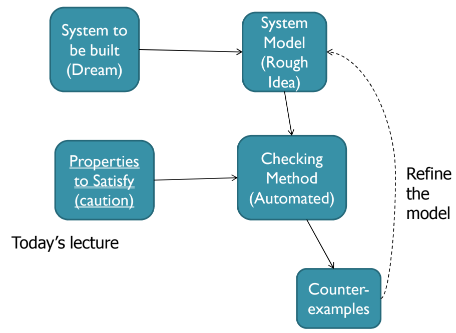
Temporal Logics

CS 4271
Abhik Roychoudhury
National University of Singapore

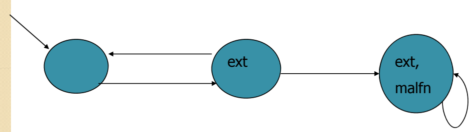
Recap: the big picture



Recap: Kripke Structure

- Model for reactive systems
  - M = (S, S0, R, L)
  - S is the set of states
  - S0 $\subseteq$ S is the set of initial states
  - R $\subseteq$ is the transition relation
    - Set of (source-state, destination-state) pairs
  - L: S $\rightarrow 2^{AP}$ is the labeling function
    - Maps each state s to a subset of AP
    - These are the atomic prop. which are true in s.

An Example



A simplified model of a spring.
AP = {ext, malfn}
ext stands for "the spring is extended"
malfn stands for "the spring is malfunctioning"

Properties

- Does the spring always remain extended ?
- Does the spring remain extended infinitely often ?
- How to specify such properties and reason about them ?
  - This Lecture !

Organization

- General Introduction
- LTL
- CTL*
- CTL – A fragment of CTL*

Atomic propositions

- All of our logics will contain atomic props.
  - These atomic props. will appear in the labeling function of the Kripke Structure you verify.
  - Kripke structure is only a model of your design.
  - Thus the atomic props. represent some relationships among variables in the design that you verify.
  - Atomic props in the previous example
    - **ext, malfn**

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.

Temporal Logics

- The temporal logics that we study today build on a "static" logic like propositional/first-order logic.
  - We work with propositional logic.
  - Used to describe properties of states.
- Temporal operators describe properties on execution traces / trees.
- Time is not explicitly mentioned in the formulae
  - Rather the properties describe how the system should evolve over time.

Example

- 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.

Organization

- General Introduction
- LTL
- CTL*
- CTL - a fragment of CTL*

LTL

- An LTL formula φ is interpreted over and infinite sequence of states π = s0,s1, …
  - Use M,π |= φ to denote that formula φ holds in path π of Kripke Structure M.
- Define semantics of LTL formulae w.r.t. a Kripke Structure M.
  - **An LTL property φ is true of a program model iff all its traces satisfy φ**
  - **M |=φ iff M,π |= φ for all path π in Kripke Structure M**

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 LTL - notations

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

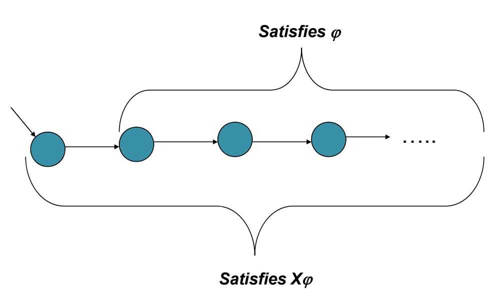
Semantics of LTL

- 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

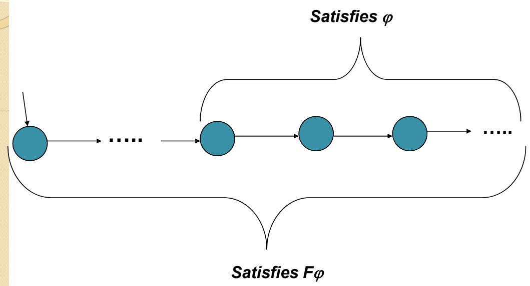
Semantics of LTL

- M,π |= Xφ iff M,π^1 |= φ
  - Path starting from next state satisfies φ
- M,π |= Fφ iff ∃k ≥ 0 M,π^k |= φ
  - Path starting from an eventually reached state satisfies φ
- M,π |= Gφ iff ∀k ≥ 0 M,π^k |= φ
  - Path always satisfies φ (all suffixes of the path satisfy φ)

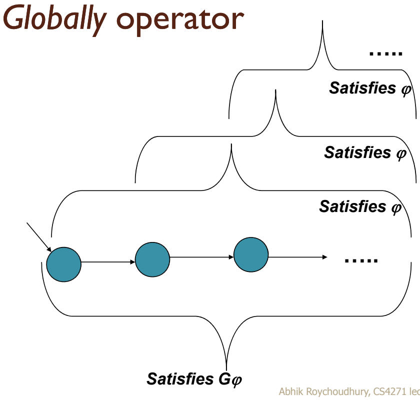
neXt-state operator in LTL



Finally operator in LTL

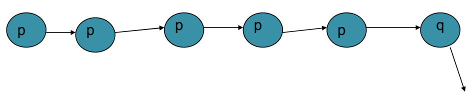


Globally operator



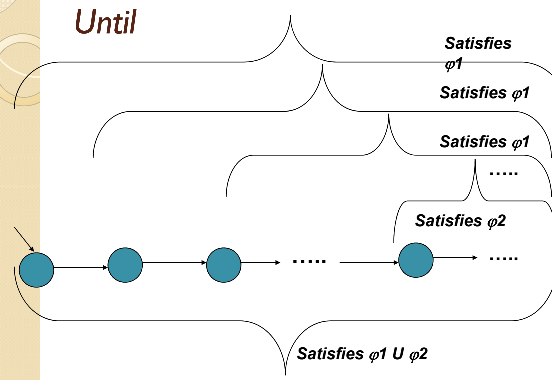
Semantics of LTL

- M,π |= φ1 U φ2 iff ∃k ≥ 0 such that
  - M,π^k |= φ2, and
  - ∀0≤ j < k M, π^j |= φ1



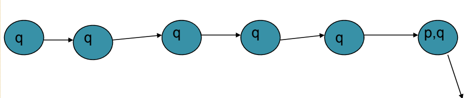
A trace satisfying pU q, where p,q ∈ Prop

Until

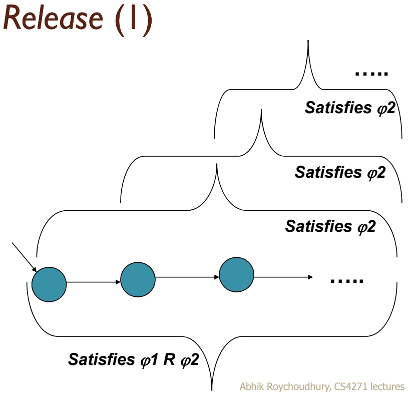


Semantics of LTL

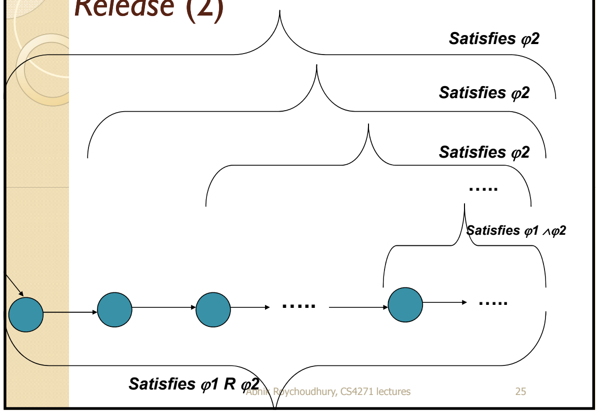
- M,π |= φ1 R φ2 iff
  - Either ∀k ≥ 0 M,π^k |= φ2
  - OR both of the following hold
    - ∃k ≥ 0 M,π^k |= φ1
    - ∀0≤ j ≤ k M,π^j |= φ2
- φ1 releases the req. for φ2 to hold.



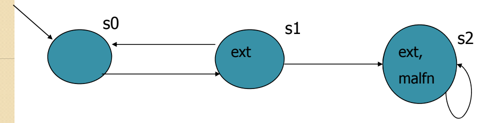
Release (1)



Release (2)



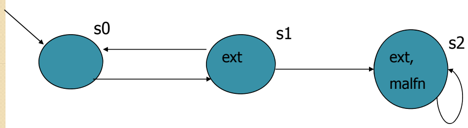
Revisiting the Spring Example



What are the traces of this spring model ?

Does this model satisfy the formula "The spring always remain extended" ? How to write it in LTL and check its correctness in this model ?

Revisiting the Spring Example



What do the properties FG ext and GF ext denote ?

Are they true in this spring model ? If not how many traces violating each property can you find ?

What about the property $\neg$ ext U malfn ?

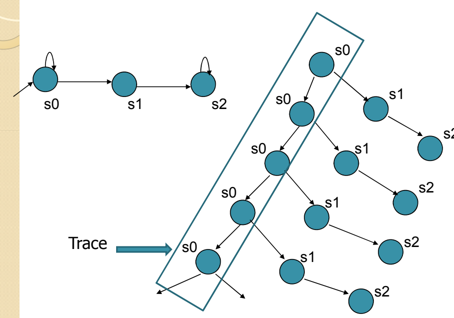
Organization

- General Introduction
- LTL
- CTL*
- CTL - a fragment of CTL*

CTL* Introduction

- LTL formulae describe properties of computation traces.
- CTL* formulae describe properties of computation trees.
- Given a Kripke Structure M, computation tree formed by starting at initial state and unfolding M to construct an infinite tree.
- A CTL* formula then describes properties that this infinite tree should satisfy.
  - LTL is contained within CTL*
  - All LTL properties are CTL* properties.
  - All CTL* properties are not LTL properties.

Example computation tree



Traces vs. Tree (IMPORTANT)

- LTL formulae describe properties of computation traces of a Kripke Structure
- CTL* formulae describe properties of computation tree of a Kripke Structure.

- Given a Kripke Structure M
  - a LTL formula φ is true iff it is true for all the traces of M
  - a CTL* formula φ is true iff it is true for the computation tree of M

Traces vs. Tree (IMPORTANT)

Given a Kripke Structure M

- a LTL formula φ is true iff it is true for all the traces of M
- a CTL* formula φ is true iff it is true for the computation tree of M

- Associate states with computation tree rooted there.
  - Interpret a CTL* formula to be true in a state s iff it is true in the computation tree rooted at s
  - Thus a CTL* formula is true in a Kripke structure M, iff it is true in the initial states of M.

CTL* formulae

- Once again choose propositional logic as the underlying static logic.
- We need to consider two flavors of formulae:
  - State formula: Property of a state
  - Path formula: a property of a computation path
- All LTL formulae are path formulae
- Are the state formulae same as the formulae in the underlying static logic ?
  - NO, refers to system evolution from a state.

Operators in CTL*

- Path Quantifiers
  - Describe properties on the branching structure of the computation tree
    - A : for all computation paths, …
    - E : there exists a computation path, …
- Temporal Operators
  - Same as LTL operators. Describe properties of a trace in the computation tree.

CTL* Syntax

- *State* = Prop | $\neg$*State* | *State* $\wedge$ *State* |
- A *Path* | E *Path*
- *Path* = *State* | $\neg$*Path* | *Path* $\wedge$ *Path* |
- X *Path* | F *Path* | G *Path* |
- *Path* U *Path* | *Path* R *Path*
  - *State* denotes formulae interpreted over states
  - *Path* denotes formulae interpreted over paths
  - CTL* is the set of all state formulae above.
  - AFp is a state formula not expressed in prop. Logic
    - (Assume p is an atomic proposition.)

Examples

- State formulae
  - AG ext
  - AG (ext $\Rightarrow$ EF malfn)
  - AG (ext $\Rightarrow$ F malfn)
- Path formulae
  - G ext
  - G (ext $\Rightarrow$ F malfn)

Path formulae

- Same as LTL formulae
- Furthermore, any state formula is a path formula.
- How to interpret a state formula over a path ?
  - Coming soon …

Path Quantifiers

- A, E
  - Denote universal/existential quantification over all computation paths starting from a state.
  - A φ holds in a state s if for all computation paths starting from s, the path formula φ holds.
  - E φ holds in a state s if there exists a computation path starting from s, for which the path formula φ holds.

Semantics of CTL*

- Define semantics of a formula w.r.t. a Kripke Structure M
- A state formula φ holds in a state s of M denoted as
  - $M, s \models \varphi$
- A path formula φ holds in a path π of M denoted as
  - $M, \pi \models \varphi$
- Recall that syntax of path formulae are
  - *Path* = *State* | $\neg$*Path* | *Path* $\wedge$ *Path* |
  - X *Path* | F *Path* | G *Path* |
  - *Path* U *Path* | *Path* R *Path*

Semantics of path formulae

- Defined as before (all LTL formulae)
  - $M,\pi \models X\varphi$
  - $M,\pi \models G\varphi$ $M,\pi \models F\varphi$
  - $M,\pi \models \varphi 1 \ U \ \varphi 2$ $M, \pi \models \varphi 1 \ R \ \varphi 2$
  - $M, \pi \models \varphi 1 \wedge \varphi 2$ $M, \pi \models \neg\varphi$
- $M, \pi \models \varphi$ (a state formula) holds if φ holds in the first state of π

Semantics of CTL* state formula

- Syntax
  - *State* = Prop | $\neg$*State* | *State* $\wedge$ *State* |
  - A *Path* | E *Path*
- Ingredients:
  - Atomic propositions
  - Negation
  - Conjunction
  - Universal Path Quantifier
  - Existential Path Quantifier
    - Using our intuitive understanding of each, can we give the formal semantics ?

Semantics of CTL* State Formulae

- $M,s \models p$ iff
  - $p \in L(s)$ where $M = (S,S0,R,L)$
- $M,s \models \neg\varphi$ iff
  - not $M,s \models \varphi$
- $M,s \models \varphi_1 \wedge \varphi_2$
  - $M,s \models \varphi_1$ and $M,s \models \varphi_2$
- $M,s \models A\varphi$ iff
  - for every path π starting from s s.t. $M, \pi \models \varphi$
- $M,s \models E\varphi$ iff
  - there exists a path π starting from s s.t. $M, \pi \models \varphi$

LTL and CTL*

- LTL is strictly less expressive than CTL*
- All LTL formulae are path formulae and not state formulae
- They can be converted to CTL* formulae by quantifying over all paths using A
  - Implicit in the semantics of LTL formulae

LTL and CTL*

- LTL formula G(ext $\Rightarrow$ F malfn)
  - Equivalent to CTL* formula
    - A(G ext $\Rightarrow$ F malfn)
- Example of a CTL* formula not expressible in LTL
  - AG(ext $\Rightarrow$ EF malfn)
  - CTL* is a strictly more powerful logic.

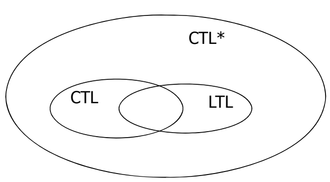
Organization

- General Introduction
- LTL
- CTL*
- CTL - a fragment of CTL*

CTL

- 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

Expressivity



CTL* and CTL syntax

- CTL*
  - *State* = Prop | $\neg$*State* | *State* $\wedge$ *State* | A *Path* | E *Path*
  - *Path* = *State* | $\neg$*Path* | *Path* $\wedge$ *Path* | X *Path* |
  - F *Path* | G *Path* | *Path* U *Path* | *Path* R *Path*
- CTL
  - *State* = Prop |$\neg$ *State* |*State* /\ *State* | A *Path* | E *Path*
  - Path = X *State* | G *State* | F *State* |
  - *State* U *State* | *State* R *State*
- This leads to:

Syntax of CTL

- φ = true | false | Prop | $\neg\ \varphi$ | $\varphi \wedge \varphi$ |
- AX φ | EX φ |
- AG φ | EG φ |
- AF φ | EF φ |
- A(φ U φ) | E(φ U φ) |
- A(φ R φ) | E (φ R φ)

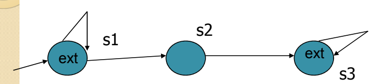
Interpreting CTL formulae

- Similar to CTL* formulae
  - Again CTL formulae are property of computation trees.
- Given a Kripke Structure M
  - a CTL formula φ is true iff it is true for the computation tree of M
- Associate states with computation tree rooted there.
  - Interpret a CTL formula to be true in a state s iff it is true in the computation tree rooted at s
  - Thus a CTL formula is true in a Kripke structure M, iff it is true in the initial states of M.

Relationship to other logics

- Sublogic of CTL*
  - AGFext not in CTL
- Incomparable to LTL
  - AGEFext not expressible in LTL
  - LTL formula FGext not expressible in CTL
    - Not equivalent to the CTL formula AFAGext

Relationship of CTL and LTL



Satisfies the LTL formula FG ext

What about the CTL formula AFAG ext ?

Starting from initial state

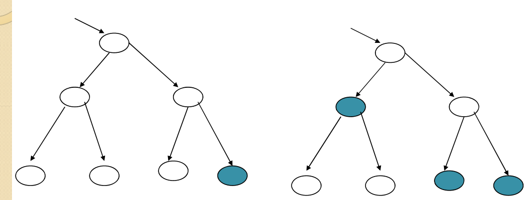
Along all outgoing paths, eventually we reach a state s.t.

along all outgoing paths globally **ext** holds

CTL operators

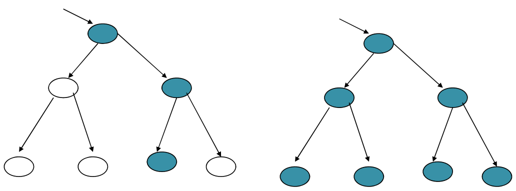
- Most common operators
  - AF, EF, AG, EG
  - Pictorial description of each now !
    - We only show a finite part of an inf. Computation tree
- Other operators: AU, EU, AR, ER, AX, EX
- EX, EG, EU can express all the ten operators (along with $\neg$ and $\wedge$)
  - This will be exploited in CTL model checking algorithm (to be discussed later !)

EFp, AFp



Shaded nodes satisfy p, white nodes do not satisfy p.

EGp, AGp



Shaded nodes satisfy p, white nodes do not satisfy p.

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 ($p \Rightarrow$ AF q)
  - Non-starvation : p request is always provided a q response.

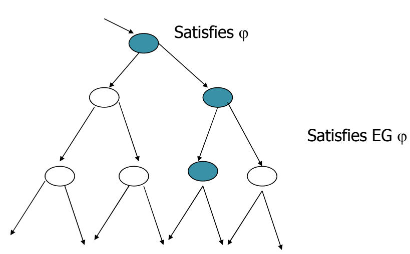
Recursive characterization of CTL formulae

- Take a formula EGφ
- $M, s_0 \models$ EG φ iff
  - There exists a path $s_0, s_1, s_2, s_3, \ldots$
  - Such that $s_i \models \varphi$ for all $i \geq 0$
- Instead think of it as follows:
  - EG φ holds iff
    - φ holds in the current state, and
    - EG φ holds in one of the successor states of the current state.

Recursive characterization of EG

- EGφ = $\varphi \wedge$ EX EG φ
- Note that
  - EX φ holds in a state s, if there exists s' s.t. (s,s') $\in$ R and φ holds in state s'
    - R is the transition relation.
  - EG φ holds in a state s, if there exists s' s.t. (s,s') $\in$ R and EGφ holds in state s'

Recursive characterization of EG



Recursive characterization of CTL

- It is possible to develop such characterizations of other CTL operators
- Online Exercise: Do it now !
  - Recursive characterization of EFφ

Sanity Check

- Give a CTL formula which can be expressed in LTL.
- Give a CTL formula which cannot be expressed in LTL.
- Give a LTL formula which cannot be expressed in CTL.
- Give a CTL* formula which cannot be expressed in CTL.
- Give a CTL* formula which cannot be expressed in LTL.

Wrapping up: Satisfaction

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

Wrapping up: Equivalent formulae

- Two temporal properties are equivalent iff they are satisified by exactly the same Kripke structures.
  - 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 ?

Wrapping up

- Model Checking
  - … is not checking for satisfiability / 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 !!

Before ending …

- Cadence SMV allows user to specify LTL properties.
  - LTL MC is achieved by internally performing CTL MC under fairness constraints.
    - Symbolic LTL MC via symbolic fair CTL MC.
  - How this is done is quite technical and will not be covered in our course.
    - Interested students can refer to Chapter 6.7 of the textbook.