Model Checking

CS560: Reasoning About Programs

Roopsha Samanta

PURDUE UNIVERSITY

Based on slides by Georg Weissenbacher

Roadmap

Previously

- Bounded model checking for programs

Today

- Model checking for Kripke structures

CTL Model Checking

Clarke

Emerson

Sifakis

Clarke & Emerson, *Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic, 1981*

Algorithmic framework for exhaustive exploration of finite-state transition systems to check temporal properties

Knaster-Tarski Theorem: Definitions

powerset of S

Given a state space S, a function/predicate transformer $f: 2^S \mapsto 2^S$ is **monotone** if $\forall X, Y \in S$: $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$

X is a **fixpoint** of function f if $f(X) = X$.

X, is a **least fixpoint** of function f if for any fixpoint Y, $X \subseteq Y$.

X is a **greatest fixpoint** of function f if for any fixpoint Y, $X \supseteq Y$.

$\mu(Y). f(Y)$

$\nu(Y). f(Y)$

Knaster-Tarski Theorem

Let S be a set of states and $f: 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. $\mu(Y).f(Y) = \cap\{Y: f(Y) = Y\} = \cup_i \ f^i(false)$
2. $\nu(Y).f(Y) = \cup\{Y: f(Y) = Y\} = \cap_i \ f^i(true)$

All the fixpoints

- ▶ Monotone functions always have a least and a greatest fix point!
- ▶ These fixpoints can be easily computed
- ▶ The meanings of CTL operators can be expressed as fixpoints of monotone functions on 2^S, enabling efficient model checking

Knaster-Tarski Theorem

Prefixed point $\{Y: f(Y) \subseteq Y\}$

Postfixed point : $\{Y: f(Y) \supseteq Y\}$

Let S be a set of states and $f: 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. $\mu(Y).f(Y) = \cap\{Y: f(Y) = Y\} = \cap\{Y: f(Y) \subseteq Y) = \cup_i \ f^i(false)$
2. $\nu(Y).f(Y) = \cup\{Y: f(Y) = Y\} = \cup\{Y: f(Y) \supseteq Y) = \cap_i \ f^i(true)$

- ▶ Monotone functions always have a least and a greatest fix point!
- ▶ These fixpoints can be easily computed
- ▶ The meanings of CTL operators can be expressed as fixpoints of monotone functions on 2^S, enabling efficient model checking

Knaster-Tarski Theorem

Simpler version when S is finite

Let S be a set of states and $f: 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. $\mu(Y).f(Y) = \cap\{Y: f(Y) = Y\} = \cap\{Y: f(Y) \subseteq Y) = \cup_i\ f^i(false)$
2. $\nu(Y).f(Y) = \cup\{Y: f(Y) = Y\} = \cup\{Y: f(Y) \supseteq Y) = \cap_i\ f^i(true)$

Let $S = \{s_0, s_1, \ldots, s_n\}$ be a set of states and $f: 2^S \mapsto 2^S$ be a monotone predicate transformer. Then:

1. the least fixpoint exists and equals $f^{n+1}(\emptyset)$ and
2. the greatest fixpoint exists and equals $f^{n+1}(S)$.

Knaster-Tarski Theorem: Definitions

A complete lattice is a partially ordered set $(L, \leq)$ where every subset of L has a glb and an lub.

A function f over a lattice $(L, \leq)$ is monotonic if for all $x, y \in L$: $x \leq y \;\Rightarrow\; f(x) \leq f(y)$

Point x is a fixpoint of function f if $f(x) = x$, a prefixed point if $f(x) \leq x$ and a postfixed point if $f(x) \geq x$

Given a state space S, the power set 2^S is a complete lattice where $\leq$ is the subset relation.

Consider a monotonic predicate transformer $f: 2^S \mapsto 2^S$ over this lattice.

Let $(L, \leq)$ be a complete lattice and $f: L \mapsto L$ be a monotone function. Then:
1. the least fixpoint exists and equals the least prefixed point,
2. the greatest fixpoint exists and equals the greatest postfixed point, and
3 the fixpoints form a complete lattice

Kanster-Tarski Theorem

Knaster, *Un th´eor`eme sur les fonctions d'ensembles, 1927*

Tarski, *A lattice-theoretical fixpoint theorem and its application, 1955*

Model Checking CTL

- For each CTL formula $\boldsymbol{\varphi}$, we will compute

$$\{s \mid \mathcal{M}, s \models \varphi\}$$

$\{s \mid s \models \phi\}$

- CTL can be expressed in terms of $\neg$, $\vee$, $\boldsymbol{EX}$, $\boldsymbol{EU}$, and $\boldsymbol{EG}$
- Will define these operators by induction:
  - $\boldsymbol{EX}\varphi \stackrel{\text{def}}{=} \{ s_0 \mid \exists s_1.\ T(s_0, s_1) \wedge \mathcal{M}, s_1 \models \varphi\}$

EX EF EG EU

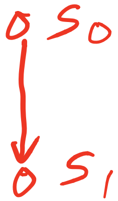
AX
AF
AG
AU

Model Checking CTL

- For each CTL formula φ, we will compute

$$\{s \mid \mathcal{M}, s \models \varphi\}$$

- CTL can be expressed in terms of $\neg$, $\vee$, $\boldsymbol{EX}$, $\boldsymbol{EU}$, and $\boldsymbol{EG}$
- Will define these operators by induction:
  - $\boldsymbol{EX}\varphi \stackrel{\text{def}}{=} \{ s_0 \mid \exists s_1.\ T(s_0, s_1) \wedge \mathcal{M}, s_1 \models \varphi\}$
    - Note: This is the pre-image of T with respect to φ



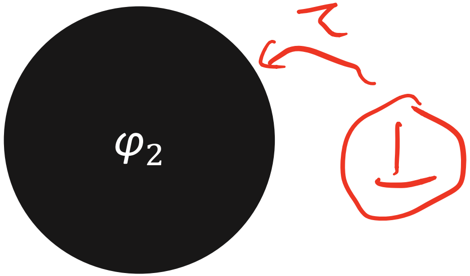
Model Checking CTL

- It remains to be shown that $\boldsymbol{EG\varphi}$ and $\boldsymbol{EU\varphi}$ can be computed
- We claim:
  - $\boldsymbol{EG}\varphi \equiv \nu Z.\varphi \land \boldsymbol{EX}\ Z$
    - i.e., $\boldsymbol{EG}\varphi$ is greatest fixed point of $\tau(Z) = \varphi \land \boldsymbol{EX}\ Z$
  - $\boldsymbol{E}(\varphi_1 \boldsymbol{U}\ \varphi_2); \equiv \mu Z.\varphi_2 \lor (\varphi_1 \land \boldsymbol{EX}\ Z)$
    - i.e., $\boldsymbol{E}(\varphi_1\ \boldsymbol{U}\ \varphi_2)$ is least fixed point of $\tau(Z) = \varphi_2 \lor (\varphi_1 \land \boldsymbol{EX}\ Z)$
    - Recall least fixed point of strongest post condition

Model Checking CTL

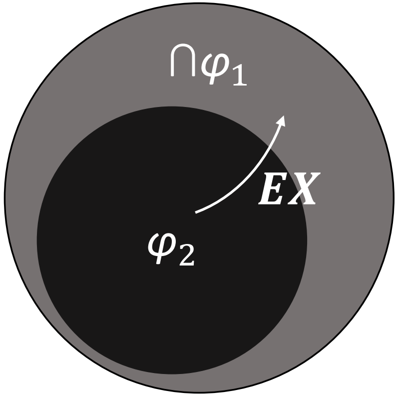
- $\boldsymbol{E}(\varphi_1 \boldsymbol{U} \varphi_2); \equiv \mu Z.\varphi_2 \vee (\varphi_1 \wedge \boldsymbol{EX}\ Z)$
  - Remember: $\boldsymbol{EX}$ is “pre-image”

- $\boldsymbol{E}(\varphi_1 \boldsymbol{U} \varphi_2)$ holds in φ_2



Model Checking CTL

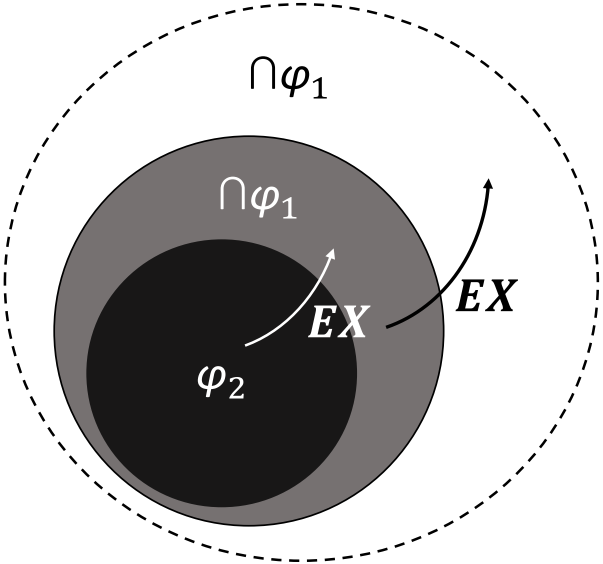
- $\boldsymbol{E}(\varphi_1 \boldsymbol{U} \varphi_2); \equiv \mu Z.\varphi_2 \vee (\varphi_1 \wedge \boldsymbol{EX}\ Z)$
  - Remember: $\boldsymbol{EX}$ is "pre-image"



- $\boldsymbol{E}(\varphi_1 \boldsymbol{U} \varphi_2)$ holds in φ_2
- And in predecessor states of φ_2 in which φ_1 holds

Model Checking CTL

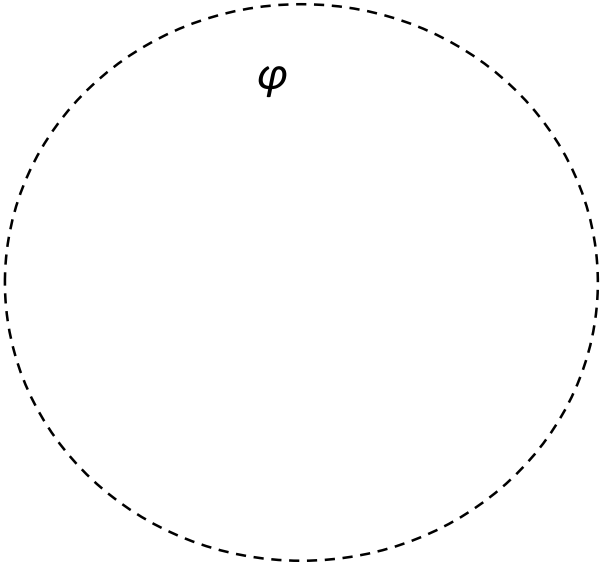
- $\boldsymbol{E}(\varphi_1 \boldsymbol{U} \varphi_2); \equiv \mu Z.\varphi_2 \vee (\varphi_1 \wedge \boldsymbol{EX}\ Z)$
  - Remember: $\boldsymbol{EX}$ is "pre-image"



- $\boldsymbol{E}(\varphi_1 \boldsymbol{U} \varphi_2)$ holds in φ_2
- And in predecessor states of φ_2 in which φ_1 holds
- Fixed point: Transitive closure of all such predecessor states

Model Checking CTL

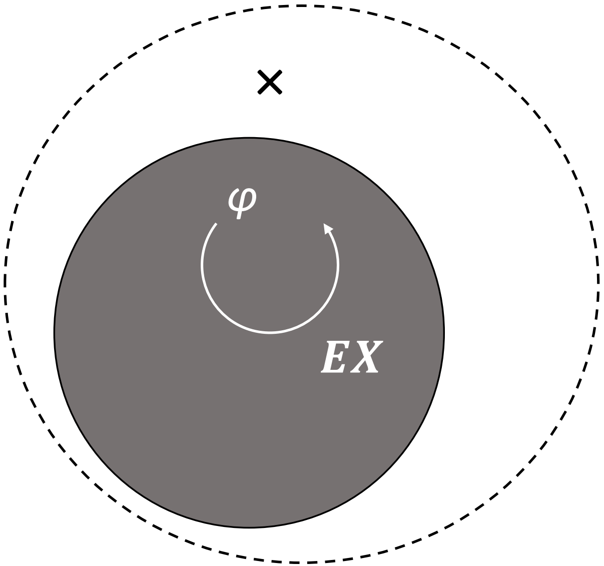
- $\boldsymbol{EG}\varphi \equiv \nu Z.\varphi \wedge \boldsymbol{EX}\ Z$
  - Remember: $\boldsymbol{EX}$ is “pre-image”



- Start with all states in which φ holds

Model Checking CTL

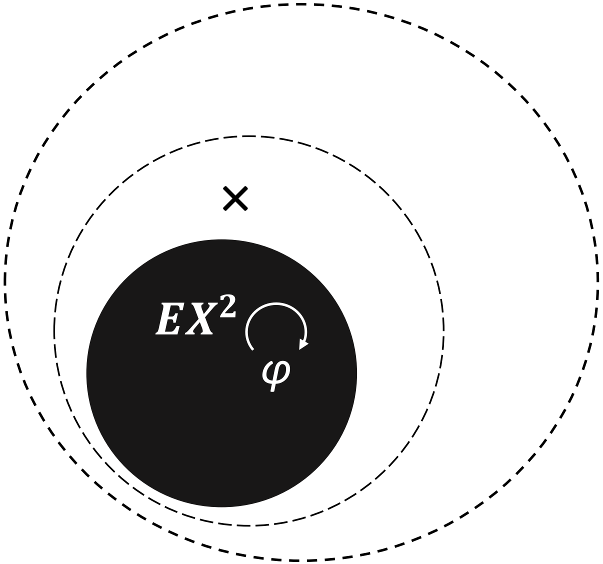
- $\boldsymbol{EG}\varphi \equiv \nu Z.\varphi \wedge \boldsymbol{EX}\ Z$
  - Remember: $\boldsymbol{EX}$ is “pre-image”



- Start with all states in which φ holds
- Shrink to states in φ such that φ still holds after 1 step

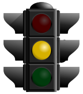
Model Checking CTL

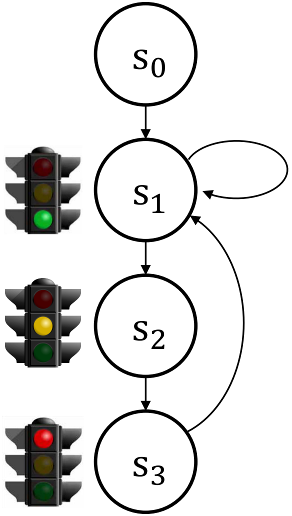
- $\boldsymbol{EG}\varphi \equiv \nu Z.\varphi \wedge \boldsymbol{EX}\ Z$
  - Remember: $\boldsymbol{EX}$ is "pre-image"

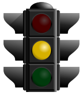


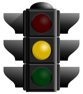
- Start with all states in which φ holds
- Shrink to states in φ such that φ still holds after 1 step
- Keep shrinking until fixed point reached

Model Checking CTL

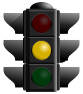
$\boldsymbol{E}($  $\boldsymbol{U}$  $)$

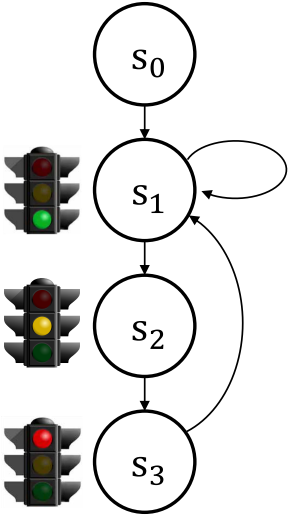


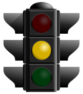
$\mu Z\,.$  $\vee\,($  $\wedge\, \boldsymbol{EX}\, Z\,)$

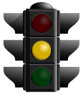
1.  $\vee\,($  $\wedge\, \boldsymbol{EX}\ \perp\,) = \{\, s_2 \,\}$

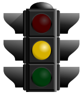
Model Checking CTL

$\boldsymbol{E}($  $\boldsymbol{U}$  $)$

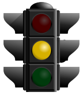


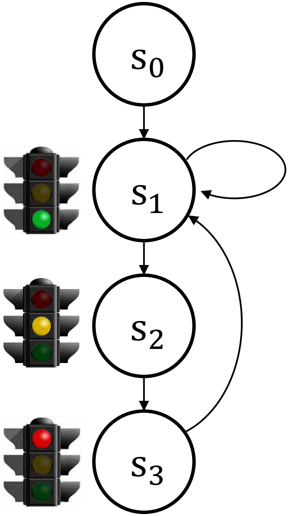
$\mu Z\,.$  $\vee\;($  $\wedge\;\boldsymbol{EX}\;Z\,)$

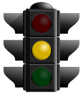
1.  $\vee\;($  $\wedge\;\boldsymbol{EX}\;\;\bot\;\;) = \{\,s_2\,\}$

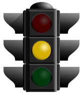
2.  $\vee\;($  $\wedge\;\boldsymbol{EX}\;\{s_2\}\;) =$

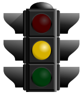
Model Checking CTL

$\boldsymbol{E}($  $\boldsymbol{U}$  $)$

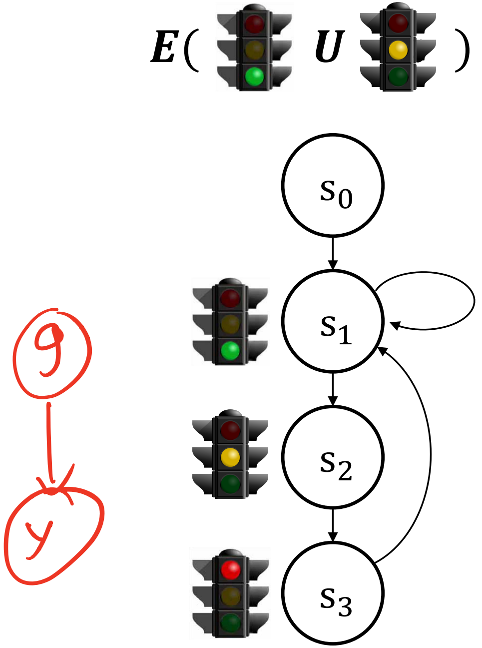


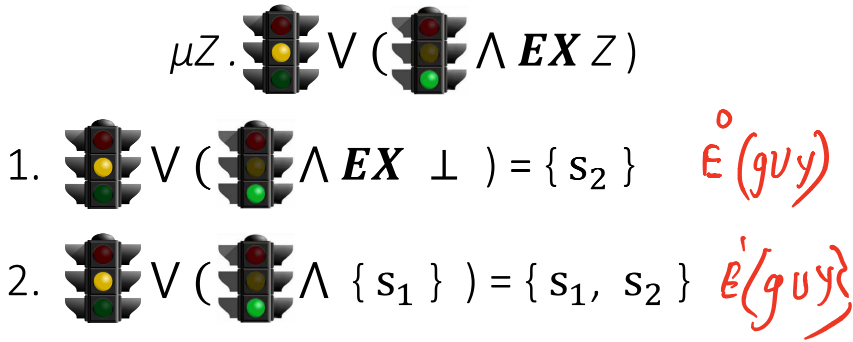
$\mu Z\,.$  $\vee\,($  $\wedge\, \boldsymbol{EX}\, Z\,)$

1.  $\vee\,($  $\wedge\, \boldsymbol{EX}\ \perp\ ) = \{\, s_2 \,\}$

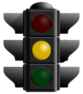
2.  $\vee\,($  $\wedge\ \{\, s_1 \,\}\ ) =$

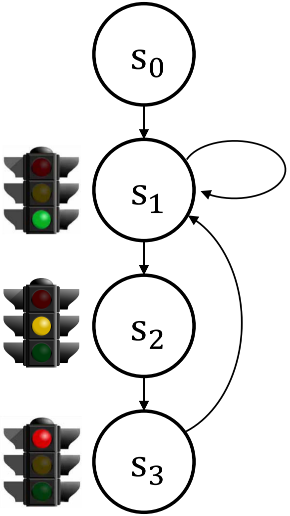
Model Checking CTL

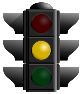


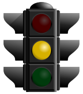


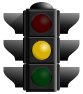
Model Checking CTL

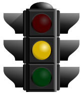
$\boldsymbol{E}($  $\boldsymbol{U}$  $)$



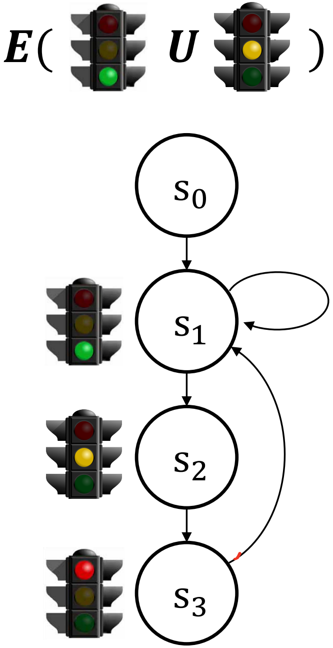
$\mu Z\,.$  $\vee\,($  $\wedge\, \boldsymbol{EX}\, Z\,)$

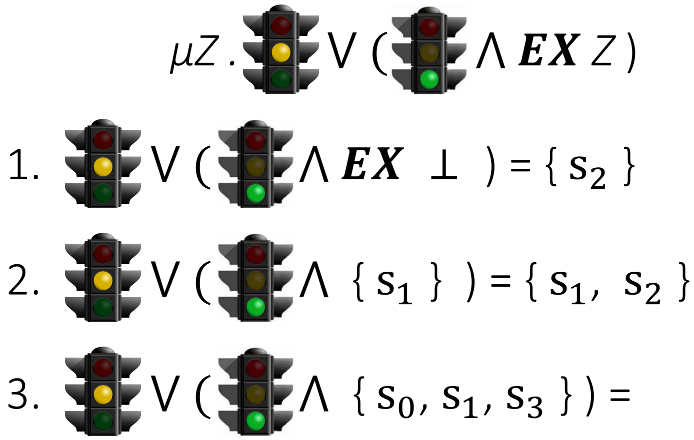
1.  $\vee\,($  $\wedge\, \boldsymbol{EX}\ \perp\ ) = \{\, s_2 \,\}$

2.  $\vee\,($  $\wedge\ \{\, s_1 \,\}\ ) = \{\, s_1,\ s_2 \,\}$

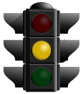
3.  $\vee\,($  $\wedge\, \boldsymbol{EX}\,\{\, s_1,\ s_2 \,\}\,) =$

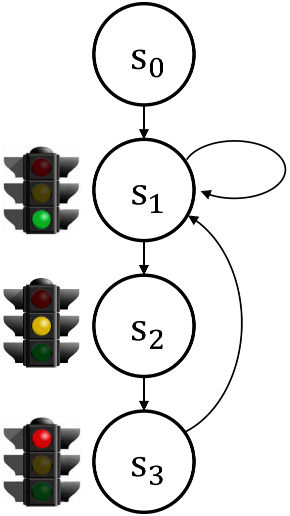
Model Checking CTL

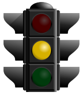


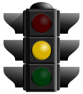


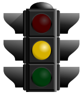
Model Checking CTL

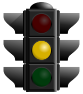
$\boldsymbol{E}($  $\boldsymbol{U}$  $)$



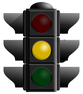
$\mu Z\,.$  $\vee\,($  $\wedge\, \boldsymbol{EX}\, Z\,)$

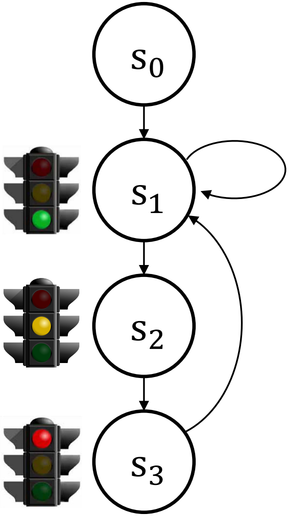
1.  $\vee\,($  $\wedge\, \boldsymbol{EX}\ \perp\ ) = \{\, s_2 \,\}$

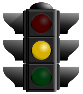
2.  $\vee\,($  $\wedge\ \{\, s_1 \,\}\ ) = \{\, s_1,\ s_2 \,\}$

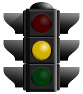
3.  $\vee\,(\,\{\, s_1 \,\}\,) =$

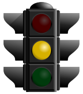
Model Checking CTL

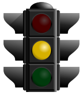
$E($  U  $)$



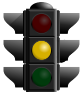
$\mu Z .$  $\vee\,($  $\wedge\ \boldsymbol{EX}\ Z\,)$

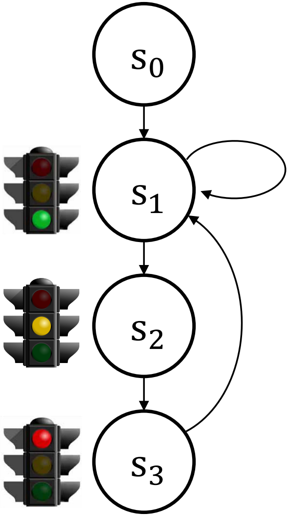
1.  $\vee\,($  $\wedge\ \boldsymbol{EX}\ \perp\,) = \{ s_2 \}$

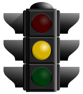
2.  $\vee\,($  $\wedge\ \{ s_1 \}\,) = \{ s_1, s_2 \}$

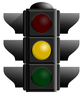
3.  $\vee\,(\{ s_1 \}) = \{ s_1, s_2 \}$

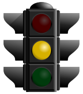
Model Checking CTL

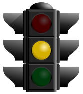
$\boldsymbol{E}($  $\boldsymbol{U}$  $)$



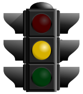
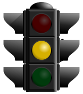
μZ .  $\vee$ ( $\wedge\ \boldsymbol{EX}\ Z$)

1.  $\vee$ ( $\wedge\ \boldsymbol{EX}\ \perp$) $= \{ s_2 \}$

2.  $\vee$ ( $\wedge\ \{ s_1 \}$) $= \{ s_1,\ s_2 \}$

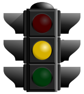
3.  $\vee$ ($\{ s_1 \}$) $= \{ s_1,\ s_2 \}$

4. Fixed Point!

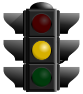
- $\mathcal{M}, s_1 \vDash \boldsymbol{E}($  $\boldsymbol{U}$  $)$
- $\mathcal{M}, s_2 \vDash \boldsymbol{E}($  $\boldsymbol{U}$  $)$

Model Checking CTL

- More complex formulas?
  - Start with innermost sub-formulas!
  - Compute nested fixed point
- Remember:

$\boldsymbol{E}($  $\boldsymbol{U}$  $) = \{ s_1,\ s_2 \}$

- So if we want to compute

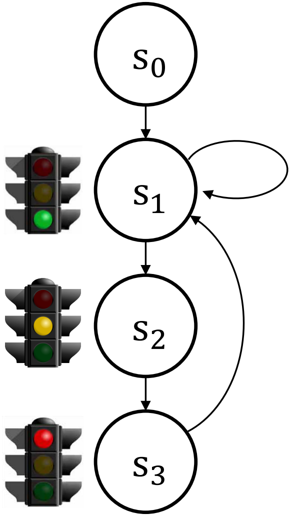
$\boldsymbol{EG}(\boldsymbol{E}($  $\boldsymbol{U}$  $))$

- We compute greatest fixed point

$$\nu Z.\{ s_1,\ s_2 \} \wedge \boldsymbol{EX}\ Z$$

Model Checking CTL

- Let's compute the greatest fixed point

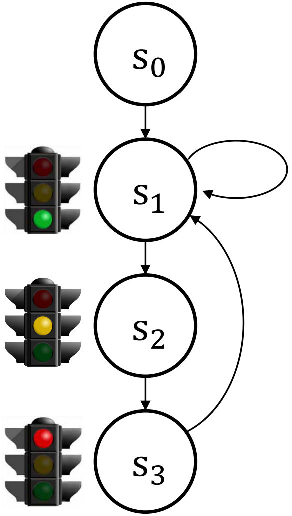


$$\nu Z . \{ s_1, \ s_2 \} \wedge \boldsymbol{EX}\, Z$$

$$1. \{ s_1, \ s_2 \} \wedge \boldsymbol{EX}\, \top$$

Model Checking CTL

- Let's compute the greatest fixed point

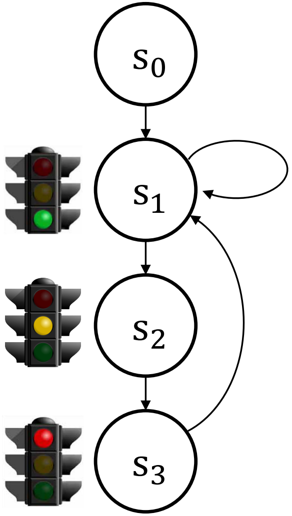


$$\nu Z.\{ s_1,\ s_2 \} \land \boldsymbol{EX}\, Z$$

$$1.\ \{ s_1,\ s_2 \} \land \top$$

Model Checking CTL

- Let's compute the greatest fixed point

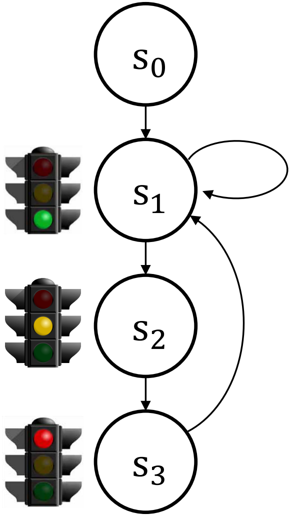


$$\nu Z . \{ s_1, \ s_2 \} \land \boldsymbol{EX}\, Z$$

$$1. \{ s_1, \ s_2 \} \land \top \ = \{ s_1, \ s_2 \}$$

Model Checking CTL

- Let's compute the greatest fixed point



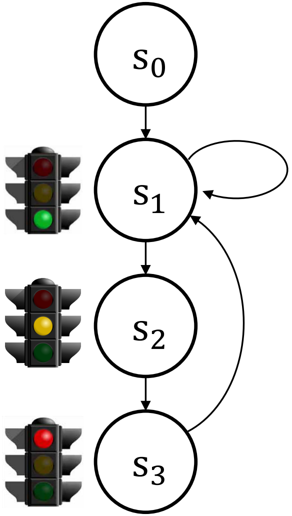
$$\nu Z.\{\, s_1,\ s_2 \,\} \wedge \boldsymbol{EX}\, Z$$

1. $\{\, s_1,\ s_2 \,\} \wedge \top \ = \{\, s_1,\ s_2 \,\}$

2. $\{\, s_1,\ s_2 \,\} \wedge \boldsymbol{EX} \{\, s_1,\ s_2 \,\}$

Model Checking CTL

- Let's compute the greatest fixed point



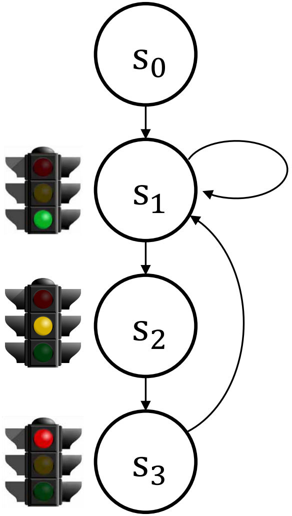
$\nu Z\,.\{\, s_1,\ s_2 \,\} \wedge \boldsymbol{EX}\, Z$

1. $\{\, s_1,\ s_2 \,\} \wedge \top \ = \{\, s_1,\ s_2 \,\}$

2. $\{\, s_1,\ s_2 \,\} \wedge \{s_0, s_1, s_3 \,\}$

Model Checking CTL

- Let’s compute the greatest fixed point



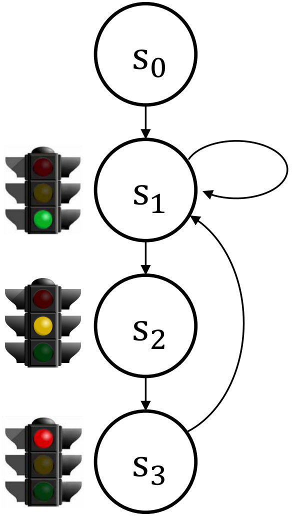
$$\nu Z . \{ s_1, s_2 \} \land \boldsymbol{EX}\, Z$$

1. $\{ s_1, s_2 \} \land \top = \{ s_1, s_2 \}$

2. $\{ s_1, s_2 \} \land \{ s_0, s_1, s_3 \} = \{ s_1 \}$

Model Checking CTL

- Let's compute the greatest fixed point



$\nu Z.\{s_1, s_2\} \wedge \boldsymbol{EX}\, Z$

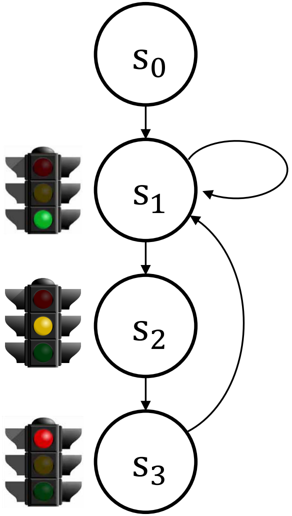
1. $\{s_1, s_2\} \wedge \top = \{s_1, s_2\}$

2. $\{s_1, s_2\} \wedge \{s_0, s_1, s_3\} = \{s_1\}$

3. $\{s_1, s_2\} \wedge \boldsymbol{EX}\, \{s_1\}$

Model Checking CTL

- Let's compute the greatest fixed point



$\nu Z.\{ s_1,\ s_2 \} \wedge \boldsymbol{EX}\, Z$

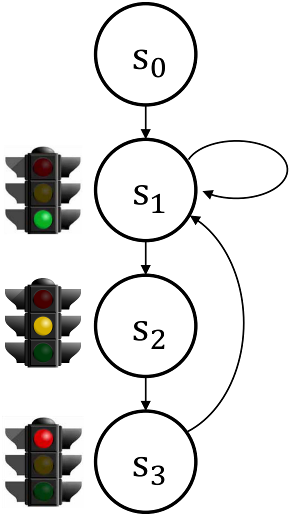
1. $\{ s_1,\ s_2 \} \wedge \top \ = \{ s_1,\ s_2 \}$

2. $\{ s_1,\ s_2 \} \wedge \{s_0, s_1, s_3 \} = \{ s_1\}$

3. $\{ s_1,\ s_2 \} \wedge \{s_0, s_1, s_3 \}$

Model Checking CTL

- Let's compute the greatest fixed point



$\nu Z\,.\{\, s_1,\ s_2 \,\} \land \boldsymbol{EX}\, Z$

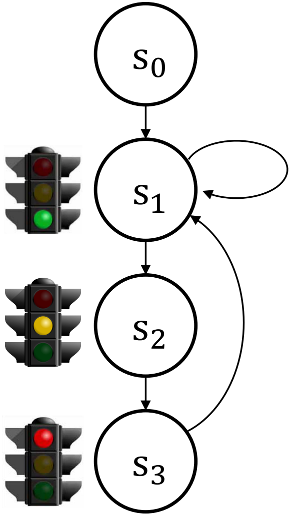
1. $\{\, s_1,\ s_2 \,\} \land \top \ = \{\, s_1,\ s_2 \,\}$

2. $\{\, s_1,\ s_2 \,\} \land \{s_0, s_1, s_3 \,\} = \{\, s_1\}$

3. $\{\, s_1,\ s_2 \,\} \land \{s_0, s_1, s_3 \,\} = \{\, s_1\}$

Model Checking CTL

- Let’s compute the greatest fixed point



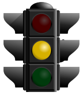
$\nu Z.\{ s_1, s_2 \} \wedge \boldsymbol{EX}\, Z$

1. $\{ s_1, s_2 \} \wedge \top = \{ s_1, s_2 \}$

2. $\{ s_1, s_2 \} \wedge \{s_0, s_1, s_3 \} = \{ s_1 \}$

3. $\{ s_1, s_2 \} \wedge \{s_0, s_1, s_3 \} = \{ s_1 \}$

4. Fixed Point!

$\mathcal{M}, s_1 \vDash \boldsymbol{EG}(\boldsymbol{E}($  $\boldsymbol{U}$  $))$

Model Checking CTL

- Worst case complexity of this algorithm?
- Checking CTL-Formula φ for $\langle S, T, I, L \rangle$ is O($|\varphi| \cdot (|S| + |T|)$)
- Why?
  - Each fixed point is O($|S| + |T|$)
  - We have to compute O($|\varphi|$) fixed points

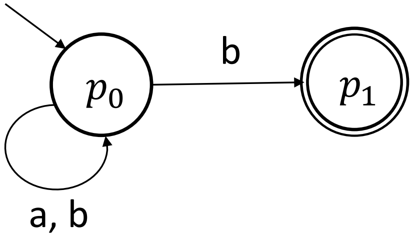
LTL Model Checking

FA in ≡ Regular lang in
Pushdown auto ≡ Context-free grammar
⋮

- We'll look at the problem from a new angle:

  Model Checking using Automata Theory

- Remember: a *finite automaton* accepts a *finite* input if a *final* state is reached



aabb

$p_0 \xrightarrow{a} p_0 \xrightarrow{a} p_0 \xrightarrow{b} p_0$
$p_0 \xrightarrow{b} p_1$

(This automaton accepts (a|b)*b – all words ending with b)

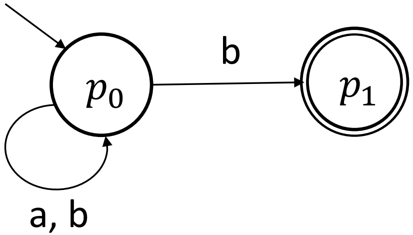
Automata Theory

Definition (Finite Automaton)

A *finite automaton* A is a tuple $<\Sigma, Q, \delta, Q_0, F>$

- Σ is the *input alphabet*
- Q is a finite set of *states*
- $\delta : Q \times \Sigma \times Q$ is the *transition relation*
- $Q_0 \subseteq Q$ is the set of *initial states*
- $F \subseteq Q$ is the set of *final states*

$$\delta = \{(p_0, b, p_1), (p_0, b, p_0), (p_0, a, p_0)\}$$



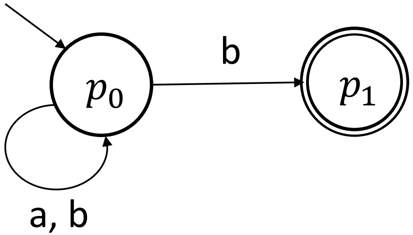
$$\Sigma = \{a, b\}, \quad Q = \{p_0, p_1\}, \quad Q_0 = \{p_0\}, \quad F = \{p_1\}$$

Automata Theory

- Let $\mathcal{A}$ be a finite automaton over an input alphabet Σ
- Then $\mathcal{L}(\mathcal{A})$ denotes the language:

$$\{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$$

- i.e., $\mathcal{L}(\mathcal{A})$ consists of all finite words accepted by $\mathcal{A}$



(This automaton accepts (a|b)*b – all words ending with b)

ω-Automata

- Maybe we can define an automaton accepting "good" *execution traces*?

ω-Automata

- Maybe we can define an automaton accepting “good” execution traces?
- Problem: An execution trace is an infinite sequence of states

ω-Automata

- Maybe we can define an automaton accepting "good" execution traces?
- Problem: An execution trace is an infinite sequence of states
- Solution: Define automata accepting *infinite* input words
  - These automata are called ω-Automata or Büchi-automata

ω-Automata

Definition (Büchi Automaton)

A *Büchi automaton* B is a tuple <Σ, Q, δ, Q_0, F>

- Σ is the *input alphabet*
- Q is a finite set of *states*
- $\delta : Q \times \Sigma \times Q$ is the *transition relation*
- $Q_0 \subseteq Q$ is the set of *initial states*
- $F \subseteq Q$ is the set of *accepting states*

ω-Automata

Definition (Büchi Automaton)

A *Büchi automaton* B is a tuple $<\Sigma, Q, \delta, Q_0, F>$

- Σ is the *input alphabet*
- Q is a finite set of *states*
- $\delta : Q \times \Sigma \times Q$ is the *transition relation*
- $Q_0 \subseteq Q$ is the set of *initial states*
- $F \subseteq Q$ is the set of *accepting states*

Wait… isn't that exactly the same definition as before?

ω-Automata

Definition (Büchi Automaton)

A *Büchi automaton* B is a tuple <Σ, Q, δ, Q_0, F>

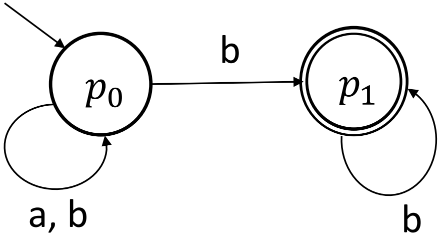
- Σ is the *input alphabet*
- Q is a finite set of *states*
- $\delta : Q \times \Sigma \times Q$ is the *transition relation*
- $Q_0 \subseteq Q$ is the set of *initial states*
- $F \subseteq Q$ is the set of *accepting* *states*

Acceptance condition

- A Büchi automaton accepts an *infinite* word $w \in \Sigma^{\omega}$ if the corresponding *run* visits at least one state in F infinitely often

ω-Automata

- We use Σ^{ω} to denote all words of infinite length
- The following automaton accepts all words $w \in \Sigma^{\omega}$ with *finitely* many as



$p_0\ p_0 \cdots\ p_0\ \underline{p_1\ p_1 \cdots p_1}$

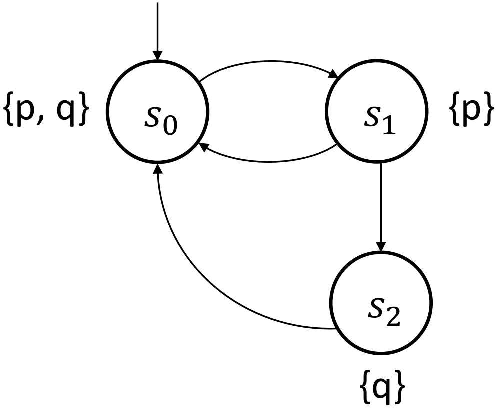
Σ = {a, b}, $\mathrm{Q} = \{p_0, p_1\}$, $\mathrm{Q}_0 = \{p_0\}$, $\mathrm{F} = \{p_1\}$

ω-Automata

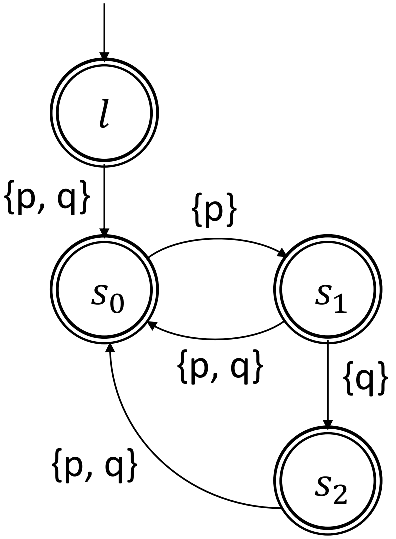
- Intuitively, an automaton B defines a set of infinite behaviors

ω-Automata

- Intuitively, an automaton B defines a set of infinite behaviors
- We can use Büchi automata to represent Kripke structures!
  - Make *all* states accepting states
  - Label incoming edges of s_i with $L(s_i)$



$\equiv$



Model Checking with ω-Automata

- We can encode any Kripke structure $\mathcal{M}$ as a Büchi automaton $B_{\mathcal{M}}$
  - For a given run s_0, s_1 … of $\mathcal{M}$, $B_{\mathcal{M}}$ accepts $L(s_0)\ L(s_1)$ …
- Now assume we have a second automaton, B_φ representing a "specification" φ
- Let B_φ be the automaton accepting "all good behaviors" according to φ
- Then $\overline{B}_\varphi$ is the automaton accepting "all bad behaviors"
- Then $\mathcal{M}$ can't behave badly if

$$\mathcal{L}(B_{\mathcal{M}}) \cap \mathcal{L}(\overline{B}_\varphi) = \emptyset$$

Emptiness checking of Büchi automata

Model Checking with ω-Automata

- We can encode any Kripke structure $\mathcal{M}$ as a Büchi automaton $B_{\mathcal{M}}$
  - For a given run s_0, s_1 … of $\mathcal{M}$, $B_{\mathcal{M}}$ accepts $L(s_0)\ L(s_1)$ …
- Now assume we have a second automaton, B_{φ} representing a “specification” φ
- Let B_{φ} be the automaton accepting “all good behaviors” according to φ
- Then $\overline{B}_{\varphi}$ is the automaton accepting “all bad behaviors”
- Then $\mathcal{M}$ can’t behave badly if

$$\mathcal{L}(B_{\mathcal{M}}) \cap \mathcal{L}(\overline{B}_{\varphi}) = \emptyset$$

$$\mathcal{L}(B_{\mathcal{M}}) \cap \mathcal{L}(\overline{B}_{\varphi})$$

Is accepted by the “intersection of the automata”

$B_{\mathcal{M}}$ and $\overline{B}_{\varphi}$

Automaton-based Model Checking of LTL

Safra's construction.

Given $\mathcal{M} = \langle S, T, I, L \rangle$ and $\boldsymbol{A}\varphi$

1. Construct $B_{\mathcal{M}}$
2. Put $\neg\varphi$ into negation normal form
3. Construct $B_{\overline{\varphi}}$ for $\neg\varphi$ in NNF
   Negating Büchi automata is hard – we want to avoid this step
4. Construct $B = B_{\mathcal{M}} \cap B_{\overline{\varphi}}$
5. Check B for emptiness ($\mathcal{L}(B) \overset{?}{=} \emptyset$)

Summary

Today

- CTL model checking using nested fixed points
- LTL model checking using automata (overview)

Next

- Reading days
- Program synthesis!