CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Lecture 13: Model-Based Testing

Gregory Gay
DIT635 - March 3, 2021

Models and Software Analysis

- Before and while building products, engineers analyze models to address design questions.
- Software is no different.
- Software models capture different ways that the software *behaves* during execution.

Behavior Modeling

- **Abstraction -** simplify a problem by identifying and focusing ***only*** on important aspects.
  - Solve a simpler problem, then apply to the big problem.
- A **model** is a simplified representation of an artifact, focusing on one facet of that artifact.
  - The model ignores all other elements of that artifact.

Software Models

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

Model-Driven Development

- Models often created during requirements analysis.
  - Allows refinement of requirement.
  - Can prove that properties hold over model.
    - **Finite State Verification** (next class) - used to analyze requirements, plan development, create test cases.
- Can generate code from models.
- **Can create tests using model.**

Model-Based Testing

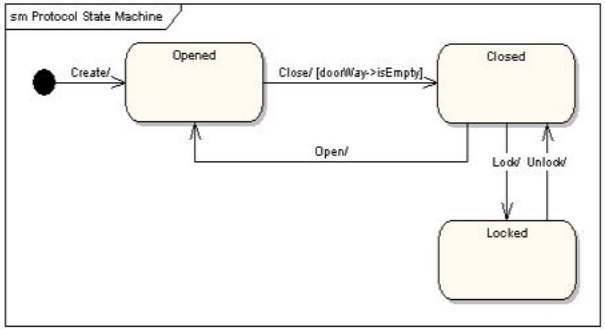
- Models describe what happens input applied to certain functionality.
- Model structure can be exploited:
  - Coverage criteria used to identify important paths.
  - Steps taken to perform functionality in different ways or to get different outcomes.

Finite State Machines

Finite Abstraction

- Program execution is sequence of states transformed by actions.
  - Software “behavior” is a sequence of state -> action -> state transitions.
- The set of all possible behaviors is often infinite.
  - Called the “**state space**” of the program.
  - Models of execution are simplifications of a functionality’s or class’s state space.

Finite State Machines



- 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.

Terminology

- **Event -** Something that happens at a point in time.
  - Operator presses a self-test button on the device.
  - The alarm goes off.
- **Condition** - Boolean property, can have a duration.
  - The fuel level is high.
  - The alarm is on.

Terminology

- **State** - Abstract description of the current value of an entity's attributes.
  - (ex: Not "X=5; Y=10", but "Normal Operating Mode")
  - The controller is in the "self-test" state after the self-test button has been pressed, and leaves it when the rest button has been pressed.
  - The tank is in the "too-low" state when the fuel level is below the set threshold for N seconds.

States, Transitions, and Guards

- States change in response to events (**transition**).
- When multiple responses to an event (multiple transitions) are possible, the choice is guided by the current conditions.
  - Also called the **guards** on a transition.
  - We take the transition that satisfies all guards.

State Transitions

Transitions are labeled in the form:

```
event [guard] / activity
```

- `event`: The event that triggered the transition.
- `guard`: Conditions that must be true to choose this transition.
- `activity`: Behavior exhibited by the object when this transition is taken.

State Transitions

Transitions are labeled in the form:

```
event [guard] / activity
```

- All three are optional.
  - Missing Activity: No output from this transition.
  - Missing Guard: Always take this transition if the event occurs.
  - Missing Event: Take this transition immediately.

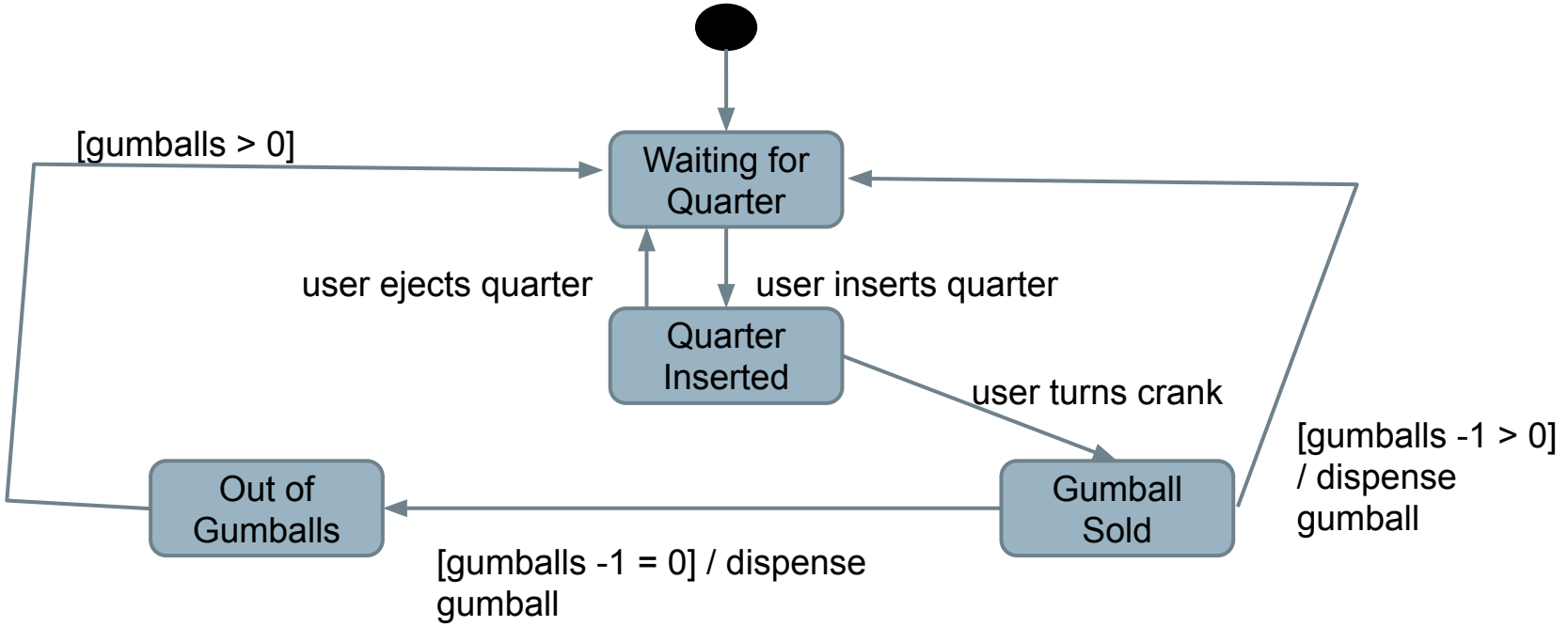
State Transition Examples

Transitions are labeled in the form:

```
event [guard] / activity
```

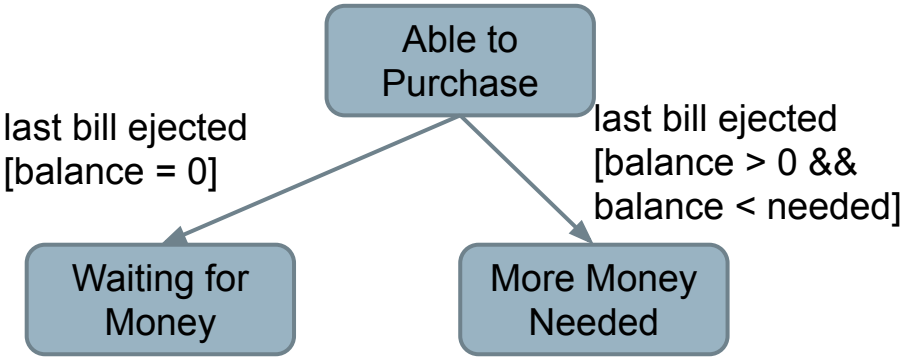
- The controller is in the “self-test” mode after the test button is pressed, and leaves it when the rest button is pressed.
  - Pressing self-test button is an **`event`**.
  - Pressing the rest button is an **`event`**.
- The tank is in the “too-low” state when fuel level is below the threshold for N seconds.
  - Fuel level below threshold for N seconds is a **`guard`**.

Example: Gumball Machine



More on Transitions

Guards must be mutually exclusive



If an event occurs and no transition is valid, then the event is ignored.

last bill ejected [balance > 0 && balance >= needed]

Internal Activities

States can react to events and conditions without transitioning using internal activities.

Typing

entry / highlight all
exit / update field
character entered / add to field
help requested [verbose] / open help page
help requested [minimal] / update status bar

- Special events: **entry** and **exit**.
- Other activities occur until a transition occurs.
  - On each “time step”.
  - Entry and exit not re-triggered without a self-transition.

Example: Maintenance Tracking

- Customers send in products for maintenance.
- Maintenance tracking function notes current stage of the maintenance process for each customer.
  - Path through process determined by a set of rules.
  - States = stages of process. Transitions shows paths through process.
- **Model only what software tracks and controls!**

Example: Maintenance Tracking

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

Waiting for Pick Up

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

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 customer accepts the estimate. If the customer does not accept, the item is returned.

Wait for Acceptance

Wait for Returning

Example: Maintenance Tracking

Repair at Station

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

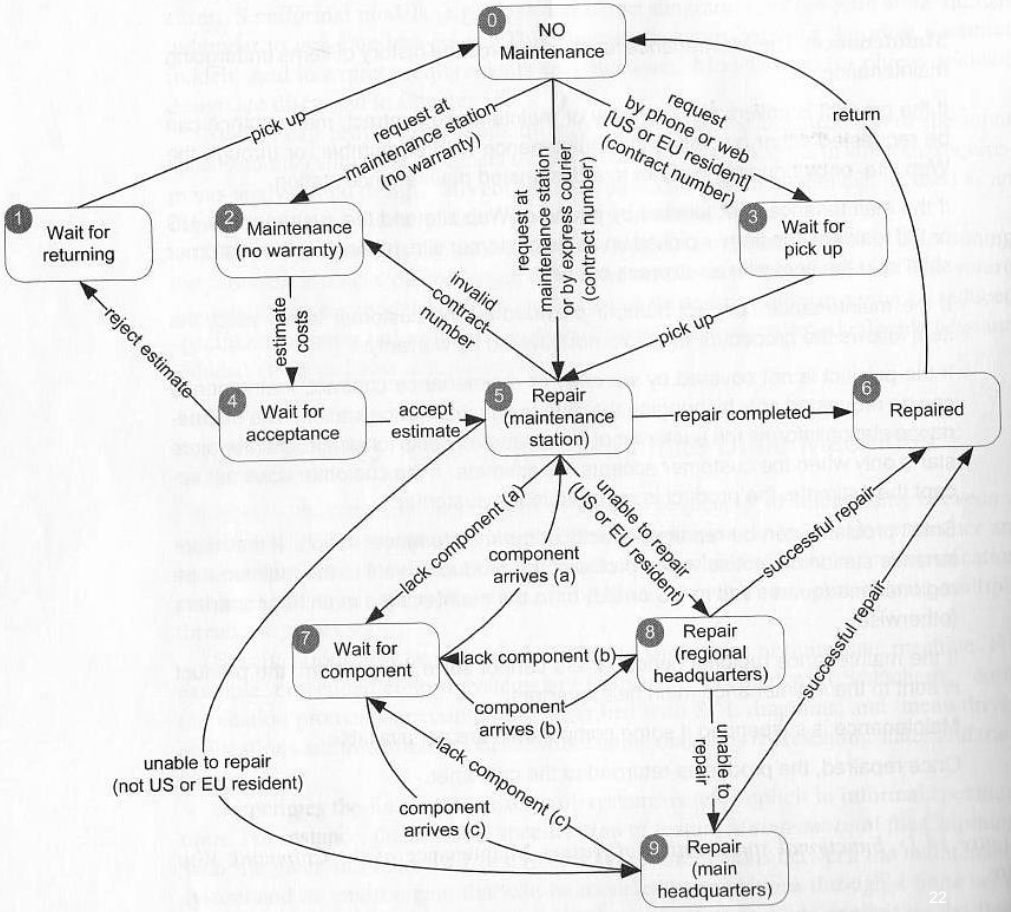
Maintenance is suspended if some components are not available.

Wait for Component

Once repaired, the product is returned to the customer.

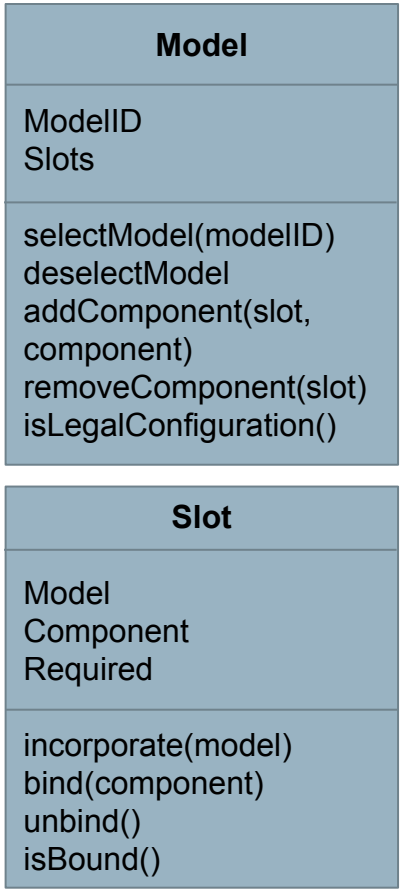
Repaired

Example: Maintenance Tracking



Example - Computer Model

- Many classes have stateful behavior.
  - States are based on the class variables and how they change behavior.
  - Transitions = method calls
  - Derive model from class and create tests.
- Ex: We sell computers on our website. Model class represents a model of computer.

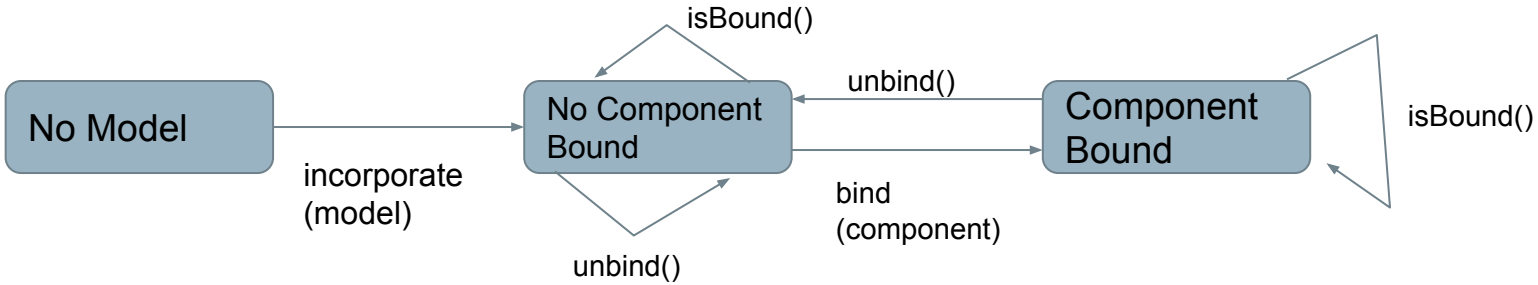


Slot Specification

Slot represents a configuration choice in all instances of a particular model of computer. A given model may have zero or more slots, each of which is marked as required or optional. If a slot is marked as required, it must be bound to a suitable component in all legal configurations. Slot offers the following services:

- **Incorporate:** Make a slot part of a model, and mark it as either required or optional. All instances of a model incorporate the same slots.
- **Bind:** Associate a compatible component with a slot.
- **Unbind:** The unbind operation breaks the binding of a component to a slot, reversing the effect of a previous bind operation.
- **IsBound**: Returns true if a component is currently bound to a slot, or false if the slot is currently empty.

Slot State Machine



- Do not derive too many states.
  - Map variables to abstract values, not a state for each possible combination of values.
- Model how a method affects a class.
  - States only need to capture interactions between methods and the class state.

Example - Model

Model represents the current configuration of a model of computer.

- A given model may have zero or more slots, each of which is marked as required or optional.
- Each slot may contain a single component.
- To be a legal model, the model ID must exist in the ModelDB, each slot marked as required must be filled, the configuration must match that of the ModelDB entry for the model ID, and the optional components must match those allowed for that model in the ModelDB.

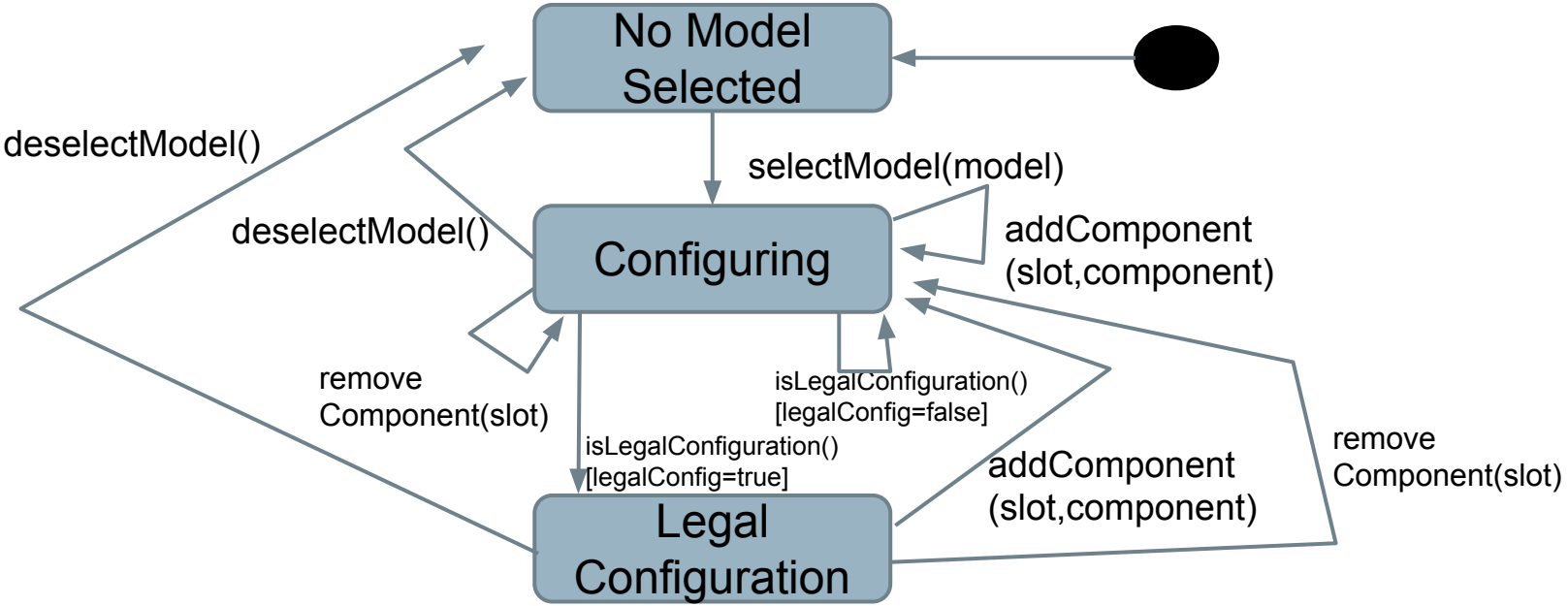
Example - Model

- **selectModel(modelID)**: Sets the model ID to the value passed in, as long as the model ID is set to “no model selected”. A model ID must be set before any other services are requested.
- **deselectModel():** Sets the model ID to “no model selected”. If the configuration was previously judged to be legal, it is no longer legal.
- **addComponent(slot, component):** Adds the selected component to the selected slot. If the configuration was previously judged to be legal, it is no longer legal.
- **removeComponent(slot)**: Removes the selected component to the selected slot. If the configuration was previously judged to be legal, it is no longer legal.
- **isLegalConfiguration():** Compares the current configuration to the entry in ModelDB. If the configuration is valid, the Model’s isLegal field is set to “true”.

Choosing States

- What does the class represent?
  - In this case: a computer model.
- What causes method results to differ?
  - Whether the model is legal or illegal.
- Can the class be in any other states?
  - We may not have set the model yet. We could still be making decisions and have not determined legality.

Choosing Transitions and Initial State



Activity - Secret Panel Controller

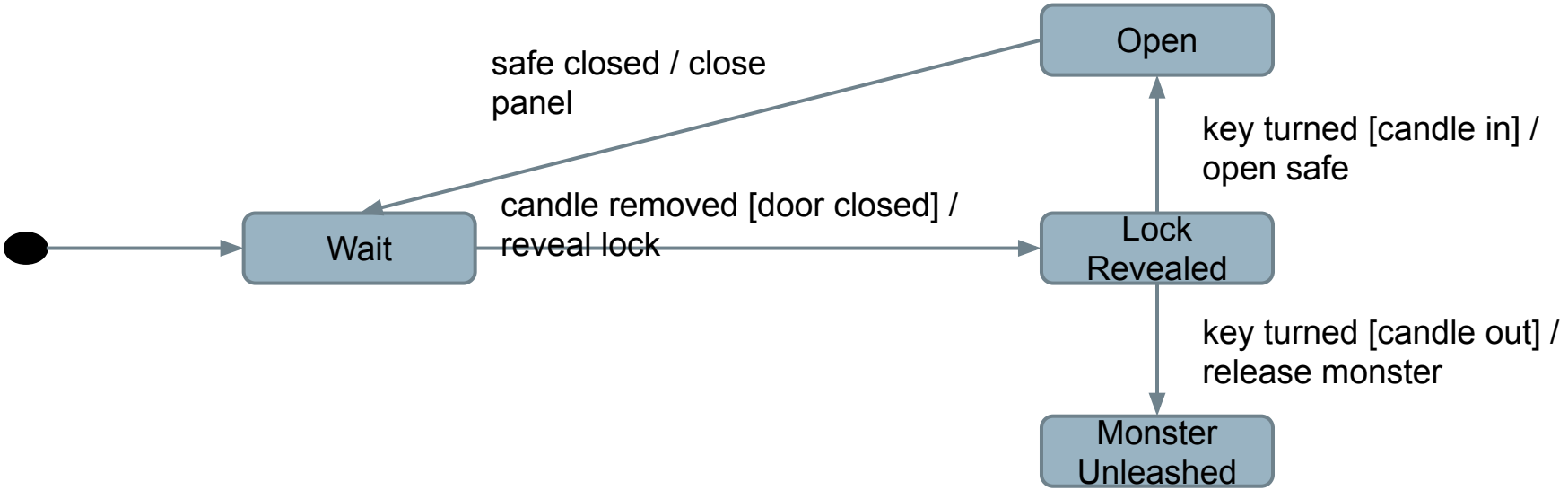
You must design a state machine for the controller of a secret panel in Dracula's castle.

Dracula wants to keep his valuables in a safe that's hard to find. So, to reveal the lock to the safe, Dracula must remove a strategic candle from its holder. This will reveal the lock only if the door is closed. Once Dracula can see the lock, he can insert his key to open the safe. For extra safety, the safe can only be opened if he replaces the candle first. If someone attempts to open the safe without replacing the candle, a monster is unleashed.

Take a short break if needed.

https://bit.ly/3bEfc0J (Problem 1)

Activity Solution



Model Coverage Criteria

Test Creation

- Test cases created from models can be applied to the real program.
  - Events translated into method/api calls.
  - Output, when abstracted, should match model output.
- Model coverage is one form of requirements coverage. Tests should be effective for verification.
  - Exercises stateful behavior thoroughly.
  - Coverage criteria based on states, transitions, paths.

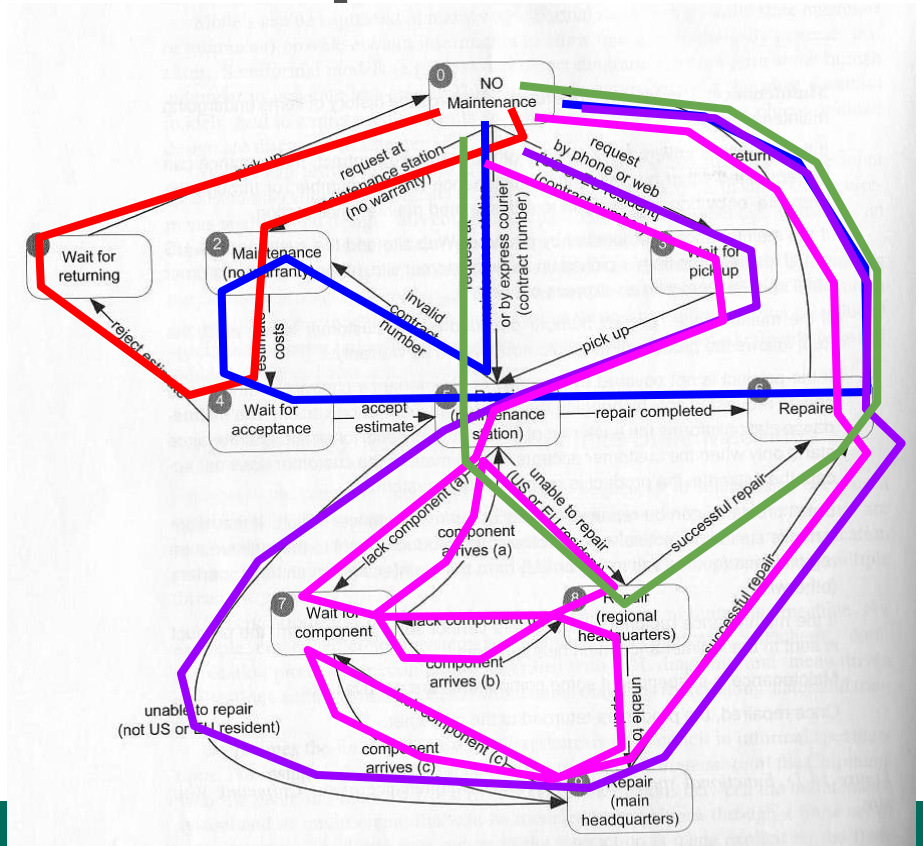
State Coverage

- **Each state must be reached by test cases.**
  - Like statement coverage - unless model has been placed in each state, faults cannot be revealed.
  - **Num. of Covered States / Number of States**
- Easy to understand and obtain, but low fault-revealing power.
  - The software takes action during 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.
  - **Num. Covered Transitions / Number of Transitions**

Example: Maintenance



- If no “final” states, we could achieve transition coverage with one large test case.
  - Smarter to target sections in different test cases.

Example Paths:

T1: request w/ no warranty (0->2) - estimate costs (2->4) - reject (4->1) - pick up (1->0)

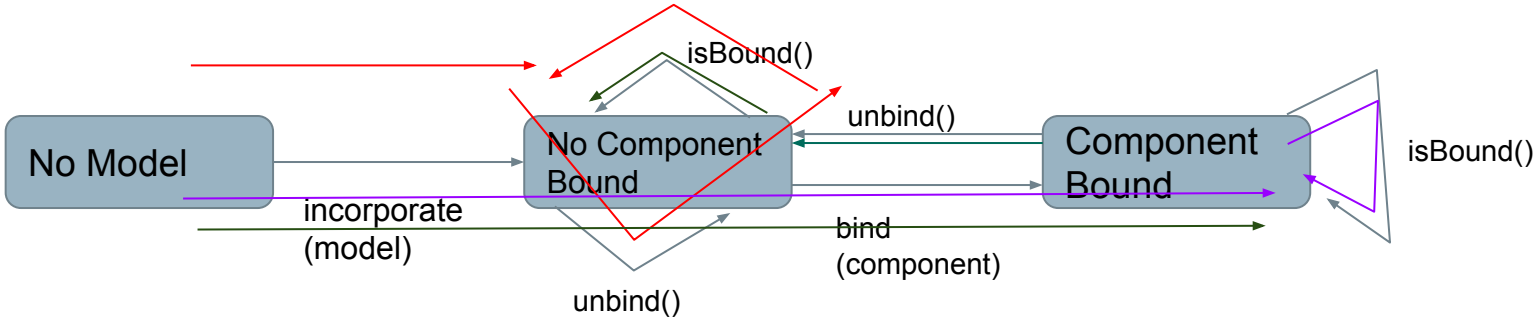
T2: 0->5->2->4->5->6->0

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

Example - Slot

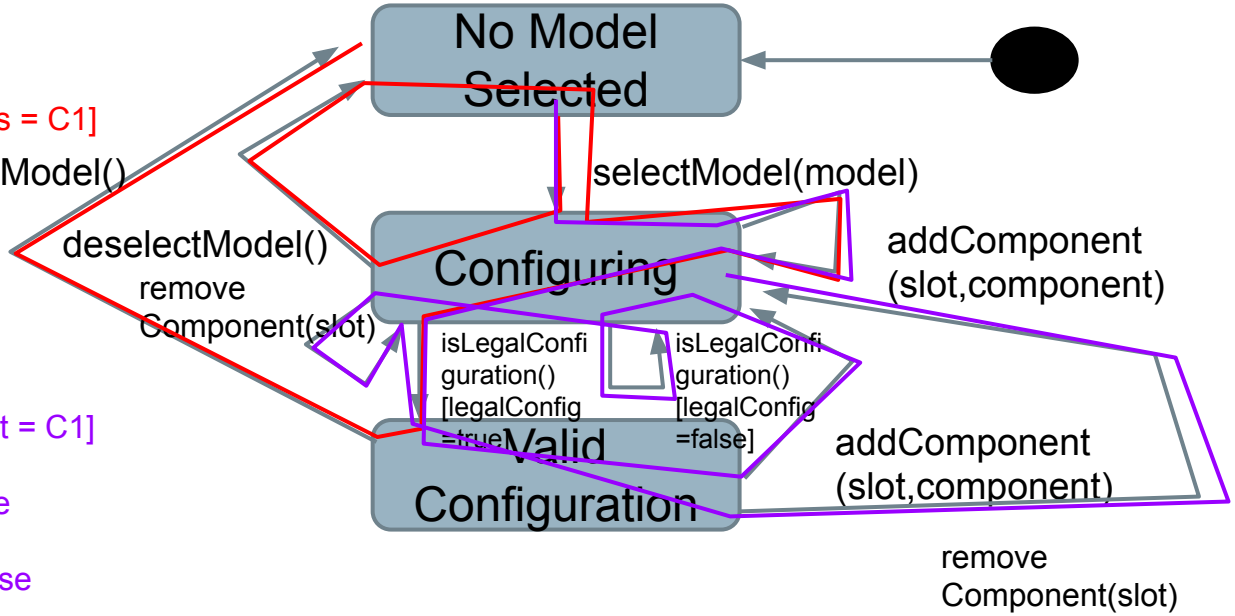


- Tests are series of method calls.
  - incorporate(model), isBound(), unbind()
  - incorporate(model), bind(component), isBound()
  - incorporate(model), bind(component), unbind(), isBound()

Example - Model

TC1:
selectModel(M1) [M1, 1 slots = C1]
deselectModel()
selectModel(M1)
addComponent(S1,C1)
isLegalConfiguration() //true
deselectModel()

TC2:
selectModel(M1) [M1, 1 slot = C1]
addComponent(S1,C1)
isLegalConfiguration() //true
addComponent(S2,C2)
isLegalConfiguration() // false
removeComponent(S2)
isLegalConfiguration() // true
removeComponent(S1)



Path Coverage Criteria

- Transition coverage based on assumption that transitions are independent.
- Many machines exhibit “history sensitivity”.
  - Transitions available depend on path taken.
    - “wait for component” in Maintenance Tracking example.
- Path-based metrics can cope with sensitivity.

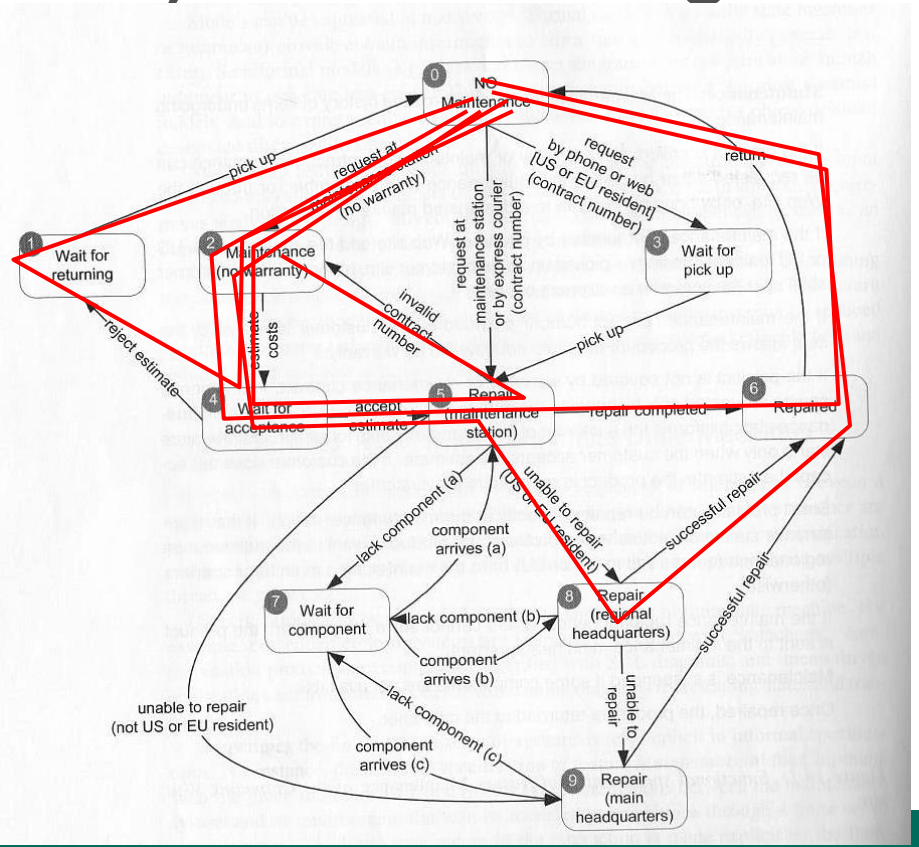
Path Coverage Metrics

- 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 (or Transition) Path Coverage

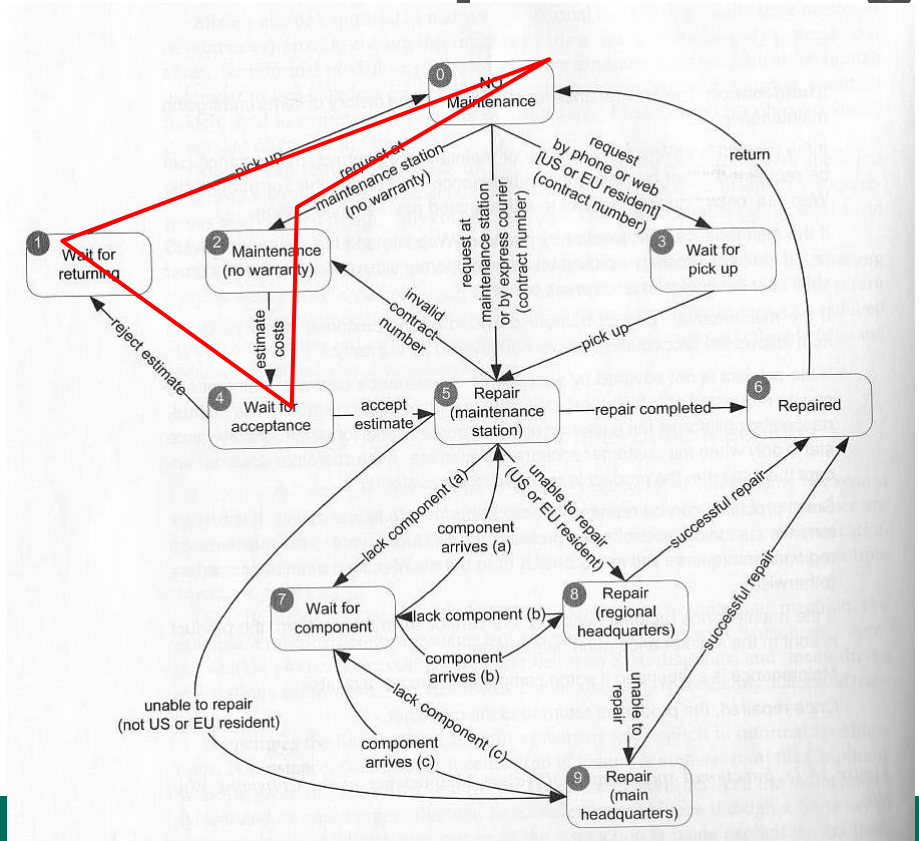
- Each subpath that traverses a state (or transition) **at most once** must be exercised.



Boundary Interior Loop Coverage

Boundary Interior Loop Coverage

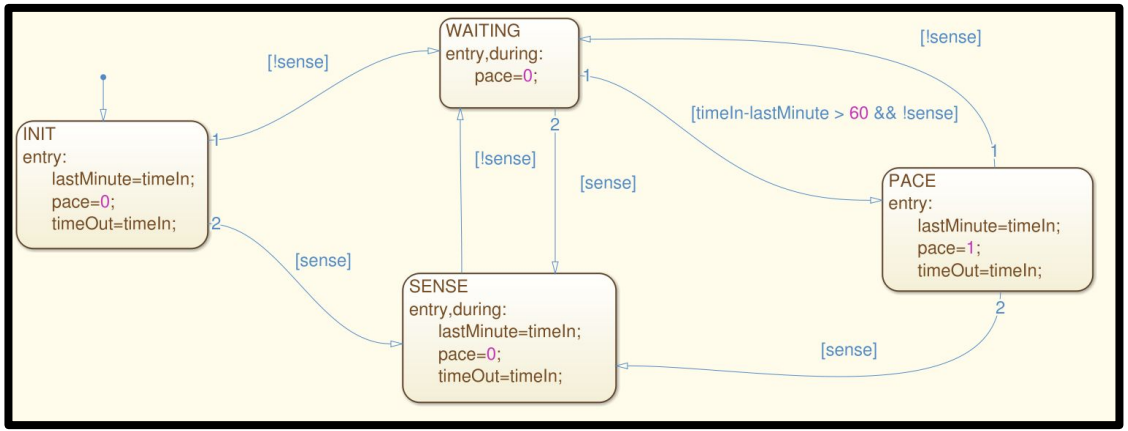
- Each loop must be exercised 1, 2, N times.
- (N = some higher number)

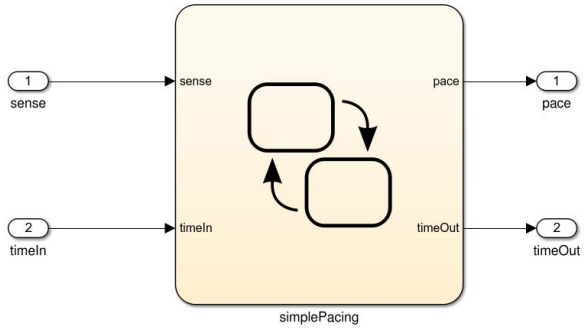


Activity

https://bit.ly/3bEfc0J (Problem 2)

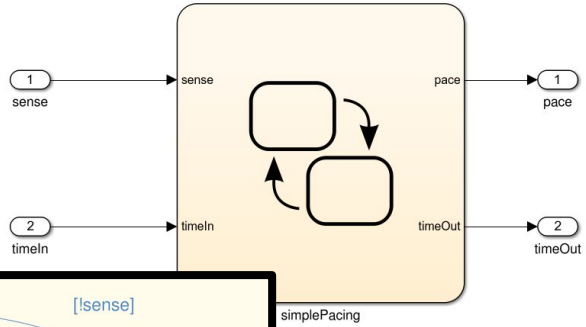
For this model, derive test suites that achieve state and transition coverage.

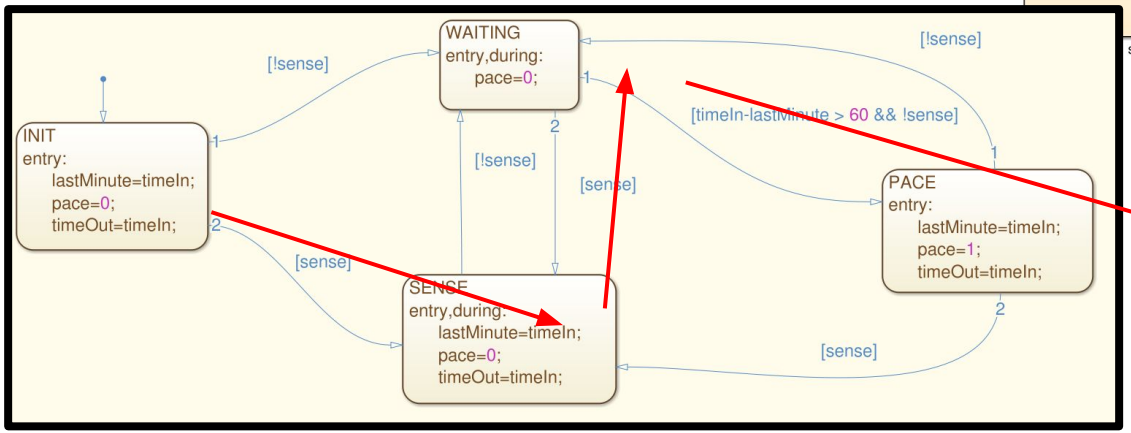




Activity - State Coverage

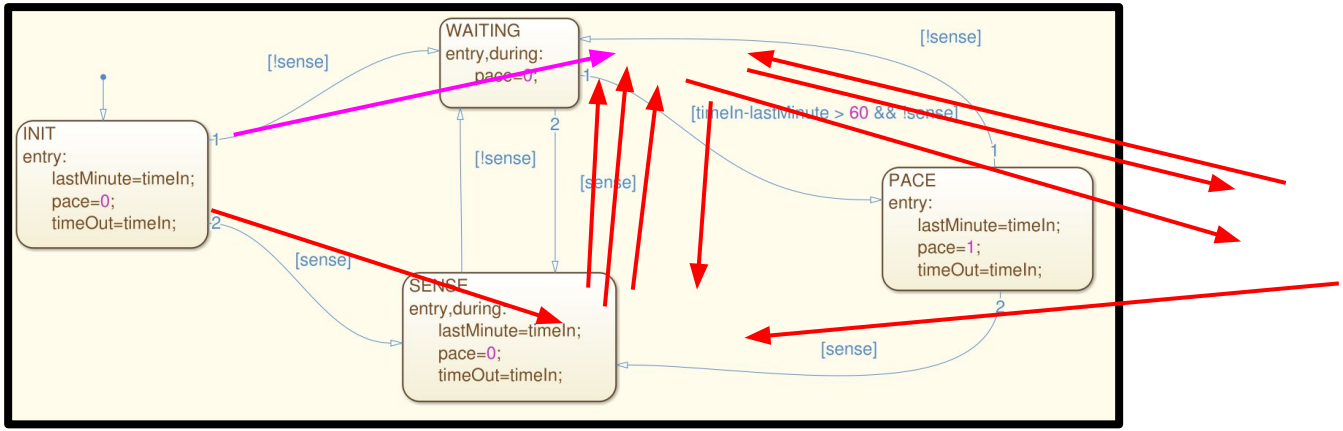
[true,1], [false,2], [false, 65]





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

- Models can be used to systematically create tests.
  - Models have structure. We can exploit that structure.
  - Exercises stateful behavior of a class or functionality.
- Helps identify important input.
- State machines model expected behavior.
  - Cover states, transitions, non-looping paths, loops.
  - Can also verify properties over models as part of verification (next class).

Next Time

- Finite State Verification
  - Optional Reading - Pezze and Young, Chapter 8
- Assignment 3
  - Due Sunday, March 14
  - Questions?

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY