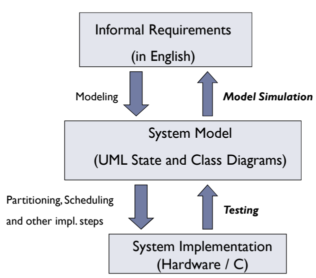
Model-based testing
Specifications – temporal logics

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

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.

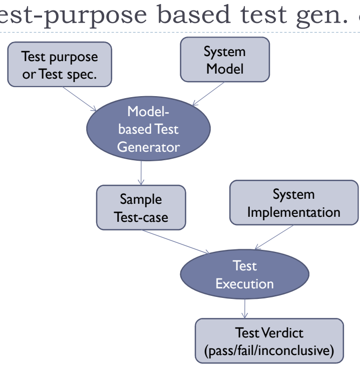
Model-based system development



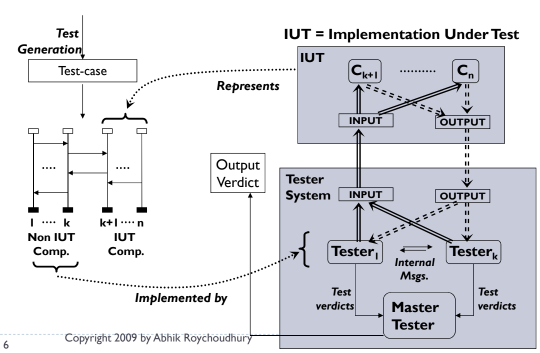
Model-based testing

- 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)*

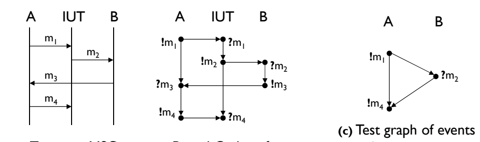
Test-purpose based test gen. & exec.



Test Execution Architecture



Test Execution – (1)



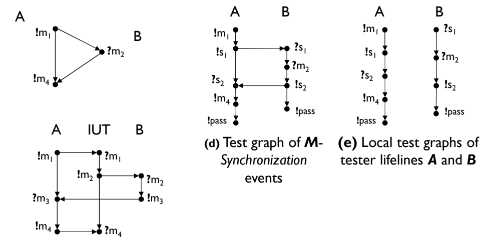
(a) Test-case MSC ***M***

(b) Partial Order of ***M***

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

Test Execution – (2)

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

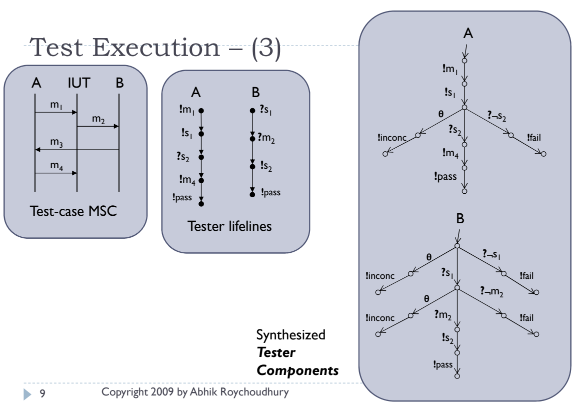


(d) Test graph of ***M***-*Synchronization* events

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

(b) Partial Order of ***M***

Test Execution – (3)

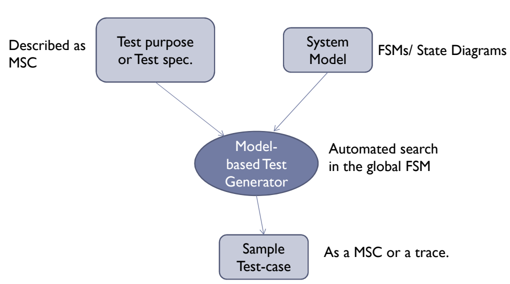


Synthesized ***Tester Components***

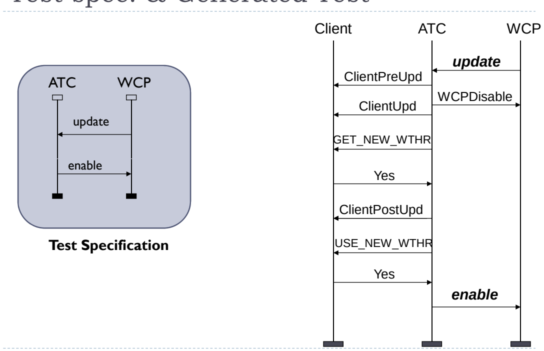
Test Verdicts

- 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



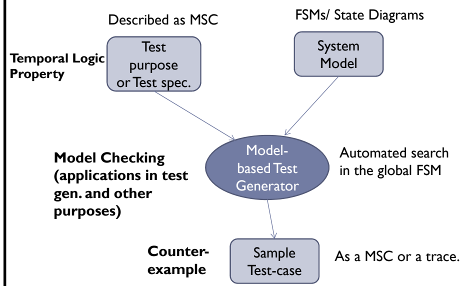
Test spec. & Generated Test



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?

Test Generation



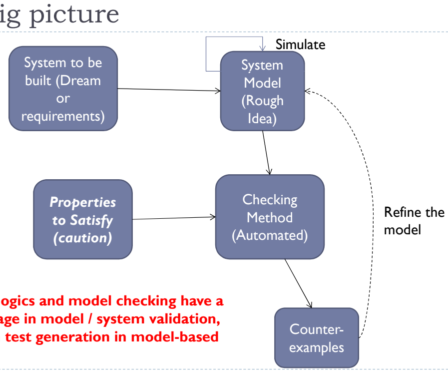
Test generation

- 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

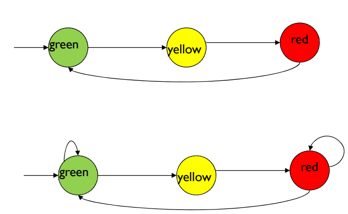
- So Far
  - What is a Model?
  - ATC – Running Example
  - How to model such requirements
  - How to validate the models
    - Simulations,
    - Model-based testing,
    - Model Checking (discussed now)
      - Temporal logics (the property specification)
      - Checking method
    - Also, model-based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

The big picture



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

Example System Model - FSMs



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.

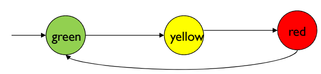
Example properties

- 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?

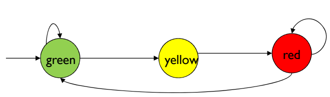
- A property is interpreted on the traces of a system model.
  - Given a trace of the system model x and a property p, we can uniquely determine a yes/no answer to whether x satisfies p.
- A property p is satisfied by a system model M, if all traces of M satisfy p.
- So, given a system model what are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

Traces of a system model



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

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) $^{\omega}$

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.

Why study new logics ?

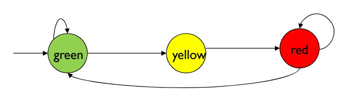
- Our properties express constraints on dynamic evolution of states.
- Propositional/first-order logics can only express properties of states, not properties of traces
- We study behaviors by looking at all execution traces of the system.
  - Linear-time Temporal Logic (LTL) is interpreted over execution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

Formally, system model is

- Model for reactive systems
  - $M = (S, S0, \rightarrow, L)$
  - S is the set of states
  - $S0 \subseteq S$ 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.

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 φ is interpreted over an infinite sequence of states π = s0,s1, …
  - Use M,π |= φ 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 |=φ iff M,π |= φ for all traces π in system model M**

Notations and Conventions

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

LTL - syntax

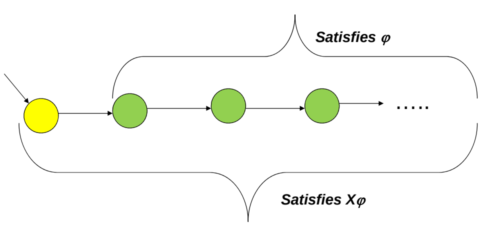
- Propositional Linear-time Temporal logic
- φ = Xφ | Gφ | Fφ | φ U φ | φ R φ |
  ¬φ | φ ∧ φ | Prop
- Prop is the set of atomic propositions
- Temporal operators
  - X (next – state)
  - F (eventually), G (globally)
  - U (until), R (release)

Semantics of propositional logic

- M,π |= p iff s0 |= p i.e. p ∈ L(s0) where L is the labeling function of Kripke Structure M
- M, π |= ¬ φ iff ¬ (M, π |= φ)
- M, π |= φ1 ∧ φ2 iff M, π |= φ1 and M, π |= φ2

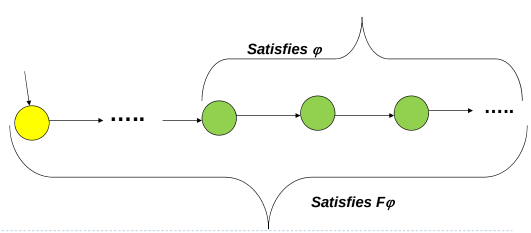
neXt-state operator of LTL

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



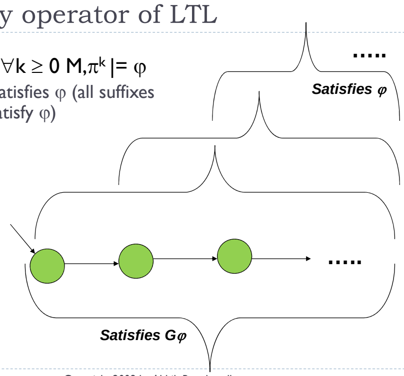
Finally operator of LTL

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



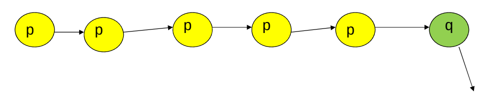
Globally operator of LTL

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



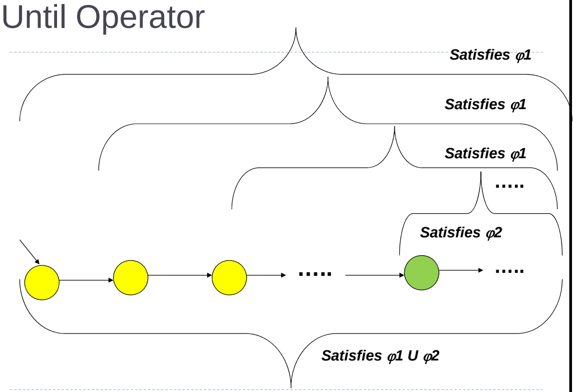
Until operator of LTL

- $M,\pi \models \varphi1\ U\ \varphi2$ iff $\exists k \geq 0$ such that
  - $M,\pi^k \models \varphi2$, and
  - $\forall 0 \leq j < k$ $M, \pi^j \models \varphi1$



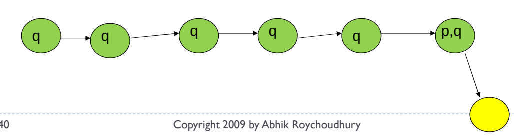
A trace satisfying pU q, where p,q ∈ Prop

Until Operator

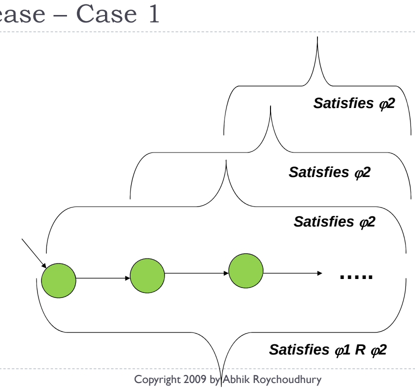


Release operator of LTL

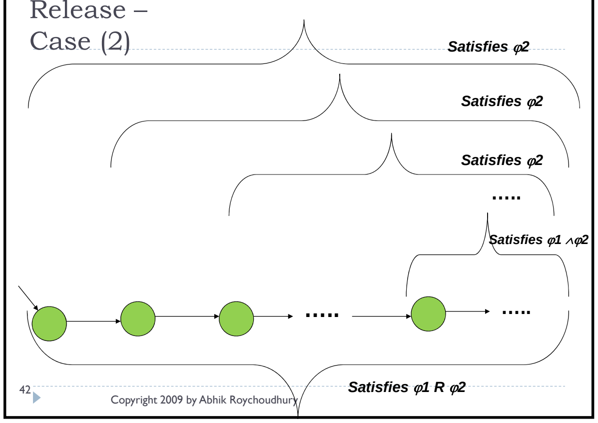
- $M,\pi \models \varphi1\ R\ \varphi2$ iff
  - Either $\forall k \geq 0$ $M,\pi^k \models \varphi2$
  - OR both of the following hold
    - $\exists k \geq 0$ $M,\pi^k \models \varphi1$
    - $\forall 0 \leq j \leq k$ $M,\pi^j \models \varphi2$
- $\varphi1$ releases the req. for $\varphi2$ to hold.



Release – Case 1



Release – Case (2)



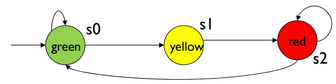
Exercise – (1)

- 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)

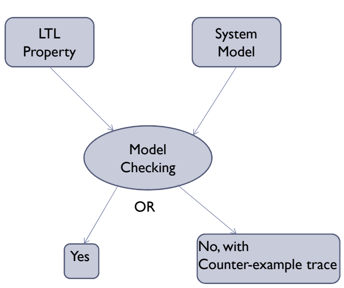
Consider a resource allocation protocol where n processes $P_1,\ldots,P_n$ are contending for exclusive access of a shared resource. Access to the shared resource is controlled by an arbiter process. The atomic proposition req_i is true only when P_i explicitly sends an access request to the arbiter. 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.

- $G\ (req_i \Rightarrow F\ gnt_i)$

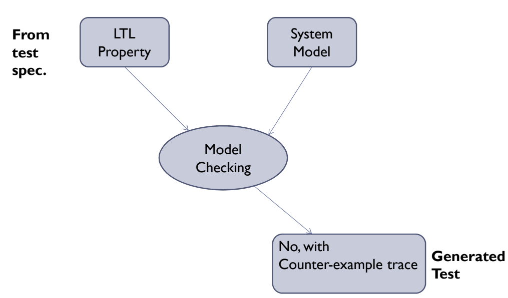
LTL Exercise – (3)

- 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 ?

Model Checking



Recap: Model Checking for model-based testing

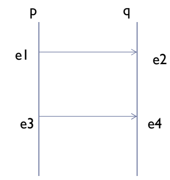


Encoding test specifications

- 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 Lin(M) = set of linearizations of M.
    - For each linearization $\sigma = e_1, e_2, \ldots, e_k$ define
      - Define $prop_\sigma = \mathbf{F}(e_1 \wedge \mathbf{X}(e_2 \wedge \mathbf{X}(\ldots \mathbf{X}(e_k)\ldots)))$
    - Define property φ_M corresponding to M as
      - $\varphi_M = \neg (\vee_{\sigma \in Lin(M)} prop_\sigma)$
- A counter-example to φ_M is a test satisfying M.

49

Example



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

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

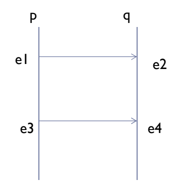
50

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 Lin(M) = set of linearizations of M.
    - For each linearization $\sigma = e_1, e_2, \ldots, e_k$ define
      - $n_\sigma = \neg(e_1 \vee e_2 \vee \ldots \vee e_k)$
      - $prop_\sigma = (n_\sigma\ \mathbf{U}\ (e_1 \wedge \mathbf{X}(n_\sigma\ \mathbf{U}(e2 \wedge \mathbf{X}(\ldots \mathbf{X}(n_\sigma\ \mathbf{U}\ ek)\ldots))))$
    - Define property φ_M corresponding to M as
      - $\varphi_M = \neg (\vee_{\sigma \in Lin(M)} prop_\sigma)$
- A counter-example to φ_M is a test satisfying M.

51

Example



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

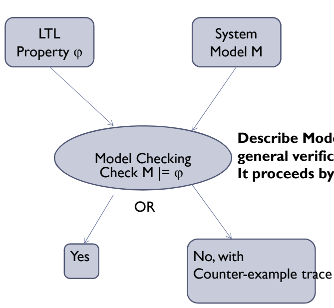
LTL property

??

(try this as an exercise)

52

Model Checking – Next class



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

53