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)
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:** \( x > 10 \land (f(y) = 30 \lor z) \) is a predicate
  - \( x > 10 \), \( f(y) = 30 \), and \( z \) are clauses
  - \( \land \) and \( \lor \) are logical operators.
Logical operators

- $a \land b$ (and)
- $a \lor b$ (or)
- $\neg a$ (negation)
- $a \rightarrow b$ (implication)
- $a \leftrightarrow b$ (equivalence: $a$ if and only if $b$)
- $a \oplus b$ (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 $a \leftrightarrow b$ and $a \rightarrow b \wedge b \rightarrow a$ 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 $p \in P$, $C_p$ 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 before as edge coverage.
- Two tests that satisfy PC for $x > 10 \land (f(y) = 30 \lor z)$ 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 PC 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 $p = a \lor b$
- 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.

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$a \lor b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
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.

CC is unwieldy at best, and impractical for predicates with more than a few clauses.
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
  - \( \text{TR(PC)} = \{p1, \neg p1, p2, \neg p2\} \)
  - \( \text{TR(CC)} = \{a, \neg a, b, \neg b, c, \neg c, d, \neg d, e, \neg e\} \)
  - \( \text{TR(CoC)} = \{a \land b \land c, \neg a \land b \land c, a \land \neg b \land c, a \land b \land \neg c, \neg a \land \neg b \land c, \neg a \land b \land \neg c, a \land \neg b \land \neg c, d \land e, \neg d \land e, d \land \neg e, \neg d \land \neg e\} \)

- **Exercise 1:** Find combinations of values for \( x, y, z \) that will satisfy the test requirements of PC, CC and CoC in turn. Observe that there are 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)
Structural logical coverage for source code

- Predicates are derived from decision points in programs.
- The vast majority of 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.
Logical operators in source code

<table>
<thead>
<tr>
<th>Logical expression</th>
<th>Java expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>a ∧ b</td>
<td>a &amp;&amp; b</td>
</tr>
<tr>
<td>a ∨ b</td>
<td>a</td>
</tr>
<tr>
<td>¬ a</td>
<td>!a</td>
</tr>
<tr>
<td>a → b</td>
<td>!a</td>
</tr>
<tr>
<td>a ↔ b</td>
<td>a == b</td>
</tr>
<tr>
<td>a ⊕ b</td>
<td>a != b</td>
</tr>
</tbody>
</table>

Note: the & | ^ ~ operators also correspond to logical operators. What are the differences between && and & , || and | ?
Exercise 2

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)
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.

```java
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
            ...
    }
    ...
}
```

<table>
<thead>
<tr>
<th>$p$</th>
<th>$r(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p1$</td>
<td>true</td>
</tr>
<tr>
<td>$p2$</td>
<td>$r(p1) &amp;&amp; \neg p1$</td>
</tr>
<tr>
<td>$p3$</td>
<td>$r(p2) &amp;&amp; p2$</td>
</tr>
</tbody>
</table>
Clause coverage and reachability

- Clause coverage alone is not enough; tests must reach the clause.
- For example, test \( (m = 11, y = 2000) \) covers \( c_4 \) \((y \% 400 == 0)\), but does not reach the predicate that contains \( c_4 \).

```java
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;
}
```

<table>
<thead>
<tr>
<th>#</th>
<th>m</th>
<th>y</th>
<th>expected</th>
<th>reach &amp; cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-45</td>
<td>2016</td>
<td>IAE</td>
<td>p1,c1</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td>2016</td>
<td>IAE</td>
<td>p1,c2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2016</td>
<td>29</td>
<td>¬p1,¬c1,¬c2,p2,c3,p3,c5</td>
</tr>
</tbody>
</table>

- 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)
  - FSM abstractions
Example _ JML contract for Time.tick()

```java
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() { ... }
```

- **Exercise 4:** Three test cases satisfy PC (and CC too). Identify them.
Example 2 _ BoundedQueue.enqueue()

```java
/*@ 
public normal_behavior 
  requires !isFull();
  ensures size() == \old(size()) + 1;
  ensures elementAt(\old(size())) == data;
  ensures (\forall int i; i >= 0 && i < \old(size())
    ==> elementAt(i) == \old(elementAt(i)))
also
public exceptional_behavior 
  requires isFull();
  signals_only IllegalStateException;
*/
void enqueue(E data) throws IllegalStateException;
```

- We need to cover predicate (and clause) isFull()
- See complete specification online: BoundedQueue and BoundedArrayQueue.
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 \( p = a \land (b \lor c) \) the determination predicates are:
  - \( d(a) = b \lor c \) — \( a \) determines \( p \) when \( b \lor c \)
  - \( d(b) = a \land \neg c \) — \( b \) determines \( p \) when \( a \land \neg c \)
  - \( d(c) = a \land \neg b \) — \( c \) determines \( p \) when \( a \land \neg b \)
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 s.t. 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 the **minor clauses**. Clause \( c \) in \( d(c) \) is the major clause.

- **Finding the determination predicate**
  - \( d(c) = p[\text{true}/c] \oplus p[\text{false}/c] \) where \( p[B/c] \) stands for \( p \) with every occurrence of the major clause \( c \) replaced by \( B \).
Deriving determination predicates

Example 1 - taking \( p = a \land (b \lor c) \)

\[
\begin{align*}
    d(a) &= p[true/a] \oplus p[false/a] = (b \lor c) \oplus false = b \lor c \\
    d(b) &= p[true/b] \oplus p[false/b] = a \oplus (a \land c) = a \land \neg c \\
    d(c) &= p[true/c] \oplus p[false/c] = a \oplus (a \land b) = a \land \neg b
\end{align*}
\]

Example 2 - taking \( p = a \lor (b \land c) \)

\[
\begin{align*}
    d(a) &= p[true/a] \oplus p[false/a] = true \oplus (b \land c) = \neg (b \land c) = \neg b \lor \neg c \\
    d(b) &= p[true/b] \oplus p[false/b] = (a \lor c) \oplus a = \neg a \land c \\
    d(c) &= p[true/c] \oplus p[false/c] = (a \lor b) \oplus a = \neg a \land b
\end{align*}
\]
General Active Clause Coverage (GAAC)

For \( p \in P \) and \( c \in C_P \) include two requirements in TR:

1. \( c \land d(c) \)
2. \( \neg c \land d(c) \)

Example: 2 predicates involving 5 clauses yields 10 test requirements

- \( P = \{p1, p2\} \), \( p1 = a \land (b \lor c) \), \( p2 = x \lor y \)
- \( d(a) = b \lor c \), \( d(b) = a \land \neg c \), \( d(c) = a \land \neg b \)
- \( d(x) = \neg y \), \( d(y) = \neg x \)
- \( TR(GACC) = \{a \land d(a), \neg a \land d(a), b \land d(b), \neg b \land d(b), c \land d(c), \neg c \land d(c), x \land d(x), \neg x \land d(x), y \land d(y), \neg y \land d(y)\} \)
GACC and subsumption of CC/PC

- Does GACC subsume PC? Not necessarily.
- By definition **GACC subsumes CC, but not PC** (though this may happen in many practical cases of interest)
  - Example: for \( p = a \leftrightarrow b \) we have \( d(a) = \text{true} \) and \( d(b) = \text{true} \)
  - So \( \text{TR(GACC)} = \{a, \neg a, b, \neg b\} \) [Obs.: equivalent to TR(CC)]
  - \( T_1 = \{[a=\text{true}, b=\text{true}], [a=\text{false}, b=\text{false}]\} \) satisfies GACC but not PC. Both assignments to \( a \) and \( b \) yield \( p = \text{true} \).
  - \( T_2 = \{[a=\text{true}, b=\text{false}], [a=\text{false}, b=\text{true}]\} \) would also satisfy GACC but not PC. Both assignments to \( a \) and \( b \) yield \( p = \text{false} \).
- Does GACC subsume PC? Not necessarily.
Correlated Active Clause Coverage (CACC)

- Idea: Correlate $c \land d(c)$ with the truth value of the predicate. 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. $c \land d(c) \land p$
  2. $\neg c \land d(c) \land \neg p$

  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 $a$ 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.
- By definition CACC subsumes GACC [thus CC] but also PC.
GACC vs CACC (example)

<table>
<thead>
<tr>
<th>#</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>( p = a \land (b \leftrightarrow c) )</th>
<th>Satisfies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>( a \land d(a) \land b \land d(b) \land c \land d(c) )</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>( b \land d(b) \land \neg c \land d(c) )</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>( \neg b \land d(b) \land c \land d(c) )</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>( a \land d(a) \land \neg b \land d(b) \land \neg c \land d(c) )</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>( \neg a \land d(a) )</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>( \neg a \land d(a) )</td>
</tr>
</tbody>
</table>

- **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)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>b ↔ c</th>
<th>p = a ∧ (b ↔ c)</th>
<th>Satisfies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>a ∧ d(a) b ∧ d(b) c ∧ d(c)</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>b ∧ d(b) ¬c ∧ d(c)</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>¬b ∧ d(b) c ∧ d(c)</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>a ∧ d(a) ¬b ∧ d(b) ¬c ∧ d(c)</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>¬a ∧ d(a)</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>¬</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>¬</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>¬a ∧ d(a)</td>
</tr>
</tbody>
</table>

- **CACC** can be satisfied by \{#1, #2, #3, #5\}.
- **Exercise 5**: Perform a similar analysis for \(p = a \lor (b \leftrightarrow c)\).
Restricted Active Clause Coverage (RACC)

- As for CACC, but
- For $p \in P$ and $c \in C_p$ include two requirements in TR:
  1. $c \land d(c) \land p$
  2. $\neg c \land d(c) \land \neg p$

that is, $p$ must evaluate to true in one case and false in the other (as in CACC)

- but additionally: the minor clause assignments must be the same in both cases.

- Obs.: RACC subsumes CACC by definition.
- RACC imposes more “uniform” tests, but is also more likely to imply infeasible requirements.
CACC vs RACC (example)

<table>
<thead>
<tr>
<th>#</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>( p = a \land (b \lor c) )</th>
<th>Satisfies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>( a \land d(a) )</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>( a \land d(a) )</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>( a \land d(a) )</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>( \neg a \land d(a) )</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>( \neg a \land d(a) )</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>( \neg a \land d(a) )</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>–</td>
</tr>
</tbody>
</table>

\[ d(a) = b \lor c \]

- **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.
Example: isLeapYear()

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

\(a: y \% 400 == 0\) \(b: y \% 4 == 0\) \(c: y \% 100 != 0\)

\(p: a \lor (b \land c)\)

\(d(a) = \neg b \lor \neg c\) \(d(b) = \neg a \land c\) \(d(c) = \neg a \land b\)

\(\text{TR}(\text{GACC}) = \{(1) a \land (\neg b \lor \neg c),\ (2) \neg a \land (\neg b \lor \neg c),\ (3) b \land \neg a \land c,\ (4) \neg b \land \neg a \land c,\ (5) \neg c \land \neg a \land b\}\)

<table>
<thead>
<tr>
<th>#</th>
<th>y</th>
<th>expected</th>
<th>clause values</th>
<th>covered GACC requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2000</td>
<td>true</td>
<td>a b \neg c</td>
<td>(1) a \land (\neg b \lor \neg c)</td>
</tr>
<tr>
<td>2</td>
<td>2001</td>
<td>false</td>
<td>\neg a \neg b c</td>
<td>(2) \neg a \land (\neg b \lor \neg c)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(4) \neg b \land \neg a \land c</td>
</tr>
<tr>
<td>3</td>
<td>1900</td>
<td>false</td>
<td>\neg a b \neg c</td>
<td>(2) \neg a \land (\neg b \lor \neg c)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(5) \neg c \land \neg a \land b</td>
</tr>
<tr>
<td>4</td>
<td>2004</td>
<td>true</td>
<td>\neg a b c</td>
<td>(3) b \land \neg a \land c</td>
</tr>
</tbody>
</table>
isLeapYear() analysis

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

<table>
<thead>
<tr>
<th>#</th>
<th>y</th>
<th>expected</th>
<th>clause values</th>
<th>covered GACC requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2000</td>
<td>true</td>
<td>a b ¬c</td>
<td>(1) a ∧ (¬b ∨ ¬c)</td>
</tr>
<tr>
<td>2</td>
<td>2001</td>
<td>false</td>
<td>¬a ¬b c</td>
<td>(2) ¬a ∧ (¬b ∨ ¬c)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(4) ¬b ∧ ¬a ∧ c</td>
</tr>
<tr>
<td>3</td>
<td>1900</td>
<td>false</td>
<td>¬a b ¬c</td>
<td>(2) ¬a ∧ (¬b ∨ ¬c)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(5) ¬c ∧ ¬a ∧ b</td>
</tr>
<tr>
<td>4</td>
<td>2004</td>
<td>true</td>
<td>¬a b c</td>
<td>(3) b ∧ ¬a ∧ c</td>
</tr>
</tbody>
</table>

- GACC coverage implies CACC coverage in this case
- RACC is also satisfied: (#1,#4) 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 ∧ b ∧ c, a ∧ ¬b ∧ ¬c, a ∧ ¬b ∧ c, ¬a ∧ ¬b ∧ ¬c
Exercise 6

```java
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)
  6. Test cases that satisfy a) CACC, b) RACC. Are there infeasible requirements?