Model-Based Testing

(DIT848 / DAT261)

Spring 2017

Lecture 11

Selecting your tests

(Coverage at the model level)

Gerardo Schneider
Department of Computer Science and Engineering
Chalmers | University of Gothenburg

About coverage criteria

- Test selection criteria help to design black-box test suites
  - They do not depend on the SUT code
- Model coverage criteria and SUT code coverage are complementary
- In white-box testing, coverage criteria are used for:
  - Measuring the adequacy of test suite
  - Deciding when to stop testing
- Coverage criteria may be used prescriptively
  - ”Try to cover all branches”
- Test generation tools can provide metrics on how well the coverage was, and which parts of the model were not covered

Test selection criteria

1. Structural model coverage criteria
2. Data coverage criteria
3. Fault-model criteria
4. Requirements-based criteria
5. Explicit test case specifications
6. Statistical test generation methods

Focus on the first 2
Not much about the rest

1. Structural model coverage

- Major issue: measure and maximize coverage of the **model**
  - Not of the SUT
- Different "families" of structural model coverage criteria:
  1. Control-flow-oriented coverage criteria
  2. Data-flow-oriented coverage criteria
  3. Transition-based coverage criteria
  4. UML-based coverage criteria

Focus on the first 3

1. Structural model coverage
1.1 Control-flow oriented

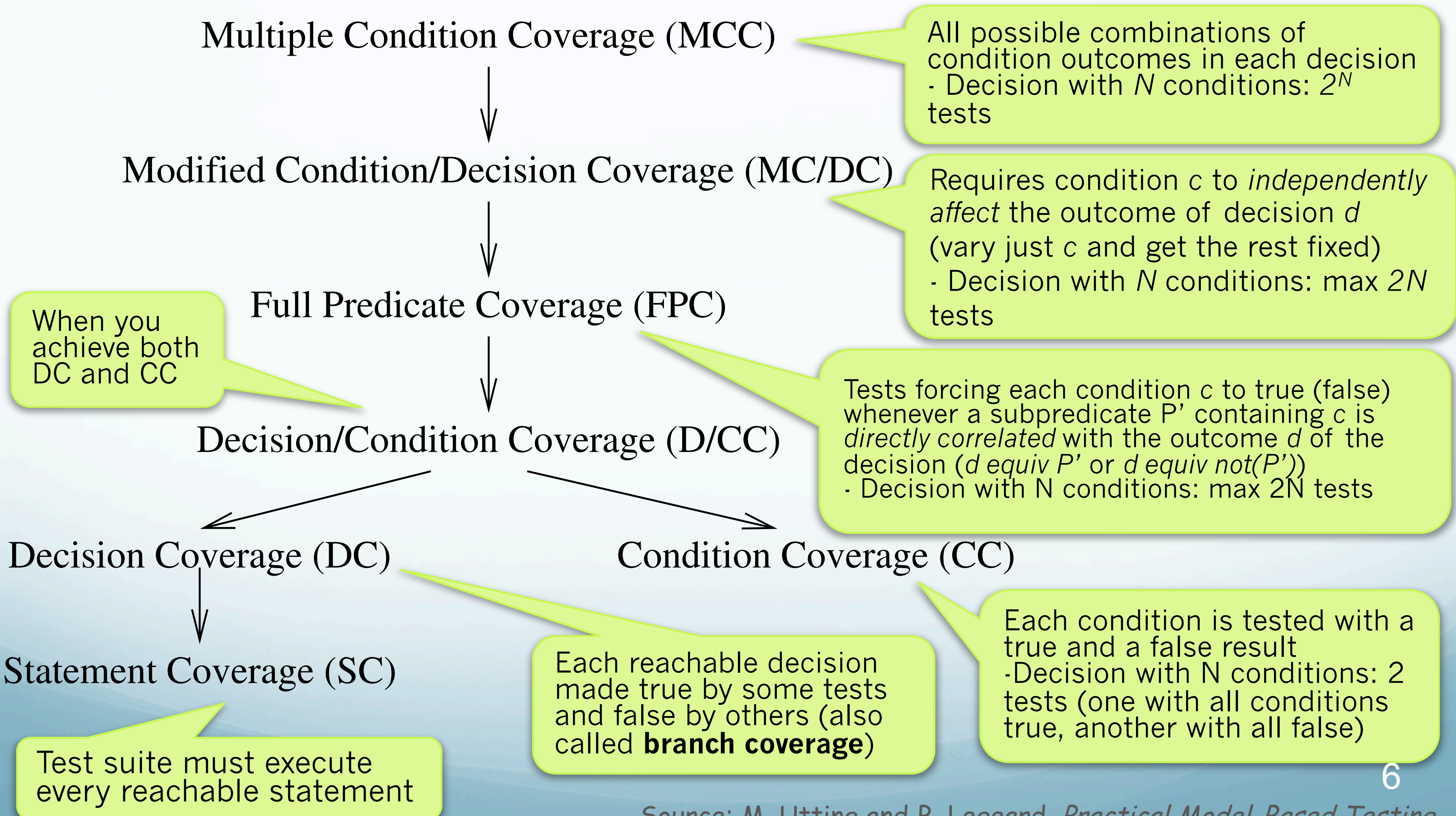
- The control-flow oriented criteria we will see are mostly for pre/post notations
  - E.g., Statement Coverage (SC) not relevant in FSM -> rather talk about *all-states* or *all-transitions* coverage
  - It makes more sense on modeling languages like B
- Still, some of the criteria are useful also for transition-based notations with data information (e.g. EFSMs)

Notation:

- **Decision** -> branch (an **if-then-else**)
- A decision contains one or more primitive **conditions** (combined by *and*, *or*, and *not* operators)

Ex: **If not(a>0) or (b>0 and c>0) then B else C**

1. Structural model coverage
1.1 Control-flow oriented



Source: M. Utting and B. Legeard, *Practical Model-Based Testing*

1. Structural model coverage
1.1 Control-flow oriented (Examples)

- **CC** – Ex: **not(a>0) or (b>0 and c>0)** -> E.g., write a test with a>0, b>0, and c>0 and another with a<=0, b<=0 and c<=0

- **DC** – Ex: **not(a>0) or (b>0 and c>0)** -> E.g., write a test with a<=0, b>0, and c>0 (decision: true) and another with a>0, b>0 and c<=0 (decision: false)

- **FPC** – Ex: **a>0 or (b>0 and c>0)** – Condition a>0 is directly correlated to the decision as making it true will make the decision true, and when fixed to false the output will be false by making b>0 and c>0 both false. Tests:

  1: fix b<=0, c<=0, set **a>0** -> decision = true

  2: fix b<=0, c<=0, set **a<=0** -> decision = false

  3: fix a<=0, set **b>0, c>0**, -> decision = true

  4: fix a<=0, set **(b>0 and c>0) to be false** -> decision = false

1. Structural model coverage
1.1 Control-flow oriented (Examples)

- **MC/DC** – Fix a and c to a given value in such a way that the test b>0 independently affects the outcome: write the 2 tests, one with b>0 and another one with b<=0 (Do the same later by fixing b and by fixing c and so on). So, all the test cases of FPC apply here (there is one more case):

  1. Fixing b,c: **a>0**, b<=0, c<=0 -> decision: true
  2. Fixing b,c: **a<=0**, b<=0, c<=0 -> decision: false
  3. Fixing a,c: a<=0, **b>0**, c>0 -> decision: true
  4. Fixing a,c: a<=0, **b<=0**, c>0 -> decision: false

  3'. Fixing a,b: a<=0, b>0, **c>0** -> decision: true (this test was already covered by test 3)

  5. Fixing a,b: a<=0, b>0, **c<=0** -> decision: false

1. Structural model coverage
1.1 Control-flow oriented

- The difference between **MC/DC** and **FPC** not very clear
  - Not uniform definition for FPC in the literature
- **MC/DC** – Standard for avionics to ensure adequate testing of the most critical software (level A)
  - Software development guidance DO-178B and DO-178C

1. Structural model coverage
1.1 Control-flow oriented

- **Often combined with transition-based and data-oriented coverage criteria**
- *Code coverage* is based on statements, decisions (branches), loops, and paths
- Some modeling notations (eg. UML/OCL, B) have no loops!
- Path coverage (test suite must execute every satisfiable path through the control-flow graph) not possible in code-based testing
  - In pre/post notations: if all combinations of decision outcomes are tested, path coverage is obtained (**?!**)
  - … so, no path coverage in previous slides ☺
- FPC as defined here is different from the book!
  - Confusing as many different definitions -> **we will not use FPC in this course!**

1. Structural model coverage
1.2 Data-flow oriented

- Control-flow graphs can be annotated with extra information on the **definition** and **use** of data variables

- Def-use pair *(d_v,u_v)* – d_v is a definition of *v*, u_v is its use

All–def–use–paths

Test suite to test all def-use pairs (d_v,u_v) and to test all paths from d_v till u_v

↓

All–uses

Test suite to test all def-use pairs (d_v,u_v) (all feasible use of all definitions)

↓

All–definitions

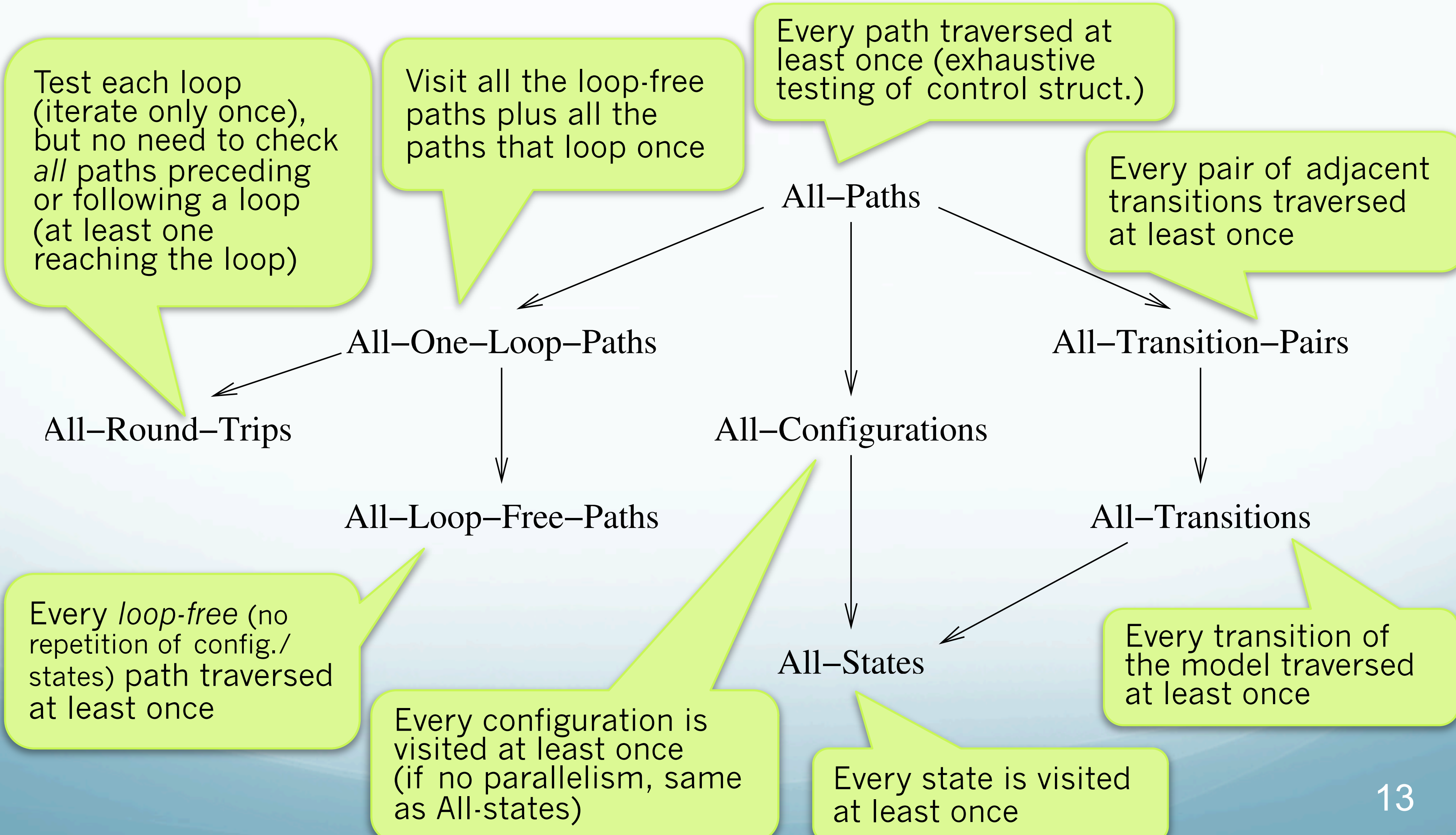
Test suite to test at least one def-use pair (d_v,u_v) for each def. d_v

Difference between **All.uses** and **All-def-uses-paths**: the latter must check all the branches linking *d* and *u*

1. Structural model coverage
1.3 Transition-based

- Transitions systems made up of states and transitions
- Depending on notation, transitions labeled with inputs, outputs, events, guards, and/or actions
- Usually models parallel systems
- A configuration is roughly a snapshot of the active states (of each parallel process)
- In this coverage criteria we restrict to **reachable** paths

1. Structural model coverage
1.3 Transition-based

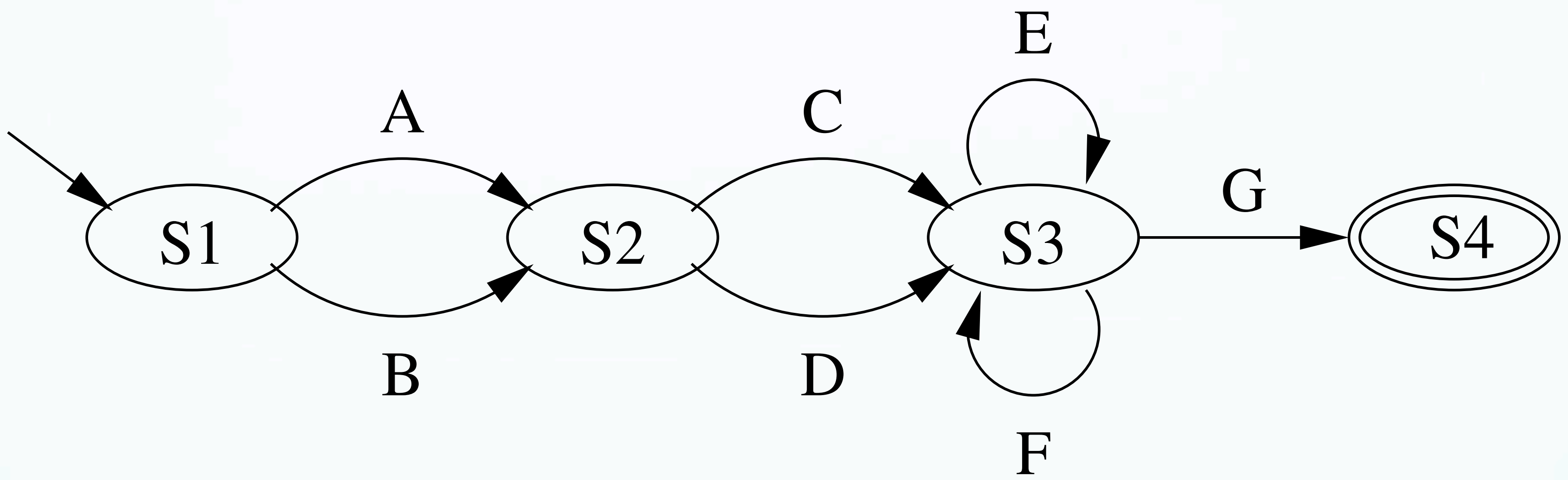


1. Structural model coverage
1.3 Transition-based

REMARK

- According to Utting & Legeard's book (p.119) All-Round-Trips should require an All-Transitions coverage too (and thus an arrow from the former to the latter should be present in the picture in previous slide). However, in this course we will take the less strict definition (consistent with the picture) that this is not the case.
- That is: **All-Round-Trips coverage does not require full coverage of all transitions, but only that all loops are part of the test suite** (finishing in the final state)

1. Structural model coverage
1.3 Transition-based

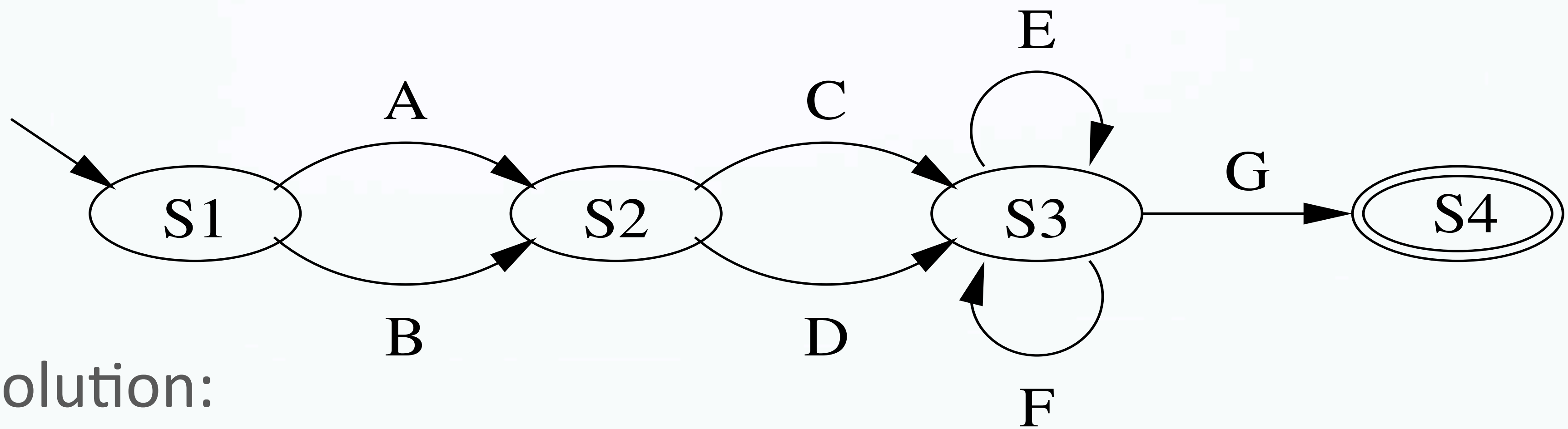


- Write down examples of each transition-based coverage criteria for the above FSM
  - All-states
  - All-configurations
  - All-transitions
  - All-transition-pairs
  - All-loop-free-paths
  - All-one-loop-paths
  - All-round-trips
  - All-paths

Groups 2-5 persons: 15 min

Source: M. Utting and B. Legeard, *Practical Model-Based Testing*

1. Structural model coverage
1.3 Transition-based



Solution:

- All-states
  - A;C;G
- All-configurations
  - Equal to All-states
- All-transitions
  - A;C;E;F;G and B;D;G
- All-transition-pairs
  - Eg..at state S2: A;C, A;D, B;C, B;D (do the same for each state)
- All-loop-free-paths
  - A;C;G, A;D;G, B;C;G, B;D;G
- All-one-loop-paths
  - 4 paths of all-loop-free-paths + combination of each of these with a single loop around either E or F transition (4+2*4=12 tests)
- All-round-trips
  - A;C;E;G, A;C;F;G (even simpler: A;C;E;F;G)
- All-paths
  - 4 paths of all-loop-free-paths but extended with any number of E and F transitions

Source: M. Utting and B. Legeard, *Practical Model-Based Testing*

2. Data coverage criteria

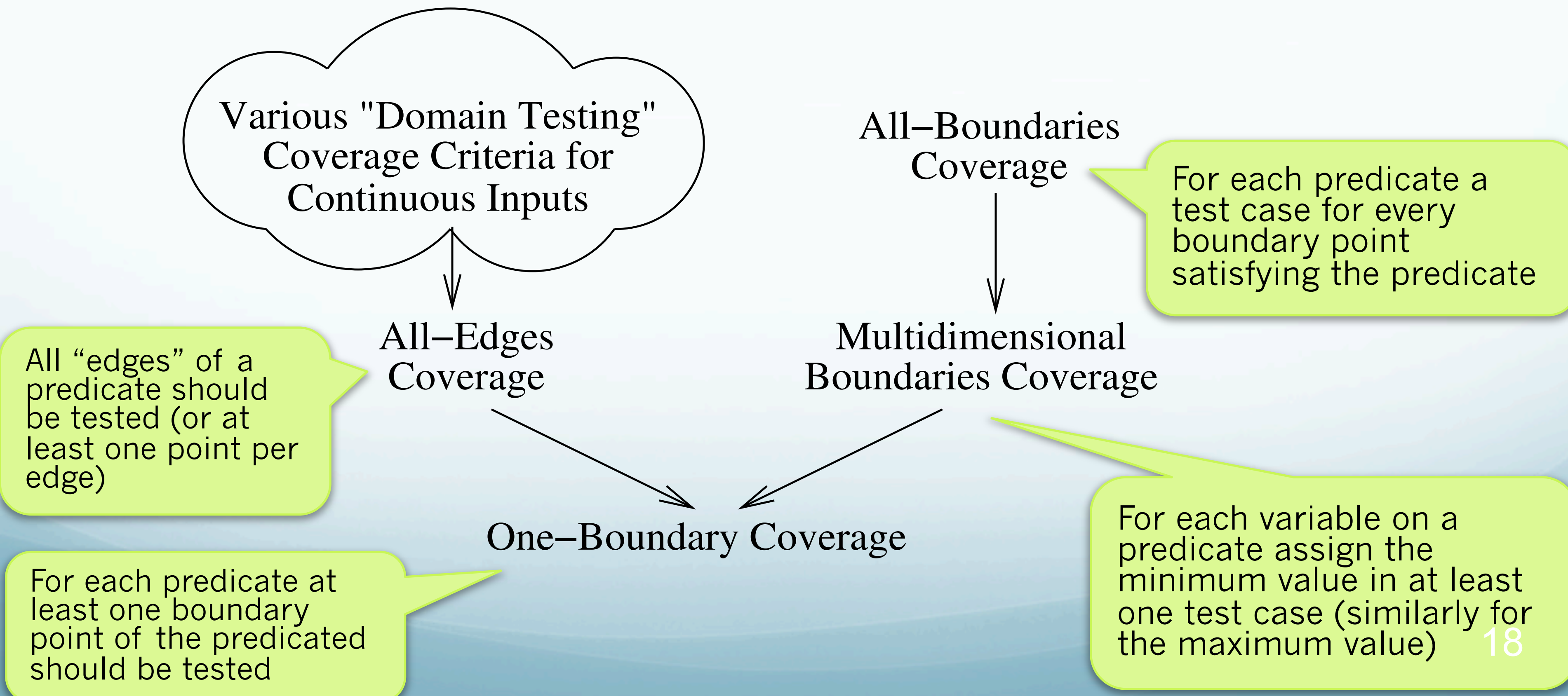
- Useful for choosing good data value representatives as test inputs
- Over a domain *D*, two extreme data coverage criteria
  - **One-value**: at least one value from *D* (in combination with other test criteria might be useful)
  - **All-values**: every value in *D*. Not practical in general
- More realistic:

1. Boundary values
2. Statistical data coverage
3. Pairwise testing

2. Data coverage criteria
2.1 Boundary value testing

Choosing values at the boundaries of input domains

- Usually constraints on values are predicates representing some regions: 1<=x<=5 and 2<=y<=7



2. Data coverage criteria
2.1 Boundary value testing

1. Write a geometrical representation of the following predicate, and consider what could be the boundary values for such predicate (integer)

   $$(x^2+y^2<=25) \ \& \ (0<=y) \ \& \ (x+y<=5)$$

2. Write boundary-oriented coverage for the case above so you achieve
   - All-boundaries coverage
   - Multidimensional-boundaries coverage
   - All-edges coverage

Groups 2-5 persons: 10 min

2. Data coverage criteria
2.1 Boundary value testing

Solution:

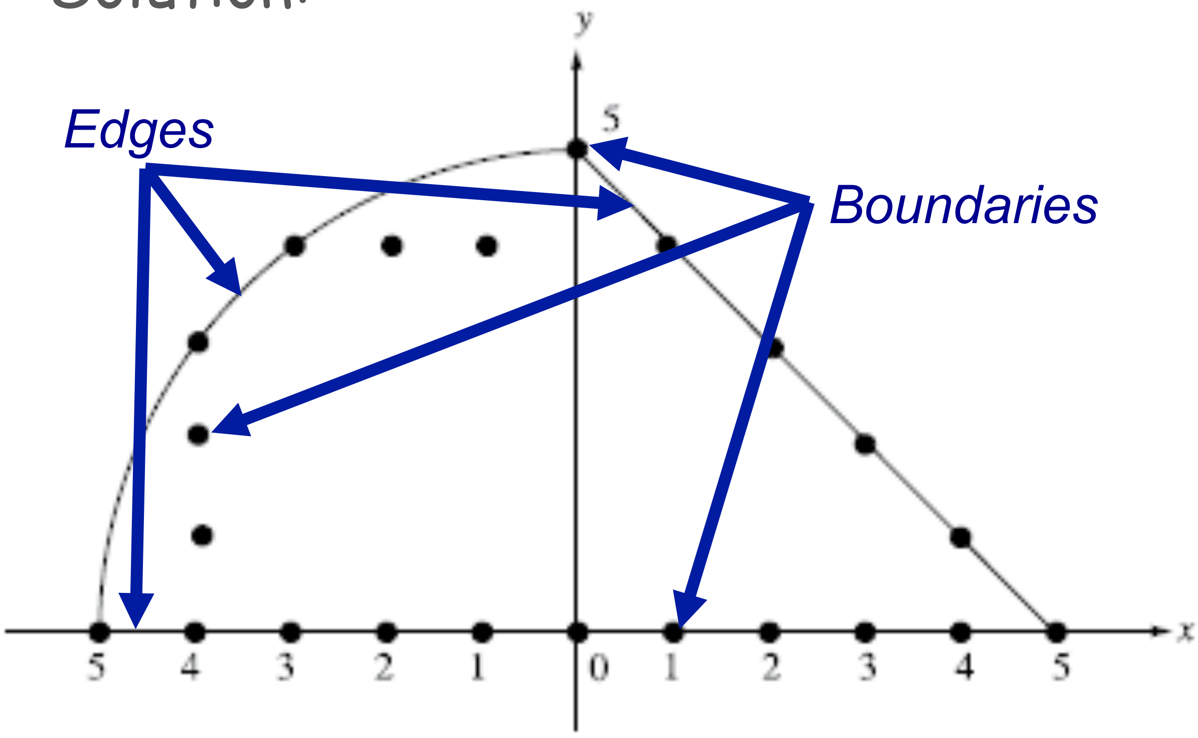


FIGURE 4.7 Example of the integer boundary points of a shape.

- All-boundaries coverage
  - The 22 boundary points depicted in the picture
- Multidimensional-boundaries coverage
  - Tests: (5,0), (-5,0), (0,5), and (x,0)
    for any -5<=X<=5
- All-edges coverage
  - Eg. (5,0) and (0,5)

Utting & Legeard book: Fig. 4.7, pp.125!

Source: M. Utting and B. Legeard, *Practical Model-Based Testing*

2. Data coverage criteria
2.2 Statistical data coverage

- Choosing random tests is as good as finding faults as *partition* testing
  - Could be more cost-effective
- Criterion: Random-value coverage (with distribution D)
  - Values of a given data variable in the test suite to follow the statistical distribution D

Example:

car_speed >50 and rain_level >5

(with car_speed: 0..300 and rain_level:0..10)

- Boundary testing: 4 tests (51 and 300 for *car*, 6 and 10 for *rain*)
- If we want 50 tests: generate them randomly with some distribution

3. Fault-based criteria

- A software testing technique using test data designed to demonstrate the absence of a set of **pre-specified faults** (known or recurrent faults)
- Mutation testing: program mutants are created by syntactic transformation of the SUT
  - Using **mutation operators**
- Executing a test suite on all mutants allows to measure the percentage of mutants **killed** by the test suite (exposing a fault in the mutant)
- Mutation of operators also guide the design of tests
  - Tests helping to distinguish a program from its mutant

4. Requirements-based criteria

- Each requirement (*a testable statement of some functionality the product must have*) should be tested
- Requirements can be used both to measure a level of coverage for the generated test case and to drive the test generation itself
- All-requirements coverage
  - Record the requirements inside the behavioral model (as annotations)
  - Formalize each requirement and use it as a test selection criterion

5. Explicit test case specifications

- Besides the model, the tester writes test case specifications in some formal notation
- Used to determine which tests to generate
- Notation could be the same as the modeling language, but not necessarily
- FSMs, regular expressions, temporal logic, Markov chains, etc.
- Give precise control over generated tests

6. Statistical test generation methods

- In MBT statistical test generation is usually used to generate test sequences from environmental models
- Usually using Markov chains (roughly, a FSM with probabilities)
- Test cases with greater probability to be generated first (and more often if organized in different classes)

Combining test selection criteria

- Criteria seen have different scopes and purposes: good to combine them
- See some interesting examples in Utting & Legeard, section 4.7 (pp.134-135)
- **Note:** Some researchers think guiding your tests based on coverage analysis is "risky"
  - *"The Risks of Coverage-Directed Test Case Generation"*, G. Gay et al. (http://www.cpe.virginia.edu/grads/pdfs/August%202015/software%20systems.PDF)

References

- M. Utting and B. Legeard, *Practical Model-Based Testing*. Elsevier - Morgan Kaufmann Publishers, 2007
  - Chapter 4

Remaining time is to be dedicated to work on your mini-project

- Revision lecture **Mon May 22, at 9:00**