Automated Program Verification and Testing 15414/15614 Fall 2016 Lecture 20: Explicit-State Model Checking, Part 2

Matt Fredrikson
mfredrik@cs.cmu.edu

November 17, 2016

Today's Lecture

LTL Model Checking

Today's Lecture

LTL Model Checking

Algorithm based on automata operations

Today's Lecture

LTL Model Checking

Algorithm based on automata operations

- Refresher on basic automata theory

Today's Lecture

LTL Model Checking

Algorithm based on automata operations

- Refresher on basic automata theory
- Introduce automata for languages of infinite words

Today's Lecture

LTL Model Checking

Algorithm based on automata operations

- Refresher on basic automata theory
- Introduce automata for languages of infinite words
- See how to apply them to model checking

Finite Automata: Refresher

A Nondeterministic Finite Automaton (NFA) is a tuple $(Q, \Sigma, \delta, Q_0, F)$:

Finite Automata: Refresher

A Nondeterministic Finite Automaton (NFA) is a tuple $(Q, \Sigma, \delta, Q_0, F)$:

- Q is a finite set of states; Q_0 initial states, F final

Finite Automata: Refresher

A Nondeterministic Finite Automaton (NFA) is a tuple $(Q, \Sigma, \delta, Q_0, F)$:

- Q is a finite set of states; Q_0 initial states, F final
- Σ is a finite alphabet

Finite Automata: Refresher

A Nondeterministic Finite Automaton (NFA) is a tuple $(Q, \Sigma, \delta, Q_0, F)$:

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

Finite Automata: Refresher

A Nondeterministic Finite Automaton (NFA) is a tuple $(Q, \Sigma, \delta, Q_0, F)$:

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

An automaton is **deterministic** (a DFA) if:

$$\forall a \in \Sigma.(q, a, q') \in \delta \land (q, a, q'') \in \delta \Rightarrow q = q''$$

Finite Automata: Refresher

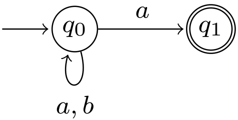
A Nondeterministic Finite Automaton (NFA) is a tuple $(Q, \Sigma, \delta, Q_0, F)$:

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

An automaton is **deterministic** (a DFA) if:

$$\forall a \in \Sigma.(q, a, q') \in \delta \wedge (q, a, q'') \in \delta \Rightarrow q = q''$$

Example:



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \ldots a_n \in \Sigma^*$ a finite word

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \dots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite sequence of states $q_0 \dots q_{n-1}$ where:

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \dots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite sequence of states $q_0 \dots q_{n-1}$ where:

- $q_0 \in Q_0$

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \ldots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite 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 $0 \leq i \leq n$

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \ldots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite 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 it ends in a final state, e.g., $q_n \in F$

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \ldots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite 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 it ends in a final state, e.g., $q_n \in F$

The word w is **accepted** by A if it has an accepting run

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \dots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite sequence of states $q_0 \dots q_{n-1}$ 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 it ends in a final state, e.g., $q_n \in F$

The word w is **accepted** by A if it has an accepting run

The **language of** A, denoted $L(A)$, is the subset of Σ^* it accepts:

$$L(A) = \{w \in \Sigma^* \mid \exists \text{ accepting run for } w\}$$

Finite Automata & Languages

Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA, $w = a_0 \dots a_n \in \Sigma^*$ a finite word

A **run** for w in A is a finite sequence of states $q_0 \dots q_{n-1}$ 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 it ends in a final state, e.g., $q_n \in F$

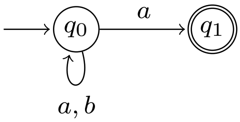
The word w is **accepted** by A if it has an accepting run

The **language of** A, denoted $L(A)$, is the subset of Σ^* it accepts:

$$L(A) = \{w \in \Sigma^* \mid \exists \text{ accepting run for } w\}$$

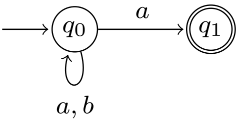
Every NFA can be converted to a DFA accepting the same language

Example



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

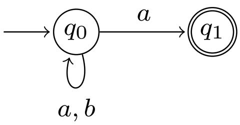
Example



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is

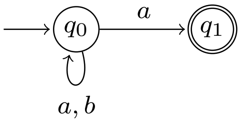
Example



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is **accepted**

Example

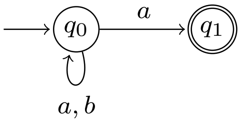


$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is **accepted**

$abababaa$ is

Example

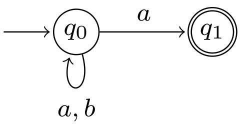


$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is **accepted**

$abababaa$ is **accepted**

Example



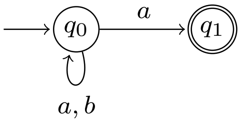
$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is **accepted**

$abababaa$ is **accepted**

$aaaaaaab$ is

Example



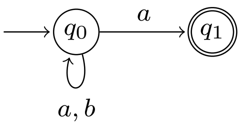
$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is **accepted**

$abababaa$ is **accepted**

$aaaaaaab$ is **rejected**

Example



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

$aaaaaaaa$ is **accepted**

$abababaa$ is **accepted**

$aaaaaaab$ is **rejected**

The language of this automaton is:

$L(A) = \{w \mid w$contains arbitrary sequence of a, b ending with $a\}$

Equivalence & Emptiness

When two NFAs represent the same language, we say they're **equivalent**

Equivalence & Emptiness

When two NFAs represent the same language, we say they're **equivalent**

A central issue in automata theory is the **emptiness problem**

Equivalence & Emptiness

When two NFAs represent the same language, we say they're **equivalent**

A central issue in automata theory is the **emptiness problem**

Given an NFA A, decide whether $L(A) = \varnothing$

Equivalence & Emptiness

When two NFAs represent the same language, we say they're **equivalent**

A central issue in automata theory is the **emptiness problem**

Given an NFA A, decide whether $L(A) = \varnothing$

This is equivalent to reachability:

$$L(A) \neq \varnothing \text{ iff } \exists q_0, q_f . q_f \text{ reachable from } q_0$$

Equivalence & Emptiness

When two NFAs represent the same language, we say they're **equivalent**

A central issue in automata theory is the **emptiness problem**

Given an NFA A, decide whether $L(A) = \varnothing$

This is equivalent to reachability:

$$L(A) \neq \varnothing \text{ iff } \exists q_0, q_f . q_f \text{ reachable from } q_0$$

This can be decided in $O(|A|)$ by depth-first search

Regular Languages

The languages recognized by NFAs are called **regular**

Regular Languages

The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular Languages

The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular languages are also represented by **regular expressions**:

Regular Languages

The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular languages are also represented by **regular expressions**:

- $\underline{\varnothing}$ is the RE denoting the empty language

Regular Languages

The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular languages are also represented by **regular expressions**:

- $\underline{\varnothing}$ is the RE denoting the empty language
- $\underline{\epsilon}$ is the RE denoting the language with the empty word

Regular Languages

The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular languages are also represented by **regular expressions**:

- $\underline{\varnothing}$ is the RE denoting the empty language
- $\underline{\epsilon}$ is the RE denoting the language with the empty word
- If E is an RE, then E^* denotes the finite repetitions of E

Regular Languages

The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular languages are also represented by **regular expressions**:

- $\underline{\varnothing}$ is the RE denoting the empty language
- $\underline{\epsilon}$ is the RE denoting the language with the empty word
- If E is an RE, then E^* denotes the finite repetitions of E
- If E_1, E_2 are REs, then $E_1 + E_2$ denotes union of their languages

Regular Languages

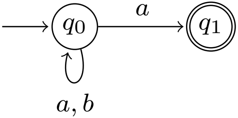
The languages recognized by NFAs are called **regular**

Regular languages contain **finite words**

Regular languages are also represented by **regular expressions**:

- $\underline{\varnothing}$ is the RE denoting the empty language
- $\underline{\epsilon}$ is the RE denoting the language with the empty word
- If E is an RE, then E^* denotes the finite repetitions of E
- If E_1, E_2 are REs, then $E_1 + E_2$ denotes union of their languages
- If E_1, E_2 are REs, then $E_1 E_2$ denotes their concatenation

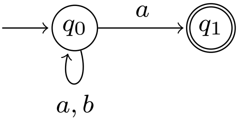
Example



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

The language of this automaton is:

Example



$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, Q_0 = \{q_0\}, F = \{q_1\}$$

The language of this automaton is:

$$L(A) = (a + b)^* a$$

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*:

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition
- $E_1 + E_2$:

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition
- $E_1 + E_2$: closed under union

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition
- $E_1 + E_2$: closed under union
- $E_1 E_2$:

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition
- $E_1 + E_2$: closed under union
- $E_1 E_2$: closed under concatenation

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition
- $E_1 + E_2$: closed under union
- $E_1 E_2$: closed under concatenation

They're also closed under intersection and complement

$$\text{If } L, L_1, L_2 \text{ are regular languages, so are } L_1 \cap L_2, \Sigma^* \setminus L$$

Properties of Regular Languages

The syntax of regular expressions implies several useful facts

- E^*: closed under finite repetition
- $E_1 + E_2$: closed under union
- $E_1 E_2$: closed under concatenation

They're also closed under intersection and complement

$$\text{If } L, L_1, L_2 \text{ are regular languages, so are } L_1 \cap L_2, \Sigma^* \setminus L$$

Given NFAs representing a language, we can construct NFAs corresponding to the application of these operations

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We'll describe such behaviors using ω**-regular languages**

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We'll describe such behaviors using ω**-regular languages**

These can be described by ω**-regular expressions** of the form:

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

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We'll describe such behaviors using ω-**regular languages**

These can be described by ω-**regular expressions** of the form:

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

- E_i and F_i are regular expressions, $\epsilon \notin L(F_i)$

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We'll describe such behaviors using ω**-regular languages**

These can be described by ω**-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

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We'll describe such behaviors using ω**-regular languages**

These can be described by ω**-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
- ω denotes **infinite repetition**

Languages of Infinite Words

NFAs and REs describe languages containing finite words

Our transition systems describe infinite behaviors

We'll describe such behaviors using ω**-regular languages**

These can be described by ω**-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
- ω denotes **infinite repetition**
- Like Kleene $*$, but ad infinitum

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\ldots\}$$

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\ldots\}$$

What about the empty word ϵ?

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\ldots\}$$

What about the empty word ϵ? $\epsilon^\omega = \epsilon$

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \dots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\dots\}$$

What about the empty word ϵ? $\epsilon^\omega = \epsilon$

Given an infinite word w, $w^\omega =$

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\ldots\}$$

What about the empty word ϵ? $\epsilon^\omega = \epsilon$

Given an infinite word w, $w^\omega = w$

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\ldots\}$$

What about the empty word ϵ? $\epsilon^\omega = \epsilon$

Given an infinite word w, $w^\omega = w$

We'll lift ω to finite languages $L \subseteq \Sigma^*$ as well:

$$L^\omega = \{w_1 w_2 w_3 \ldots \mid w_i \in L\}$$

Infinite Repetition ω

For a word ab, we know that $(ab)^*$ denotes the set

$$\{ab, abab, ababab, \ldots\}$$

What does $(ab)^\omega$ denote?

$$(ab)^\omega = \{ababababab\ldots\}$$

What about the empty word ϵ? $\epsilon^\omega = \epsilon$

Given an infinite word w, $w^\omega = w$

We'll lift ω to finite languages $L \subseteq \Sigma^*$ as well:

$$L^\omega = \{w_1 w_2 w_3 \ldots \mid w_i \in L\}$$

If L doesn't contain ϵ, L^ω is an infinite language

Example

How do we write mutual exclusion as an ω-regular expression?

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2$$

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2$$

First, we need to define the alphabet

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2$$

First, we need to define the alphabet

- Need to reason about the set of all propositions that might hold

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg crit_1 \vee \neg crit_2$$

First, we need to define the alphabet

- Need to reason about the set of all propositions that might hold
- Setting $\Sigma = 2^P$ (the atomic propositions) seems reasonable

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2$$

First, we need to define the alphabet

- Need to reason about the set of all propositions that might hold
- Setting $\Sigma = 2^P$ (the atomic propositions) seems reasonable
- In this case, $P = \{\mathit{crit}_1, \mathit{crit}_2\}$

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg crit_1 \vee \neg crit_2$$

First, we need to define the alphabet

- Need to reason about the set of all propositions that might hold
- Setting $\Sigma = 2^P$ (the atomic propositions) seems reasonable
- In this case, $P = \{crit_1, crit_2\}$

Then symbols are $\{\}, \{crit_1\}, \{crit_1, crit_2\}, \ldots$

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2$$

First, we need to define the alphabet

- Need to reason about the set of all propositions that might hold
- Setting $\Sigma = 2^P$ (the atomic propositions) seems reasonable
- In this case, $P = \{\mathit{crit}_1, \mathit{crit}_2\}$

Then symbols are $\{\}, \{\mathit{crit}_1\}, \{\mathit{crit}_1, \mathit{crit}_2\}, \ldots$

Our expression is:

Example

How do we write mutual exclusion as an ω-regular expression?

Recall, this was the safety property (invariant):

$$\mathbf{G}\ \neg \mathit{crit}_1 \vee \neg \mathit{crit}_2$$

First, we need to define the alphabet

- Need to reason about the set of all propositions that might hold
- Setting $\Sigma = 2^P$ (the atomic propositions) seems reasonable
- In this case, $P = \{\mathit{crit}_1, \mathit{crit}_2\}$

Then symbols are $\{\}, \{\mathit{crit}_1\}, \{\mathit{crit}_1, \mathit{crit}_2\}, \ldots$

Our expression is:

$$(\{\} + \{\mathit{crit}_1\} + \{\mathit{crit}_2\})^\omega$$

Automata on Infinite Words

NFA : Regular :: : ω-Regular

Automata on Infinite Words

NFA : Regular :: **Nondeterministic Buchi Automata** : ω-Regular

Nondeterministic Buchi Automaton (NBA)

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

- Σ 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

Automata on Infinite Words

NFA : Regular :: **Nondeterministic Buchi Automata** : ω-Regular

Nondeterministic Buchi Automaton (NBA)

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

- Σ 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

Buchi Automata: Infinite Runs & Acceptance

Let $w = a_0 a_1 \ldots$ be an infinite word in Σ^ω

Buchi Automata: Infinite Runs & Acceptance

Let $w = a_0 a_1 \ldots$ be an infinite word in Σ^ω

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

Buchi Automata: Infinite Runs & Acceptance

Let $w = a_0 a_1 \ldots$ be an infinite word in Σ^ω

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

- $q_0 \in Q_0$

Buchi Automata: Infinite Runs & Acceptance

Let $w = a_0 a_1 \ldots$ be an infinite word in Σ^ω

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 $0 \leq i \leq n$

Buchi Automata: Infinite Runs & Acceptance

Let $w = a_0 a_1 \ldots$ be an infinite word in Σ^ω

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:

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

Buchi Automata: Infinite Runs & Acceptance

Let $w = a_0 a_1 \ldots$ be an infinite word in Σ^ω

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

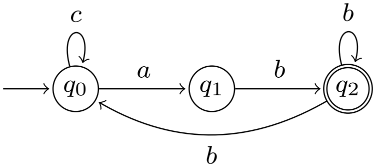
- $q_0 \in Q_0$
- $(q_i, a_i, q_{i+1}) \in \delta$ for all $0 \le i \le n$

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

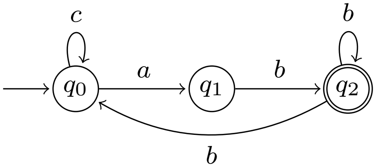
$$\{q \in Q \mid \forall i \ge 0, \exists j \ge i . q_j = q\} \cap F \neq \varnothing$$

A language is ω-regular language iff it is recognizable by an NBA

Example

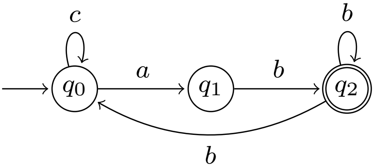


Example



What runs does the word c^ω have?

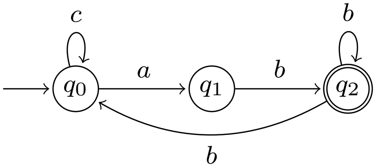
Example



What runs does the word c^ω have?

$$q_1^\omega$$

Example

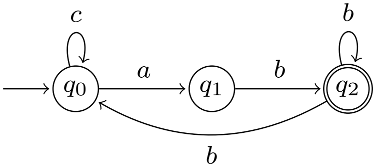


What runs does the word c^ω have?

$$q_1^\omega$$

What about ab^ω?

Example



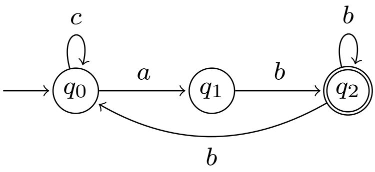
What runs does the word c^ω have?

$$q_1^\omega$$

What about ab^ω?

$$q_1 q_2 q_3^\omega$$

Example



What runs does the word c^ω have?

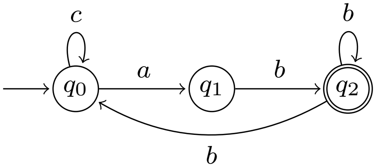
$$q_1^\omega$$

What about ab^ω?

$$q_1 q_2 q_3^\omega$$

Is $(cabb)^\omega$ accepted?

Example



What runs does the word c^ω have?

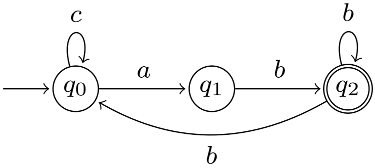
$$q_1^\omega$$

What about ab^ω?

$$q_1 q_2 q_3^\omega$$

Is $(cabb)^\omega$ accepted? What is its run?

Example



What runs does the word c^ω have?

$$q_1^\omega$$

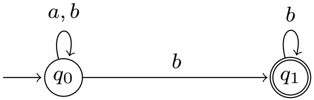
What about ab^ω?

$$q_1 q_2 q_3^\omega$$

Is $(cabb)^\omega$ accepted? What is its run?

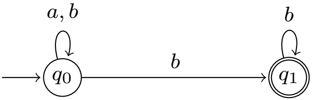
$$(q_1 q_1 q_2 q_3)^\omega$$

Example



What ω-regular expression does this accept?

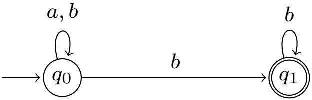
Example



What ω-regular expression does this accept?

$$(a+b)^* b^\omega$$

Example

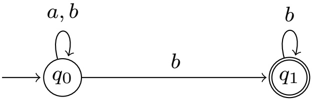


What ω-regular expression does this accept?

$$(a+b)^*b^\omega$$

What does it mean?

Example



What ω-regular expression does this accept?

$$(a + b)^* b^\omega$$

What does it mean? a **occurs only finitely many times**

Example: No send after read

Suppose we want to describe a safety property:

The client must never send a packet after reading a classified file

Example: No send after read

Suppose we want to describe a safety property:

The client must never send a packet after reading a classified file

Let $P = \{\mathit{Send}, \mathit{Read}\}$

Example: No send after read

Suppose we want to describe a safety property:

The client must never send a packet after reading a classified file

Let $P = \{\mathit{Send}, \mathit{Read}\}$

Technically, our Σ should be: $\{\{\}, \{\mathit{Send}\}, \{\mathit{Read}\}, \{\mathit{Send}, \mathit{Read}\}\}$

Example: No send after read

Suppose we want to describe a safety property:

The client must never send a packet after reading a classified file

Let $P = \{\mathit{Send}, \mathit{Read}\}$

Technically, our Σ should be: $\{\{\}, \{\mathit{Send}\}, \{\mathit{Read}\}, \{\mathit{Send}, \mathit{Read}\}\}$

We'll be a bit sloppy, and let Σ be formulas over $\mathit{Send}, \mathit{Read}$

Example: No send after read

Then we can write an ω-regular expression:

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^{\omega} + (\mathit{Read})(\neg \mathit{Send})^{\omega}$$

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^\omega + (\mathit{Read})(\neg \mathit{Send})^\omega$$

And we can encode this as an NBA:

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \textit{Read})^{\omega} + (\textit{Read})(\neg \textit{Send})^{\omega}$$

And we can encode this as an NBA:

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^{\omega} + (\mathit{Read})(\neg \mathit{Send})^{\omega}$$

And we can encode this as an NBA:

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^\omega + (\mathit{Read})(\neg \mathit{Send})^\omega$$

And we can encode this as an NBA:

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^\omega + (\mathit{Read})(\neg \mathit{Send})^\omega$$

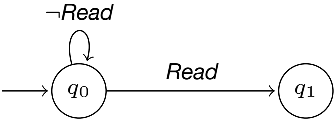
And we can encode this as an NBA:

Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^{\omega} + (\mathit{Read})(\neg \mathit{Send})^{\omega}$$

And we can encode this as an NBA:

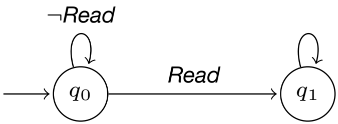


Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^\omega + (\mathit{Read})(\neg \mathit{Send})^\omega$$

And we can encode this as an NBA:

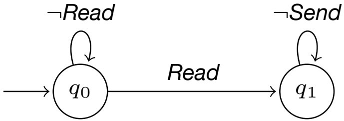


Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^{\omega} + (\mathit{Read})(\neg \mathit{Send})^{\omega}$$

And we can encode this as an NBA:

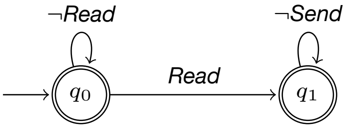


Example: No send after read

Then we can write an ω-regular expression:

$$(\neg \mathit{Read})^{\omega} + (\mathit{Read})(\neg \mathit{Send})^{\omega}$$

And we can encode this as an NBA:



Example: Partial correctness

Now a more complicated example:

Whenever the precondition is satisfied and the program terminates, the postcondition must be satisfied

Example: Partial correctness

Now a more complicated example:

Whenever the precondition is satisfied and the program terminates, the postcondition must be satisfied

Our alphabet: formulas over $\{\mathit{Pre}, \mathit{Post}, \mathit{Done}\}$

Example: Partial correctness

Now a more complicated example:

Whenever the precondition is satisfied and the program terminates, the postcondition must be satisfied

Our alphabet: formulas over $\{\mathit{Pre}, \mathit{Post}, \mathit{Done}\}$

What's our ω-regular expression?

Example: Partial correctness

Now a more complicated example:

Whenever the precondition is satisfied and the program terminates, the postcondition must be satisfied

Our alphabet: formulas over $\{\mathit{Pre}, \mathit{Post}, \mathit{Done}\}$

What's our ω-regular expression?

$$\neg \mathit{Pre}\,\mathit{true}^{\omega} + \mathit{Pre}\,\neg \mathit{Done}^{\omega} + \mathit{Pre}\,\neg \mathit{Done}^{*}(\mathit{Done} \wedge \mathit{Post})^{\omega}$$

Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre} \mathit{true}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{*} (\mathit{Done} \wedge \mathit{Post})^{\omega}$$

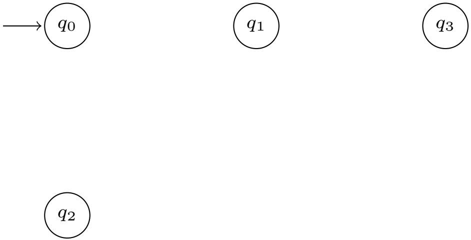
And a corresponding NBA:

Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre}\,\mathit{true}^{\omega} + \mathit{Pre}\,\neg \mathit{Done}^{\omega} + \mathit{Pre}\,\neg \mathit{Done}^{*}\,(\mathit{Done} \wedge \mathit{Post})^{\omega}$$

And a corresponding NBA:

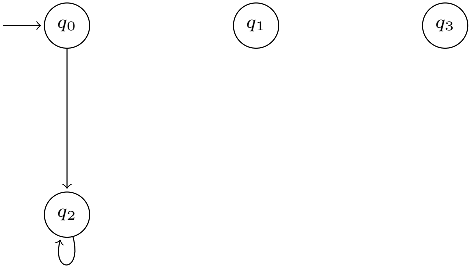


Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre} \mathit{true}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{*} (\mathit{Done} \wedge \mathit{Post})^{\omega}$$

And a corresponding NBA:

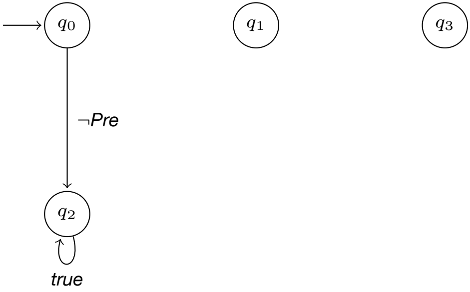


Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre}\,\mathit{true}^\omega + \mathit{Pre}\,\neg \mathit{Done}^\omega + \mathit{Pre}\,\neg \mathit{Done}^* (\mathit{Done} \wedge \mathit{Post})^\omega$$

And a corresponding NBA:

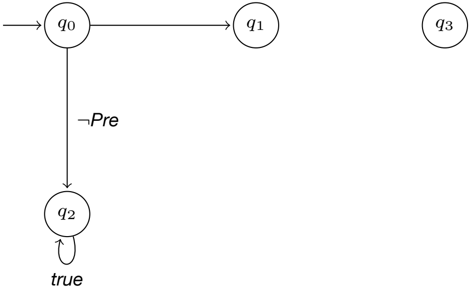


Example: Partial correctness

What's our ω-regular expression?

$$\neg Pre\, true^\omega + Pre\, \neg Done^\omega + Pre\, \neg Done^* (Done \wedge Post)^\omega$$

And a corresponding NBA:

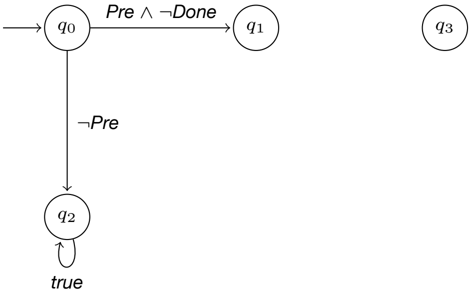


Example: Partial correctness

What's our ω-regular expression?

$$\neg Pre\, true^{\omega} + Pre \neg Done^{\omega} + Pre \neg Done^{*}(Done \wedge Post)^{\omega}$$

And a corresponding NBA:

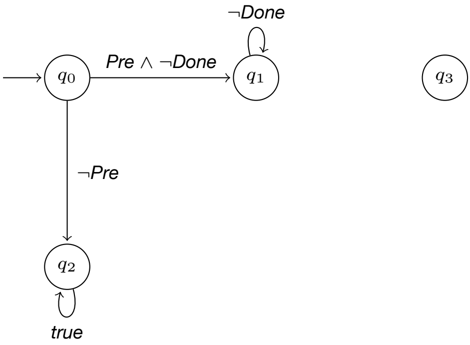


Example: Partial correctness

What's our ω-regular expression?

$$\neg \textit{Pre} \textit{true}^{\omega} + \textit{Pre}\neg\textit{Done}^{\omega} + \textit{Pre}\neg\textit{Done}^{*}(\textit{Done} \wedge \textit{Post})^{\omega}$$

And a corresponding NBA:

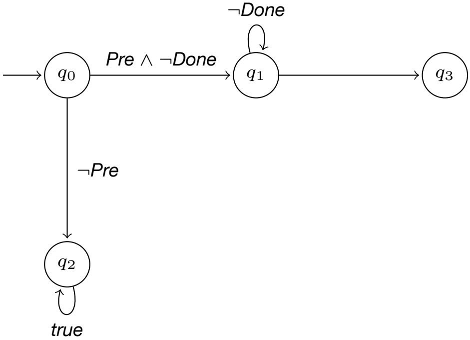


Example: Partial correctness

What's our ω-regular expression?

$$\neg Pre\, true^{\omega} + Pre\, \neg Done^{\omega} + Pre\, \neg Done^{*}\,(Done \wedge Post)^{\omega}$$

And a corresponding NBA:

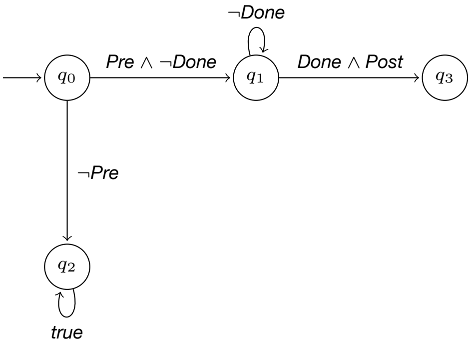


Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre} \mathit{true}^\omega + \mathit{Pre} \neg \mathit{Done}^\omega + \mathit{Pre} \neg \mathit{Done}^* (\mathit{Done} \wedge \mathit{Post})^\omega$$

And a corresponding NBA:

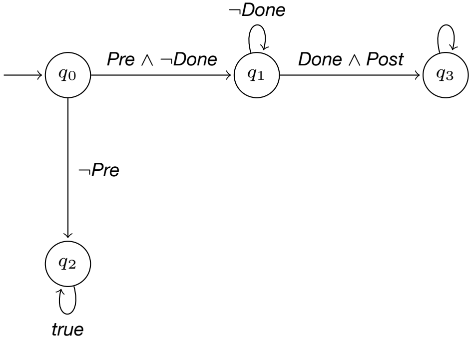


Example: Partial correctness

What's our ω-regular expression?

$$\neg Pre true^{\omega} + Pre \neg Done^{\omega} + Pre \neg Done^{*} (Done \wedge Post)^{\omega}$$

And a corresponding NBA:

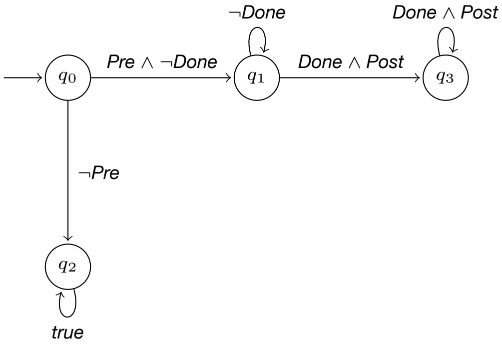


Example: Partial correctness

What's our ω-regular expression?

$$\neg \textit{Pre}\textit{true}^{\omega} + \textit{Pre}\neg\textit{Done}^{\omega} + \textit{Pre}\neg\textit{Done}^{*}(\textit{Done} \wedge \textit{Post})^{\omega}$$

And a corresponding NBA:

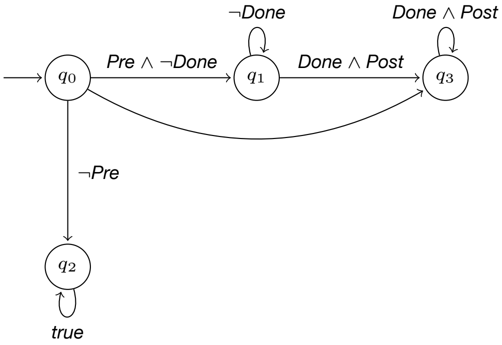


Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre} \mathit{true}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{*} (\mathit{Done} \wedge \mathit{Post})^{\omega}$$

And a corresponding NBA:

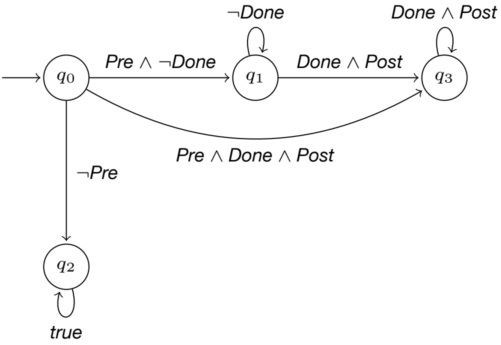


Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre}\,\mathit{true}^{\omega} + \mathit{Pre}\,\neg \mathit{Done}^{\omega} + \mathit{Pre}\,\neg \mathit{Done}^{*}(\mathit{Done} \wedge \mathit{Post})^{\omega}$$

And a corresponding NBA:

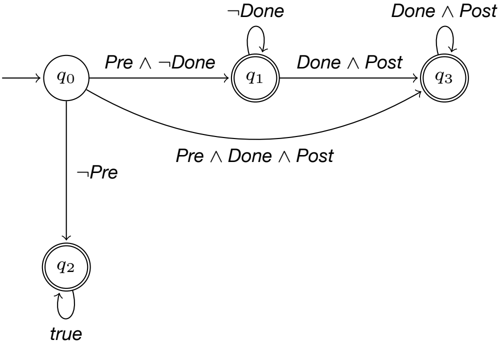


Example: Partial correctness

What's our ω-regular expression?

$$\neg \mathit{Pre} \mathit{true}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{\omega} + \mathit{Pre} \neg \mathit{Done}^{*} (\mathit{Done} \land \mathit{Post})^{\omega}$$

And a corresponding NBA:



ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement
- Each of these corresponds to operations on NBA

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement
- Each of these corresponds to operations on NBA

But these aren't necessarily the same operations as for NFAs

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement
- Each of these corresponds to operations on NBA

But these aren't necessarily the same operations as for NFAs

- E.g., for intersection, word needs to go through both sets of accepting states infinitely often

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement
- Each of these corresponds to operations on NBA

But these aren't necessarily the same operations as for NFAs

- E.g., for intersection, word needs to go through both sets of accepting states infinitely often
- Complement is tricky: NBAs aren't closed under determinization

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement
- Each of these corresponds to operations on NBA

But these aren't necessarily the same operations as for NFAs

- E.g., for intersection, word needs to go through both sets of accepting states infinitely often
- Complement is tricky: NBAs aren't closed under determinization

Emptiness is decidable in linear time

ω-Regular Closure Properties, Complexity

Like regular languages, ω-regular enjoy closure properties

- Union
- Intersection
- Complement
- Each of these corresponds to operations on NBA

But these aren't necessarily the same operations as for NFAs

- E.g., for intersection, word needs to go through both sets of accepting states infinitely often
- Complement is tricky: NBAs aren't closed under determinization

Emptiness is decidable in linear time

- This is important for model checking, as we'll see

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Let A_ϕ be an NBA representing the specification

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Let A_ϕ be an NBA representing the specification

- A_ϕ describes the **allowed traces**

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Let A_ϕ be an NBA representing the specification

- A_ϕ describes the **allowed traces**
- Its language corresponds to “good” computations

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Let A_ϕ be an NBA representing the specification

- A_ϕ describes the **allowed traces**
- Its language corresponds to "good" computations

Then A satisfies the specification A_ϕ exactly when:

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Let A_ϕ be an NBA representing the specification

- A_ϕ describes the **allowed traces**
- Its language corresponds to "good" computations

Then A satisfies the specification A_ϕ exactly when:

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

Automata-Theoretic Model Checking

Let A be an NBA representing some computation

Let A_ϕ be an NBA representing the specification

- A_ϕ describes the **allowed traces**
- Its language corresponds to "good" computations

Then A satisfies the specification A_ϕ exactly when:

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

The set of traces in A is contained in the set of "good" computations

Automata-Theoretic Model Checking

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

$$L(A) \subseteq L(S) \Leftrightarrow L(A) \cap \overline{L(A_\phi)} = \varnothing$$

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

Automata-Theoretic Model Checking

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

$$L(A) \subseteq L(S) \Leftrightarrow L(A) \cap \overline{L(A_\phi)} = \varnothing$$

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

Automata-Theoretic Model Checking

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

$$L(A) \subseteq L(S) \Leftrightarrow L(A) \cap \overline{L(A_\phi)} = \varnothing$$

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?

Automata-Theoretic Model Checking

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

$$L(A) \subseteq L(S) \Leftrightarrow L(A) \cap \overline{L(A_\phi)} = \varnothing$$

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?

- $L(A) \cap \overline{L(A_\phi)} \neq \varnothing$ gives an ω-regular language

Automata-Theoretic Model Checking

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

$$L(A) \subseteq L(S) \Leftrightarrow L(A) \cap \overline{L(A_\phi)} = \varnothing$$

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?

- $L(A) \cap \overline{L(A_\phi)} \neq \varnothing$ gives an ω-regular language
- Any word in this language is a prohibited trace

Automata-Theoretic Model Checking

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

$$L(A) \subseteq L(S) \Leftrightarrow L(A) \cap \overline{L(A_\phi)} = \varnothing$$

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?

- $L(A) \cap \overline{L(A_\phi)} \neq \varnothing$ gives an ω-regular language
- Any word in this language is a prohibited trace
- We pick an aribtrary word, find an appropriate prefix

Automata-Theoretic LTL Checking

We would like to solve the LTL model checking problem:

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

Automata-Theoretic LTL Checking

We would like to solve the LTL model checking problem:

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

To do this, we'll need to represent M and ϕ as NBAs

Automata-Theoretic LTL Checking

We would like to solve the LTL model checking problem:

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

To do this, we'll need to represent M and ϕ 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 ϕ, decide whether $M, \pi \models \phi$ for each π starting in an initial state.

To do this, we'll need to represent M and ϕ as NBAs

Intuitively, this should pose no problem

- M is a nondeterministic system over infinite paths

Automata-Theoretic LTL Checking

We would like to solve the LTL model checking problem:

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

To do this, we'll need to represent M and ϕ 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

Automata-Theoretic LTL Checking

We would like to solve the LTL model checking problem:

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

To do this, we'll need to represent M and ϕ 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

Modeling Systems as NBA

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 $R \subseteq S \times S$

Modeling Systems as NBA

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:

Modeling Systems as NBA

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

Modeling Systems as NBA

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

Modeling Systems as NBA

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 $R \subseteq S \times S$

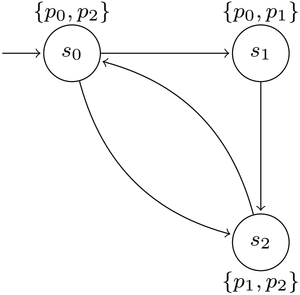
Recalling this definition, the main difference seems to be:

- Transitions have no labels
- The "natural" alphabet P labels states, not transitions
- There are no accepting states

Kripke Structure $\mapsto$ NBA

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

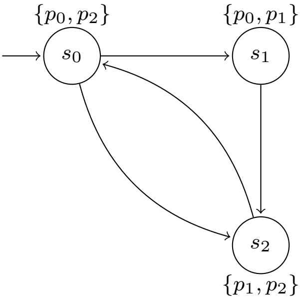
We want NBA $A = (\Sigma, Q, Q_0, F, \delta)$
where:



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

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

- $\Sigma = 2^P$

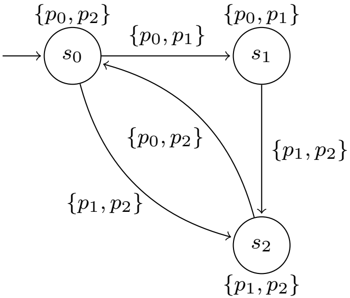


Kripke Structure $\mapsto$ NBA

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 $L(q') = \alpha$

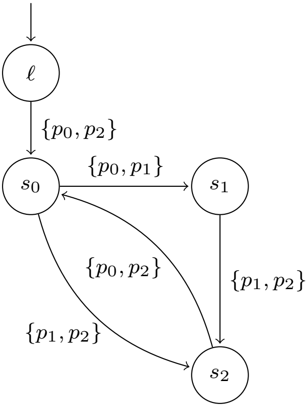


Kripke Structure $\mapsto$ NBA

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 $L(q') = \alpha$

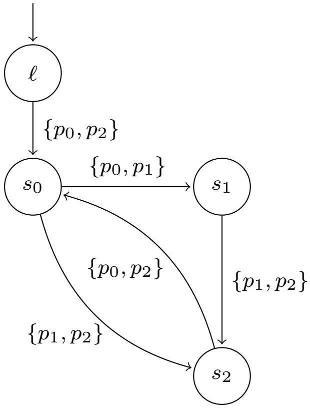


Kripke Structure $\mapsto$ NBA

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

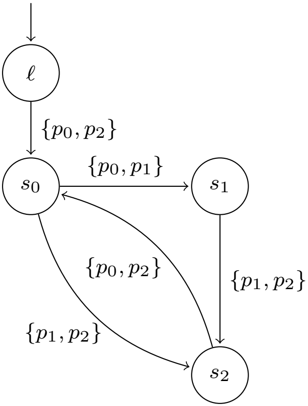


Kripke Structure $\mapsto$ NBA

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?

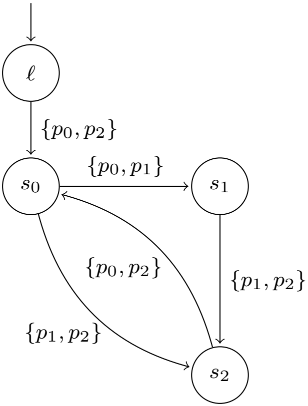


Kripke Structure $\mapsto$ NBA

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 $F = Q$

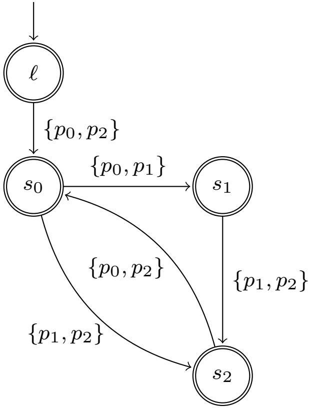


Kripke Structure $\mapsto$ NBA

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 $F = Q$



NBA for LTL Formulas

The final piece: converting LTL to NBA

NBA for LTL Formulas

The final piece: converting LTL to NBA

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

NBA for LTL Formulas

The final piece: converting LTL to NBA

The "leaves" of LTL formulas are propositional formulas over P

$$\mathbf{G}\ \mathbf{F}\ (p \vee q) \qquad \mathbf{G}\ (\neg c_1 \vee \neg c_2) \qquad \mathbf{G}\ (p \rightarrow \mathbf{F}\ q)$$

NBA for LTL Formulas

The final piece: converting LTL to NBA

The "leaves" of LTL formulas are propositional formulas over P

$$\mathbf{G}\ \mathbf{F}\ (p \vee q) \qquad \mathbf{G}\ (\neg c_1 \vee \neg c_2) \qquad \mathbf{G}\ (p \to \mathbf{F}\ q)$$

We'll use formulas over P to represent alphabet symbolically

NBA for LTL Formulas

The final piece: converting LTL to NBA

The "leaves" of LTL formulas are propositional formulas over P

$$\textbf{G}\ \textbf{F}\ (p \vee q) \qquad \textbf{G}\ (\neg c_1 \vee \neg c_2) \qquad \textbf{G}\ (p \rightarrow \textbf{F}\ q)$$

We'll use formulas over P to represent alphabet symbolically

For example, if we have a transition:

$$q_0 \xrightarrow{p_0 \vee p_1} q_1$$

NBA for LTL Formulas

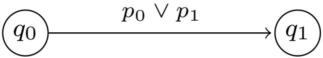
The final piece: converting LTL to NBA

The "leaves" of LTL formulas are propositional formulas over P

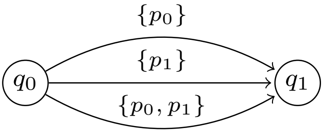
$$\mathbf{G}\ \mathbf{F}\ (p \vee q) \qquad \mathbf{G}\ (\neg c_1 \vee \neg c_2) \qquad \mathbf{G}\ (p \rightarrow \mathbf{F}\ q)$$

We'll use formulas over P to represent alphabet symbolically

For example, if we have a transition:



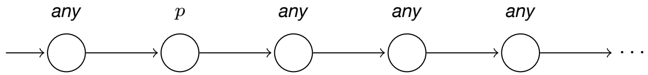
Then this is shorthand for:



LTL to NBA: Example (**X** operator)

Let's start with the next operator

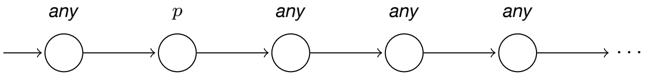
X p



LTL to NBA: Example (**X** operator)

Let's start with the next operator

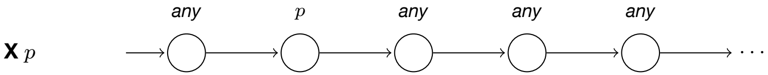
X p



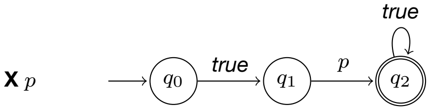
What is the corresponding NBA?

LTL to NBA: Example (**X** operator)

Let's start with the next operator



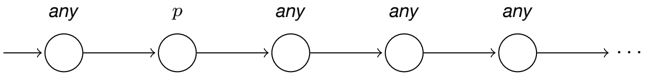
What is the corresponding NBA?



LTL to NBA: Example (**X** operator)

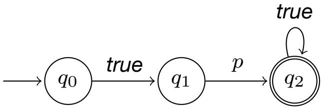
Let's start with the next operator

X p



What is the corresponding NBA?

X p

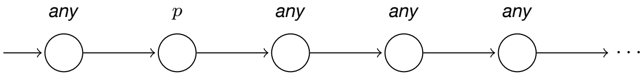


- It doesn't matter what the current state is

LTL to NBA: Example (**X** operator)

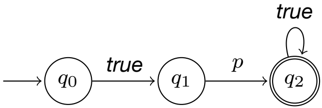
Let's start with the next operator

X p



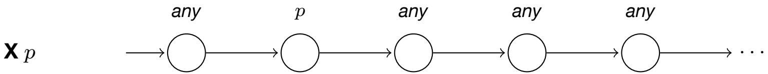
What is the corresponding NBA?

X p

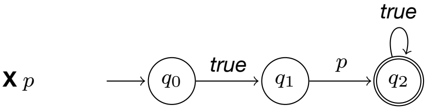


- It doesn't matter what the current state is
- The next state must satisfy p

Let's start with the next operator



What is the corresponding NBA?

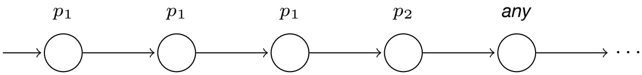


- It doesn't matter what the current state is
- The next state must satisfy p
- After that, any path suffices for acceptance

LTL to NBA: Example (**U** operator)

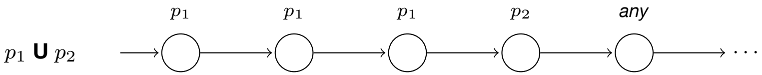
Now the until operator

p_1 **U** p_2



LTL to NBA: Example (**U** operator)

Now the until operator

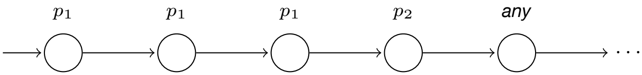


What is the corresponding NBA?

LTL to NBA: Example (**U** operator)

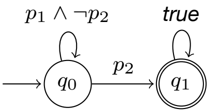
Now the until operator

p_1 **U** p_2



What is the corresponding NBA?

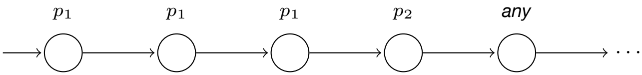
p_1 **U** p_2



LTL to NBA: Example (**U** operator)

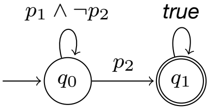
Now the until operator

p_1 **U** p_2



What is the corresponding NBA?

p_1 **U** p_2

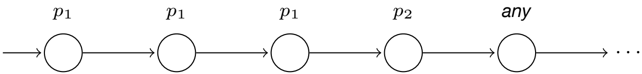


- p_1 holds arbitrarily long in the beginning

LTL to NBA: Example (**U** operator)

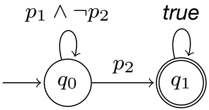
Now the until operator

p_1 **U** p_2



What is the corresponding NBA?

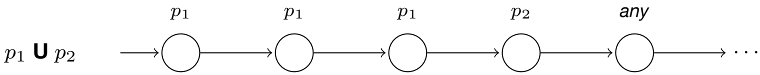
p_1 **U** p_2



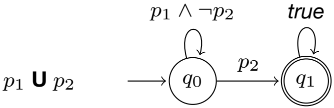
- p_1 holds arbitrarily long in the beginning
- To pass into accepting, p_2 must hold at some point

LTL to NBA: Example (**U** operator)

Now the until operator



What is the corresponding NBA?



- p_1 holds arbitrarily long in the beginning
- To pass into accepting, p_2 must hold at some point
- Afterwards, anything goes

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

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

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

- **F** $p \Leftrightarrow$ *true* **U** p
- **G** $p \Leftrightarrow \neg$**F** $\neg p$

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

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

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

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

However, composing temporal operators is expensive in general

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

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

However, composing temporal operators is expensive in general

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

LTL to NBA: Remaining Operators

X and **U** are sufficient to express **F** , **G** , **R**

- **F** $p \Leftrightarrow$ *true* **U** p
- **G** $p \Leftrightarrow \neg$**F** $\neg p$
- p_1 **R** $p_2 \Leftrightarrow \neg(\neg p_1$ **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

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ 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 ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate ϕ before building NBA

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate ϕ before building NBA
3. Check emptiness of $L(A \cap \overline{A_\phi})$

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

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

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate ϕ 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: $O(|M| \cdot 2^{|\phi|})$

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate ϕ 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 $|A_1| \cdot |A_2|$

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate ϕ 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_ϕ of size $2^{|\phi|}$

Summary: Automata-Based LTL Model Checking

Given a Kripke structure M and LTL ϕ:

1. Convert M into Buchi automaton A, ϕ into A_ϕ
2. Negate ϕ by building complement $\overline{A_\phi}$
   - **Note**: Complement can blow up exponentially!
   - In practice, negate ϕ 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_ϕ 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_ϕ 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_ϕ first
2. Begin taking intersection at initial states of A

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_ϕ first
2. Begin taking intersection at initial states of A
3. Perform DFS incrementally at each step

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_ϕ 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_ϕ 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_ϕ 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!

Next Lecture

- Symbolic model checking
- If time: introduce a model-checking tool