Temporal Logic
- Branching-time logic

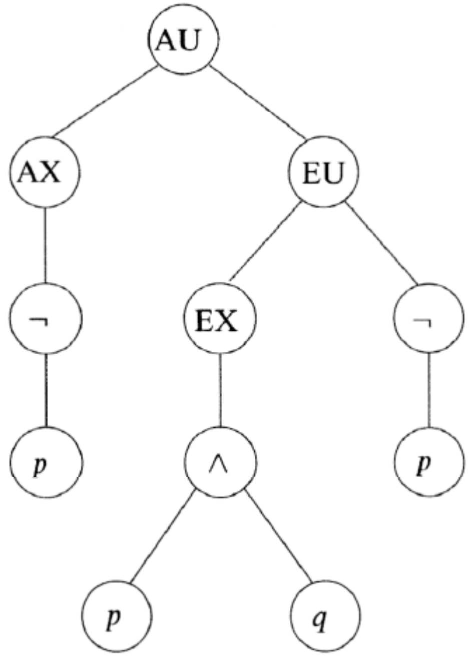
Moonzoo Kim
CS Division of EECS Dept.
KAIST

LTL vs. CTL

- LTL implicitly quantifies universally over paths
  - a state of a system satisfies an LTL formula if all paths from the given state satisfy it
  - properties which use both universal and existential path quantifiers cannot in general be model checked using LTL.
    - property ϕ which use only universal path quantifiers can be checked using LTL by checking $\neg\phi$
- Branching-time logic solve this limitation by quantifying paths explicitly
  - There is a reachable state satisfying q: EF q
    - Note that we can check this property by checking LTL formula ϕ=G $\neg$q
      - If ϕ is true, the property is false. If ϕ is false, the property is true
  - From all reachable states satisfying p, it is possible to maintain p continuously until reaching a state satisfying q: AG (p $\rightarrow$ E (p U q))
  - Whenever a state satisfying p is reached, the system can exhibit q continuously forevermore: AG (p $\rightarrow$ EG q)
  - There is a reachable state from which all reachable states satisfy p: EF AG p

Syntax of Computation Tree Logic (CTL)

- Def 3.12 ϕ = $\perp$ | $\top$ | p | $\neg\, \phi$ | $\phi \wedge \phi$ | $\phi \vee \phi$ | $\phi \rightarrow \phi$ |AX ϕ | EX ϕ | AF ϕ | EF ϕ | AG ϕ | EG ϕ | A (ϕ U ϕ) | E (ϕ U ϕ)
  - A: along all paths
  - E: along at least one path
- Precedence
  - AG, EG, AF, EF,AX, EX, $\wedge$, $\vee$, $\rightarrow$, AU, EU
- Note that the following formulas are not well-formed CTL formulas
  - EF G r
  - A $\neg$G $\neg$ p
  - F (r U q)
  - EF (r U q)
  - AEF r
  - A ((r U q) $\wedge$ (p U r))



A [(AX ¬p) U (E [(EX p∧q) U ¬p)]]

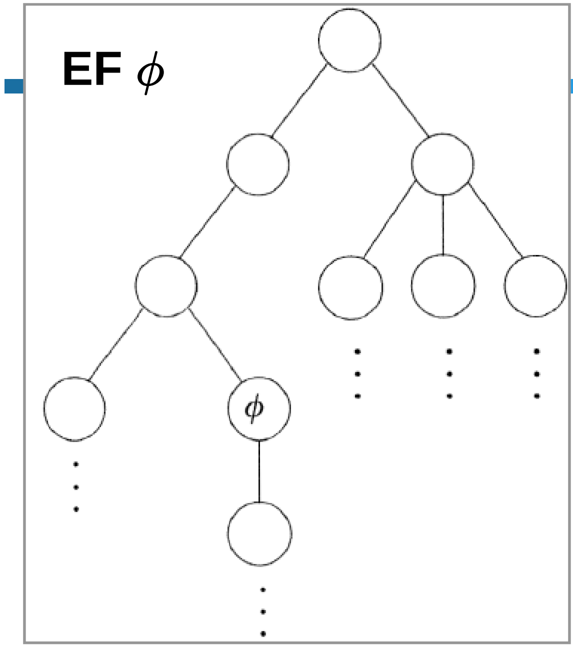
Semantics of CTL (1/2)

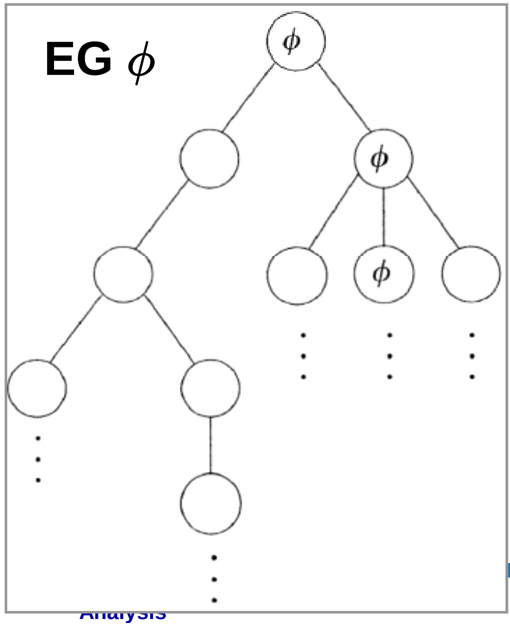
- Def 3.15 Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, s in S, ϕ a CTL formula. The relation $\mathcal{M}, s \models \phi$ is defined by structural induction on ϕ. We omit $\mathcal{M}$ if context is clear.
  - $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \not\models \bot$
  - $\mathcal{M}, s \models p$ iff $p \in L(s)$
  - $\mathcal{M}, s \models \neg \phi$ iff $\mathcal{M}, s \not\models \phi$
  - $\mathcal{M}, s \models \phi_1 \wedge \phi_2$ iff $\mathcal{M}, s \models \phi_1$ and $\mathcal{M}, s \models \phi_2$
  - $\mathcal{M}, s \models \phi_1 \vee \phi_2$ iff $\mathcal{M}, s \models \phi_1$ or $\mathcal{M}, s \models \phi_2$
  - $\mathcal{M}, s \models \phi_1 \rightarrow \phi_2$ iff $\mathcal{M}, s \not\models \phi_1$ or $\mathcal{M}, s \models \phi_2$
  - $\mathcal{M}, s \models AX\ \phi$ iff for all s_1 s.t. $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus AX says "in every next state"
  - $\mathcal{M}, s \models EX\ \phi$ iff for some s_1 s.t. $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus EX says "in some next state"
  - $\mathcal{M}, s \models AX\ \phi$ iff for all s_1 s.t. $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus AX says "in every next state"
  - $\mathcal{M}, s \models EX\ \phi$ iff for some s_1 s.t. $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus EX says "in some next state"

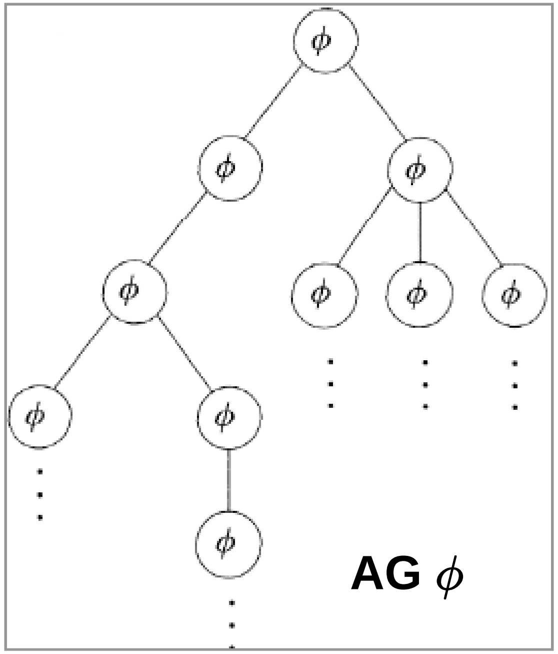
Semantics of CTL (2/2)

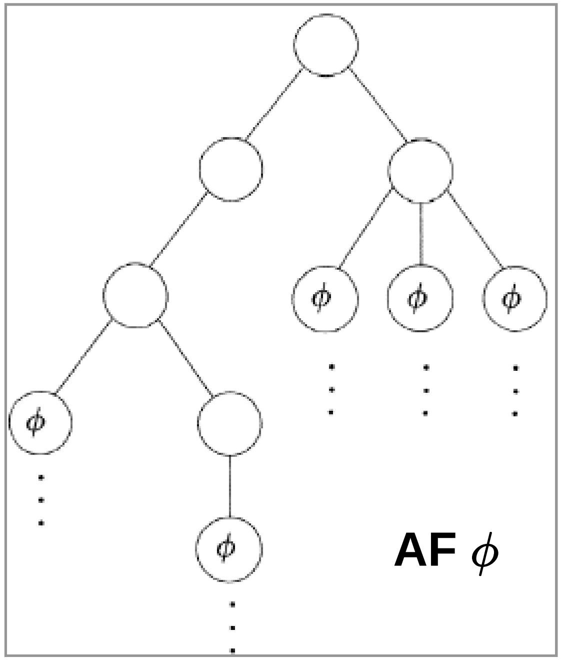
- Def 3.15 Let $\mathcal{M}$ = (S, $\rightarrow$, L) be a model for CTL, s in S, ϕ a CTL formula. The relation $\mathcal{M}$,s $\vDash \phi$ is defined by structural induction on ϕ. We omit $\mathcal{M}$ if context is clear.
  - $\mathcal{M}$,s $\vDash$ AG ϕ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ 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 ϕ 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 ϕ 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 ϕ 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 [ϕ_1 U ϕ_2] iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, that path satisfies ϕ_1 U ϕ_2
  - $\mathcal{M}$,s $\vDash$ E [ϕ_1 U ϕ_2] iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$ where s_1 equals s, that path satisfies ϕ_1 U ϕ_2

Example (1/2)



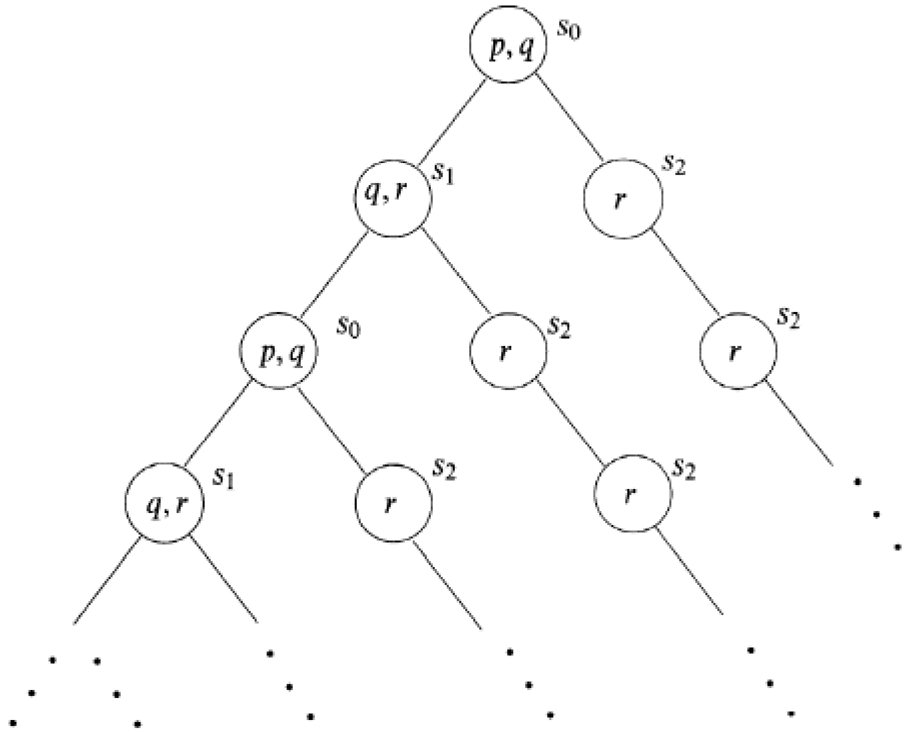


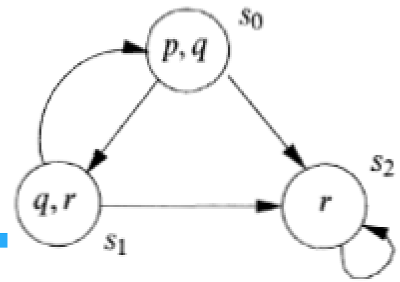




Example (2/2)

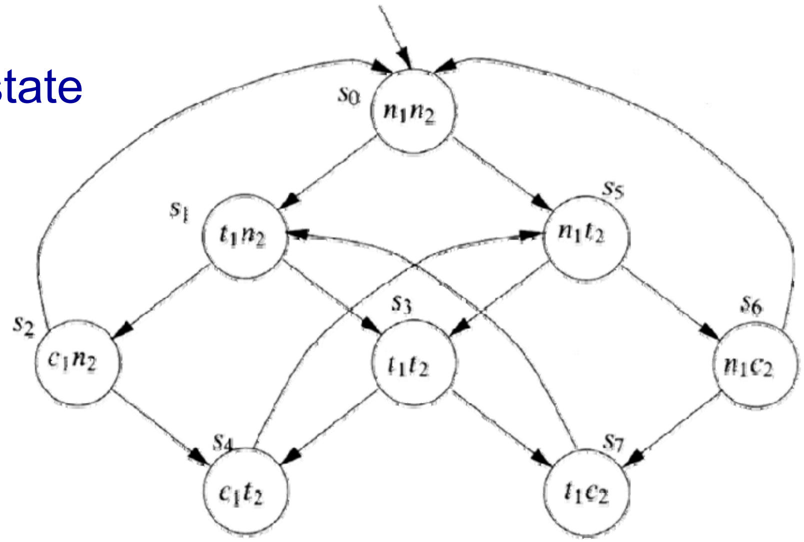
- $\mathcal{M}, s_0 \models p \wedge q$, $\mathcal{M}, s_0 \models \neg r$, $\mathcal{M}, s_0 \models$
- $\mathcal{M}, s_0 \models$ EX $(q \wedge r)$
- $\mathcal{M}, s_0 \models \neg$AX$(q \wedge r)$
- $\mathcal{M}, s_0 \models \neg$EF$(p \wedge r)$
- $\mathcal{M}, s_2 \models$ EG r
- $\mathcal{M}, s_0 \models$ AF r
- $\mathcal{M}, s_0 \models$ E [(p $\wedge$ q) U r]
- $\mathcal{M}, s_0 \models$ A [p U r]
- $\mathcal{M}, s_0 \models$ AG (p $\vee$ q $\vee$ r $\rightarrow$ EF EG r)





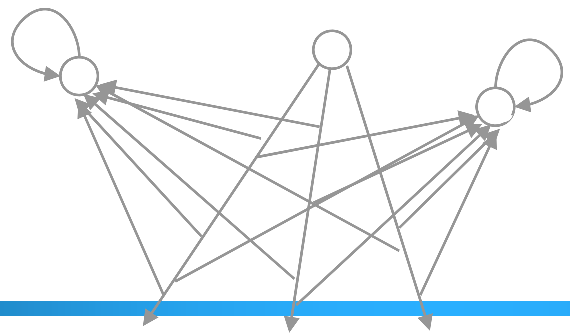
Practical patterns of specification (1/2)

- 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 ?

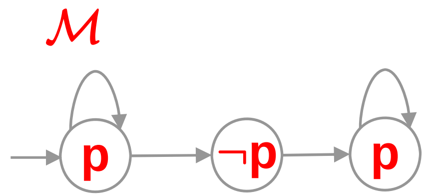


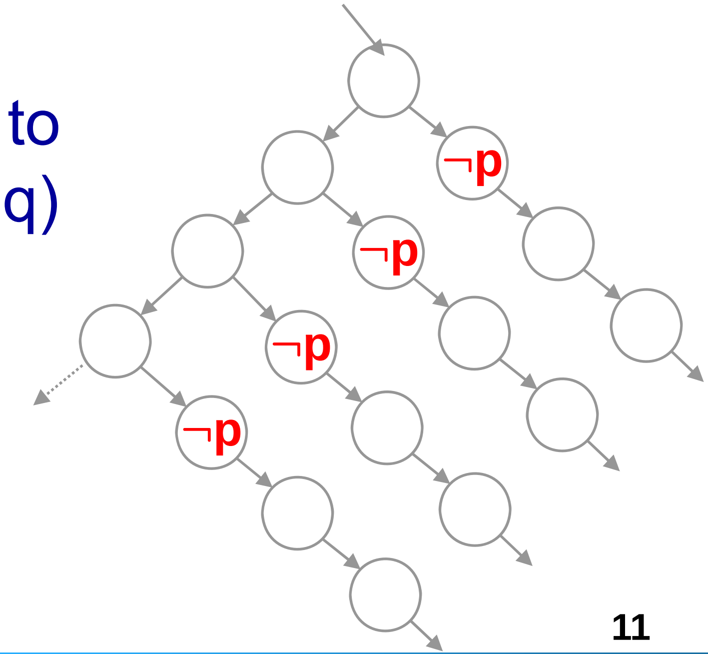
Equivalence between CTL formulas

- 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 $\neg\ \phi$
- $\neg$ AX $\phi \equiv$ EX $\neg\ \phi$
- AF $\phi \equiv$ A [T U ϕ]
- EF $\phi \equiv$ E [T U ϕ]

CTL is not more expressive than LTL

- CTL cannot select a range of paths
  - F G p in LTL is not equivalent to AF AG p
    - $\mathcal{M}, s_0 \models$ F G p but $\mathcal{M}, s_0 \not\models$ AF AG p
    - AF AG p is strictly stronger than F G p
    - AF EG p is strictly weaker than F G p
- Similarly, F p $\rightarrow$ F q is not equivalent to AF p $\rightarrow$ AF q, neither to AG (p $\rightarrow$ AF q)
- Remark
  - F X p ≡ X F p in LTL
  - AF AX p is not equivalent to AX AF p





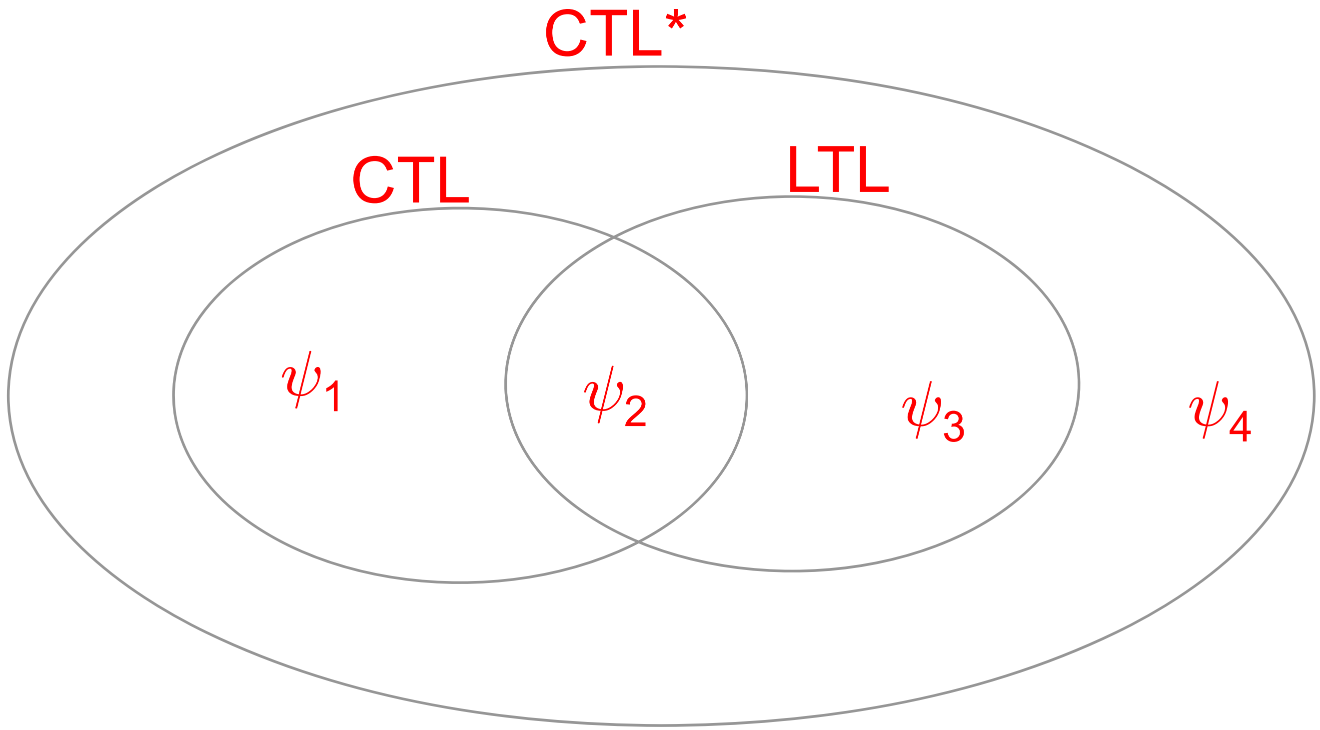
Comparison between LTL and CTL

	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, CWB-NC

CTL*

- CTL* combines the expressive powers of LTL and CTL
- Syntax of CTL*
  - State formula ϕ ::= T | p | $\neg\ \phi$ | $\phi \wedge \phi$ | A [α] | E[α]
  - Path formula α ::= ϕ | $\neg\ \alpha$ | $\alpha \wedge \alpha$ | α U α | G α | F α | X α
- LTL is a subset of CTL*
  - LTL formula α is equivalent to A[α] in CTL*
- CTL is a subset of CTL*
  - We restrict α ::= ϕ U ϕ | G ϕ | F ϕ | X ϕ
    - No boolean connectives in path formula
      - Not real limitation. See page 6
    - No nesting of the path modalities X,F, and G

Relationship between LTL,CTL, and CTL*



Complexity of Model Checking

- Let $\mathcal{M}$ be a target transition system with *N* states and *M* transitions
- Upper bound of model checking complexity
  - LTL-formula ϕ : $O((N+M)\cdot 2^{|\phi|})$
  - CTL-formula ϕ : $O((N+M)\cdot|\phi|)$
  - CTL*-formula ϕ : $O((N+M)\cdot 2^{|\phi|})$
- Lower bound of model checking complexity
  - LTL-formula ϕ : PSpace-hard -> PSpace-complete
    - Note that P $\subseteq$ NP $\subseteq$ PSpace $\subseteq$ EXP $\subseteq$ EXPSpace
  - CTL-formula ϕ : P-hard -> P-complete
  - CTL*-formula ϕ : PSpace-hard -> PSpace-complete
- For more details, "The Complexity of Temporal Logic Model Checking" by Ph. Schnoebelen
  - *Advances in Modal Logic, Volume 4, 1-44, 2002*

GCTL Formulas in CWB-NC

- tt, ff : true, false
- {act_list} is satisfied by an action a if a appears in act_list
- {- act_list} is satisfied by an action a if a is not included in act_list
- ~ p is true if p is false
- Example
  - prop can_deadlock = E F ~{- }
  - prop recv_guarantee = A G ({send} -> F{'receive})
  - prop fair_recv_guarantee =
    A ((G F {- t}) -> (G {send} -> F {'receive}))

Peterson's Mutual Exclusion Protocol

```
proc Sys = (P1|P2|Flag_1_0|Flag_2_0|Turn1|NCRIT0)
\{flag1_0,flag1_1,flag2_0,flag2_1,turn1,turn2,test_flag1_0,test_flag1_1,t
      est_flag2_0,test_flag2_1,test_turn1,test_turn2,inc_cnt,dec_cnt}
proc P1   = flag1_1.turn1.P1'
proc P1'  = test_flag2_0.P1'' + test_turn2.P1'' + test_flag2_1.P1' +
      test_turn1.P1'
proc P1'' = inc_cnt.dec_cnt.flag1_0.P1
proc P2   = flag2_1.turn2.P2'
proc P2'  = test_flag1_0.P2'' + test_turn1.P2'' + test_flag1_1.P2' +
      test_turn2.P2'
proc P2'' = inc_cnt.dec_cnt.flag2_0.P2
proc UpdateF1 = 'flag1_0.Flag_1_0 + 'flag1_1.Flag_1_1
proc Flag_1_0 = 'test_flag1_0.Flag_1_0 + UpdateF1
proc Flag_1_1 = 'test_flag1_1.Flag_1_1 + UpdateF1
proc UpdateF2 = 'flag2_0.Flag_2_0 + 'flag2_1.Flag_2_1
proc Flag_2_0 = 'test_flag2_0.Flag_2_0 + UpdateF2
proc Flag_2_1 = 'test_flag2_1.Flag_2_1 + UpdateF2
proc UpdateT = 'turn1.Turn1 + 'turn2.Turn2
proc Turn1   = 'test_turn1.Turn1 + UpdateT
proc Turn2   = 'test_turn2.Turn2 + UpdateT
proc NCRIT0  = 'inc_cnt.cnt_1.NCRIT1
proc NCRIT1  = 'inc_cnt.cnt_2.NCRIT2 + 'dec_cnt.cnt_0.NCRIT0
proc NCRIT2  = 'dec_cnt.cnt_1.NCRIT1
```

*** Verification through equivalence**
*** obseq, trace inclusion**

proc Spec = cnt_1.cnt_0.Spec

*** Verification through model checking**

prop ab1 =
A G ({cnt_1} -> X ({t} W {cnt_0}))

prop ab2 =
A G ({cnt_0} -> X ({t} W {cnt_1}))

prop ab3 =
A G ~{cnt_2}

prop REQ = ab1 /\ ab2 /\ ab3