Temporal Logics

Abhik Roychoudhury
Department of Computer Science
National University of Singapore

Note

- I have several slides entitled
  - **Class Practice**
- These are for online practice **during** the class, to allow more classroom interaction.
- I will hesitate to move to the next topics until we solve these on the way.

Organization

- Basics/Warmup
  - Transition Sys -- Follow-up from last class
- Linear Time Logics
- Branching Time Logics
- Fixed point characterizations

Background

- Checking of programs
  - P $|= \varphi$
    - How to represent P ?
      - Implementation
    - How to represent φ?
      - Specification
- Today we will answer the second question

What about the first question?

- Assume a transition system like model of the program
  - M = (I, S, $\rightarrow$)
    - I is set of initial states
    - S is set of states
    - $\rightarrow$ is set of transitions
- What are the states and transitions in a program?

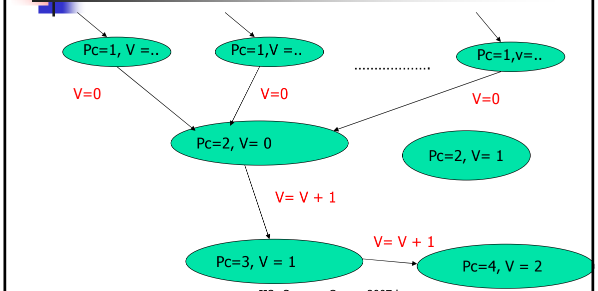
States and Transitions

- States of a sequential program
  - (PC, Value of x, Value of y, …)
  - Infinitely many states in general
- Transition of a program
  - s1 $\rightarrow$ s2
  - s1, s2 are states as defined above
  - Move from s1 to s2 accomplished by executing a statement

Exercise 0

- Draw the transition system corresponding to the following program
  - V = 0;
  - V = V + 1;
  - V = V + 1;

Solution



Exercise 1

- What changes will we need to make in defining states and transitions of say a multi-threaded program?
  - Simple Example
    - Initially x == 0, y == 0
    - do{ x = 1} forever || do { y = 1 } forever

Related to Exercise 1

- Once a "thread" of a concurrent program is scheduled
  - How long can it execute without interruption from other "threads" ?
    - One statement ?
    - One instruction ?
  - This determines possible behaviors
    - Examples on this now !

One statement at a time

- Consider an asynchronous composition of two processes i.e. in
- any time step only one of them makes a move. These processes communicate via a single shared variable x. Both processes are executing the following infinite loop:
  - ***while true do x := x + x***
- Every time one of the processes is scheduled, it atomically
- executes x := x + x and then again another process is scheduled. The initial value of x is 1.
- What will be the values of x reached during system execution and why ?

One instruction at a time

- Suppose the infinite loop is compiled by a naïve compiler as follows. The sequence of instructions
- executed by process A and process B are shown.
- The processes are running asynchronously, and each time a process is scheduled, only its next instruction is executed
- atomically. Initially x = 1.

What values will x reach during system execution in this situation ? Explain your answer. Note that x is a shared global variable and reg_A^i, reg_B^i are local registers in processes A and B respectively.

A version of this problem is originally credited to the Thread Game by Prof. J. Strother Moore (University of Texas).

Class Practice

- L1: reg_A^1 = x
- reg_A^2 = x
- reg_A^3= reg_A^1+reg_A^2
- x = reg_A^3
- go to L1

- L2: reg_B^1 = x
- reg_B^2 = x
- reg_B^3= reg_B^1+reg_B^2
- x = reg_B^3
- go to L2

What are the possible values of x at the end of the program, assuming that at least one instruction is executed when a thread is scheduled ?

This is a follow-up from last lecture, did you try this exercise ?

How to abstract?

- Our definition of states and transitions results in infinitely many states even for sequential programs. What abstractions can you suggest?
  - Should we abstract the PC?
  - Should we abstract the variable values?

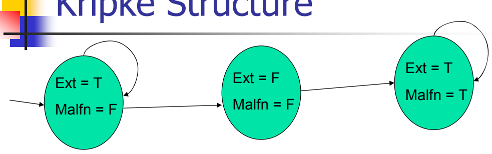
How to abstract?

- Revisit the program
  - V = 0;
  - V = V +1 ;
  - V = V + 1;
- Abstract V using the propositions
  - {V == 0}
- Reconstruct the transition system.

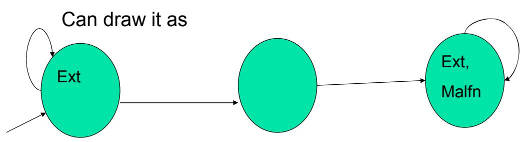
Program Models

- Kripke Structure
  - Transition System (States and Transitions)
    - Can depict the control flow in a program.
  - Set of Propositions Prop
    - Abstraction of the data variables.
  - Truth/Falsehood of each proposition in Prop in each state of the transition system.
    - Abstraction of data flow in the program.

Kripke Structure



Can draw it as



Kripke Structure

- M = (S, I, $\rightarrow$, L)
  - S, set of states
  - I $\subseteq$ S, set of initial states
  - $\rightarrow$, Transition Relation
  - L, Labeling function
    - Label states with propositions – these are the propositions which are true in the state.

Kripke Structure

- L : S → 2^{AP} is a labeling function which assigns a set of atomic propositions to a state
  - L allows us to describe the truth/falsehood of a proposition in the various system states.
  - The propositions refer to valuations of state variables.
- Kripke structures are powerful enough to model behaviors of
  - sequential as well as concurrent systems,
  - software as well as hardware.

Model Checking

- Generate models (Kripke Structure) from the program
  - Represent the control flow explicitly (Control Flow Graph or variants).
  - Abstract data store via predicate abstraction.
    - Implicitly blows up the CFG.
  - State space search now involves traversing the blown up graph (note: symbolic search).

Our Program Checking

- Specify property in temporal logic (Today's lecture)
  - Clock time is not explicitly represented.
  - Temporal modalities to capture dynamic program behavior
- Verify property automatically via search
  - Return "yes" if true
  - Return counterexample "evidence" if false.

Organization

- Basics/Warmup
- Linear Time Temporal Logics
- Branching Time Logics
- Fixed point characterizations

Temporal Logic

- Propositional / First-order logic formulae
  - Capture properties of states
  - Cannot capture properties of state changes by default.
- Temporal Logic formulae
  - Capture properties of evolution of states (program behavior)
  - Possible program behaviors can be
    - Execution traces OR Execution Tree

What is temporal ?

- Consider ordering of events in system execution.
  - Describes properties of such orderings
    - Whenever V = 0, U is not 0
    - Always V = 0
    - Whenever V=0, eventually U = 0
  - Exact time is not represented e.g.
    - V =0 will happen at t = 22 secs.

Purpose of TL

- Describe properties of program behavior
- What are the units of behavior
  - Computation Tree of the program
  - Computation traces of the program
- Linear and branching time logics.
- TL can specify behavior of
  - Terminating and non-terminating programs.
  - Sequential as well as concurrent programs.

LTL: What ?

- Linear-time Temporal Logic
- An LTL formula is interpreted over infinite execution traces.
- LTL can capture properties of
  - Usual terminating programs
  - Non-terminating reactive programs which are continuously interacting with environment
    - Example: Controller software

LTL syntax

- An LTL property φ is true for a program model iff all its traces satisfy φ
  - **Why should be there be more than one trace?**
- φ = Xφ | Gφ | Fφ | φ U φ | φ R φ |
  $\neg\varphi$ | $\varphi \wedge \varphi$ | Prop
- Prop denotes the set of Propositions.
- X, G, F, U, R are temporal operators.
- Building a temporal logic above propositional logic (included above)

LTL semantics

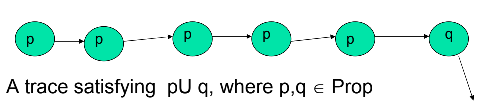
- $M,\pi \models \varphi$
  - Path $\pi = s_0, s_1, s_2, \ldots$ in model M satisfies property φ
- $M,\pi^k \models \varphi$
  - Notation for
    Path s_k , s_{k+1} , … in model M satisfies the property φ
- Semantics for different temporal operators are now given.

Semantics of LTL operators

- $M,\pi \models X\varphi$ iff $M,\pi^1 \models \varphi$
  - Path starting from next state satisfies φ
- $M,\pi \models F\varphi$ iff $\exists k \geq 0$ $M,\pi^k \models \varphi$
  - Path starting from an eventually reached state satisfies φ
- $M,\pi \models G\varphi$ iff $\forall k \geq 0$ $M,\pi^k \models \varphi$
  - Path always satisfies φ (all suffixes of the path satisfy φ)

Semantics of LTL operators

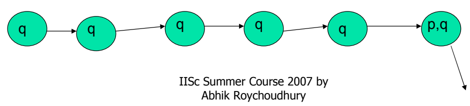
- $M,\pi \models \varphi1$ U $\varphi2$ iff $\exists k \geq 0$ such that
  - $M,\pi^k \models \varphi2$, and
  - $\forall 0 \leq j < k$ $M, \pi^j \models \varphi1$



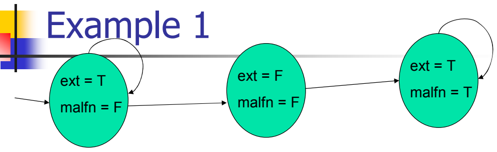
A trace satisfying pU q, where p,q $\in$ Prop

Semantics of LTL operators

- $M,\pi \models \varphi1 \; R \; \varphi2$ 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.



Example 1

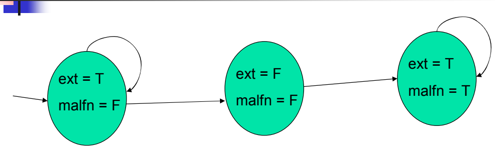


Property: G ext

M |= G ext iff for traces π in M we have M, π |= G ext

Any trace which does not satisfy G ext is a counter-example. How many counterexamples can you find in the above ??

Example 2



What about the formulae: FG ext, GF ext ??

How many counter-examples for each ?

Class Practice

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.

- **$G\,(req_i \Rightarrow (req_i \; U \; gnt_i))$**

Class Practice

- 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 ?

Class Practice

- Express each of the following properties (stated in English) as an LTL formula. Assume that p, q and r are atomic propositions.
  - Always if p occurs, then eventually q occurs followed immediately by r.
  - Any occurrence of p is followed eventually by an occurrence of q. Furthermore, r never occurs between p and q.

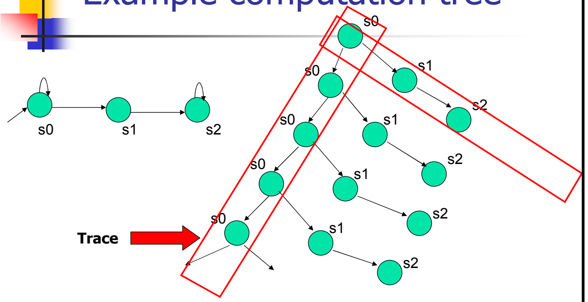
Organization

- Warmup
- Linear Time Logics
- Branching Time Logics
- Fixed-point characterizations

Computation traces/trees

- LTL describes properties of computation traces (a property holds in a sys. if all traces satisfy it).
- Alternate way to characterize system dynamics
  - Computation tree
  - Start from an initial state and unfold the Kripke structure to construct an infinite tree (in general).
- Logics to describe properties of such trees
  - Existential / Universal quantification over paths
  - Temporal operators to specify properties of a path.

Example computation tree



A logic for trees

- s = Prop | $\neg$ s | s /\ s | **A** p | **E** p
- p = **X** s | **G** s | **F** s | s **U** s | s **R** s

- p denotes formulae of paths.
- s denotes formulae of states.
- The temporal operators are as before.
- Computation Tree logic (CTL).
  - All state formulae as defined above.

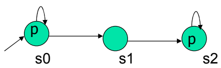
CTL semantics

- A model satisfies a CTL formula iff its initial states satisfy the formula.
- A state x satisfies the formula A p iff all outgoing paths from x satisfy the formula p
  - Note that p must be a path formula
- A state x satisfies the formula E p iff there exists an outgoing path from x satisfying the formula p
- What about the temporal operators ?

CTL semantics

- Semantics of temporal operators (X,F,G,U,R)
  - Minimally modified to handle …
  - A path satisfies a state formula iff its first state satisfies the formula.
  - Try a path proof obligation π |= FEG p
- Exercise : Do it Now !
  - Meaning of AG, EG, AF, EF (intuitively)
  - Duality of (R, U), (F, G), (A, E)
  - Express F in terms of U
  - Minimal set of temporal & path operators

A Quick Exercise



Model M

Suppose we want to check

M |= AGEF p

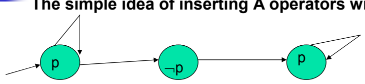
Show all the state and path proof obligations that will be encountered by following through the semantic rules of Computation Tree Logic (CTL).

CTL formulae

- AG f (invariant property)
  - M |= AG f (CTL formula) if and only if M satisfies the LTL formula G f
- AG (f ⇒ EF g)
  - what does it mean ?
  - Involves both path quantifiers
  - Not expressible in LTL
- Does that mean CTL is strictly more powerful than LTL ?

CTL and LTL

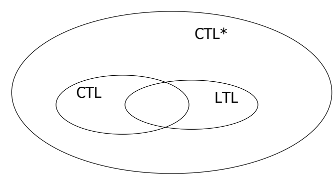
The simple idea of inserting A operators will not work.



Satisfies the LTL formula FG p

What about the CTL formula AFAG p ?

Expressivity



CTL*

- s = Prop | ¬s | s ∧ s | A p | E p
- p = s | ¬p | p ∧ p | Xp | Fp | Gp |
- p U p | p R p
  - s denotes formulae interpreted over states
  - p denotes formulae interpreted over paths
  - CTL* is the set of all state formulae above
  - A p is a state formula not expressed in prop. Logic
- Semantics of path formulae
  - Minimally modified to handle …
  - A path satisfies a state formula iff its first state satisfies the formula.

CTL and CTL*

- CTL is a sublogic of CTL*
  - Temporal operators in CTL*: X,F,G,U,R
  - Path Quantifiers: A, E
  - CTL enforces the occurrence of a temporal operator to be immediately preceded by a path quantifier.
  - AGFφ is not allowed.

Some common CTL formulae

- **AG p**
  - **Invariant: always p.**
- **EF p**
  - **Reachability: of a state where p holds.**
- **AF p**
  - **Inevatibility of reaching a state where p holds.**
- **AG EF p**
  - **Recovery: from any state we can reach a state where p holds.**
- **AG (p ⇒ AF q)**
  - **Non-starvation : *p* request is always provided *q* response.**

Class Practice

- Consider the LTL formula GFp and the CTL formula AGEFp where p is an atomic proposition. Give an example of a Kripke Structure which satisfies AGEFp but does not satisfy GFp. You may assume that p is the only atomic proposition for constructing the labeling function.

- D) Are the following LTL formulae equivalent
  - **G(p ⇒ X p)**
  - **G(p ⇒ G p)**

Class Practice

- Construct a CTL formula which is satisfied by the initial state of the system model in Figure (a), but not satisfied by the initial state of the system model of Figure (b). Since I have shown only the valuation for the atomic proposition q in the different states, it will be the only atomic proposition appearing in your formula. (We can assume that there are other atomic propositions as well, but their valuations are not shown).

Class Practice

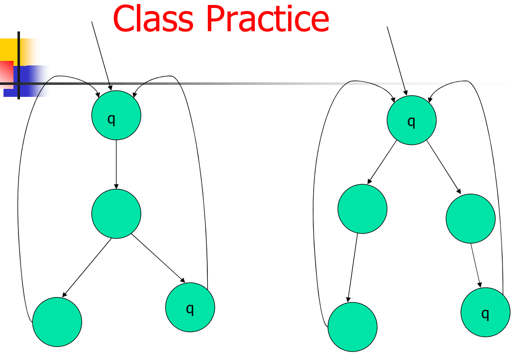


Figure (a) **Figure (b)**

Satisfaction

- A CTL formula is satisfiable if some state of some Kripke structure satisfies it.
  - Otherwise unsatisfiable. Examples ??
  - Similarly for LTL formula .
- A CTL formula is valid if all states of all Kripke structures satisfy it.

Formula Equivalence

- Two temporal properties are equivalent iff they are satisified by exactly the same states of any Kripke structure.
  - EF p and E(true U p)
- Where does model checking stand ??
  - Is it checking for satisfiability of a temporal property ? Is it checking for validity ?

Model Checking

- … is not checking for satisifiability / validity.
- It is checking for satisfaction of a temporal property for a given Kripke structure.
  - This is a very different problem from traditional satisfiability checking !!
- We will discuss MC in next class.
  - But some warm up for now !

Organization

- Warmup
- Linear Time Logics
- Branching Time Logics
- Fixed point characterizations

Fixed point characterizations

- An alternate semantic understanding of temporal formulae such as CTL properties.
- Yields a procedure for model checking these properties directly
  - Correct by construction model checking algorithm.

Intuition -- (1)

- Give semantics of CTL formulae by associating a formula with the states in a Kripke structure in which the formula will be true.
- A set of states drawn from a Kripke Structure forms a predicate.
- Define functions on sets of states
  - **F: $2^S \rightarrow 2^S$**
  - **S is the set of all states drawn from Kripke structure**
  - **Such functions are called predicate transformers.**

Intuition – (2)

- Define a CTL formula as the set of states obtained by
  - **Repeated application of a predicate transformer**
  - **Starting from null set or set of all states.**
  - **Ending when one more application of the transformer does not change the result**
    - **Fixed point is reached.**
- For a predicate transformer, there can be several fixed-points. It is possible to show that for predicate transformers with certain properties
  - Fixed point reached by starting from null set is the least fixed point
  - Fixed point reached by starting from set of all states is the greatest fixed point

Intuition (3)

- This gives a characterization of CTL formulae as least or greatest fixed points of transformers.
  - It also gives a computational mechanism for verifying these CTL formulae.
  - Given a Kripke structure M and a formula f, apply the predicate transformer for f until fixed point reached
  - Check whether the initial states of M are in the fixed point.

Set of States

- M = (S, I, $\rightarrow$, L)
  - Assume S is finite
  - $\rightarrow$ (transition relation)
  - L (Labeling function)
- $x \in 2^S$
  - x is a predicate over S
- Powerset 2^S comes with a natural partial order
  - Set inclusion

Monotone Functions

- A function f : $2^S \rightarrow 2^S$
  - Monotonic: $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$
  - Fixed point: f(X) = X
  - X, Y are subsets of S (set of states)
- Since it is transforming sets of states to another set of states, also called
  - Predicate Transformer

Example

- S = {s0, s1}
- f : $2^S \rightarrow 2^S$
  - $f(X) = X \cup \{s0\}$
  - Show that f is monotonic.
  - What are the fixed points of f ?
- f : $2^S \rightarrow 2^S$
  - Exercise: Give an example of non-monotone function.

Role of Monotone functions

- Always have least/greatest fixed points
- Meaning of CTL operators cast as lfp/gfp of monotone func. over 2^S
  - AG,EG,AF,EF,AU,EU,…
- The fixed points can be computed
  - Straightforward checking procedure
    - Compute fixed-point and check for initial states within the fixed-point.

Why fixed points are important ?

- Meaning of CTL operators can be expressed as lfp, gfp of monotone functions.
  - EG, EU, AG, AF, ….
- These lfp, gfp can be easily computed.
- Leads to a straightforward model checking algorithm for CTL formulae.

Class Practice

- If S is finite, then for a monotone function f : $2^S \rightarrow 2^S$
  - $\exists I \in nat$ lfp f = $f^I(\phi)$
  - $\exists I \in nat$ gfp f = $f^I(S)$
- Class Exercise:
  - Let us prove this theorem now.

LFP computation procedure

- Function lfp (f : PredicateTransformer): Predicate
- Begin
- Q := ϕ; // Null-set
- Q1 := f(Q);
- while (Q1 $\neq$ Q) do
- Q := Q1; Q1 := f(Q)
- endwhile;
- return(Q)
- End.

GFP computation procedure

- Function gfp (f : PredicateTransformer): Predicate
- Begin
- Q := S; // All states in the Kripke Structure
- Q1 := f(Q);
- while (Q1 $\neq$ Q) do
- Q := Q1; Q1 := f(Q)
- endwhile;
- return(Q)
- End.

Defining CTL operators

- M = (S,I, $\rightarrow$, L) is a finite Kripke structure
- φ is a CTL formula
- $[\varphi] \subseteq S$ denotes the set of states satisfying φ
- Say we define a predicate transformer
  - $f_\varphi(Y) = [\varphi] \cap \{s \mid \exists s' \ \ s \rightarrow s' \text{ and } s' \in Y\}$

Defining EG

- $\{s \mid \exists s' \ \ s \rightarrow s' \text{ and } s' \in Y\}$
  - Stands for [EX Y], the set of states in M which satisfy EX Y.
  - For convenience we will avoid the […]
- $f_\varphi(Y) = \varphi \cap$ ***EX*** Y
  - Show that this transformer is monotonic.
  - You will need the definition of EX

Defining EG

- Show that EGφ is a fixed point of
  - $f_\varphi(Y) = \varphi \cap$ ***EX*** Y.
  - Any state satisfying EGφ satisfies φ and there is an outgoing state satisfying EGφ
  - The other direction of the proof …
- Show that EGφ is the gfp of
  - $f_\varphi(Y) = \varphi \cap$ ***EX*** Y
  - Any state in a fixed point of f_φ satisfies EG φ
    - Complete the proof exploiting this intuition.

Defining EU

- E(φ U ψ) is the least fixed point of
  - $f_{\varphi,\psi}(Y) = \psi \cup (\varphi \cap \text{EX } Y)$
- Cast the EU checking algo in terms of the LFP computation procedure outlined earlier
- Full discussion on CTL MC in next class.

Property Equivalences

- EG $\varphi = \varphi \wedge$ EXEGφ
- $E(\varphi \text{ U } \psi) = \psi \vee (\varphi \wedge \text{ EX } E(\varphi \text{ U } \psi))$
- We can derive similar equivalences using the fixed point characterization of other CTL properties.

Possible Readings

- Chapter 3 of "Model Checking"
  - (Clarke, Grumberg, Peled)
- Chapter 3 of Logic in Computer Science
  - (Huth and Ryan)
    - Chapter 3.9 contains discussion on fixed point characterizations.
- More articles from course web-page
  - http://drona.csa.iisc.ernet.in/~abhik