

Model-based testing Specifications – temporal logics

Abhik Roychoudhury
<http://www.comp.nus.edu.sg/~abhik>

Copyright 2009 by Abhik Roychoudhury

Flow of today's lecture

- ▶ Test generated from models
 - ▶ Run on implementation.
- ▶ How to find a "suitable" test case?
 - ▶ What is the purpose of testing?
- ▶ Finding a "suitable" test case guided by test specification
 - ▶ Given a test specification, we search the model to find a test?
- ▶ Two questions
 - ▶ How to describe test specifications – temporal logics.
 - ▶ How to search the system model – model checking.

Copyright 2009 by Abhik Roychoudhury

Model-based system development

```

    graph TD
        IR[Informal Requirements (in English)] -- Modeling --> SM[System Model (UML State and Class Diagrams)]
        SM -- Model Simulation --> IR
        SM -- "Partitioning, Scheduling and other impl. steps" --> SI[System Implementation (Hardware / C)]
        SI -- Testing --> SM
    
```

Copyright 2009 by Abhik Roychoudhury

Model-based testing

- ▶ Generate test-cases from model, run them on the implementation.
- ▶ What are the criteria for generating test cases?
 - ▶ Generate a suite of test cases to ensure a structural coverage of the model
 - ▶ State coverage, Transition coverage for State Diagrams.
 - ▶ Generate test cases from the model based on some test specification
 - ▶ How to describe the test specification?
 - Temporal logic (discussed later)
 - ▶ How to find a test satisfying a test specification?
 - Model checking (discussed later)

Copyright 2009 by Abhik Roychoudhury

Test-purpose based test gen. & exec.

```

    graph TD
        TP[Test purpose or Test spec.] --> MTG((Model-based Test Generator))
        SM[System Model] --> MTG
        MTG --> STC[Sample Test-case]
        SI[System Implementation] --> TE((Test Execution))
        STC --> TE
        TE --> TV[Test Verdict (pass/fail/inconclusive)]
    
```

Copyright 2009 by Abhik Roychoudhury

Test Execution Architecture

The diagram illustrates the Test Execution Architecture. It shows a **Test Generation** process that produces a **Test-case**. This test-case is used to interact with the **IUT = Implementation Under Test**, which consists of components C_{k+1} through C_n . The IUT has **INPUT** and **OUTPUT** ports. The test-case is implemented by a **Tester System** containing **Tester₁** through **Tester_k** and a **Master Tester**. The Tester System sends **Test verdicts** to the Master Tester, which then sends **Test verdicts** back to the IUT. The Tester System also receives **Internal Msgs.** from the IUT. The overall process results in an **Output Verdict**.

Copyright 2009 by Abhik Roychoudhury

Test Execution - (1)

(a) Test-case MSC **M**

(b) Partial Order of **M**

(c) Test graph of events involving interaction between tester components and IUT.

▶ 7 Copyright 2009 by Abhik Roychoudhury

Test Execution - (2)

(c) Test graph of events involving interaction between tester components and IUT.

(d) Test graph of **M** with Synchronization events

(e) Local test graphs of tester lifelines **A** and **B**

(b) Partial Order of **M**

▶ 8 Copyright 2009 by Abhik Roychoudhury

Test Execution - (3)

Test-case MSC

Tester lifelines

Synthesized **Tester Components**

▶ 9 Copyright 2009 by Abhik Roychoudhury

Test Verdicts

- ▶ **Pass**
 - ▶ All the tester components convey "Pass" to a Master tester.
- ▶ **Fail**
 - ▶ At least one tester component returns fail.
- ▶ **Inconclusive**
 - ▶ None of the tester components return fail, and
 - ▶ At least one tester component returns inconclusive.

▶ 10 Copyright 2009 by Abhik Roychoudhury

Test-purpose based test generation

Described as MSC

Test purpose or Test spec.

System Model FSMs/ State Diagrams

Model-based Test Generator Automated search in the global FSM

Sample Test-case As a MSC or a trace.

▶ 11 Copyright 2009 by Abhik Roychoudhury

Test spec. & Generated Test

ATC WCP

Test Specification

Client ATC WCP

ClientPreUpd update

ClientUpd WCPDisable

GET_NEW_WTHR

Yes

ClientPostUpd

USE_NEW_WTHR

Yes enable

▶ 12 Copyright 2009 by Abhik Roychoudhury

Test spec. & Generated test

- ▶ Test spec. is in the form of an MSC M.
- ▶ Def. 1
 - ▶ A trace σ satisfies a test specification M if σ contains at least one linearization of M as a **contiguous subsequence**.
- ▶ Def. 2
 - ▶ A trace σ satisfies a test specification M if σ contains at least one linearization of M as a **subsequence**.
- ▶ Which def. did we follow in the previous slide?

▶ 13 Copyright 2009 by Abhik Roychoudhury

Test Generation

The flowchart shows the process of test generation. It starts with 'Temporal Logic Property' (described as MSC) and 'System Model' (FSMs/ State Diagrams). These feed into a 'Model-based Test Generator'. The generator performs 'Automated search in the global FSM' (Model Checking) to produce a 'Sample Test-case' (As a MSC or a trace). This test case can be used as a 'Counter-example' or for 'Model Checking (applications in test gen. and other purposes)'.

▶ 14 Copyright 2009 by Abhik Roychoudhury

Test generation

- ▶ Test purpose
 - ▶ Defined as MSC or Sequence Diagram
 - ▶ Can be described using *temporal logic* (taught now)
- ▶ System model
 - ▶ Described as FSM or State Diagram
- ▶ Test generation method
 - ▶ Finite search inside the System model's FSM
 - ▶ Accomplished by *model checking* (taught in next week)
- ▶ Output of test generation method
 - ▶ A test case described as a trace or a MSC
 - ▶ Satisfies the test purpose MSC.

▶ 15 Copyright 2009 by Abhik Roychoudhury

Organization

- ▶ So Far
 - ▶ What is a Model?
 - ▶ ATC – Running Example
 - ▶ How to model such requirements
 - ▶ How to validate the models
 - ▶ Simulations,
 - ▶ Model-based testing,
 - ▶ Model Checking (discussed now)
 - Temporal logics (the property specification)
 - Checking method
 - ▶ Also, model-based testing accomplished by model checking

▶ 16 Copyright 2009 by Abhik Roychoudhury

The big picture

The flowchart shows the overall process: 'System to be built (Dream or requirements)' leads to 'System Model (Rough Idea)'. 'Properties to Satisfy (caution)' leads to 'Checking Method (Automated)'. The 'System Model' is simulated, and the 'Checking Method' produces 'Counter-examples'. These counter-examples are used to 'Refine the model', which then feeds back into the 'System Model'.

Temporal logics and model checking have a general usage in model / system validation, apart from test generation in model-based testing.

▶ 17 Copyright 2009 by Abhik Roychoudhury

Example System Model - FSMs

The diagrams show a sequence of states: green, yellow, red. The top diagram shows a simple linear sequence: green → yellow → red. The bottom diagram shows a more complex sequence with a self-loop on the red state and a return path from red to green: green → yellow → red → red → green.

Infinite length traces
Possible to have infinitely many traces.

▶ 18 Copyright 2009 by Abhik Roychoudhury

Temporal Logic

- ▶ On June 1, 2007, I am teaching temporal logics which will be followed by teaching of model checking on June 8, 2007
- ▶ Teaching of temporal logics occurs 1 week before the teaching of model checking.
- ▶ Teaching of temporal logics is *always eventually* followed by the teaching of model checking.
- ▶ Teaching of temporal logics is *always immediately* followed by the teaching of model checking.

▶ 19 Copyright 2009 by Abhik Roychoudhury

Example properties

- ▶ The light is *always* green.
- ▶ *Whenever* the light is red, it *eventually* becomes green.
- ▶ *Whenever* the light is green, it remains green *until* it becomes yellow.
- ▶ ...
- ▶ Are these properties true for the 2 example models in the previous slide?
 - ▶ Let us try the second property for example ...

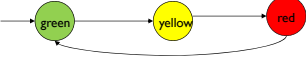
▶ 20 Copyright 2009 by Abhik Roychoudhury

When is a property satisfied?

- ▶ A property is **interpreted** on the traces of a system model.
 - ▶ Given a trace of the system model x and a property p , we can uniquely determine a yes/no answer to whether x satisfies p .
- ▶ A property p is satisfied by a system model M , if all traces of M satisfy p .
- ▶ **So, given a system model what are its traces?**

▶ 21 Copyright 2009 by Abhik Roychoudhury

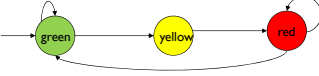
Traces of a system model



- ▶ **Only one trace, it has infinite length**
 - ▶ $(green, yellow, red) - \text{repeated forever}$ Written as $(green, yellow, red)^\omega$

▶ 22 Copyright 2009 by Abhik Roychoudhury

Traces of a system model



- ▶ **Infinitely many traces, each of infinite length**
 - ▶ $(green)^\omega$ - 1 trace
 - ▶ $(green)^* yellow (red)^\omega$ - many traces
 - ▶ $(green)^* yellow (red)^* (green)^\omega$
 - ▶ ...
 - ▶ $(green, yellow, red)^\omega$

▶ 23 Copyright 2009 by Abhik Roychoudhury

Property Specification Language

- ▶ Properties in our property spec. language will be interpreted over **infinite length** traces.
 - ▶ Finite length traces can be converted into infinite length traces by putting a self-loop at last state.
- ▶ A property is satisfied by a system model if all execution traces satisfy the property.
 - ▶ In general, we cannot test the property on each exec. trace – infinitely many of them.
 - ▶ Model checking is smarter – we discuss it later!
- ▶ We formally describe the property spec. lang. or logic

▶ 24 Copyright 2009 by Abhik Roychoudhury

Why study new logics ?

- ▶ Need a formalism to specify properties to be checked
- ▶ Our properties refer to dynamic system behaviors
 - ▶ Eventually, the system reaches a stable state
 - ▶ Never a deadlock can occur
- ▶ We want to maintain more than input-output properties (which are typical for transformational systems).
 - ▶ Input-output property: for input > 0 , output should be > 0
 - ▶ No notion of output or end-state in reactive systems.

▶ 25

Copyright 2009 by Abhik Roychoudhury

Why study new logics ?

- ▶ Our properties express constraints on dynamic evolution of states.
- ▶ Propositional/first-order logics can only express properties of states, not properties of traces
- ▶ We study behaviors by looking at all execution traces of the system.
 - ▶ Linear-time Temporal Logic (LTL) is interpreted over execution traces of a system model.

▶ 26

Copyright 2009 by Abhik Roychoudhury

Formally, system model is

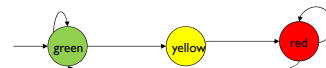
- ▶ Model for reactive systems
 - ▶ $M = (S, S_0, \rightarrow, L)$
 - ▶ S is the set of states
 - ▶ $S_0 \subseteq S$ is the set of initial states
 - ▶ $\rightarrow \subseteq S \times S$ is the transition relation
 - ▶ Set of (source-state, destination-state) pairs
 - ▶ L : is the labeling function mapping S to 2^{AP}
 - ▶ Maps each state s to a subset of AP
 - ▶ These are the atomic prop. which are true in s .

▶ 27

Copyright 2009 by Abhik Roychoudhury

Atomic Propositions

- ▶ All of our properties will contain atomic props.
 - ▶ These atomic props. will appear in the labeling function of the system model you verify.
 - ▶ The atomic props. represent some relationships among variables in the design that you verify.
 - ▶ Atomic props in the following example
 - ▶ **green, yellow, red (marked inside the states with obvious labeling function).**



▶ 28

Copyright 2009 by Abhik Roychoudhury

Linear-time Temporal Logic

- ▶ The temporal logic that we study today build on a "static" logic like propositional logic.
 - ▶ Used to describe/constrain properties inside states.
- ▶ Temporal operators describe properties on execution traces.
 - ▶ Used to describe/constrain evolution of states.
- ▶ Time is **not** explicitly mentioned in the formulae
 - ▶ Properties describe how the system should evolve over time.

▶ 29

Copyright 2009 by Abhik Roychoudhury

Linear-time Temporal Logic

- ▶ Does not capture exact timing of events, but rather the relative order of events
- ▶ We capture properties of the following form.
 - ▶ Whenever event e occurs, eventually event e' must occur.
- ▶ We do **not** capture properties of the following form.
 - ▶ At $t = 2$ e occurs followed by e' occurring at $t = 4$.

▶ 30

Copyright 2009 by Abhik Roychoudhury

Notations and Conventions

- ▶ An LTL formula φ is interpreted over an infinite sequence of states $\pi = s_0, s_1, \dots$
 - ▶ Use $M, \pi \models \varphi$ to denote that formula φ holds in path π of system model M .
- ▶ Define semantics of LTL formulae w.r.t. a system model M .
 - ▶ **An LTL property φ is true of a system model iff all its traces satisfy φ**
 - ▶ **$M \models \varphi$ iff $M, \pi \models \varphi$ for all traces π in system model M**

▶ 31 Copyright 2009 by Abhik Roychoudhury

Notations and Conventions

- ▶ $M, \pi \models \varphi$
 - ▶ Path $\pi = s_0, s_1, s_2, \dots$ in model M satisfies property φ
- ▶ $M, \pi^k \models \varphi$
 - ▶ Path s_k, s_{k+1}, \dots in model M satisfies property φ
- ▶ We now use these notations to define the syntax & semantics of LTL.

▶ 32 Copyright 2009 by Abhik Roychoudhury

LTL - syntax

- ▶ Propositional Linear-time Temporal logic
- ▶ $\varphi = X\varphi \mid G\varphi \mid F\varphi \mid \varphi \cup \varphi \mid \varphi R \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \text{Prop}$
- ▶ Prop is the set of atomic propositions
- ▶ Temporal operators
 - ▶ X (next - state)
 - ▶ F (eventually), G (globally)
 - ▶ U (until), R (release)

▶ 33 Copyright 2009 by Abhik Roychoudhury

Semantics of propositional logic

- ▶ $M, \pi \models p$ iff $s_0 \models p$ i.e. $p \in L(s_0)$ where L is the labeling function of Kripke Structure M
- ▶ $M, \pi \models \neg \varphi$ iff $\neg (M, \pi \models \varphi)$
- ▶ $M, \pi \models \varphi_1 \wedge \varphi_2$ iff $M, \pi \models \varphi_1$ and $M, \pi \models \varphi_2$

▶ 34 Copyright 2009 by Abhik Roychoudhury

neXt-state operator of LTL

- ▶ $M, \pi \models X\varphi$ iff $M, \pi^1 \models \varphi$
 - ▶ Path starting from **next state** satisfies φ

▶ 35 Copyright 2009 by Abhik Roychoudhury

Finally operator of LTL

- ▶ $M, \pi \models F\varphi$ iff $\exists k \geq 0 M, \pi^k \models \varphi$
 - ▶ Path starting from an **eventually** reached state satisfies φ

▶ 36 Copyright 2009 by Abhik Roychoudhury

Globally operator of LTL

- ▶ $M, \pi \models G\phi$ iff $\forall k \geq 0 M, \pi^k \models \phi$
- ▶ Path **always** satisfies ϕ (all suffixes of the path satisfy ϕ)

▶ 37 Copyright 2009 by Abhik Roychoudhury

Until operator of LTL

- $M, \pi \models \phi_1 U \phi_2$ iff $\exists k \geq 0$ such that
 - $M, \pi^k \models \phi_2$, and
 - $\forall 0 \leq j < k M, \pi^j \models \phi_1$

A trace satisfying $p U q$, where $p, q \in Prop$

▶ 38 Copyright 2009 by Abhik Roychoudhury

Until Operator

▶ 39 Copyright 2009 by Abhik Roychoudhury

Release operator of LTL

- $M, \pi \models \phi_1 R \phi_2$ iff
 - Either $\forall k \geq 0 M, \pi^k \models \phi_2$
 - OR both of the following hold
 - $\exists k \geq 0 M, \pi^k \models \phi_1$
 - $\forall 0 \leq j \leq k M, \pi^j \models \phi_2$
- ϕ_1 releases the req. for ϕ_2 to hold.

▶ 40 Copyright 2009 by Abhik Roychoudhury

Release - Case 1

▶ 41 Copyright 2009 by Abhik Roychoudhury

Release - Case (2)

▶ 42 Copyright 2009 by Abhik Roychoudhury

Exercise – (1)

- ▶ The light is *always* green.
- ▶ *Whenever* the light is red, it *eventually* becomes green.
- ▶ *Whenever* the light is green, it remains green *until* it becomes yellow.
- ▶ *Whenever* the light is yellow, it becomes red *immediately after*.

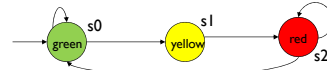
- ▶ Encode these properties in LTL.

▶ 43

Copyright 2009 by Abhik Roychoudhury

Exercise – (2)

- ▶ Check whether the four LTL properties in the previous slide are satisfied by our simple traffic light controller.



▶ 44

Copyright 2009 by Abhik Roychoudhury

LTL Exercise – (3)

Consider a resource allocation protocol where n processes P_1, \dots, P_n are contending for exclusive access of a shared resource. Access to the shared resource is controlled by an arbiter process. The atomic proposition req_i is true only when P_i explicitly sends an access request to the arbiter. The atomic proposition gnt_i is true only when the arbiter grants access to P_i . Now suppose that the following LTL formula holds for our resource allocation protocol.

- ▶ $G (req_i \Rightarrow F gnt_i)$

▶ 45

Copyright 2009 by Abhik Roychoudhury

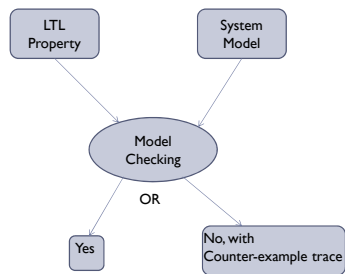
LTL Exercise – (3)

- ▶ Explain in English what the property means.
- ▶ Is this a desirable property of the protocol ?
- ▶ Suppose that the resource allocation protocol has a distributed implementation so that each process is implemented in a different site. Does the LTL property affect the communication overheads among the processes in any way ?

▶ 46

Copyright 2009 by Abhik Roychoudhury

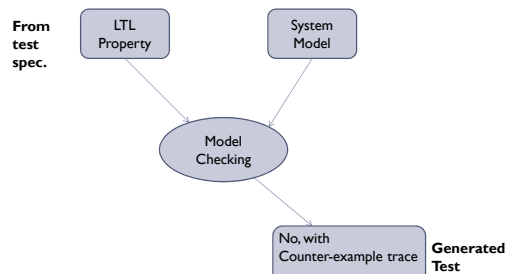
Model Checking



▶ 47

Copyright 2009 by Abhik Roychoudhury

Recap: Model Checking for model-based testing



▶ 48

Copyright 2009 by Abhik Roychoudhury

Encoding test specifications

- ▶ **Def. 1**
 - ▶ A trace σ satisfies a test specification M if σ contains at least one linearization of M as a **contiguous subsequence**.
 - ▶ Given MSC M,
 - ▶ define $\text{Lin}(M)$ = set of linearizations of M.
 - ▶ For each linearization $\sigma = e_1, e_2, \dots, e_k$ define
 - Define $\text{prop}_\sigma = F(e_1 \wedge X(e_2 \wedge X(\dots X(e_k) \dots)))$
 - ▶ Define property ϕ_M corresponding to M as
 - $\phi_M = \neg (\bigvee_{\sigma \in \text{Lin}(M)} \text{PROP}_\sigma)$
 - ▶ **A counter-example to ϕ_M is a test satisfying M.**

▶ 49 Copyright 2009 by Abhik Roychoudhury

Example

Possible linearizations
 $e1, e2, e3, e4$
 $e1, e3, e2, e4$

LTL property
 $\neg (F(e1 \wedge X(e2 \wedge X(e3 \wedge X e4))))$
 \vee
 $F(e1 \wedge X(e3 \wedge X(e2 \wedge X e4))))$

▶ 50 Copyright 2009 by Abhik Roychoudhury

Encoding test specifications

- ▶ **Def. 2**
 - ▶ A trace σ satisfies a test specification M if σ contains at least one linearization of M as a **subsequence**.
 - ▶ Given MSC M,
 - ▶ define $\text{Lin}(M)$ = set of linearizations of M.
 - ▶ For each linearization $\sigma = e_1, e_2, \dots, e_k$ define
 - $n_\sigma = \neg (e_1 \vee e_2 \vee \dots \vee e_k)$
 - $\text{prop}_\sigma = (n_\sigma \mathbf{U} (e_1 \wedge X(n_\sigma \mathbf{U} (e_2 \wedge X(\dots X(n_\sigma \mathbf{U} e_k) \dots))))$
 - ▶ Define property ϕ_M corresponding to M as
 - $\phi_M = \neg (\bigvee_{\sigma \in \text{Lin}(M)} \text{PROP}_\sigma)$
 - ▶ **A counter-example to ϕ_M is a test satisfying M.**

▶ 51 Copyright 2009 by Abhik Roychoudhury

Example

Possible linearizations
 $e1, e2, e3, e4$
 $e1, e3, e2, e4$

LTL property
 ??
 (try this as an exercise)

▶ 52 Copyright 2009 by Abhik Roychoudhury

Model Checking – Next class

Describe Model Checking as a general verification procedure. It proceeds by search.

OR

Yes

No, with Counter-example trace

▶ 53 Copyright 2009 by Abhik Roychoudhury