EE360T/382V Software Testing

khurshid@ece.utexas.edu

February 15, 2020

Overview

Now – Complete Logic coverage

Last time – Continued logic coverage

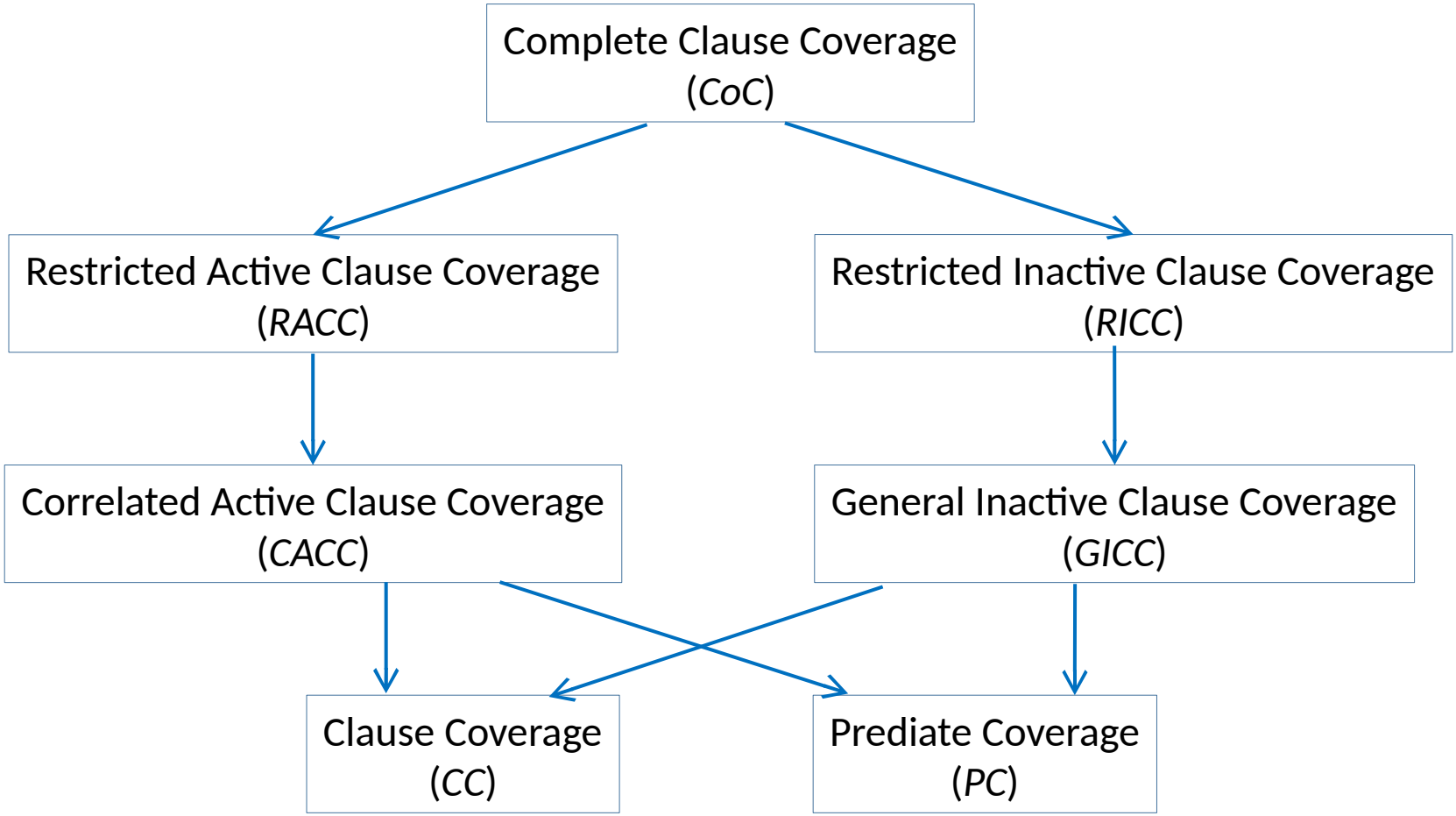
Next time – Input domain partitioning

Recall: Criteria based on structures

The textbook focuses on four kinds of structures to define criteria:

- Graphs (Chapter 2)
  - E.g., control-flow graphs (CFGs)
- Logical expressions (Chapter 3)
  - E.g., if-conditions
- Input domain characterization (Chapter 4)
  - E.g., sorted array
- Syntactic structures (Chapter 5)
  - E.g., mutation

Recall: Subsumption



3.4 Specification-based logic coverage

Formal and informal specs describe expected behaviors

- Can be written in a number of ways
- Usually composed of logical expressions
  - Allow logic coverage criteria to be applied
- E.g., method preconditions and postconditions

Example informal precondition

```
public static int cal(int month1, int day1,
        int month2, int day2, int year) {
    // calculate the number of days between
    // the two given days in the same year
    // precondition: day1 and day2 must be in
    //   the same year
    //   1 <= month1, month2 <= 12
    //   1 <= day1, day2 <= 31
    //   month1 <= month2
    // the range for year: 1 ... 10000

    ...
}
```

Example precondition formalization

```
//   1 <= month1, month2 <= 12
//   1 <= day1, day2 <= 31
//   month1 <= month2
// the range for year: 1 ... 10000

(month1 >= 1 && month1 <= 12 && month2 >= 1 &&
month2 <= 12 && month1 <= month2 && day1 >= 1
&& day1 <= 31 && day2 >= 1 && day2 <= 31 && year
>= 1 && year <= 10000)
```

- Executable as a check, e.g., in an assertion

This predicate has a simple structure

- *Conjunctive normal form*
- For CACC or RACC, set all minor clauses to true
  - With 11 clauses, 12 tests satisfy CACC or RACC (if clauses not mutually dependent)

Another example precondition formalization

Consider a singly-linked acyclic list

- Precondition for all public methods is the class invariant: this.repOk();

```
public boolean repOk(); // representation okay check
```

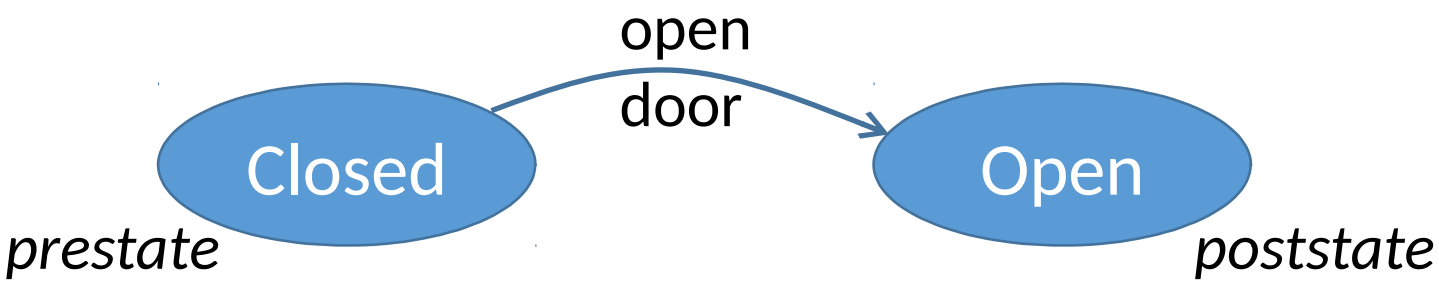
3.5 Logic coverage of finite state machines

A finite state machine (FSM) is a graph where:

- Node – state; and
- Edge – transition from *prestate* to *poststate*
- Transitions may have:
  - Preconditions or guards
    - Conditions that enable transition
  - Triggering events
    - Changes in variable values that cause transitions to be taken

Use predicates from transitions and apply logic coverage criteria

Example: elevator door open transition



Precondition: elevSpeed == 0

Trigger: openButton is pressed

Creating tests w.r.t. transition <*prestate*, *poststate*>

Reachability – find a path from the initial state to the *prestate*, e.g, using DFS

- Need prefix values to reach *prestate*
- Solve predicates on the transitions on the path

Some FSMs may have exit states that must be reached by test paths

- Find a path from *poststate* to an exit state
- Need postfix values to reach the exit state

States can provide test oracles – check actual program state w.r.t. expected *poststate*

Mapping problem – need to map variable values between state machine and program

?/!