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.

Model-based system development

Informal Requirements
(in English)
Modeling
System Model
(UML State and Class Diagrams)
Partitioning, Scheduling
and other impl. steps
System Implementation
(Hardware / C)

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)

Test-purpose based test gen. & exec.

Model-based Test Generation

Sample Test-case
System Model
Test Execution
Test Verdict
(pass/fail/inconclusive)

Test Execution Architecture

IUT = Implementation Under Test

Tester
System

Implemented by

Tester
System

Test verdicts

Test execution architecture
Test Execution – (1)

- Test-case MSC
- Partial Order of M
- Test graph of events involving interaction between tester components and IUT.

Test Execution – (2)

- Test graph of events involving interaction between tester components and IUT.

Test Execution – (3)

- Test-case MSC
- Tester lifelines

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.

Test-purpose based test generation

- Test purpose or Test spec.
- System Model
- FSMs/State Diagrams

Automated search in the global FSM

As a MSC or a trace.

Test spec. & Generated Test

- Test Specification

Client
- ClientPostUpd
- USE_NEW_WTHR
- Yes

ATC
- WCP
- update
- enable

Yes

ATC
- update
- enable

WCP
- WCPDisable
- enable
Test spec. & Generated test

- Test spec. is in the form of an MSC M.

Def. 1
- A trace $\sigma$ satisfies a test specification M if $\sigma$ contains at least one linearization of M as a contiguous subsequence.

Def. 2
- A trace $\sigma$ satisfies a test specification M if $\sigma$ contains at least one linearization of M as a subsequence.

Which def. did we follow in the previous slide?

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.

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)
      - Checking method
    - Also, model-based testing accomplished by model checking

Example System Model - FSMs

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

Infinite length traces
Possible to have infinitely many traces.
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.

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 …

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?

Traces of a system model

- Only one trace, it has infinite length
  - (green, yellow, red)$^{\omega}$ – repeated forever
- Infinitely many traces, each of infinite length
  - (green)$^{\omega}$ – 1 trace
  - (green)$^{3}$ yellow (red)$^{\omega}$ – many traces
  - (green)/yellow (red)$^{3}$ (green)$^{\omega}$
  - ...
  - (green, yellow, red)$^{\omega}$

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
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.

Eventually, the system reaches a stable state.

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 props. which are true in \( s \).

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).

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.

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 \( t = 4 \).
Notations and Conventions

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

We now use these notations to define the syntax & semantics of LTL.

LTL - syntax

- Propositional Linear-time Temporal logic
  - $\varphi = X\varphi \mid G\varphi \mid F\varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid \neg \varphi \mid Prop$
- Temporal operators
  - $X$ (next - state)
  - $F$ (eventually), $G$ (globally)
  - $U$ (until), $R$ (release)

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 $\neg (M, \pi \models \varphi)$
- $M, \pi \models \varphi_1 \land \varphi_2$ iff $M, \pi \models \varphi_1$ and $M, \pi \models \varphi_2$

neXt-state operator of LTL

- $M, \pi \models X\varphi$ iff $M, \pi_1 \models \varphi$
  - Path starting from next state satisfies $\varphi$

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 $\varphi$
Globally operator of LTL

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

A trace satisfying $pUq$, where $p, q \in \text{Prop}$

Copyright 2009 by Abhik Roychoudhury

Until operator of LTL

- $M, \pi \models \varphi_1 U \varphi_2$ iff $\exists k \geq 0$ such that
  - $M, \pi^k \models \varphi_2$, and
  - $\forall 0 \leq j < k \, M, \pi^j \models \varphi_1$

Copyright 2009 by Abhik Roychoudhury

Release operator of LTL

- $M, \pi \models \varphi_1 R \varphi_2$ iff
  - Either $\forall k \geq 0 \, M, \pi^k \models \varphi_2$
  - OR both of the following hold
    - $\exists k \geq 0 \, M, \pi^k \models \varphi_1$
    - $\forall 0 \leq j < k \, M, \pi^j \models \varphi_2$
- $\varphi_1$ releases the req. for $\varphi_2$ to hold.

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.

Exercise – (2)

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

LTL Exercise – (3)

Consider a resource allocation protocol where n processes P1, …, Pn are contending for exclusive access of a shared resource. Access to the shared resource is controlled by an arbiter process. The atomic proposition reqi is true only when Pi explicitly sends an access request to the arbiter. The atomic proposition gnti is true only when the arbiter grants access to Pi. Now suppose that the following LTL formula holds for our resource allocation protocol.

\[ G (req_i \Rightarrow F gnt_i) \]

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?
Encoding test specifications

Def. 1
A trace $\sigma$ satisfies a test specification $M$ if $\sigma$ 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 $\sigma = e_1, e_2, \ldots, e_n$ define
  - $\text{prop}_\sigma = F(e_1 \land X(e_2 \land X(\ldots X(e_n)\ldots)))$
- Define property $\phi_M$ corresponding to $M$ as
  - $\phi_M = \neg \left( \lor_{\sigma \in \text{Lin}(M)} \text{prop}_\sigma \right)$
- A counter-example to $\phi_M$ is a test satisfying $M$. 

Example

<table>
<thead>
<tr>
<th>P</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>

Possible linearizations
- $e_1, e_2, e_3, e_4$
- $e_1, e_3, e_2, e_4$

LTL property
- $\neg (F(e_1 \land X(e_2 \land X(e_3 \land X(e_4))))$
- $F(e_1 \land X(e_2 \land X(e_3 \land X(e_4))))$

A counter-example to $\phi_M$ is a test satisfying $M$. 

Encoding test specifications

Def. 2
A trace $\sigma$ satisfies a test specification $M$ if $\sigma$ 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 $\sigma = e_1, e_2, \ldots, e_n$ define
  - $\text{prop}_\sigma = \neg (e_1 \lor e_2 \lor \ldots \lor e_n)$
  - $\text{prop}_\sigma = (\bigvee_{\sigma \in \text{Lin}(M)} \text{prop}_\sigma)$
- Define property $\phi_M$ corresponding to $M$ as
  - $\phi_M = \neg \left( \lor_{\sigma \in \text{Lin}(M)} \text{prop}_\sigma \right)$
- A counter-example to $\phi_M$ is a test satisfying $M$. 

Example

<table>
<thead>
<tr>
<th>P</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$e_2$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>

Possible linearizations
- $e_1, e_2, e_3, e_4$
- $e_1, e_3, e_2, e_4$

LTL property
- $\neg (F(e_1 \land X(e_2 \land X(e_3 \land X(e_4))))$
- $F(e_1 \land X(e_2 \land X(e_3 \land X(e_4))))$

A counter-example to $\phi_M$ is a test satisfying $M$. 

Model Checking – Next class

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

Model Checking Check $M \models \phi$

Yes

No, with Counter-example trace