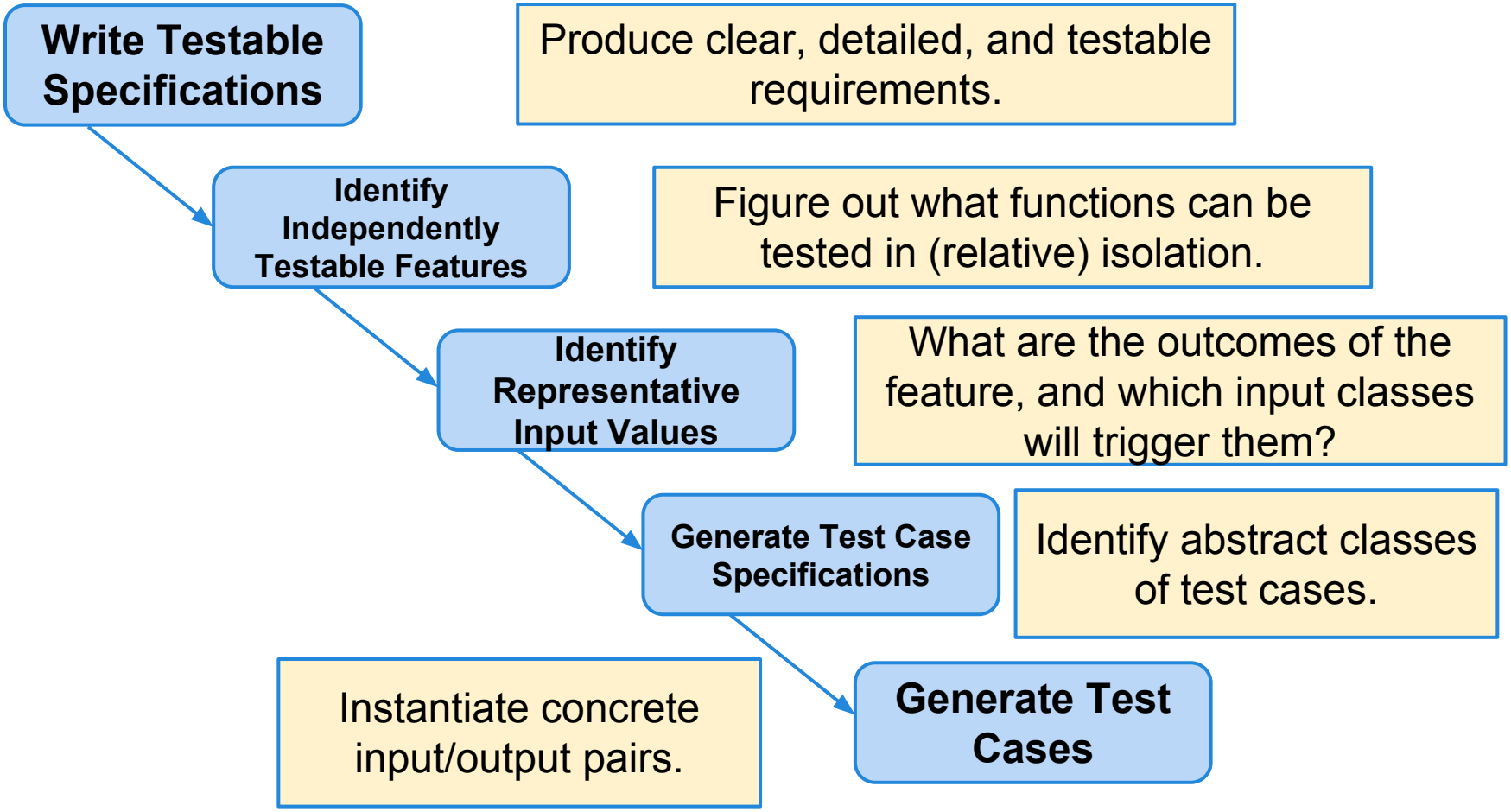
Model-Based Testing:
State Machines

CSCE 747 - Lecture 11 - 02/14/2017

Creating Requirements-Based Tests



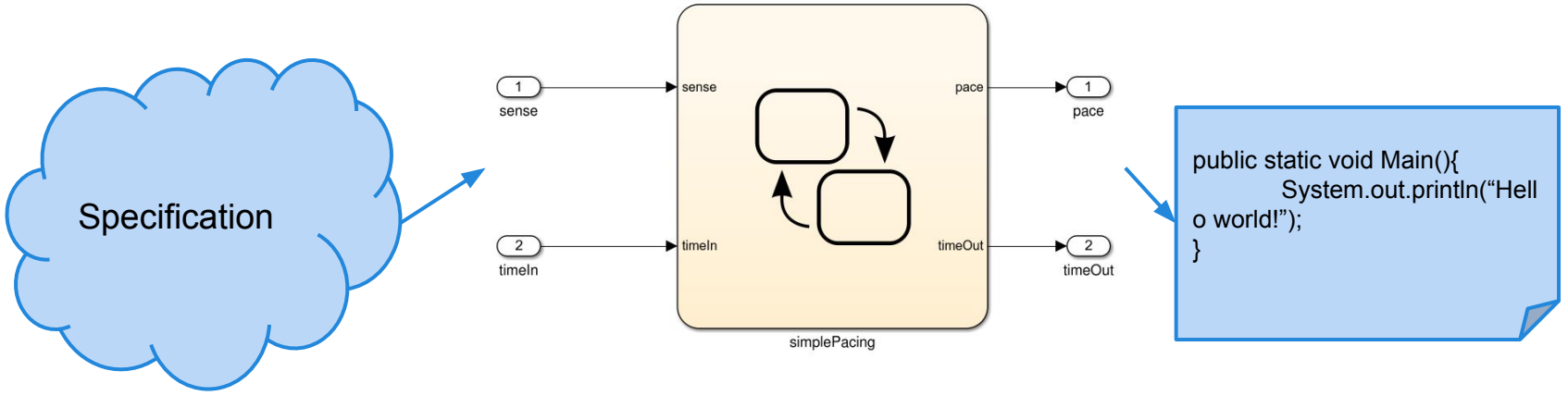
Creating Requirements-Based Tests

- This process is effective for identifying the independent partitions for each input.
  - Leaving us with a large number of test specifications
- Humans must still identify constraints on combinations of input choices and identify a subset of important test specifications.
- An alternative approach - build a model from the specification, and derive tests from the *structure* of the model.

Models

- A **model** is an abstraction of the system being developed.
  - By abstracting away unnecessary details, extremely powerful analyses can be performed.
- Can be extracted from specifications and design plans
  - Illustrate the *intended* behavior of the system.
  - Often take the form of state machines.
    - Events cause the system to react, changing its internal state.

What Can We Do With This Model?



If the model satisfies the specification...

And If the model is well-formed, consistent, and complete.

And If the model accurately represents the program.

… Then we can derive test cases from the model that can be applied to the program. If the model and program do not agree, then there is a fault.

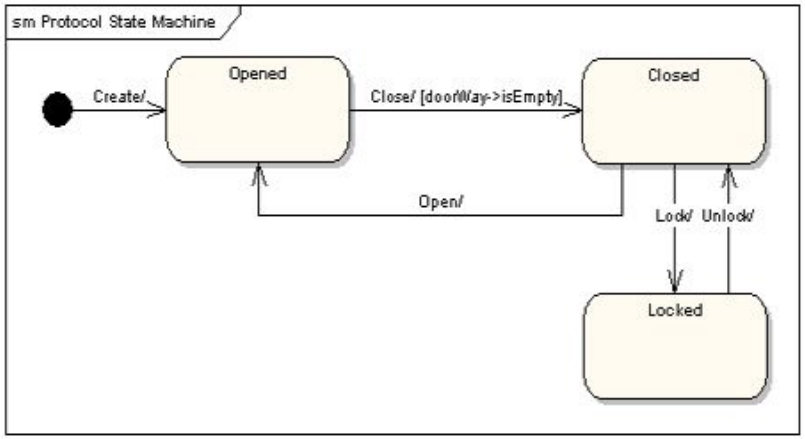
Model-Based Testing

- Models describe the *structure* of the input space.
  - They identify what will happen when types of input are applied to the system.
- That structure can be exploited:
  - Identify input partitions.
  - Identify constraints on inputs.
  - Identify significant input combinations.
- Can derive and satisfy coverage metrics for certain types of models.

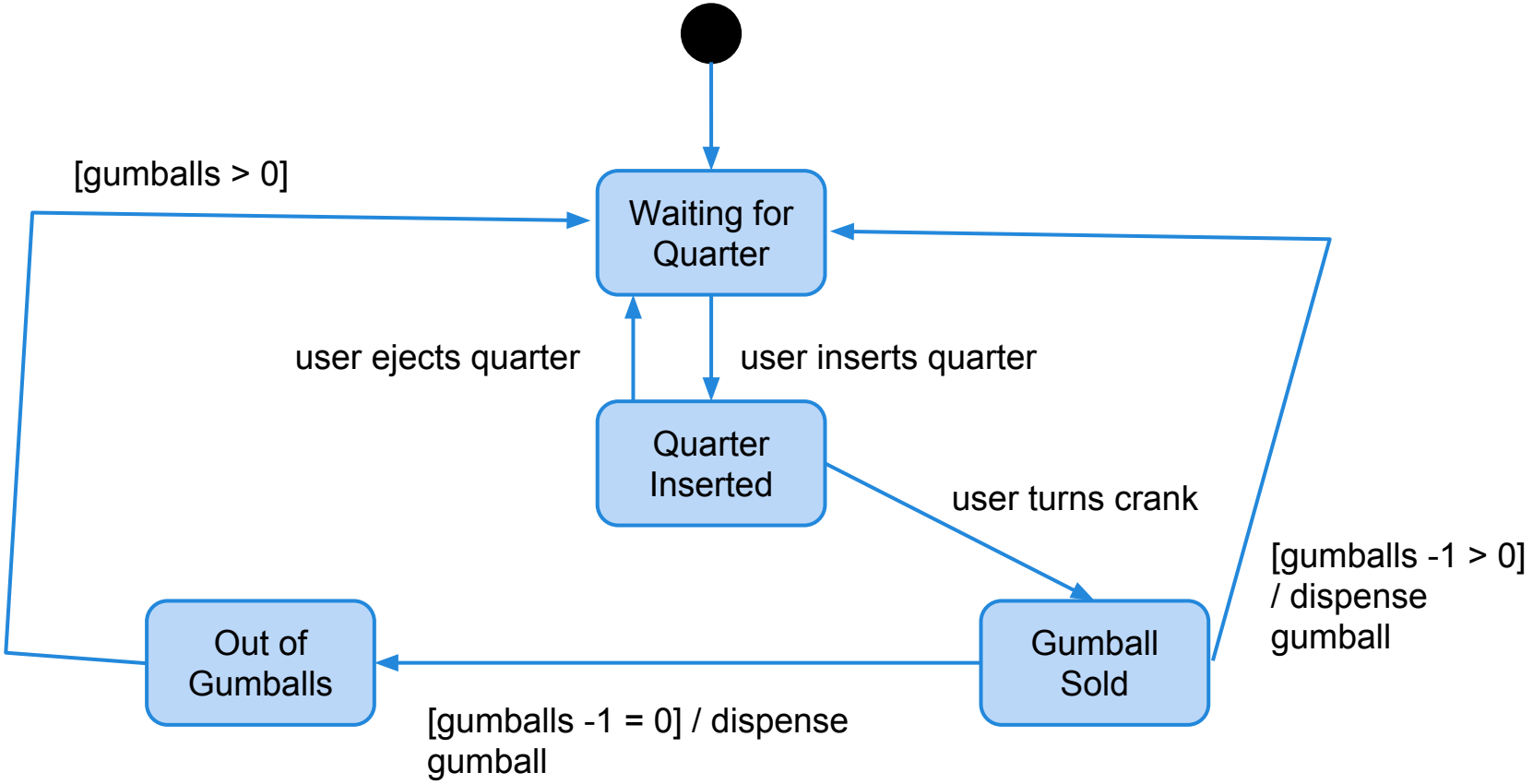
Finite State Machines

Finite State Machines

- A directed graph.
- Nodes represent states
  - An abstract description of the current value of an entity's attributes.
- Edges represent transitions between states.
  - Events cause the state to change.
  - Labeled `event [guard] / activity`
    - `event`: The event that triggered the transition.
    - `guard`: Conditions that must be true to choose a transition.
    - `activity`: Behavior exhibited by the object when this transition is taken.



Example: Gumball Machine



Example: Maintenance

If the product is covered by warranty or maintenance contract, maintenance can be requested through the web site or by bringing the item to a designated maintenance station.

No Maintenance

If the maintenance is requested by web and the customer is a US resident, the item is picked up from the customer. Otherwise, the customer will ship the item.

Request - No Warranty

Waiting for Pick Up

If the product is not covered by warranty or the warranty number is not valid, the item must be brought to a maintenance station. The station informs the customer of the estimated cost. Maintenance starts when the c

Wait for Acceptance

Wait for Returning

Repair at Station

customer does not accept, the item is retur

If the maintenance station cannot solve the problem, the product is sent to the regional headquarters (if in the US) or the main headquarters (otherwise). If the regional headquarters cannot solve the problem, the product is sent to main headquarters.

Repair at Regional HQ

Repair at Main HQ

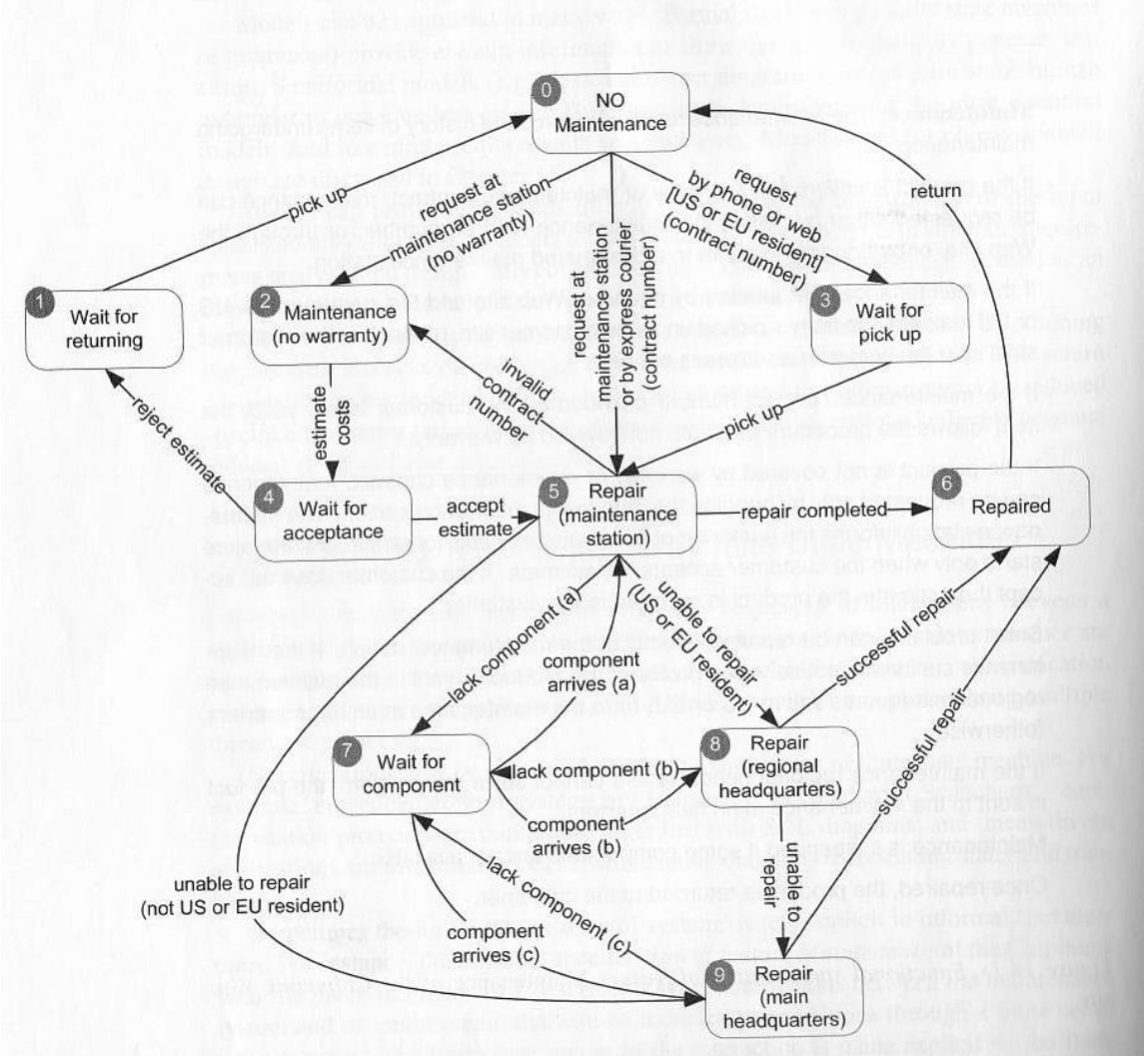
Maintenance is suspended if some components are not available.

Wait for Component

Once repaired, the product is returned to the customer.

Repaired

Example: Maintenance



Finite State Space

- Most systems have an *infinite* number of states.
  - For a communication protocol, there are an infinite number of possible messages that can be passed.
- To model such systems, non-finite components must be ignored or abstracted until the model is finite.
  - For the communication protocol, the message text *doesn't matter*. How it is used does matter.
  - Requires an *abstraction function* to map back to the real system.

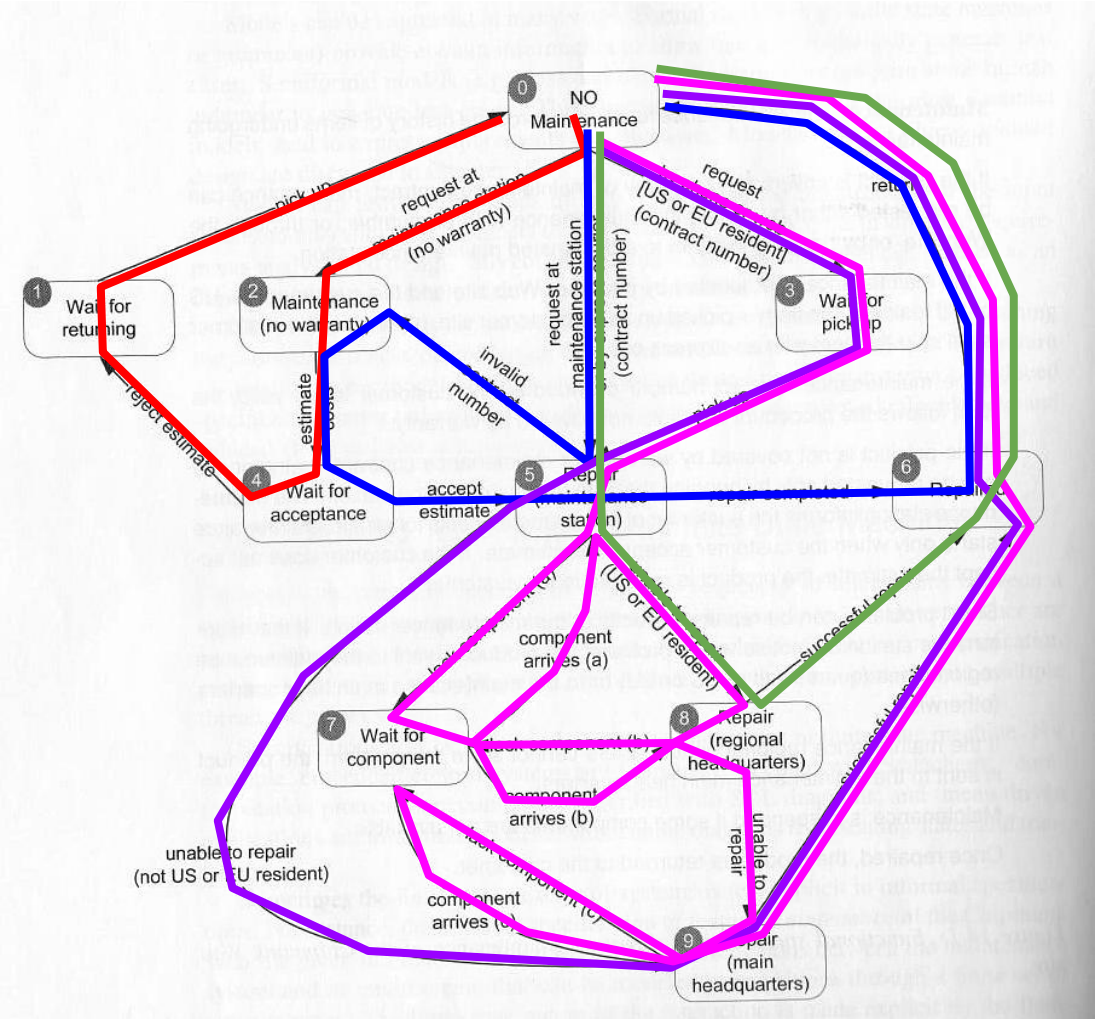
State Coverage

- Each state has been reached by one or more test cases.
- Analog to statement coverage - unless the model has been placed in each state, all faults cannot be revealed.
- Easy to understand and obtain, but low fault-revealing power.
  - The software takes action during the *transitions*, and most states can be reached through multiple transitions.

Transition Coverage

- A transition specifies a pre/post-condition.
  - “If the system is in state S and sees event I, then after reacting to it, the system will be in state T.”
  - A faulty system could violate any of these precondition, postcondition pairs.
- Coverage requires that every transition be covered by one or more test cases.
  - Subsumes state coverage.

Example: Maintenance



- Test cases often given as a list of states or transitions to be covered.
- No “final” states, could achieve transition coverage with one large test case.
  - Smarter to break down FSM and target sections in isolation.

Example Suite:

T1: 0-2-4-1-0

T2: 0-5-2-4-5-6-0

T3: 0-3-5-9-6-0

T4: 0-3-5-7-5-8-7-8-9-7-9-6-0

T5: 0-5-8-6-0

History Sensitivity

- Transition coverage based on assumption that transitions out of a state are independent of transitions into a state.
- Many machines exhibit “history sensitivity”.
  - Transitions available depend on the *history* of previous actions.
  - AKA - the path to the current state.
  - Can be a sign of a bad model design.
    - “wait for component” in example.
  - Path-based metrics can cope with sensitivity.

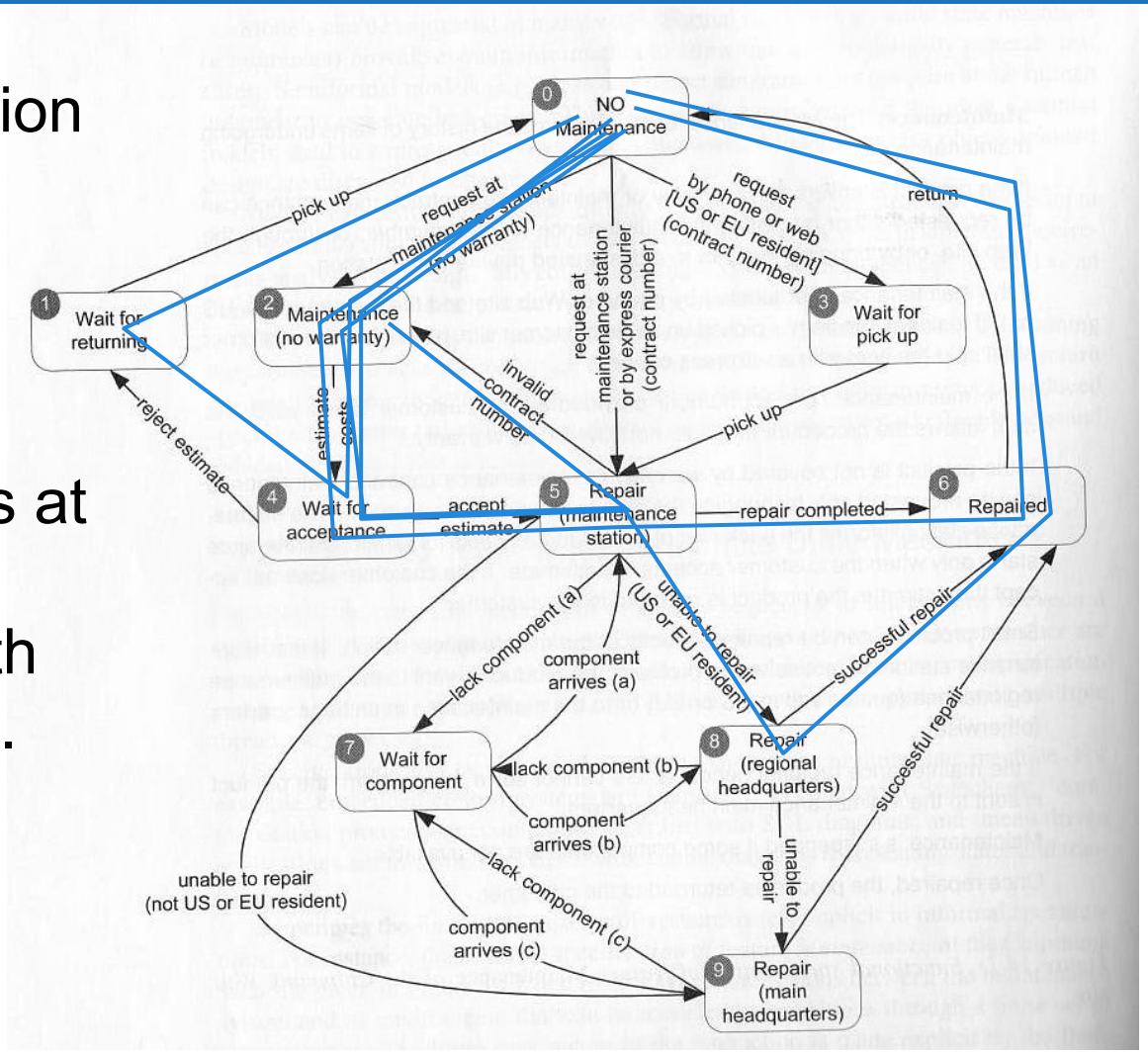
Path Coverage Metrics

- Single State Path Coverage
  - Requires that each subpath that traverses states at most once to be included in a path that is exercised.
- Single Transition Path Coverage
  - Requires that each subpath that traverses a transition at most once to be included in a path that is exercised.
- Boundary Interior Loop Coverage
  - Each distinct loop must be exercised minimum, an intermediate, and a large number of times.

Single State/Transition Path Coverage

Single State/Transition Path Coverage

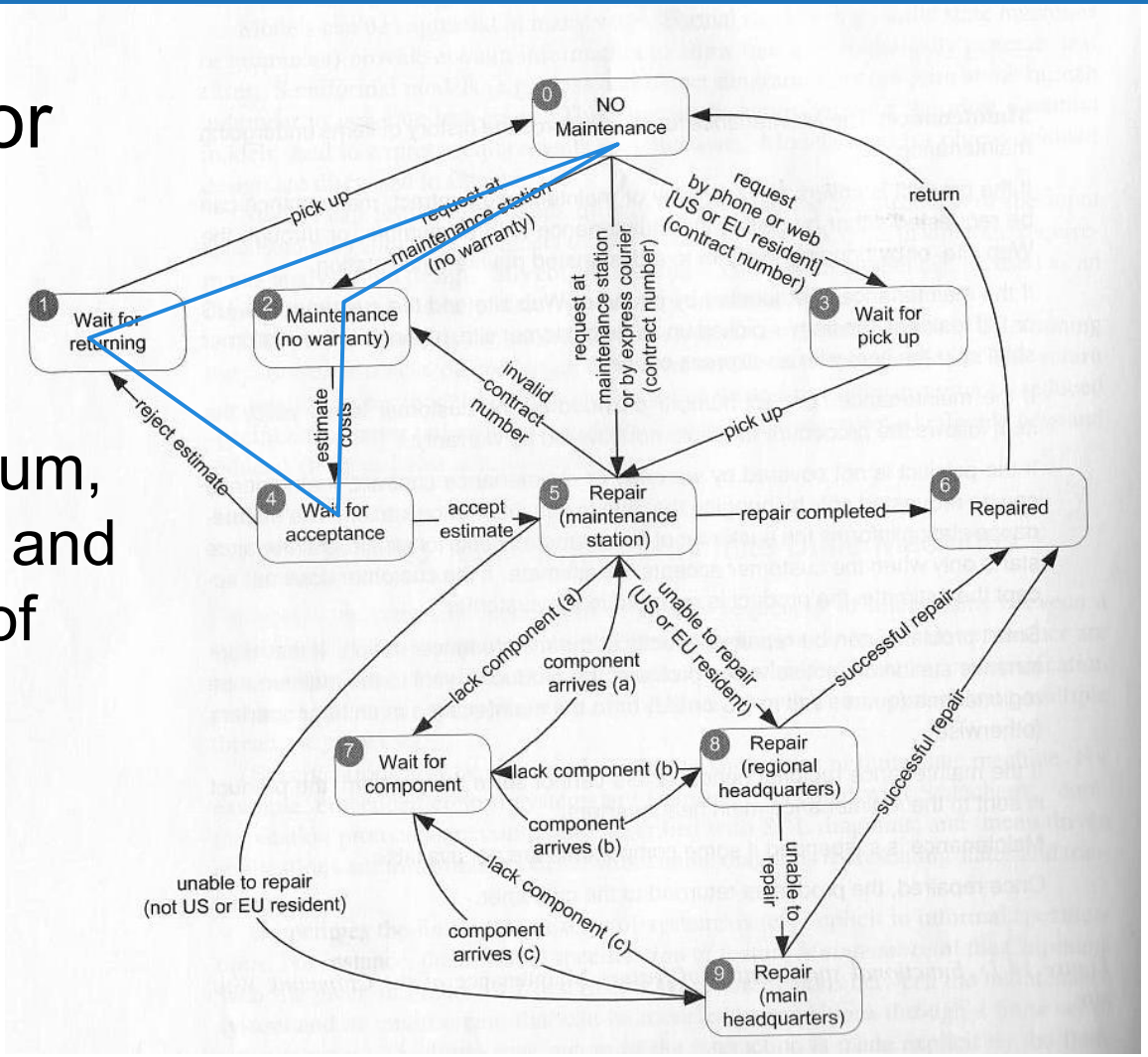
- Requires that each subpath that traverses states/transitions at most once to be included in a path that is exercised.



Boundary Interior Loop Coverage

Boundary Interior Loop Coverage

- Each distinct loop must be exercised minimum, an intermediate, and a large number of times.

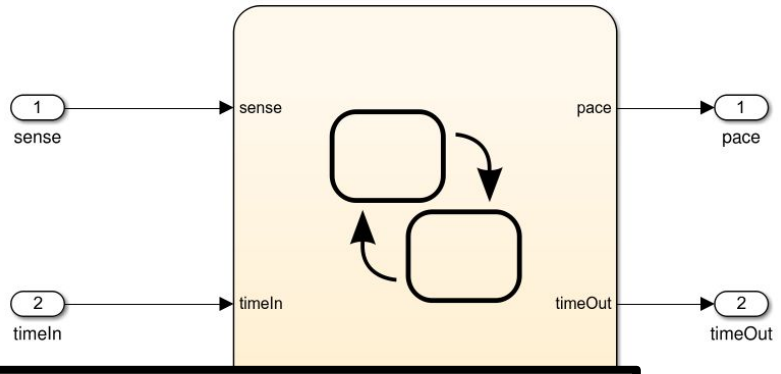


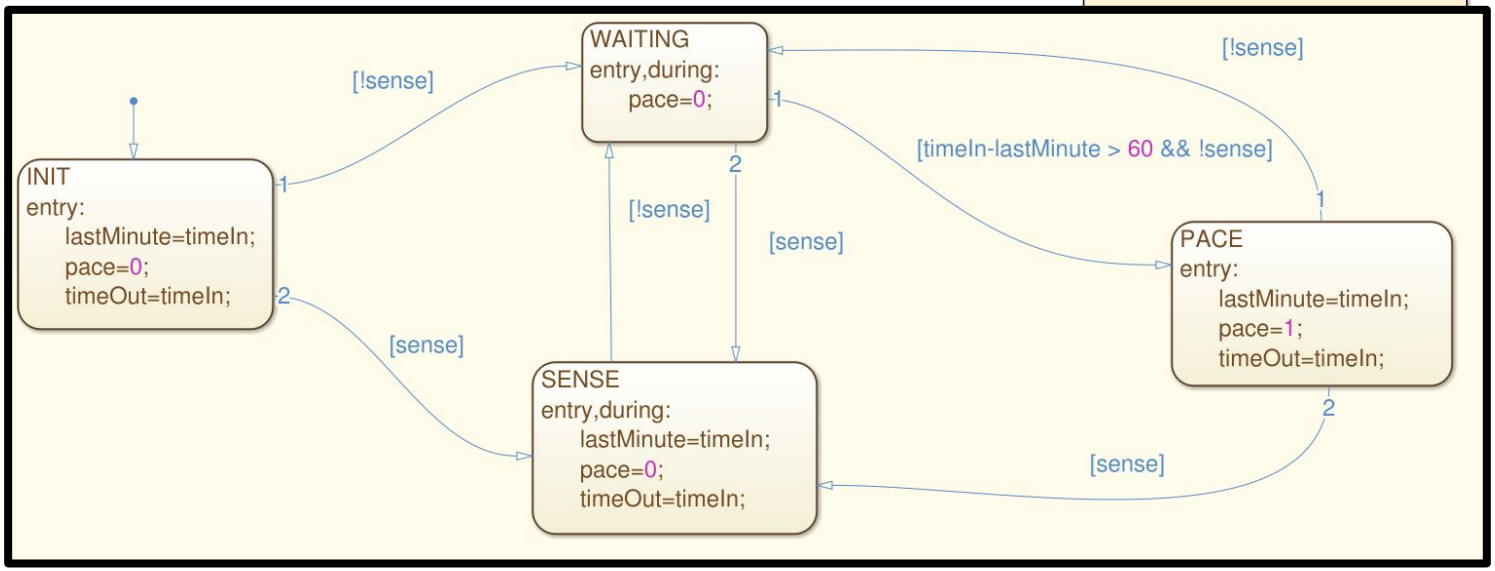
Test Generation

- Test cases created for models can be applied to programs.
  - Events can be translated into method input.
  - System output, when abstracted, should match model output.
- Model coverage is one form of requirements coverage. Tests should be effective for verification.

Activity

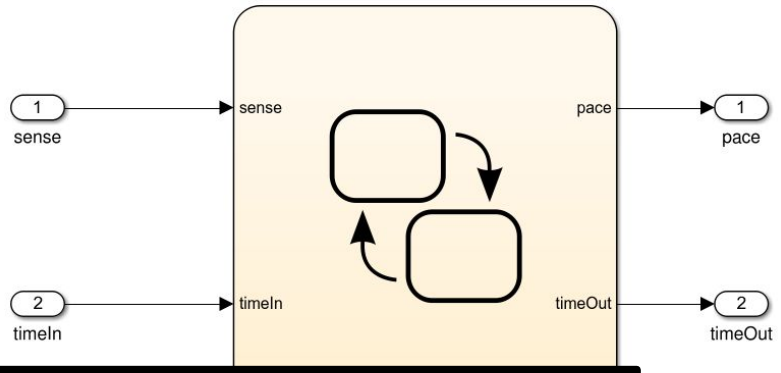
For this model, derive test suites that achieve state and transition coverage.

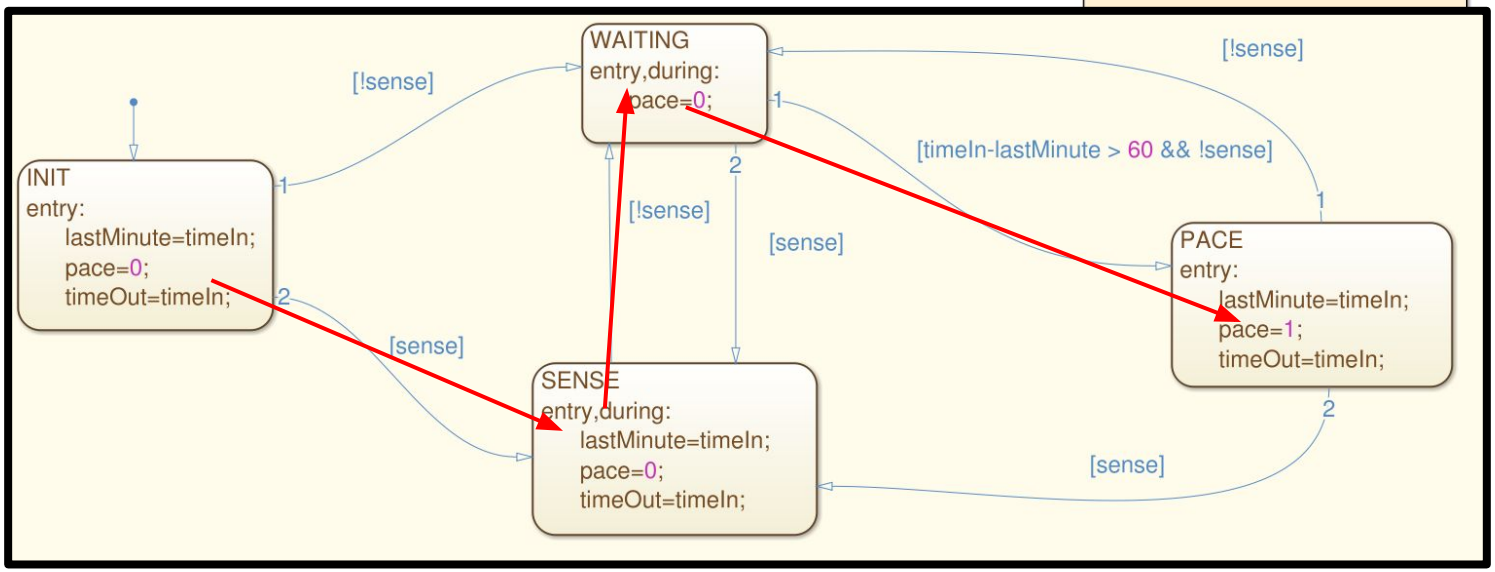




Activity - State Coverage

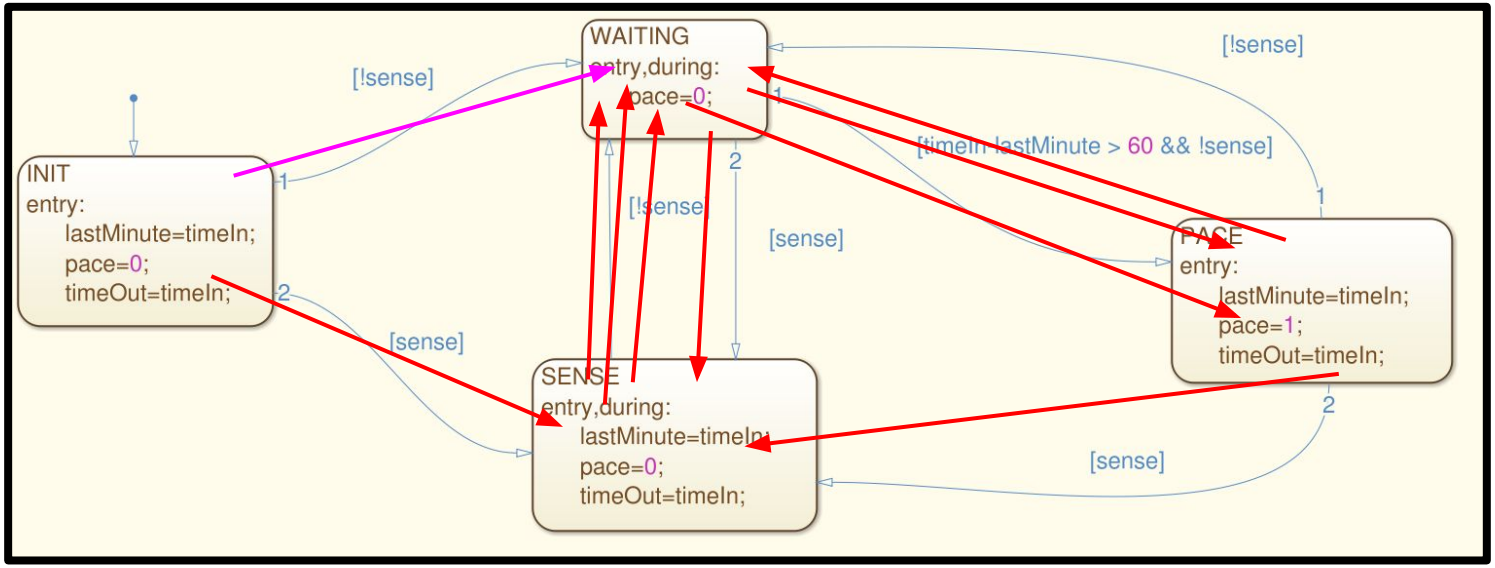
[true,1], [false,2], [false, 65]





Activity - Transition Coverage

1. [true,1], [false,2], [false, 65], [true, 66], [false, 77], [true, 78], [false, 79], [false, 140], [false, 141]
2. [false, 1]



We Have Learned

- If we build models from functional specifications, those models can be used to systematically generate test cases.
- Helps identify important combinations of input to the system.
- Coverage metrics based on the type of model guide test selection.

Next Time

- More Model-Based Testing
  - Decision Structures
  - Grammars

- Homework:
  - Homework 2 - questions?