CTL Model Checking

Luca Tesei

Reactive Systems Verification

MSc in Computer Science

University of Camerino

Topics

- Recursive definition of the Sat set
- Minimum fixpoint based algorithm to calculate the Sat of Exist Until
- Maximum fixpoint based algorithm to calculate the Sat of Exist Box
- Examples
- Complexity of CTL model checking

Material

Reading:

Chapter 6 of the book: Section 6.4

More:

The slides in the following pages are taken from the material of the course "Introduction to Model Checking" held by Prof. Dr. Ir. Joost-Pieter Katoen at Aachen University.

Overview

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
```

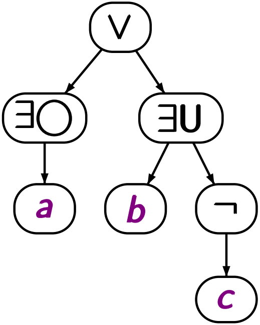
Example: CTL model checking

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

Example: CTL model checking

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

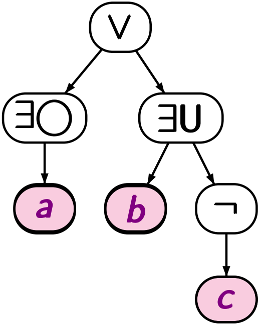
syntax tree for Φ



Example: CTL model checking

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

syntax tree for Φ



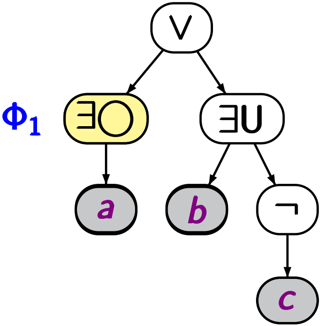
processed in
bottom-up fashion

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

Example: CTL model checking

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

syntax tree for Φ



processed in
bottom-up fashion

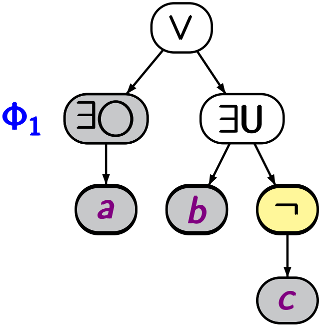
compute $Sat(a), Sat(b), Sat(c)$

$Sat(\Phi_1) = \ldots$

Example: CTL model checking

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

syntax tree for Φ



processed in
bottom-up fashion

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

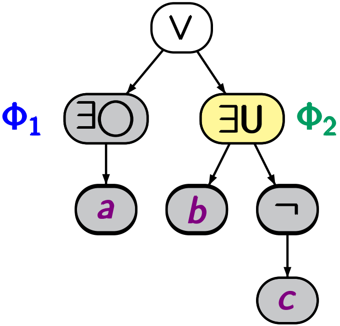
$Sat(\Phi_1) = \ldots$

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

Example: CTL model checking

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

syntax tree for Φ



processed in bottom-up fashion

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

$Sat(\Phi_1) = \ldots$

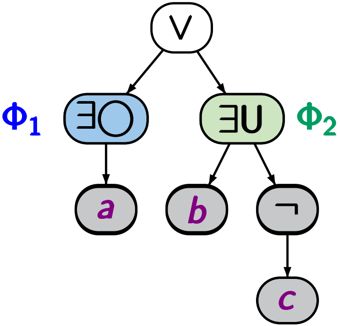
$Sat(\neg c) = S \setminus Sat(c)$

$Sat(\Phi_2) = \ldots$

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



processed in
bottom-up fashion

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

$Sat(\Phi_1) = \ldots$

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

$Sat(\Phi_2) = \ldots$

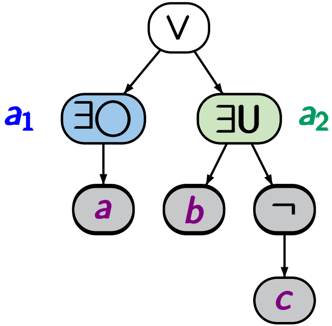
replace Φ_1 with a_1

replace Φ_2 with a_2

Example: CTL 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 $Sat(a), Sat(b), Sat(c)$

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

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

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

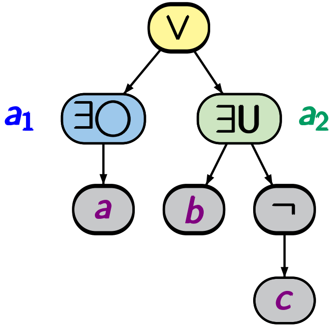
replace Φ_1 with a_1

replace Φ_2 with a_2

Example: CTL model checking

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

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 **∃-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 $\exists\bigcirc$, $\exists \mathsf{U}$, $\exists\Box$.

CTL formulas in ∃-normal form:

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

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 $\exists\bigcirc$, $\exists \mathsf{U}$, $\exists\Box$.

CTL formulas in ∃-normal form:

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

CTL formula ⇝ **CTL** formula in ∃-normal form

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

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

Recursive computation of the satisfaction sets

Recursive computation of the satisfaction sets

$$Sat(\mathit{true}) = S$$

Recursive computation of the satisfaction sets

$$Sat(true) = S$$

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

Recursive computation of the satisfaction sets

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

Recursive computation of the satisfaction sets

$$Sat(true) = S$$

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

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

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

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

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

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

via fixed point computation

Recall: expansion law for ∃U

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

Recall: expansion law for $\exists U$

$$\exists(\Phi_1 \, U \, \Phi_2) \;\equiv\; \Phi_2 \vee \left(\Phi_1 \wedge \exists\bigcirc\exists(\Phi_1 \, U \, \Phi_2)\right)$$

$$Sat(\exists(\Phi_1 \, U \, \Phi_2)) = Sat(\Phi_2) \cup$$
$$\left\{ s \in Sat(\Phi_1) : Post(s) \cap Sat(\exists(\Phi_1 \, U \, \Phi_2)) \neq \varnothing \right\}$$

Fixed point characterization of $\exists U$

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

$$Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)) = Sat(\Phi_2) \cup \\ \big\{ s \in Sat(\Phi_1) : Post(s) \cap Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)) \neq \varnothing \big\}$$

i.e., the set $T = Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2))$ is a **fixed point** of the higher-order function $\Omega : 2^S \to 2^S$ given by:

$$\Omega(T) \;=\; Sat(\Phi_2) \cup \big\{ s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing \big\}$$

Fixed point characterization of $\exists U$

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

$$Sat(\exists(\Phi_1 \mathsf{U} \Phi_2)) = Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap Sat(\exists(\Phi_1 \mathsf{U} \Phi_2)) \neq \varnothing\}$$

satisfies the following conditions:

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

(2) If $s \in Sat(\Phi_1)$ and $Post(s) \cap Sat(\exists(\Phi_1 \mathsf{U} \Phi_2)) \neq \varnothing$ then $s \in Sat(\exists(\Phi_1 \mathsf{U} \Phi_2))$

Least fixed point characterization of ∃U

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

$$Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)) = Sat(\Phi_2) \cup$$
$$\big\{ s \in Sat(\Phi_1) : Post(s) \cap Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)) \neq \varnothing \big\}$$

satisfies the following conditions:

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

(2) If $s \in Sat(\Phi_1)$ and $Post(s) \cap Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2)) \neq \varnothing$ then $s \in Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2))$

$Sat(\exists(\Phi_1 \mathsf{U} \Phi_2))$ is the smallest set s.t. (1) and (2) hold

The always operator

The always operator

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

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

Greatest fixed point characterization of $\exists\Box$

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

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

i.e., $Sat(\exists\Box\Phi)$ is the greatest fixed point of the operator $\Omega : 2^S \to 2^S$ given by:

$$\Omega(V) = \{s \in Sat(\Phi) : Post(s) \cap V \neq \varnothing\}$$

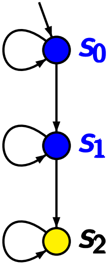
Greatest fixed point characterization of $\exists\Box$

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

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

i.e., $Sat(\exists\Box\Phi)$ is the greatest fixed point of the operator $\Omega : 2^S \rightarrow 2^S$ given by:

$$\Omega(V) \; = \; \{ s \in Sat(\Phi) : Post(s) \cap V \neq \varnothing \}$$



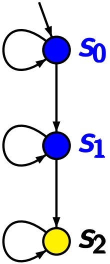
Greatest fixed point characterization of $\exists\Box$

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

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

i.e., $Sat(\exists\Box\Phi)$ is the greatest fixed point of the operator $\Omega : 2^S \to 2^S$ given by:

$$\Omega(V) = \{ s \in Sat(\Phi) : Post(s) \cap V \neq \varnothing \}$$



$V = \{s_0\}$ satisfies $(*)$

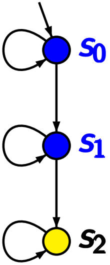
Greatest fixed point characterization of $\exists\Box$

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

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

i.e., $Sat(\exists\Box\Phi)$ is the greatest fixed point of the operator $\Omega : 2^S \to 2^S$ given by:

$$\Omega(V) \; = \; \{ s \in Sat(\Phi) : Post(s) \cap V \neq \varnothing \}$$



$V = \{s_0\}$ satisfies (*)

$V \subsetneq Sat(\exists\Box a) = \{s_0, s_1\}$

Until versus weak until

Until versus weak until

The formulas $\Psi = \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$ and $\Psi = \exists(\Phi_1 \,\mathsf{W}\, \Phi_2)$ fulfill the expansion law

$$\Psi \equiv \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\Psi)$$

Until versus weak until

The formulas $\Psi = \exists(\Phi_1 \,\mathsf{U}\, \Phi_2)$ and $\Psi = \exists(\Phi_1 \,\mathsf{W}\, \Phi_2)$ fulfill the expansion law

$$\Psi \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists\bigcirc\Psi)$$

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

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

Until versus weak until

The formulas $\Psi = \exists(\Phi_1 \, \mathsf{U} \, \Phi_2)$ and $\Psi = \exists(\Phi_1 \, \mathsf{W} \, \Phi_2)$ fulfill the expansion law

$$\Psi \;\equiv\; \Phi_2 \vee (\Phi_1 \wedge \exists \bigcirc \Psi)$$

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

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

weak until: $Sat(\exists(\Phi_1 \, \mathsf{W} \, \Phi_2))$ = greatest set V s.t.

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

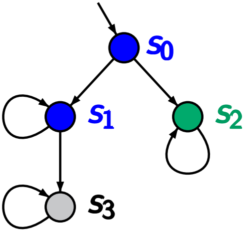
Fixed point equations for $\exists$U

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

$$(*) \quad Sat(b) \cup \{s \in Sat(a) : Post(s) \cap T \neq \varnothing\} \subseteq T$$

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

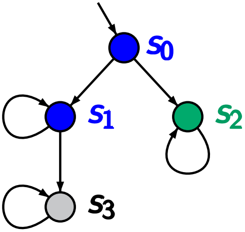
$$(*) \quad Sat(b) \cup \{ s \in Sat(a) : Post(s) \cap T \neq \varnothing \} \subseteq T$$



Fixed point equations for ∃U and ∃W

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

$(*) \quad Sat(b) \cup \{s \in Sat(a) : Post(s) \cap T \neq \varnothing\} \subseteq T$

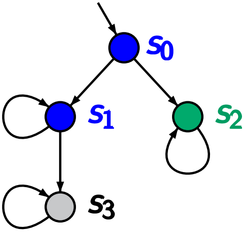


$T = \{s_0, s_1, s_2\}$ satisfies $(*)$

Fixed point equations for ∃U and ∃W

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

$$(*) \quad Sat(b) \cup \{s \in Sat(a) : Post(s) \cap T \neq \varnothing\} \subseteq T$$



$T = \{s_0, s_1, s_2\}$ satisfies $(*)$

$Sat(\exists(a \,\mathsf{U}\, b)) = \{s_0, s_2\} \subsetneq T$

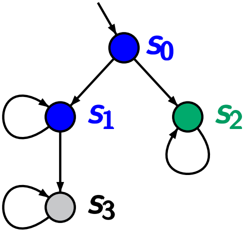
Fixed point equations for ∃U and ∃W

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

$$(*) \quad Sat(b) \cup \{ s \in Sat(a) : Post(s) \cap T \neq \varnothing \} \subseteq T$$

$Sat(\exists(a \,\mathsf{W}\, b))$ = greatest set of states V s.t.

$$(**) \quad V \subseteq Sat(b) \cup \{ s \in Sat(a) : Post(s) \cap V \neq \varnothing \}$$



$T = \{s_0, s_1, s_2\}$ satisfies $(*)$

$Sat(\exists(a \,\mathsf{U}\, b)) = \{s_0, s_2\} \subsetneq T$

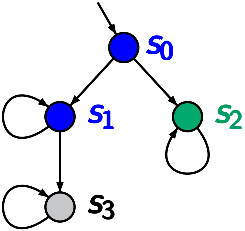
Fixed point equations for ∃U and ∃W

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

$$(*) \quad Sat(b) \cup \{s \in Sat(a) : Post(s) \cap T \neq \varnothing\} \subseteq T$$

$Sat(\exists(a \, \mathsf{W} \, b))$ = greatest set of states V s.t.

$$(**) \quad V \subseteq Sat(b) \cup \{s \in Sat(a) : Post(s) \cap V \neq \varnothing\}$$



$T = \{s_0, s_1, s_2\}$ satisfies $(*)$

$Sat(\exists(a \, \mathsf{U} \, b)) = \{s_0, s_2\} \subsetneq T$

$V = \{s_0, s_2\}$ satisfies $(**)$

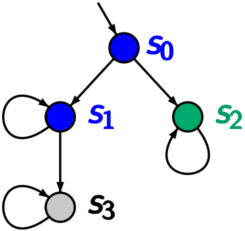
Fixed point equations for ∃U and ∃W

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

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

$Sat(\exists(a \mathsf{W} b))$ = greatest set of states V s.t.

(**) $V \subseteq Sat(b) \cup \{s \in Sat(a) : Post(s) \cap V \neq \varnothing\}$



$T = \{s_0, s_1, s_2\}$ satisfies (*)

$Sat(\exists(a \mathsf{U} b)) = \{s_0, s_2\} \subsetneq T$

$V = \{s_0, s_2\}$ satisfies (**), but

$V \subsetneq Sat(\exists(a \mathsf{W} b)) = \{s_0, s_1, s_2\}$

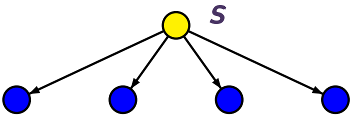
Universally quantified formulas

Universally quantified formulas

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

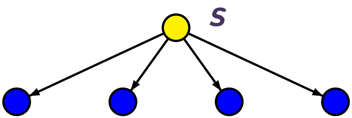
Universally quantified formulas

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



Universally quantified formulas

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

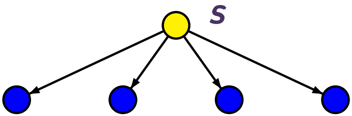


$Sat(\forall\Box a)$ = greatest set T of states s.t.

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

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*(...)

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

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

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

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

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

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

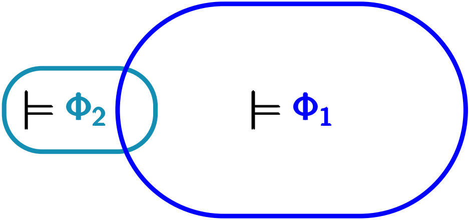
$$Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing\} \subseteq T$$

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.

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



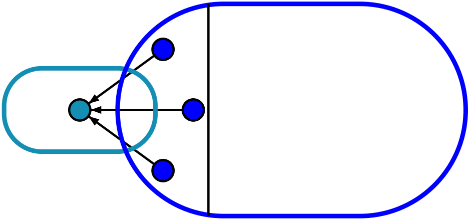
$$T_0 := Sat(\Phi_2)$$

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.

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



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

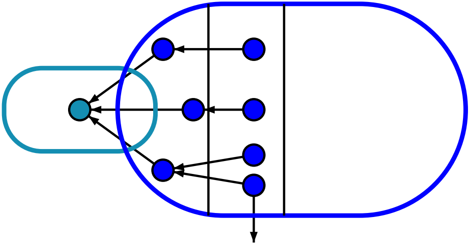
$$T_{n+1} := T_n \cup \{s \in Sat(\Phi_1) : Post(s) \cap T_n \neq \varnothing\}$$

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.

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



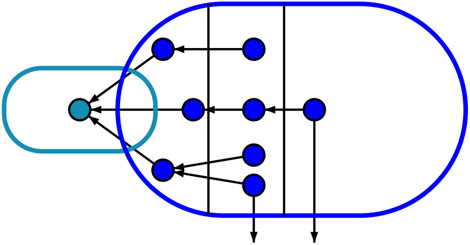
$$T_0 := Sat(\Phi_2)$$

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

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

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

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



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

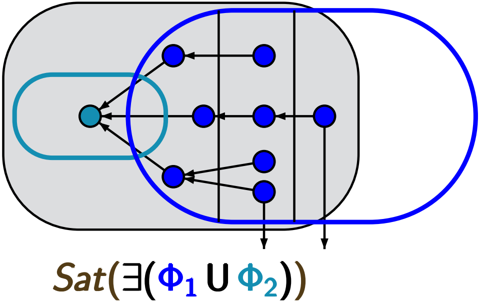
$$T_{n+1} := T_n \cup \{s \in Sat(\Phi_1) : Post(s) \cap T_n \neq \varnothing\}$$

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.

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

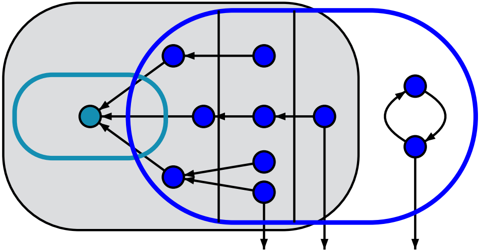


CTL model checking: until operator

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

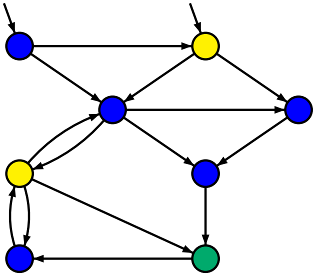
$Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2))$ = least set T of states s.t.

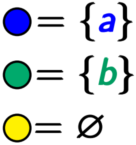
$$Sat(\Phi_2) \cup \{ s \in Sat(\Phi_1) : Post(s) \cap T \neq \varnothing \} \subseteq T$$



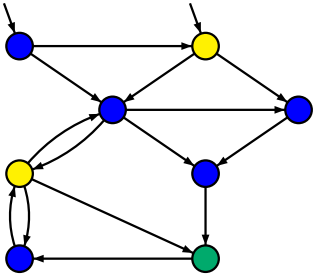
$Sat(\exists(\Phi_1 \, \mathsf{U} \, \Phi_2))$

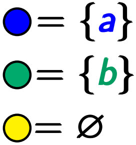
Example: until operator





Example: until operator





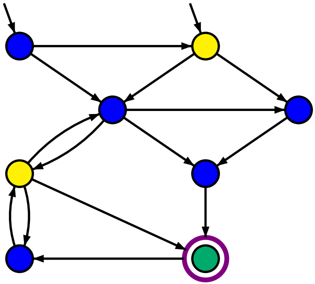
computation of $Sat(\exists(a \,\mathsf{U}\, b))$

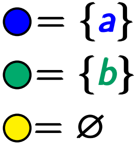
add all states $s \in Sat(b)$ to T

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





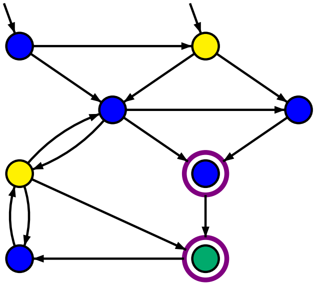
computation of $Sat(\exists(a \mathsf{U} b))$

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



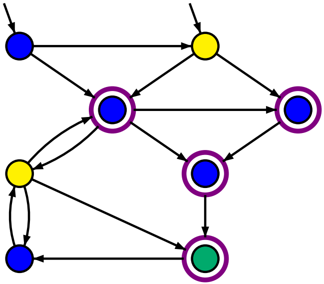
$\bullet = \{a\}$ (blue)
$\bullet = \{b\}$ (green)
$\bullet = \varnothing$ (yellow)

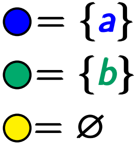
computation of $Sat(\exists(a \,\mathsf{U}\, b))$

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





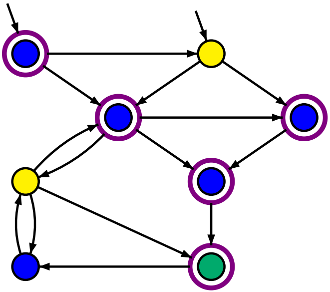
computation of $Sat(\exists(a \,\mathsf{U}\, b))$

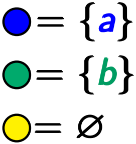
add all states $s \in Sat(b)$ to T

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





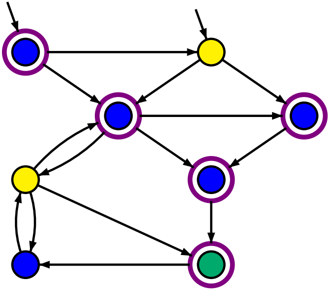
computation of $Sat(\exists(a \, \mathsf{U} \, b))$

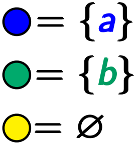
add all states $s \in Sat(b)$ to T

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





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

CTL model checking: treatment of $\exists$U

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

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

CTL model checking: treatment of $\exists U$

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

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

$E := Sat(\Phi_2)$ ⟵ 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 ≠ ∅ 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 ≠ ∅ DO
    select a state s' ∈ E and remove s' from E
```

CTL model checking: treatment of ∃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 $E \neq \varnothing$ DO

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

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

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

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

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

WHILE $E \neq \varnothing$ DO

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

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

        IF $s \in Sat(\Phi_1) \setminus T$ THEN add s to T and E FI

    OD

OD

CTL model checking: treatment of $\exists U$

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

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

$E := Sat(\Phi_2)$ ⟵ 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

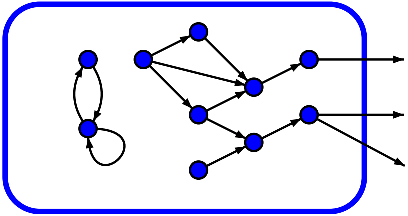
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

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

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

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

$Sat(\Phi)$



CTL model checking: always operator

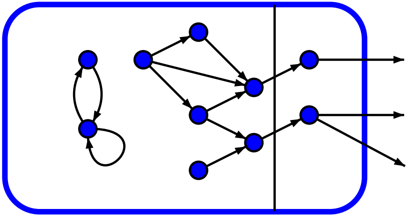
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

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

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

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

$Sat(\Phi)$



CTL model checking: always operator

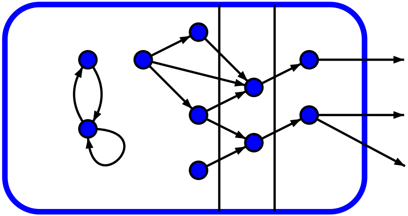
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

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

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

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

$Sat(\Phi)$



CTL model checking: always operator

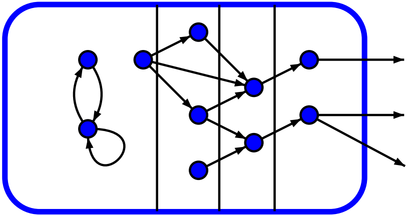
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

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

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

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

$Sat(\Phi)$



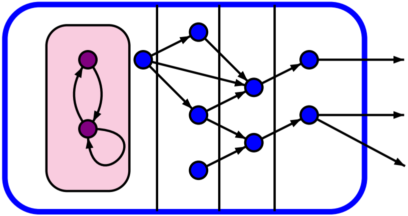
expansion law: $\exists\Box\Phi \equiv \Phi \wedge \exists\bigcirc\exists\Box\Phi$

$Sat(\exists\Box\Phi)$ = 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 $Sat(\exists\Box\Phi)$

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

$E := S \setminus T \quad \leftarrow$ 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 ≠ ∅ DO
   pick a state s' ∈ E and remove s' from E
   FOR ALL s ∈ 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
```

$T := Sat(\Phi)$ ← 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

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

$E := S \setminus T \leftarrow$ 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 $E \neq \varnothing$ DO
  pick a state $s' \in E$ and remove s' from E
  FOR ALL $s \in Pre(s')$ DO
   IF $s \in T$ and $Post(s) \cap (T \cup E) = \varnothing$ 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 ≠ ∅ 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
```

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

```
T := Sat(Φ); E := S \ T
FOR ALL  s ∈ Sat(Φ) DO  c[s] := |Post(s)| OD
```

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

```
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
```

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 $E \neq \varnothing$ DO
  pick a state $s' \in E$ and remove s' from E
  FOR ALL $s \in Pre(s')$ DO
    IF $s \in T$ and $Post(s) \cap (T \cup E) = \varnothing$ 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 $E \neq \varnothing$ DO

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

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

IF $s \in T$ and ~~$Post(s) \cap (T \cup E) = \varnothing$~~ 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 ≠ ∅ DO
   pick a state s′ ∈ E and remove s′ from E
   FOR ALL s ∈ Pre(s′) DO
    IF s ∈ 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

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

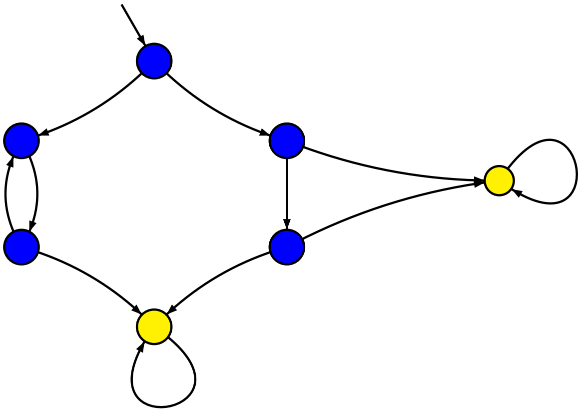
FOR ALL $s \in Sat(\Phi)$ DO $c[s] := |Post(s)|$ OD

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

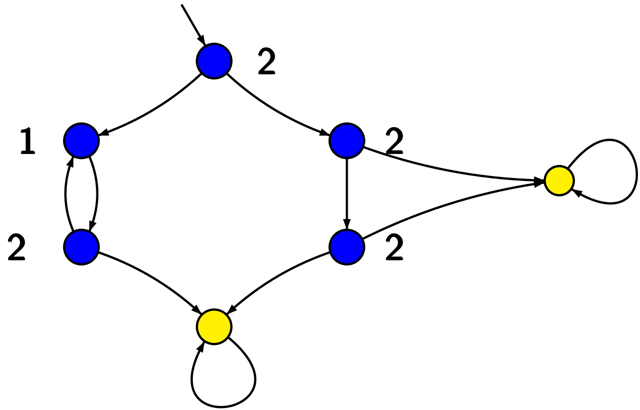
```
WHILE E ≠ ∅ DO
   pick a state s' ∈ E and remove s' from E
   FOR ALL s ∈ Pre(s') DO
    IF s ∈ T THEN
         c[s] := c[s] − 1
         IF c[s] = 0 THEN
                remove s from T and add s to E FI
    FI
OD
```

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

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

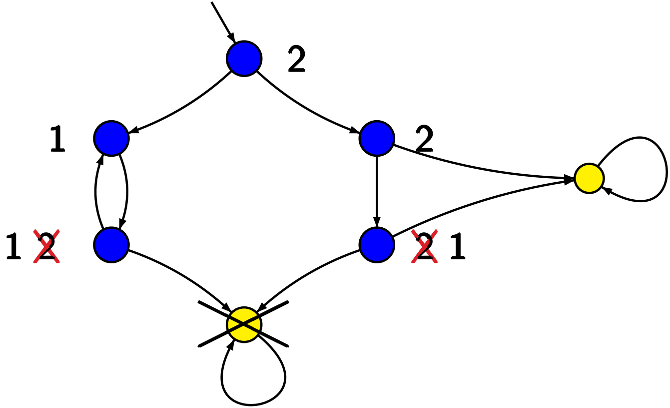


computation of $T = Sat(\exists\Box blue)$



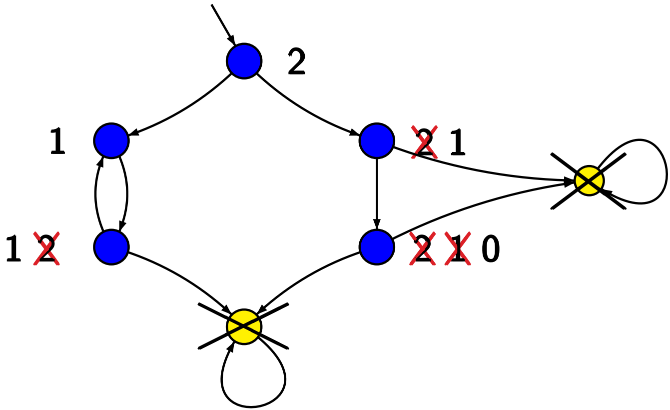
Example: CTL model checking for $\exists\Box$

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



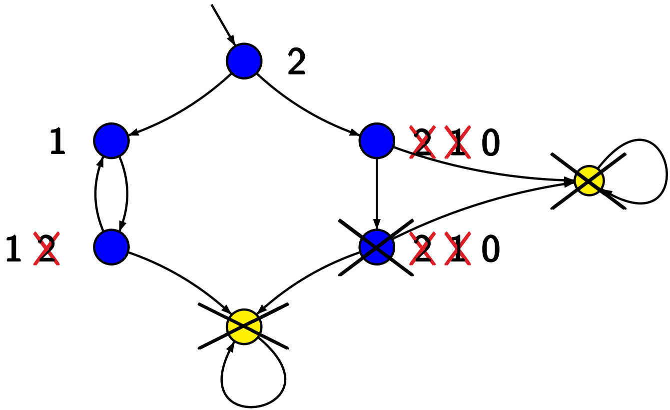
Example: CTL model checking for $\exists\Box$

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



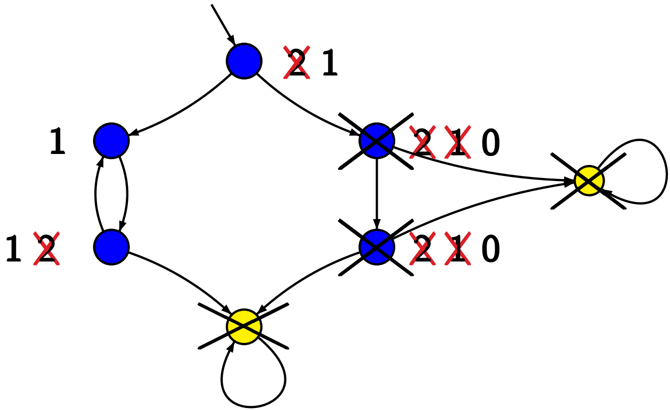
Example: CTL model checking for $\exists\Box$

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

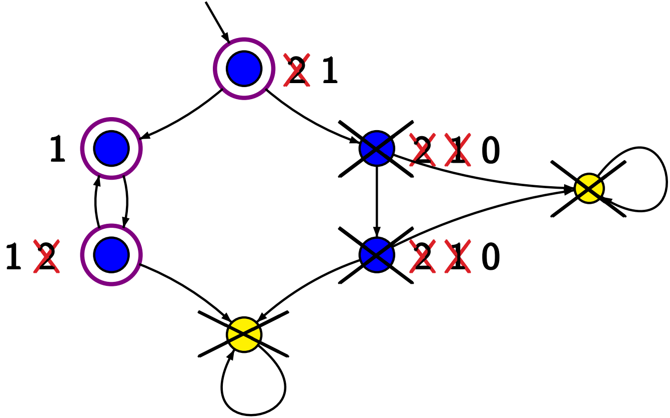


Example: CTL model checking for $\exists\Box$

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



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



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 $\{s \in S : \mathit{Post}(s) \cap \mathit{Sat}(\Phi) \neq \varnothing\}$
$\exists(\Phi_1 \mathsf{U} \Phi_2)$:	...
$\exists\Box\Phi$:	...

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 $\{s \in S : Post(s) \cap Sat(\Phi) \neq \varnothing\}$
$\exists(\Phi_1 \mathsf{U} \Phi_2)$:	...
$\exists\square\Phi$:	...

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

time complexity: ?

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

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

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

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

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

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

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

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

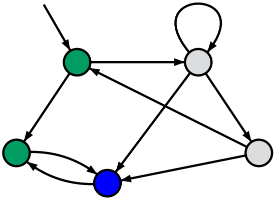
$$T_0 = Sat(\Phi_2)$$

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

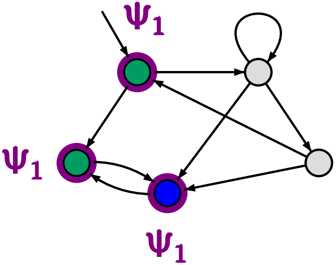
$$Sat(\exists\Box\Phi) = \bigcap_{n \geq 0} V_n \text{ where}$$

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

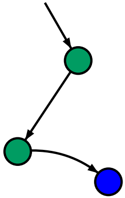
Example: CTL model checking



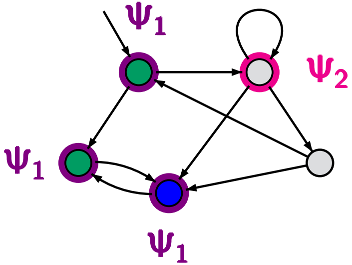
$$\Phi = \exists\Diamond\neg(\exists(a \, \mathsf{U} \, b) \vee \exists\Box\neg a)$$



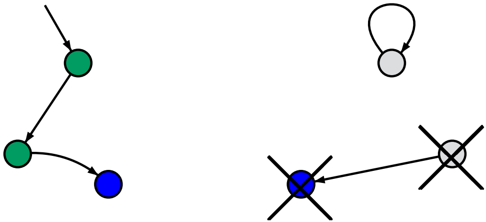
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \, \mathsf{U} \, b)}_{\Psi_1} \vee \exists\Box\neg a)$$



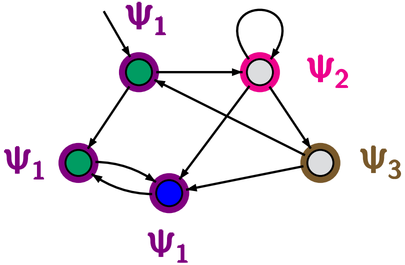
Example: CTL model checking



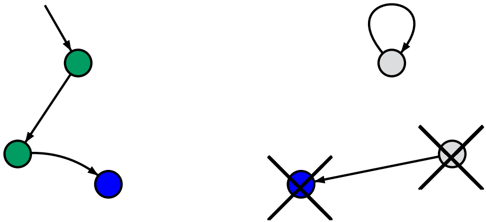
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \,\mathsf{U}\, b)}_{\Psi_1} \vee \underbrace{\exists\Box\neg a}_{\Psi_2})$$



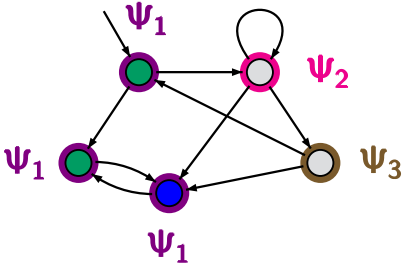
Example: CTL model checking



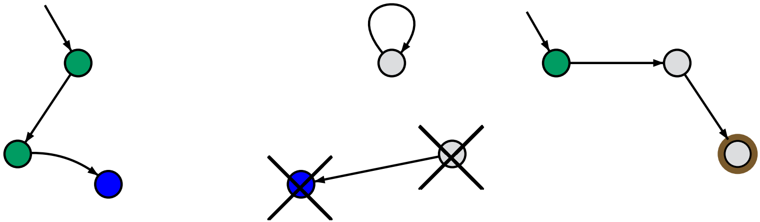
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \,\mathsf{U}\, b)}_{\Psi_1} \vee \underbrace{\exists\Box\neg a}_{\Psi_2}) = \exists\Diamond \underbrace{\neg(\Psi_1 \vee \Psi_2)}_{\Psi_3}$$



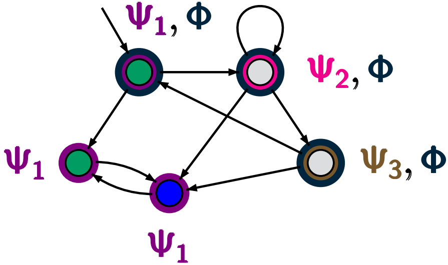
Example: CTL model checking



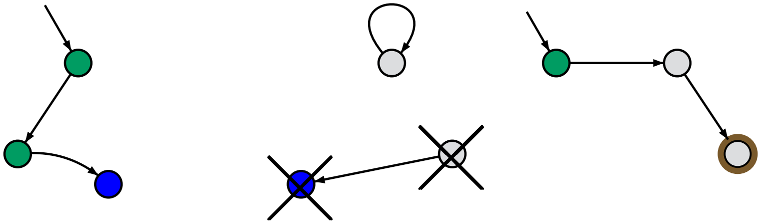
$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a \cup b)}_{\Psi_1} \vee \underbrace{\exists\Box\neg a}_{\Psi_2}) = \exists\Diamond\underbrace{\neg(\Psi_1 \vee \Psi_2)}_{\Psi_3}$$



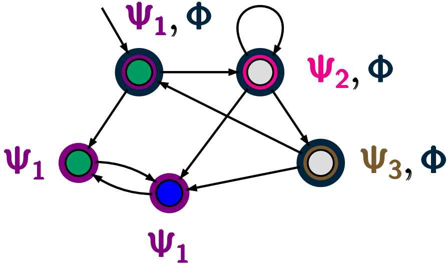
Example: CTL model checking



$$\Phi = \exists\lozenge\neg(\underbrace{\exists(a \,\mathsf{U}\, b)}_{\Psi_1} \vee \underbrace{\exists\square\neg a}_{\Psi_2}) = \exists\lozenge\underbrace{\neg(\Psi_1 \vee \Psi_2)}_{\Psi_3}$$

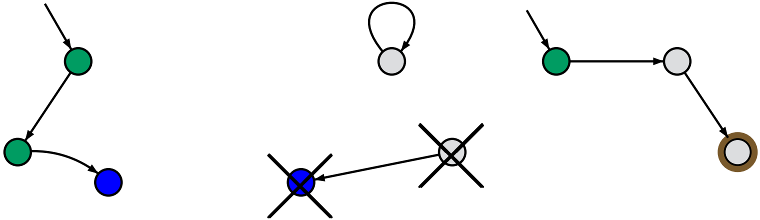


Example: CTL model checking



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

$$\Phi = \exists\Diamond\neg(\underbrace{\exists(a\,\mathsf{U}\,b)}_{\Psi_1} \vee \underbrace{\exists\Box\neg a}_{\Psi_2}) = \exists\Diamond\underbrace{\neg(\Psi_1 \vee \Psi_2)}_{\Psi_3}$$



Complexity of CTL model checking

Complexity of CTL model checking

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

Complexity of CTL/LTL model checking

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

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

Complexity of CTL/LTL model checking

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

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

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

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

Complexity of CTL/LTL model checking

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

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

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

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

If $\Phi \equiv \varphi$ then "often" we have: $|\Phi| = \exp(|\varphi|)$

CTL vs LTL

general observation:

> **CTL** formulas are often "essentially longer" than equivalent **LTL** formulas, provided there is one.

CTL vs LTL

general observation:

CTL formulas are often "essentially longer" than equivalent **LTL** formulas, provided there is one.

Recall: for each **CTL** formula Φ we have:

If Φ is equivalent to some **LTL** formula then:

$\Phi \equiv \varphi$ where φ arises from Φ by deleting all path quantifiers $\forall$, $\exists$ from Φ

CTL vs LTL

general observation:

CTL formulas are often "essentially longer" than equivalent LTL formulas, provided there is one.

Recall: for each **CTL** formula Φ we have:

If Φ is equivalent to some **LTL** formula then:

$\Phi \equiv \varphi$ where φ arises from Φ by deleting all path quantifiers $\forall$, $\exists$ from Φ

In particular: $|\varphi| \leq |\Phi|$

If Φ is equivalent to some **LTL** formula then:

$\Phi \equiv \varphi$ where φ arises from Φ by deleting all path quantifiers $\forall$, $\exists$ from Φ

In particular: $|\varphi| \leq |\Phi|$

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

If Φ is equivalent to some **LTL** formula then:

$\Phi \equiv \varphi$ where φ arises from Φ by deleting all path quantifiers $\forall$, $\exists$ from Φ

In particular: $|\varphi| \leq |\Phi|$

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$

CTL vs LTL

If Φ is equivalent to some **LTL** formula then:

$\Phi \equiv \varphi$ where φ arises from Φ by deleting all path quantifiers $\forall$, $\exists$ from Φ

In particular: $|\varphi| \leq |\Phi|$

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula

If Φ is equivalent to some **LTL** formula then:

$\Phi \equiv \varphi$ where φ arises from Φ by deleting all path quantifiers $\forall$, $\exists$ from Φ

In particular: $|\varphi| \leq |\Phi|$

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula
- there is no **CTL** formula of polynomial length that is equivalent to φ_n

LTL-encoding the Hamilton path problem

LTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + LTL formula φ_n

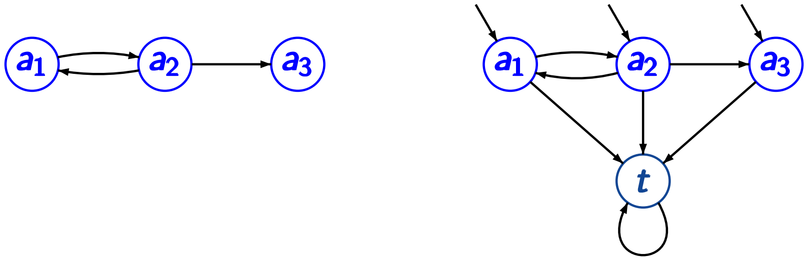
s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\varphi_n$

LTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + LTL formula φ_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\varphi_n$

digraph G $\rightsquigarrow$ transition system $\mathcal{T}_G$

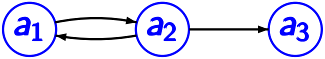


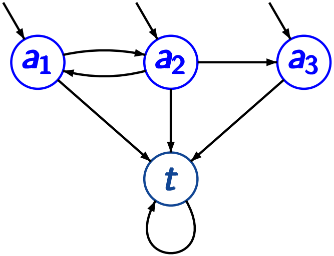
LTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + LTL formula φ_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\varphi_n$

digraph G $\rightsquigarrow$ transition system $\mathcal{T}_G$





$AP = \{a_1, a_2, a_3\}$

LTL-encoding the Hamilton path problem

digraph G with n nodes $\rightsquigarrow$ transition system $\mathcal{T}_G$ + LTL formula φ_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\varphi_n$

$$\varphi_n = \bigwedge_{1 \le i \le n} \Big(\Diamond a_i \wedge \Box (a_i \longrightarrow \bigcirc \Box \neg a_i) \Big)$$

LTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + LTL formula φ'_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\varphi'_n$

$$\varphi'_n = \bigwedge_{1 \leq i \leq n} \Big(\Diamond a_i \wedge \Box (a_i \longrightarrow \bigcirc \Box \neg a_i) \Big)$$

$$\wedge \bigwedge_{1 \leq i \leq n} \Box \Big(a_i \longrightarrow \bigwedge_{k \neq i} \neg a_k \Big)$$

LTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + LTL formula φ'_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\varphi'_n$

$$
\begin{aligned}
\varphi'_n \;=\; & \bigwedge_{1\leq i\leq n} \Big(\Diamond a_i \wedge \Box(a_i \longrightarrow \bigcirc\Box\neg a_i)\Big) \\
\wedge\; & \bigwedge_{1\leq i\leq n} \Box\Big(a_i \longrightarrow \bigwedge_{k\neq i} \neg a_k\Big) \\
\wedge\; & \Box\Big(\bigwedge_{1\leq i\leq n} \neg a_i \longrightarrow \bigcirc \bigwedge_{1\leq i\leq n} \neg a_i\Big)
\end{aligned}
$$

CTL-encoding the Hamilton path problem

CTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + CTL formula Φ_n

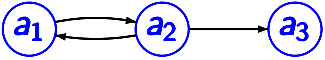
s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\Phi_n$

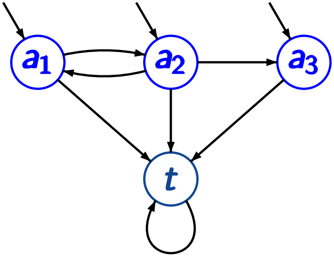
CTL-encoding the Hamilton path problem

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + CTL formula Φ_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\Phi_n$

digraph G $\rightsquigarrow$ transition system $\mathcal{T}_G$





$AP = \{a_1, a_2, a_3\}$

digraph G with n nodes	$\rightsquigarrow$	transition system $\mathcal{T}_G$ + CTL formula Φ_n

s.t. G has a Hamilton path iff $\mathcal{T}_G \not\models \neg\Phi_n$

CTL formula Φ_n, e.g., for $n = 3$:

$$\begin{aligned}
&(a_1 \wedge \exists\bigcirc(a_2 \wedge \exists\bigcirc a_3)) \vee (a_1 \wedge \exists\bigcirc(a_3 \wedge \exists\bigcirc a_2)) \vee \\
&(a_2 \wedge \exists\bigcirc(a_1 \wedge \exists\bigcirc a_3)) \vee (a_2 \wedge \exists\bigcirc(a_3 \wedge \exists\bigcirc a_1)) \vee \\
&(a_3 \wedge \exists\bigcirc(a_1 \wedge \exists\bigcirc a_2)) \vee (a_3 \wedge \exists\bigcirc(a_2 \wedge \exists\bigcirc a_1))
\end{aligned}$$

Formulas φ'_n

LTL formula φ'_n such that $\mathit{Words}(\varphi'_n)$ is

$$\Big\{ \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \Big\}$$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that $Words(\varphi'_n)$ is

$$\Big\{ \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \Big\}$$

CTL formula Φ'_n:

$$\bigvee \{ \Psi(i_1, \ldots, i_n) : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \}$$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that $\mathit{Words}(\varphi'_n)$ is

$$\big\{\{a_{i_1}\}\ldots\{a_{i_n}\}\varnothing^\omega : (i_1,...,i_n) \text{ permutation of } (1,...,n)\big\}$$

CTL formula Φ'_n:

$$\bigvee\{\Psi(i_1,\ldots,i_n) : (i_1,...,i_n) \text{ permutation of } (1,...,n)\}$$

$$\Psi(i_1,i_2,\ldots,i_n) \;=\; a_{i_1} \wedge \bigwedge_{k\neq i_1} \neg a_k \wedge \exists\bigcirc\Psi(i_2,\ldots,i_n)$$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that *Words*(φ'_n) is

$$\big\{ \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega : (i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n) \big\}$$

CTL formula Φ'_n:

$$\bigvee \{ \Psi(i_1, \ldots, i_n) : (i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n) \}$$

$$\Psi(i_1, i_2, \ldots, i_n) = a_{i_1} \wedge \bigwedge_{k \neq i_1} \neg a_k \wedge \exists \bigcirc \Psi(i_2, \ldots, i_n)$$

$$\Psi(i) = a_i \wedge \bigwedge_{k \neq i} \neg a_k \wedge \exists \bigcirc \exists \Box \bigwedge_k \neg a_k$$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that $\mathit{Words}(\varphi'_n)$ is

$$\big\{ \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \big\}$$

CTL formula Φ'_n:

$$\bigvee \{ \Psi(i_1, \ldots, i_n) : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \}$$

$$\Psi(i_1, i_2, \ldots, i_n) = a_{i_1} \wedge \bigwedge_{k \neq i_1} \neg a_k \wedge \exists \bigcirc \Psi(i_2, \ldots, i_n)$$

$$\Psi(i) = a_i \wedge \bigwedge_{k \neq i} \neg a_k \wedge \exists \bigcirc \exists \Box \bigwedge_k \neg a_k$$

show: $\neg \varphi'_n \equiv \neg \Phi'_n$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that ***Words***(φ'_n) is

$$\big\{\{a_{i_1}\}\ldots\{a_{i_n}\}\,\varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n)\big\}$$

CTL formula Φ'_n:

$$\bigvee\big\{\Psi(i_1, \ldots, i_n) : (i_1, ..., i_n) \text{ permutation of } (1, ..., n)\big\}$$

$$\Psi(i_1, i_2, \ldots, i_n) \;=\; a_{i_1} \wedge \bigwedge_{k \neq i_1} \neg a_k \wedge \exists\bigcirc \Psi(i_2, \ldots, i_n)$$

$$\Psi(i) \;=\; a_i \wedge \bigwedge_{k \neq i} \neg a_k \wedge \exists\bigcirc\exists\Box \bigwedge_k \neg a_k$$

show: $\neg\varphi'_n \equiv \neg\Phi'_n \longleftarrow$

for all TS $\mathcal{T}$:

$\mathcal{T} \models \neg\varphi'_n \iff \mathcal{T} \models \neg\Phi'_n$

LTL formula φ'_n such that $Words(\varphi'_n)$ is

$$\big\{ \{a_{i_1}\} \ldots \{a_{i_n}\}\, \varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \big\}$$

CTL formula Φ'_n:

$$\bigvee \big\{ \Psi(i_1, \ldots, i_n) : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \big\}$$

$$\Psi(i_1, i_2, \ldots, i_n) \;=\; a_{i_1} \wedge \bigwedge_{k \neq i_1} \neg a_k \wedge \exists \bigcirc \Psi(i_2, \ldots, i_n)$$

$$\Psi(i) \;=\; a_i \wedge \bigwedge_{k \neq i} \neg a_k \wedge \exists \bigcirc \exists \Box \bigwedge_k \neg a_k$$

show: $\neg \varphi'_n \equiv \neg \Phi'_n \longleftarrow$

for all TS $\mathcal{T}$:

$$\mathcal{T} \not\models \neg \varphi'_n \iff \mathcal{T} \not\models \neg \Phi'_n$$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that *Words*(φ'_n) is

$$\Big\{ \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \Big\}$$

CTL formula Φ'_n:

$$\bigvee \{ \Psi(i_1, \ldots, i_n) : (i_1, ..., i_n) \text{ permutation of } (1, ..., n) \}$$

$\mathcal{T} \not\models \neg\Phi'_n$ iff $\exists$ initial state s_0 with $s_0 \not\models \neg\Phi'_n$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that $Words(\varphi'_n)$ is

$$\big\{\{a_{i_1}\}\ldots\{a_{i_n}\}\varnothing^\omega : (i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n)\big\}$$

CTL formula Φ'_n:

$$\bigvee\{\Psi(i_1,\ldots,i_n) : (i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n)\}$$

$\mathcal{T} \not\models \neg\Phi'_n$ iff $\exists$ initial state s_0 with $s_0 \not\models \neg\Phi'_n$

iff $\exists$ initial state s_0 with $s_0 \models \Phi'_n$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that $Words(\varphi'_n)$ is

$$\{\{a_{i_1}\}\ldots\{a_{i_n}\}\varnothing^\omega : (i_1, ..., i_n) \text{ permutation of } (1, ..., n)\}$$

CTL formula Φ'_n:

$$\bigvee\{\Psi(i_1, \ldots, i_n) : (i_1, ..., i_n) \text{ permutation of } (1, ..., n)\}$$

$\mathcal{T} \not\models \neg\Phi'_n$ iff $\exists$ initial state s_0 with $s_0 \not\models \neg\Phi'_n$

iff $\exists$ initial state s_0 with $s_0 \models \Phi'_n$

iff $\exists$ permutation $(i_1, \ldots, i_n)$ of $(1, \ldots, n)$

s.t. $\{a_{i_1}\}\ldots\{a_{i_n}\}\varnothing^\omega \in Traces(\mathcal{T})$

Formulas φ'_n and Φ'_n

LTL formula φ'_n such that $Words(\varphi'_n)$ is

$$\big\{ \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega : (i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n) \big\}$$

CTL formula Φ'_n:

$$\bigvee \{ \Psi(i_1, \ldots, i_n) : (i_1, \ldots, i_n) \text{ permutation of } (1, \ldots, n) \}$$

$$\begin{aligned}
\mathcal{T} \not\models \neg\Phi'_n \quad & \text{iff} \quad \exists \text{ initial state } s_0 \text{ with } s_0 \not\models \neg\Phi'_n \\
& \text{iff} \quad \exists \text{ initial state } s_0 \text{ with } s_0 \models \Phi'_n \\
& \text{iff} \quad \exists \text{ permutation } (i_1, \ldots, i_n) \text{ of } (1, \ldots, n) \\
& \qquad\ \ \text{s.t. } \{a_{i_1}\} \ldots \{a_{i_n}\} \varnothing^\omega \in Traces(\mathcal{T}) \\
& \text{iff} \quad \mathcal{T} \not\models \neg\varphi'_n
\end{aligned}$$

CTL vs LTL

CTL vs LTL

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n\geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

CTL vs LTL

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n\geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$.

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$. Then:

- $|\varphi_n| = |\varphi'_n| + 1 = \mathcal{O}(n^2)$

CTL vs LTL

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$. Then:

- $|\varphi_n| = |\varphi'_n| + 1 = \mathcal{O}(n^2)$
- φ_n has an equivalent **CTL** formula,

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$. Then:

- $|\varphi_n| = |\varphi'_n| + 1 = \mathcal{O}(n^2)$
- φ_n has an equivalent **CTL** formula, e.g., $\neg\Phi'_n$

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$. Then:

- $|\varphi_n| = |\varphi'_n| + 1 = \mathcal{O}(n^2)$
- φ_n has an equivalent **CTL** formula, e.g., $\neg\Phi'_n$

Suppose there is a **CTL** formula of polynomial length that is equivalent to φ_n.

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n\geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$. Then:

- $|\varphi_n| = |\varphi'_n| + 1 = \mathcal{O}(n^2)$
- φ_n has an equivalent **CTL** formula, e.g., $\neg\Phi'_n$

Suppose there is a **CTL** formula of polynomial length that is equivalent to φ_n. Then:

Hamilton path problem $\in P$

If $P \neq NP$ then there is a sequence $(\varphi_n)_{n \geq 0}$ of **LTL** formulas such that:

- $|\varphi_n| = \mathcal{O}(poly(n))$
- φ_n has an equivalent **CTL** formula, but no equivalent **CTL** formula of polynomial length

Proof. Let $\varphi_n = \neg\varphi'_n$. Then:

- $|\varphi_n| = |\varphi'_n| + 1 = \mathcal{O}(n^2)$
- φ_n has an equivalent **CTL** formula, e.g., $\neg\Phi'_n$

Suppose there is a **CTL** formula of polynomial length that is equivalent to φ_n. Then:

Hamilton path problem $\in P$ and $P = NP$