Automated Program Verification and Testing
15414/15614 Fall 2016
Lecture 23:
Symbolic Model Checking

Matt Fredrikson
mfredrik@cs.cmu.edu

November 22, 2016
NFAs and REs describe languages containing finite words
NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors
NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We’ll describe such behaviors using $\omega$-regular languages
Languages of Infinite Words (Review)

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We’ll describe such behaviors using \( \omega \)-regular languages

These can be described by \( \omega \)-regular expressions of the form:

\[
E_1 F_1^\omega + \cdots + E_n F_n^\omega
\]
NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We’ll describe such behaviors using \( \omega \)-regular languages

These can be described by \( \omega \)-regular expressions of the form:

\[
E_1 F_1^\omega + \cdots + E_n F_n^\omega
\]

- \( E_i \) and \( F_i \) are regular expressions, \( \epsilon \not\in L(F_i) \)
NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We’ll describe such behaviors using $\omega$-regular languages

These can be described by $\omega$-regular expressions of the form:

$$E_1 F_1^\omega + \cdots + E_n F_n^\omega$$

- $E_i$ and $F_i$ are regular expressions, $\epsilon \notin L(F_i)$
- Union and concatenation work as they did before
NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We’ll describe such behaviors using \( \omega \text{-regular languages} \)

These can be described by \( \omega \text{-regular expressions} \) of the form:

\[
E_1 F_1^\omega + \cdots + E_n F_n^\omega
\]

- \( E_i \) and \( F_i \) are regular expressions, \( \epsilon \notin L(F_i) \)
- Union and concatenation work as they did before
- \( \omega \) denotes infinite repetition
NFAs and REs describe languages containing finite words.

Our transition systems describe infinite behaviors.

We’ll describe such behaviors using $\omega$-regular languages.

These can be described by $\omega$-regular expressions of the form:

$$E_1 F_1^\omega + \cdots + E_n F_n^\omega$$

- $E_i$ and $F_i$ are regular expressions, $\epsilon \not\in L(F_i)$
- Union and concatenation work as they did before.
- $\omega$ denotes infinite repetition.
- Like Kleene $\ast$, but ad infinitum.
Automata on Infinite Words (Review)

NFA : Regular :: **Nondeterministic Buchi Automata** : $\omega$-Regular

**Nondeterministic Buchi Automaton (NBA)**

A NBA $M$ is a tuple $(\Sigma, Q, Q_0, F, \delta)$, where:

- $\Sigma$ is an alphabet
- $Q$ is a finite set of states
- $Q_0 \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of accepting states
- $\delta \subseteq Q \times \Sigma \times Q$ is the transition function

The “syntax” is the same as NFAs; obviously the semantics is different
Let $w = a_0a_1\ldots$ be an infinite word in $\Sigma^\omega$
Let $w = a_0 a_1 \ldots$ be an infinite word in $\Sigma^\omega$

A run for $w$ in $A$ is an infinite sequence of states $q_0 \ldots q_{n-1}$ where:
Let \( w = a_0 a_1 \ldots \) be an infinite word in \( \Sigma^\omega \)

A run for \( w \) in \( A \) is an infinite sequence of states \( q_0 \ldots q_{n-1} \) where:

- \( q_0 \in Q_0 \)
Let \( w = a_0 a_1 \ldots \) be an infinite word in \( \Sigma^\omega \)

A **run** for \( w \) in \( A \) is an infinite sequence of states \( q_0 \ldots q_{n-1} \) where:

1. \( q_0 \in Q_0 \)
2. \( (q_i, a_i, q_{i+1}) \in \delta \) for all \( 0 \leq i \leq n \)
Let $w = a_0 a_1 \ldots$ be an infinite word in $\Sigma^\omega$

A run for $w$ in $A$ is an infinite sequence of states $q_0 \ldots q_{n-1}$ where:

- $q_0 \in Q_0$
- $(q_i, a_i, q_{i+1}) \in \delta$ for all $0 \leq i \leq n$

A run is **accepting** if $q_i \in F$ for **infinitely many indices** $i$:

$$\{ q \in Q \mid \forall i \geq 0, \exists j \geq i. q_j = q \} \cap F \neq \emptyset$$
Let \( w = a_0 a_1 \ldots \) be an infinite word in \( \Sigma^\omega \)

A run for \( w \) in \( A \) is an infinite sequence of states \( q_0 \ldots q_{n-1} \) where:

- \( q_0 \in Q_0 \)
- \( (q_i, a_i, q_{i+1}) \in \delta \) for all \( 0 \leq i \leq n \)

A run is **accepting** if \( q_i \in F \) for **infinitely many indices** \( i \):

\[
\{ q \in Q \mid \forall i \geq 0, \exists j \geq i . q_j = q \} \cap F \neq \emptyset
\]

A language is \( \omega \)-regular language iff it is recognizable by an NBA
Let $A$ be an NBA representing some computation
Let $A$ be an NBA representing some computation
Let $A_\phi$ be an NBA representing the specification
Let $A$ be an NBA representing some computation

Let $A_\phi$ be an NBA representing the specification

- $A_\phi$ describes the **allowed traces**
Let $A$ be an NBA representing some computation

Let $A_\phi$ be an NBA representing the specification
- $A_\phi$ describes the **allowed traces**
- Its language corresponds to “good” computations
Let $A$ be an NBA representing some computation

Let $A_\phi$ be an NBA representing the specification

- $A_\phi$ describes the **allowed traces**
- Its language corresponds to “good” computations

Then $A$ satisfies the specification $A_\phi$ exactly when:
Let $A$ be an NBA representing some computation

Let $A_\phi$ be an NBA representing the specification
  ▶ $A_\phi$ describes the **allowed traces**
  ▶ Its language corresponds to “good” computations

Then $A$ satisfies the specification $A_\phi$ exactly when:

$$L(A) \subseteq L(A_\phi)$$
Let $A$ be an NBA representing some computation

Let $A_\phi$ be an NBA representing the specification

$\bullet$ $A_\phi$ describes the **allowed traces**

$\bullet$ Its language corresponds to “good” computations

Then $A$ satisfies the specification $A_\phi$ exactly when:

$$L(A) \subseteq L(A_\phi)$$

The set of traces in $A$ is contained in the set of “good” computations
How do we check that \( L(A) \subseteq L(A_\phi) \)?

\[
L(A) \subseteq L(S) \iff L(A) \cap \overline{L(A_\phi)} = \emptyset
\]

In other words, \( A \) satisfies \( A_\phi \) if none of its traces is prohibited.
How do we check that $L(A) \subseteq L(A_\phi)$?

$$L(A) \subseteq L(S) \iff L(A) \cap \overline{L(A_\phi)} = \emptyset$$

In other words, $A$ satisfies $A_\phi$ if none of its traces is prohibited.

We can use closed NBA operations + emptiness check to do MC.
How do we check that $L(A) \subseteq L(A_\phi)$?

$$L(A) \subseteq L(S) \iff L(A) \cap \overline{L(A_\phi)} = \emptyset$$

In other words, $A$ satisfies $A_\phi$ if **none of its traces is prohibited**

We can use closed NBA operations + emptiness check to do MC

What about counterexamples?
How do we check that $L(A) \subseteq L(A_\phi)$?

$L(A) \subseteq L(S) \iff L(A) \cap \overline{L(A_\phi)} = \emptyset$

In other words, $A$ satisfies $A_\phi$ if none of its traces is prohibited.

We can use closed NBA operations + emptiness check to do MC.

What about counterexamples?

$\Rightarrow L(A) \cap \overline{L(A_\phi)} \neq \emptyset$ gives an $\omega$-regular language.
How do we check that $L(A) \subseteq L(A_\phi)$?

$$L(A) \subseteq L(S) \iff L(A) \cap \overline{L(A_\phi)} = \emptyset$$

In other words, $A$ satisfies $A_\phi$ if none of its traces is prohibited.

We can use closed NBA operations + emptiness check to do MC.

What about counterexamples?

- $L(A) \cap \overline{L(A_\phi)} \neq \emptyset$ gives an $\omega$-regular language.
- Any word in this language is a prohibited trace.
How do we check that $L(A) \subseteq L(A_ϕ)$?

$$L(A) \subseteq L(S) \iff L(A) \cap \overline{L(A_ϕ)} = \emptyset$$

In other words, $A$ satisfies $A_ϕ$ if none of its traces is prohibited

We can use closed NBA operations + emptiness check to do MC

What about counterexamples?

- $L(A) \cap \overline{L(A_ϕ)} \neq \emptyset$ gives an $\omega$-regular language
- Any word in this language is a prohibited trace
- We pick an arbitrary word, find an appropriate prefix
We would like to solve the LTL model checking problem:

Given a Kripke structure $M$ and LTL formula $\phi$, decide whether $M, \pi \models \phi$ for each $\pi$ starting in an initial state.
We would like to solve the LTL model checking problem:

Given a Kripke structure $M$ and LTL formula $\phi$, decide whether $M, \pi \models \phi$ for each $\pi$ starting in an initial state.

To do this, we’ll need to represent $M$ and $\phi$ as NBAs
We would like to solve the LTL model checking problem:

Given a Kripke structure $M$ and LTL formula $\phi$, decide whether $M, \pi \models \phi$ for each $\pi$ starting in an initial state.

To do this, we’ll need to represent $M$ and $\phi$ as NBAs.

Intuitively, this should pose no problem.
Automata-Theoretic LTL Checking

We would like to solve the LTL model checking problem:

Given a Kripke structure $M$ and LTL formula $\phi$, decide whether $M, \pi \models \phi$ for each $\pi$ starting in an initial state.

To do this, we’ll need to represent $M$ and $\phi$ as NBAs

Intuitively, this should pose no problem

- $M$ is a nondeterministic system over infinite paths
We would like to solve the LTL model checking problem:

Given a Kripke structure $M$ and LTL formula $\phi$, decide whether $M, \pi \models \phi$ for each $\pi$ starting in an initial state.

To do this, we’ll need to represent $M$ and $\phi$ as NBAs.

Intuitively, this should pose no problem

- $M$ is a nondeterministic system over infinite paths
- We’ve seen NBAs that “look like” LTL properties
We would like to solve the LTL model checking problem:

Given a Kripke structure $M$ and LTL formula $\phi$, decide whether $M, \pi \models \phi$ for each $\pi$ starting in an initial state.

To do this, we’ll need to represent $M$ and $\phi$ as NBAs.

Intuitively, this should pose no problem:
- $M$ is a nondeterministic system over infinite paths.
- We’ve seen NBAs that “look like” LTL properties.

However, this is the source of complexity in LTL model checking.
A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of atomic propositions $P$
- States $S$
- Initial states $I \subseteq S$
- Labeling $L : S \mapsto 2^P$
- Transition relation $R \subseteq S \times S$
### Kripke structure

A Kripke structure \( M = (P, S, I, L, R) \) consists of:
- Set of *atomic propositions* \( P \)
- States \( S \)
- Initial states \( I \subseteq S \)
- Labeling \( L : S \mapsto 2^P \)
- Transition relation \( R \subseteq S \times S \)

Recalling this definition, the main difference seems to be:

Transitions have no labels

The "natural" alphabet \( P \) labels states, not transitions

There are no accepting states
### Kripke structure

A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of *atomic propositions* $P$
- States $S$
- Initial states $I \subseteq S$
- Labeling $L : S \mapsto 2^P$
- Transition relation $R \subseteq S \times S$

Recalling this definition, the main difference seems to be:

- Transitions have no labels
A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of *atomic propositions* $P$
- States $S$
- Initial states $I \subseteq S$
- Labeling $L : S \rightarrow 2^P$
- Transition relation $R \subseteq S \times S$

Recalling this definition, the main difference seems to be:

- Transitions have no labels
- The “natural” alphabet $P$ labels states, not transitions
A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of atomic propositions $P$
- States $S$
- Initial states $I \subseteq S$
- Labeling $L : S \mapsto 2^P$
- Transition relation $R \subseteq S \times S$

Recalling this definition, the main difference seems to be:

- Transitions have no labels
- The “natural” alphabet $P$ labels states, not transitions
- There are no accepting states
We’re given a Kripke structure 
\[ M = (P, S, I, L, R) \]

We want NBA \( A = (\Sigma, Q, Q_0, F, \delta) \)

where:

\[
\begin{align*}
\text{▶} & \quad \forall (q; q') \in R \\
& \quad \text{if:} \\
\text{▶} & \quad 1. (q; q') \in R \\
& \quad \text{and} \\
& \quad L(q') = q \\
\text{▶} & \quad 2. q = \ell; q' \\
& \quad \text{and} \\
& \quad L(q') = \delta(q, \ell) \\
\text{▶} \quad Q = S \setminus \{q_0\}, \text{a distinguished initial state}
\end{align*}
\]

What about \( F \)?

Every execution "accepted" by the system, so 
\[ F = Q \]
We’re given a Kripke structure $M = (P, S, I, L, R)$

We want NBA $A = (\Sigma, Q, Q_0, F, \delta)$ where:

- $\Sigma = 2^P$
We’re given a Kripke structure
\[ M = (P, S, I, L, R) \]

We want NBA \( A = (\Sigma, Q, Q_0, F, \delta) \)
where:
- \( \Sigma = 2^P \)
- \( (q, \alpha, q') \in \delta \) if:
  - 1. \((q, q') \in R \) and \( L(q') = \alpha \)
We’re given a Kripke structure $M = (P, S, I, L, R)$

We want NBA $A = (\Sigma, Q, Q_0, F, \delta)$ where:

- $\Sigma = 2^P$
- $(q, \alpha, q') \in \delta$ if:
  1. $(q, q') \in R$ and $L(q') = \alpha$
  2. $q = \ell, q' \in I$ and $L(q') = \alpha$
We’re given a Kripke structure
\( M = (P, S, I, L, R) \)

We want NBA \( A = (\Sigma, Q, Q_0, F, \delta) \)

where:

- \( \Sigma = 2^P \)
- \( (q, \alpha, q') \in \delta \) if:
  1. \( (q, q') \in R \) and \( L(q') = \alpha \)
  2. \( q = \ell, q' \in I \) and \( L(q') = \alpha \)

- So \( Q = S \cup \{\ell\} \), a distinguished initial state
We’re given a Kripke structure
\( M = (P, S, I, L, R) \)

We want NBA \( A = (\Sigma, Q, Q_0, F, \delta) \)
where:

- \( \Sigma = 2^P \)
- \( (q, \alpha, q') \in \delta \) if:
  1. \( (q, q') \in R \) and \( L(q') = \alpha \)
  2. \( q = \ell, q' \in I \) and \( L(q') = \alpha \)

- So \( Q = S \cup \{\ell\} \), a distinguished initial state
- What about \( F \)?
We’re given a Kripke structure 
\( M = (P, S, I, L, R) \)

We want NBA \( A = (\Sigma, Q, Q_0, F, \delta) \) where:

- \( \Sigma = 2^P \)
- \( (q, \alpha, q') \in \delta \) if:
  1. \( (q, q') \in R \) and \( L(q') = \alpha \)
  2. \( q = \ell, q' \in I \) and \( L(q') = \alpha \)

- So \( Q = S \cup \{\ell\} \), a distinguished initial state
- What about \( F \)?
- Every execution “accepted” by the system, so \( F = Q \)
We’re given a Kripke structure
\[ M = (P, S, I, L, R) \]

We want NBA \( A = (\Sigma, Q, Q_0, F, \delta) \) where:

- \( \Sigma = 2^P \)
- \( (q, \alpha, q') \in \delta \) if:
  1. \( (q, q') \in R \) and \( L(q') = \alpha \)
  2. \( q = \ell, q' \in I \) and \( L(q') = \alpha \)

- So \( Q = S \cup \{\ell\} \), a distinguished initial state
- What about \( F \)?
- Every execution “accepted” by the system, so \( F = Q \)
The final piece: converting LTL to NBA
The final piece: converting LTL to NBA

The “leaves” of LTL formulas are propositional formulas over $P$
The final piece: converting LTL to NBA

The “leaves” of LTL formulas are propositional formulas over $P$

\[ \text{G F} (p \lor q) \quad \text{G} (\neg c_1 \lor \neg c_2) \quad \text{G} (p \rightarrow \text{F} q) \]
The final piece: converting LTL to NBA

The “leaves” of LTL formulas are propositional formulas over $P$

\[
\begin{align*}
G & F (p \lor q) \\
G & (\neg c_1 \lor \neg c_2) \\
G & (p \rightarrow F q)
\end{align*}
\]

We’ll use formulas over $P$ to represent alphabet symbolically
The final piece: converting LTL to NBA

The “leaves” of LTL formulas are propositional formulas over $P$

$$\mathbf{G} \mathbf{F} (p \lor q) \quad \mathbf{G} (\neg c_1 \lor \neg c_2) \quad \mathbf{G} (p \rightarrow \mathbf{F} q)$$

We’ll use formulas over $P$ to represent alphabet symbolically

For example, if we have a transition:

$$q_0 \quad p_0 \lor p_1 \quad q_1$$
The final piece: converting LTL to NBA

The “leaves” of LTL formulas are propositional formulas over $P$

\[ \mathbf{G} \mathbf{F} (p \lor q) \quad \mathbf{G} (\neg c_1 \lor \neg c_2) \quad \mathbf{G} (p \rightarrow \mathbf{F} q) \]

We’ll use formulas over $P$ to represent alphabet symbolically

For example, if we have a transition:

\[
\begin{align*}
q_0 & \quad p_0 \lor p_1 \quad p_1 \lor p_0 \quad q_1
\end{align*}
\]

Then this is shorthand for:

\[
\begin{align*}
\{p_0\} & \quad \{p_1\} \quad \{p_0, p_1\} \quad q_1
\end{align*}
\]
Let’s start with the next operator

\( \mathbf{X} p \)
LTL to NBA: Example (\(\mathbf{X}\) operator)

Let’s start with the next operator

\[ \mathbf{X} p \]

What is the corresponding NBA?
Let’s start with the next operator

\[ \mathbf{X} p \]

What is the corresponding NBA?
Let’s start with the next operator

\[ \mathbf{X} p \]

What is the corresponding NBA?

\[ \mathbf{X} p \]

- It doesn’t matter what the current state is
Let's start with the next operator

$$\mathbf{X} p$$

What is the corresponding NBA?

- It doesn’t matter what the current state is
- The next state must satisfy $$p$$
LTL to NBA: Example ($\mathbf{X}$ operator)

Let’s start with the next operator

$$\mathbf{X}p$$

What is the corresponding NBA?

- It doesn’t matter what the current state is
- The next state must satisfy $p$
- After that, any path suffices for acceptance
Now the until operator

\[ p_1 \mathbf{U} p_2 \]

- \( p_1 \)
- \( p_1 \)
- \( p_1 \)
- \( p_2 \)
- \( \text{any} \)
- \( \ldots \)

What is the corresponding NBA?
LTL to NBA: Example (U operator)

Now the until operator

\[ p_1 \mathbf{U} p_2 \]

What is the corresponding NBA?
LTL to NBA: Example (U operator)

Now the until operator

$p_1 \mathbf{U} p_2 \quad \rightarrow \quad p_1 \quad p_1 \quad p_1 \quad p_2 \quad \text{any} \quad \rightarrow \quad \cdots$

What is the corresponding NBA?

$p_1 \mathbf{U} p_2 \quad \rightarrow \quad q_0 \quad p_2 \quad q_1$

$p_1 \land \neg p_2 \quad \text{true}$
Now the until operator

\[ p_1 \mathbf{U} p_2 \]

What is the corresponding NBA?

- \( p_1 \) holds arbitrarily long in the beginning
Now the until operator

$p_1 \mathbf{U} p_2$

What is the corresponding NBA?

$p_1 \mathbf{U} p_2$

- $p_1$ holds arbitrarily long in the beginning
- To pass into accepting, $p_2$ must hold at some point
Now the until operator

$$p_1 \mathbf{U} p_2 \rightarrow p_1 \rightarrow p_1 \rightarrow p_1 \rightarrow p_2 \rightarrow \text{any}$$

What is the corresponding NBA?

$$p_1 \mathbf{U} p_2 \rightarrow q_0 \rightarrow p_2 \rightarrow q_1$$

- $p_1$ holds arbitrarily long in the beginning
- To pass into accepting, $p_2$ must hold at some point
- Afterwards, anything goes
$\mathbf{X}$ and $\mathbf{U}$ are sufficient to express $\mathbf{F}$, $\mathbf{G}$, $\mathbf{R}$

However, composing temporal operators is expensive in general.
In the worst case, the size of the NBA is exponential in $\mathcal{O}(|\phi|)!$.
This is the source of complexity in LTL model checking.
$X$ and $U$ are sufficient to express $F$, $G$, $R$

- $F \ p \Leftrightarrow true \ U \ p$

However, composing temporal operators is expensive in general.

In the worst case, the size of the NBA is exponential in $j_\phi$.

This is the source of complexity in LTL model checking.
X and U are sufficient to express F, G, R

- $F_p \Leftrightarrow \text{true} \quad U_p$
- $G_p \Leftrightarrow \neg F \quad \neg p$

However, composing temporal operators is expensive in general.
In the worst case, the size of the NBA is exponential in $|\phi|!$.

This is the source of complexity in LTL model checking.
X and U are sufficient to express F, G, R

- $F p \iff true \ U p$
- $G p \iff \neg F \neg p$
- $p_1 R p_2 \iff \neg (\neg p_1 U \neg p_2)$

However, composing temporal operators is expensive in general. In the worst case, the size of the NBA is exponential in $|\phi|$. This is the source of complexity in LTL model checking.
X and U are sufficient to express F, G, R

- $F p \iff \text{true } U p$
- $G p \iff \neg F \neg p$
- $p_1 R p_2 \iff \neg (\neg p_1 U \neg p_2)$

However, composing temporal operators is expensive in general.
**LTL to NBA: Remaining Operators**

\[ X \text{ and } U \text{ are sufficient to express } F, G, R \]

\[ \begin{align*}
F \ p & \iff \text{true } U \ p \\
G \ p & \iff \neg F \ \neg p \\
R \ p_1 \ p_2 & \iff \neg (\neg \ p_1 \ U \ \neg p_2)
\end{align*} \]

However, composing temporal operators is expensive in general

In the worst case, the size of the NBA is exponential in \(|\phi|\)!
X and U are sufficient to express F, G, R

- $F_p \iff true \ U \ p$
- $G_p \iff \neg F \ \neg p$
- $p_1 \ R \ p_2 \iff \neg (\neg p_1 \ U \ \neg p_2)$

However, composing temporal operators is expensive in general

In the worst case, the size of the NBA is exponential in $|\phi|$!

This is the source of complexity in LTL model checking
Given a Kripke structure $M$ and LTL $\phi$: 

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A\phi$.
2. Negate $\phi$ by building complement $A\neg\phi$.
   \textit{Note: Complement can blow up exponentially!}
   In practice, negate $\phi$ before building NBA.
3. Check emptiness of $L(A\setminus A\phi)$.
4. If not empty, return a word (prefix) $w \in L(A\setminus A\phi)$.

Worst case complexity: $O(|M|^2|\phi|)$.

Intersection $A_1 \setminus A_2$ produces automaton of size $|A_1| |A_2|$. 

LTL to NBA produces $A\phi$ of size $2^{|\phi|}$. 

Emptiness check is depth-first search – linear time.
Given a Kripke structure $M$ and LTL $\phi$:
1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$
Summary: Automata-Based LTL Model Checking

Given a Kripke structure $M$ and LTL $\phi$:
1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$
2. Negate $\phi$ by building complement $\overline{A_\phi}$

Worst case complexity: $O(j_M j_\phi^2)$
Summary: Automata-Based LTL Model Checking

Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$

2. Negate $\phi$ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
Summary: Automata-Based LTL Model Checking

Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$

2. Negate $\phi$ by building complement $\overline{A_\phi}$
   
   - **Note**: Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA

Worst case complexity: $O(j_M^2 j_\phi)$
Summary: Automata-Based LTL Model Checking

Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$

2. Negate $\phi$ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA

3. Check emptiness of $L(A \cap \overline{A_\phi})$

Worst case complexity: $O(|M|^2|\phi|)$

Intersection $A_1 \cap A_2$ produces automaton of size $|A_1| \times |A_2|$

LTL to NBA produces $A_\phi$ of size $2^{|\phi|}$

Emptiness check is depth-first search – linear time
Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$
2. Negate $\phi$ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA
3. Check emptiness of $L(A \cap \overline{A_\phi})$
4. If not empty, return a word (prefix) $w \in L(A \cap \overline{A_\phi})$

Worst case complexity: $O(j_M j^2 j_\phi)$

**Intersection** $A_1 \cap A_2$ produces automaton of size $j_A j A_2$

**LTL to NBA** produces $A_\phi$ of size $2 j_\phi$

**Emptiness check** is depth-first search – linear time
Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$

2. Negate $\phi$ by building complement $\overline{A_\phi}$
   - Note: Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA

3. Check emptiness of $L(A \cap \overline{A_\phi})$

4. If not empty, return a word (prefix) $w \in L(A \cap \overline{A_\phi})$

Worst case complexity: $O(|M| \cdot 2^{|\phi|})$
Given a Kripke structure $M$ and LTL $\phi$:  
1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$
2. Negate $\phi$ by building complement $\overline{A_\phi}$
   - **Note:** Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA
3. Check emptiness of $L(A \cap \overline{A_\phi})$
4. If not empty, return a word (prefix) $w \in L(A \cap \overline{A_\phi})$

Worst case complexity: $O(|M| \cdot 2^{|\phi|})$
   - Intersection $A_1 \cap A_2$ produces automaton of size $|A_1| \cdot |A_2|$
Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$

2. Negate $\phi$ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA

3. Check emptiness of $L(A \cap \overline{A_\phi})$

4. If not empty, return a word (prefix) $w \in L(A \cap \overline{A_\phi})$

Worst case complexity: $O(|M| \cdot 2^{|\phi|})$

- Intersection $A_1 \cap A_2$ produces automaton of size $|A_1| \cdot |A_2|$
- LTL to NBA produces $A_\phi$ of size $2^{|\phi|}$
Given a Kripke structure $M$ and LTL $\phi$:

1. Convert $M$ into Buchi automaton $A$, $\phi$ into $A_\phi$
2. Negate $\phi$ by building complement $A_\phi$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate $\phi$ before building NBA
3. Check emptiness of $L(A \cap A_{\overline{\phi}})$
4. If not empty, return a word (prefix) $w \in L(A \cap A_{\overline{\phi}})$

Worst case complexity: $O(|M| \cdot 2^{|\phi|})$
- Intersection $A_1 \cap A_2$ produces automaton of size $|A_1| \cdot |A_2|$
- LTL to NBA produces $A_\phi$ of size $2^{|\phi|}$
- Emptiness check is depth-first search – linear time
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A}_\phi$

Once we have the NBA, all we do is depth-first search
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$

Once we have the NBA, all we do is depth-first search

In practice, the search can proceed with the construction
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$.

Once we have the NBA, all we do is depth-first search.

In practice, the search can proceed with the construction:

1. Construct property automaton $A_\phi$ first.
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$

Once we have the NBA, all we do is depth-first search

In practice, the search can proceed with the construction

1. Construct property automaton $A_\phi$ first
2. Begin taking intersection at initial states of $A$

This works because bugs are often easy to find – software is buggy!
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$

Once we have the NBA, all we do is depth-first search

In practice, the search can proceed with the construction

1. Construct property automaton $A_\phi$ first
2. Begin taking intersection at initial states of $A$
3. Perform DFS incrementally at each step

In many cases, counterexamples are found early before DFS backtracks too much

This works because bugs are often easy to find – software is buggy!
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$

Once we have the NBA, all we do is depth-first search

In practice, the search can proceed with the construction

1. Construct property automaton $A_\phi$ first
2. Begin taking intersection at initial states of $A$
3. Perform DFS incrementally at each step
4. Whenever DFS needs a state that hasn’t been built, add it
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$.

Once we have the NBA, all we do is depth-first search.

In practice, the search can proceed with the construction:
1. Construct property automaton $A_\phi$ first.
2. Begin taking intersection at initial states of $A$.
3. Perform DFS incrementally at each step.
4. Whenever DFS needs a state that hasn’t been built, add it.

In many cases, counterexamples are found early before DFS backtracks too much.
On-the-fly model checking

The expensive part of this algorithm is in constructing $A \cap \overline{A_\phi}$

Once we have the NBA, all we do is depth-first search

In practice, the search can proceed with the construction

1. Construct property automaton $A_\phi$ first
2. Begin taking intersection at initial states of $A$
3. Perform DFS incrementally at each step
4. Whenever DFS needs a state that hasn’t been built, add it

In many cases, counterexamples are found early before DFS backtracks too much

This works because bugs are often easy to find – software is buggy!
Symbolic MC algorithms are used in most “industrial strength” tools
Symbolic Model Checking

Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem
Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem

**Key idea:** Logical formulas can represent sets of states compactly
Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem

Key idea: Logical formulas can represent sets of states compactly
  - Represent set of reachable states, transitions as predicates
Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem

**Key idea:** Logical formulas can represent sets of states compactly

- Represent set of reachable states, transitions as predicates
- Characterize temporal operators as *predicate transformers*
Symbolic Model Checking

Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem

**Key idea:** Logical formulas can represent sets of states compactly
► Represent set of reachable states, transitions as predicates
► Characterize temporal operators as **predicate transformers**
► Apply transformers until we represent all satisfying states
Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem

**Key idea:** Logical formulas can represent sets of states compactly
- Represent set of reachable states, transitions as predicates
- Characterize temporal operators as *predicate transformers*
- Apply transformers until we represent all satisfying states
- Use *fixpoints* to determine when this happens
Symbolic MC algorithms are used in most “industrial strength” tools

Effective way to mitigate state explosion problem

**Key idea:** Logical formulas can represent sets of states compactly
- Represent set of reachable states, transitions as predicates
- Characterize temporal operators as **predicate transformers**
- Apply transformers until we represent all satisfying states
- Use **fixpoints** to determine when this happens

Exploit efficient representations of formulas to further improve cost
We’re given a Kripke structure $M = (P, S, I, L, R)$
We’re given a Kripke structure $M = (P, S, I, L, R)$

Given a set of states $X$, we can think of it as:

- A subset of $S$
- Or, a predicate (Boolean function) on $S$:
  \[ X(s) = \begin{cases} 1 & s \in X \\ 0 & s \notin X \end{cases} \]

These representations are equivalent. We’ll use them interchangeably.
We’re given a Kripke structure $M = (P, S, I, L, R)$

Given a set of states $X$, we can think of it as:

- A subset of $S$, $X \subseteq S$
We’re given a Kripke structure $M = (P, S, I, L, R)$

Given a set of states $X$, we can think of it as:

- A subset of $S$, $X \subseteq S$
- Or, a predicate *(Boolean function)* on $S$, $X : S \mapsto \{0, 1\}$
Symbolic Transition Systems

We’re given a Kripke structure $M = (P, S, I, L, R)$

Given a set of states $X$, we can think of it as:

- A subset of $S$, $X \subseteq S$
- Or, a predicate (Boolean function) on $S$, $X : S \rightarrow \{0, 1\}$
- If $s \in X$, then the predicate $X$ maps $s$ to 1: $X(s) = 1 \iff s \in X$
We’re given a Kripke structure $M = (P, S, I, L, R)$

Given a set of states $X$, we can think of it as:

- A subset of $S$, $X \subseteq S$
- Or, a predicate (Boolean function) on $S$, $X : S \mapsto \{0, 1\}$
- If $s \in X$, then the predicate $X$ maps $s$ to 1: $X(s) = 1 \iff s \in X$

These representations are equivalent

We’ll use them interchangably
Symbolic Transition Systems

We’ll represent states by their atomic propositions:
We’ll represent states by their atomic propositions:

- Need to assume that states are uniquely determined by their propositions
We’ll represent states by their atomic propositions:

- Need to assume that states are uniquely determined by their propositions
- i.e., for any \( s, s' \in S \) where \( s \neq s' \), \( L(s) \neq L(s') \)
We’ll represent states by their atomic propositions:

- Need to assume that states are uniquely determined by their propositions
- i.e., for any $s, s' \in S$ where $s \neq s'$, $L(s) \neq L(s')$
- Then if $L(s) = p_1, \ldots, p_n$, we’ll refer to $s$ by writing:
  $$p_1 \land \cdots \land p_n$$
Symbolic Transition Systems

We’ll represent states by their atomic propositions:

- Need to assume that states are uniquely determined by their propositions
- I.e., for any $s, s' \in S$ where $s \neq s'$, $L(s) \neq L(s')$
- Then if $L(s) = p_1, \ldots, p_n$, we’ll refer to $s$ by writing:
  $$p_1 \land \cdots \land p_n$$
- If $\phi$ is a formula over atomic propositions, then
  $$\phi$$ refers to the set $\{s \in S \mid s \models \phi\}$
We’ll represent states by their atomic propositions:

- Need to assume that states are uniquely determined by their propositions
- I.e., for any \( s, s' \in S \) where \( s \neq s' \), \( L(s) \neq L(s') \)
- Then if \( L(s) = p_1, \ldots, p_n \), we’ll refer to \( s \) by writing:
  \[ p_1 \land \cdots \land p_n \]
- If \( \phi \) is a formula over atomic propositions, then
  \( \phi \) refers to the set \( \{ s \in S \mid s \models \phi \} \)

Recall: this is similar to how we treated assertions in Hoare logic
Symbolic Transition Systems

We also represent transitions as predicates

To refer to “next state”, prime the proposition symbols

So the predicate $\left(p_1^:\neg p_2\right)^\left(p_1'^:p_2'\right)$:

1. Begins in the state where $p_1$ is true and $p_2$ is false
2. Ends in the state where both $p_1$ and $p_2$ are true
Symbolic Transition Systems

We also represent transitions as predicates

Transitions reference **ordered pairs** of states \((s, s')\)
Symbolic Transition Systems

We also represent transitions as predicates.

Transitions reference \textbf{ordered pairs} of states \((s, s')\).

The transition relation is just a set of these pairs, so as a predicate,

\[
R(s, s') = 1 \iff (s, s') \in R
\]
We also represent transitions as predicates

Transitions reference **ordered pairs** of states \((s, s')\)

The transition relation is just a set of these pairs, so as a predicate,

\[
R(s, s') = 1 \iff (s, s') \in R
\]

We’ll represent transition predicates using atomic propositions:
Symbolic Transition Systems

We also represent transitions as predicates

Transitions reference **ordered pairs** of states \((s, s')\)

The transition relation is just a set of these pairs, so as a predicate,

\[
R(s, s') = 1 \iff (s, s') \in R
\]

We’ll represent transition predicates using atomic propositions:

- To refer to “next state”, prime the proposition symbols
We also represent transitions as predicates

Transitions reference **ordered pairs** of states \((s, s')\)

The transition relation is just a set of these pairs, so as a predicate,

\[
R(s, s') = 1 \iff (s, s') \in R
\]

We’ll represent transition predicates using atomic propositions:

- To refer to “next state”, prime the proposition symbols
- So the predicate \((p_1 \land \neg p_2) \land (p'_1 \land p'_2)\):
Symbolic Transition Systems

We also represent transitions as predicates

Transitions reference **ordered pairs** of states \((s, s')\)

The transition relation is just a set of these pairs, so as a predicate,

\[ R(s, s') = 1 \iff (s, s') \in R \]

We’ll represent transition predicates using atomic propositions:

1. To refer to “next state”, prime the proposition symbols
2. So the predicate \((p_1 \land \neg p_2) \land (p'_1 \land p'_2)\):
   1. Begins in the state where \(p_1\) is true and \(p_2\) is false
We also represent transitions as predicates.

Transitions reference **ordered pairs** of states \((s, s')\).

The transition relation is just a set of these pairs, so as a predicate,

\[
R(s, s') = 1 \iff (s, s') \in R
\]

We’ll represent transition predicates using atomic propositions:

- To refer to “next state”, prime the proposition symbols.
- So the predicate \((p_1 \land \neg p_2) \land (p'_1 \land p'_2)\):
  1. Begins in the state where \(p_1\) is true and \(p_2\) is false.
  2. Ends in the state where both \(p_1\) and \(p_2\) are true.
Example: Symbolic Representation

Atomic propositions:
- \( v_0 = 0 \)
- \( v_1 = 1 \)

Transition relation:
- \( f(00; 01) \)
- \( f(01; 10) \)
- \( f(10; 11) \)
- \( f(11; 00) \)

Symbolically:
- \( (v_0 = 0 \land v_1 = 0) \land (v_0' = 0 \land v_1' = 1) \)
- \( (v_0 = 0 \land v_1 = 1) \land (v_0' = 1 \land v_1' = 1) \)
- \( (v_0 = 1 \land v_1 = 0) \land (v_0' = 1 \land v_1' = 0) \)
- \( (v_0 = 1 \land v_1 = 1) \land (v_0' = 0 \land v_1' = 0) \)

Initial state:
- \( v_0 = 0 \land v_1 = 1 \)
Example: Symbolic Representation

Atomic propositions:

\[
\begin{align*}
\text{Initial state:} & \quad v_0 = 0 \land v_1 = 0 \\
& \land v'_{0} = 0 \land v'_{1} = 1 \\
& \land v_{0} = 0 \land v_{1} = 1 \\
& \land v'_{0} = 1 \land v'_{1} = 1 \\
& \land v_{0} = 1 \land v_{1} = 0 \\
& \land v'_{0} = 1 \land v'_{1} = 0 \\
& \land v_{0} = 1 \land v_{1} = 1 \\
& \land v'_{0} = 0 \land v'_{1} = 0
\end{align*}
\]

Transition relation:

\[
\begin{align*}
& (00; 01) \\
& (01; 10) \\
& (10; 11) \\
& (11; 00)
\end{align*}
\]
Example: Symbolic Representation

Atomic propositions:
\[ \{v_0 = 0, v_1 = 1\} \]
Example: Symbolic Representation

Atomic propositions:

\[ \{v_0 = 0, v_1 = 1\} \]

Transition relation:
Example: Symbolic Representation

Atomic propositions:
\[ \{v_0 = 0, v_1 = 1\} \]

Transition relation:
\[ \{(00, 01), (01, 10), (10, 11), (11, 00)\} \]
Example: Symbolic Representation

Atomic propositions:
\[ \{ v_0 = 0, v_1 = 1 \} \]

Transition relation:
\[ \{(00, 01), (01, 10), (10, 11), (11, 00)\} \]

Symbolically:
Atomic propositions:
\[ \{v_0 = 0, v_1 = 1\} \]

Transition relation:
\[ \{(00, 01), (01, 10), (10, 11), (11, 00)\} \]

Symbolically:
\[
\begin{align*}
(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \\
\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \\
\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \\
\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\end{align*}
\]
Atomic propositions:
\[ \{ v_0 = 0, v_1 = 1 \} \]

Transition relation:
\[ \{(00, 01), (01, 10), (10, 11), (11, 00)\} \]

Symbolically:
\[ (v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \]
\[ \lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \]
\[ \lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \]
\[ \lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0) \]

Initial state:
Atomic propositions:
\[ \{v_0 = 0, v_1 = 1\} \]

Transition relation:
\[ \{(00, 01), (01, 10), (10, 11), (11, 00)\} \]

Symbolically:
\[
\begin{align*}
(v_0 &= 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \\
\lor (v_0 &= 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \\
\lor (v_0 &= 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \\
\lor (v_0 &= 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\end{align*}
\]

Initial state: \( v_0 = 0 \land v_0 = 1 \)
Example: Symbolic Representation

Symbolic transitions:

\[(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1)\]
\[\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0)\]
\[\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1)\]
\[\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)\]

Initial state: \(v_0 = 0 \land v_0 = 1\)
Example: Symbolic Representation

Symbolic transitions:

\[(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1)\]
\[\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0)\]
\[\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1)\]
\[\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)\]

Initial state: \(v_0 = 0 \land v_0 = 1\)

The transitions are a predicate

\[\psi_R(v_0, v_1, v'_0, v'_1)\]
Example: Symbolic Representation

Symbolic transitions:

\[(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1)\]
\[\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0)\]
\[\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1)\]
\[\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)\]

Initial state: \(v_0 = 0 \land v_0 = 1\)

The transitions are a predicate

\[\psi_R(v_0, v_1, v'_0, v'_1)\]

- Over four Boolean \(\{0, 1\}\) variables
Example: Symbolic Representation

Symbolic transitions:

- \((v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1)\)
- \((v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0)\)
- \((v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1)\)
- \((v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)\)

Initial state: \(v_0 = 0 \land v_0 = 1\)

The transitions are a predicate

\[\psi_R(v_0, v_1, v'_0, v'_1)\]

- Over four Boolean \(\{0, 1\}\) variables
- Variables completely determine state of system
Example: Symbolic Representation

Symbolic transitions:

\[(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1)\]
\[\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0)\]
\[\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1)\]
\[\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)\]

Initial state: \(v_0 = 0 \land v_0 = 1\)

The transitions are a predicate

\[\psi_R(v_0, v_1, v'_0, v'_1)\]

- Over four Boolean \(\{0, 1\}\) variables
- Variables completely determine state of system

Same for the initial state: \(\psi_I(v_0, v_1)\)
GCD program:

```c
while (n1 != n2) {
    if (n1 > n2) {
        n1 := n1 - n2;
    } else {
        n2 := n2 - n1;
    }
}
```

Atomic propositions:

- If \( n_1 = x_1 x_2 \mathbb{Z} \quad g \)
- If \( n_2 = x_1 x_2 \mathbb{Z} \quad g \)

Each state corresponds to a unique pair of these values.

We want the initial states to have positive \( n_1, n_2 \):

- \( n_1 = 0 \)
- \( n_2 = 0 \)

What about the transition relation?

\[
(n_1 > n_2) \Rightarrow (n_1' = n_1, n_2' = n_2) \\
(n_2 > n_1) \Rightarrow (n_1' = n_2 - n_1, n_2' = n_2) \\
(n_1 = n_2) \Rightarrow (n_1' = n_1, n_2' = n_2)
\]
Example: Symbolic Representation

Atomic propositions:

GCD program:

```plaintext
while \((n_1 \neq n_2)\) {
    if \((n_1 > n_2)\) {
        n_1 := n_1 - n_2;
    } else {
        n_2 := n_2 - n_1;
    }
}
```
Example: Symbolic Representation

GCD program:

```
while (n₁ ≠ n₂) {
    if (n₁ > n₂) {
        n₁ := n₁ - n₂;
    } else {
        n₂ := n₂ - n₁;
    }
}
```

Atomic propositions:

\[ \{ n₁ = x \mid x \in \mathbb{Z} \} \cup \{ n₂ = x \mid x \in \mathbb{Z} \} \]
Example: Symbolic Representation

GCD program:

```plaintext
while (n1 ≠ n2) {
  if (n1 > n2) {
    n1 := n1 - n2;
  } else {
    n2 := n2 - n1;
  }
}
```

Atomic propositions:

\[ \{ n_1 = x \mid x \in \mathbb{Z} \} \cup \{ n_2 = x \mid x \in \mathbb{Z} \} \]

Each state corresponds to unique pair of these
Example: Symbolic Representation

GCD program:

```c
while (n1 ≠ n2) {
  if (n1 > n2) {
    n1 := n1 - n2;
  } else {
    n2 := n2 - n1;
  }
}
```

Atomic propositions:

\[
\{ n_1 = x \mid x \in \mathbb{Z} \} \cup \{ n_2 = x \mid x \in \mathbb{Z} \}
\]

Each state corresponds to unique pair of these.

We want the initial states to have positive \( n_1, n_2 \):
Example: Symbolic Representation

GCD program:

```c
while (n_1 \neq n_2) {
    if (n_1 > n_2) {
        n_1 := n_1 - n_2;
    } else {
        n_2 := n_2 - n_1;
    }
}
```

Atomic propositions:

\[
\{ n_1 = x \mid x \in \mathbb{Z} \} \cup \{ n_2 = x \mid x \in \mathbb{Z} \}
\]

Each state corresponds to unique pair of these

We want the initial states to have positive \( n_1, n_2 \):

\[
0 \leq n_1 \land 0 \leq n_2 = \{ s \in S \mid s \models 0 \leq n_1 \land 0 \leq n_2 \}
\]
Example: Symbolic Representation

GCD program:

```plaintext
while (n₁ ≠ n₂) {
    if (n₁ > n₂) {
        n₁ := n₁ - n₂;
    } else {
        n₂ := n₂ - n₁;
    }
}
```

Atomic propositions:

$$\{ n₁ = x \mid x \in \mathbb{Z} \} \cup \{ n₂ = x \mid x \in \mathbb{Z} \}$$

Each state corresponds to unique pair of these

We want the initial states to have positive \(n₁, n₂\):

$$0 \leq n₁ \land 0 \leq n₂ = \{ s \in S \mid s \models 0 \leq n₁ \land 0 \leq n₂ \}$$

What about the transition relation?
Example: Symbolic Representation

GCD program:

```plaintext
while(n_1 \neq n_2) {
    if(n_1 > n_2) {
        n_1 := n_1 - n_2;
    } else {
        n_2 := n_2 - n_1;
    }
}
```

Atomic propositions:

\[ \{n_1 = x \mid x \in \mathbb{Z}\} \cup \{n_2 = x \mid x \in \mathbb{Z}\} \]

Each state corresponds to unique pair of these

We want the initial states to have positive \( n_1, n_2 \):

\[ 0 \leq n_1 \land 0 \leq n_2 = \{s \in S \mid s \models 0 \leq n_1 \land 0 \leq n_2\} \]

What about the transition relation?

\[ (n_1 > n_2 \land n'_1 = n_1 - n_2 \land n'_2 = n_2) \]
Example: Symbolic Representation

GCD program:

```c
while (n1 ≠ n2) {
  if (n1 > n2) {
    n1 := n1 - n2;
  } else {
    n2 := n2 - n1;
  }
}
```

Atomic propositions:

\[
\{ n_1 = x \mid x \in \mathbb{Z} \} \cup \{ n_2 = x \mid x \in \mathbb{Z} \}
\]

Each state corresponds to unique pair of these

We want the initial states to have positive \( n_1, n_2 \):

\[
0 \leq n_1 \land 0 \leq n_2 = \{ s \in S \mid s \models 0 \leq n_1 \land 0 \leq n_2 \}
\]

What about the transition relation?

\[
(n_1 > n_2 \land n_1' = n_1 - n_2 \land n_2' = n_2) \lor (n_2 > n_1 \land n_2' = n_2 - n_1 \land n_1' = n_1)
\]
Example: Symbolic Representation

GCD program:

\[
\begin{align*}
\text{while}(n_1 \neq n_2) \{ \\
\text{if}(n_1 > n_2) \{ \\
\quad n_1 := n_1 - n_2; \\
\} \text{ else } \{ \\
\quad n_2 := n_2 - n_1; \\
\}
\}
\end{align*}
\]

Atomic propositions:
\[
\{n_1 = x \mid x \in \mathbb{Z}\} \cup \{n_2 = x \mid x \in \mathbb{Z}\}
\]

Each state corresponds to unique pair of these

We want the initial states to have positive \(n_1, n_2\):
\[
0 \leq n_1 \land 0 \leq n_2 = \{s \in S \mid s \models 0 \leq n_1 \land 0 \leq n_2\}
\]

What about the transition relation?
\[
\begin{align*}
(n_1 > n_2 & \land n_1' = n_1 - n_2 \land n_2' = n_2) \\
\lor (n_2 > n_1 & \land n_2' = n_2 - n_1 \land n_1' = n_1) \\
\lor (n_1 = n_2 & \land n_1' = n_1 \land n_2' = n_2)
\end{align*}
\]
Predicate Transformers

**Observe:** We can “lift” the transition relation to sets of states:

\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. X(s') \land R(s, s') \} \\
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. X(s) \land R(s, s') \}
\]
Predicate Transformers

**Observe:** We can “lift” the transition relation to sets of states:

\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. X(s') \land R(s, s') \} \\
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. X(s) \land R(s, s') \}
\]

These functions are **predicate transformers**
**Predicate Transformers**

**Observe:** We can “lift” the transition relation to sets of states:

\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. X(s') \land R(s, s') \}\]

\[
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. X(s) \land R(s, s') \}\]

These functions are **predicate transformers**

Consider the transition relation from GCD:

\[
\tau(n_1, n_2, n'_1, n'_2) \iff (n_1 > n_2 \land n'_1 = n_1 - n_2 \land n'_2 = n_2) \lor
\]

\[
(n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1) \lor
\]

\[
(n_1 = n_2 \land n'_1 = n_1 \land n'_2 = n_2)
\]
Observe: We can “lift” the transition relation to sets of states:

\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. X(s') \land R(s, s') \} \\
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. X(s) \land R(s, s') \}
\]

These functions are **predicate transformers**

Consider the transition relation from GCD:

\[
\tau(n_1, n_2, n'_1, n'_2) \iff (n_1 > n_2 \land n'_1 = n_1 - n_2 \land n'_2 = n_2) \lor \\
(n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1) \lor \\
(n_1 = n_2 \land n'_1 = n_1 \land n'_2 = n_2)
\]

What are the post-states of \( n_1 = 5 \land n_2 = 15 \)?
**Predicate Transformers**

**Observe:** We can “lift” the transition relation to sets of states:

\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. \text{X}(s') \land R(s, s') \}\n\]

\[
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. \text{X}(s) \land R(s, s') \}\n\]

These functions are **predicate transformers**

Consider the transition relation from GCD:

\[
\tau(n_1, n_2, n'_1, n'_2) \iff (n_1 > n_2 \land n'_1 = n_1 - n_2 \land n'_2 = n_2) \lor \n\]
\[
(n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1) \lor \n\]
\[
(n_1 = n_2 \land n'_1 = n_1 \land n'_2 = n_2) \n\]

What are the post-states of \( n_1 = 5 \land n_2 = 15 \)?

\[
\exists n_1, n_2. n_1 = 5 \land n_2 = 15 \land \tau(n_1, n_2, n'_1, n'_2) \n\]
**Predicate Transformers**

**Observe:** We can “lift” the transition relation to sets of states:

\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. X(s') \land R(s, s') \}\]

\[
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. X(s) \land R(s, s') \}\]

These functions are **predicate transformers**

Consider the transition relation from GCD:

\[
\tau(n_1, n_2, n'_1, n'_2) \iff (n_1 > n_2 \land n'_1 = n_1 - n_2 \land n'_2 = n_2) \lor (n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1) \lor (n_1 = n_2 \land n'_1 = n_1 \land n'_2 = n_2)
\]

What are the post-states of \( n_1 = 5 \land n_2 = 15 \)?

\[
\exists n_1, n_2. n_1 = 5 \land n_2 = 15 \land \tau(n_1, n_2, n'_1, n'_2)
\]

\[
\iff \exists n_1, n_2. n_1 = 5 \land n_2 = 15 \land n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1
\]
Observe: We can “lift” the transition relation to sets of states:
\[
\text{Pre}(X) = \{ s \in S \mid \exists s' \in S. X(s') \land R(s, s') \} \\
\text{Post}(X) = \{ s' \in S \mid \exists s \in X. X(s) \land R(s, s') \}
\]

These functions are **predicate transformers**

Consider the transition relation from GCD:
\[
\tau(n_1, n_2, n'_1, n'_2) \iff \\
\begin{align*}
(n_1 > n_2 \land n'_1 = n_1 - n_2 \land n'_2 = n_2) & \lor \\
(n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1) & \lor \\
(n_1 = n_2 \land n'_1 = n_1 \land n'_2 = n_2)
\end{align*}
\]

What are the post-states of \( n_1 = 5 \land n_2 = 15 \)?
\[
\exists n_1, n_2. n_1 = 5 \land n_2 = 15 \land \tau(n_1, n_2, n'_1, n'_2) \\
\iff \exists n_1, n_2. n_1 = 5 \land n_2 = 15 \land n_2 \geq n_1 \land n'_2 = n_2 - n_1 \land n'_1 = n_1 \\
\iff n'_2 = 10 \land n'_1 = 5
\]
Let $\tau : 2^S \mapsto 2^S$ be a predicate transformer.
Fixpoints

Let $\tau : 2^S \mapsto 2^S$ be a predicate transformer

- $\tau$ is **monotonic** iff $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$
Fixpoints

Let \( \tau : 2^S \mapsto 2^S \) be a predicate transformer

- \( \tau \) is **monotonic** iff \( P \subseteq Q \) implies \( \tau(P) \subseteq \tau(Q) \)

- A **fixpoint** of \( \tau \) is a predicate (set) \( Z \) where \( \tau(Z) = Z \)
Let $\tau : 2^S \rightarrow 2^S$ be a predicate transformer

- $\tau$ is **monotonic** iff $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$

- A **fixpoint** of $\tau$ is a predicate (set) $Z$ where $\tau(Z) = Z$

- A **least fixpoint** of $\tau$, written $\mu Z. \tau(Z)$, is:
Let $\tau : 2^S \mapsto 2^S$ be a predicate transformer

- $\tau$ is **monotonic** if $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$

- A **fixpoint** of $\tau$ is a predicate (set) $Z$ where $\tau(Z) = Z$

- A **least fixpoint** of $\tau$, written $\mu Z. \tau(Z)$, is:
  1. A fixpoint of $\tau$, so $\tau(\mu Z. \tau(Z)) = Z$
Fixpoints

Let $\tau : 2^S \rightarrow 2^S$ be a predicate transformer

- $\tau$ is **monotonic** iff $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$

- A **fixpoint** of $\tau$ is a predicate (set) $Z$ where $\tau(Z) = Z$

- A **least fixpoint** of $\tau$, written $\mu Z. \tau(Z)$, is:
  1. A fixpoint of $\tau$, so $\tau(\mu Z. \tau(Z)) = Z$
  2. A subset of any other fixpoint
Let \( \tau : 2^S \rightarrow 2^S \) be a predicate transformer

- \( \tau \) is **monotonic** iff \( P \subseteq Q \) implies \( \tau(P) \subseteq \tau(Q) \)

- A **fixpoint** of \( \tau \) is a predicate (set) \( Z \) where \( \tau(Z) = Z \)

- A **least fixpoint** of \( \tau \), written \( \mu Z. \tau(Z) \), is:
  1. A fixpoint of \( \tau \), so \( \tau(\mu Z. \tau(Z)) = Z \)
  2. A subset of any other fixpoint

- A **greatest fixpoint** of \( \tau \), written \( \nu Z. \tau(Z) \), is:
  1. A fixpoint of \( \tau \), so \( \tau(\nu Z. \tau(Z)) = Z \)
  2. A superset of any other fixpoint
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

\[
\text{Z} : (\text{Z}) = \bigcap f \text{Z} \cup f \text{Z} \bigcap \text{Z} \\
\text{Z} : (\text{Z}) = \bigcup f \text{Z} \cup f \text{Z} \bigcap \text{Z}
\]
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

\[ \mu \exists \exists Z. \tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \]
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

- \( \mu Z. \tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \)
- \( \nu Z. \tau(Z) = \bigcup \{ Z \mid \tau(Z) \supseteq Z \} \)
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

- $\mu Z. \tau(Z) = \bigcap \{Z \mid \tau(Z) \subseteq Z\}$
- $\nu Z. \tau(Z) = \bigcup \{Z \mid \tau(Z) \supseteq Z\}$

Now assume that $\mathcal{S}$ is finite and $\tau$ is monotonic. We have:
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

1. $\mu Z. \tau(Z) = \bigcap \{Z \mid \tau(Z) \subseteq Z\}$
2. $\nu Z. \tau(Z) = \bigcup \{Z \mid \tau(Z) \supseteq Z\}$

Now assume that $S$ is finite and $\tau$ is monotonic. We have:

1. $\forall i. \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset)$
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

- \[ \mu Z. \tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \]
- \[ \nu Z. \tau(Z) = \bigcup \{ Z \mid \tau(Z) \supseteq Z \} \]

Now assume that \( S \) is finite and \( \tau \) is monotonic. We have:

1. \[ \forall i. \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset) \text{ (monotonicity)} \]
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

\[
\mu Z. \tau (Z) = \cap \{ Z \mid \tau (Z) \subseteq Z \}
\]
\[
\nu Z. \tau (Z) = \cup \{ Z \mid \tau (Z) \supseteq Z \}
\]

Now assume that \( S \) is finite and \( \tau \) is monotonic. We have:

1. \( \forall i. \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset) \) (monotonicity)
2. \( \exists i. \tau^i(\emptyset) = \tau^{i+1}(\emptyset) \)
Fixpoint Theory

### Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

\[ \mu Z.\tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \]
\[ \nu Z.\tau(Z) = \bigcup \{ Z \mid \tau(Z) \supseteq Z \} \]

Now assume that \( S \) is finite and \( \tau \) is monotonic. We have:

1. \( \forall i.\tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset) \) (monotonicity)
2. \( \exists i.\tau^i(\emptyset) = \tau^{i+1}(\emptyset) \) (1 and \( S \) finite)
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

\[ \mu Z. \tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \]

\[ \nu Z. \tau(Z) = \bigcup \{ Z \mid \tau(Z) \supseteq Z \} \]

Now assume that \( S \) is finite and \( \tau \) is monotonic. We have:

1. \( \forall i. \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset) \) (monotonicity)
2. \( \exists i. \tau^i(\emptyset) = \tau^{i+1}(\emptyset) \) (1 and \( S \) finite)
3. If \( Z \) a fixpoint of \( \tau \), then \( \forall i. \tau^i(\emptyset) \subseteq Z \)
A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

\[ \mu Z. \tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \]

\[ \nu Z. \tau(Z) = \bigcup \{ Z \mid \tau(Z) \supseteq Z \} \]

Now assume that \( S \) is finite and \( \tau \) is monotonic. We have:

1. \( \forall i. \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset) \) (monotonicity)
2. \( \exists i. \tau^i(\emptyset) = \tau^{i+1}(\emptyset) \) (1 and \( S \) finite)
3. If \( Z \) a fixpoint of \( \tau \), then \( \forall i. \tau^i(\emptyset) \subseteq Z \) (def. fixpoint)
Theorem (Tarski, 1955)

A monotonic predicate transformer always has a least and greatest fixpoint. Moreover, they are given by:

- \( \mu Z. \tau(Z) = \bigcap \{ Z \mid \tau(Z) \subseteq Z \} \)
- \( \nu Z. \tau(Z) = \bigcup \{ Z \mid \tau(Z) \supseteq Z \} \)

Now assume that \( S \) is finite and \( \tau \) is monotonic. We have:

1. \( \forall i. \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset) \) (monotonicity)
2. \( \exists i. \tau^i(\emptyset) = \tau^{i+1}(\emptyset) \) (1 and \( S \) finite)
3. If \( Z \) a fixpoint of \( \tau \), then \( \forall i. \tau^i(\emptyset) \subseteq Z \) (def. fixpoint)

So, we obtain \( \mu Z. \tau(Z) \) by applying \( \tau \) repeatedly to \( \emptyset \)
We have a simple algorithm that gives us fixpoints
Computing Fixpoints

We have a simple algorithm that gives us fixpoints

```plaintext
function lfp(τ) {
    Q := false;
    Q' := τ(Q);
    while (Q ≠ Q') {
        Q := Q';
        Q := τ(Q');
    }
    return Q;
}
```
We have a simple algorithm that gives us fixpoints

**function lfp(τ)**

\[
\begin{align*}
Q & := \text{false}; \\
Q’ & := \tau(Q); \\
\text{while}(Q \neq Q’) & \{ \\
Q & := Q’; \\
Q & := \tau(Q’); \\
\} \\
\text{return } Q;
\end{align*}
\]

**function gfp(τ)**

\[
\begin{align*}
Q & := \text{true}; \\
Q’ & := \tau(Q); \\
\text{while}(Q \neq Q’) & \{ \\
Q & := Q’; \\
Q & := \tau(Q’); \\
\} \\
\text{return } Q;
\end{align*}
\]
We can define the semantics of CTL in terms of fixpoints and predicate transformers.
We can define the semantics of CTL in terms of fixpoints and predicate transformers

- Least fixpoints correspond to *eventualities*
We can define the semantics of CTL in terms of fixpoints and predicate transformers

- Least fixpoints correspond to **eventualities**
- Greatest fixpoints correspond to **global assertions**
We can define the semantics of CTL in terms of fixpoints and predicate transformers

- Least fixpoints correspond to **eventualities**
- Greatest fixpoints correspond to **global assertions**

Identify a CTL formula $f$ with the predicate $\{ s \in S \mid M, s \models f \}$

We can define the semantics of CTL in terms of fixpoints and predicate transformers

- Least fixpoints correspond to **eventualities**
- Greatest fixpoints correspond to **global assertions**

Identify a CTL formula $f$ with the predicate $\{s \in S \mid M, s \models f\}$

Our “base” operator is $\text{EX } \phi$, given by the predicate transformer:

$$\tau(v) = \exists v'. \phi(v') \land R(v, v')$$
We can define the semantics of CTL in terms of fixpoints and predicate transformers

- Least fixpoints correspond to **eventualities**
- Greatest fixpoints correspond to **global assertions**

Identify a CTL formula \( f \) with the predicate \( \{ s \in S \mid M, s \models f \} \)

Our “base” operator is \( \mathbf{EX} \phi \), given by the predicate transformer:

\[
\tau(v) = \exists v'. \phi(v') \land R(v, v')
\]

Then we define a sufficient set of operators using fixpoints:
We can define the semantics of CTL in terms of fixpoints and predicate transformers

- Least fixpoints correspond to **eventualities**
- Greatest fixpoints correspond to **global assertions**

Identify a CTL formula $f$ with the predicate $\{s \in S \mid M, s \models f\}$

Our “base” operator is $\text{EX } \phi$, given by the predicate transformer:

$$\tau(v) = \exists v'. \phi(v') \land R(v, v')$$

Then we define a sufficient set of operators using fixpoints:

- $\text{EG } \phi = \nu Z. \phi \land \text{EX } Z$
We can define the semantics of CTL in terms of fixpoints and predicate transformers:

- Least fixpoints correspond to **eventualities**
- Greatest fixpoints correspond to **global assertions**

Identify a CTL formula $f$ with the predicate $\{ s \in S \mid M, s \models f \}$

Our “base” operator is $\textbf{EX} \phi$, given by the predicate transformer:

$$\tau(v) = \exists v'. \phi(v') \land R(v, v')$$

Then we define a sufficient set of operators using fixpoints:

- $\textbf{EG} \phi = \nu Z. \phi \land \textbf{EX} Z$
- $\textbf{E} (\phi_1 \cup \phi_2) = \mu Z. \phi_2 \lor (\phi_1 \land \textbf{EX} Z)$
Example: $E(p \mathbf{U} q)$

$$\tau(Z) = q \lor (p \land E X Z)$$

We've reached the fixpoint $Z$.
Example: \( E (p \mathbf{U} q) \)

\[
\tau(Z) = q \lor (p \land \mathbf{EX} Z)
\]

\[
\text{First compute } \tau(\text{false}) = \tau(\emptyset)
\]
Example: $E(p \ U \ q)$

\[ \tau(Z) = q \lor (p \land \textbf{EX} \ Z) \]

\[ \{p\} \xrightarrow{\text{false}} s_1 \xrightarrow{p} s_2 \xrightarrow{q} \{q\} \]

\[ \{p\} \xrightarrow{} s_0 \xrightarrow{} \{\} \xrightarrow{} s_3 \xrightarrow{} \{\} \]

Then $\tau^1(\text{false}) = \tau(\{s_2\})$
Example: $E ( p \ U q )$

$$\tau(Z) = q \lor (p \land \textbf{EX} Z)$$

Then $\tau^2(\text{false}) = \tau(\{s_1, s_2\})$
Example: $E (p U q)$

$$\tau(Z) = q \lor (p \land \textbf{EX} Z)$$

Then $\tau^3(\text{false}) = \tau(\{s_0, s_1, s_2\})$
Example: $E (p \ U \ q)$

$$\tau(Z) = q \lor (p \land \textbf{EX} Z)$$

---

Then $\tau^4(\text{false}) = \tau(\{s_0, s_1, s_2\}) = \tau^3(\text{false})$
Example: \( E(p \ U \ q) \)

\[
\tau(Z) = q \lor (p \land \text{EX } Z)
\]

Then \( \tau^4(false) = \tau(s_0, s_1, s_2) = \tau^3(false) \)

We’ve reached the fixpoint \( \mu Z. \tau(Z) \)
Checking $\text{EX } \phi$ is fairly straightforward
Checking \( \textbf{EX} \ \phi \) is fairly straightforward

Recall: We want to know if all initial states \( I \) satisfy \( \textbf{EX} \ \phi \)
Checking $\textbf{EX} \quad \phi$ is fairly straightforward

Recall: We want to know if all initial states $I$ satisfy $\textbf{EX} \quad \phi$

Our predicate transformer was: $\exists v'. \phi(v') \land R(v, v')$
Checking $\textbf{EX} \; \phi$ is fairly straightforward

Recall: We want to know if all initial states $I$ satisfy $\textbf{EX} \; \phi$

Our predicate transformer was: $\exists v'. \phi(v') \land R(v, v')$

Then we check that the following formula is valid:

$$\psi_I(v) \rightarrow (\exists v'. \phi(v') \land R(v, v'))$$
Checking **EX** $\phi$ is fairly straightforward

Recall: We want to know if all initial states $I$ satisfy **EX** $\phi$

Our predicate transformer was: $\exists v'.\phi(v') \land R(v, v')$

Then we check that the following formula is valid:

$$\psi_I(v) \rightarrow (\exists v'.\phi(v') \land R(v, v'))$$

If it is, then $\phi$ holds at all initial states
Symbolic Model Checking (EX): Example

Suppose we want to check \( \text{EX} \ v_0 = 1 \)

\[
\psi_I(v_0, v_1) \iff v_0 = 0 \land v_1 = 0
\]

\[
\psi_R(v_0, v_1, v'_0, v'_1) \iff
(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1)
\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0)
\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1)
\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\]
Symbolic Model Checking (EX): Example

Suppose we want to check $\text{EX } v_0 = 1$

We apply the transformer for $\text{EX}$:

\[
\psi_I(v_0, v_1) \iff v_0 = 0 \land v_1 = 0
\]

\[
\psi_R(v_0, v_1, v'_0, v'_1) \iff
\begin{align*}
& (v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \\
\lor & (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \\
\lor & (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \\
\lor & (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\end{align*}
\]
Symbolic Model Checking (\textbf{EX}): Example

Suppose we want to check $\textbf{EX} \ v_0 = 1$

We apply the transformer for $\textbf{EX}$:

$$\exists v'_0, v'_1. v'_0 = 1 \land \psi_R(v_0, v_1, v'_0, v'_1)$$

\[\psi_I(v_0, v_1) \iff v_0 = 0 \land v_1 = 0\]

\[\psi_R(v_0, v_1, v'_0, v'_1) \iff\]

\[\begin{align*}
(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \\
\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \\
\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \\
\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\end{align*}\]
Symbolic Model Checking (\( \mathbf{EX} \)): Example

Suppose we want to check \( \mathbf{EX} \ v_0 = 1 \)

We apply the transformer for \( \mathbf{EX} \) :

\[
\exists v'_0, v'_1. v'_0 = 1 \land \psi_R(v_0, v_1, v'_0, v'_1)
\]

Then conjoin the initial states:

\[
v_0 = 0 \land v_1 = 0 \land \\
\exists v'_0, v'_1. v'_0 = 1 \land \psi_R(v_0, v_1, v'_0, v'_1)
\]

\[
\psi_I(v_0, v_1) \iff v_0 = 0 \land v_1 = 0
\]

\[
\psi_R(v_0, v_1, v'_0, v'_1) \iff \\
(v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \\
\lor (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \\
\lor (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \\
\lor (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\]
Symbolic Model Checking ($\mathbf{EX}$): Example

Suppose we want to check $\mathbf{EX} \; v_0 = 1$

We apply the transformer for $\mathbf{EX}$:

$$\exists v' \land v'_0 \land v'_1 = 1 \land \psi_R(v_0, v_1, v'_0, v'_1)$$

Then conjoin the initial states:

$$v_0 = 0 \land v_1 = 0 \land v'_0 = 1 \land \psi_R(v_0, v_1, v'_0, v'_1)$$

This formula is false, so there are no states that satisfy

\[
\psi_I(v_0, v_1) \iff v_0 = 0 \land v_1 = 0
\]

\[
\psi_R(v_0, v_1, v'_0, v'_1) \iff
\begin{align*}
& (v_0 = 0 \land v_1 = 0 \land v'_0 = 0 \land v'_1 = 1) \\
\lor & (v_0 = 0 \land v_1 = 1 \land v'_0 = 1 \land v'_1 = 0) \\
\lor & (v_0 = 1 \land v_1 = 0 \land v'_0 = 1 \land v'_1 = 1) \\
\lor & (v_0 = 1 \land v_1 = 1 \land v'_0 = 0 \land v'_1 = 0)
\end{align*}
\]
We have that $\textbf{EG } \phi = \nu Z.\phi \land \textbf{EX } Z$
We have that $\textbf{EG } \phi = \nu Z. \phi \wedge \textbf{EX } Z$

So to check $\textbf{EG } \phi$: 

1. Find the fixpoint of $\nu Z. \phi \wedge \textbf{EX } Z$

2. Check the validity of $I$!
We have that $\text{EG } \phi = \nu Z.\phi \land \text{EX } Z$

So to check $\text{EG } \phi$:
1. Find the fixpoint $\psi$ of $\tau = \nu Z.\phi \land \text{EX } Z$
We have that $\text{EG} \phi = \nu Z.\phi \land \text{EX} \ Z$

So to check $\text{EG} \phi$:
1. Find the fixpoint $\psi$ of $\tau = \nu Z.\phi \land \text{EX} \ Z$
2. Check the validity of $\psi \models \phi$
We have that $\mathbf{EG} \, \phi = \nu Z.\phi \land \mathbf{EX} \, Z$

So to check $\mathbf{EG} \, \phi$:
1. Find the fixpoint $\psi$ of $\tau = \nu Z.\phi \land \mathbf{EX} \, Z$
2. Check the validity of $\psi_I \rightarrow \psi$

We know that we can compute greatest fixpoints by:
Symbolic Model Checking (EG)

We have that $\textbf{EG} \, \phi = \nu Z. \phi \land \textbf{EX} \, Z$

So to check $\textbf{EG} \, \phi$:
1. Find the fixpoint $\psi$ of $\tau = \nu Z. \phi \land \textbf{EX} \, Z$
2. Check the validity of $\psi_I \rightarrow \psi$

We know that we can compute greatest fixpoints by:
1. Applying the predicate transformer to $true$
We have that $\textbf{EG} \phi = \nu Z . \phi \land \textbf{EX} Z$

So to check $\textbf{EG} \phi$:
1. Find the fixpoint $\psi$ of $\tau = \nu Z . \phi \land \textbf{EX} Z$
2. Check the validity of $\psi_I \rightarrow \psi$

We know that we can compute greatest fixpoints by:
1. Applying the predicate transformer to $\textit{true}$
2. Repeating, until the predicate doesn’t change
Symbolic Model Checking (EG)

We have that $\textbf{EG} \phi = \nu Z.\phi \land \textbf{EX} Z$

So to check $\textbf{EG} \phi$:

1. Find the fixpoint $\psi$ of $\tau = \nu Z.\phi \land \textbf{EX} Z$
2. Check the validity of $\psi_I \rightarrow \psi$

We know that we can compute greatest fixpoints by:

1. Applying the predicate transformer to $true$
2. Repeating, until the predicate doesn’t change

But before we can do this, must show $\nu Z.\phi \land \textbf{EX} Z$ is monotonic
We have that $\textbf{E} (\phi_1 \textbf{U} \phi_2) = \mu Z.\phi_2 \lor (\phi_1 \land \textbf{EX} Z)$
Symbolic Model Checking ($E (\phi_1 U \phi_2)$)

We have that $E (\phi_1 U \phi_2) = \mu Z. \phi_2 \lor (\phi_1 \land EX Z)$

We proceed exactly as we did for $EG$, but compute $lfp$ instead
We have that \( E (\phi_1 \ U \phi_2) = \mu Z. \phi_2 \lor (\phi_1 \land EX Z) \)

We proceed exactly as we did for \( EG \), but compute \( lfp \) instead

Notice: this algorithm is very similar to the explicit-state one
We have that $E(\phi_1 U \phi_2) = \mu Z. \phi_2 \lor (\phi_1 \land EX Z)$

We proceed exactly as we did for $EG$, but compute $lfp$ instead.

Notice: this algorithm is very similar to the explicit-state one.

1. Compute the set of states satisfying the CTL formula.
We have that $E (\phi_1 U \phi_2) = \mu Z. \phi_2 \lor (\phi_1 \land EX Z)$

We proceed exactly as we did for $EG$, but compute $lfp$ instead.

Notice: this algorithm is very similar to the explicit-state one:

1. Compute the set of states satisfying the CTL formula.
2. Check that all initial states are in the result.
We have that $E (\phi_1 U \phi_2) = \mu Z . \phi_2 \lor (\phi_1 \land EX Z)$

We proceed exactly as we did for $EG$, but compute $lfp$ instead.

Notice: this algorithm is very similar to the explicit-state one.

1. Compute the set of states satisfying the CTL formula.
2. Check that all initial states are in the result.

But what have we gained by doing it this way?
Efficient encodings for symbolic model checking

- Binary decision diagrams: concise, canonical representations of Boolean functions
- Bounded propositional encodings
- Reducing MC problems to SAT instances