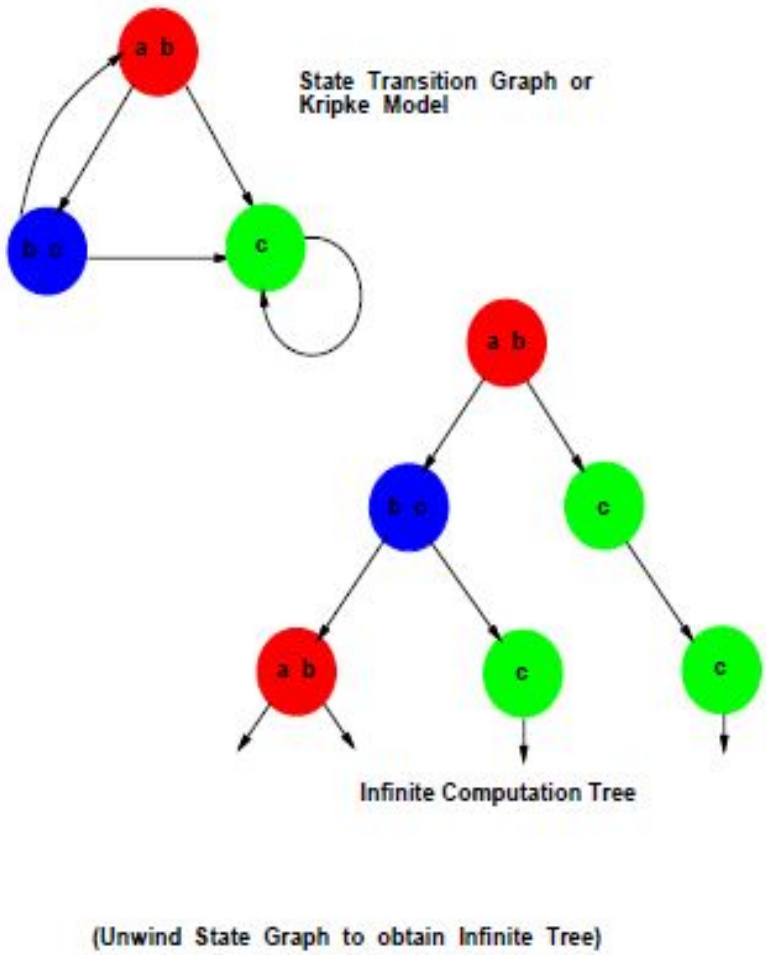
Computational Tree Logic (CTL)
Branching Time Semantics

1. A tree-like structure in which the future is not determined.
2. Computation tree of an automaton is the unfolding of the automaton.
3. CTL can specify:
   - When some initial condition is satisfied.
   - Then all possible executions of a program avoid some undesirable condition.
4. MC explores all possible transitions out of program states satisfying the initial condition and ensures all such executions satisfy the property.



Syntax of Computational Tree Logic (CTL)

CTL (state) formulas:

$$\Phi \ ::= \ \textit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

CTL path formulas:

$$\varphi \ ::= \ \bigcirc\Phi \mid \Phi_1 \,\mathbf{U}\, \Phi_2 \mid \Diamond\Phi \mid \Box\Phi$$

$\bigcirc \ \widehat{=}$ next $\qquad \Diamond \ \widehat{=}$ eventually

$\mathbf{U} \ \widehat{=}$ until $\qquad \Box \ \widehat{=}$ always

A path quantifier can prefix an assertion composed of arbitrary combinations of the usual liner-time operator

1. Path quantifier (state formulas) : A – “for every path”, E – “there exists a path”
2. Each of Linear Time operator (path formulas) must be immediately preceded by a path quantifier: e.g., AG EF p
3. Linear-time operators:
   - X p (O p) – p holds next time
   - F p (<> p)– p holds sometime in the future
   - G p ([] p) – p holds globally (always) in the future
   - P U q – p holds until q holds

Examples for CTL formulas

CTL (state) formulas:

$$\Phi \;::=\; \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

CTL path formulas:

$$\varphi \;::=\; \bigcirc\Phi \mid \Phi_1 \,\mathbf{U}\, \Phi_2 \mid \Diamond\Phi \mid \Box\Phi$$

mutual exclusion (safety) $\forall\Box(\neg \mathit{crit}_1 \vee \neg \mathit{crit}_2)$

"every request will be answered eventually"

$$\forall\Box(\mathit{request} \rightarrow \forall\Diamond \mathit{response})$$

traffic lights $\forall\Box(\mathit{yellow} \rightarrow \forall\bigcirc \mathit{red})$

reset possiblity $\forall\Box\exists\Diamond \mathit{start}$

Semantics of CTL

define a satisfaction relation $\models$ for CTL formulas over AP and a given TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$ without terminal states

- interpretation of state formulas over the states
- interpretation of path formulas over the paths (infinite path fragments)

Satisfaction relation for state formulas

$s \models \mathit{true}$

$s \models a$ iff $a \in L(s)$

$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$

$s \models \neg\Phi$ iff $s \not\models \Phi$

$s \models \exists\varphi$ iff there is a path $\pi \in \mathit{Paths}(s)$ s.t. $\pi \models \varphi$

$s \models \forall\varphi$ iff for each path $\pi \in \mathit{Paths}(s)$: $\pi \models \varphi$

satisfaction set for state formula Φ:

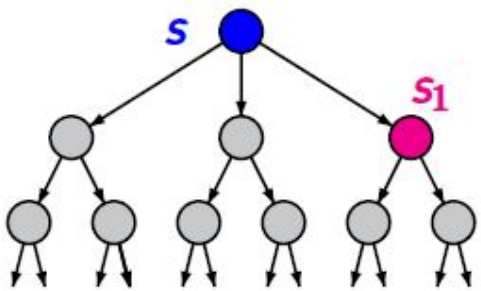
$$\mathit{Sat}(\Phi) \stackrel{\text{def}}{=} \{s \in S : s \models \Phi\}$$

Semantics of the next operator

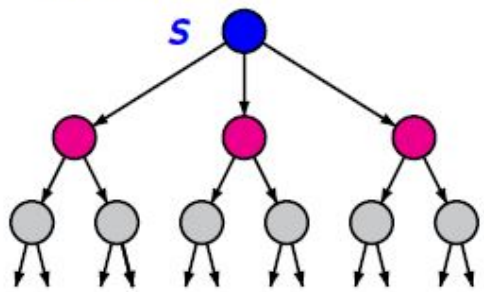
$s \models \exists \bigcirc \Phi$ iff there exists $\pi = s\, s_1\, s_2 \ldots \in \mathit{Paths}(s)$
s.t. $\pi \models \bigcirc \Phi$, i.e., $s_1 \models \Phi$

$s \models \forall \bigcirc \Phi$ iff for all $\pi = s\, s_1\, s_2 \ldots \in \mathit{Paths}(s)$:
$\pi \models \bigcirc \Phi$, i.e., $s_1 \models \Phi$

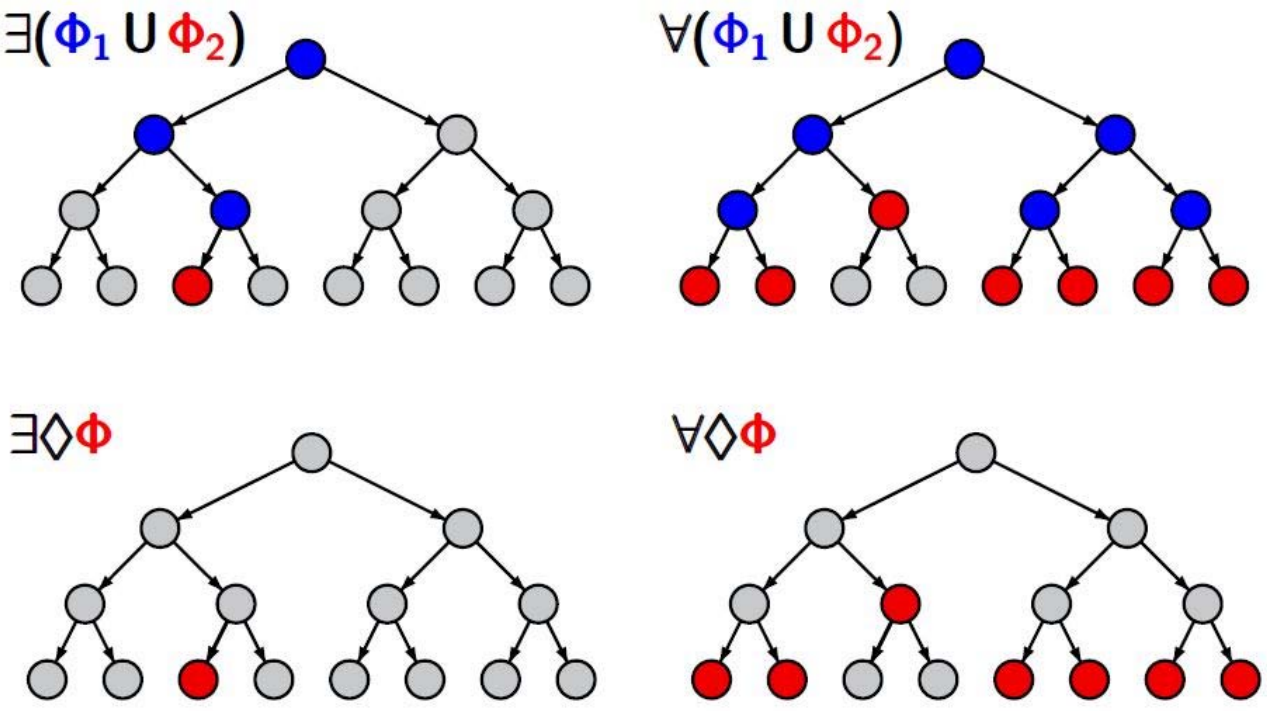
$\exists \bigcirc \Phi$



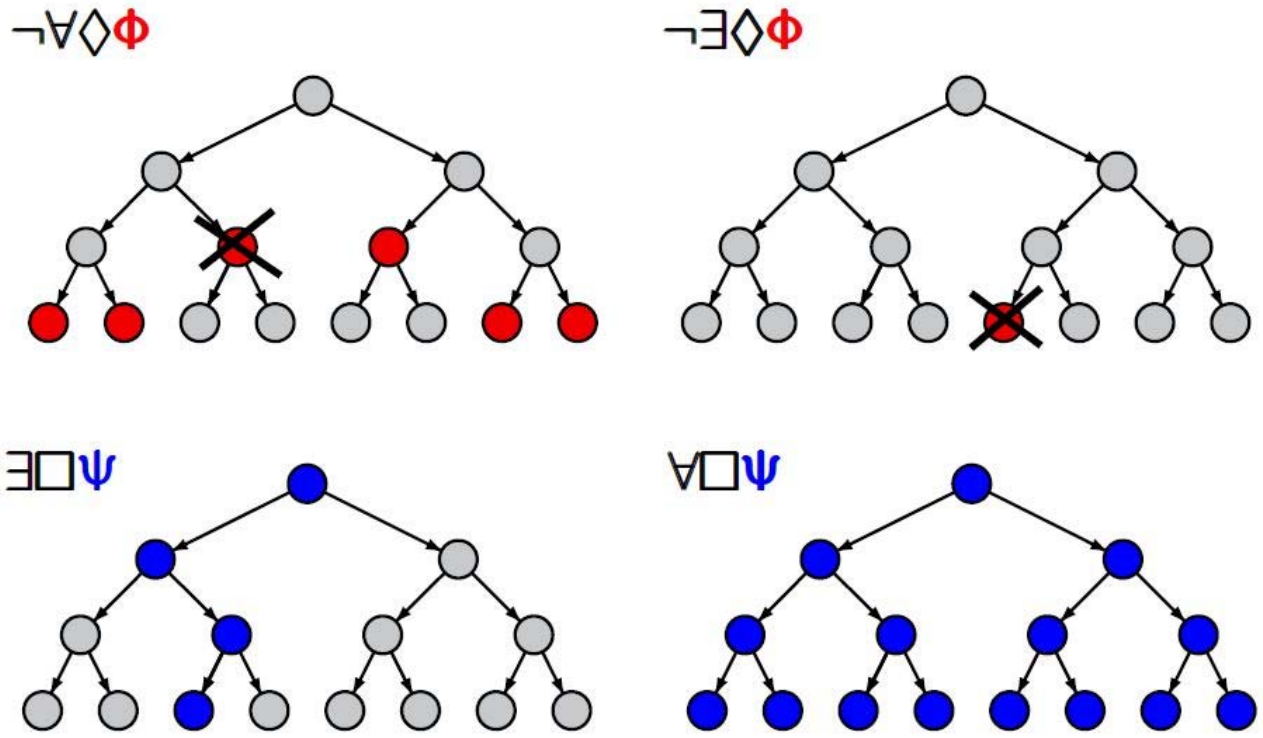
$\forall \bigcirc \Phi$



Semantics of until and eventually



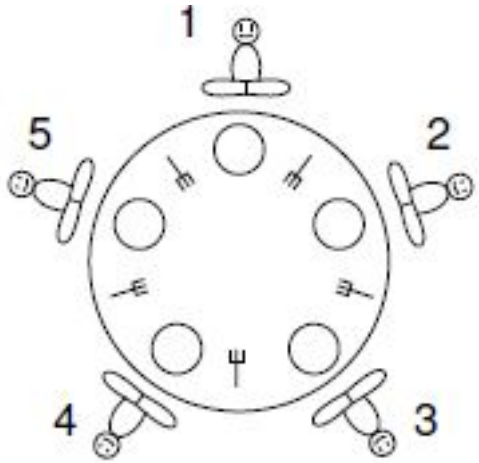
Semantics of eventually and always



Typical CTL formulas

- $\mathbf{EF}(Started \wedge \neg Ready)$: it is possible to get to a state where *Started* holds but *Ready* does not hold.
- $\mathbf{AG}(Req \Rightarrow \mathbf{AF}\, Ack)$: if a *Request* occurs, then it will be eventually *Acknowledged*.
- $\mathbf{AG}(\mathbf{AF}\ DeviceEnabled)$: *DeviceEnabled* holds infinitely often on every computation path.
- $\mathbf{AG}(\mathbf{EF}\ Restart)$: from any state it is possible to get to the *Restart* state.

Example: dining philosopher



Five philosophers are sitting around a table, taking turns at thinking and eating.

We shall express a couple of properties in CTL. Let us assume the following atomic propositions:

$e_i \mathrel{\widehat{=}}$ philosopher i is currently eating

$f_i \mathrel{\widehat{=}}$ philosopher i has just finished eating

Example: dining philosopher

"Philosophers 1 and 4 will never eat at the same time."

$$\mathbf{AG}\,\neg(e_1 \wedge e_4)$$

"Philosopher 2 will be the first to eat."

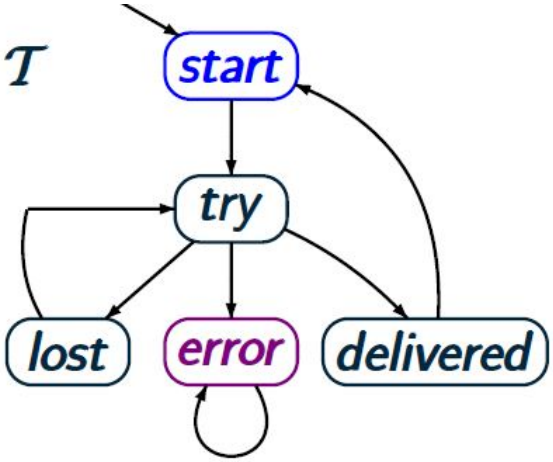
$$\neg(e_1 \vee e_3 \vee e_4 \vee e_5)\ \mathbf{AU}\ e_2$$

Specifying "infinitely often" in CTL

If s is a state in a TS and $a \in AP$ then:

$$
\begin{array}{ll}
 & s \models_{\mathrm{CTL}} \forall\Box\forall\Diamond a \\
\text{iff} & \text{for all paths } \pi = s_0\, s_1\, s_2 \ldots \in \mathit{Paths}(s): \\
 & \qquad \overset{\infty}{\exists}\, i \geq 0. \quad \text{s.t.} \quad s_i \models a \\
\text{iff} & s \models_{\mathrm{LTL}} \Box\Diamond a
\end{array}
$$

Example: CTL semantics



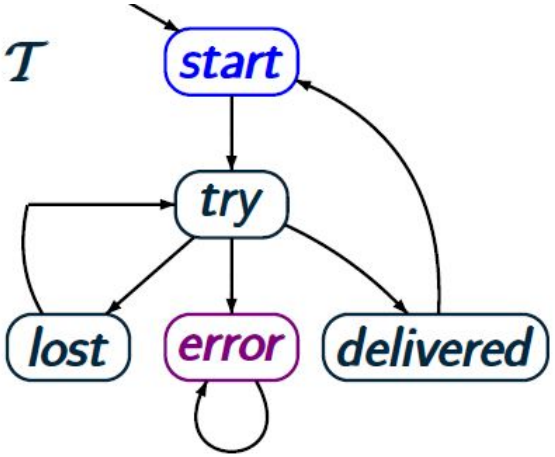
$\mathcal{T} \models \exists\Diamond\forall\Box\neg start$?

$$\Phi_1 = \exists\Diamond \boxed{\forall\Box\neg start} \rightsquigarrow \exists\Diamond \boxed{error}$$

$$Sat(\forall\Box\neg start) = \{error\}$$

$$Sat(\exists\Diamond\forall\Box\neg start) = ?$$

Example: CTL semantics



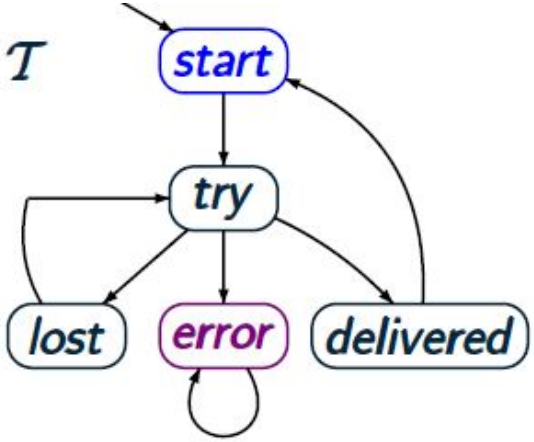
$\mathcal{T} \models \exists\Diamond\forall\Box\neg start \quad \surd$

$$\Phi_1 = \exists\Diamond \boxed{\forall\Box\neg start} \rightsquigarrow \exists\Diamond \boxed{error}$$

$$Sat(\forall\Box\neg start) = \{error\}$$

$$Sat(\exists\Diamond\forall\Box\neg start) = Sat(\exists\Diamond error) = \text{“all states”}$$

Example: CTL semantics



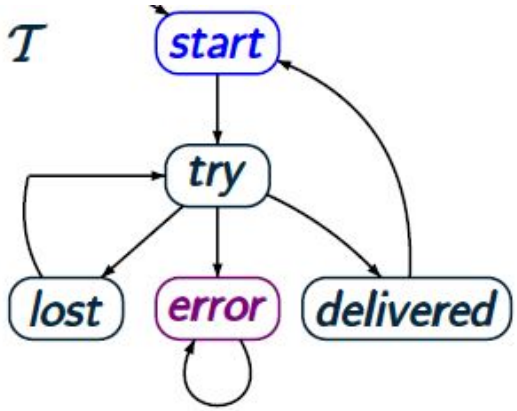
$\mathcal{T} \models \exists\Diamond\forall\Box\neg start$

$\mathcal{T} \models \forall\bigcirc\exists\bigcirc\forall\Box\neg start$?

$\Phi_2 = \forall\bigcirc \boxed{\exists\bigcirc\forall\Box\neg start} \rightsquigarrow \forall\bigcirc \boxed{\exists\bigcirc error}$

$$Sat(\forall\Box\neg start) = \{error\}$$
$$Sat(\exists\bigcirc\forall\Box\neg start) = \{error, try\}$$

Example: CTL semantics



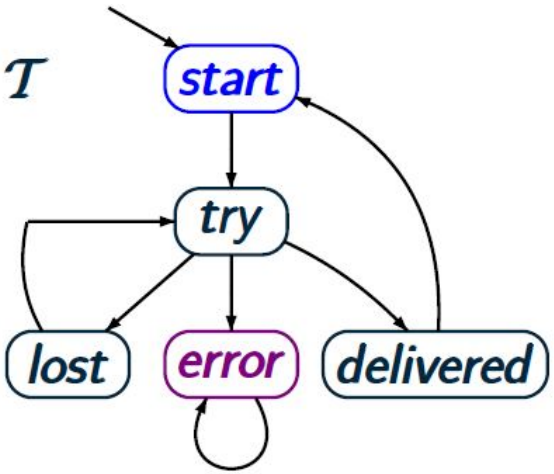
$\mathcal{T} \models \exists\Diamond\forall\Box\neg start$

$\mathcal{T} \models \forall\bigcirc\exists\bigcirc\forall\Box\neg start$?

$\Phi_2 = \forall\bigcirc\exists\bigcirc\forall\Box\neg start \quad \rightsquigarrow \quad \boxed{\forall\bigcirc(error \vee try)}$

$$
\begin{aligned}
Sat(\forall\Box\neg start) &= \{error\} \\
Sat(\exists\bigcirc\forall\Box\neg start) &= \{error, try\} \\
Sat(\forall\bigcirc\exists\bigcirc\forall\Box\neg start) &= \ ?
\end{aligned}
$$

Example: CTL semantics



$$\mathcal{T} \models \exists\Diamond\forall\Box\neg start$$

$$\mathcal{T} \models \forall\bigcirc\exists\bigcirc\forall\Box\neg start \quad \surd$$

$$\Phi_2 = \forall\bigcirc\exists\bigcirc\forall\Box\neg start \quad \rightsquigarrow \quad \boxed{\forall\bigcirc(error \vee try)}$$

$$\begin{aligned} Sat(\forall\Box\neg start) &= \{error\} \\ Sat(\exists\bigcirc\forall\Box\neg start) &= \{error, try\} \\ Sat(\forall\bigcirc\exists\bigcirc\forall\Box\neg start) &= \{error, lost, start\} \end{aligned}$$

More examples

CTL formula Φ	LTL formula φ
a	a
$\forall\bigcirc a$	$\bigcirc a$
$\forall(a \mathbf{U} b)$	$a \mathbf{U} b$
$\forall\Box a$	$\Box a$
$\forall\Diamond a$	$\Diamond a$

$\forall\Box\forall\Diamond a$	$\Box\Diamond a$	but: $\forall\Diamond\forall\Box a \not\equiv \Diamond\Box a$

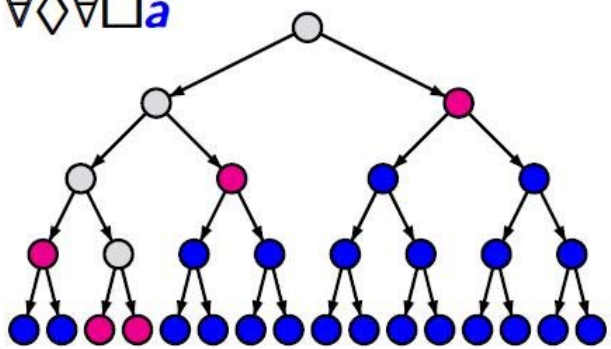
infinitely often a

The CTL formula AFAG a

$s \models \forall\Diamond\forall\Box a$ iff on each path π from s
there is a state t with $t \models \forall\Box a$

i.e., all states in the computation tree of t fulfill a

$\forall\Diamond\forall\Box a$

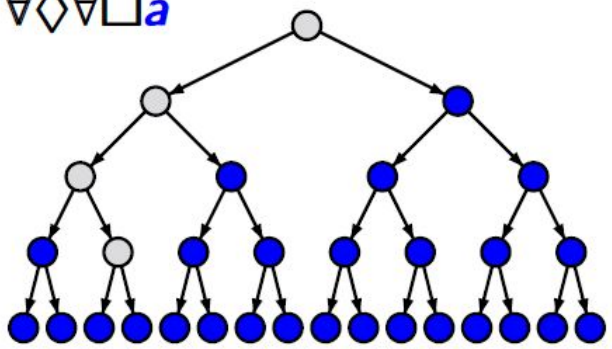


The CTL formula A<>A[]a

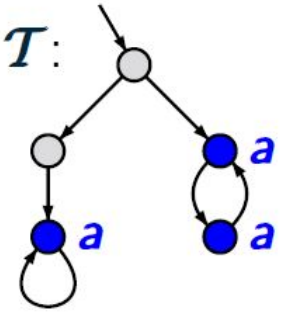
$s \models \forall\Diamond\forall\Box a$ iff on each path π from s
there is a state t with $t \models \forall\Box a$

i.e., all states in the computation tree of t fulfill a

$\forall\Diamond\forall\Box a$



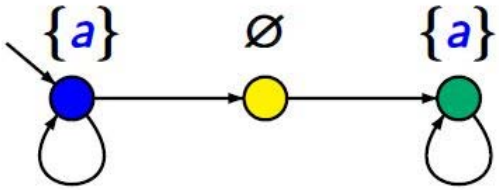
$\mathcal{T}$:



$\mathcal{T} \models \forall\Diamond\forall\Box a$

<>[]a is not equivalent to A<>A[]a

transition system $\mathcal{T}$

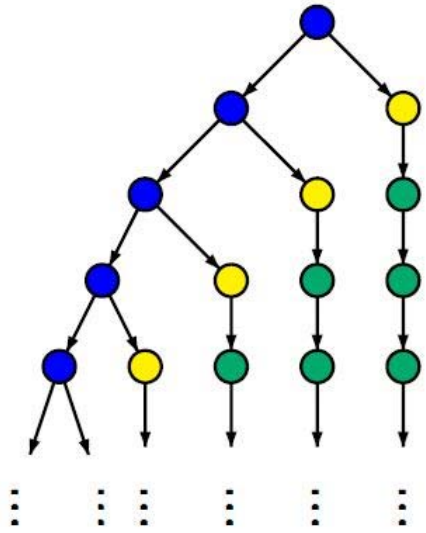


$\mathcal{T} \models_{\mathrm{LTL}} \Diamond\Box a$

$\mathcal{T} \not\models_{\mathrm{CTL}} \forall\Diamond\forall\Box a$

$Sat(\forall\Box a) = \{\bullet\}$

computation tree



CTL Model Checking (MC)

given: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
CTL formula Φ over AP

question: does $\mathcal{T} \models \Phi$ hold ?

idea:

- compute $Sat(\Phi) = \{s \in S : s \models \Phi\}$
- check whether $S_0 \subseteq Sat(\Phi)$

given: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
CTL formula Φ over AP

question: does $\mathcal{T} \models \Phi$ hold ?

inner subformulas first

```
FOR ALL subformulas Ψ of Φ DO
    compute Sat(Ψ)


OD
```

given: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
CTL formula Φ over AP

question: does $\mathcal{T} \models \Phi$ hold ?

inner subformulas first

```
FOR ALL subformulas Ψ of Φ DO
    compute Sat(Ψ)
    replace Ψ by a new atomic proposition a_Ψ
    FOR ALL s ∈ Sat(Ψ) DO add a_Ψ to L(s) OD
OD
```

CTL MC

given: finite TS $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$
CTL formula Φ over AP

question: does $\mathcal{T} \models \Phi$ hold ?

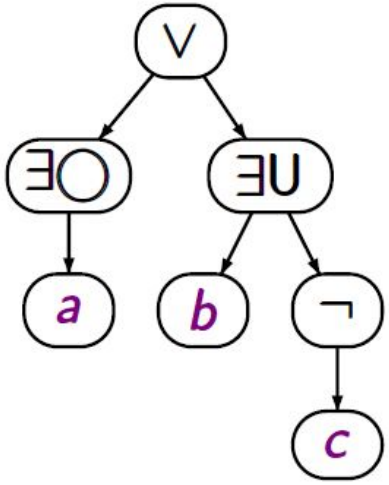
inner subformulas first

```
FOR ALL subformulas Ψ of Φ DO
    compute Sat(Ψ)
    replace Ψ by a new atomic proposition a_Ψ
    FOR ALL s ∈ Sat(Ψ) DO add a_Ψ to L(s) OD
OD
IF S0 ⊆ Sat(Φ) THEN output "yes"
               ELSE output "no"
FI
```

Example: CTL MC

$$\Phi = \quad \exists\bigcirc a \;\vee\; \exists(b \,\mathsf{U}\, \neg c)$$

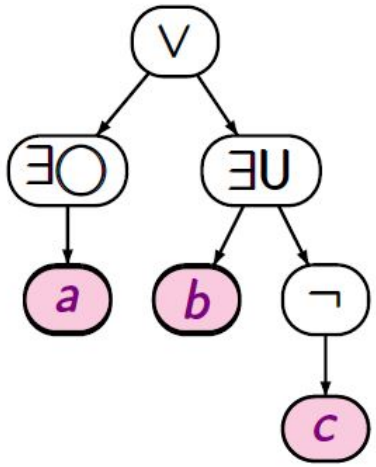
syntax tree for Φ



Example: CTL MC

$$\Phi = \quad \exists\bigcirc a \;\vee\; \exists(b \,\mathsf{U}\, \neg c)$$

syntax tree for Φ



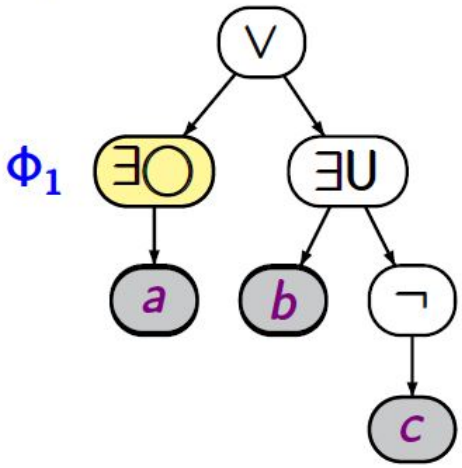
processed in
bottom-up fashion

compute $Sat(a), Sat(b), Sat(c)$

Example: CTL MC

$$\Phi = \underbrace{\exists\bigcirc a}_{\Phi_1} \vee \exists(b \,\mathsf{U}\, \neg c)$$

syntax tree for Φ



processed in
bottom-up fashion

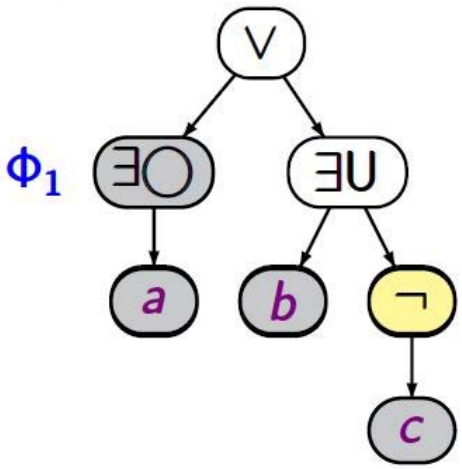
compute $Sat(a), Sat(b), Sat(c)$

$Sat(\Phi_1) = \ldots$

Example: CTL MC

$$\Phi = \underbrace{\exists\bigcirc a}_{\Phi_1} \vee \exists(b \,\mathsf{U}\, \neg c)$$

syntax tree for Φ



processed in
bottom-up fashion

compute $Sat(a), Sat(b), Sat(c)$

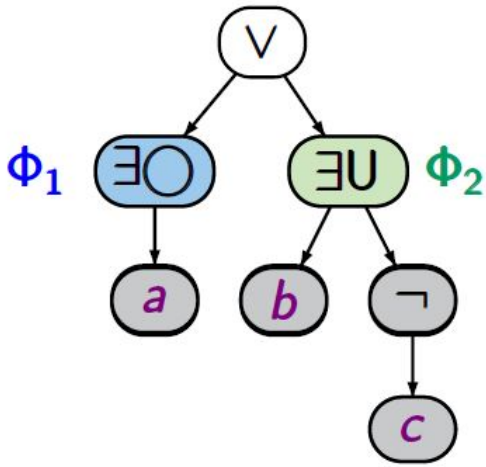
$Sat(\Phi_1) = \ldots$

$Sat(\neg c) = S \setminus Sat(c)$

Example: CTL MC

$$\Phi = \underbrace{\exists\bigcirc a}_{\Phi_1} \vee \underbrace{\exists(b \,\mathsf{U}\, \neg c)}_{\Phi_2}$$

syntax tree for Φ



processed in
bottom-up fashion

compute $Sat(a), Sat(b), Sat(c)$

$Sat(\Phi_1) = \ldots$

$Sat(\neg c) = S \setminus Sat(c)$

$Sat(\Phi_2) = \ldots$

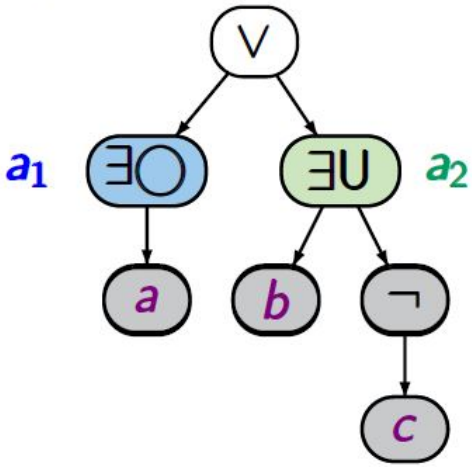
replace Φ_1 with a_1

replace Φ_2 with a_2

Example: CTL MC

$$\Phi = \underbrace{\exists\bigcirc a}_{\Phi_1} \vee \underbrace{\exists(b \,\mathsf{U}\, \neg c)}_{\Phi_2} \quad \rightsquigarrow \quad a_1 \vee a_2$$

syntax tree for Φ



processed in
bottom-up fashion

compute $Sat(a), Sat(b), Sat(c)$

$Sat(\Phi_1) = \ldots = Sat(a_1)$

$Sat(\neg c) = S \setminus Sat(c)$

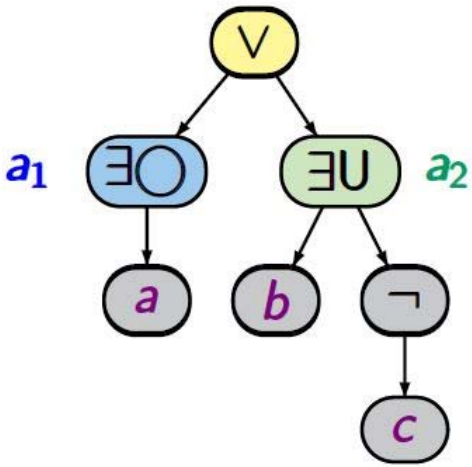
$Sat(\Phi_2) = \ldots = Sat(a_2)$

replace Φ_1 with a_1
replace Φ_2 with a_2

Example: CTL MC

$$\Phi = \underbrace{\exists\bigcirc a}_{\Phi_1} \vee \underbrace{\exists(b \,\mathsf{U}\, \neg c)}_{\Phi_2} \quad \rightsquigarrow \quad a_1 \vee a_2$$

syntax tree for Φ



processed in bottom-up fashion

compute $Sat(a), Sat(b), Sat(c)$

$Sat(\Phi_1) = \ldots = Sat(a_1)$

$Sat(\neg c) = S \setminus Sat(c)$

$Sat(\Phi_2) = \ldots = Sat(a_2)$

replace Φ_1 with a_1

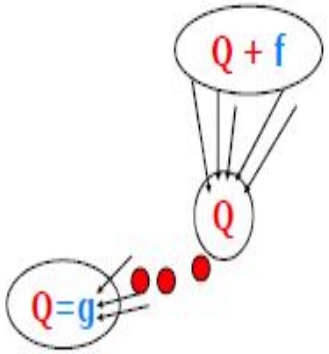
replace Φ_2 with a_2

$Sat(\Phi) = Sat(a_1) \cup Sat(a_2)$

Model Checking Algorithm for E(f U g)
Let f =φ and g=ψ

- Mark all nodes where f is true and all nodes where g is true

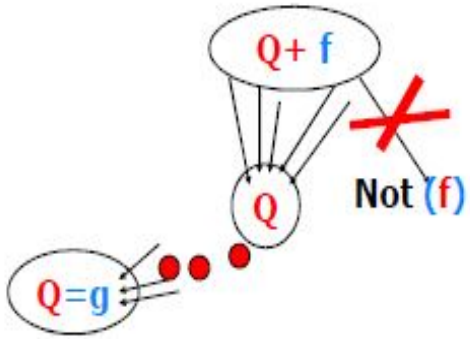
- Start from all nodes where g is true and
- Perform backwards reachability analysis
- Each step backwards, store all nodes in Q where f is true
- Repeat the above step, until it converges
- Q contains all nodes satisfying E(f U g)



function Sat_{EU} $(\phi, \psi : Formula)$: **set of** $State$;
(* precondition: true *)
begin var Q, Q' : **set of** $State$;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{ s \mid \exists s' \in Q. (s, s') \in R \} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{EU}(\phi, \psi) = \{ s \in S \mid \mathcal{M}, s \models \mathsf{E}[\phi \,\mathsf{U}\, \psi] \}$ *)
end

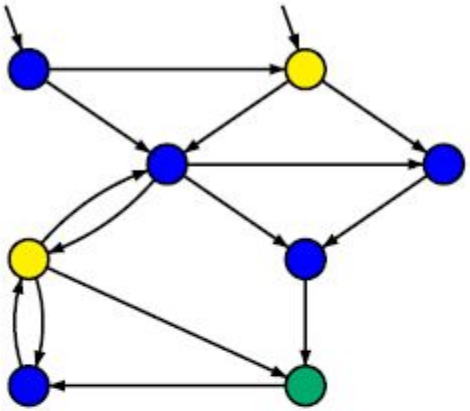
Model Checking algorithm for A(f U g)
Let f=φ and g=ψ

- Similar to the case for E(f U g)
- But each step backwards, store all nodes in Q where (f or g) is true, and the stored nodes do not lead to a node where (f or g) is false
- Repeat the above step, until it converges
- Q contains all nodes satisfying A(f U g)



function Sat_{AU} $(\phi, \psi : Formula)$: **set of** $State$;
(* precondition: true *)
begin var Q, Q' : **set of** $State$;
$Q, Q' := Sat(\psi), \varnothing$;
do $Q \neq Q' \longrightarrow$
$Q' := Q$;
$Q := Q \cup (\{s \mid \forall s'.(s, s') \in R \Rightarrow s' \in Q\} \cap Sat(\phi))$
od;
return Q
(* postcondition: $Sat_{AU}(\phi, \psi) = \{ s \in S \mid \mathcal{M}, s \models \mathsf{A}[\phi \,\mathsf{U}\, \psi] \}$ *)
end

Example: CTL until operator



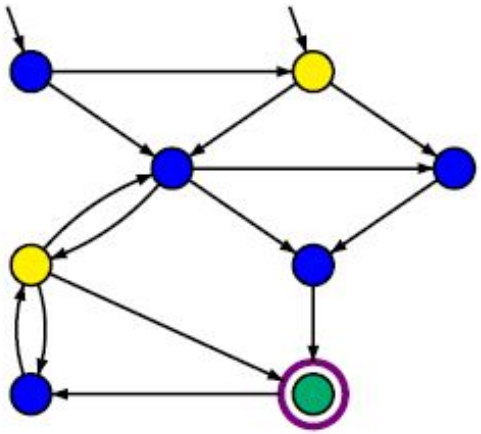
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

computation of $Sat(\exists(a \cup b))$

add all states $s \in Sat(b)$ to T
as long as there are unprocessed states in T:

- choose such a state $s \in T$
- add all states $s' \in Pre(s) \cap Sat(a)$ to T

Example: CTL until operator



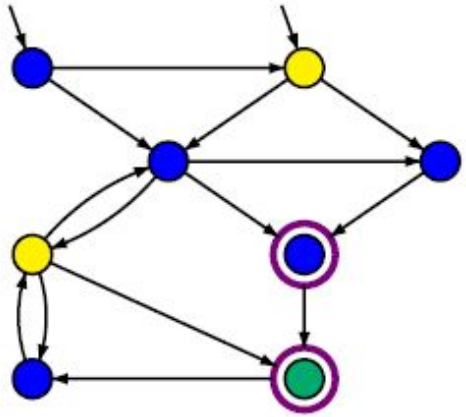
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

computation of $Sat(\exists(a \cup b))$

add all states $s \in Sat(b)$ to T
as long as there are unprocessed states in T:

- choose such a state $s \in T$
- add all states $s' \in Pre(s) \cap Sat(a)$ to T

Example: CTL until operator



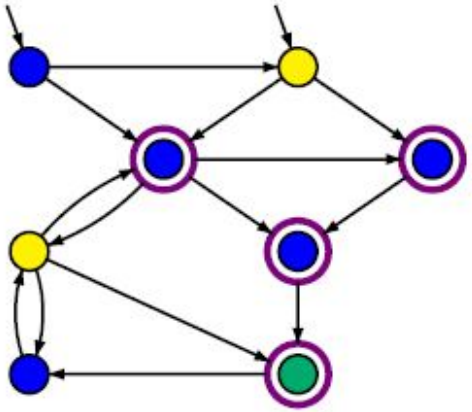
● = $\{a\}$
● = $\{b\}$
○ = $\varnothing$

computation of $Sat(\exists(a \cup b))$

add all states $s \in Sat(b)$ to T
as long as there are unprocessed states in T:

- choose such a state $s \in T$
- add all states $s' \in Pre(s) \cap Sat(a)$ to T

Example: CTL until operator



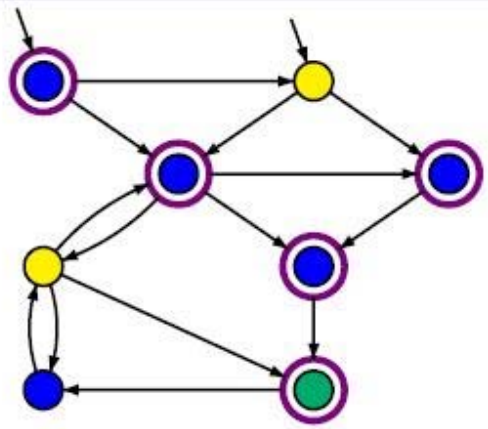
● = $\{a\}$
● = $\{b\}$
○ = $\varnothing$

computation of $Sat(\exists(a \mathsf{U} b))$

add all states $s \in Sat(b)$ to T
as long as there are unprocessed states in T:

- choose such a state $s \in T$
- add all states $s' \in Pre(s) \cap Sat(a)$ to T

Example: CTL until operator



● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

computation of $Sat(\exists(a \mathsf{U} b)) = T$

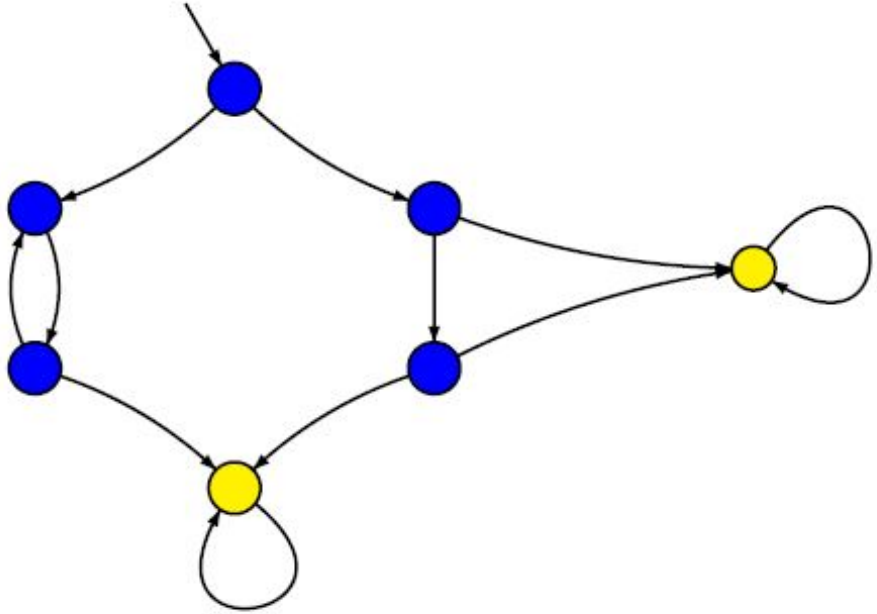
add all states $s \in Sat(b)$ to T

as long as there are unprocessed states in T:

- choose such a state $s \in T$
- add all states $s' \in Pre(s) \cap Sat(a)$ to T

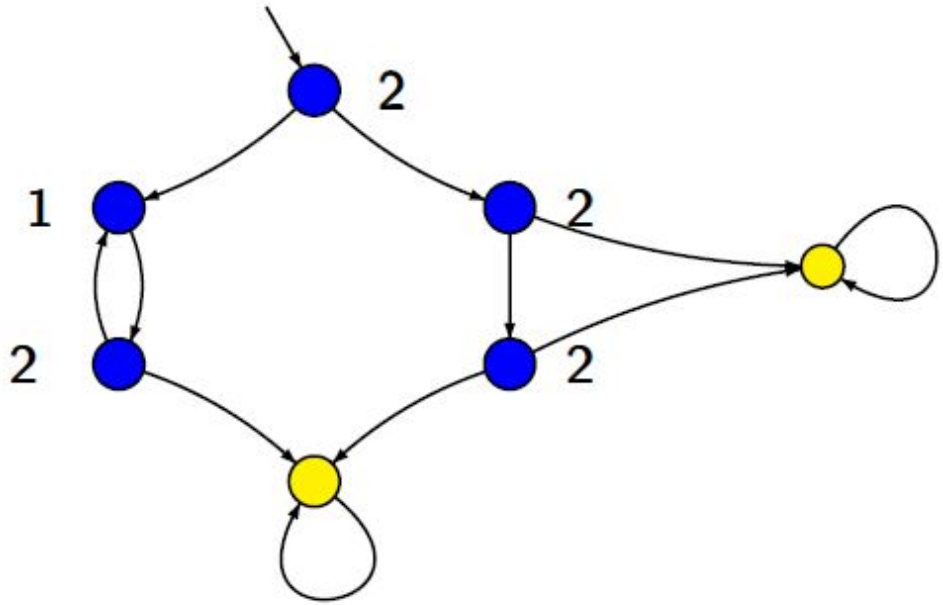
Example: CTL Model Checking for EG

computation of $T = Sat(\exists\Box blue)$



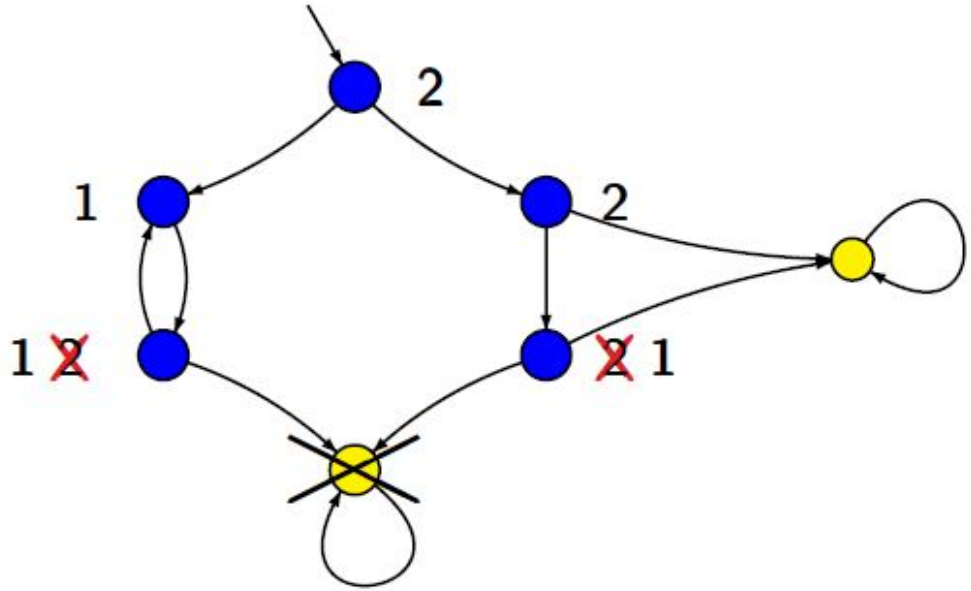
Example: CTL MC for EG

computation of $T = Sat(\exists\Box blue)$



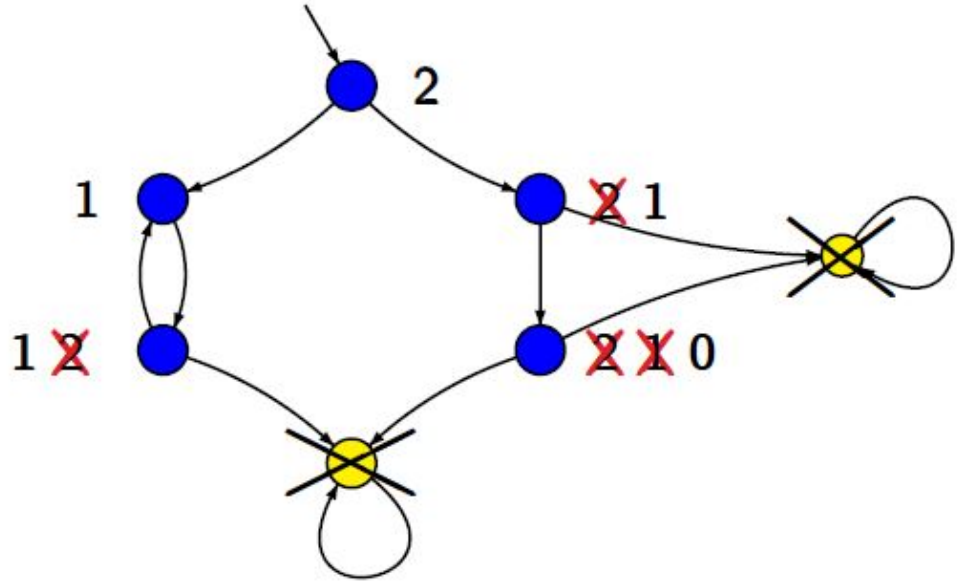
Example: CTL MC for EG

computation of $T = Sat(\exists\Box blue)$



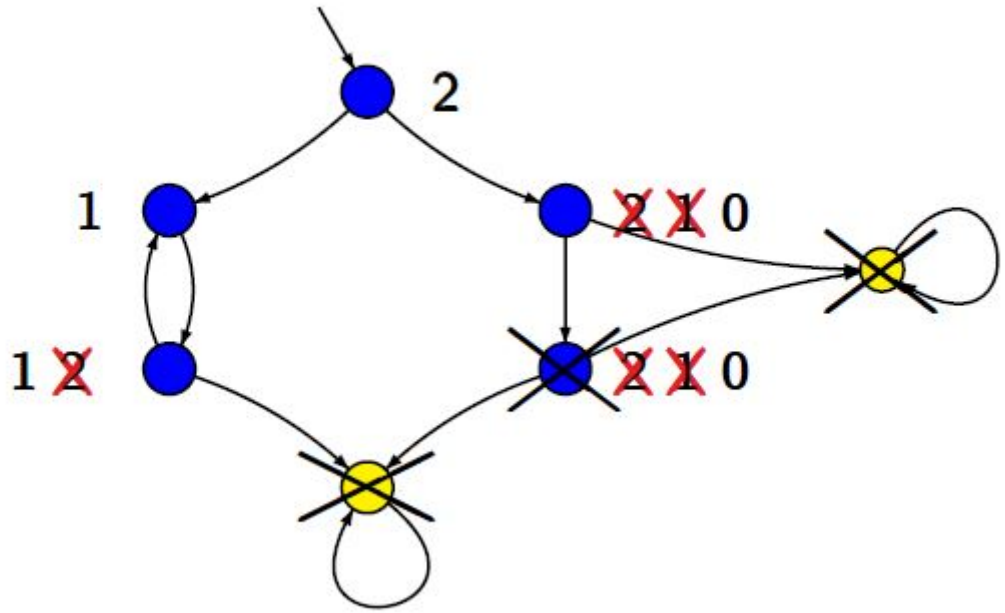
Example: CTL MC for EG

computation of $T = Sat(\exists\Box blue)$



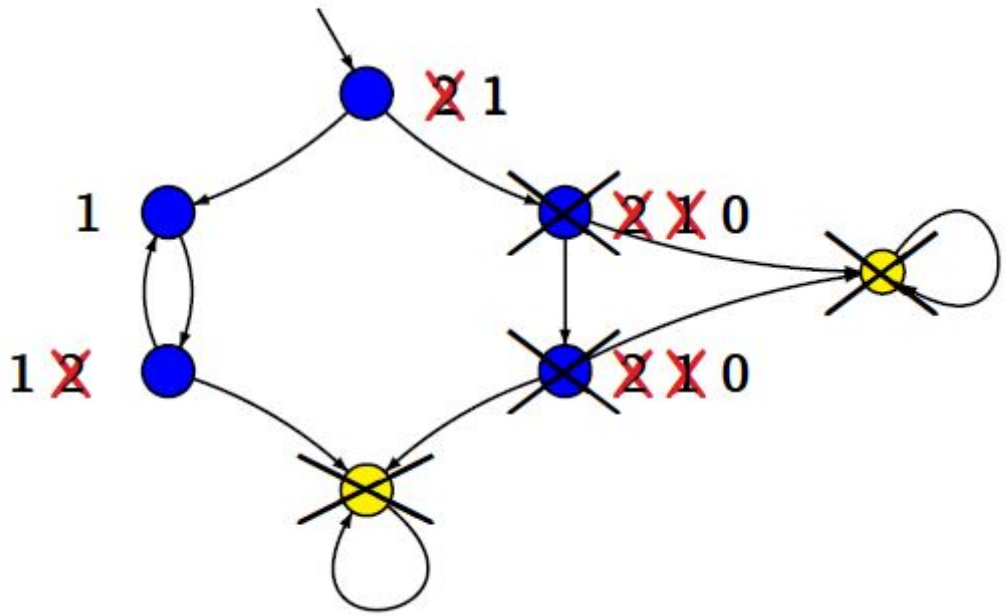
Example: CTL MC for EG

computation of $T = Sat(\exists\Box blue)$



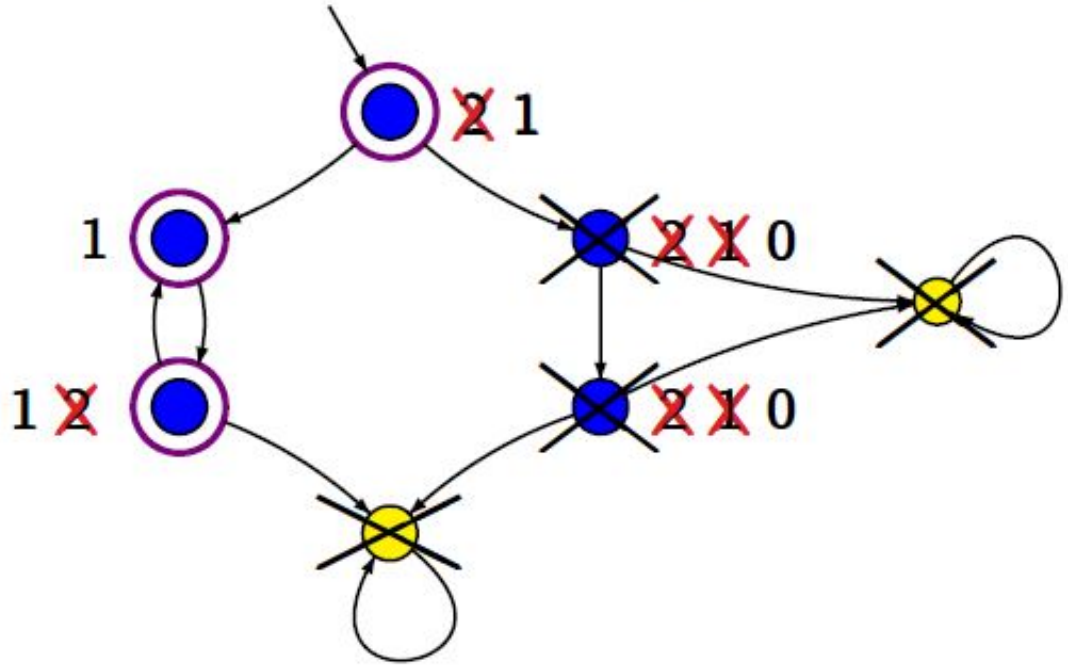
Example: CTL MC for EG

computation of $T = Sat(\exists\Box blue)$

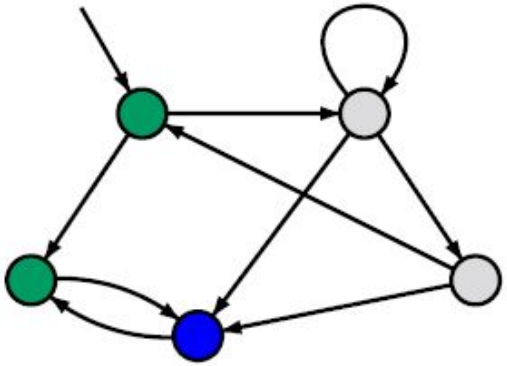


Example: CTL MC for EG

computation of $T = Sat(\exists\Box blue)$

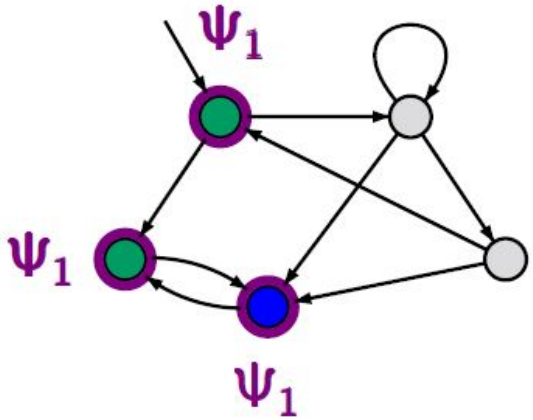


Example: CTL MC

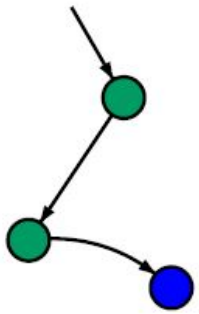


$$\Phi = \exists\Diamond\neg(\,\exists(a \,\mathsf{U}\, b) \;\vee\; \exists\Box\neg a\,)$$

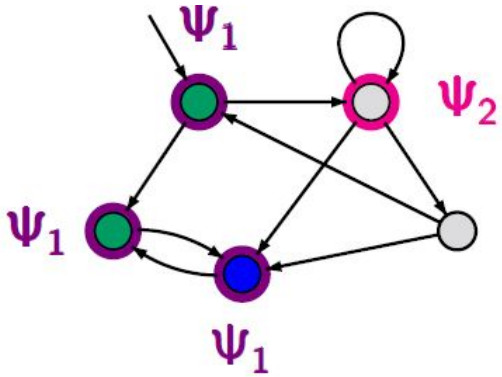
Example: CTL MC



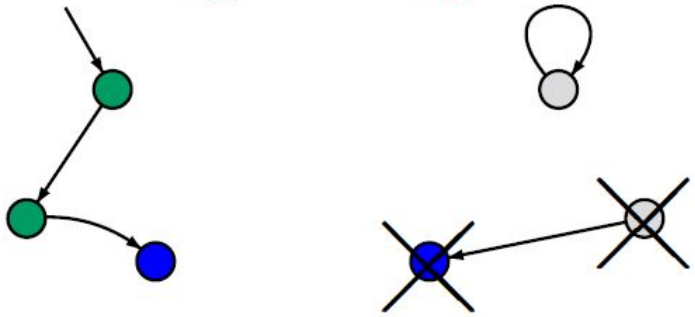
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \mathbf{U} b)}_{\psi_1} \vee \exists\Box\neg a)$$



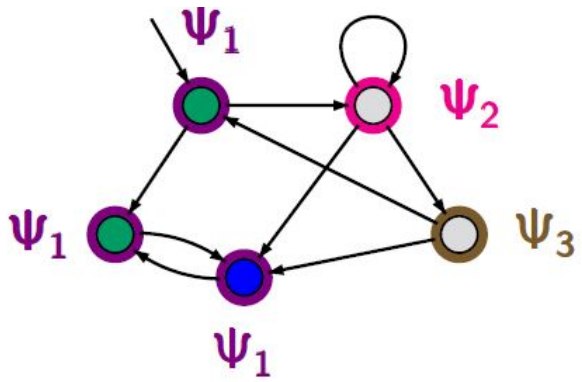
Example: CTL MC



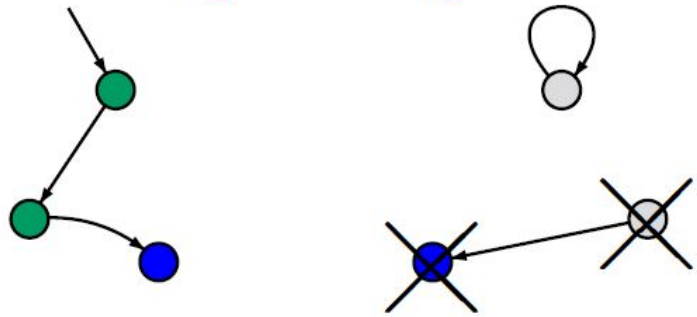
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \mathbf{U} b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2})$$



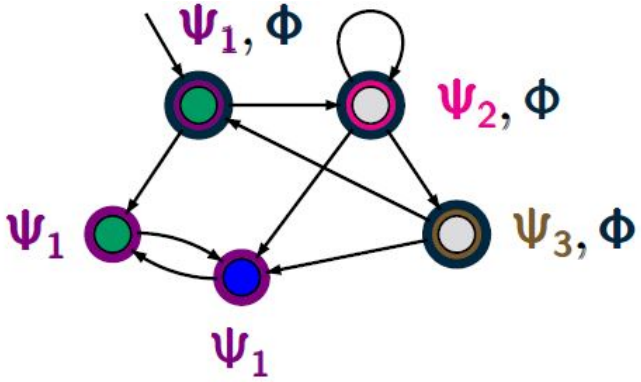
Example: CTL MC



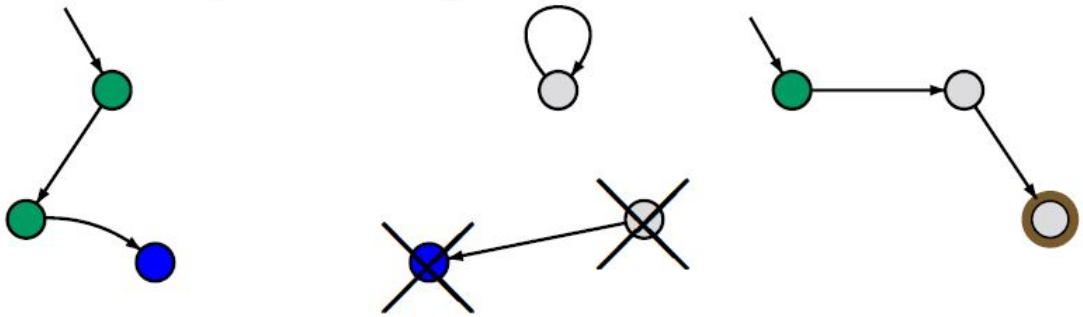
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \mathrm{U}\, b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2}) = \exists\Diamond\underbrace{\neg(\psi_1 \vee \psi_2)}_{\psi_3}$$



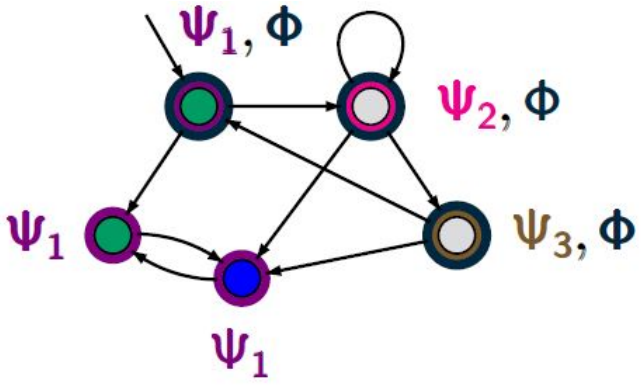
Example: CTL MC



$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \mathrel{\mathsf{U}} b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2}) = \exists\Diamond\underbrace{\neg(\psi_1 \vee \psi_2)}_{\psi_3}$$

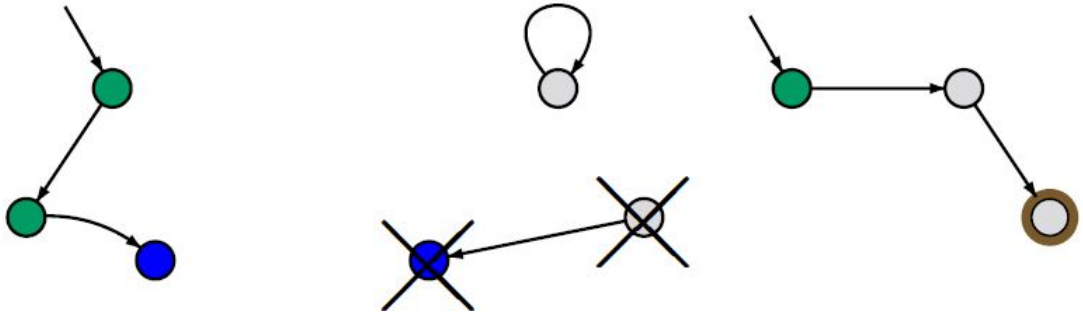


Example: CTL MC



$$\mathcal{T} \models \Phi$$

$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \mathrel{\mathsf{U}} b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2}) = \exists\Diamond\underbrace{\neg(\psi_1 \vee \psi_2)}_{\psi_3}$$



Expressiveness of CTL and LTL

CTL and LTL have a large overlap, i.e. properties expressible in both logics.
Examples:

Invariants (e.g., "p never holds.")

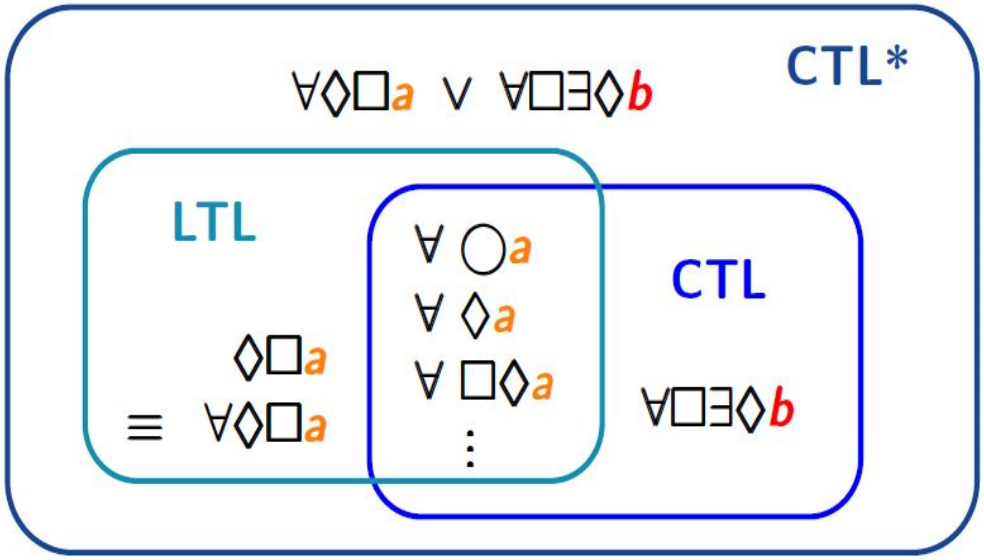
$$\mathbf{AG}\,\neg p \qquad \text{or} \qquad G\neg p$$

Reactivity ("Whenever p happens, eventually q will happen.")

$$\mathbf{AG}(p \rightarrow \mathbf{AF}\,q) \qquad \text{or} \qquad \mathbf{G}(p \rightarrow \mathbf{F}\,q)$$

*Expressiveness of CTL, LTL, and CTL**

- **CTL** is a sublogic of **CTL***
- **LTL** is a sublogic of **CTL***
- **CTL*** is more expressive than **LTL** and **CTL**



- Three logics have different expressive powers.
- There is no CTL formula is equivalent to the LTL formula

  FG p != AF(AG p)
- There is no LTL formula is equivalent to the CTL formula

  AG(EF p)
- The disjunction *AFGp \/ AG(EF p)* is a CTL* formula that is not expressible in either CTL or LTL