Temporal Logic
- Branching-time logic (2/2)

Moonzoo Kim
CS Dept. KAIST

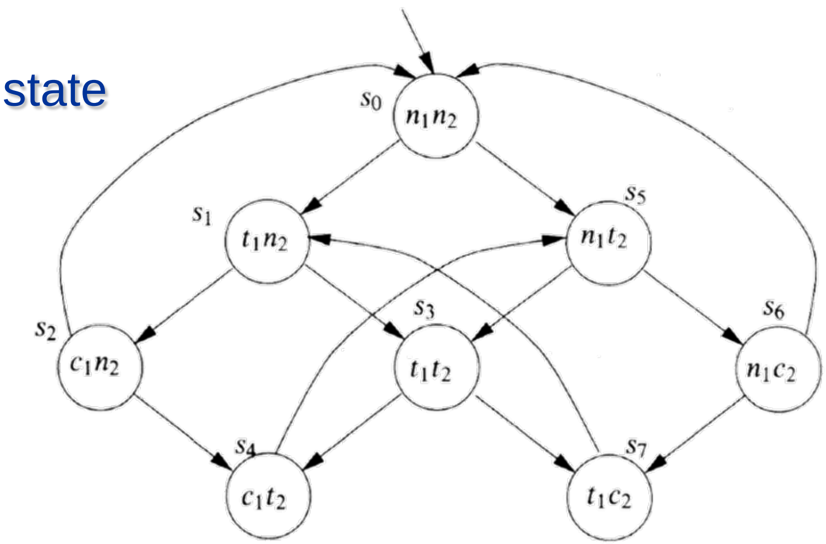
Syntax and semantics of CTL

- Def 3.12 $\phi = \perp \mid \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid$
  $AX\ \phi \mid EX\ \phi \mid AF\ \phi \mid EF\ \phi \mid AG\ \phi \mid EG\ \phi \mid A\ (\phi\ U\ \phi) \mid E\ (\phi\ U\ \phi)$
- Def 3.15 Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, s in S, ϕ a CTL formula. The relation $\mathcal{M},s \vDash \phi$ is defined by structural induction on ϕ.

- $\mathcal{M},s \vDash AX\ \phi$ iff for all s_1 s.t. $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \vDash \phi$. Thus AX says "in every next state"
- $\mathcal{M},s \vDash EX\ \phi$ iff for some s_1 s.t. $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \vDash \phi$. Thus EX says "in some next state"
- $\mathcal{M},s \vDash AG\ \phi$ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, and all s_i along the path, we have $\mathcal{M},s_i \vDash \phi$.
- $\mathcal{M},s \vDash EG\ \phi$ iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, and all s_i along the path, we have $\mathcal{M},s_i \vDash \phi$.
- $\mathcal{M},s \vDash AF\ \phi$ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, and there is some s_i s.t. $\mathcal{M},s_i \vDash \phi$.
- $\mathcal{M},s \vDash EF\ \phi$ iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, and there is some s_i s.t. $\mathcal{M},s_i \vDash \phi$.
- $\mathcal{M},s \vDash A\ [\phi_1\ U\ \phi_2]$ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, that path satisfies $\phi_1\ U\ \phi_2$
- $\mathcal{M},s \vDash E\ [\phi_1\ U\ \phi_2]$ iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, that path satisfies $\phi_1\ U\ \phi_2$

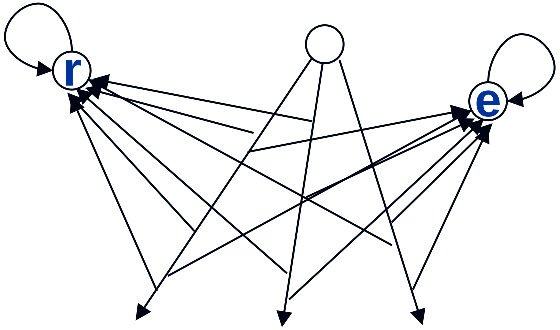
Practical patterns of specification (1/2)

- It is possible to get to a state where `started` holds, but `ready` doesn't
  - EF (`started` $\wedge$ $\neg$`ready`)
- For any state, if a request occurs, then it will eventually be acknowledged
  - AG (`requested` $\rightarrow$ AF `acknowledged`)
- A certain process is enabled infinitely often on every computation path
  - AG (AF `enabled`)
- Whatever happens, a certain process will eventually be permanently deadlocked
  - AF (AG `deadlock`)
- From any state it is possible to get to a restart state
  - AG (EF `restart`)
- Mutual exclusion protocol
  - Non-blocking: a process can always request to enter its critical section
    - AG ($n_1 \rightarrow$ EX t_1)
    - Note that this was not expressible in LTL
  - No strict sequencing: processes need not enter their critical section in strict sequence.
    - EF ($c_1 \wedge$ E [c_1 U ($\neg c_1 \wedge$ E[$\neg c_2$ U c_1])])
    - This was also not expressible in LTL, though we expressed its negation.



Practical patterns of specification (2/2)

- An upwards travelling lift at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:
  - AG (floor2 $\wedge$ directionup $\wedge$ ButtonPressed5 $\rightarrow$ A [directionup U floor5])
- The lift can remain idle on the third floor with its dorrs closed
  - AG (floor3 $\wedge$ idle $\wedge$ doorclosed $\rightarrow$ EG (floor3 $\wedge$ idle $\wedge$ doorclosed))
- The property that if the process is enabled infinitely often, then it runs infinitely often, is not expressible in CTL
  - What about AG AF enabled $\rightarrow$ AG AF running ?



Equivalence between CTL formulas

- Def 3.16 Two CTL formulas ϕ and ψ are said to be semantically equivalent if any state in any model which satisfies one of them also satisfies the other
  - $\phi \equiv \psi$
- $\neg$ AF $\phi \equiv$ EG $\neg\phi$
- $\neg$ EF $\phi \equiv$ AG $\neg$ ϕ
- $\neg$ AX $\phi \equiv$ EX $\neg$ ϕ
- AF $\phi \equiv$ A [T U ϕ]
- EF $\phi \equiv$ E [T U ϕ]

- AG $\phi \equiv \phi \wedge$ AX AG ϕ
- EG $\phi \equiv \phi \wedge$ EX EG ϕ
- AF $\phi \equiv \phi \vee$ AX AF ϕ
- EF $\phi \equiv \phi \vee$ EX EF ϕ
- A [ϕ U ψ] $\equiv \psi$ ($\phi \wedge$ AX A[ϕ U ψ])
- E [ϕ U ψ] $\equiv \psi$ ($\phi \wedge$ EX E[ϕ U ψ])

We can define the six connectives on the left in terms of AX and EX in a non-circular way using fixed-point characterization of CTL

Adequate sets of CTL connectives

- Thm 3.17 A set of temporal connectives in CTL is adequate if, and only if, it contains at least one of {AX, EX}, at least one of {EG, AF, AU} and EU
- $A[\phi \text{ U } \psi] \equiv A[\neg(\neg\psi \text{ U } (\neg\phi \wedge \neg\psi)) \wedge F\,\psi]$

  $\equiv \neg E\neg[\neg(\neg\psi \text{ U } (\neg\phi \wedge \neg\psi)) \wedge F\,\psi]$

  $\equiv \neg E[(\neg\psi \text{ U } (\neg\phi \wedge \neg\psi)) \vee G\neg\psi]$

  $\equiv \neg(E\,[(\neg\psi \text{ U } (\neg\phi \wedge \neg\psi)) \vee EG\,\neg\psi]$

  Note that the proof has intermediate formulas of CTL* which violates the syntax of CTL

Comparison between LTL and CTL

	LTL	CTL
Difficulty of specification	intuitive and easier	Difficult and unintuitive
Model checking complexity	Exponential time	Polynomial time
Limitation	Cannot specify branching behavior	Cannot specify a range of paths
Main target area	Requirement property for software	Requirement property for hardware
Tools	FormalCheck, SPIN, Intel's Prover, NuSMV	NuSMV, VIS