CTL model checking

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

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

CTL model checking

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 $S_0 \subseteq Sat(\Phi)$

CTL model checking

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

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

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

OD
```

CTL model checking

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
```

CTL model checking

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 model checking

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 S_0 ⊆ Sat(Φ) THEN output "yes"
                ELSE output "no"
FI
```

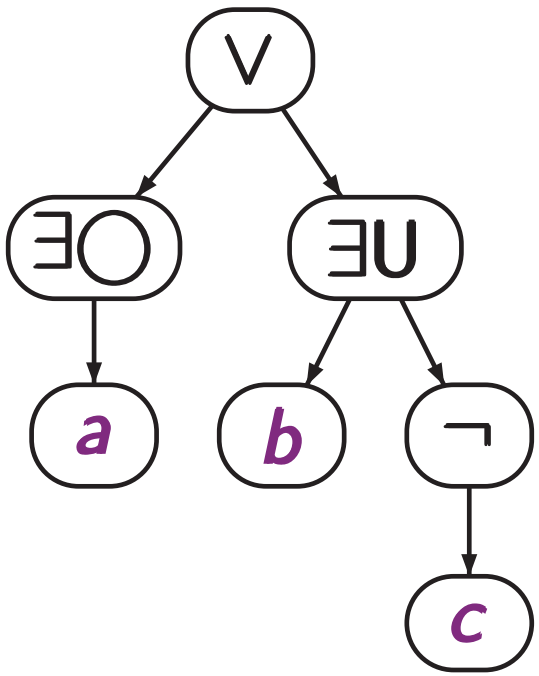
Example: CTL model checking

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

Example: CTL model checking

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

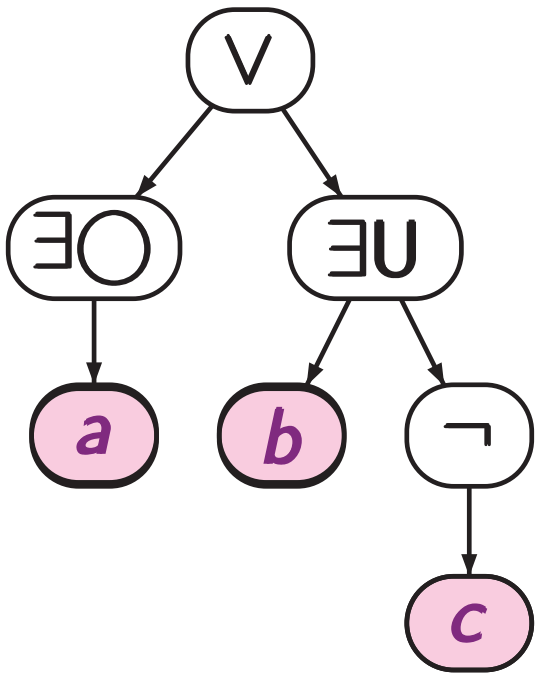
syntax tree for Φ



Example: CTL model checking

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

syntax tree for Φ



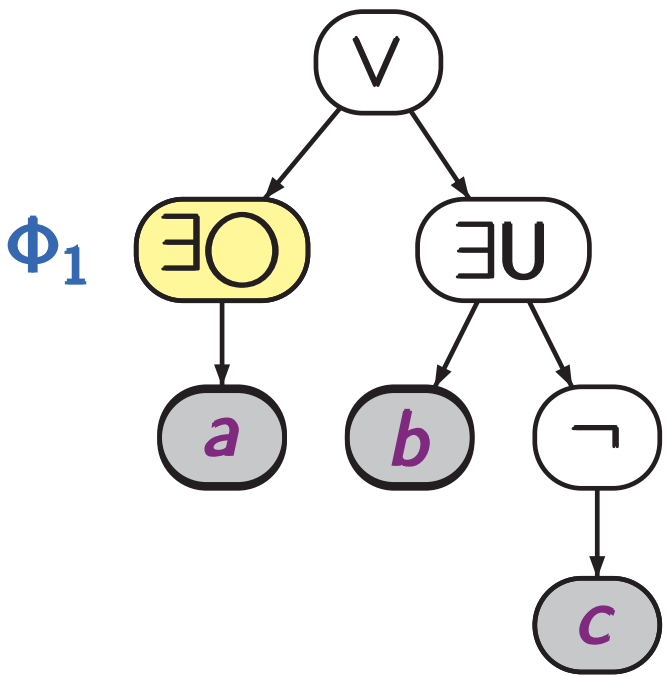
compute $Sat(a), Sat(b), Sat(c)$

processed in
bottom-up fashion

Example: CTL model checking

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

syntax tree for Φ



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

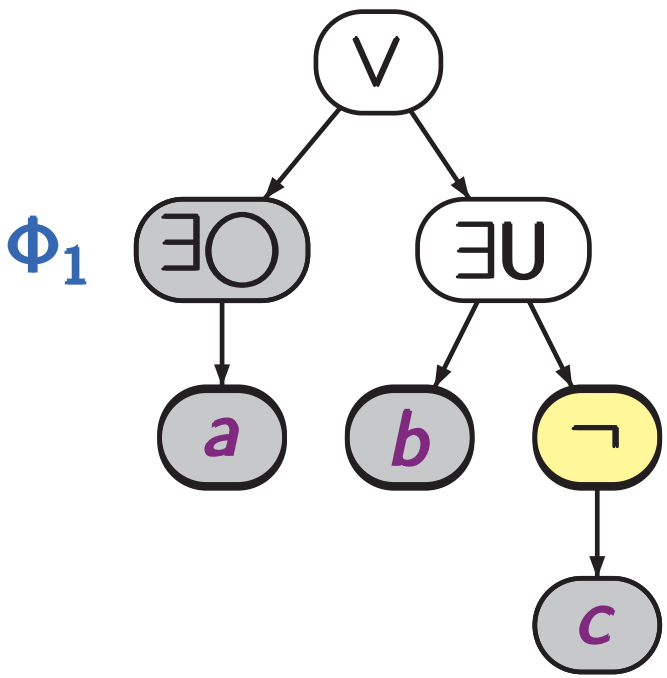
$Sat(\Phi_1) = \ldots$

processed in
bottom-up fashion

Example: CTL model checking

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

syntax tree for Φ



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

$Sat(\Phi_1) = \ldots$

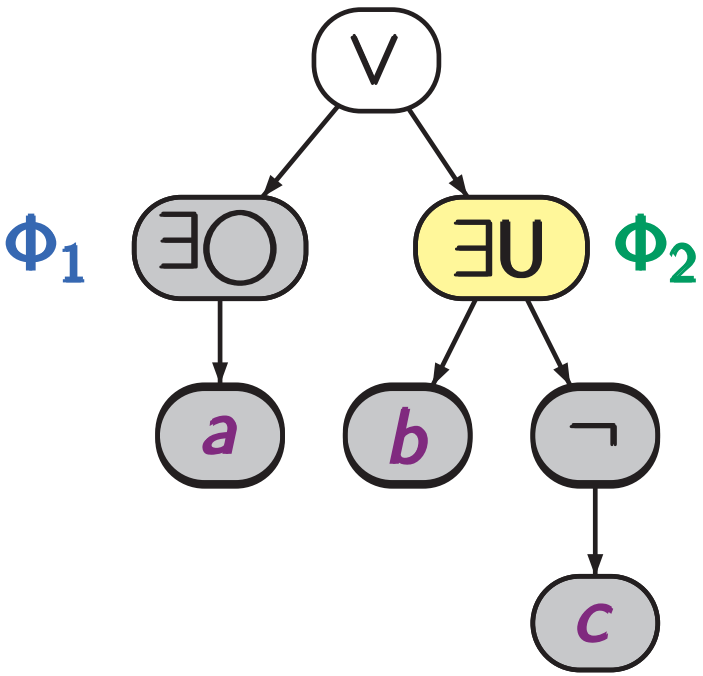
$Sat(\neg c) = S \setminus Sat(c)$

processed in
bottom-up fashion

Example: CTL model checking

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

syntax tree for Φ



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

$Sat(\Phi_1) = \ldots$

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

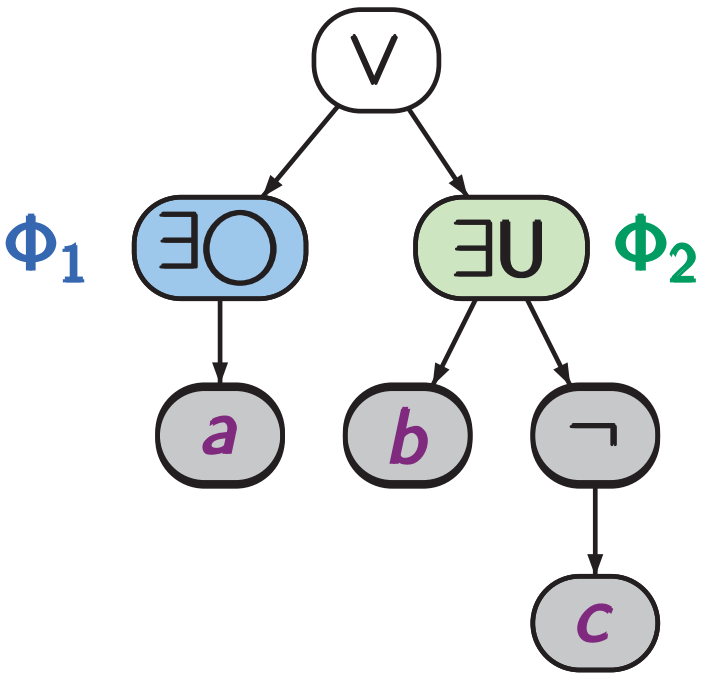
$Sat(\Phi_2) = \ldots$

processed in
bottom-up fashion

Example: CTL model checking

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

syntax tree for Φ



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

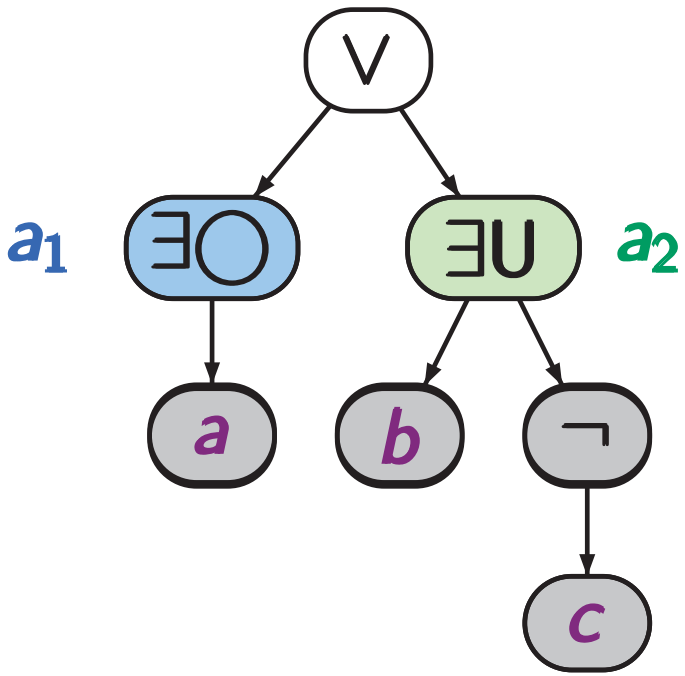
replace Φ_2 with a_2

processed in bottom-up fashion

Example: CTL model checking

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

syntax tree for Φ



processed in
bottom-up fashion

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

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

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

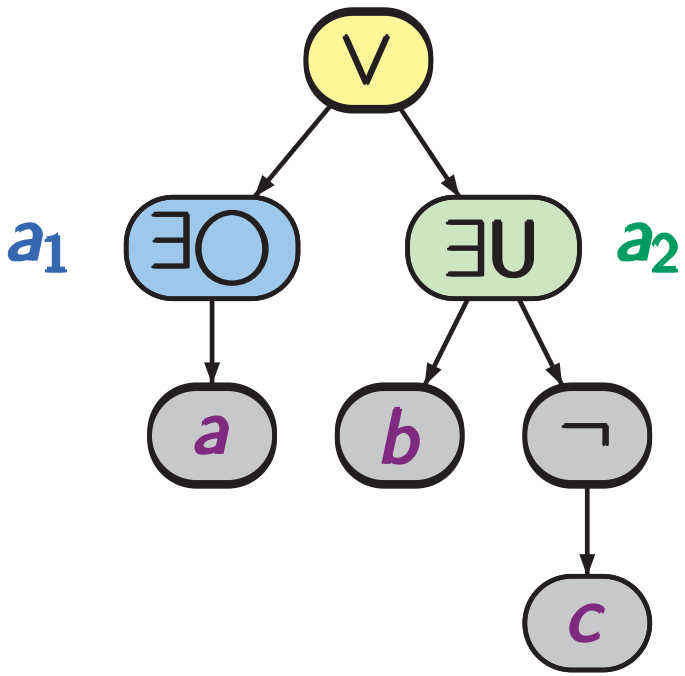
$Sat(\Phi_2) = \ldots = Sat(a_2)$

replace Φ_1 with a_1
replace Φ_2 with a_2

Example: CTL model checking

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

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

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

CTL model checking

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

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

method: regard in bottom-up manner all subformulas Ψ of Φ and compute their satisfaction sets

$$Sat(\Psi) = \{s \in S : s \models \Psi\}$$

CTL model checking

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

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

method: regard in bottom-up manner all subformulas Ψ of Φ and compute their satisfaction sets

$$Sat(\Psi) = \{s \in S : s \models \Psi\}$$

here: explanations for the case that Φ is in existential normal form

analogous algorithms can be designed for standard CTL (and the derived operators)

Recall: Existential normal form for CTL

For each **CTL** formula there is an equivalent formula in **$\exists$-normal form**, i.e., a **CTL** formula with the basis modalities $\exists\bigcirc$, $\exists\mathbf{U}$, $\exists\Box$.

Recall: Existential normal form for CTL

For each **CTL** formula there is an equivalent formula in **∃-normal form**, i.e., a **CTL** formula with the basis modalities $\exists\bigcirc$, $\exists\mathbf{U}$, $\exists\Box$.

CTL formulas in ∃-normal form:

$$\begin{aligned}\Psi \quad ::= \quad & \mathit{true} \mid a \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid \\ & \exists\bigcirc\Psi \mid \exists(\Psi_1 \,\mathbf{U}\, \Psi_2) \mid \exists\Box\Psi\end{aligned}$$

Recall: Existential normal form for CTL

For each **CTL** formula there is an equivalent formula in **∃-normal form**, i.e., a **CTL** formula with the basis modalities $\exists\bigcirc$, $\exists\mathbf{U}$, $\exists\Box$.

CTL formulas in ∃-normal form:

$$\Psi ::= \mathit{true} \mid a \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid \exists\bigcirc\Psi \mid \exists(\Psi_1 \,\mathbf{U}\, \Psi_2) \mid \exists\Box\Psi$$

CTL formula $\rightsquigarrow$ **CTL** formula in ∃-normal form

$$\forall\bigcirc\Phi \rightsquigarrow \neg\exists\bigcirc\neg\Phi$$

$$\forall(\Phi_1 \,\mathbf{U}\, \Phi_2) \rightsquigarrow \neg\exists(\neg\Phi_2 \,\mathbf{U}\, (\neg\Phi_1 \wedge \neg\Phi_2)) \wedge \neg\exists\Box\neg\Phi_2$$

Recursive computation of the satisfaction sets

Recursive computation of the satisfaction sets

$$Sat(true) = S$$

Recursive computation of the satisfaction sets

$$Sat(true) = S$$

$$Sat(a) = \{s \in S : a \in L(s)\}$$

Recursive computation of the satisfaction sets

$$
\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s \in S : a \in L(s)\} \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi)
\end{aligned}
$$

Recursive computation of the satisfaction sets

$$
\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s \in S : a \in L(s)\} \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) &= Sat(\Phi_1) \cap Sat(\Phi_2)
\end{aligned}
$$

Recursive computation of the satisfaction sets

$$
\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s \in S : a \in L(s)\} \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) &= Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\exists \bigcirc \Phi) &= \{s \in S : Post(s) \cap Sat(\Phi) \neq \varnothing\}
\end{aligned}
$$

Recursive computation of the satisfaction sets

$$
\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s \in S : a \in L(s)\} \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) &= Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\exists \bigcirc \Phi) &= \{s \in S : Post(s) \cap Sat(\Phi) \neq \varnothing\} \\
Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) &= \ldots \\
Sat(\exists \Box \Phi) &= \ldots
\end{aligned}
$$

treatment of $\exists \mathsf{U}$ and $\exists \Box$:

via fixed point computation

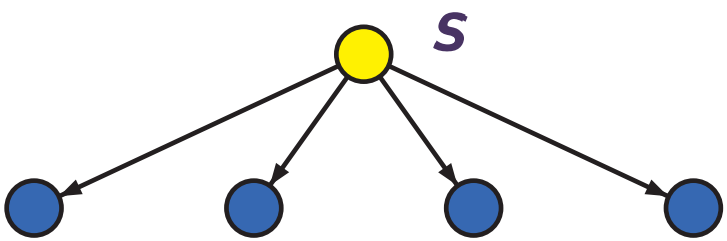
Universally quantified formulas

Universally quantified formulas

$$Sat(\forall\bigcirc a) = \{s \in S : Post(s) \subseteq Sat(a)\}$$

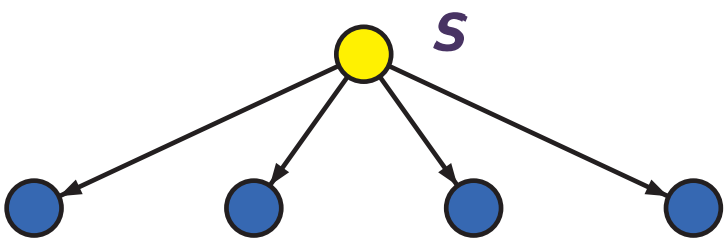
Universally quantified formulas

$$Sat(\forall\bigcirc a) = \{s \in S : Post(s) \subseteq Sat(a)\}$$



Universally quantified formulas

$$Sat(\forall\bigcirc a) = \{s \in S : Post(s) \subseteq Sat(a)\}$$

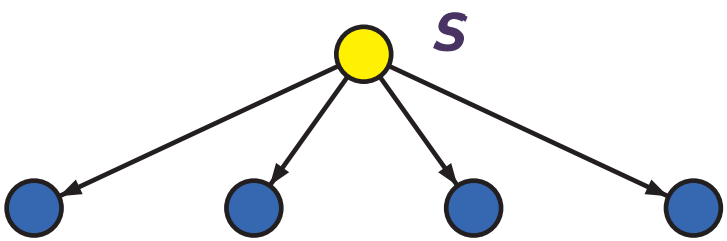


$Sat(\forall\Box a) =$ greatest set T of states s.t.

$$T \subseteq \{s \in Sat(a) : Post(s) \subseteq T\}$$

Universally quantified formulas

$$Sat(\forall\bigcirc a) = \{s \in S : Post(s) \subseteq Sat(a)\}$$



$Sat(\forall\Box a)$ = greatest set T of states s.t.

$$T \subseteq \{s \in Sat(a) : Post(s) \subseteq T\}$$

$Sat(\forall(a \,\mathsf{U}\, b))$ = smallest set T of states s.t.

$$Sat(b) \cup \{s \in Sat(a) : Post(s) \subseteq T\} \subseteq T$$

Recursive computation of *Sat*(...)

Recursive computation of $Sat(\ldots)$

$$Sat(\Phi_1 \wedge \Phi_2) = Sat(\Phi_1) \cap Sat(\Phi_2)$$

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

$$Sat(\exists\bigcirc\Phi) = \{s \in S : Post(s) \cap Sat(\Phi) = \varnothing\}$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ = smallest set T of states s.t.

- $Sat(\Phi_2) \subseteq T$
- $s \in Sat(\Phi_1)$ and $Post(s) \cap T \neq \varnothing \implies s \in T$

$Sat(\exists\Box\Phi)$ = greatest set V of states s.t.

- $V \subseteq Sat(\Phi)$
- $s \in V \implies Post(s) \cap V \neq \varnothing$

CTL model checking: until operator

CTL model checking: until operator

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

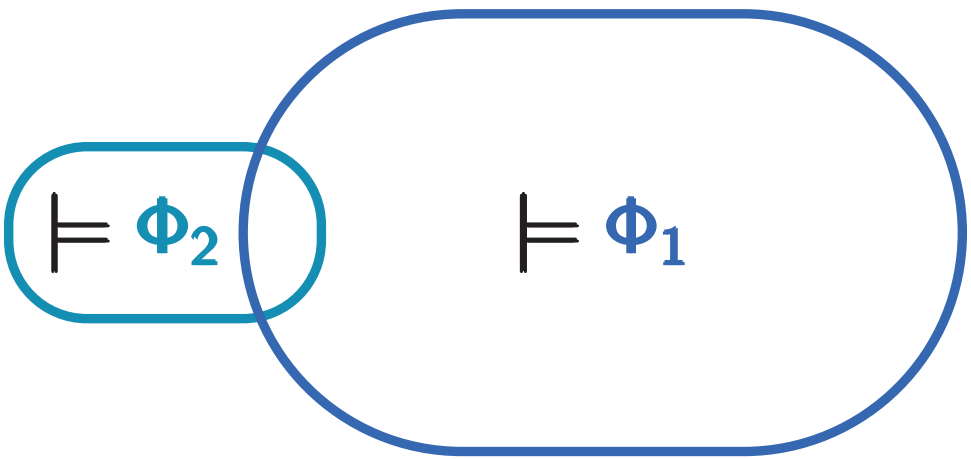
$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$

CTL model checking: until operator

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



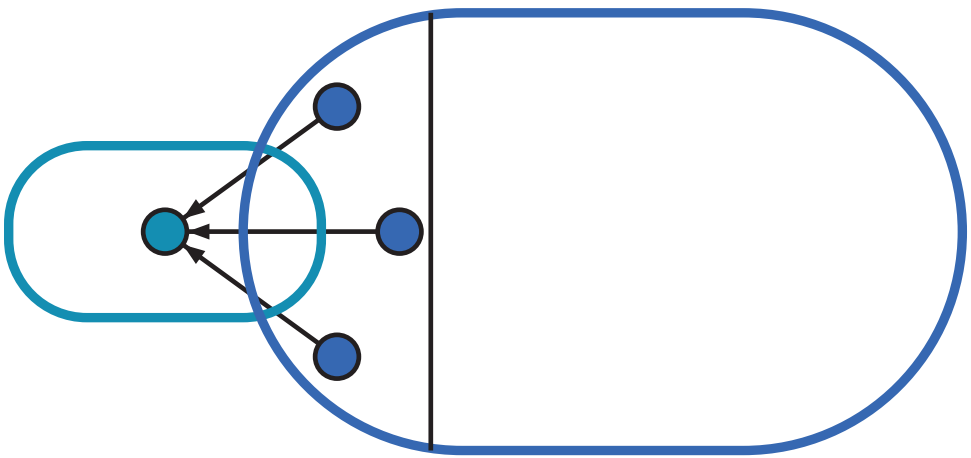
$$T_0 := Sat(\Phi_2)$$

CTL model checking: until operator

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



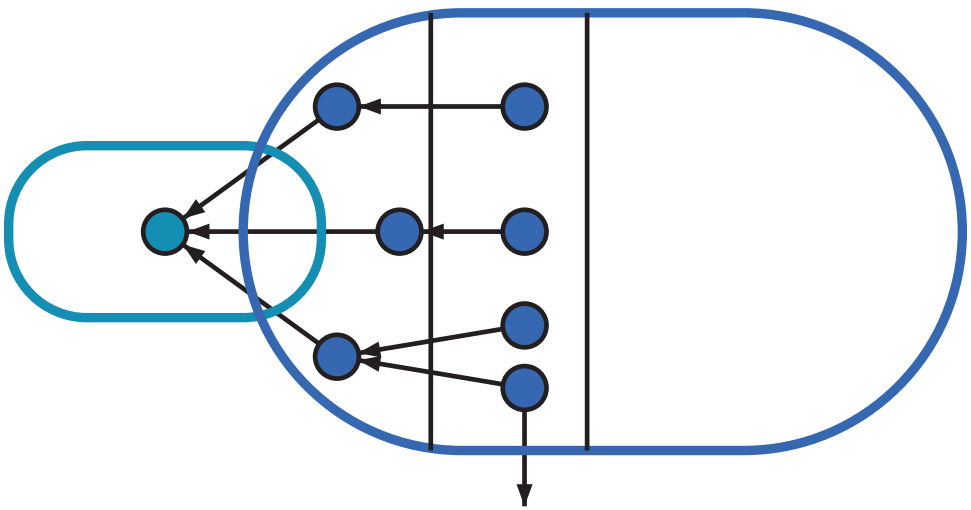
$$T_0 := Sat(\Phi_2)$$

$$T_{n+1} := T_n \cup \{s \in Sat(\Phi_1) : Post(s) \cap T_n \neq \varnothing\}$$

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



$$T_0 := Sat(\Phi_2)$$

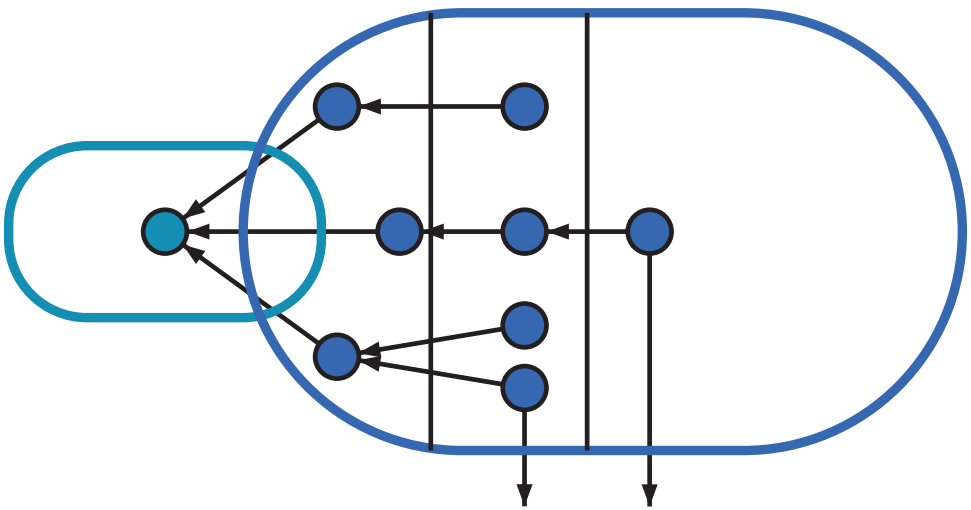
$$T_{n+1} := T_n \cup \{s \in Sat(\Phi_1) : Post(s) \cap T_n \neq \varnothing\}$$

CTL model checking: until operator

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



$$T_0 := Sat(\Phi_2)$$

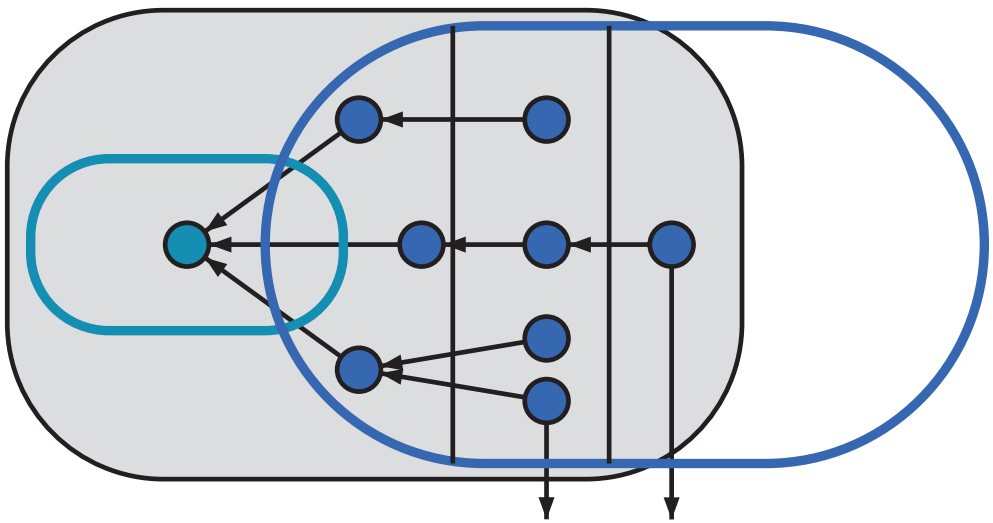
$$T_{n+1} := T_n \cup \{s \in Sat(\Phi_1) : Post(s) \cap T_n \neq \varnothing\}$$

CTL model checking: until operator

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



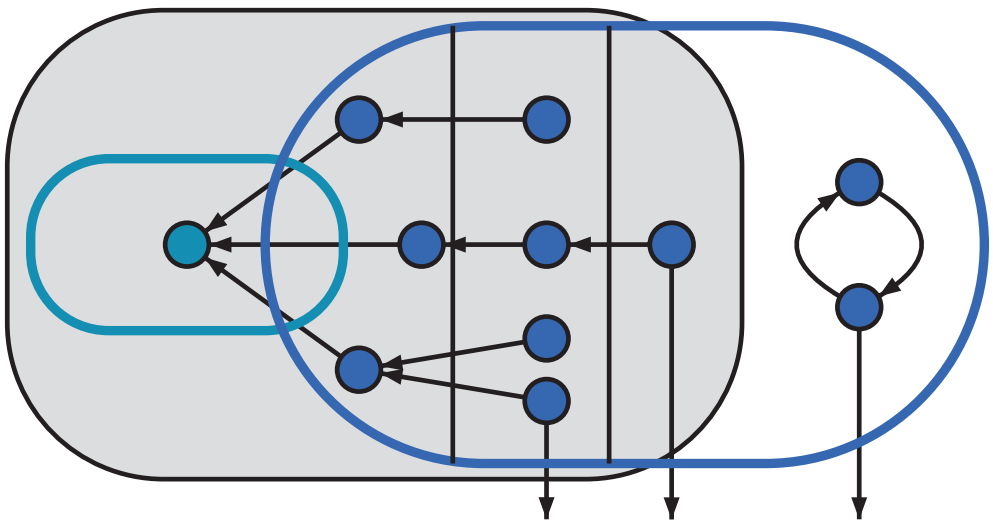
$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$

CTL model checking: until operator

$$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2) \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$$

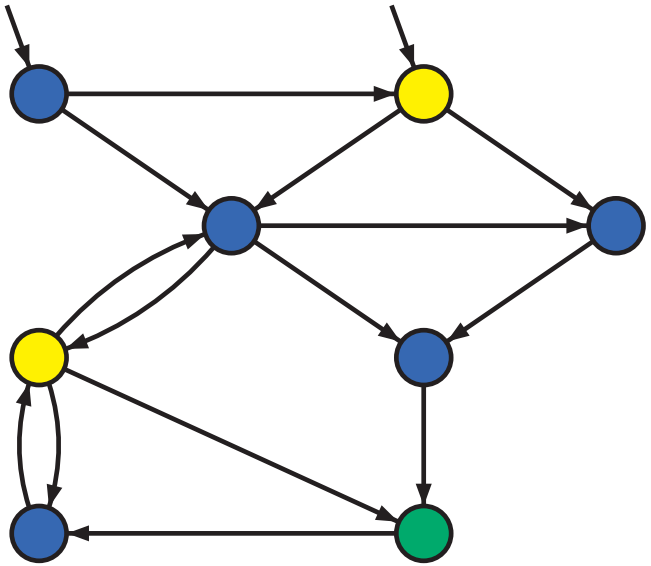
$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) =$ least set T of states s.t.

$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$

Example: until operator

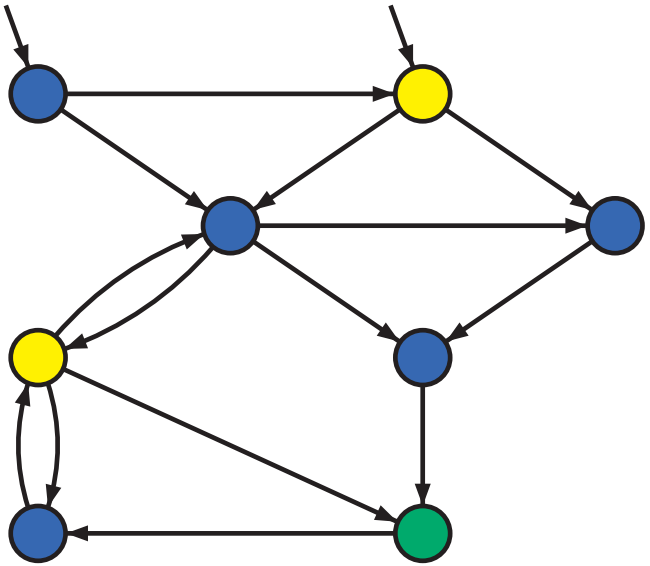


● = $\{a\}$

● = $\{b\}$

● = $\varnothing$

Example: until operator



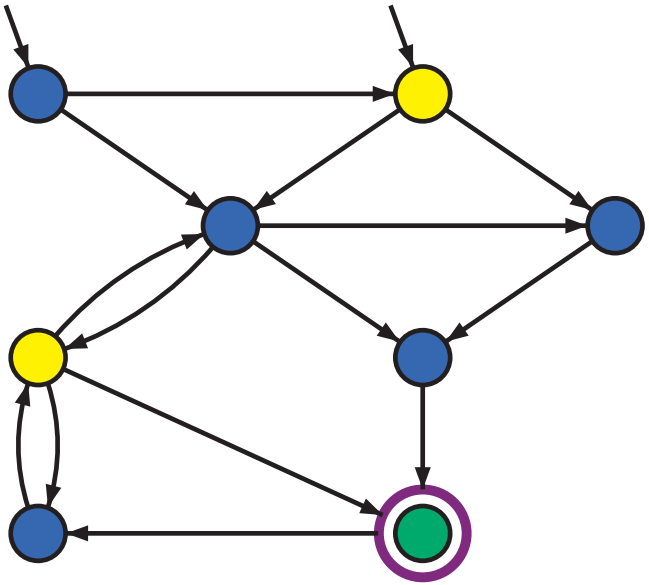
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

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: until operator



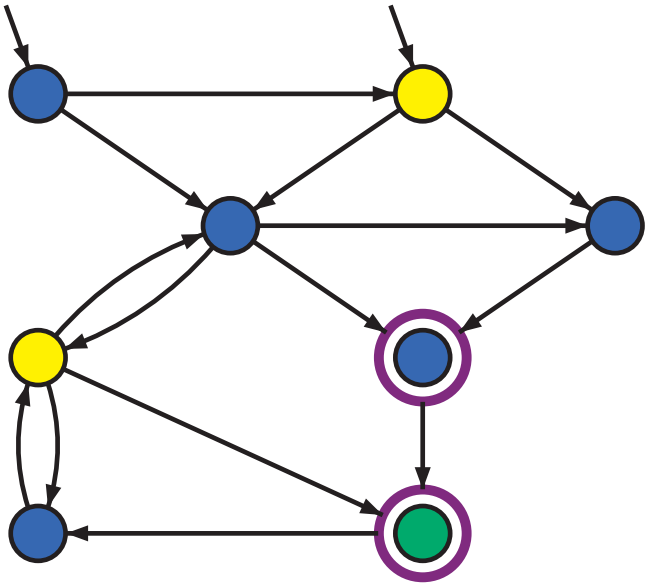
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

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: until operator



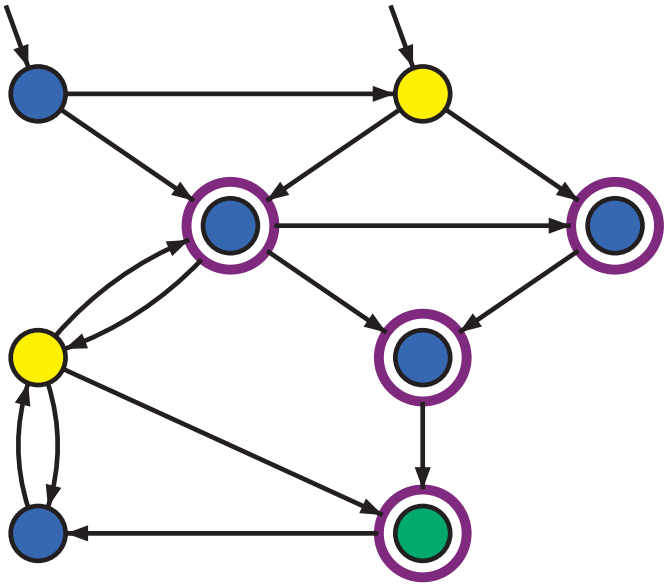
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

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: until operator



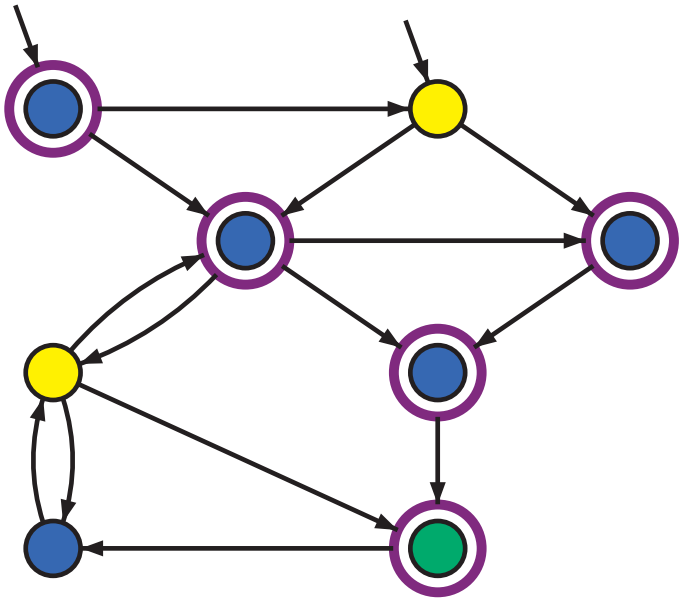
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

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: until operator



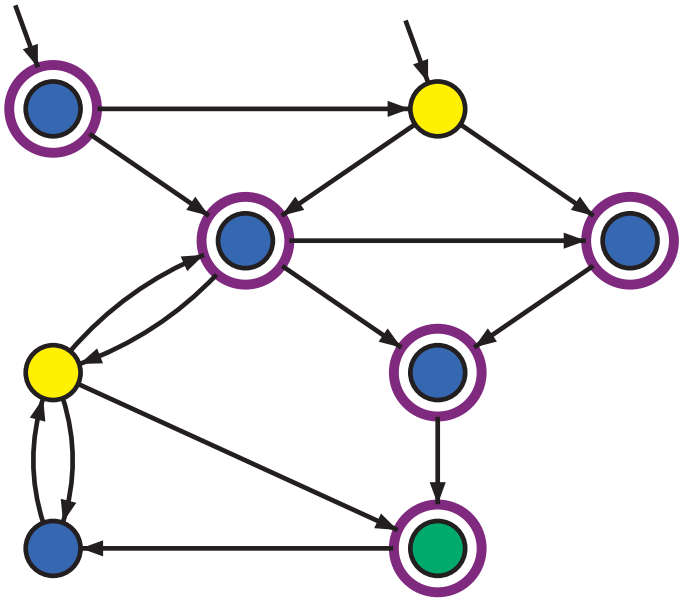
● = $\{a\}$
● = $\{b\}$
● = $\varnothing$

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: until operator



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

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

compute $\mathit{Sat}(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

`WHILE` $E \neq \varnothing$ `DO`

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

`WHILE` $E \neq \varnothing$ `DO`

select a state $s' \in E$ and remove s' from E

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

```
WHILE E ≠ ∅ DO
    select a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
```

CTL model checking: treatment of $\exists U$

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

```
WHILE E ≠ ∅ DO
    select a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
        IF s ∈ Sat(Φ1) \ T THEN add s to T and E FI
    OD
OD
```

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

```
WHILE E ≠ ∅ DO
    select a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
        IF s ∈ Sat(Φ1) \ T THEN add s to T and E FI
    OD
OD
return T
```

compute $Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2))$ via an enumerative backward search

$T := Sat(\Phi_2)$ ⟵ collects all states $s \models \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$

$E := Sat(\Phi_2)$ ⟵ set of states still to be expanded

```
WHILE E ≠ ∅ DO
    select a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
        IF s ∈ Sat(Φ1) \ T THEN add s to T and E FI
    OD
OD
return T
```

complexity: $\mathcal{O}(size(\mathcal{T}))$

CTL model checking: always operator

CTL model checking: always operator

expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = greatest set T of states with

$$T \subseteq \{s \in Sat(\Phi) : Post(s) \cap T \neq \varnothing\}$$

CTL model checking: always operator

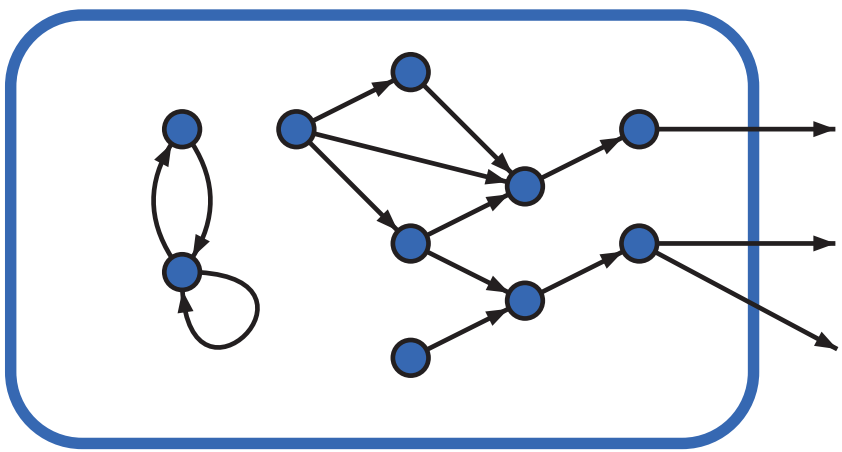
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = greatest set T of states with

$$T \subseteq \{s \in Sat(\Phi) : Post(s) \cap T \neq \varnothing\}$$

$$T_0 := Sat(\Phi), \quad T_{n+1} := \{s \in T_n : Post(s) \cap T_n \neq \varnothing\}$$

$Sat(\Phi)$



CTL model checking: always operator

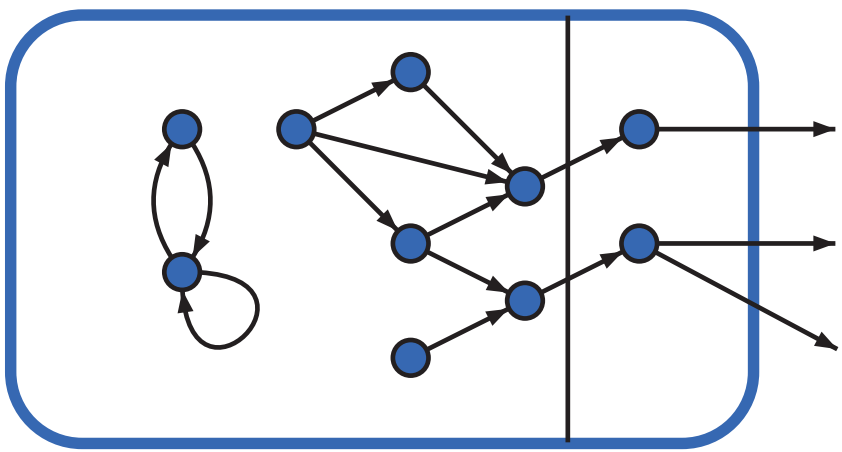
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = greatest set T of states with

$$T \subseteq \{s \in Sat(\Phi) : Post(s) \cap T \neq \varnothing\}$$

$$T_0 := Sat(\Phi), \quad T_{n+1} := \{s \in T_n : Post(s) \cap T_n \neq \varnothing\}$$

$Sat(\Phi)$



CTL model checking: always operator

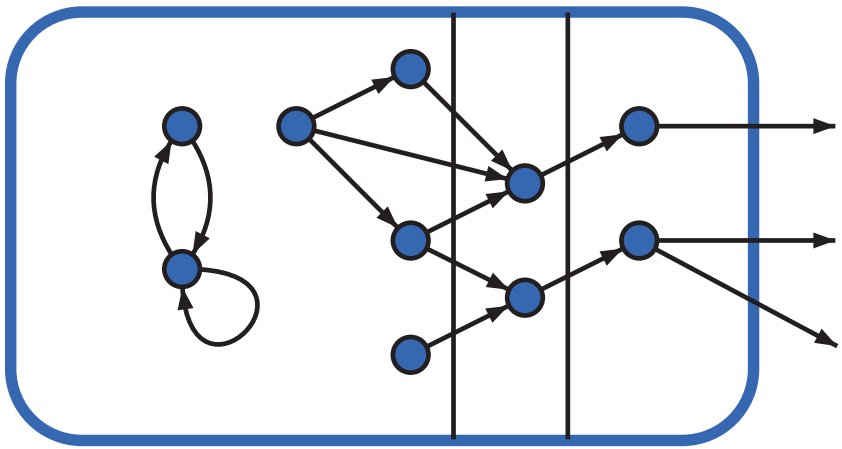
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = greatest set T of states with

$$T \subseteq \{s \in Sat(\Phi) : Post(s) \cap T \neq \varnothing\}$$

$$T_0 := Sat(\Phi), \quad T_{n+1} := \{s \in T_n : Post(s) \cap T_n \neq \varnothing\}$$

$Sat(\Phi)$



CTL model checking: always operator

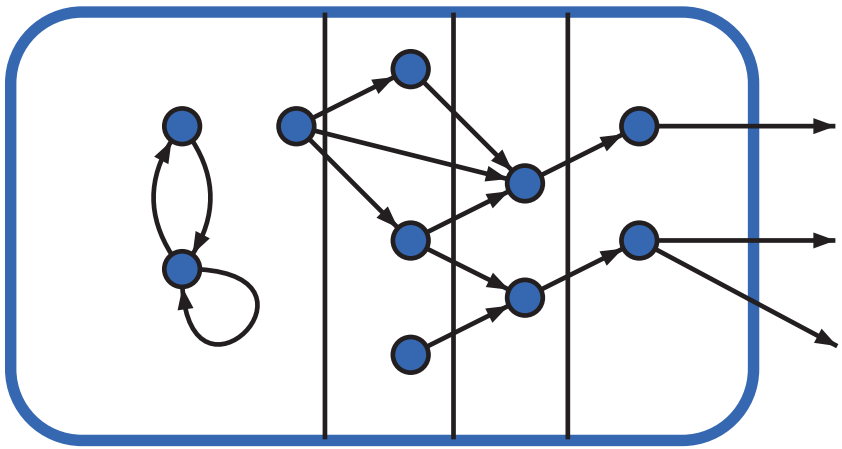
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = greatest set T of states with

$$T \subseteq \{s \in Sat(\Phi) : Post(s) \cap T \neq \varnothing\}$$

$$T_0 := Sat(\Phi), \quad T_{n+1} := \{s \in T_n : Post(s) \cap T_n \neq \varnothing\}$$

$Sat(\Phi)$



CTL model checking: always operator

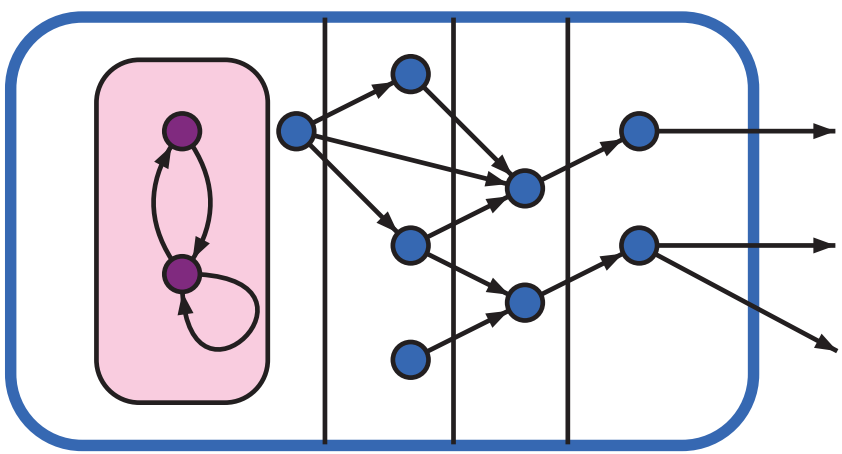
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = greatest set T of states with

$$T \subseteq \{s \in Sat(\Phi) : Post(s) \cap T \neq \varnothing\}$$

$$T_0 := Sat(\Phi), \quad T_{n+1} := \{s \in T_n : Post(s) \cap T_n \neq \varnothing\}$$

$Sat(\Phi)$



Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi) \leftarrow$ organizes the candidates for $s \models \exists\Box\Phi$

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

WHILE $E \neq \varnothing$ DO

pick a state $s' \in E$ and remove s' from E

OD

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

WHILE $E \neq \emptyset$ DO

pick a state $s' \in E$ and remove s' from E

FOR ALL $s \in Pre(s')$ DO

OD

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

```
WHILE E ≠ ∅ DO
   pick a state s' ∈ E and remove s' from E
   FOR ALL s ∈ Pre(s') DO
    IF s ∈ T and Post(s) ∩ T = ∅ THEN
         remove s from T and add s to E
    FI
OD
```

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

```
WHILE E ≠ ∅ DO
    pick a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
      IF s ∈ T and Post(s) ∩ T = ∅ THEN
          remove s from T and add s to E
      FI
OD
return T
```

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

```
WHILE E ≠ ∅ DO
    pick a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
      IF s ∈ T and Post(s) ∩ T = ∅ THEN
          remove s from T and add s to E
      FI
OD
return T
```

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

```
WHILE E ≠ ∅ DO
    pick a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
      IF s ∈ T and Post(s) ∩ T = ∅ THEN
          remove s from T and add s to E
      FI
OD
return T
```

naïve implementation:
quadratic time complexity

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

```
WHILE E ≠ ∅ DO
    pick a state s' ∈ E and remove s' from E
    FOR ALL s ∈ Pre(s') DO
      IF s ∈ T and Post(s) ∩ T = ∅ THEN
          remove s from T and add s to E
      FI
OD
return T
```

linear time implementation:
uses counters $c[s]$

Computation of $Sat(\exists\Box\Phi)$

$T := Sat(\Phi)$ ← organizes the candidates for $s \models \exists\Box\Phi$

$E := S \setminus T$ ← set of states to be expanded

```
WHILE E ≠ ∅ DO
   pick a state s' ∈ E and remove s' from E
   FOR ALL s ∈ Pre(s') DO
    IF s ∈ T and Post(s) ∩ (T ∪ E) = ∅ THEN
         remove s from T and add s to E
    FI
OD
return T
```

linear time implementation:

uses counters $c[s]$ for $|Post(s) \cap (T \cup E)|$

Computation of $Sat(\exists\Box\Phi)$ using counters

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

WHILE $E \neq \emptyset$ DO
  pick a state $s' \in E$ and remove s' from E
  FOR ALL $s \in Pre(s')$ DO
    IF $s \in T$ and $Post(s) \cap (T \cup E) = \emptyset$ THEN
      remove s from T and add s to E
    FI
OD

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

use counters $c[s]$ for $|Post(s) \cap (T \cup E)|$

WHILE $E \neq \emptyset$ DO
  pick a state $s' \in E$ and remove s' from E
  FOR ALL $s \in Pre(s')$ DO
    IF $s \in T$ and $Post(s) \cap (T \cup E) = \emptyset$ THEN
      remove s from T and add s to E
    FI
OD

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

use counters $c[s]$ for $|Post(s) \cap (T \cup E)|$

WHILE $E \neq \emptyset$ DO

pick a state $s' \in E$ and remove s' from E

FOR ALL $s \in Pre(s')$ DO

IF $s \in T$ and $Post(s) \cap (T \cup E) = \emptyset$ THEN

remove s from T and add s to E

FI

OD

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

loop invariant: $c[s] = |Post(s) \cap (T \cup E)|$ for $s \in T$

WHILE $E \neq \emptyset$ DO

pick a state $s' \in E$ and remove s' from E

FOR ALL $s \in Pre(s')$ DO

IF $s \in T$ and $Post(s) \cap (T \cup E) = \emptyset$ THEN

remove s from T and add s to E

FI

OD

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

loop invariant: $c[s] = |Post(s) \cap (T \cup E)|$ for $s \in T$

WHILE $E \neq \emptyset$ DO

pick a state $s' \in E$ and remove s' from E

FOR ALL $s \in Pre(s')$ DO

IF $s \in T$ and ~~$Post(s) \cap (T \cup E) = \emptyset$~~ THEN

remove s from T and add s to E

FI

OD

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

loop invariant: $c[s] = |Post(s) \cap (T \cup E)|$ for $s \in T$

WHILE $E \neq \emptyset$ DO

pick a state $s' \in E$ and remove s' from E

FOR ALL $s \in Pre(s')$ DO

IF $s \in T$ THEN

$c[s] := c[s] - 1$

IF $c[s] = 0$ THEN

remove s from T and add s to E FI

FI

OD

Computation of $Sat(\exists\Box\Phi)$ using counters

$T := Sat(\Phi)$; $E := S \setminus T$

FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

loop invariant: $c[s] = |Post(s) \cap (T \cup E)|$ for $s \in T$

WHILE $E \neq \emptyset$ DO

pick a state $s' \in E$ and remove s' from E

FOR ALL $s \in Pre(s')$ DO

IF $s \in T$ THEN

$c[s] := c[s] - 1$

IF $c[s] = 0$ THEN

remove s from T and add s to E FI

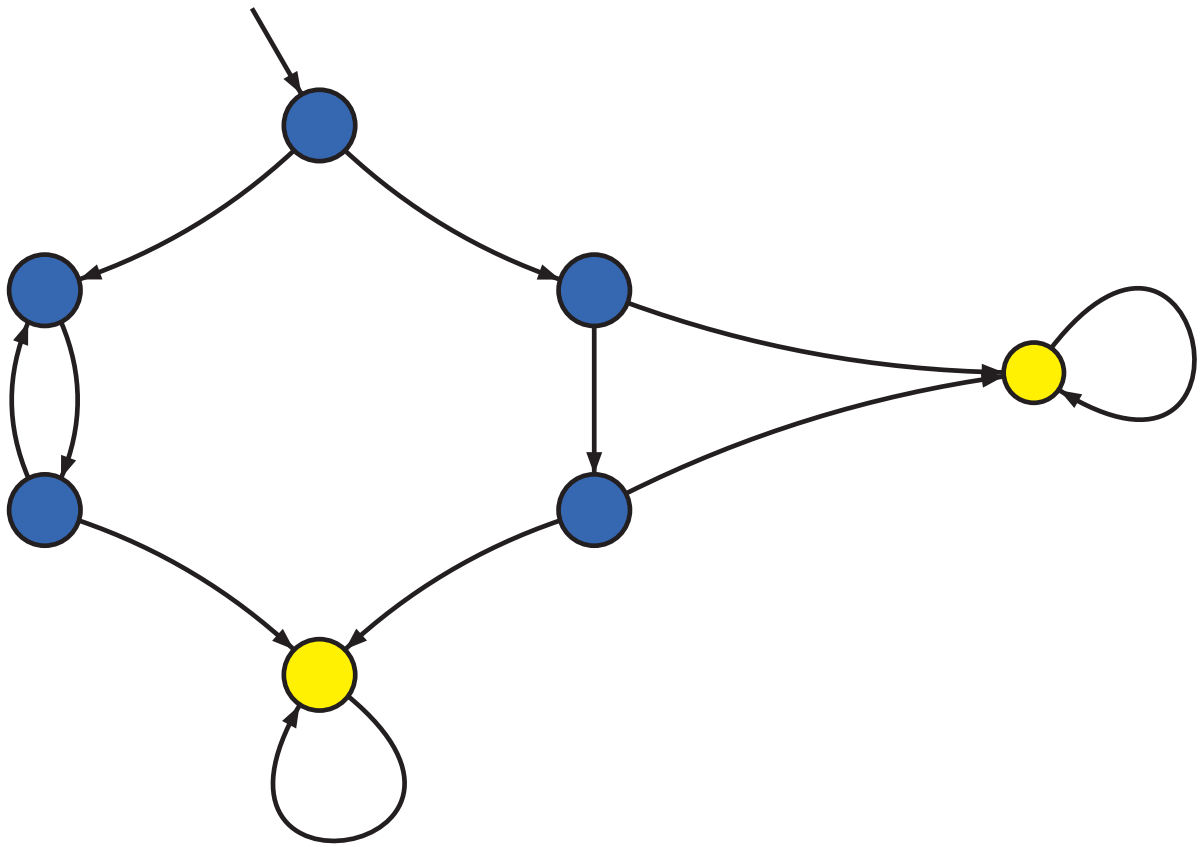
FI

OD

complexity:
$\mathcal{O}(size(\mathcal{T}))$

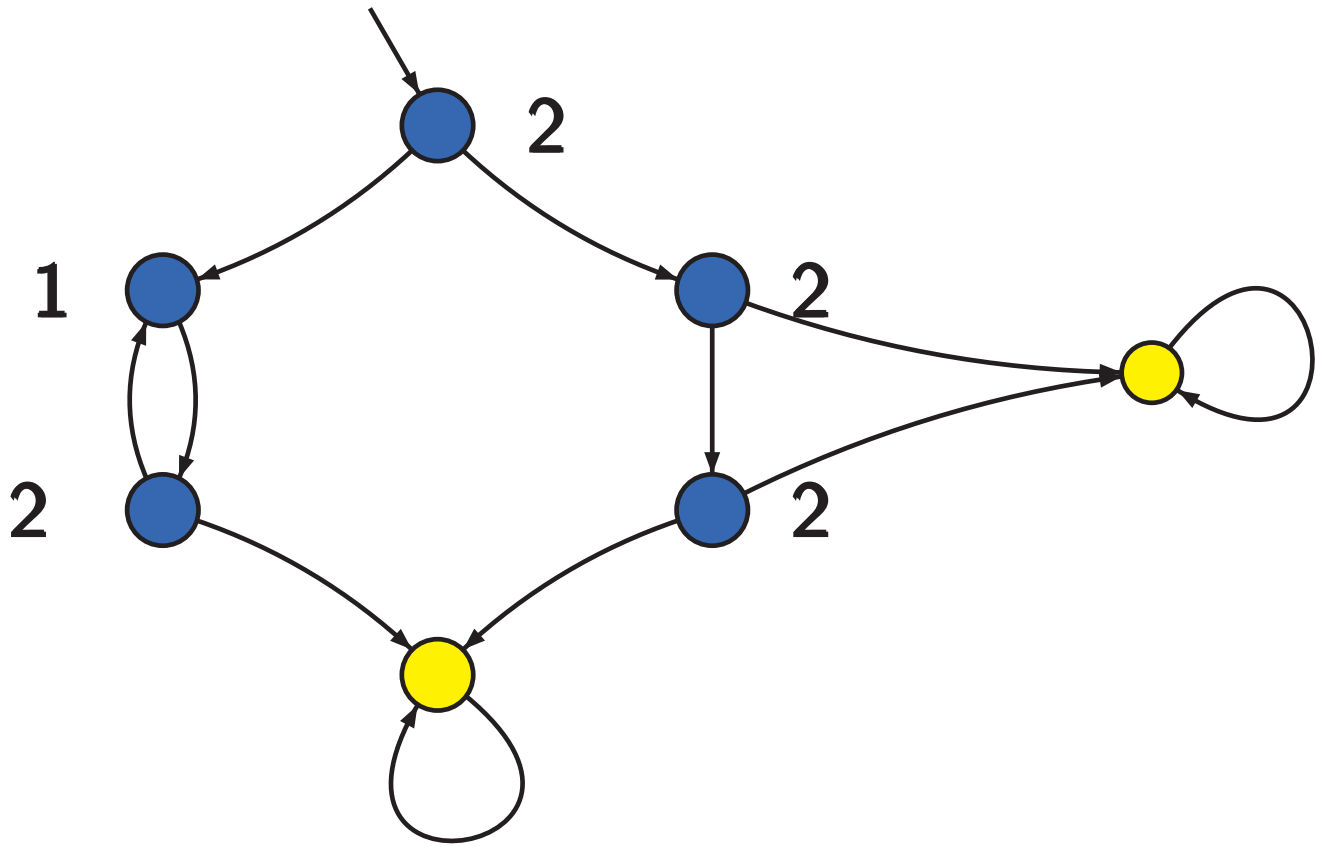
Example: CTL model checking for $\exists\Box$

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



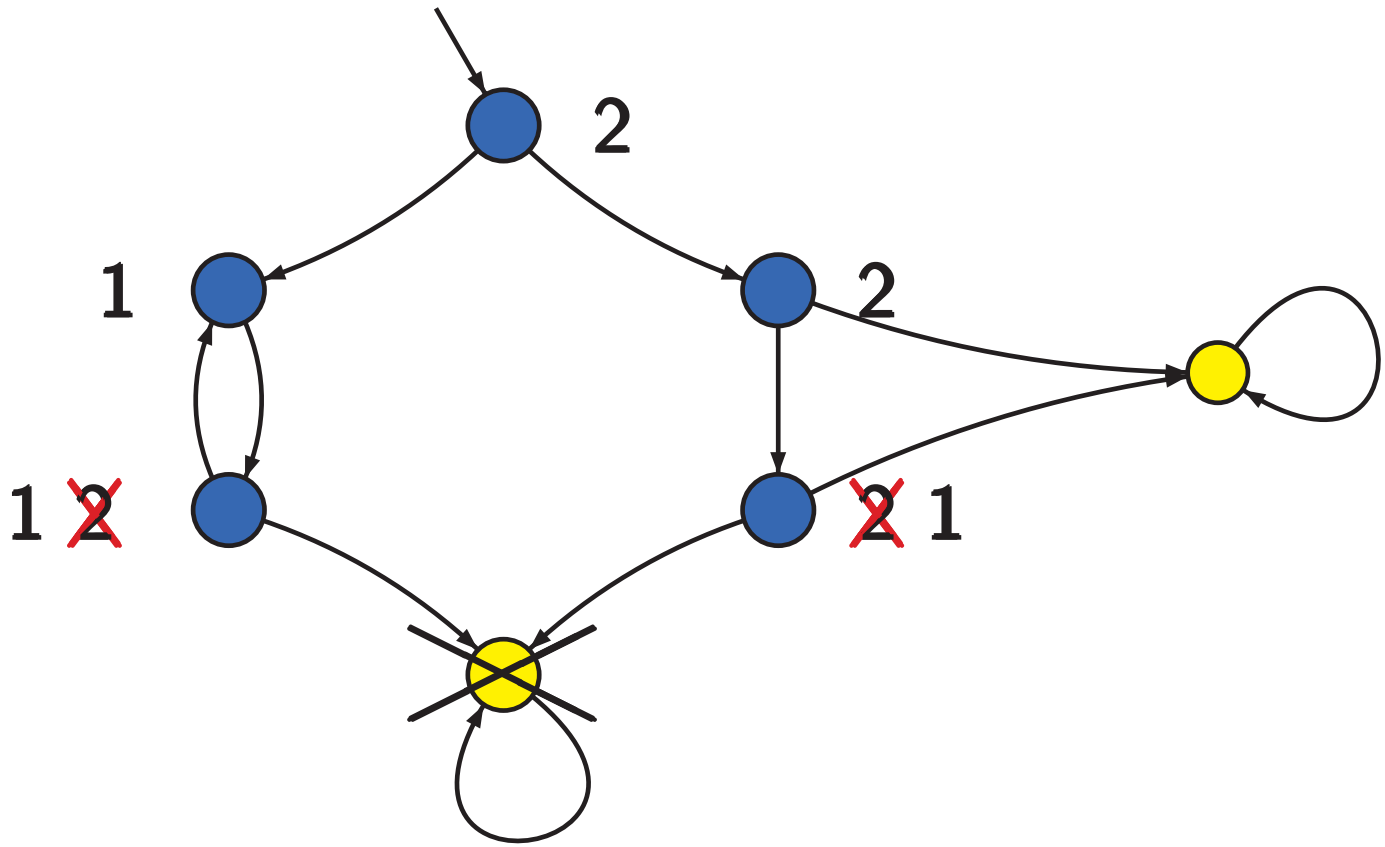
Example: CTL model checking for ∃□

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



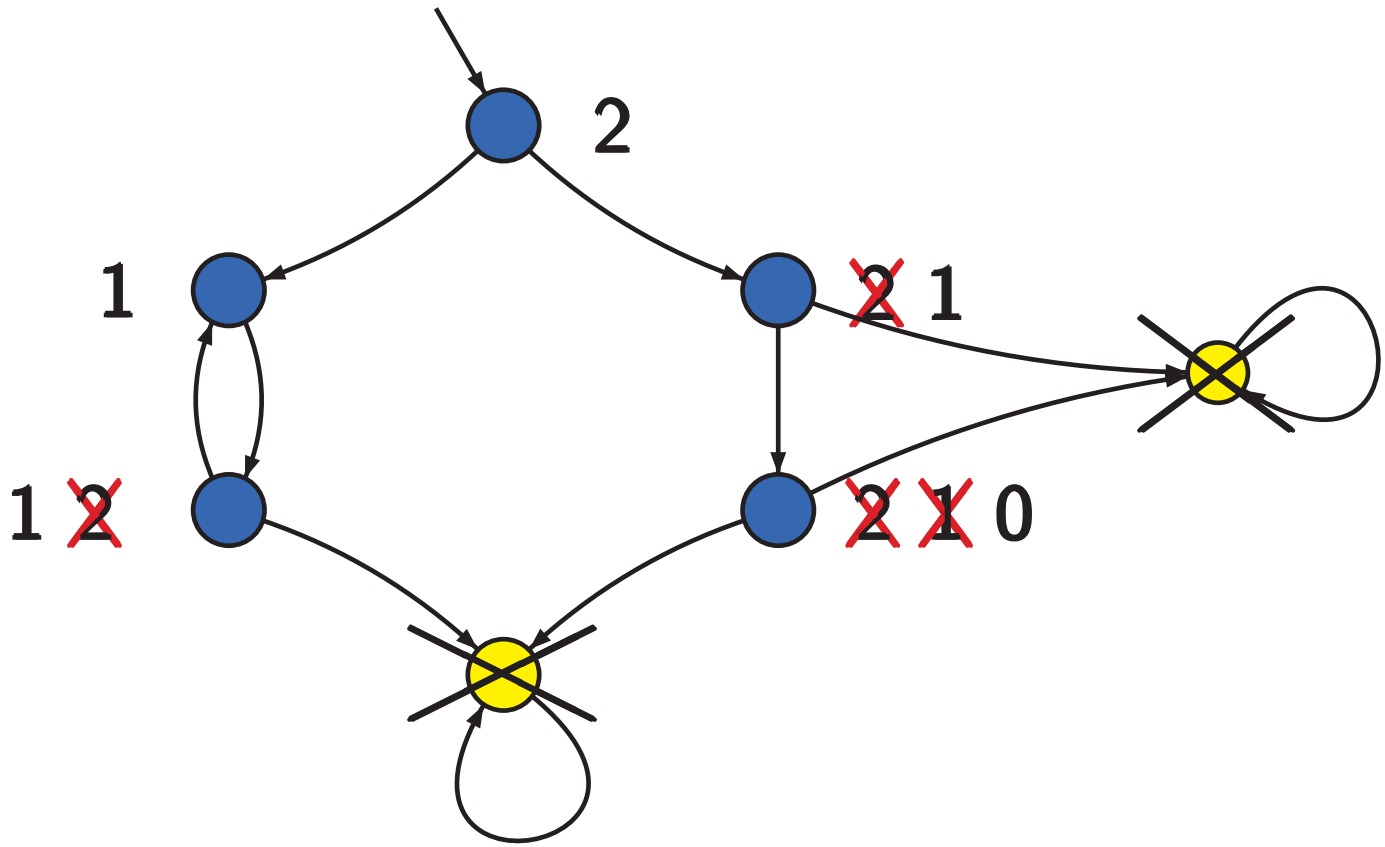
Example: CTL model checking for $\exists\Box$

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



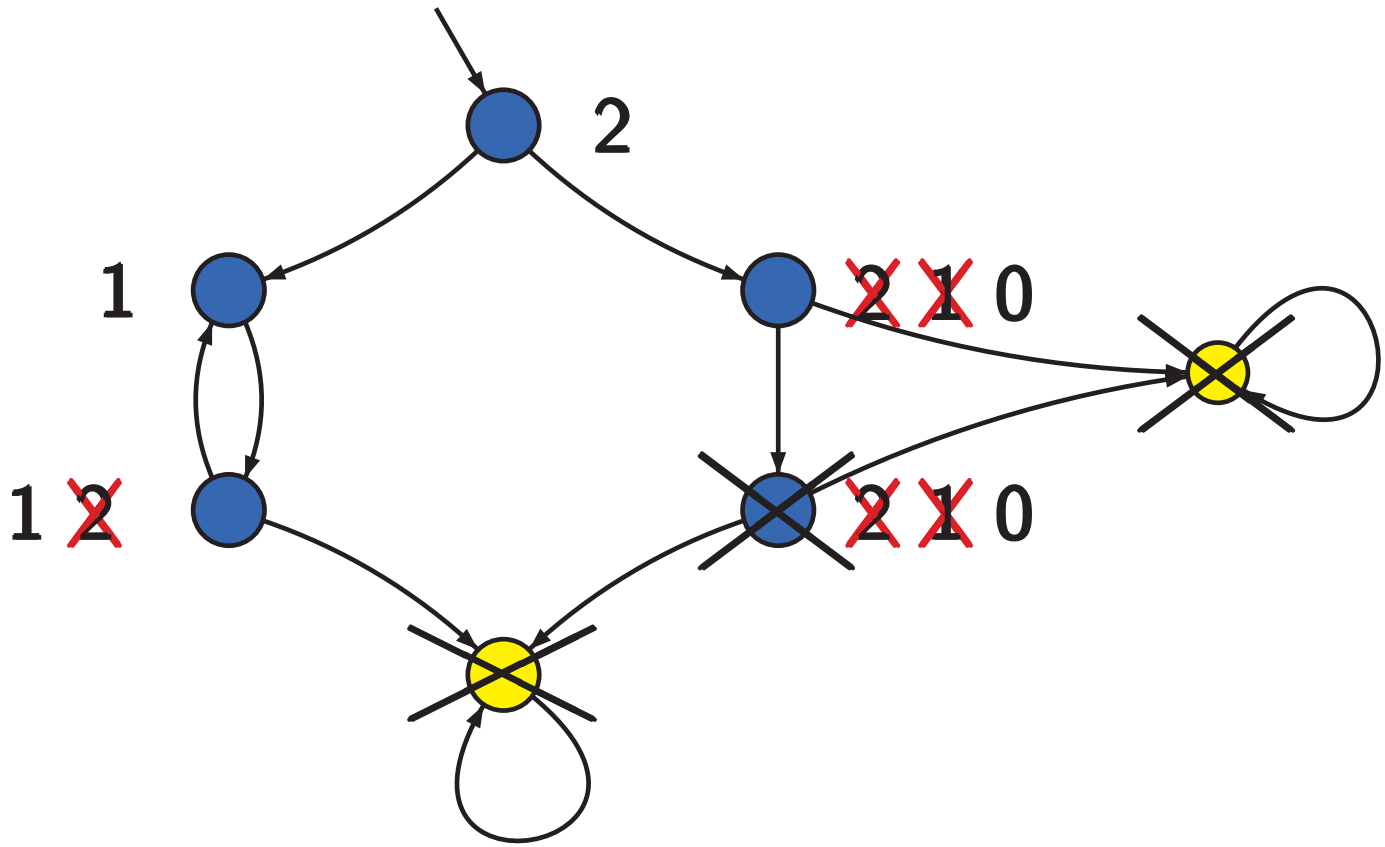
Example: CTL model checking for $\exists\Box$

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



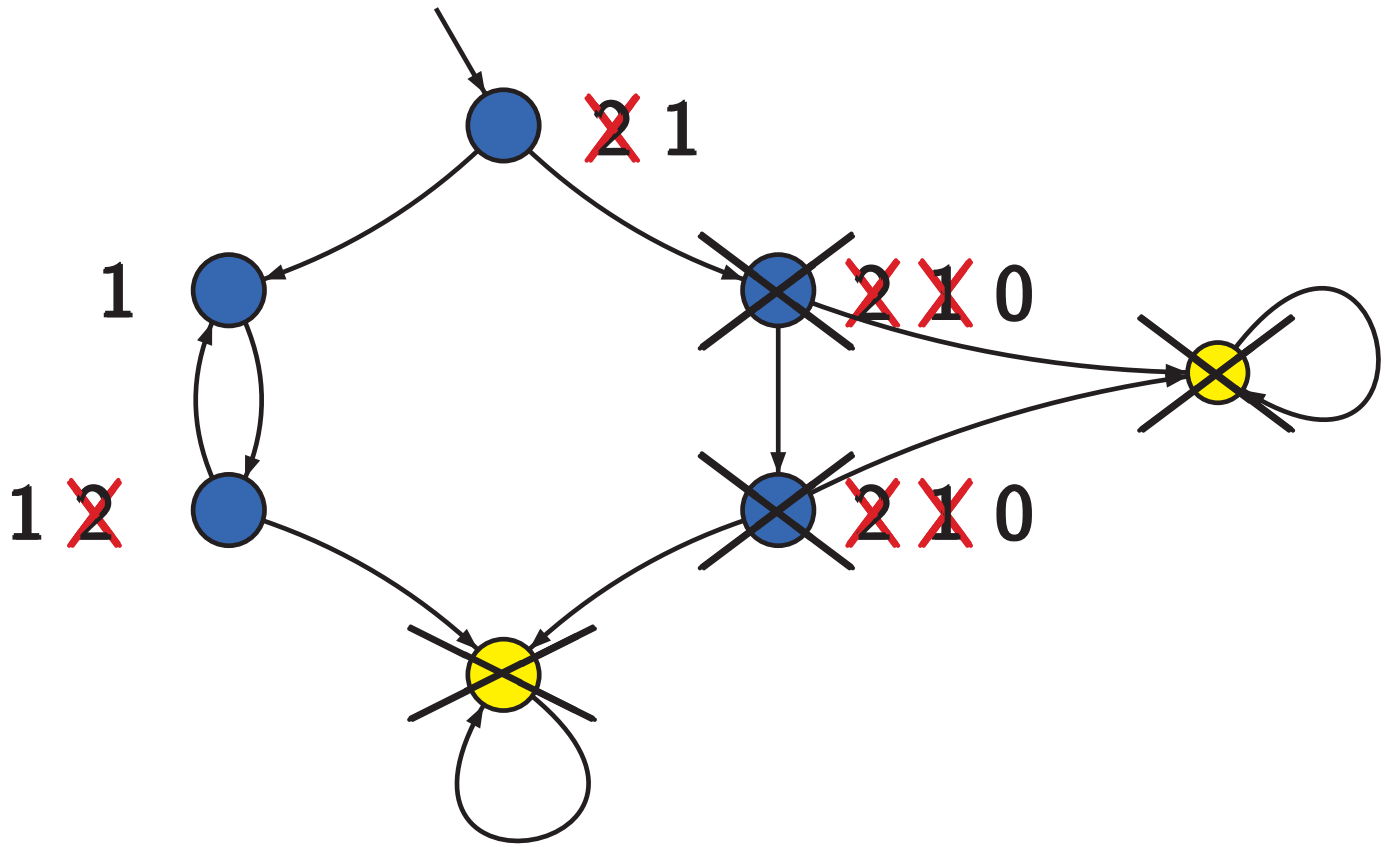
Example: CTL model checking for ∃□

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



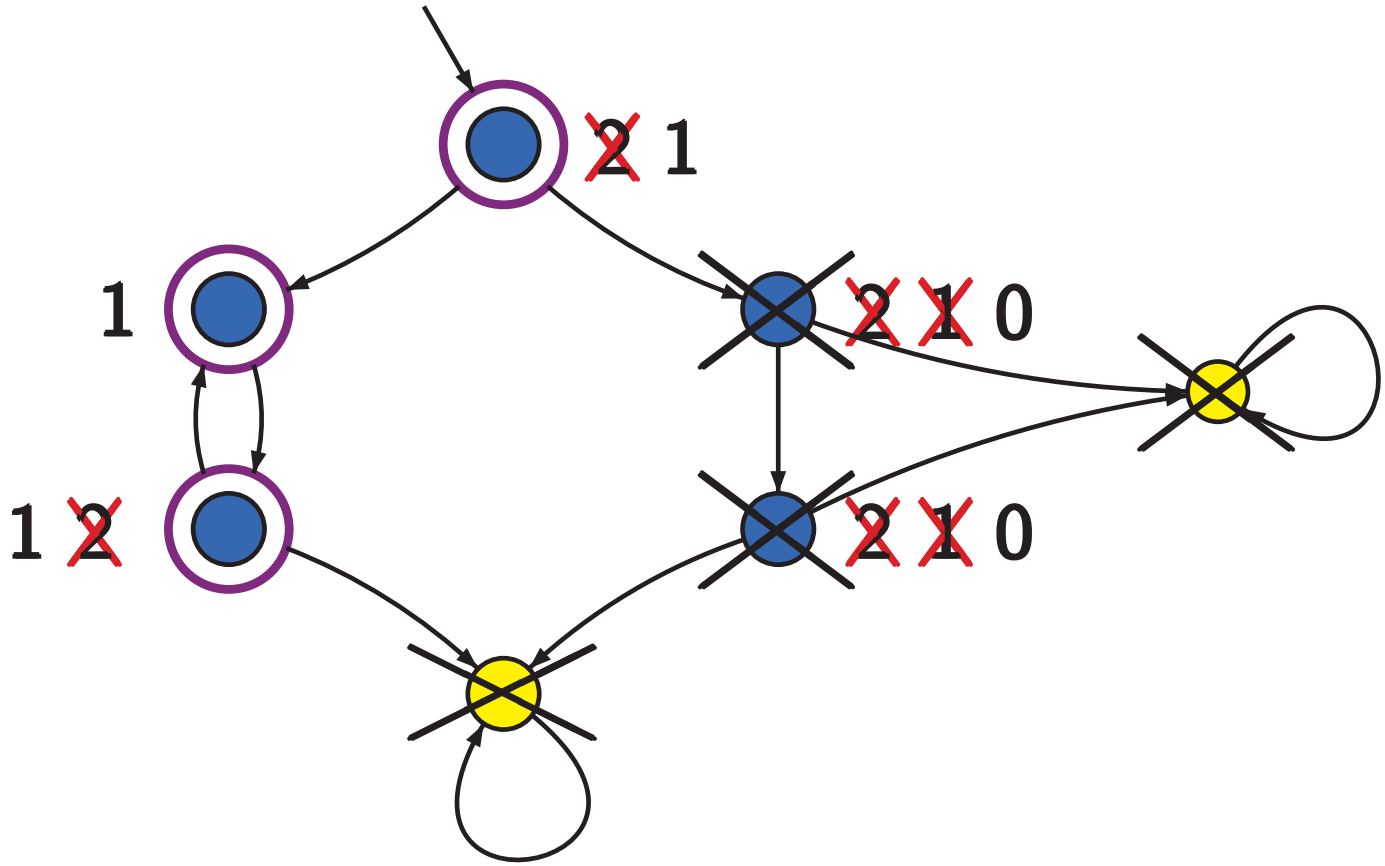
Example: CTL model checking for ∃□

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



Example: CTL model checking for ∃□

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



Recursive computation of satisfaction sets

case Φ is

true: return S

$a \in \mathit{AP}$: return $\{s \in S : a \in L(s)\}$

$\neg\Phi$: return $S \setminus \mathit{Sat}(\Phi)$

$\Phi_1 \wedge \Phi_2$: return $\mathit{Sat}(\Phi_1) \cap \mathit{Sat}(\Phi_2)$

$\exists\bigcirc\Phi$: return $\{s \in S : \mathit{Post}(s) \cap \mathit{Sat}(\Phi) \neq \varnothing\}$

$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$: ...

$\exists\Box\Phi$: ...

Recursive computation of satisfaction sets

case Φ is

$true$: return S

$a \in AP$: return $\{s \in S : a \in L(s)\}$

$\neg\Phi$: return $S \setminus Sat(\Phi)$

$\Phi_1 \wedge \Phi_2$: return $Sat(\Phi_1) \cap Sat(\Phi_2)$

$\exists\bigcirc\Phi$: return $\{s \in S : Post(s) \cap Sat(\Phi) \neq \varnothing\}$

$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$: ...

$\exists\Box\Phi$: ...

time complexity: ?

Recursive computation of satisfaction sets

`case` Φ `is`

true: return S

$a \in AP$: return $\{s \in S : a \in L(s)\}$

$\neg\Phi$: return $S \setminus \mathit{Sat}(\Phi)$

$\Phi_1 \wedge \Phi_2$: return $\mathit{Sat}(\Phi_1) \cap \mathit{Sat}(\Phi_2)$

$\exists\bigcirc\Phi$: return $\{s \in S : \mathit{Post}(s) \cap \mathit{Sat}(\Phi) \neq \varnothing\}$

$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$: ... $\longleftarrow$ complexity $\mathcal{O}(\mathit{size}(\mathcal{T}))$

$\exists\Box\Phi$: ... $\longleftarrow$ complexity $\mathcal{O}(\mathit{size}(\mathcal{T}))$

time complexity: ?

Recursive computation of satisfaction sets

case Φ is

true:	return S
$a \in \mathit{AP}$:	return $\{s \in S : a \in L(s)\}$
$\neg\Phi$:	return $S \setminus \mathit{Sat}(\Phi)$
$\Phi_1 \wedge \Phi_2$:	return $\mathit{Sat}(\Phi_1) \cap \mathit{Sat}(\Phi_2)$
$\exists\bigcirc\Phi$:	return $\{s \in S : \mathit{Post}(s) \cap \mathit{Sat}(\Phi) \neq \varnothing\}$
$\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$:	... $\longleftarrow$ complexity $\mathcal{O}(\mathit{size}(\mathcal{T}))$
$\exists\Box\Phi$:	... $\longleftarrow$ complexity $\mathcal{O}(\mathit{size}(\mathcal{T}))$

time complexity: $\mathcal{O}(\mathit{size}(\mathcal{T}) \cdot |\Phi|)$

Recursive computation of $Sat(\ldots)$

$$Sat(\Phi_1 \wedge \Phi_2) = Sat(\Phi_1) \cap Sat(\Phi_2)$$

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

$$Sat(\exists\bigcirc\Phi) = \{s \in S : Post(s) \cap Sat(\Phi) = \varnothing\}$$

$$Sat(\exists(\Phi_1 \,\mathsf{U}\, \Phi_2)) = \bigcup_{n\geq 0} T_n \text{ where}$$

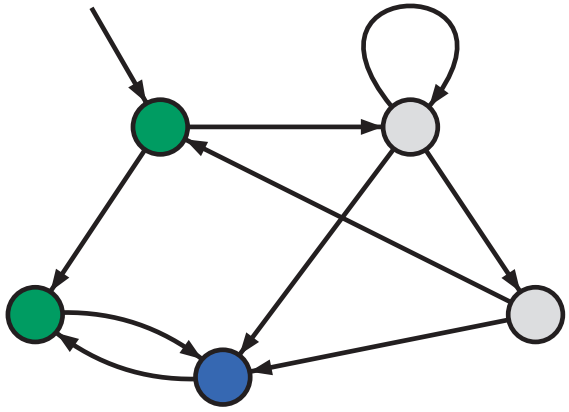
$$T_0 = Sat(\Phi_2)$$

$$T_{n+1} = \{s \in Sat(\Phi_1) : Post(s) \cap T_n \neq \varnothing\}$$

$$Sat(\exists\Box\Phi) = \bigcap_{n\geq 0} V_n \text{ where}$$

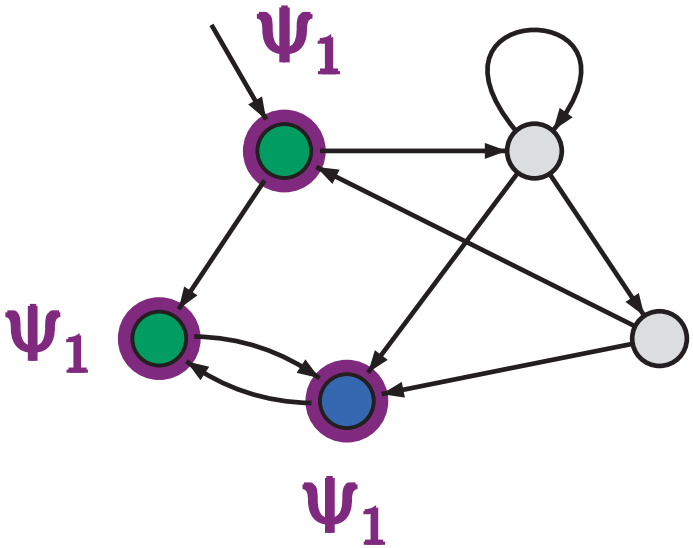
$$V_0 = Sat(\Phi);\quad V_{n+1} = \{s \in V_n : Post(s) \cap V_n \neq \varnothing\}$$

Example: CTL model checking

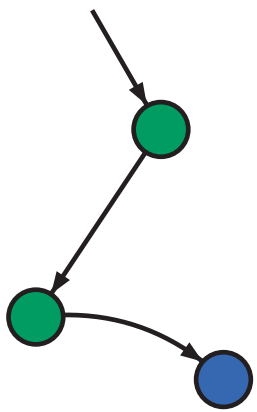


$$\Phi = \exists\Diamond\neg(\,\exists(a\,\mathsf{U}\,b)\ \vee\ \exists\Box\neg a\,)$$

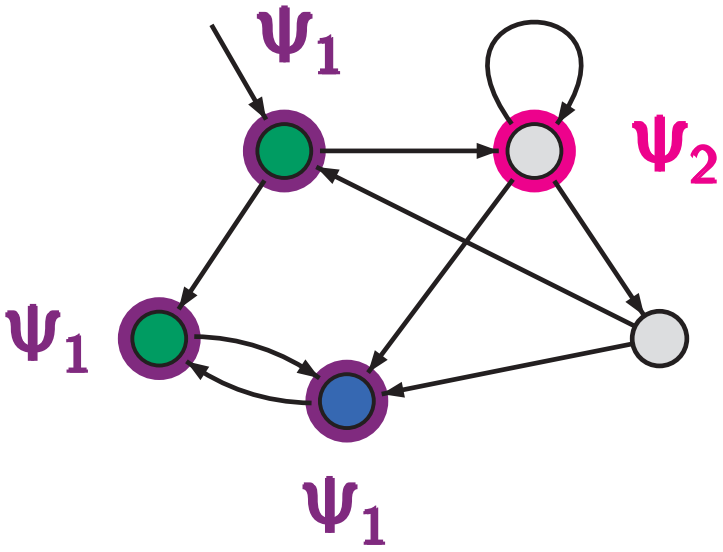
Example: CTL model checking



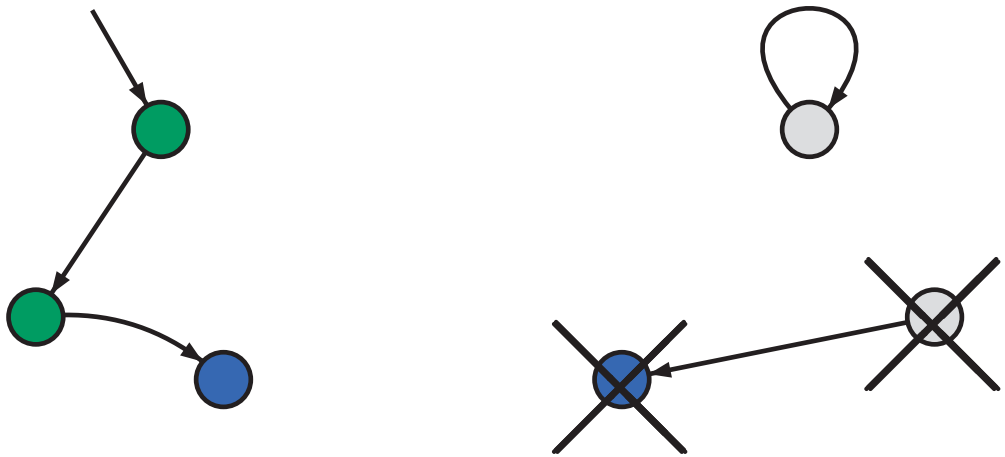
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \,\mathsf{U}\, b)}_{\psi_1} \vee \exists\Box\neg a)$$



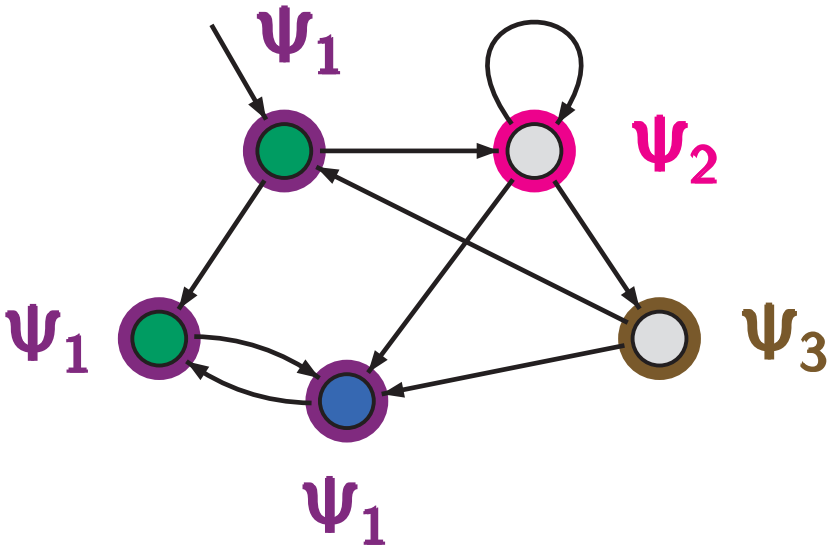
Example: CTL model checking



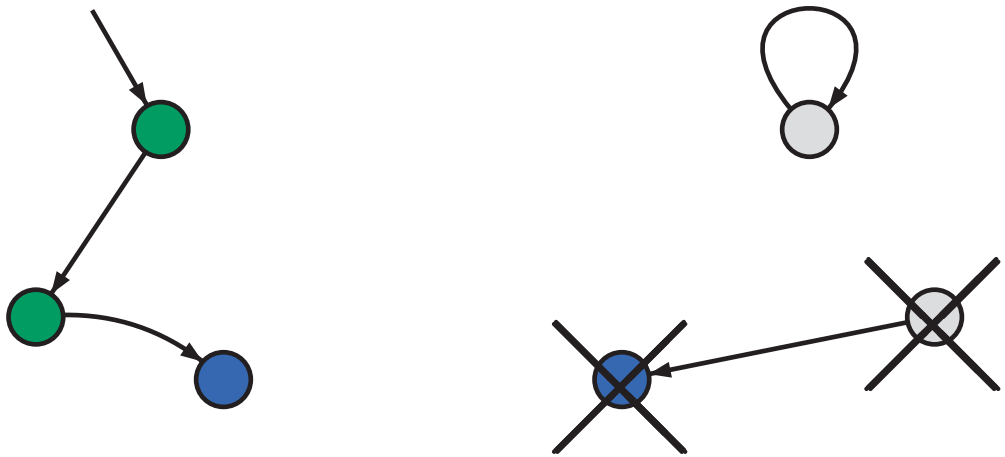
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \,\mathsf{U}\, b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2})$$



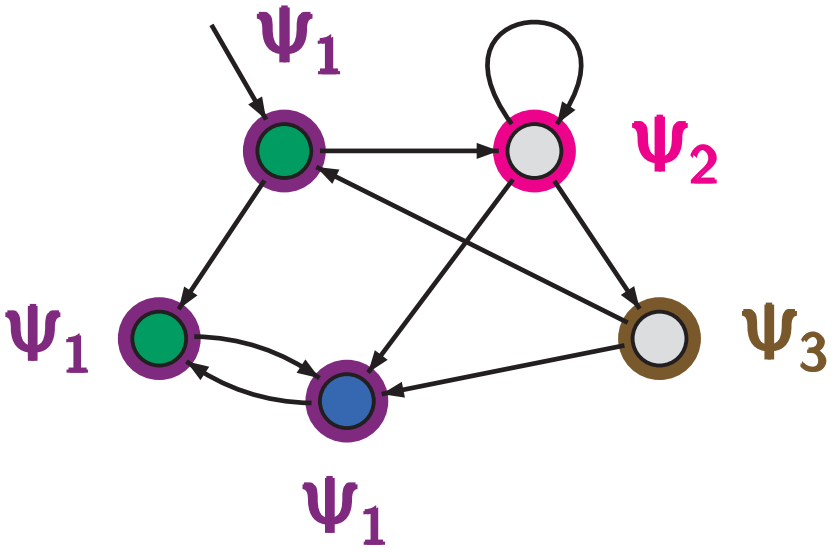
Example: CTL model checking



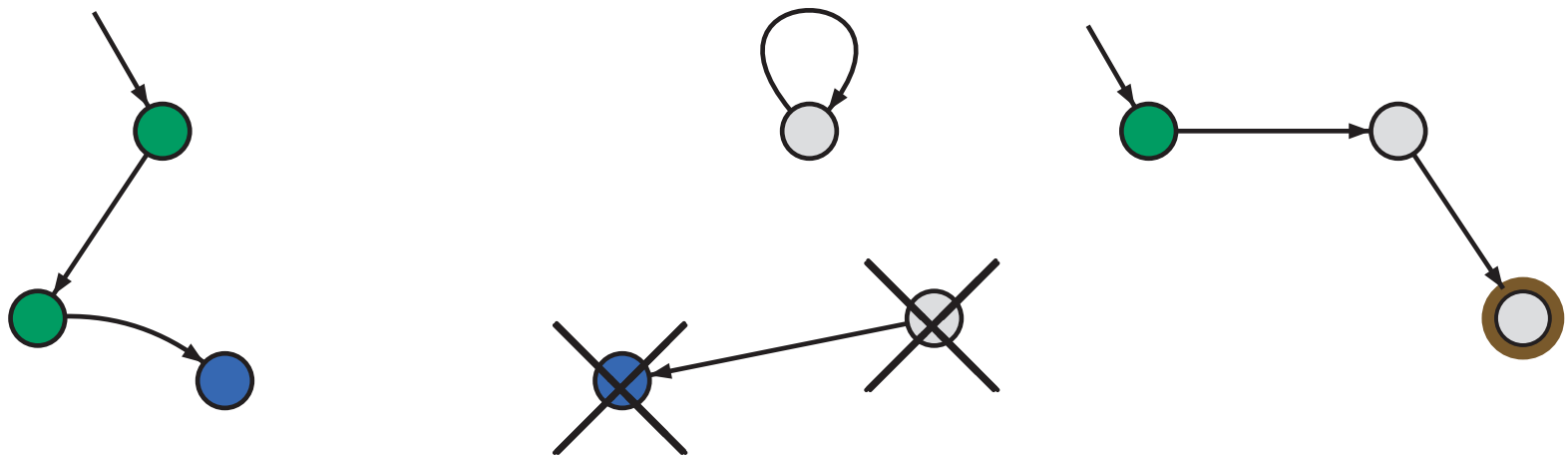
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a\,\mathsf{U}\,b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2}) = \exists\Diamond\underbrace{\neg(\psi_1 \vee \psi_2)}_{\psi_3}$$



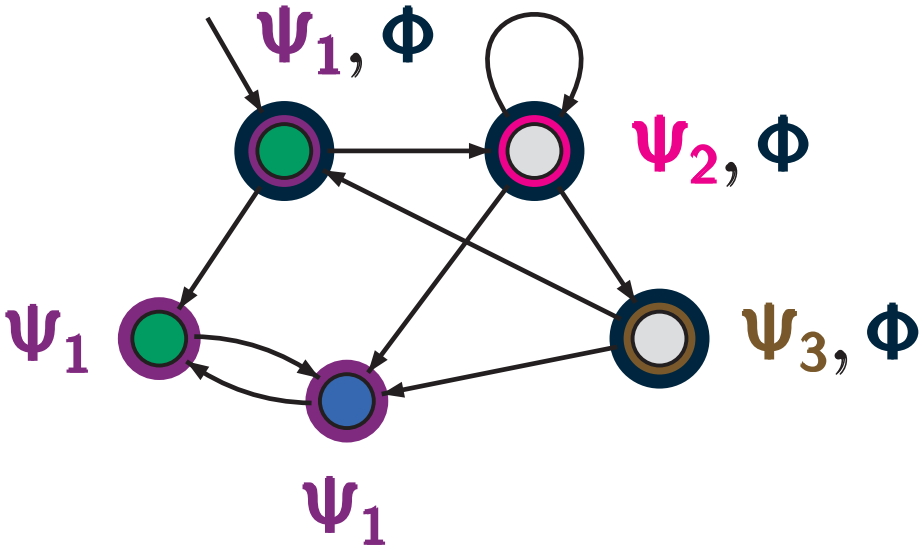
Example: CTL model checking



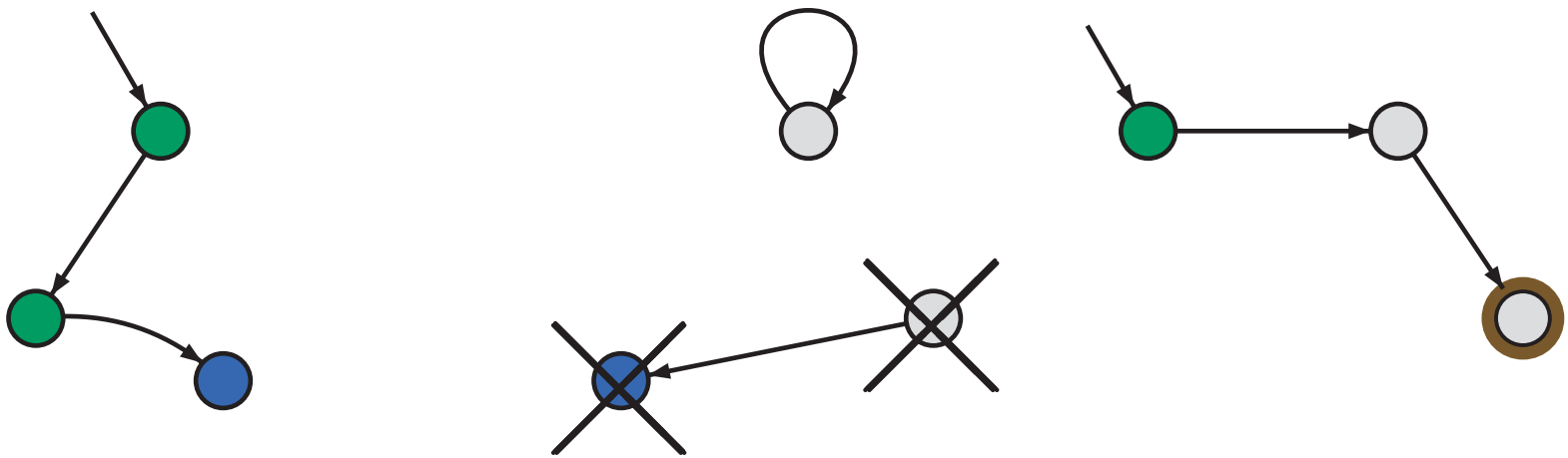
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a\,\mathsf{U}\,b)}_{\psi_1} \vee \underbrace{\exists\Box\neg a}_{\psi_2}) = \exists\Diamond\underbrace{\neg(\psi_1 \vee \psi_2)}_{\psi_3}$$



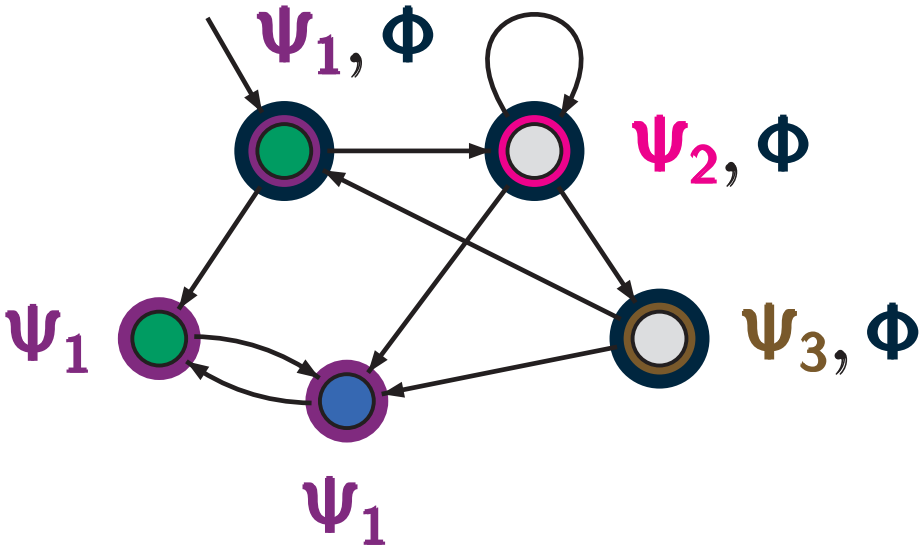
Example: CTL model checking



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

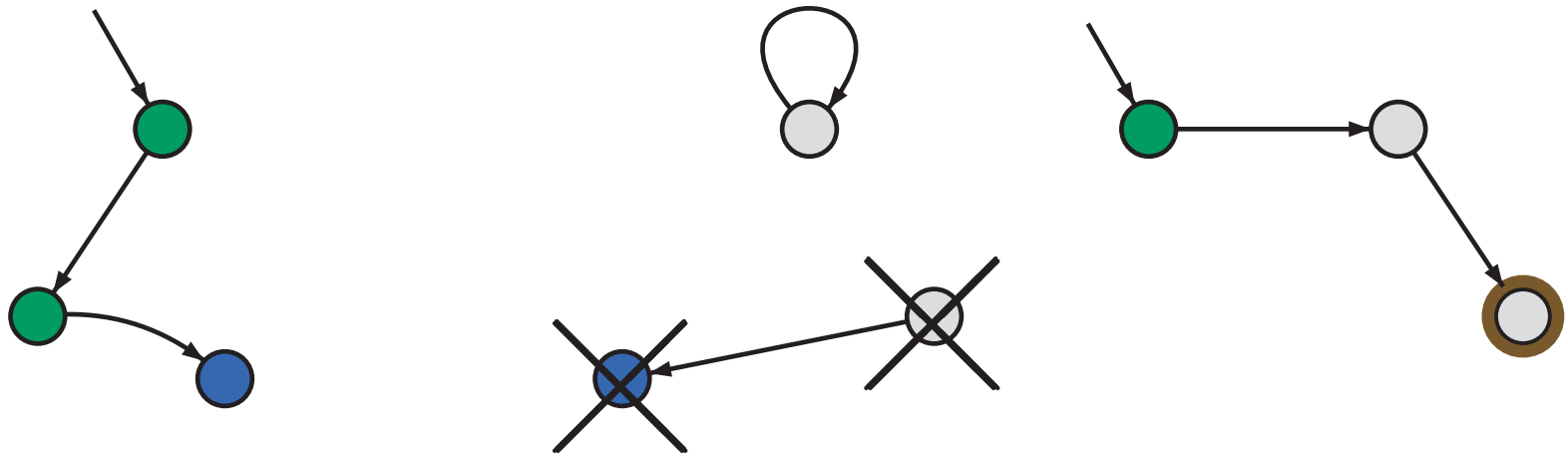


Example: CTL model checking



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

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



Complexity of CTL model checking

Complexity of CTL model checking

CTL model checking: $\mathcal{O}(\mathit{size}(\mathcal{T}) \cdot |\Phi|)$

Complexity of CTL/LTL model checking

CTL model checking: $\mathcal{O}\big(\mathit{size}(\mathcal{T}) \cdot |\Phi|\big)$

LTL model checking: $\mathcal{O}\big(\mathit{size}(\mathcal{T}) \cdot \mathbf{exp}(|\varphi|)\big)$

Complexity of CTL/LTL model checking

CTL model checking: $\mathcal{O}(\mathit{size}(\mathcal{T}) \cdot |\Phi|)$

LTL model checking: $\mathcal{O}(\mathit{size}(\mathcal{T}) \cdot \mathbf{exp}(|\varphi|))$

model complexity, i.e., for fixed specification:

CTL and **LTL**: $\mathcal{O}(\mathit{size}(\mathcal{T}))$