Or**	*Exercise consolidation over A or else B.*
TC	**pOrpAnd**	*Exercise consolidation over (A or else B) and then C.*

2.3.3 MCDC expectations regarding exemption regions.

MCDC expectations regarding exemption regions.

Requirement(s)

Exemption regions shall operate over MCDC violations as they do for statement coverage violations.

Testing Strategy

Check proper behavior on decisions placed in a restricted set of possible contexts, to validate that the exemption facility works for mcdc violations specifically. Proper behavior over context variations is deemed validated by the statement coverage testcases.

(*)	Chapter	Description
TC	**PrecondInBody**	*Exercise the use of an exemption region to exempt a functional precondition*

2.3.4 MCDC expectations for mixes of various statement and decision constructs

MCDC expectations for mixes of various statement and decision constructs representative of real application code.

Requirement(s)

The Core requirements are honored on programs mixing arbitrary Ada constructs together, with arbitrary levels of syntactic nesting (such as loops within tests within subprograms etc).

Testing Strategy

We exercise multiple cases of functional code featuring a variety of constructs nested within each other (For, While, Case, If), and for every case check that the behavior matches expectations in a wide range of possible situations:

- Loops entered or not,
- Possible case selections taken alone or combined with others,
- If controls evaluated True only, False only or both,
- Combinations of these allowed by the nesting structure.

(*)	Chapter	Description
TC	**If_ElsAbs**	*Exercise an If/Elsif construct involving complex decisions with computations*
TC	**If_For_If_Case**	*Exercise a toplevel If statement with an else sequence embedding a For loop,*
TC	**While_If4**	*Exercise a three way If statement within a While loop.*
TC	**While_If_Case_If**	*Exercise a simple If statement nested in one of the possible selections of a*

2.3.5 MCDC expectations with respect to potentially confusing constructs,

MCDC expectations with respect to potentially confusing constructs, e.g. multiple decisions sharing a line.

Requirement(s)

The Core MCDC requirements shall remain satisfied in presence of constructs that could fool simple minded analyzers.

Testing Strategy

Check a panel of cases where code construct particularities could concievably cause inaccuracies or errors in coverage diagnostics if the tool were to implement too simple analysis schemes:

(*)	Chapter	Description
TC	**InhibitedCode**	/TOR_Doc/Ada/mcdc/Robustness/InhibitedCode
TC_Set	**MultiDecision-Line**	*Check that when multiple decisions occur on the same source line, individual*
TC_Set	. . TwoFormals	/TOR_Doc/Ada/mcdc/Robustness/MultiDecisionLine/TwoFormals
TC	. . AndAB_AndAB	/TOR_Doc/Ada/mcdc/Robustness/MultiDecisionLine/TwoFormals/AndAB_And
TC	. . AndAB_AndCD	/TOR_Doc/Ada/mcdc/Robustness/MultiDecisionLine/TwoFormals/AndAB_And
TC	. . AndAB_OrAB	/TOR_Doc/Ada/mcdc/Robustness/MultiDecisionLine/TwoFormals/AndAB_OrA
TC	. . AndAB_OrCD	/TOR_Doc/Ada/mcdc/Robustness/MultiDecisionLine/TwoFormals/AndAB_OrC
TC_Set	**MultiLineDecisions**	*Check that decisions spanning multiple lines are recognized and processed*
TC_Set	. . And	/TOR_Doc/Ada/mcdc/Robustness/MultiLineDecisions/And
TC	. . If	/TOR_Doc/Ada/mcdc/Robustness/MultiLineDecisions/And/If
TC	. . Return	/TOR_Doc/Ada/mcdc/Robustness/MultiLineDecisions/And/Return
TC_Set	. . Or	/TOR_Doc/Ada/mcdc/Robustness/MultiLineDecisions/Or
TC	. . If	/TOR_Doc/Ada/mcdc/Robustness/MultiLineDecisions/Or/If
TC	. . Return	/TOR_Doc/Ada/mcdc/Robustness/MultiLineDecisions/Or/Return
TC_Set	**Recurse**	*Check that recursive decision evaluations are processed correctly.*
TC	. . And	/TOR_Doc/Ada/mcdc/Robustness/Recurse/And
TC	. . Andor_RA	/TOR_Doc/Ada/mcdc/Robustness/Recurse/Andor_RA
TC	. . Andor_RC	/TOR_Doc/Ada/mcdc/Robustness/Recurse/Andor_RC

This page is intentionally left blank.

CHAPTER

THREE

LANGUAGE AGNOSTIC TORS AND TESTCASES

Language agnostic TORs and Testcases

	Requirement Group	Description
rq	Report	*Describe the tool output report format and check compliance to it*

3.1 Describe the tool output report format and check compliance to it

Describe the tool output report format and check compliance to it

3.1.1 Requirement(s)

The tool qualified output is the synthetic report produced by the –annotate=report command line option. The output report shall:

- Start with an explicit “COVERAGE REPORT” indication
- End with an explicit “END OF REPORT” indication
- Feature up to four sections in between: “Assessment Context”, “Coverage violations”, “Exempted Regions” (optionally), and “Analysis summary”
- The “Assessment Context” section shall summarize all the elements of relevance to characterize the reported assessment unambiguously:
  - Tool version identifier
  - Date and time of execution
  - Full command line issued to produce the report
  - Coverage criteria assessed (e.g. “stmt” or “stmt+decision”)
  - Information about each of the input trace files:
    - trace tag & file name
    - trace production date & time
    - program executable file name
- The “Coverage violations” section shall feature a set of subsections, one for each criterion assessed with
  - A list of all the criterion obligations unsatisfied (violated) by the set of executions conveyed by the provided execution traces, and
  - A count of those violations

- The “Exempted Regions”, when present, shall
  - Summarize information about all the exemption regions in the scope of the examined coverage obligations, and
  - End with a count of those regions

  It shall only be present when exemption regions were declared in the sources of interest.
- The “Analysis summary” section shall expose
  - A single synthetic line for each coverage criterion assessed, with a count of the non-exempted violations of that particular criterion.
  - A single synthetic line providing the count of exempted regions.

3.1.2 Testing Strategy

We provide testcases to check that the reports produced by the tool satisfy the requirements in a range of different situations:

(*)	Chapter	Description
TC	**AllCriteria**	`/TOR_Doc/Common/Report/AllCriteria`
TC	**EmptySections**	*Check output report compliance to expectations on a case where no violation or*
TC	**Exemptions**	`/TOR_Doc/Common/Report/Exemptions`
TC	**MultipleTraces**	*Check output report compliance to expectations on a case involving a multiple*
TC	**SingleTrace**	`/TOR_Doc/Common/Report/SingleTrace`

In each of these testcases, we validate the presence and structure of all the expected sections and items in the “Assessment Context” part. We don’t verify the general correctness of reported violations vs expectations here. This is defined and validated by sibling chapters of this TOR document, in particular those dedicated to specific coverage criteria.

The output report correctness also gets validated as part of those other chapters, with the testsuite engine checking that:

- All the reported coverage violations are found in the correct report section,
- The synthetic counters at the end of sections reporting violations or exempted regions is correct (matches the actual number of items reported in the section),
- The synthetic counters reported in the global “Analysis Summary” section are the same as those reported in individual sections,
- There is an exact correspondance between the reported items and the testcase expectations, with a test passing only if everything reported was stated as expected and everything expected was found to be reported.

CHAPTER

FOUR

APPENDIX

This is split as the following set of sections:

This page is intentionally left blank.

CHAPTER

FIVE

TEST CASES

5.1 Check the correctness of DC assessments on decisions that are not entirely evaluated (don't yield

Check the correctness of DC assessments on decisions that are not entirely evaluated (don't yield an outcome) even though the statement they control is executed, typically when evaluation is interrupted by an exception occurrence.

We exercise a range of expressions where each condition could be made to raise an exception in a panel of cases, checking that

- a `decision never evaluated` diagnostic is emitted when no evaluation succeeds at all, and that
- exception occurrences don't influence the coverage achieved by otherwise successful evaluations.

(*)	Chapter	Description
TC_Set	**And**	*DC testcases for a decision of the form A and then B*
TC	.. IndexCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/And/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/And/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/And/RemoteCheck
TC_Set	**Flip**	*DC testcases for a decision of the form not A, negated simple expression.*
TC	.. IndexCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Flip/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Flip/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Flip/RemoteCheck
TC_Set	**Or**	*DC testcases for a decision of the form A or else B*
TC	.. IndexCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Or/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Or/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Or/RemoteCheck
TC_Set	**Value**	*DC testcases for a decision of the form A, simple expression without*
TC	.. IndexCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Value/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Value/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/decision/1_Core/NoEval/Value/RemoteCheck

5.1.1 DC testcases for a decision of the form `A and then B`

DC testcases for a decision of the form `A and then B`

5.1.2 DC testcases for a decision of the form `not A`, negated simple expression.

DC testcases for a decision of the form `not A`, negated simple expression.

5.1.3 DC testcases for a decision of the form `A or else B`

DC testcases for a decision of the form `A or else B`

5.1.4 DC testcases for a decision of the form A, simple expression without

DC testcases for a decision of the form A, simple expression without negation.

5.2 Check the correctness of DC assessments on decisions featuring various possible forms

Check the correctness of DC assessments on decisions featuring various possible forms of conditions, not only Boolean variables.

Exercise basic decisions involving various possible kinds of conditions (attribute references, subprogram calls, ...), either isolated or mixed with other kinds in a variety of program contexts.

We make sure here to exercise cases known to require internal comparisons, to demonstrate that the tool knows to distinguish internal tests aimed at computing a condition value versus tests aimed at computing the decision from the condition values.

(*)	Chapter	Description
TC	**Attributes**	*Check DC on expressions with one or two operands involving Ada attributes*
TC	**Calls**	*Check DC on expressions with one or two function call conditions.*
TC	**Subtypes**	*Check DC on expressions with conditions of a Boolean subtype or derived type.*
TC	**Tests**	*Check DC on expressions with conditions involving explicit tests,*

5.2.1 Check DC on expressions with one or two operands involving Ada attributes

Check DC on expressions with one or two operands involving Ada attributes such as String'Length.

5.2.2 Check DC on expressions with one or two function call conditions.

Check DC on expressions with one or two function call conditions.

5.2.3 Check DC on expressions with conditions of a Boolean subtype or derived type.

Check DC on expressions with conditions of a Boolean subtype or derived type.

5.2.4 Check DC on expressions with conditions involving explicit tests,

Check DC on expressions with conditions involving explicit tests, e.g. value comparisons.

5.3 Check the correctness of DC assessments on expressions of various topologies everywhere they

Check the correctness of DC assessments on expressions of various topologies everywhere they might appear.

We exercise a panel of expression topologies over Boolean variables (single value, negated, two or more combined with and-then or or-else, ...), each placed in a variety of program contexts. We distinguish explicit control-flow constructs from other contexts.

(*)	Chapter	Description

Continued on next page

Table 5.1 – continued from previous page

TC_Set	**And**	*Check DC on a decision of the form A and then B*
TC	.. Actual	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Actual
TC	.. Aggregate	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Aggregate
TC	.. Assign	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Assign
TC	.. Case	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Case
TC	.. DeclInit	/TOR_Doc/Ada/decision/1_Core/Topologies/And/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Discriminant
TC	.. Elsif	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Elsif
TC	.. Exit	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Exit
TC	.. IfElse	/TOR_Doc/Ada/decision/1_Core/Topologies/And/IfElse
TC	.. IfNoElse	/TOR_Doc/Ada/decision/1_Core/Topologies/And/IfNoElse
TC_Set	.. Pragmas	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Pragmas
TC	.. DbgBody	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Pragmas/DbgBody
TC	.. PreBody	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Pragmas/PreBody
TC	.. PreDecl	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Pragmas/PreDecl
TC	.. PreMix	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Pragmas/PreMix
TC	.. Return	/TOR_Doc/Ada/decision/1_Core/Topologies/And/Return
TC	.. While	/TOR_Doc/Ada/decision/1_Core/Topologies/And/While
TC_Set	**Flip**	*Check DC on a decision of the form not X*
TC	.. Elsif	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/Elsif
TC	.. IfElse	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/IfElse
TC	.. IfNoElse	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/IfNoElse
TC_Set	.. Pragmas	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/Pragmas
TC	.. PreBody	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/Pragmas/PreBody
TC	.. PreDecl	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/Pragmas/PreDecl
TC	.. While	/TOR_Doc/Ada/decision/1_Core/Topologies/Flip/While
TC_Set	**Or**	*Check DC on a decision of the form A or else B*
TC	.. Actual	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Actual
TC	.. Aggregate	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Aggregate
TC	.. Assign	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Assign
TC	.. Case	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Case
TC	.. DeclInit	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Discriminant
TC	.. Elsif	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Elsif
TC	.. Exit	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Exit
TC	.. IfElse	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/IfElse
TC	.. IfNoElse	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/IfNoElse
TC_Set	.. Pragmas	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Pragmas
TC	.. DbgBody	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Pragmas/DbgBody
TC	.. PreBody	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Pragmas/PreBody
TC	.. PreDecl	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Pragmas/PreDecl
TC	.. Return	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/Return
TC	.. While	/TOR_Doc/Ada/decision/1_Core/Topologies/Or/While
TC_Set	**Value**	*Check DC on a decision of the simplest*
TC	.. Actual	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Actual
TC	.. Aggregate	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Aggregate
TC	.. Assign	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Assign
TC	.. DeclInit	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Discriminant
TC	.. Elsif	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Elsif
TC	.. IfElse	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/IfElse
TC	.. IfNoElse	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/IfNoElse
TC_Set	.. Pragmas	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Pragmas
TC	.. PreBody	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Pragmas/PreBody
TC	.. PreDecl	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Pragmas/PreDecl
TC	.. Return	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/Return

Continued on next page

Table 5.1 – continued from previous page

TC	.. While	/TOR_Doc/Ada/decision/1_Core/Topologies/Value/While

5.3.1 Check DC on a decision of the form `A and then B`

Check DC on a decision of the form `A and then B` in various contexts.

5.3.2 Check DC on a decision of the form `not X`

Check DC on a decision of the form `not X` in various contexts.

5.3.3 Check DC on a decision of the form `A or else B`

Check DC on a decision of the form `A or else B` in various contexts.

5.3.4 Check DC on a decision of the simplest

Check DC on a decision of the simplest possible form (not involving any boolean operator), in various contexts.

5.4 Exercise consolidation over `A and then B`.

Exercise consolidation over `A and then B`.

5.4.1 ada sources

- andthen.adb
- andthen.ads
- test_andthen_v0.adb
- test_andthen_v1.adb
- test_andthen_v2.adb

5.4.2 c sources

5.4.3 h sources

5.5 Exercise consolidation over `A or else B`.

Exercise consolidation over `A or else B`.

5.5.1 ada sources

- orelse.adb
- orelse.ads
- test_orelse_v0.adb
- test_orelse_v1.adb

- test_orelse_v2.adb

5.5.2 c sources

5.5.3 h sources

5.6 Exercise the use of an exemption region to exempt a functional precondition

Exercise the use of an exemption region to exempt a functional precondition expressed manually (explicit check) at the beginning of a subprogram body.

Check:

- Situations where the precondition is always evaluated True (nominal case, exempted DC violation)
- Situations where the precondition is evaluated only False (force error case, exempted DC violation)
- Consolidation of the previous cases: precondition exercized both ways (no exempted violation)

5.6.1 ada sources

- ranges.adb
- ranges.ads
- test_ranges_invalid.adb
- test_ranges_no_overlap.adb
- test_ranges_overlap.adb

5.6.2 c sources

5.6.3 h sources

5.7 Exercise an If/Elsif construct involving complex decisions with computations

Exercise an If/Elsif construct involving complex decisions with computations and the use of a 'abs' operator in the Elsif part.

5.7.1 ada sources

- add.adb
- add.ads
- test_add_fx_tf.adb
- test_add_fx_tt.adb
- test_add_tf_fx.adb
- test_add_tt_xx.adb

5.7.2 c sources

5.7.3 h sources

5.8 Exercise a toplevel If statement with an else sequence embedding a For loop,

Exercise a toplevel If statement with an else sequence embedding a For loop, whose body features an inner If statement that controls access to a Case statement eventually.

We implement a double testing strategy, with:

 # Statement coverage oriented scenarii, exercising SC possibilities (loop entered or not, If part of an If statement reached or not, Case statement selections reached or not, ...) and combinations of such.

 # Decision coverage oriented scenarii, exercising DC possibilities (such or such decision True only, False only, or both True and False) and combinations of such.

5.8.1 ada sources

- sensors-status.adb
- sensors-status.ads
- test_broken.adb
- test_check.adb
- test_check_brok.adb
- test_f_ff.adb
- test_f_ft.adb
- test_f_fu.adb
- test_f_tx.adb
- test_fu_ff.adb
- test_fu_ft.adb
- test_fu_fu.adb
- test_fu_tx.adb
- test_ok.adb
- test_ok_brok.adb
- test_ok_check.adb
- test_ok_check_brok.adb
- test_t_0.adb
- test_und_brok.adb
- test_und_check.adb
- test_und_ok.adb
- test_und_ok_brok.adb
- test_und_ok_check.adb
- test_undecide.adb

5.8.2 c sources

5.8.3 h sources

5.9 Exercise a simple If statement within a While loop.

Exercise a simple If statement within a While loop.

Check cases where

- The loop is not entered
- The loop is entered and the if control decision never evaluates True
- The loop is entered and the if control decision never evaluates False
- The loop is entered and the if control decision evaluates True and False

5.9.1 ada sources

- sensors-predicates.adb
- sensors-predicates.ads
- slists-count.adb
- slists-count.ads
- test_iff.adb
- test_ift.adb
- test_iftf.adb
- test_noentry.adb

5.9.2 c sources

5.9.3 h sources

5.10 Exercise a three way If statement within a While loop.

Exercise a three way If statement within a While loop.

Check cases where

- The loop is not entered
- The loop is entered and all the possible combinations of if controls are checked

5.10.1 ada sources

- doif_fx_ff.adb
- doif_fx_ft.adb
- doif_fx_tx.adb
- doif_tf_ff.adb
- doif_tf_ft.adb
- doif_tf_tx.adb

- doif_tt_xx.adb
- slists-fault.adb
- slists-fault.ads
- test_df_df.adb
- test_df_dt.adb
- test_df_fu.adb
- test_fu_dt.adb
- test_fu_fu.adb
- test_fx_ff.adb
- test_fx_ft.adb
- test_fx_tx.adb
- test_noentry.adb
- test_tf_ff.adb
- test_tf_ft.adb
- test_tf_tx.adb
- test_tt_xx.adb

5.10.2 c sources

5.10.3 h sources

5.11 Exercise a simple If statement nested in one of the possible selections of a

Exercise a simple If statement nested in one of the possible selections of a Case statement, itself conditioned by an outer If statement executed as part of a While loop.

We implement a double testing strategy, with:

- Statement coverage oriented scenarii, exercising SC possibilities (loop entered or not, If part of an If statement reached or not, Case statement selections reached or not, ...) and combinations of such.
- Decision coverage oriented scenarii, exercising DC possibilities (such or such decision True only, False only, or both True and False) and combinations of such.

5.11.1 ada sources

- slists-forall.adb
- slists-forall.ads
- test_act_inh.adb
- test_activate.adb
- test_f_0_0.adb
- test_fu_ff_0.adb
- test_fu_ft_0.adb
- test_fu_fu_0.adb

- test_fu_fu_ff.adb
- test_fu_fu_ft.adb
- test_fu_fu_fu.adb
- test_fu_fu_tx.adb
- test_fu_tx_0.adb
- test_fu_tx_ff.adb
- test_fu_tx_ft.adb
- test_fu_tx_tx.adb
- test_inhibit.adb
- test_nocase.adb
- test_noentry.adb
- test_noinhibit.adb

5.11.2 c sources

5.11.3 h sources

5.12 Check that no stmt coverage violations are reported for explicitly deactivated

Check that no stmt coverage violations are reported for explicitly deactivated sequences, in a subprogram or package elaboration body.

5.12.1 ada sources

- ctl.ads
- ops.adb
- ops.ads
- test_ops_0.adb
- test_ops_inc.adb

5.12.2 c sources

5.12.3 h sources

5.13 Check that Boolean expressions spanning multiple lines are handled correctly.

Check that Boolean expressions spanning multiple lines are handled correctly.

5.14 Check that the tool properly distinguishes independant simple nested

Check that the tool properly distinguishes independant simple nested constructs. Exercise a simple If statement, without an Else or Elsif part, nested within another one, with both Ifs controlled by independant conditions.

Check all the possible combinations of True-only/False-only/both valuations of the outer and inner conditions.

5.14.1 ada sources

- andthen.adb
- test_f_f.adb
- test_f_fu.adb
- test_f_t.adb
- test_fu_f.adb
- test_fu_fu.adb
- test_fu_t.adb
- test_t_f.adb
- test_t_fu.adb
- test_t_t.adb

5.14.2 c sources

5.14.3 h sources

5.15 Check the correctness of MCDC assessments on decisions that are not entirely evaluated (don't

Check the correctness of MCDC assessments on decisions that are not entirely evaluated (don't yield an outcome) even though the statement they control is executed, typically when evaluation of a sub-decision is short-circuited.

We verify that `decision never evaluated` diagnostics are emitted instead of statement coverage violations.

(*)	Chapter	Description
TC	**ShortCut**	*Check MCDC on A and then Call (B or else C), where the nested decision may be*

5.15.1 Check MCDC on `A and then Call (B or else C)`, where the nested decision may be

Check MCDC on `A and then Call (B or else C)`, where the nested decision may be unevaluated because of short-circuit semantics from A False at the outer level.

5.16 Check the correctness of MCDC assessments on decisions featuring all the possible forms

Check the correctness of MCDC assessments on decisions featuring all the possible forms of conditions, not only Boolean variables.

We exercise basic decisions involving various possible kinds of conditions (attribute references, subprogram calls, ...), either isolated or mixed with other kinds in a variety of program contexts. We make sure here to exercise cases known to require internal comparisons, to demonstrate that the tool knows to distinguish internal tests from those representative of each condition valuation.

(*)	Chapter	Description
TC	**Aggregates**	*Check MCDC on a simple expression with two operands involving comparison with*
TC	**Attributes**	*Check MCDC on simple expressions with one or two operands involving Ada attributes*
TC	**Components**	*Check MCDC on simple decisions over two operands featuring record component*
TC	**Concats**	*Check MCDC on simple expressions with one or two operands involving Ada string*
TC_Set	**Mixed**	*TITLE*
TC	. . AttrSlice	`/TOR_Doc/Ada/mcdc/1_Core/Operands/Mixed/AttrSlice`
TC	. . AttrSliceAggr	`/TOR_Doc/Ada/mcdc/1_Core/Operands/Mixed/AttrSliceAggr`
TC	**Mods**	*Check MCDC on simple expressions with one or two operands involving Ada string*
TC	**Ranges**	*Check MCDC on a simple expression with two operands involving a range test.*
TC	**Slices**	*Check MCDC on a simple decision over two operands featuring array slices*
TC	**Subtypes**	*Check MCDC on a simple decision where conditions are of a Boolean subtype or*

5.16.1 Check MCDC on a simple expression with two operands involving comparison with

Check MCDC on a simple expression with two operands involving comparison with an array aggregate.

5.16.2 Check MCDC on simple expressions with one or two operands involving Ada attributes

Check MCDC on simple expressions with one or two operands involving Ada attributes known to require internal computation with tests, such as String'Length.

5.16.3 Check MCDC on simple decisions over two operands featuring record component

Check MCDC on simple decisions over two operands featuring record component references.

5.16.4 Check MCDC on simple expressions with one or two operands involving Ada string

Check MCDC on simple expressions with one or two operands involving Ada string concatenations, known to require internal computations.

5.16.5 TITLE

5.16.6 Check MCDC on simple expressions with one or two operands involving Ada string

Check MCDC on simple expressions with one or two operands involving Ada string concatenations, known to require internal computations.

5.16.7 Check MCDC on a simple expression with two operands involving a range test.

Check MCDC on a simple expression with two operands involving a range test.

5.16.8 Check MCDC on a simple decision over two operands featuring array slices

Check MCDC on a simple decision over two operands featuring array slices

5.16.9 Check MCDC on a simple decision where conditions are of a Boolean subtype or

Check MCDC on a simple decision where conditions are of a Boolean subtype or derived type.

5.17 Check the correctness of MCDC assessments on expressions of arbitrary topology

Check the correctness of MCDC assessments on expressions of arbitrary topology everywhere they might appear.

We exercise a panel of expression topologies over Boolean variables (single value, negated, two or more combined with and-then or or-else, ...), each placed in a variety of program contexts. MCDC is equivalent to DC for decisions with only one condition so we have no testcase for such decisions here.

(*)	Chapter	Description
TC_Set	**And**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Index
TC_Set	.. Pragmas	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Pragmas
TC	.. DebugBody	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Pragmas/DebugBody
TC	.. Mix	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Pragmas/Mix
TC	.. PreBody	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Pragmas/PreBody
TC	.. PreDecl	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Pragmas/PreDecl
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/And/Xreturn

Continued on next page

Table 5.2 – continued from previous page

TC_Set	**AndCOr**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndCOr/Xreturn
TC_Set	**AndIdOr**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndIdOr/Xreturn
TC_Set	**AndNot**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndNot/Xreturn
TC_Set	**AndpOrp**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/AndpOrp/Xreturn

Continued on next page

Table 5.2 – continued from previous page

TC_Set	**Coupled**	*Check proper behavior in*
TC	.. Obvious	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Coupled/Obvious
TC	.. cc4	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Coupled/cc4
TC	.. cc6	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Coupled/cc6
TC_Set	**Diamonds**	*Check MCDC on decisions with diamonds in*
TC	.. c3	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Diamonds/c3
TC_Set	**NotAnd**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAnd/Xreturn
TC_Set	**NotAndNot**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotAndNot/Xreturn
TC_Set	**NotOr**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOr/Xreturn
TC_Set	**NotOrNot**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Exit

Continued on next page

Table 5.2 – continued from previous page

TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/NotOrNot/Xreturn
TC_Set	**Or**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Index
TC_Set	.. Pragmas	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Pragmas
TC	.. DebugBody	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Pragmas/DebugBody
TC	.. PreBody	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Pragmas/PreBody
TC	.. PreDecl	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Pragmas/PreDecl
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/Or/Xreturn
TC_Set	**OrNot**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/OrNot/Xreturn
TC_Set	**pAndpOr**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Assignment
TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pAndpOr/Xreturn
TC_Set	**pOrpAndpOrp**	*Check MCDC on a decision of the form*
TC	.. Actual	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Actual
TC	.. Aggregate	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Aggregate
TC	.. Assignment	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Assignment

Continued on next page

Table 5.2 – continued from previous page

TC	.. Case	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Case
TC	.. DeclInit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/DeclInit
TC	.. Discriminant	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Discriminant
TC	.. Exit	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Exit
TC	.. For	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/For
TC	.. If	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/If
TC	.. Index	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Index
TC	.. Return	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Return
TC	.. While	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/While
TC	.. Xreturn	/TOR_Doc/Ada/mcdc/1_Core/Topologies/pOrpAndpOrp/Xreturn

5.17.1 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A and then B` in various contexts.

5.17.2 Check MCDC on a decision of the form

5.17.3 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A and then OrElse (C, B)` in various contexts.

5.17.4 Check MCDC on a decision of the form

5.17.5 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A and then Identity (B or else C)` in various contexts.

5.17.6 Check MCDC on a decision of the form

5.17.7 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A and then (not B)` in various contexts.

5.17.8 Check MCDC on a decision of the form

5.17.9 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A and then (B or else C)` in various contexts.

5.17.10 Check proper behavior in

Check proper behavior in various situations involving coupled conditions.

5.17.11 Check MCDC on decisions with diamonds in

Check MCDC on decisions with diamonds in the control-flow binary decision diagram, representative of situations where object branch coverage does not imply mcdc.

5.17.12 Check MCDC on a decision of the form

Check MCDC on a decision of the form `(not A) and then B` in various contexts.

5.17.13 Check MCDC on a decision of the form

5.17.14 Check MCDC on a decision of the form

Check MCDC on a decision of the form `(not A) and then (not B)` in various contexts.

5.17.15 Check MCDC on a decision of the form

Check MCDC on a decision of the form `(not A) or else B` in various contexts.

5.17.16 Check MCDC on a decision of the form

5.17.17 Check MCDC on a decision of the form

Check MCDC on a decision of the form `(not A) or else (not B)` in various contexts.

5.17.18 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A or else B` in various contexts.

5.17.19 Check MCDC on a decision of the form

5.17.20 Check MCDC on a decision of the form

Check MCDC on a decision of the form `A or else (not B)` in various contexts.

5.17.21 Check MCDC on a decision of the form

Check MCDC on a decision of the form `(A and then B) or else C` in various contexts.

5.17.22 Check MCDC on a decision of the form

Check MCDC on a decision of the form `(A or else B) and then (C or else D)` in various contexts.

5.18 Exercise consolidation over `A and then B`.

Exercise consolidation over `A and then B`.

5.18.1 ada sources

- andthen.adb
- andthen.ads
- test_andthen_v0.adb
- test_andthen_v1.adb
- test_andthen_v2.adb
- test_andthen_v3.adb

5.18.2 c sources

5.18.3 h sources

5.19 Exercise consolidation over `Andthen (X => A and then B, Y => C or else D),`

5.20 Exercise consolidation over `Andthen (X => A and then B, Y => C or else D),`

Exercise consolidation over `Andthen (X => A and then B, Y => C or else D)`, with an `X and then Y` evaluation within the Andthen function.

Arrange to have four distinct executions corresponding to the possible values of each primary decision (True/False for A and then B together with True/False for C or else D), and verify that consolidating any two of these executions together yields the expected results for each decision.

5.20.1 ada sources

- a1o2.adb
- a1o2.ads
- test_a1o2_ff.adb
- test_a1o2_ft.adb
- test_a1o2_tf.adb
- test_a1o2_tt.adb

5.20.2 c sources

5.20.3 h sources

5.21 Exercise consolidation over `A or else B`.

Exercise consolidation over `A or else B`.

5.21.1 ada sources

- orelse.adb
- orelse.ads
- test_orelse_v0.adb
- test_orelse_v1.adb
- test_orelse_v2.adb
- test_orelse_v3.adb

5.21.2 c sources

5.21.3 h sources

5.22 Exercise consolidation over `(A or else B) and then C`.

Exercise consolidation over `(A or else B) and then C`.

5.22.1 ada sources

- porpand.adb
- porpand.ads
- test_porpand_v1.adb
- test_porpand_v2.adb
- test_porpand_v3.adb
- test_porpand_v4.adb
- test_porpand_v5.adb

5.22.2 c sources

5.22.3 h sources

5.23 Exercise the use of an exemption region to exempt a functional precondition

Exercise the use of an exemption region to exempt a functional precondition expressed in a subprogram declarative part.

Check outcome expectations for

- situations where the precondition is always evaluated True (nominal case, decision violations exempted)
- situations where the precondition is evaluated only False (force error case, decision violations exempted)
- consolidation of the previous cases with precondition exercized both ways and demonstration of independant effect missing for at least one conditions (mcdc violations exempted)
- consolidation of the previous cases with precondition exercized both ways and demonstration of independant effect for all the conditions (no exempted violation)

5.23.1 ada sources

- ranges.adb
- ranges.ads
- test_ranges_invalid_a.adb
- test_ranges_invalid_b.adb
- test_ranges_no_overlap.adb
- test_ranges_overlap.adb

5.23.2 c sources

5.23.3 h sources

5.24 Exercise an If/Elsif construct involving complex decisions with computations

Exercise an If/Elsif construct involving complex decisions with computations and the use of a 'abs' operator in the Elsif part.

5.24.1 ada sources

- add.adb
- add.ads
- test_add_a_d.adb
- test_add_a_xx.adb
- test_add_ab_c.adb
- test_add_ab_cd.adb
- test_add_ab_xx.adb
- test_add_b_xx.adb
- test_add_fx_tf.adb
- test_add_fx_tt.adb
- test_add_tf_fx.adb
- test_add_tt_xx.adb
- test_add_xx_c.adb
- test_add_xx_cd.adb
- test_add_xx_d.adb

5.24.2 c sources

5.24.3 h sources

5.25 Exercise a toplevel If statement with an else sequence embedding a For loop,

Exercise a toplevel If statement with an else sequence embedding a For loop, whose body features an inner If statement that controls access to a Case statement eventually.

We implement a double testing strategy, with:

 # Statement coverage oriented scenarii, exercising SC possibilities (loop entered or not, If part of an If statement reached or not, Case statement selections reached or not, ...) and combinations of such.

 # Decision coverage oriented scenarii, exercising DC possibilities (such or such decision True only, False only, or both True and False) and combinations of such.

5.25.1 ada sources

- sensors-status.adb
- sensors-status.ads
- test_broken.adb
- test_check.adb
- test_check_brok.adb
- test_f_ff.adb
- test_f_ft.adb
- test_f_fu.adb
- test_f_tx.adb
- test_fu_ff.adb
- test_fu_ft.adb
- test_fu_fu.adb
- test_fu_tx.adb
- test_ok.adb
- test_ok_brok.adb
- test_ok_check.adb
- test_ok_check_brok.adb
- test_t_0.adb
- test_und_brok.adb
- test_und_check.adb
- test_und_ok.adb
- test_und_ok_brok.adb
- test_und_ok_check.adb
- test_undecide.adb

5.25.2 c sources

5.25.3 h sources

5.26 Exercise a three way If statement within a While loop.

Exercise a three way If statement within a While loop.

Check cases where

- The loop is not entered
- The loop is entered and all the possible combinations of if controls are checked

5.26.1 ada sources

- doif_fx_ff.adb
- doif_fx_ft.adb
- doif_fx_tx.adb
- doif_tf_ff.adb
- doif_tf_ft.adb
- doif_tf_tx.adb
- doif_tt_xx.adb
- slists-fault.adb
- slists-fault.ads
- test_df_df.adb
- test_df_dt.adb
- test_df_fu.adb
- test_fu_dt.adb
- test_fu_fu.adb
- test_fx_ff.adb
- test_fx_ft.adb
- test_fx_tx.adb
- test_noentry.adb
- test_tf_ff.adb
- test_tf_ft.adb
- test_tf_tx.adb
- test_tt_xx.adb

5.26.2 c sources

5.26.3 h sources

5.27 Exercise a simple If statement nested in one of the possible selections of a

Exercise a simple If statement nested in one of the possible selections of a Case statement, itself conditioned by an outer If statement executed as part of a While loop.

We implement a double testing strategy, with:

- Statement coverage oriented scenarii, exercising SC possibilities (loop entered or not, If part of an If statement reached or not, Case statement selections reached or not, ...) and combinations of such.
- Decision coverage oriented scenarii, exercising DC possibilities (such or such decision True only, False only, or both True and False) and combinations of such.

5.27.1 ada sources

- slists-forall.adb
- slists-forall.ads
- test_act_inh.adb
- test_activate.adb
- test_f_0_0.adb
- test_fu_a_a.adb
- test_fu_a_ff.adb
- test_fu_a_ft.adb
- test_fu_a_tx.adb
- test_fu_b_0.adb
- test_fu_ff_0.adb
- test_fu_ft_0.adb
- test_fu_tx_0.adb
- test_fu_tx_ff.adb
- test_fu_tx_ft.adb
- test_fu_tx_tx.adb
- test_inhibit.adb
- test_nocase.adb
- test_noentry.adb
- test_noinhibit.adb

5.27.2 c sources

5.27.3 h sources

5.28 Check that no stmt coverage violations are reported for explicitly deactivated

Check that no stmt coverage violations are reported for explicitly deactivated sequences, in a subprogram or package elaboration body.

5.28.1 ada sources

- ctl.ads
- ops.adb
- ops.ads
- test_ops_0.adb
- test_ops_inc.adb

5.28.2 c sources

5.28.3 h sources

5.29 Check that when multiple decisions occur on the same source line, individual

Check that when multiple decisions occur on the same source line, individual decisions are properly recognized.

5.30 Check that decisions spanning multiple lines are recognized and processed

Check that decisions spanning multiple lines are recognized and processed correctly.

5.31 Check that recursive decision evaluations are processed correctly.

Check that recursive decision evaluations are processed correctly.

5.32 Check SC of active pragmas

Check SC of active pragmas

Check that active pragmas which are not elaborated are reported as uncovered. Exercise Assert, Debug, Precondition and Postcondition pragmas placed in various source contexts.

5.32.1 ada sources

- pragmas.adb
- pragmas.ads
- support_pragmas.adb
- support_pragmas.ads
- test_pragmas_assert_debug.adb
- test_pragmas_full.adb
- test_pragmas_no.adb
- test_pragmas_pre_post.adb

5.32.2 c sources

5.32.3 h sources

5.33 Check SC of inactive pragmas

Check SC of inactive pragmas

Check that inactive pragmas are ignored; Place Assert pragmas at various locations in a couple of subprogram bodies, and verify that they are never reported uncovered, even when the subprograms aren't even called.

5.33.1 ada sources

- asserts.adb
- asserts.ads
- test_asserts_0.adb
- test_asserts_t.adb

5.33.2 c sources

5.33.3 h sources

5.34 Check SC of Discrete Subtype Definitions

Check SC of Discrete Subtype Definitions

Check that a declaration that contains a discrete subtype definition is reported as uncovered iif the declaration is not elaborated. Exercise cases where the subtype definition is the only statement that can be covered.

5.34.1 ada sources

- discrete_subtype_defs.adb
- discrete_subtype_defs.ads
- test_discrete_subtype_defs_full.adb
- test_discrete_subtype_defs_no.adb

- test_discrete_subtype_defs_part_1.adb
- test_discrete_subtype_defs_part_2.adb

5.34.2 c sources

5.34.3 h sources

5.35 Check SC of Object Declarations involving heap or stack dynamic allocation

Check SC of Object Declarations involving heap or stack dynamic allocation

Check that an object declaration is reported as uncovered iif it is not elaborated. Exercise object declarations for which we expect to have execution of code from explicit initialization expressions involving

- dynamic heap allocation for standard “new” allocators
- dynamic stack allocation for functions returning unconstrained

Check local and global declarations.

5.35.1 ada sources

- heap.adb
- heap.ads
- test_heap_0.adb
- test_heap_all.adb
- test_heap_neg.adb
- test_heap_pos.adb
- test_vectors_0.adb
- test_vectors_neg.adb
- test_vectors_pos.adb
- vectors.adb
- vectors.ads

5.35.2 c sources

5.35.3 h sources

5.36 Check SC of Object Declarations involving static or fixed-sized stack

Check SC of Object Declarations involving static or fixed-sized stack allocations only.

Check that an object declaration is reported as uncovered iif it is not elaborated. Exercise object declarations for which we expect to have implicit execution of code for the initializations:

- Object declarations with explicit initialization expressions;

- Declarations of objects whose types define implicit initialization (access types and record types with component initializers).

Check local and global declarations. Check declarations that allow static or fixed-size stack allocations only, which don't need secondary stack (e.g. from functions returning unconstrained) or dynamic memory allocation (e.g. from regular "new" allocators).

5.36.1 ada sources

- access_swap.adb
- decls_pack_1.adb
- decls_pack_1.ads
- decls_pack_2.adb
- decls_pack_2.ads
- integer_swap.adb
- matrix_swap.adb
- private_swap.adb
- record_derived_swap.adb
- record_impl_init_swap.adb
- record_swap.adb
- test_object_declarations_full.adb
- test_object_declarations_no.adb
- test_object_declarations_part_1.adb
- test_object_declarations_part_2.adb

5.36.2 c sources

5.36.3 h sources

5.37 Check SC of Subtype Indications

Check SC of Subtype Indications

Check that a declaration that contains a subtype indication is reported as uncovered iif the declaration is not elaborated. Exercise cases where the subtype indication is the only statement to cover in subtype, object or component declarations.

5.37.1 ada sources

- subtype_indications.adb
- subtype_indications.ads
- test_subtype_indications_full.adb
- test_subtype_indications_no.adb
- test_subtype_indications_part_1.adb
- test_subtype_indications_part_2.adb

5.37.2 c sources

5.37.3 h sources

5.38 Check SC related to Variant Parts in record type declarations

Check SC related to Variant Parts in record type declarations

The optional variant part of a record type declaration works like an implicit case statement. The choice of a specific variant for an object triggers the evaluation of the corresponding initialization expressions (if any), which, via subprogram calls, might execute code that does not have an explicit connection with the place of the object declaration.

Exercise variant record declarations for which distinct variants trigger execution of distinct sequences of statements and check that only the code associated to the chosen variants is reported as covered.

5.38.1 ada sources

- check_variants.adb
- check_variants.ads
- test_variants_full.adb
- test_variants_no.adb
- test_variants_part_big.adb
- test_variants_part_small.adb
- variant_3_g.adb
- variant_3_g.ads
- variant_others_g.adb
- variant_others_g.ads
- variants.ads
- variants_support.adb
- variants_support.ads

5.38.2 c sources

5.38.3 h sources

5.39 Check SC of Assignment statements (ARM 5.1)

Check SC of Assignment statements (ARM 5.1)

Exercise assignments in various source contexts (within procedure, function or package elaboration bodies, within conditional control or nor), in sequence or single.

5.39.1 ada sources

- assignment_statements.adb
- assignment_statements.ads
- assignment_statements_elab.adb

- assignment_statements_elab.ads
- test_assignment_statements_full.adb
- test_assignment_statements_no.adb
- test_assignment_statements_part.adb

5.39.2 c sources

5.39.3 h sources

5.40 Check SC of Null statements (ARM 5.2)

Check SC of Null statements (ARM 5.2)

Check various code fragments that contain null statements such as:

- case statement alternative;
- null procedure;
- last statement in a sequence, with a label attached to it used to skip earlier statements in the sequence.

Check that only those null statements that are not executed are reported as uncovered.

5.40.1 ada sources

- null_statements.adb
- null_statements.ads
- test_null_statements_full.adb
- test_null_statements_no.adb
- test_null_statements_part.adb

5.40.2 c sources

5.40.3 h sources

5.41 Check SC of loop Exit statements (ARM 5.7)

Check SC of loop Exit statements (ARM 5.7)

Check that Exit statements are reported covered when execution reaches them, or uncovered otherwise. Exercise exit statements in various sorts of loops (loop, while, for) alone or combined with other exits for the same loop.

5.41.1 ada sources

- exit_statements.adb
- exit_statements.ads
- exit_statements_support.ads
- test_exit_statements_first_exit.adb
- test_exit_statements_full.adb

- test_exit_statements_no.adb
- test_exit_statements_no_exit.adb
- test_exit_statements_second_exit.adb

5.41.2 c sources

5.41.3 h sources

5.42 Check SC with Goto statements (ARM 5.8)

Check SC with Goto statements (ARM 5.8)

Check that Exit statements are reported covered when execution reaches them, or uncovered otherwise, and that their control-flow effects are handled correctly.

Exercise backward and forward transfers of control within:

- straight sequences of statements
- block, case, if or loop statements

5.42.1 ada sources

- goto_statements_block.adb
- goto_statements_block.ads
- goto_statements_case.adb
- goto_statements_case.ads
- goto_statements_if.adb
- goto_statements_if.ads
- goto_statements_loop.adb
- goto_statements_loop.ads
- goto_statements_straight.adb
- goto_statements_straight.ads
- test_goto_statements_block_1.adb
- test_goto_statements_block_2.adb
- test_goto_statements_block_no.adb
- test_goto_statements_case_11.adb
- test_goto_statements_case_12.adb
- test_goto_statements_case_2.adb
- test_goto_statements_case_full.adb
- test_goto_statements_case_no.adb
- test_goto_statements_case_others.adb
- test_goto_statements_if_else.adb
- test_goto_statements_if_elsif.adb
- test_goto_statements_if_full.adb

- test_goto_statements_if_if1.adb
- test_goto_statements_if_if2.adb
- test_goto_statements_if_no.adb
- test_goto_statements_loop_1.adb
- test_goto_statements_loop_2.adb
- test_goto_statements_loop_full.adb
- test_goto_statements_loop_no.adb
- test_goto_statements_loop_no_iteration.adb
- test_goto_statements_straight_1.adb
- test_goto_statements_straight_2.adb
- test_goto_statements_straight_3.adb
- test_goto_statements_straight_full.adb
- test_goto_statements_straight_no.adb

5.42.2 c sources

5.42.3 h sources

5.43 Check SC with IF statements taking [implicit] ELSE branches only.

Check SC with IF statements taking [implicit] ELSE branches only.

5.43.1 ada sources

- test_else_path.adb

5.43.2 c sources

5.43.3 h sources

5.44 Check SC with IF statements taking ELSIF branches only.

Check SC with IF statements taking ELSIF branches only.

Exercise IF constructs with one or more ELSIF branches. Check results on taking either the first one, the last one, or one in the middle.

5.44.1 ada sources

- test_elsif_path_first.adb
- test_elsif_path_last.adb
- test_elsif_path_middle.adb

5.44.2 c sources

5.44.3 h sources

5.45 Check SC with IF statements taking IF branches only.

Check SC with IF statements taking IF branches only.

5.45.1 ada sources

- test_if_path.adb

5.45.2 c sources

5.45.3 h sources

5.46 Check SC with IF statements taking all their branches.

Check SC with IF statements taking all their branches.

5.46.1 ada sources

- test_multiple_paths.adb

5.46.2 c sources

5.46.3 h sources

5.47 Check SC with IF statements not executed at all.

Check SC with IF statements not executed at all.

5.47.1 ada sources

- test_no_execution.adb

5.47.2 c sources

5.47.3 h sources

5.48 Check SC with CASE statements executed several times, selecting

Check SC with CASE statements executed several times, selecting different alternatives for each

5.48.1 ada sources

- test_several_alternatives_full.adb
- test_several_alternatives_no_others.adb
- test_several_alternatives_others.adb

5.48.2 c sources

5.48.3 h sources

5.49 Check SC with CASE statements not executed at all

Check SC with CASE statements not executed at all

5.49.1 ada sources

- test_no_execution.adb

5.49.2 c sources

5.49.3 h sources

5.50 Check SC with CASE statements executed only once, selecting a single

Check SC with CASE statements executed only once, selecting a single alternative for each

5.50.1 ada sources

- test_one_alternative.adb

5.50.2 c sources

5.50.3 h sources

5.51 Check SC with LOOP statements that execute at least one full

Check SC with LOOP statements that execute at least one full iteration over the nested sequence

5.51.1 ada sources

- test_full_iteration.adb

5.51.2 c sources

5.51.3 h sources

5.52 Check SC with LOOP statements that execute only a partial iteration

Check SC with LOOP statements that execute only a partial iteration over the nested sequence, interrupted by an exit statement

Check, in particular, that the nested statements past the exit point are reported uncovered.

5.52.1 ada sources

- test_incomplete_iteration.adb

5.52.2 c sources

5.52.3 h sources

5.53 Check SC with LOOP statements not executed at all

Check SC with LOOP statements not executed at all

Check, in particular, that the loop headers and all the nested statements are reported uncovered.

5.53.1 ada sources

- test_no_execution.adb

5.53.2 c sources

5.53.3 h sources

5.54 Check SC with LOOP statements that don't even start an iteration over

Check SC with LOOP statements that don't even start an iteration over the nested sequence, guarded out by the header control expression

5.54.1 ada sources

- test_no_iteration.adb

5.54.2 c sources

5.54.3 h sources

5.55 Check SC with nested block statements

Check SC with nested block statements

5.55.1 ada sources

- nested_block_statements.adb
- test_nested_block_statements_inner.adb
- test_nested_block_statements_middle.adb
- test_nested_block_statements_no.adb
- test_nested_block_statements_outer.adb

5.55.2 c sources

5.55.3 h sources

5.56 Check SC with block statements with/without declarative part or exception

Check SC with block statements with/without declarative part or exception handlers, containing or nested-within a variety of language constructs

Check, in particular, blocks located within compound statements, part of regular or instanciated subprograms, and that contain For loops, If constructs or exception handlers.

5.56.1 ada sources

- block_statements.adb
- block_statements.ads
- instances.ads
- test_block_statements_blocks.adb
- test_block_statements_full.adb
- test_block_statements_no.adb
- test_block_statements_no_exec.adb

5.56.2 c sources

5.56.3 h sources

5.57 Check SC on a simple subprogram called from a wide range of source contexts

Check SC on a simple subprogram called from a wide range of source contexts

(*)	Chapter	Description
TC	**Actual**	*Check SC on a simple subprogram called from a subprogram actual parameter*
TC	**CompInit**	*Check SC on a simple subprogram called from a record default component initialization*
TC	**DeclInit**	*Check SC on a simple subprogram called from a local object initialization*
TC	**FormalDef**	*Check SC on a simple subprogram called as part of a subprogram formal default*
TC	**LocalElab**	*Check SC on a simple subprogram called from a local package elaboration body*
TC	**ProcBody**	*Check SC on a simple subprogram called from a procedure body*

5.57.1 Check SC on a simple subprogram called from a subprogram actual parameter

Check SC on a simple subprogram called from a subprogram actual parameter evaluation

5.57.2 Check SC on a simple subprogram called from a record default component initialization

Check SC on a simple subprogram called from a record default component initialization

5.57.3 Check SC on a simple subprogram called from a local object initialization

Check SC on a simple subprogram called from a local object initialization

5.57.4 Check SC on a simple subprogram called as part of a subprogram formal default

Check SC on a simple subprogram called as part of a subprogram formal default value construction

5.57.5 Check SC on a simple subprogram called from a local package elaboration body

Check SC on a simple subprogram called from a local package elaboration body

5.57.6 Check SC on a simple subprogram called from a procedure body

Check SC on a simple subprogram called from a procedure body

5.58 Check SC within a panel of library level subprograms reached through

Check SC within a panel of library level subprograms reached through a variety of call mechanisms

Check:

- Library-level functions and procedures;
- Subprograms declared in library package specs, in the visible or the private part;
- Subprograms declared at the toplevel in a library package body.

Reached by way of:

- Ordinary calls by means of a call statement/function call expression;
- Dynamic calls through access-to-subprogram value;
- Implicit calls as the result of evaluation of default initialization expressions.

5.58.1 ada sources

- library_level_fun.adb
- library_level_fun.ads
- library_level_proc.adb
- library_level_proc.ads
- subprogram_pack.adb
- subprogram_pack.ads
- test_subprogram_decls_default_init.adb
- test_subprogram_decls_full.adb
- test_subprogram_decls_indirect_calls.adb
- test_subprogram_decls_no.adb
- test_subprogram_decls_ordinary_calls.adb

5.58.2 c sources

5.58.3 h sources

5.59 Check SC within a panel of nested subprograms reached through a variety of

Check SC within a panel of nested subprograms reached through a variety of call mechanisms

Check:

- Functions and procedures nested within the declarative part of an outer subprogram body, ...
- Either directly or via an intermediate local package

Reached by way of direct calls or indirect through record component initializations.

5.59.1 ada sources

- nest.adb
- nest.ads
- test_nest_dlf.adb
- test_nest_dlp.adb
- test_nest_dpf.adb
- test_nest_dpp.adb
- test_nest_ilf.adb
- test_nest_ilf_dlf.adb
- test_nest_ilf_dlp.adb
- test_nest_ilf_dpf.adb
- test_nest_ilf_dpp.adb
- test_nest_ilfpf.adb
- test_nest_ipf.adb
- test_nest_ipf_dlf.adb
- test_nest_ipf_dlp.adb
- test_nest_ipf_dpf.adb
- test_nest_ipf_dpp.adb

5.59.2 c sources

5.59.3 h sources

5.60 Check SC with simple subprogram overridings

Check SC with simple subprogram overridings

5.60.1 ada sources

- notes.adb
- notes.ads
- test_notes_0.adb
- test_notes_all.adb
- test_notes_df.adb
- test_notes_dt.adb

5.60.2 c sources

5.60.3 h sources

5.61 Check SC with complex overridings for tagged types

Check SC with complex overridings for tagged types

Check multiple levels of inheritance and different kinds of subprogram invocation:

- Ordinary calls by means of a call statement/function call expression;
- Dynamic calls through OO dispatching or access-to-subprogram value;
- Implicit calls as the result of evaluation of default initialization expressions;

5.61.1 ada sources

- derived_1.adb
- derived_1.ads
- derived_2.adb
- derived_2.ads
- subprogram_pack.adb
- subprogram_pack.ads
- test_derived_subprograms_full.adb
- test_derived_subprograms_indirect_1.adb
- test_derived_subprograms_indirect_2.adb
- test_derived_subprograms_no.adb
- test_derived_subprograms_part.adb

5.61.2 c sources

5.61.3 h sources

5.62 Check SC with multi-level Inlining, where subprogram S3 is inlined in S2, in

Check SC with multi-level Inlining, where subprogram S3 is inlined in S2, in turn inlined in S1.

5.62.1 ada sources

- inlined_subprograms.adb
- inlined_subprograms.ads
- test_inlined_subprograms_full.adb
- test_inlined_subprograms_no.adb
- test_inlined_subprograms_part.adb

5.62.2 c sources

5.62.3 h sources

5.63 Check SC with a subprogram inlined in two other ones, each exercising

Check SC with a subprogram inlined in two other ones, each exercising a distinct part of the inlined body.

5.63.1 ada sources

- ops.adb
- ops.ads
- test_ops_0.adb
- test_ops_down.adb
- test_ops_fu.adb
- test_ops_up.adb

5.63.2 c sources

5.63.3 h sources

5.64 Check SC with return statements in Function

Check SC with return statements in Function

5.64.1 ada sources

- fret.adb
- fret.ads
- test_fret_full.adb
- test_fret_no.adb
- test_fret_part.adb

5.64.2 c sources

5.64.3 h sources

5.65 Check SC with return statements in Procedures

Check SC with return statements in Procedures

5.65.1 ada sources

- pret.adb
- pret.ads
- test_pret_full.adb
- test_pret_no.adb
- test_pret_part.adb

5.65.2 c sources

5.65.3 h sources

5.66 Check the correctness of SC assessments for subprograms declared in various package regions

Check the correctness of SC assessments for subprograms declared in various package regions

In particular, check:

- subprogams declared in the public or private private of a package spec,
- subprograms declared in a package body only

5.66.1 ada sources

- ops.adb
- ops.ads
- test_ops_0.adb
- test_ops_full.adb
- test_ops_internal.adb
- test_ops_private.adb

5.66.2 c sources

5.66.3 h sources

5.67 Check SC on package elaboration statements

Check SC on package elaboration statements

Check proper coverage assessment of elaboration statements in both local and library level packages. Check that the only situation when code from a package body is reported uncovered is for a local package declared in a subprogram body that is not called.

5.67.1 ada sources

- library_level_proc.adb
- pack_1.adb
- pack_1.ads
- pack_2.adb
- pack_2.ads
- test_packages_full.adb
- test_packages_no.adb
- test_packages_part_1.adb
- test_packages_part_2.adb

5.67.2 c sources

5.67.3 h sources

5.68 Check that object renamings are handled correctly

Check that object renamings are handled correctly

Exercise various source constructs containing object renamings. Verify that they are recognized and processed as regular object declarations.

5.68.1 ada sources

- libray_level_renamings.ads
- local_renamings.adb
- local_renamings.ads
- renamed_objects.adb
- renamed_objects.ads
- test_object_renamings_all.adb
- test_object_renamings_no.adb
- test_object_renamings_part_1.adb
- test_object_renamings_part_2.adb

5.68.2 c sources

5.68.3 h sources

5.69 Check that package or subprogram renamings are handled correctly

Check that package or subprogram renamings are handled correctly

Exercise procedure, function and package renaming declarations, both local and library-level.

5.69.1 ada sources

- lib_level_fun.adb
- lib_level_fun_renaming.ads
- lib_level_proc.adb
- lib_level_proc_renaming.ads
- pack.adb
- pack.ads
- renaming_pack.ads
- test_subprogram_renamings_full.adb
- test_subprogram_renamings_no.adb
- test_subprogram_renamings_part.adb

5.69.2 c sources

5.69.3 h sources

5.70 Exercise a mix of subprogram and package subunits, with multiple levels

Exercise a mix of subprogram and package subunits, with multiple levels of nesting.

5.70.1 ada sources

- pack-inner-fun-proc.adb
- pack-inner-fun.adb
- pack-inner.adb
- pack-new_value.adb
- pack-update.adb
- pack.adb
- pack.ads
- test_subunits_full.adb
- test_subunits_no.adb
- test_subunits_part.adb

5.70.2 c sources

5.70.3 h sources

5.71 Exercise a package subunits declared in various source contexts.

Exercise a package subunits declared in various source contexts.

5.71.1 ada sources

- ops-isub.adb
- ops-lsub.adb
- ops-psub.adb
- ops-vsub.adb
- ops.adb
- ops.ads

5.71.2 c sources

5.71.3 h sources

5.72 Exercise subprogram subunits declared in various source contexts.

Exercise subprogram subunits declared in various source contexts.

5.72.1 ada sources

- ops-isub.adb
- ops-psub.adb
- ops-vsub.adb
- ops.adb
- ops.ads

5.72.2 c sources

5.72.3 h sources

5.73 Check SC with expressions that can fail to evaluate because of exceptions

Check SC with expressions that can fail to evaluate because of exceptions

Check the effects of an expression evaluation interruption by an exception raise, for a set of expressions and a set of possible exception origins, as part of a simple statement or of the control expression in compound statements.

For every variant, check a variety of situations where

- No exception gets raised at all
- An exception gets raised as part of all the evaluation attempts, on the first or the last operand evaluation,
- An exception would be raised by one operand but is not because of the shortcircuit semantics,
- An exception is raised by some evaluation but not others

(*)	Chapter	Description
		Continued on

Table 5.3 – continued from previous page

TC_Set	**And**	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/If/RemoteChec
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return/IndexC
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return/RangeC
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/And/Return/Remote
TC_Set	**Flip**	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If/IndexChec
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If/RangeChec
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/If/RemoteChe
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return/Index
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return/Range
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Flip/Return/Remot
TC_Set	**Or**	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If/IndexCheck
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If/RangeCheck
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/If/RemoteCheck
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return/IndexCh
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return/RangeCh
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Or/Return/RemoteC
TC_Set	**Value**	*TITLE*
TC_Set	.. If	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If/IndexChe
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If/RangeChe
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/If/RemoteCh
TC_Set	.. Return	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return
TC	.. IndexCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return/Inde
TC	.. RangeCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return/Rang
TC	.. RemoteCheck	/TOR_Doc/Ada/stmt/1_Core/11_Exceptions/CutEvals/Value/Return/Remo

5.73.1 TITLE

5.73.2 TITLE

5.73.3 TITLE

5.73.4 TITLE

5.74 Check SC correct recognition of alternate exception handlers for user

Check SC correct recognition of alternate exception handlers for user defined or language standard exceptions

Exercise conditional explicit raise of user defined exception or implicit raise of predefined exception, covered by distinct or common handlers. Check that

- Exception handlers are all reported as uncovered when no exception is raised,

- Only the statements associated to the correct handler are reported as covered when an exception is raised,

Check exception handlers with explicit exception names and including the “others” choice.

5.74.1 ada sources

- pack.adb
- pack.ads
- test_handlers_all_exception_raise.adb
- test_handlers_no.adb
- test_handlers_no_exception_raise.adb
- test_handlers_predefined_exception_raise.adb
- test_handlers_user_exception_raise.adb

5.74.2 c sources

5.74.3 h sources

5.75 Check SC of sequences potentially skipped by exceptions that propagate through

Check SC of sequences potentially skipped by exceptions that propagate through multiple levels of block nesting within a subprogram body.

Cases to check are restricted by ZFP limitation on exception propagation, so the only case to check is nested block statements.

5.75.1 ada sources

- pack.adb
- pack.ads
- test_propagation_blocks_1.adb
- test_propagation_blocks_2.adb
- test_propagation_blocks_ce.adb
- test_propagation_blocks_multiple.adb
- test_propagation_blocks_no.adb
- test_propagation_blocks_no_exception.adb

5.75.2 c sources

5.75.3 h sources

5.76 Check SC with multiple kinds of raise operations, implicit or explicit,

Check SC with multiple kinds of raise operations, implicit or explicit, for language or user defined exceptions in various program contexts.

Cases to check are restricted by ZFP limitation on exception propagation, intra subprogram only.

5.76.1 ada sources

- pack.adb
- pack.ads
- test_exceptions_all_exception_raise.adb
- test_exceptions_full.adb
- test_exceptions_no.adb
- test_exceptions_no_exception_raise.adb
- test_exceptions_predefined_exception_raise.adb
- test_exceptions_user_exception_raise.adb

5.76.2 c sources

5.76.3 h sources

5.77 Check that in case of a library-level instantiation the code of the generic

Check that in case of a library-level instantiation the code of the generic unit is not reported as covered if the instantiation is not executed or elaborated. Check that if a generic unit is not instantiated then its code is not reported as covered. This test case does not check that the code of generic unit is reported as uncovered if the unit is not instantiated or if the instantiation is not executed/elaborated, because for unused generics in some cases no coverage information is generated.

5.77.1 ada sources

- new_value.ads
- pack_instance_lib_level.ads
- stacks.ads
- test_instantiations_elab_only.adb
- test_instantiations_full.adb
- test_instantiations_no.adb
- test_instantiations_part_1.adb
- test_instantiations_part_2.adb

- update.ads

5.77.2 c sources

5.77.3 h sources

5.78 Check that in case of a local instantiation the code of the generic unit is

Check that in case of a local instantiation the code of the generic unit is not reported as covered if the instantiation is not executed or elaborated. Check that if a generic unit is not instantiated then its code is not reported as covered. This test case does not check that the code of generic unit is reported as uncovered if the unit is not instantiated or if the instantiation is not executed/elaborated, because for unused generics in some cases no coverage information is generated.

5.78.1 ada sources

- local_instantiations.adb
- local_instantiations.ads
- test_instantiations_elab_only.adb
- test_instantiations_full.adb
- test_instantiations_part_1.adb
- test_instantiations_part_2.adb
- test_instantiations_part_3.adb
- test_instantiations_part_4.adb
- test_instantiations_part_5.adb
- test_instantiations_part_6.adb

5.78.2 c sources

5.78.3 h sources

5.79 Exercise a three way if statement with three different

Exercise a three way if statement with three different drivers, each running into one of the three possible cases.

Verify that

- Each individual case reports the other branches uncovered,
- The combination of all the cases reports nothing uncovered,
- The combination of two cases reports only the third piece uncovered.

5.79.1 ada sources

- ranges.adb
- ranges.ads

- test_ranges_inrange.adb
- test_ranges_outmax.adb
- test_ranges_outmin.adb

5.79.2 c sources

5.79.3 h sources

5.80 Check that basic blocks of statements are recognized in various possible

Check that basic blocks of statements are recognized in various possible contexts.

(*)	Chapter	Description
TC	**ElabBody**	*Check that statements are recognized in package elaboration bodies and*
TC	**Subprograms**	*Check that statements are recognized in subprogram bodies and nested blocks*

5.80.1 Check that statements are recognized in package elaboration bodies and

Check that statements are recognized in package elaboration bodies and nested blocks therein.

"with" a package known to feature straight elaboration code only and verify that nothing is reported uncovered for this body. Arrange to have a local block there, to check that nested statements are handled as well.

5.80.2 Check that statements are recognized in subprogram bodies and nested blocks

Check that statements are recognized in subprogram bodies and nested blocks therein.

Exercise several straight subprograms exposed by a package . Verify that statements in subprograms not called are reported uncovered, and that nothing is reported about statements in subprograms that are called. The subprograms contain local block declarations so proper handling of these is verified as well.

5.81 Check that various forms of control-flow transfers are handled properly.

Check that various forms of control-flow transfers are handled properly.

(*)	Chapter	Description
TC	**Goto**	*Exercise a function where flow control is achieved with goto*
TC	**LocalRaise**	*Exercise a function where flow control is achieved with*
TC	**LoopExit**	*Exercise a function where flow control is achieved with loop*
TC	**Return**	*Exercise a function where flow control is achieved with goto*

5.81.1 Exercise a function where flow control is achieved with goto

Exercise a function where flow control is achieved with goto statements to interrupt sequences.

5.81.2 Exercise a function where flow control is achieved with

Exercise a function where flow control is achieved with local exception processing.

5.81.3 Exercise a function where flow control is achieved with loop

Exercise a function where flow control is achieved with loop statements and exits to interrupt sequences.

5.81.4 Exercise a function where flow control is achieved with goto

Exercise a function where flow control is achieved with goto and return statements to interrupt sequences.

5.82 Check processing of exemptions applying to a mix of statements and

Check processing of exemptions applying to a mix of statements and declarations in several units.

Exercise a package that exposes non-exempted code and a few exemption regions for a mix of statements and declarations in subprogram bodies, the package specification and the package elaboration sequence.

5.82.1 ada sources

- exemptions.adb
- exemptions.ads
- test_exemptions_all_exempted_code_call.adb
- test_exemptions_all_non_exempted_code_call.adb
- test_exemptions_exempted_code_call.adb
- test_exemptions_no_call.adb
- test_exemptions_non_exempted_code_call.adb

5.82.2 c sources

5.82.3 h sources

5.83 Check processing of exemptions applying to groups of statements within the

Check processing of exemptions applying to groups of statements within the root sequence of a subprogram body.

Exercise non-exempted code and a couple of exemption regions within a subprogram body, all protected by a local exception handler.

5.83.1 ada sources

- stacks.adb
- stacks.ads
- test_0.adb
- test_pop_u.adb
- test_push_0.adb
- test_push_o.adb

- test_pushpop_0.adb
- test_pushpop_o.adb
- test_pushpop_ou.adb
- test_pushpop_u.adb

5.83.2 c sources

5.83.3 h sources

5.84 Check processing of exemptions applying to groups of statements within a

Check processing of exemptions applying to groups of statements within a package body elaboration sequence, in both a regular and a generic package.

Exercise a regular or instanciated package which exposes a non-exempted subprogram and an exemption region for a conditioned set of statements within the package elaboration sequence.

5.84.1 ada sources

- com.adb
- com.ads
- comi_init.ads
- comi_noinit.ads
- gcom.adb
- gcom.ads
- test_com.adb
- test_gcom_f.adb
- test_gcom_t.adb
- test_gcom_tf.adb

5.84.2 c sources

5.84.3 h sources

5.85 Check processing of exemptions applying to groups of statements within local

Check processing of exemptions applying to groups of statements within local exception handlers.

Exercise non-exempted code and an exemption region for a short sequence of statements within a local exception handler.

5.85.1 ada sources

- stacks.adb
- stacks.ads
- test_0.adb
- test_pop_u.adb
- test_push_0.adb
- test_push_o.adb
- test_pushpop_0.adb
- test_pushpop_o.adb
- test_pushpop_ou.adb
- test_pushpop_u.adb

5.85.2 c sources

5.85.3 h sources

5.86 Check processing of exemptions applying to a mix of statements and

Check processing of exemptions applying to a mix of statements and declarations in several regions of a single subprogram body.

Exercise a subprogram featuring non-exempted code and several disjoint exemption regions, in both the toplevel sequence of statements and a local exception handler.

5.86.1 ada sources

- multiple_exemptions.adb
- mx.adb
- mx.ads
- test_0.adb
- test_exempt_0.adb
- test_exempt_all.adb
- test_exempt_xr12.adb
- test_exempt_xr13.adb
- test_exempt_xr23.adb

5.86.2 c sources

5.86.3 h sources

5.87 Check processing of exemptions applying to groups of entire subprograms.

Check processing of exemptions applying to groups of entire subprograms.

Exercise a package that exposes a couple of non-exempted subprogram bodies and an exemption region that encompasses another couple of subprograms entirely.

5.87.1 ada sources

- stacks.adb
- stacks.ads
- test_0.adb
- test_pop_u.adb
- test_push_0.adb
- test_push_o.adb
- test_pushpop_0.adb
- test_pushpop_o.adb
- test_pushpop_ou.adb
- test_pushpop_u.adb

5.87.2 c sources

5.87.3 h sources

5.88 Check processing of exemptions applying to whole subprogram bodies.

Check processing of exemptions applying to whole subprogram bodies.

Exercise a subprogram in which all the statements are part of a single exemption region.

5.88.1 ada sources

- test_tipos_0.adb
- test_tipos_all.adb
- test_tipos_other.adb
- test_tipos_pos.adb
- tipos.adb
- tipos.ads

5.88.2 c sources

5.88.3 h sources

5.89 Exercise a toplevel If statement with an else sequence embedding a For loop,

Exercise a toplevel If statement with an else sequence embedding a For loop, whose body features an inner If statement that controls access to a Case statement eventually.

Check variations over the outer If control, the For loop entry, the inner If control, and the possible Case selections, first independently from each other and then combined together.

5.89.1 ada sources

- sensors-status.adb
- sensors-status.ads
- test_broken.adb
- test_check.adb
- test_check_brok.adb
- test_ok.adb
- test_ok_brok.adb
- test_ok_check.adb
- test_ok_check_brok.adb
- test_und_brok.adb
- test_und_check.adb
- test_und_ok.adb
- test_und_ok_brok.adb
- test_und_ok_check.adb
- test_undecide.adb

5.89.2 c sources

5.89.3 h sources

5.90 Exercise a simple If statement within a While loop.

Exercise a simple If statement within a While loop.

Check cases where

- The loop is not entered
- The loop is entered and the if control decision never evaluates True
- The loop is entered and the if control decision never evaluates False
- The loop is entered and the if control decision evaluates True and False

5.90.1 ada sources

- sensors-predicates.adb
- sensors-predicates.ads
- slists-count.adb
- slists-count.ads
- test_iff.adb
- test_ift.adb
- test_iftf.adb
- test_noentry.adb

5.90.2 c sources

5.90.3 h sources

5.91 Exercise a three way If statement within a While loop.

Exercise a three way If statement within a While loop.

Check cases where

- The loop is not entered
- The loop is entered and all the possible combinations of if controls are checked

5.91.1 ada sources

- slists-fault.adb
- slists-fault.ads
- test_if_fft.adb
- test_if_ftf.adb
- test_if_ftt.adb
- test_if_tff.adb
- test_if_tft.adb
- test_if_ttf.adb
- test_if_ttt.adb
- test_noentry.adb

5.91.2 c sources

5.91.3 h sources

5.92 Exercise a simple If statement nested in one of the possible selections of a

Exercise a simple If statement nested in one of the possible selections of a Case statement, itself conditioned by an outer If statement executed as part of a While loop.

Check situations where:

- The loop is not entered
- The loop is entered and the first If statement control evaluates only False, hence the inner Case statement is never reached
- The inner Case statement is reached for each possible selection alone, then for combinations of them. The inner If statement is exercised both ways.

5.92.1 ada sources

- slists-forall.adb
- slists-forall.ads
- test_act_inh.adb
- test_activate.adb
- test_inhibit.adb
- test_nocase.adb
- test_noentry.adb
- test_noinhibit.adb

5.92.2 c sources

5.92.3 h sources

5.93 Check that coverage is not affected by the presence of subprograms oveloading.

Check that coverage is not affected by the presence of subprograms oveloading. Check different level of nesting.

5.93.1 ada sources

- overloads.adb
- overloads.ads
- services.adb
- services.ads
- sub_services-flipb.adb
- sub_services-flipx.adb

- sub_services.adb
- sub_services.ads
- test_flip_0.adb
- test_flip_b.adb
- test_flip_full.adb
- test_flip_x.adb
- test_serv_aa.adb
- test_serv_ab.adb
- test_serv_ba.adb
- test_sub_flip_0.adb
- test_sub_flip_b.adb
- test_sub_flip_full.adb
- test_sub_flip_x.adb

5.93.2 c sources

5.93.3 h sources

5.94 Check that SC assessments remain correct despite misleading identations.

Check that SC assessments remain correct despite misleading identations.

5.94.1 ada sources

- nop.adb
- nop.ads
- test_nop.adb

5.94.2 c sources

5.94.3 h sources

5.95 Check that no stmt coverage violations are reported for explicitly deactivated

Check that no stmt coverage violations are reported for explicitly deactivated sequences, in a subprogram or package elaboration body.

5.95.1 ada sources

- ctl.ads
- ops.adb

- ops.ads
- test_ops_0.adb
- test_ops_inc.adb

5.95.2 c sources

5.95.3 h sources

5.96 Check that all/none of the straightline statements in a big subprogram are

Check that all/none of the straightline statements in a big subprogram are reported uncovered when the subprogram is called/not-called.

5.96.1 ada sources

- bump.adb
- bump.ads
- test_bump_0.adb
- test_bump_all.adb

5.96.2 c sources

5.96.3 h sources

5.97 Check that statements spanning multiple lines are handled properly.

Check that statements spanning multiple lines are handled properly.

Exercise a subprogram which features a single statement spanning multiple lines. Verify that a single violation designating the first line only is reported when the statement is not covered (when the subprogram is not called), and that nothing is reported uncovered otherwise.

5.97.1 ada sources

- andnot.adb
- test_andnot_0.adb
- test_andnot_tf.adb

5.97.2 c sources

5.97.3 h sources

5.98 Check that multiple statements located on the same source line are

Check that multiple statements located on the same source line are handled properly.

Exercise cases with both unconditional and conditional statements on a single source line.

5.98.1 ada sources

- andnot.adb
- halfadd.adb
- test_andnot_0.adb
- test_andnot_tf.adb
- test_halfadd_0.adb
- test_halfadd_ff.adb
- test_halfadd_ft.adb
- test_halfadd_full.adb
- test_halfadd_tf.adb
- test_halfadd_tt.adb

5.98.2 c sources

5.98.3 h sources

5.99 Check that sequences of multiple nop statements (e.g. null or pragma) are

Check that sequences of multiple nop statements (e.g. null or pragma) are handled properly.

Exercise a package that exposes a subprogram containing two statements with no associated code in sequence, followed by a statement with code.

Verify that the two statements with no code are reported with the same coverage status as the following statement with code.

5.99.1 ada sources

- multibackprop.adb
- multibackprop.ads
- test_multibackprop_0.adb
- test_multibackprop_f.adb
- test_multibackprop_t.adb

5.99.2 c sources

5.99.3 h sources

5.100 Check that the tool isn't fooled into thinking that a statement is not covered

Check that the tool isn't fooled into thinking that a statement is not covered just because part of its execution (expression evaluation) is shortcircuited.

Verify that `return A and then B` is not reported uncovered when exercised with A False only.

5.100.1 ada sources

- andthen.adb
- andthen.ads
- test_andthen_0.adb
- test_andthen_ff.adb
- test_andthen_ft.adb
- test_andthen_tf.adb
- test_andthen_tt.adb

5.100.2 c sources

5.100.3 h sources

5.101 Check output report compliance to expectations on a case involving a single

Check output report compliance to expectations on a case involving a single trace and at least one violation per coverage criterion.

Proceed by exercising a single program once, in a way known to incur improper coverage of the all the criteria.

5.101.1 ada sources

- test_expr_some.adb

5.101.2 c sources

5.101.3 h sources

5.102 Check output report compliance to expectations on a case where no violation or

Check output report compliance to expectations on a case where no violation or exempted region is expected.

Proceed by exercising once a single functional unit where no exemption region is declared, and in a way know to ensure full coverage.

5.102.1 ada sources

- test_expr_full.adb

5.102.2 c sources

5.102.3 h sources

5.103 Check output report compliance to expectations on a case with

Check output report compliance to expectations on a case with an exemption region declared.

5.103.1 ada sources

- ranges.adb
- ranges.ads
- test_ranges_invalid.adb
- test_ranges_overlap.adb

5.103.2 c sources

5.103.3 h sources

5.104 Check output report compliance to expectations on a case involving a multiple

Check output report compliance to expectations on a case involving a multiple traces and expected violations.

Proceed by exercising a single functional unit in multiple different ways, still known to incur improper coverage of the most basic criterion overall.

5.104.1 ada sources

- test_expr_0.adb
- test_expr_ft.adb

5.104.2 c sources

5.104.3 h sources

5.105 Check output report compliance to expectations on a case involving a single

Check output report compliance to expectations on a case involving a single trace and expected violations.

Proceed by exercising a single program once, in a way known to incur improper coverage of the most basic criterion.

5.105.1 ada sources

- test_expr_tf.adb

5.105.2 c sources

5.105.3 h sources

INDEX

A

Ada
- DECISION Coverage, 19
- MCDC Coverage, 22
- STATEMENT Coverage, 5
- Toplevel Requirement Group, 5

D

DECISION Coverage
- Ada, 19

M

MCDC Coverage
- Ada, 22

R

Requirements
- Ada DECISION Coverage, 19
- Ada MCDC Coverage, 22
- Ada STATEMENT Coverage, 5

S

STATEMENT Coverage
- Ada, 5