Logic-based test coverage

- Basic approach
- Clauses and predicates
- Basic coverage criteria: CC, PC, CoC
- Structural logic coverage of source code
- Logic coverage of specifications
- Active clause coverage criteria (GACC, CACC, RACC)

Ciências ULisboa

slides: Eduardo Marques, Vasco Vasconcelos, Francisco Martins, João Neto

Logic Coverage

- Approach: Test criteria are based on logical expressions found in the specification or code of the SUT.
- **Predicate**: An expression that evaluates to a boolean value (true or false) and may contain boolean-valued expressions connected by logical operators.
- **Clause:** A boolean expression that does not make use of logical operators.
- Example: $x > 10 \wedge (f(y) = 30 \vee z)$ is a predicate
  - $x > 10$, $f(y) = 30$, and z are clauses
  - $\wedge$ and $\vee$ are logical operators.

Logical operators

- $\mathbf{a \land b}$ (and)
- $\mathbf{a \lor b}$ (or)
- $\mathbf{\neg a}$ (negation)
- $\mathbf{a \rightarrow b}$ (implication)
- $\mathbf{a \leftrightarrow b}$ (equivalence: a if and only if b)
- $\mathbf{a \oplus b}$ (exclusive or/choice, also known as “xor”)

Test requirements, syntax and semantics of logical expressions

- In the previous lectures we treated logic expressions according to their semantic meaning, not their syntax.
- As a consequence, expressions $\mathbf{a} \leftrightarrow \mathbf{b}$ and $\mathbf{a} \rightarrow \mathbf{b} \wedge \mathbf{b} \rightarrow \mathbf{a}$ yield the same test requirements.
- This is about to change.

Clauses and predicates

- Let **P** be a **set of predicates**
- Let **C** the **set of clauses** in the predicates in **P**.
- For each predicate $p \in P$, C_p is the **set of clauses** in p, that is, $C_p = \{c \mid c \in p\}$.
- Then **C** is the union of the clauses in each predicate in **P**,

  that is, $C = \bigcup_{p \in P} C_p$.

Predicate coverage (PC)

- For each predicate $p \in P$, TR contains two requirements: p evaluates to true, and p evaluates to false.
- The graph version of predicate coverage was introduced in before as edge coverage.
- Two tests that satisfy PC for $x > 10 \wedge (f(y) == 30 \vee z)$ are
  - $(x=20, f(y)=50, z=true)$ and $(x=0, f(y)=50, z=true)$
- An obvious failing of this criterion is that the individual clauses are not always exercised.

Clause coverage (CC)

- For each clause $c \in C$, TR contains two requirements: c evaluates to *true* and c evaluates to *false*.
- Two tests that satisfy CC for $x > 10 \land (f(y) == 30 \lor z)$ are
  - $(x=20, f(y)=50, z=true)$ and $(x=0, f(y)=30, z=false)$

CC does not subsume PC
PC does not subsume CC

- Take predicate $p = a \vee b$
- Test set $T_{23} = \{2,3\}$ satisfies CC, but not PC, because p is never false.
- Test set $T_{24} = \{2, 4\}$ satisfies PC, but not CC, because b is never true.
- The most direct approach to rectify this problem is to try all combinations of clauses.

	a	b	$a \vee b$
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

Combinatorial Coverage (CoC)

- For each $p \in P$, TR contains test requirements for each possible combination of truth values of clauses in C_p.
- A predicate p with n independent clauses have 2^n possible assignments of truth values.
- CoC is impractical for predicates with more than a few clauses.

	a	b	c	$(a \vee b) \wedge c$
1	T	T	T	T
2	T	T	F	F
3	T	F	T	T
4	T	F	F	F
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

Example

- For **p1** = $x > y \land (x = z-1 \lor x > z)$ the clauses are:
  - **a**: $x > y$ **b**: $x = z-1$ **c**: $x > z$
- For **p2** = $z > 0 \lor z > x+y$ the clauses are:
  - **d**: $z > 0$ **e**: $z > x+y$
- For **P** = {**p1**, **p2**} we have
  - **TR**(**PC**) = {**p1, ¬p1, p2, ¬p2**}
  - **TR**(**CC**) = {**a, ¬a, b, ¬b, c, ¬c, d, ¬d, e, ¬e**}
  - **TR**(**CoC**) = {**a∧b∧c, ¬a∧b∧c, a∧¬b∧c, a∧b∧¬c, ¬a∧¬b∧c, ¬a∧b∧¬c, a∧¬b∧¬c, ¬a∧¬b∧¬c, d∧e, ¬d∧e, d∧¬e, ¬d∧¬e**}
- **Exercise 1**: Find combinations of values for x, y, z that will satisfy the test requirements of PC and CC. Are there infeasible requirements for CoC?

Subsumption relations

- **PC** does not subsume **CC**
- **CC** does not subsume **PC**
- **CoC** subsumes **CC** and **PC** of course (it covers all possible combinations)

Logical operators in source code

Logical expression	Java expression
$a \land b$	`a && b`
$a \lor b$	`a || b`
$\neg a$	`!a`
$a \rightarrow b$	`!a || b`
$a \leftrightarrow b$	`a == b`
$a \oplus b$	`a != b`

- **Note**: the & | ^ ~ operators also correspond to logical operators. What are the differences between && and & , || and | ?

Exercise 2

```java
public static int daysInMonth(int m, int y) {
  if (m <= 0 || m > 12)
    throw new IllegalArgumentException("Invalid month: " + m);
  if (m == 2) {
    if (y % 400 == 0 || y % 4 == 0 && y % 100 != 0)
      return 29;
    else
      return 28;
  }
  if (m <= 7) {
    if (m % 2 == 1)
      return 31;
    return 30;
  }
  if (m % 2 == 0)
    return 31;
  return 30;
}
```

- Predicates and clauses
  - **p1**: `c1 || c2`, where **c1**: `m <= 0`; **c2**: `m > 12`
  - **p2**: `c3`, where **c3**: `m == 2`
  - **p3**: `c4 || c5 && c6`, where **c4**: `y % 400 == 0`; **c5**: `y % 4 == 0`; **c6**: `y % 100 != 0`
  - **p4**: `c7`, where **c7**: `m <= 7`
  - **p5**: `c8`, where **c8**: `m % 2 == 1`
  - **p6**: `c9`, where **c9**: `m % 2 == 0`

- Identify TR(CC), TR(PC), and TR(CoC)

Structural logical coverage for source code

- Predicates are derived from decision points in programs
- Most predicates in programs have only 1 clause
  - Programmers tend to write predicates with a maximum of 2 or 3 clauses → criteria is not the problem
- The primary complexity of applying logic coverage to programs has to do with reachability
- Getting values that satisfy those requirements is only part of the problem; getting to the statement is sometimes more difficult

Reachability

- The test cases must include values to reach the predicate.
- For large programs, satisfying reachability can be enormously complex.
- Test requirements are often expressed in terms of program variables that may be defined locally.
- Local variables may have to be resolved in terms of the input variables. Consider:
  - `int x = lookup(complexFunction(input1, input2))`
- If the function includes randomness or is time sensitive, or if the input cannot be controlled by the tester, it may be impossible to satisfy the test requirement with certainty.

Reachability predicates

- The **reachability problem**: analyse a point in the program to find values that will force execution to reach the point
- Build a table that relates predicate p to the **reachability predicate** r(p) of p**:** a boolean expression on the **input variables** that enables p to be reached

```
public static int daysInMonth(int m, int y) {
  if (m <= 0 || m > 12) // p1
    throw …;
  if (m == 2) { // p2
    if (y % 400 == 0 || y % 4 == 0 && y % 100 != 0) // p3
    …;
  }
  …
}
```

p	r(p)
p1	true
p2	r(p1) && !p1
p3	r(p2) && p2

Clause coverage and reachability

- Clause coverage alone is not enough; tests must reach the clause.
- For example, test (m = 11, y = 2000) covers **c4** (y % 400 == 0), but does not reach the predicate that contains **c4**.

```
public static int daysInMonth(int m, int y) {
  if (m <= 0 || m > 12) // p1
   …;
  if (m == 2) { // p2
    if (y % 400 == 0 || y % 4 == 0 && y % 100 != 0) // p3
    …;
  }
  …
}
```

Exercise 3

```java
public static int daysInMonth(int m, int y) {
  if (m <= 0 || m > 12) // p1, c1, c2
    throw new IllegalArgumentException("Invalid month: " + m);
  if (m == 2) { // p2, c3
    if (y % 400 == 0 || y % 4 == 0 && y % 100 != 0) //p3, c4-6
      return 29;
    else
      return 28;
  }
  if (m <= 7) { // p4, c7
    if (m % 2 == 1) // p5, c8
      return 31;
    return 30;
  }
  if (m % 2 == 0) // p6, c9
    return 31;
  return 30;
}
```

#	m	y	expected	reach & cover
1	-45	2016	IAE	p1,c1,¬c2
2	27	2016	IAE	p1,¬c1,c2
3	2	2016	29	¬p1,¬c1,¬c2,p2, c3,p3,¬c4,c5,c6

- Build reachability predicates for the 6 predicates
- Identify test cases that satisfy PC, CC, CoC (complete the table)
- Are there infeasible requirements?

Specification-based Logic Coverage

- Software specifications include logical expressions, allowing the logic coverage criteria to be applied.
- For instance these may take the form of:
  - Contracts: informal (e.g., Javadoc) or formal (e.g., JML)
  - Finite State Machine modelling

Example _ JML contract for Time.tick()

```
public Time(int h, int m) { … }
public int getHours() { … }
public int getMinutes() { … }

/*@ public normal_behavior
  @   requires getMinutes() < 59;
  @   ensures getMinutes() == \old(getMinutes()) + 1;
  @   ensures getHours()   == \old(getHours());
  @ also
  @ public normal_behavior
  @   requires getMinutes() == 59 && getHours() < 23;
  @   ensures getMinutes() == 0
  @   ensures getHours()   == \old(getHours()) + 1;
  @ also
  @ public normal_behavior
  @   requires getMinutes() == 59 && getHours() == 23;
  @   ensures getMinutes() == 0;
  @   ensures getHours()   == 0;
  @*/
  public void tick() { … }
```

JML pre-conditions define the predicates of interest

- **Exercise 4:** Three test cases satisfy PC (and CC too). Identify them.

Predicate determination

- PC and CC do not subsume each other; CoC may easily become unpractical or lead to too many infeasible requirements.
- When we introduce tests at the clause level, we want also to have an effect on the predicate.
- Determination, the conditions under which a clause influences the outcome of a predicate.
- Idea: if you flip the clause, and the predicate changes value, then the clause determines the predicate.
- For $p = a \wedge (b \vee c)$ the determination predicates are:
  - $d(a) = b \vee c$ — a determines p when $b \vee c$
  - $d(b) = a \wedge \neg c$ — b determines p when $a \wedge \neg c$
  - $d(c) = a \wedge \neg b$ — c determines p when $a \wedge \neg b$
- From the testing perspective, we would like to test each clause under circumstances where the clause determines the predicate.

Determination (more formally)

- Determination predicate
  - Let $p \in P$ and $c \in C_p$. We say that c **determines** p if there is a logical assignment (determination predicate) **d(c)** to all other clauses such that changing the value of c changes the value of p.
- Major and minor clauses (terminology)
  - The **major** clause is the clause on which we are focusing; all other clauses are the minor clauses.
  - Clause c in $d(c)$ is the major clause.
- **Finding the determination predicate**
  - **d(c)** $= p[true/c] \oplus p[false/c]$ where $p[B/c]$ stands for p with every occurrence of the major clause c replaced by B.

Deriving determination predicates

$\boxed{\mathbf{d(c) =} \; p[true/c] \oplus p[false/c]}$

- Example 1 - taking $p = a \wedge (b \vee c)$
  - $d(a) = p[true/a] \oplus p[false/a] = (b \vee c) \oplus false = b \vee c$
  - $d(b) = p[true/b] \oplus p[false/b] = a \oplus (a \wedge c) = a \wedge \neg c$
  - $d(c) = p[true/c] \oplus p[false/c] = a \oplus (a \wedge b) = a \wedge \neg b$
- Example 2 - taking $p = a \vee (b \wedge c)$
  - $d(a) = p[true/a] \oplus p[false/a] = true \oplus (b \wedge c) = \neg (b \wedge c) = \neg b \vee \neg c$
  - $d(b) = p[true/b] \oplus p[false/b] = (a \vee c) \oplus a = \neg a \wedge c$
  - $d(c) = p[true/c] \oplus p[false/c] = (a \vee b) \oplus a = \neg a \wedge b$

General Active Clause Coverage (GACC)

- For $p \in P$ and $c \in C_p$ include two requirements in TR:
  1. $c \wedge d(c)$
  2. $\neg c \wedge d(c)$
- Example: 2 predicates involving 5 clauses yields 10 test requirements
  - **P** = {**p1**, **p2**} , **p1** = **a** ∧ **(b** ∨ **c)** , **p2 = x** ∨ **y**
  - **d(a)** = **b** ∨ **c,** **d(b)** = **a** ∧ **¬c,** **d(c)** = **a** ∧ **¬b**
  - **d(x)** = **¬y,** **d(y) = ¬x**
  - **TR(GACC)** = {**a** ∧ **d(a), ¬a** ∧ **d(a), b** ∧ **d(b), ¬b** ∧ **d(b), c** ∧ **d(c), ¬c** ∧ **d(c)**, **x** ∧ **d(x), ¬x** ∧ **d(x), y** ∧ **d(y), ¬y** ∧ **d(y)**}

GACC and subsumption of CC/PC

- Does GACC subsume PC ? Not necessarily.
- **GACC subsumes CC, but not PC** (though this may happen in many practical cases of interest)
  - Example: for $p = a \leftrightarrow b$ we have $d(a) = true$ and $d(b) = true$
  - So $TR(GACC) = \{a, \neg a, b, \neg b\}$ [Obs.: equivalent to TR(CC)]
  - $T1 = \{[a=true, b=true], [a=false, b=false]\}$ satisfies GACC but not PC. Both assignments to a and b yield $p = true$.
  - $T2 = \{[a=true, b=false], [a=false, b=true]\}$ would also satisfy GACC but not PC. Both assignments to a and b yield $p = false$.

Correlated Active Clause Coverage (CACC)

- Idea: Correlate $c \wedge d(c)$ with the truth value of the predicate p. Note that c and p do not have to have the same value.
- For $p \in P$ and $c \in C_p$ include two requirements in TR:
  1. $c \wedge d(c) \wedge p$
  2. $\neg c \wedge d(c) \wedge \neg p$
  - that is, p must evaluate to true in one case and false in the other.
- Example: given $p = a \leftrightarrow b$, we may have for clause a the test set {TT, FT}, and for clause b test set {TT, TF}. Merging the two we obtain test set {TT, TF, FT} that satisfies CACC.
- CACC subsumes GACC [thus CC] but **also** PC.

GACC vs CACC (example)

#	a	b	c	p = a ∧ (b ↔ c)	Satisfies
1	T	T	T	T	a∧d(a) b∧d(b) c∧d(c)
2	T	T	F	F	b∧d(b) ¬c∧d(c)
3	T	F	T	F	¬b∧d(b) c∧d(c)
4	T	F	F	T	a∧d(a) ¬b∧d(b) ¬c∧d(c)
5	F	T	T	F	¬a ∧ d(a)
6	F	T	F	F	–
7	F	F	T	F	–
8	F	F	F	F	¬a∧d(a)

Determination

d(a) = b ↔ c

d(b) = a

d(c) = a

- **GACC** is satisfied by {#1, #4, #5} for instance. This choice of assignments will not cover the CACC requirements for b and c for the case where p must be false.

GACC vs CACC (example)

#	a	b	c	b ↔ c	p = a ∧ (b ↔ c)	Satisfies
1	T	T	T	T	T	a ∧ d(a) b ∧ d(b) c ∧ d(c)
2	T	T	F	F	F	b ∧ d(b) ¬c ∧ d(c)
3	T	F	T	F	F	¬b ∧ d(b) c ∧ d(c)
4	T	F	F	F	T	a ∧ d(a) ¬b ∧ d(b) ¬c ∧ d(c)
5	F	T	T	T	F	¬a ∧ d(a)
6	F	T	F	F	F	–
7	F	F	T	F	F	–
8	F	F	F	F	F	¬a ∧ d(a)

- **CACC** can be satisfied by {#1, #2, #3, #5}.
- **Exercise 5**: Perform a similar analysis for **p** = **a** ∨ **(b** ↔ **c)**.

Restricted Active Clause Coverage (RACC)

- For $p \in P$ and $c \in C_p$ include two requirements in TR:
  1. $c \wedge d(c) \wedge p$
  2. $\neg c \wedge d(c) \wedge \neg p$
  - the minor clause assignments must be the same in both cases.
- RACC subsumes CACC.
- RACC imposes more "uniform" tests but is also more likely to imply infeasible requirements.

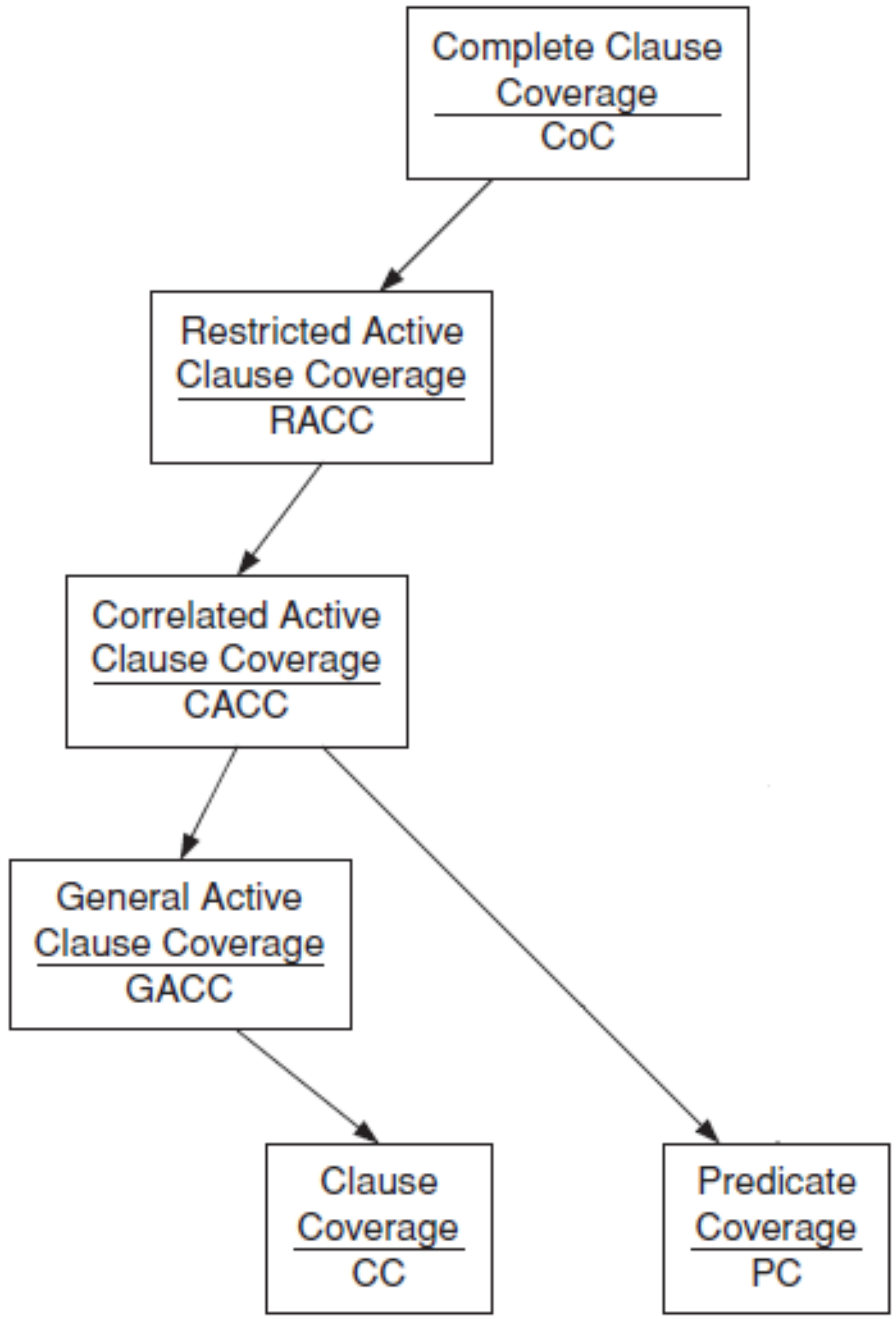
CACC vs RACC (example)

#	a	b	c	p = a ∧ (b ∨ c)	Satisfies
1	T	T	T	T	a ∧ d(a)
2	T	T	F	T	a ∧ d(a)
3	T	F	T	T	a ∧ d(a)
4	T	F	F	F	–
5	F	T	T	F	¬a ∧ d(a)
6	F	T	F	F	¬a ∧ d(a)
7	F	F	T	F	¬a ∧ d(a)
8	F	F	F	F	–

d(a) = b ∨ c

- **CACC coverage for a**: 9 possible choices: #1, #2, or #3 combined with one of #5, #6, or #7.
- **RACC coverage for a**: only 3 possible choices: #1 combined with #5, test #2 with #6, and #3 with #7.

Subsumption relations



Example: `isLeapYear()`

```
public static boolean isLeapYear(int y) {
  return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
}
```

```
a: y % 400 == 0    b: y % 4 == 0    c: y % 100 != 0
p: a ∨ (b ∧ c)

d(a) = ¬b ∨ ¬c    d(b) = ¬a ∧ c    d(c) = ¬a ∧ b
```

```
TR(GACC)={(1) a ∧ (¬b ∨ ¬c), (2) ¬a ∧ (¬b ∨ ¬c),
          (3) b ∧ ¬a ∧ c,    (4) ¬b ∧ ¬a ∧ c,
          (3) c ∧ ¬a ∧ b,    (5) ¬c ∧ ¬a ∧ b}
```

#	y	expected	clause values	covered GACC requirements
1	2000	true	a b ¬c	(1) a ∧ (¬b ∨ ¬c)
2	2001	false	¬a ¬b c	(2) ¬a ∧ (¬b ∨ ¬c) (4) ¬b ∧ ¬a ∧ c
3	1900	false	¬a b ¬c	(2) ¬a ∧ (¬b ∨ ¬c) (5) ¬c ∧ ¬a ∧ b
4	2004	true	¬a b c	(3) b ∧ ¬a ∧ c

`isLeapYear()` analysis

```java
public static boolean isLeapYear(int y) {
  return y % 400 == 0 || y % 4 == 0 && y % 100 != 0;
}
```

#	y	expected	clause values	covered GACC requirements
1	2000	true	a b ¬c	(1) $a \land (\neg b \lor \neg c)$
2	2001	false	¬a ¬b c	(2) $\neg a \land (\neg b \lor \neg c)$ (4) $\neg b \land \neg a \land c$
3	1900	false	¬a b ¬c	(2) $\neg a \land (\neg b \lor \neg c)$ (5) $\neg c \land \neg a \land b$
4	2004	true	¬a b c	(3) $b \land \neg a \land c$

- GACC coverage implies CACC coverage in this case
- RACC is also satisfied: (#1,#3) pair for a; (#2,#4) for b; (#3,#4) for c
- {#1, #2} satisfies CC and PC
- {#1, #3} satisfies PC but not CC
- CoC would lead to the following infeasible requirements
    - $a \land b \land c$, $a \land \neg b \land \neg c$, $a \land \neg b \land c$, $\neg a \land \neg b \land \neg c$

Exercise 6

```
public static TClass triangleType(int a, int b, int c) {
    if (a <= 0 || b <= 0 || c <= 0)  // p1
      return INVALID;
    if (a >= b + c || b >= a + c || c >= a + b) // p2
        return INVALID;
    int count = 0;
    if (a == b) // p3
      count++;
    if (a == c) // p4
      count++;
    if (b == c) // p5
      count++;
    if (count == 0) // p6
      return SCALENE;
    if (count == 1) // p7
      return ISOSCELES;
    return EQUILATERAL;
  }
```

- Identify:
    1. The reachability predicates
    2. TR(CC) and TR(PC)
    3. Test cases that satisfy a) PC, b) CC
    4. Determination predicates for the clauses of p1 and p2
    5. TR(CACC) for p1 and p2
    6. For p1 and p2, test cases that satisfy a) CACC, b) RACC. Are there infeasible requirements?