DD2459 Software Reliability
Lecture 5

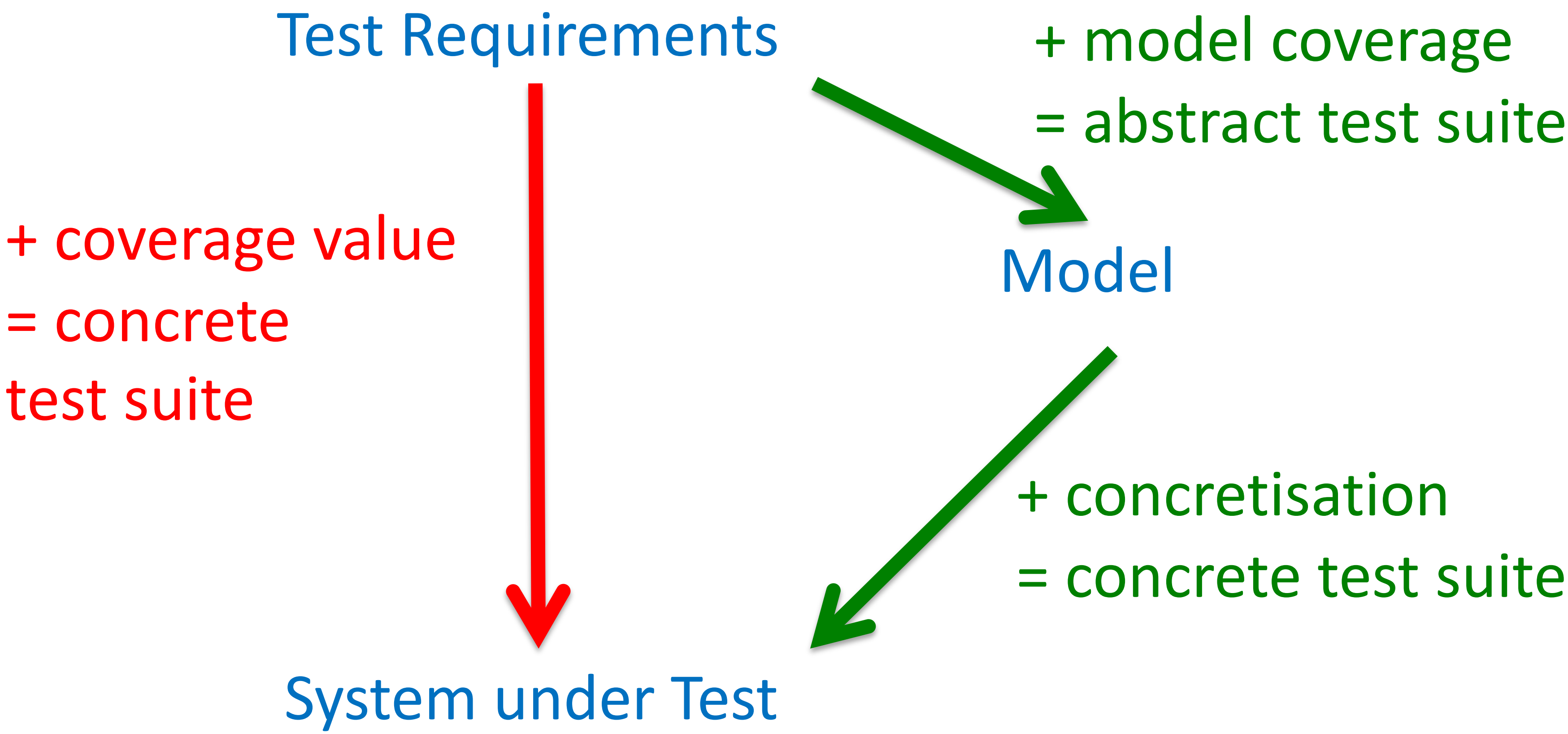
Introduction to
Model-Based Testing (MBT)

(see Amman and Offut,
Chapter 2, Section 2.5)

Part 1. Overview and Principles

What is Model-based Testing?

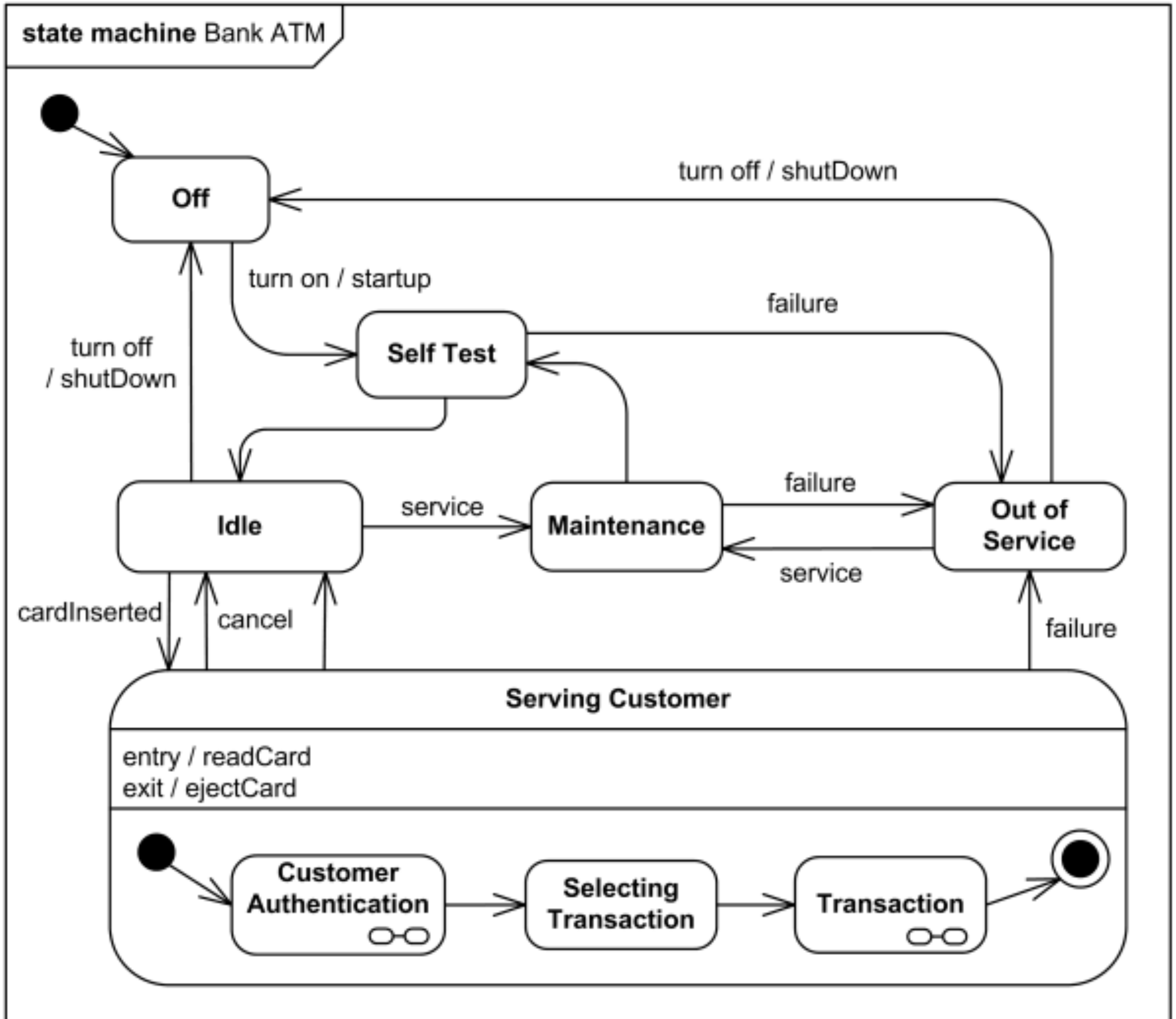
Traditional Code Testing vs. Model-based Testing



What models to use?

- To solve the oracle problem we basically need executable *dynamic models*
  - Statecharts
  - Sequence diagrams (use cases)
  - Executable code

ATM Model


state machine Bank ATM
Off
turn off / shutDown
turn on / startup
failure
Self Test
turn off
/ shutDown
Idle
service
Maintenance
failure
Out of
Service
service
cardInserted
cancel
failure
Serving Customer
entry / readCard
exit / ejectCard
Customer
Authentication
Selecting
Transaction
Transaction

MBT Advantages

- Model need only reproduce some features of a system under test
- Simplification of code (abstraction)
- Select relevant model features! (What to test?)
- False positives and negatives?
- Quicker and easier to generate tests
- Use graph coverage
- Test generation can be automated
- Tool support (Conformiq, Spec Explorer …)

Solving the Oracle Problem?

- Model can determine test verdicts
- AKA. conformance testing (a “golden model”)
- Verdict construction can be reduced to equality or membership test.

$$x = y \quad \text{or} \quad x \in S$$

- Needs exact synchronization between model and code otherwise false positives/negatives
- Difficult with legacy code (no model) and agile development (no time).

Conformance Testing

- **Claim**: There is a sense in which conformance testing just pushes the testing problem elsewhere
- **Why?**: How do we validate the model itself?
  - Simulation?
  - Testing? (systematic simulation?)
  - Formal verification? (too big or complex?)

Model-Based Testing
in practise

1. Take a system model M e.g. statechart
2. Take a coverage model C, e.g:
   2.1 Node coverage
   2.2. Edge coverage
3. Construct test cases to reach x% coverage of M
   3.1. Manually
   3.2. Constraint solver (added value of a tool)
4. Translate test cases into scripts, run, record, and compare outcomes with model M

Leading MBT Tools

We are aware of 30+ MBT tools.

Leading tools at present are:

- Simulink Design Verifier

  (visual Simulink model and cause-effect graphs)
- SpecExplorer - 2013

  (text ASM model and spec#)
- GraphWalker (visual)

  (FSM model and A* algorithm)
- Conformiq (visual)

  (UML Statecharts + glass box coverage models)

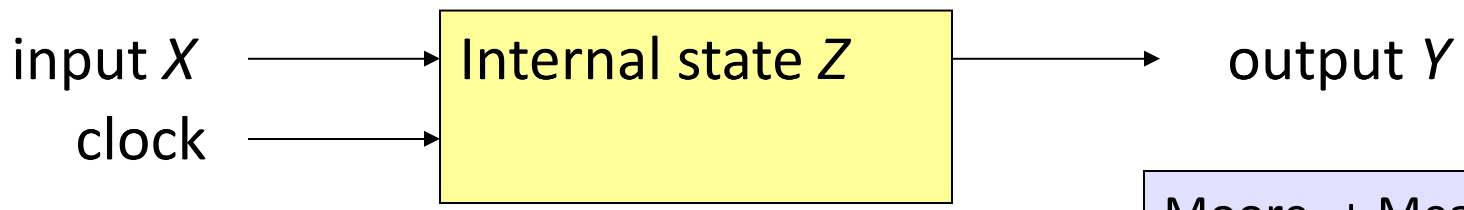
When to use MBT?

- Model-based testing can be conducted as part of model-based development
- Model-based development = describing software using accurate modeling languages e.g. UML

UML Statecharts

- A ***UML statechart***, is an object-based variant of Harel's *statechart* language.
- Statecharts overcome limitations of finite state machines, without losing benefits.
- Combine aspects of Moore and Mealy machines
- New concepts:
  - Hierarchically nested states
  - Orthogonal regions
  - Extended actions
  - History

Moore and Mealy Automata

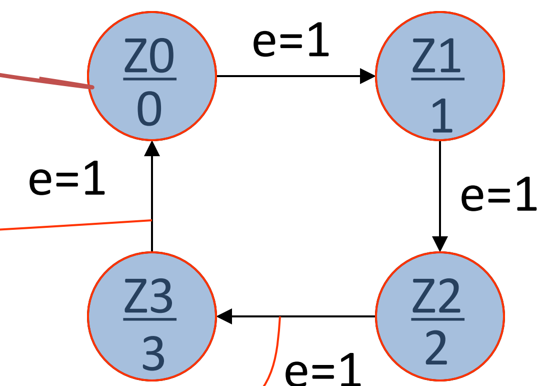


Next state Z^+ computed by function δ

Output computed by function λ

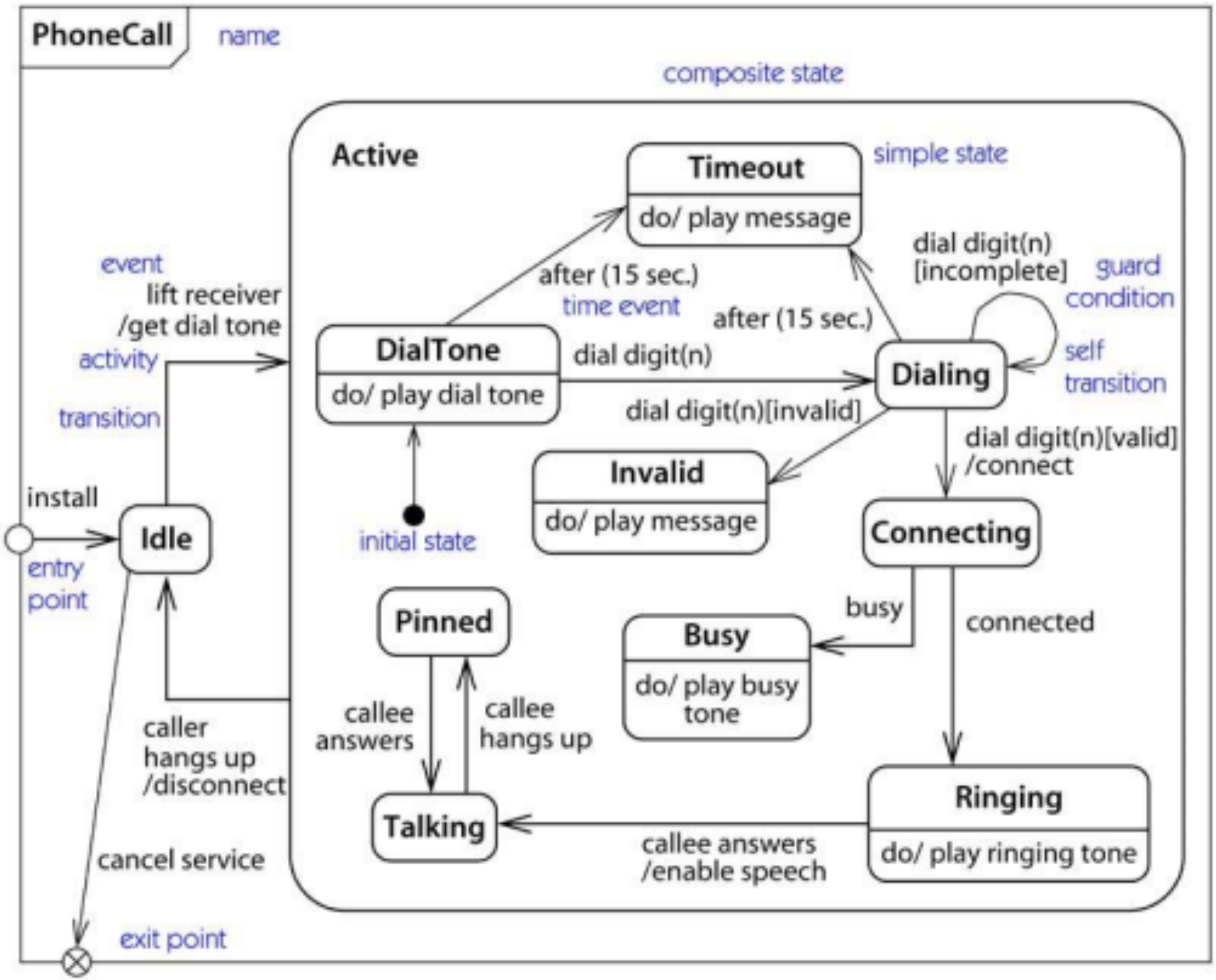
Moore- + Mealy automata=finite state machines (FSMs)

- Moore-automata:
  $Y = \lambda\,(Z);\quad Z^+ = \delta\,(X, Z)$
- Mealy-automata
  $Y = \lambda\,(X, Z);\quad Z^+ = \delta\,(X, Z)$



UML 2.4

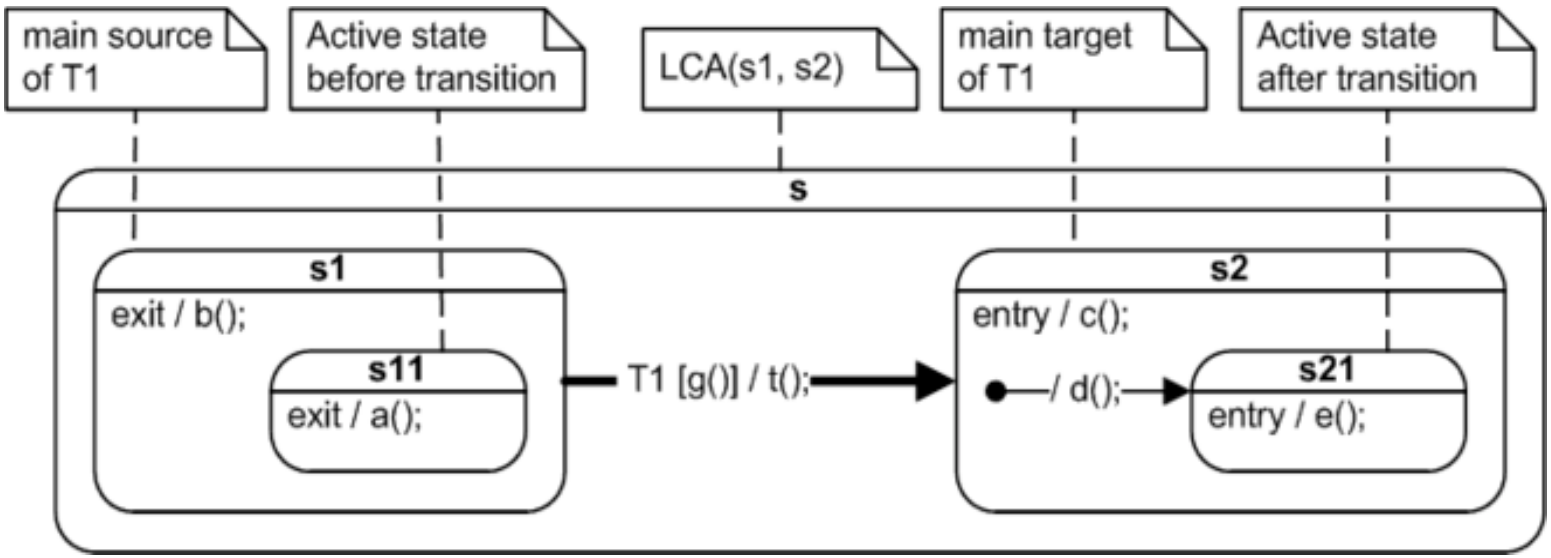
- Two kinds of state machines.
- Behavioral state machines are used to model the behavior of individual entities (e.g., class instances)
- Protocol state machines are used to express usage protocols and can be used to specify the legal usage scenarios of classifiers, interfaces, and ports.
- Behavioral state machine is subclassed by protocol state machine.


PhoneCall
name
composite state
Active
Timeout
do/ play message
simple state
dial digit(n)
[incomplete]
guard
condition
event
lift receiver
/get dial tone
after (15 sec.)
time event
after (15 sec.)
activity
DialTone
do/ play dial tone
dial digit(n)
Dialing
self
transition
transition
dial digit(n)[invalid]
dial digit(n)[valid]
/connect
Invalid
do/ play message
install
Idle
initial state
Connecting
entry
point
Pinned
busy
Busy
do/ play busy
tone
connected
callee
answers
callee
hangs up
caller
hangs up
/disconnect
Talking
Ringing
do/ play ringing tone
callee answers
/enable speech
cancel service
exit point

Execution order

- UML specifies that taking a state transition executes the following actions in the following sequence

1. Evaluate the guard condition associated with the transition and perform the following steps only if the guard evaluates to TRUE.
2. Exit the source state configuration.
3. Execute the actions associated with the transition.
4. Enter the target state configuration.



Taking T1 causes the evaluation of guard g();
Exit of s11, s1,
Action sequence a(); b(); t(); c(); d(); and e();
Entry of s2, s21
(assuming guard g() evaluates to TRUE)

MS Spec Explorer (Spec#)

Text-based *state machine models*

Class Client {

bool entered;

Map<Client,Seq<**string**>> unreceivedMsgs;

[Action] **void** Enter()

Action of abstract state machine on state entry

requires !entered; { required condition

entered = true;

}

```
[Action] void Send(string message)
    requires entered; {
    foreach (Client c in enumof(Client), c != this,
    c.entered)
        c.unreceivedMsgs[this] += Seq{message};
}
```

A Chat room model

DD2459 Software Reliability
Lecture 5

Introduction to
Model-Based Testing (MBT)

(see Amman and Offut,
Chapter 2, Section 2.5)

Part 2. MBT for ATCG

Model-based Automated Test-Case Generation (ATCG)

- Manual construction of test cases is a difficult, time-consuming and error-prone activity that requires expertise.
- Automated Test-Case Generation (TCG) has been the "holy grail" of testing for some time.
- Is this even possible? If so, how?
- Black/white-box testing
- Want algorithms which generate test cases with a known level of coverage

TCG Technologies

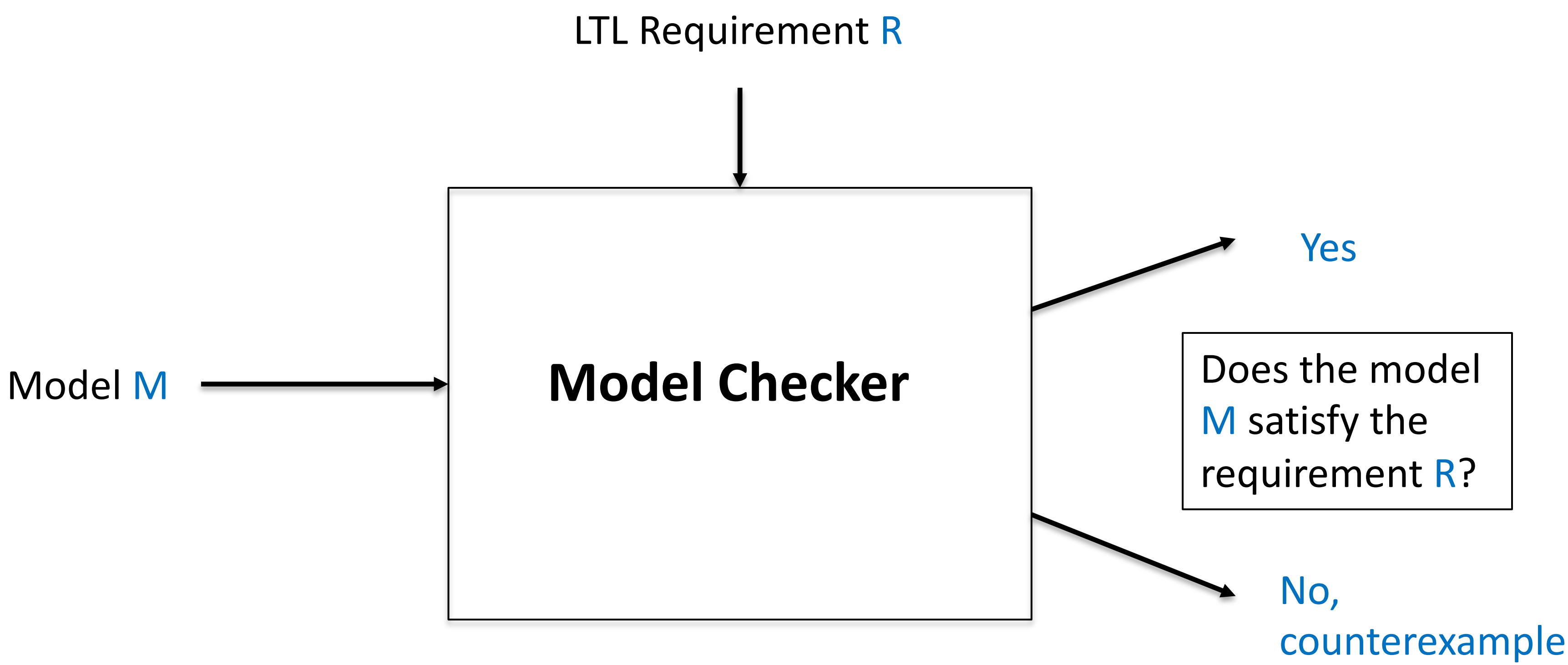
- Model-based test case generation can be done with various technologies:

1. *graph search,*
2. *constraint solving*
3. *theorem proving*
4. *model checking,*

We will focus on model checking

Model Checking

- A model checker is a tool which takes as input
  - An automaton model M
  - A logical formula ϕ
- If ϕ is a true statement about all possible behaviors of M then the model checker confirms it (proof)
- If ϕ is a false statement about M the model checker constructs a counterexample (a simulation sequence)
- A counterexample to ϕ satisfies !ϕ
- A simulation sequence can be executed as a test case

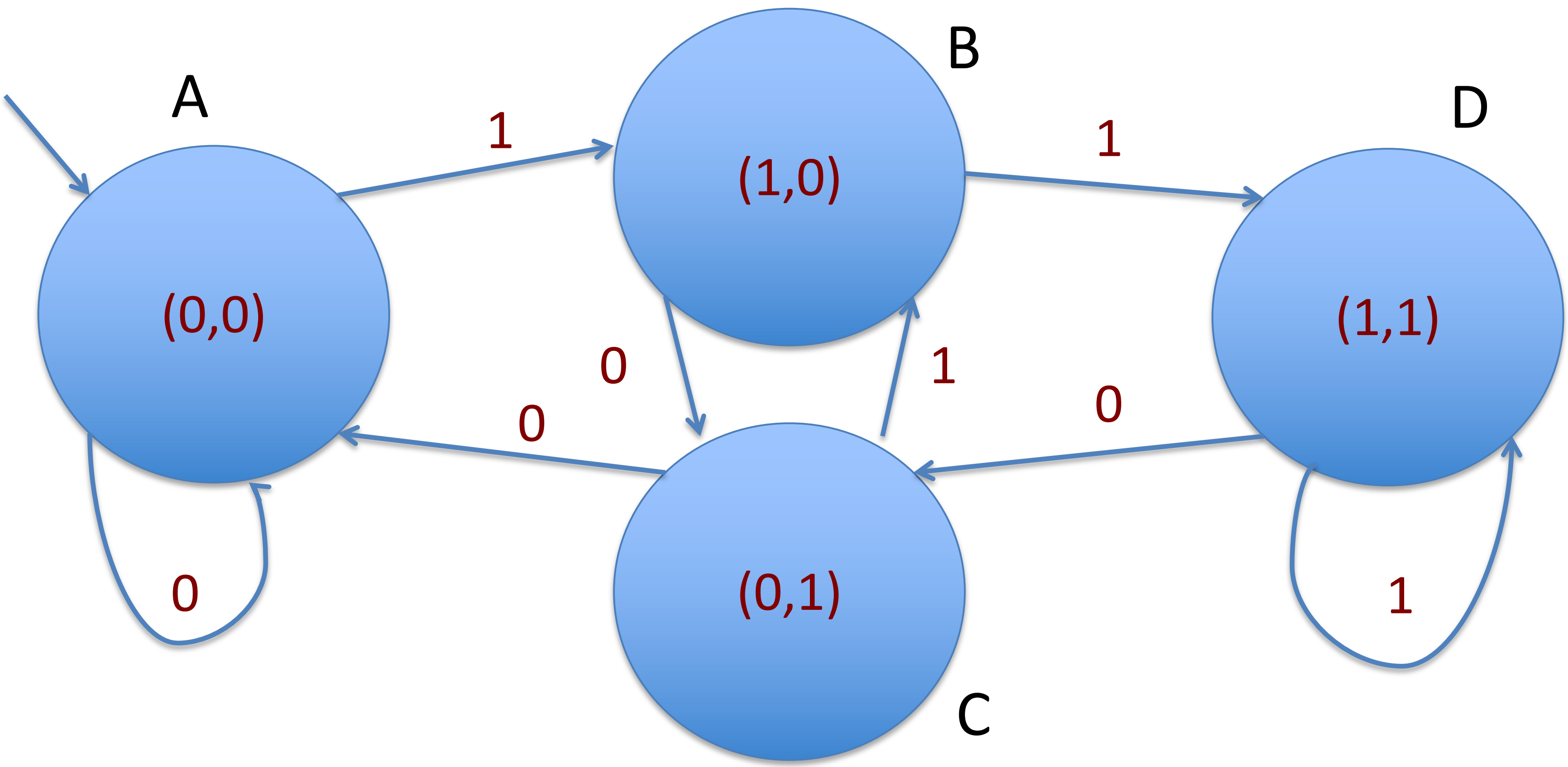


Recall from Lecture 4,

A counterexample is a trace that satisfies !R

Example: Two Bit Shift Register as a Moore Automaton

- $Q = \{A, B, C, D\}$, $\Sigma = \{0,1\}$, $q_0=A$,
- output = (Bit1, Bit2) i.e. “shift right”



2-Bit Shift Reg in NuSMV using .smv format

```
MODULE main
VAR
-- system outputs
   Bit1 : boolean; -- Boolean variable
   Bit2 : boolean;
   state : {A, B, C, D}; -- scalar variable

IVAR
-- system inputs
    input : boolean;

ASSIGN
   init(state) := A;
   init(Bit1) := 0;
   init(Bit2) := 0;
```

```
next(state) := case
   state = A & input = 1 : B;
   state = B & input = 0 : C;
   state = B & input = 1 : D;
   state = C & input = 0 : A;
   state = C & input = 1 : B;
   state = D & input = 0 : C;
   TRUE : state;
esac;
   next(Bit1) := case lab 3 esac;
   next(Bit2) := case lab 3 esac;
```

Linear Temporal Logic LTL in .smv format

- Use temporal logic to express automaton requirements
- Boolean variables
- `A, B, …, X, Y, .. MyVar`, etc.
- Boolean operators
- `!(`ϕ`), (`ϕ` & `θ`), (`ϕ` |`θ`), (`ϕ` -> `θ`)` ,…
- Temporal (time) operators
- `F(`ϕ`)` (sometime in the future ϕ)
- `G(`ϕ`)` (always in the future ϕ)
- `(`ϕ` U `θ`)` (ϕ holds until θ holds)
- `X(`ϕ`)` (next ϕ holds)
- Write $X^n(\phi)$ for $X(X(\ldots X(\phi)))$ (ϕ holds in n steps)

Examples of LTL

Right now it is Wednesday

    Wednesday

Tomorrow is Wednesday

    X (Wednesday)

(A) Thursday (always) immediately follows Wednesday

    G(Wednesday -> X (Thursday))

(A) Saturday (always) follows Wednesday

    G(Wednesday -> F(Saturday))

Yesterday was Wednesday

    G(Wednesday <-> X (Thursday)) & Thursday

- Exercise: define the sequence of days precisely, i.e. just one solution
- Question: are there any English statements you can't make in LTL?
- Question: what use cases can you express in LTL?

Basic Identities

It makes sense to say that

$$G(\varphi) = \varphi \;\&\; X(\varphi) \;\&\; X(X(\varphi)) \;\&\; \ldots$$

or

$$G(\varphi) = \mathop{\&}_{i >= 0} X^i(\varphi)$$

Also

$$F(\varphi) = \varphi \mid X(\varphi) \mid X(X(\varphi)) \;\ldots$$

$$F(\varphi) = \mathop{\Big|}_{i >= 0} X^i(\varphi)$$

Useful Logical Identities for LTL

- Boolean identities

  $!(!(\varphi)) \Leftrightarrow \varphi$, $!(\phi \mid \varphi) \Leftrightarrow (!\phi \;\&\; !\varphi)$,

  $(\phi \;\text{->}\; \varphi) \Leftrightarrow (!(\phi) \mid \varphi)$ etc.

- LTL identities

  $!(G(!(\phi))) \Leftrightarrow F(\phi)$

  $!(X(\phi)) \Leftrightarrow X(!(\phi))$

  $G(\varphi \;\&\; \phi) \Leftrightarrow G(\varphi) \;\&\; G(\phi)$

  $G(\varphi) \Leftrightarrow \varphi \;\&\; X\;(G(\varphi))$

  $G\;(G(\varphi)) \Leftrightarrow G(\varphi)$

- <u>Exercise</u>: using these identities, prove:

  $!(F(!(\varphi))) \Leftrightarrow G(\varphi)$

  $F(\varphi \mid \phi) \Leftrightarrow F(\varphi) \mid F(\phi)$

  $F(\varphi) \Leftrightarrow \varphi \mid X(F(\varphi))$

- Remark TCG usually involves negating formulas, so its useful to <u>understand what a negation means</u>

LTL Specifications
.smv format

```
LTLSPEC
   G( Bit1 <-> (X Bit2) )
   -- always the value of Bit1 now equals the
   -- value of Bit2 in the next time step
   -- This is obviously TRUE!

   G( Bit1 <-> (X Bit1) )
   -- always the value of Bit1 now equals the
   -- value of Bit1 in the next time step
   -- This is obviously FALSE!
```

NuSMV Output Example

```
-- specification G( Bit1 <-> (X Bit2) )
-- is true
-- specification G( Bit1 <-> (X Bit1) )
-- is false
-- as demonstrated by the following execution
sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
state = A Bit1 = false Bit2 = false
-> Input: 1.2 <-
input = 1
-> State 1.2 <-
state = B Bit1 = true Bit2 = false
```

Automated Glass-box TCG

- We can use a model checker to generate counterexamples to formulas (i.e. test cases) with specific structural properties.
- This is done by inspecting the graph structure of the automaton
- i.e. white/glass box testing
- "Most" use of model checkers concerns this.

Test Requirements
Trap Properties

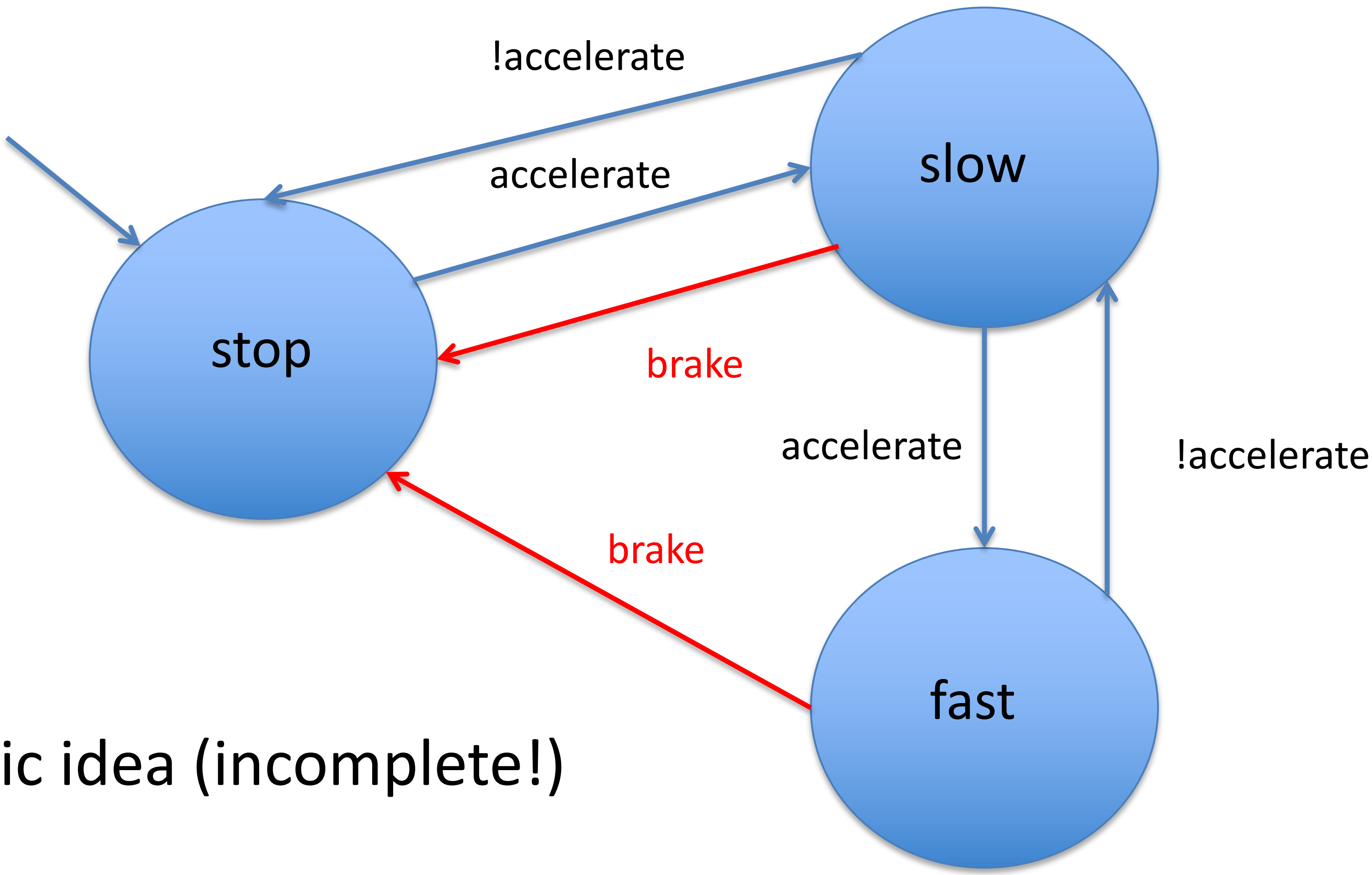
- Recall a test requirement is a *requirement that can be satisfied by one or more test cases*
- Basic idea is to capture each test requirement as an LTL formula known as a "trap property"

Example Suppose the test requirement is "*cover state D of shift register*"

Trap property is `G(!( state = D ))`

This formula is `False` and any counterexample must be a path that goes through state `D`.

Case Study:
Car Controller Model (CC)



Basic idea (incomplete!)

```
MODULE main
VAR
   state: {stop, slow, fast};
   -- velocity states

IVAR
   accelerate: boolean; -- gas pedal
   brake: boolean; -- brake pedal

ASSIGN
   Init(state) := stop;

   Next(state) := case
      accelerate & !brake & state = stop : slow;
      accelerate & !brake & state = slow : fast;
      !accelerate & !brake & state = fast : slow;
      !accelerate & !brake & state = slow: stop;
      brake: stop;
      TRUE: state;
   esac;
```

Trap properties for Structural Coverage Models

- Let’s use NuSMV to automatically construct test suites that satisfy the different glass box coverage models introduced in Lecture 2.
- Examples:
  - Node coverage NC
  - Edge coverage EC
  - Condition coverage PC
- How to interpret these concepts?

Node Coverage for CC

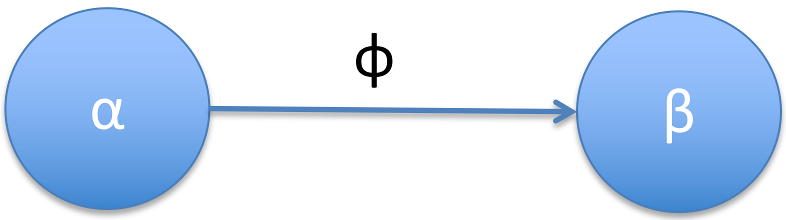
- Want a path that visits each node
- Simple approach: write 1 trap property per node
- General form:
- `G(!(state = <state_name>))`
- Counterexamples satisfy:
- `F(state = <state_name> )`
- Example:

```
G(!(state = stop));
```

- Clearly this will give redundant test cases, but method is still linear in state-space size.
- Lab Exercise 3: define the remaining 2 trap formulas for car controller

Edge coverage for CC

- Want to traverse each edge between any pair of nodes



- General form G((state = α & ϕ) -> X (!(state = β))
- Counterexample satisfies F(state = α & ϕ & X(state = β))
- Example:

```
G(state=stop & accelerate ->
     X (!(state=slow)) )
```

- Lab Exercise 3: define the remaining 5 trap formulas for car controller

Requirements-based TCG

- Can we also perform requirements-based test case generation?
- Want to test the requirements are fulfilled rather than explore structure of the code.
- Can look at negation of a requirement

Car Controller
LTL requirements

1. Whenever the brake is activated, car has to stop quickly

```
G(brake -> X(state=stop) )
```

2. When accelerating and not braking, velocity has to increase gradually

```
G(!brake & accelerate & state=stop ->
          X(state=slow) )
G(!brake & accelerate & state=slow ->
          X(state=fast) )
```

3. When not accelerating and not braking, velocity has to decrease gradually

```
G(!brake& !accelerate & state=fast ->
          X(state=slow) )
G(!brake & accelerate & state=slow ->
          X(state=stop) )
```

Safety Requirements

- A safety requirement describes a behavior that may not occur on any path.
- "*Something bad never happens*"
- To verify, all execution paths must be checked exhaustively
- Safety properties usually have the form `G!`ϕ where ϕ defines the "*bad thing*"
- Counterexamples (test cases) are finite

Liveness Requirements

- A liveness requirement describes a behavior that must hold on all execution paths
- "Something good eventually happens"
- Safety does not imply liveness
- Liveness does not imply safety
- Liveness properties often have the form

  $\mathtt{F}(\theta)$ or $\mathtt{G}(\phi \mathtt{\ ->\ } \mathtt{X}^{n}\theta)$ or $\mathtt{G}(\phi \mathtt{\ ->\ } \mathtt{F}\theta)$

  where θ describes the "good" thing and ϕ is some precondition needed for it to occur.
- Counterexamples may be finite or infinite (why?)

TCG for Model-Based Requirements Testing

- Suppose we have an LTL requirement ϕ
- Feed into NuSMV an FSM model A of the SUT, together with the negated formula $!\phi$
- Choose any counterexample (a behaviour b) (there should be lots if A is a correct model and is a correct requirement).
- b satisfies $!!\ \phi$ i.e. b is an example of correct behavior
- Feed the inputs of b into the SUT and observe the output
- If output from the SUT matches b then pass else fail

Car Controller Examples

(1) `F(brake & X !(velocity=stop))`

(2) `F(!brake & accelerate & velocity=stop -> X(velocity=slow))`

Concluding Remarks

- Model-based testing has created a lot of interest and opened up new questions
- Tools market is emerging
- What is best modeling language?
- Automated TCG is possible using off-the-shelf tools, e.g. model checkers
- New technologies such as machine learning avoid manual model construction.

Lecture 5 Summary

- MBT uses a model of code for test construction
- Model can be text or graphics
- Graph coverage can be re-used for model coverage
- Models help solve oracle problem (conformance testing)
- Tools allow automated test generation as path traversal in a model.
- ATCG based on *model checking*, *graph search*, *constraint solving* or *theorem proving*