Regular Expressions

Arjun Chandrasekhar

Regular Expressions

Regular Expressions

- Another way to describe languages

Regular Expressions

- Another way to describe languages
- A formula that can be used to generate strings

Regular Expressions

- Another way to describe languages
- A formula that can be used to generate strings
- Formed by combining smaller regular expressions using the three regular operations

Regular Expressions

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

1. σ for some $\sigma \in \Sigma$

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

1. σ for some $\sigma \in \Sigma$
2. ϵ

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

1. σ for some $\sigma \in \Sigma$
2. ϵ
3. $\emptyset$

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

1. σ for some $\sigma \in \Sigma$
2. ϵ
3. $\emptyset$
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

1. σ for some $\sigma \in \Sigma$
2. ϵ
3. $\emptyset$
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or

Regular Expressions

Let Σ be a alphabet. We say R is a **regular expression** if R is:

1. σ for some $\sigma \in \Sigma$
2. ϵ
3. $\emptyset$
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or
6. R_1^* where R_1 is a regular expression

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = a$$

A) ϵ (empty string)

B) a

C) b

D) aaaaaaaaaa

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = a$$

A) ϵ (empty string)

B) a ✓

C) b

D) aaaaaaaaaa

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = \epsilon$$

A) ϵ (empty string)

B) a

C) b

D) aaaaaaaaaa

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = \epsilon$$

A) ϵ (empty string) ✓

B) a

C) b

D) aaaaaaaaaa

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = \emptyset$$

A) ϵ (empty string)

B) a

C) b

D) aaaaaaaaaa

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = \emptyset$$

A) ϵ (empty string)

B) a

C) b

D) aaaaaaaaaa

E) None of the above ✓

F) invalid regex

Empty set is different from empty string!

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = a \circ b^*$$

A) ϵ (empty string)

B) a

C) abababab

D) abbbbbbb

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = a \circ b^*$$

A) ϵ (empty string)

B) a ✓

C) abababab

D) abbbbbbb ✓

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b)^*$$

A) ϵ (empty string)

B) ab

C) abababab

D) abbbbbbb

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b)^*$$

A) ϵ (empty string) ✓

B) ab ✓

C) abababab ✓

D) abbbbbbb

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b)^* \circ b \circ b \circ a$$

A) ϵ (empty string)

B) bba

C) abab

D) ababbba

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b)^* \circ b \circ b \circ a$$

A) ϵ (empty string)

B) bba ✓

C) abab

D) ababbba ✓

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b^*) \cup (b \circ a^*)$$

A) ϵ (empty string)

B) abbb

C) aaab

D) b

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b^*) \cup (b \circ a^*)$$

A) ϵ (empty string)

B) abbb ✓

C) aaab

D) b ✓

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b^*) \cup (c \circ d^*)$$

A) ϵ (empty string)

B) abbb

C) aaab

D) b

E) None of the above

F) invalid regex

Regular expressions

Let $\Sigma = \{a, b\}$ Which strings are generated by the following regex?

$$R = (a \circ b^*) \cup (c \circ d^*)$$

A) ϵ (empty string)

B) abbb

C) aaab

D) b

E) None of the above

F) invalid regex ✓

Regular Expressions

Some notes:

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R
- The operator precedence is $* > \circ > \cup$

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R
- The operator precedence is $* > \circ > \cup$
  - Parentheses can be used to override this

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R
- The operator precedence is $* > \circ > \cup$
  - Parentheses can be used to override this
- Concatenation is often done implicitly

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R
- The operator precedence is $* > \circ > \cup$
  - Parentheses can be used to override this
- Concatenation is often done implicitly
  - Can write *bba* instead of $b \circ b \circ a$

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R
- The operator precedence is $* > \circ > \cup$
  - Parentheses can be used to override this
- Concatenation is often done implicitly
  - Can write *bba* instead of $b \circ b \circ a$
- Σ is often shorthand for $\sigma_1 \cup \sigma_2 \ldots \sigma_n$

Regular Expressions

Some notes:

- Don't confuse Kleene star and linux wildcard
- Don't confuse ϵ and $\emptyset$; they are different
- $L(R)$ is the *language of* R, i.e. the set of strings generated by R
- The operator precedence is $* > \circ > \cup$
  - Parentheses can be used to override this
- Concatenation is often done implicitly
  - Can write *bba* instead of $b \circ b \circ a$
- Σ is often shorthand for $\sigma_1 \cup \sigma_2 \ldots \sigma_n$
- R^+ is shorthand for RR^* (which is shorthand for $R \circ R^*$)

Applications of Regular Expressions

Applications of Regular Expressions

- Lexical-analyzer generators, such as lex and flex. A lexical-analyzer is the part of a compiler that breaks a program into tokens. Regular expressions specify the valid tokens of a programming language.

Applications of Regular Expressions

- Lexical-analyzer generators, such as lex and flex. A lexical-analyzer is the part of a compiler that breaks a program into tokens. Regular expressions specify the valid tokens of a programming language.
- String search tools that are built into operating system utilities (like awk and grep in Unix), text editors, and programming language libraries. Regular expressions describe the strings that are being searched for.

Applications of Regular Expressions

- Lexical-analyzer generators, such as lex and flex. A lexical-analyzer is the part of a compiler that breaks a program into tokens. Regular expressions specify the valid tokens of a programming language.
- String search tools that are built into operating system utilities (like awk and grep in Unix), text editors, and programming language libraries. Regular expressions describe the strings that are being searched for.
- The regular expressions in these tools typically have a richer set of operators, to facilitate more easily describing strings.

Example regexes

Example regexes

Let $\Sigma = \{0, 1\}$

Example regexes

Let $\Sigma = \{0, 1\}$

1. $\Sigma^* = (0 \cup 1)^*$ = all binary strings (you have already seen this one!)

Example regexes

Let $\Sigma = \{0, 1\}$

1. $\Sigma^* = (0 \cup 1)^*$ = all binary strings (you have already seen this one!)
2. $\Sigma^* 001 \Sigma^* = \{\text{w} \mid \text{w contains 001 as a substring}\}$

Example regexes

Let $\Sigma = \{0, 1\}$

1. $\Sigma^* = (0 \cup 1)^*$ = all binary strings (you have already seen this one!)
2. $\Sigma^* 001 \Sigma^* = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
3. $0 \cup 1 \cup (0\Sigma^*0) \cup (1\Sigma^*1) = \{w \mid w \text{ starts and ends with the same symbol}\}$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$\mathsf{L} = \{\mathsf{w} \mid \text{every odd position is 1}\}$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$\text{L} = \{\text{w} \mid \text{every odd position is } 1\}$$

$$R = (1\Sigma)^* \circ (\epsilon \cup 1)$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$\mathsf{L} = \{\mathsf{w} \mid \mathsf{w} \text{ contains an even number of 0s}\}$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$L = \{w \mid w \text{ contains an even number of 0s}\}$$

$$(1^*01^*0)^*1^*$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$L = \{w \mid w \text{ contains an even number of 0s}\}$$

$$(1^*01^*0)^*1^*$$

$$1^*(01^*01^*)^*$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$L = \{w \mid w \text{ contains exactly two 1s}\}$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$L = \{w \mid w \text{ contains exactly two 1s}\}$$

$$(0^*1)(0^*1)0^*$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$\mathsf{L} = \{\mathsf{w} \mid \mathsf{w}\text{ contains an even number of 0s or exactly two 1s}\}$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

L $= \{$w $|$ w contains an even number of 0s or exactly two 1s$\}$

$$((1^*01^*0)^*1^*) \cup (0^*10^*10^*)$$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

L $= \{$w $|$ w contains an even number of 0s and exactly two 1s$\}$

Regex practice

Let $\Sigma = \{0, 1\}$. Write a regex for:

$$L = \{w \mid w \text{ contains an even number of 0s and exactly two 1s}\}$$

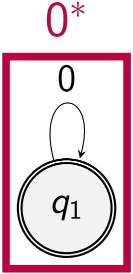
$$(00)^*1(00)^*1(00)^*$$
$$\cup$$
$$0(00)^*10(00)^*1(00)^*$$
$$\cup$$
$$(00)^*10(00)^*10(00)^*$$
$$\cup$$
$$0(00)^*1(00)^*10(00)^*$$

Regex to NFA

Design an NFA with 5 states to recognize $0^* \cup 1^*0^+$

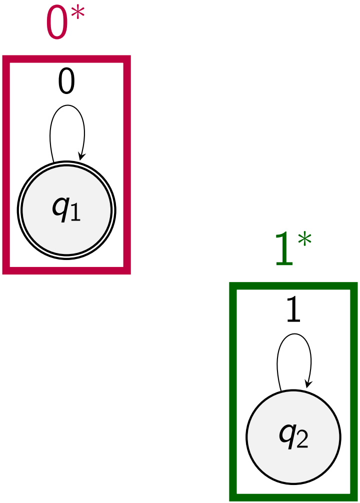
Regex to NFA

Design an NFA with 5 states to recognize $0^* \cup 1^*0^+$



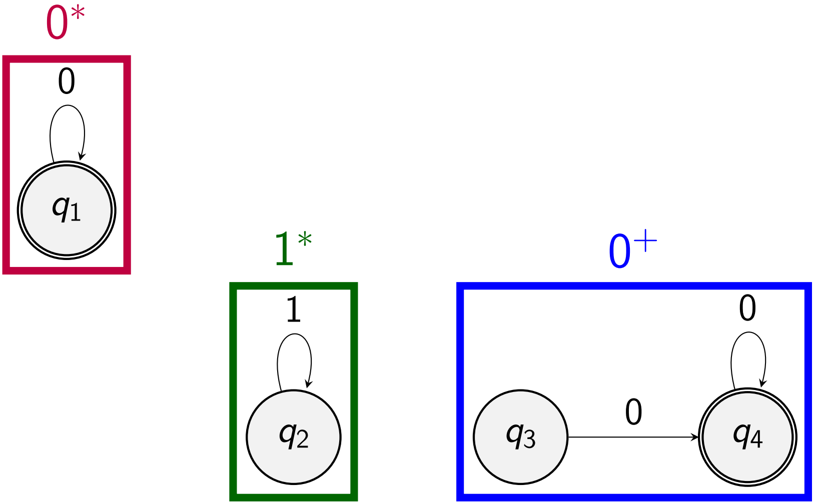
Regex to NFA

Design an NFA with 5 states to recognize $0^* \cup 1^*0^+$



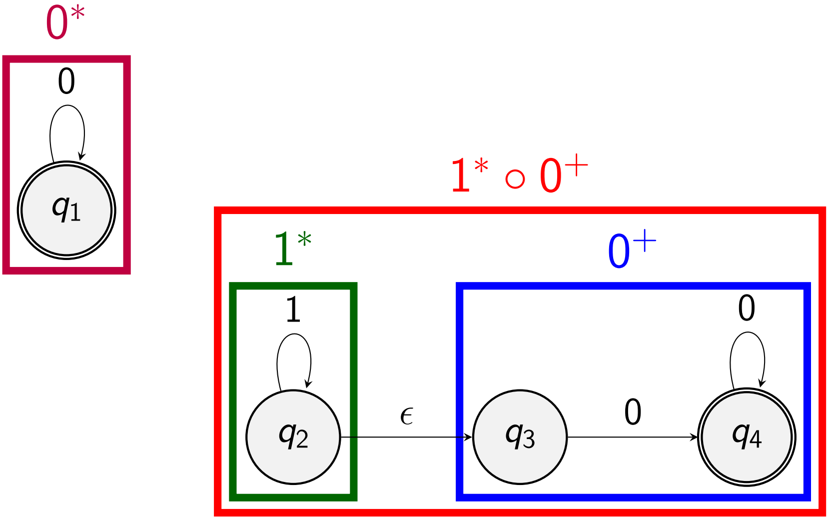
Regex to NFA

Design an NFA with 5 states to recognize $0^* \cup 1^*0^+$



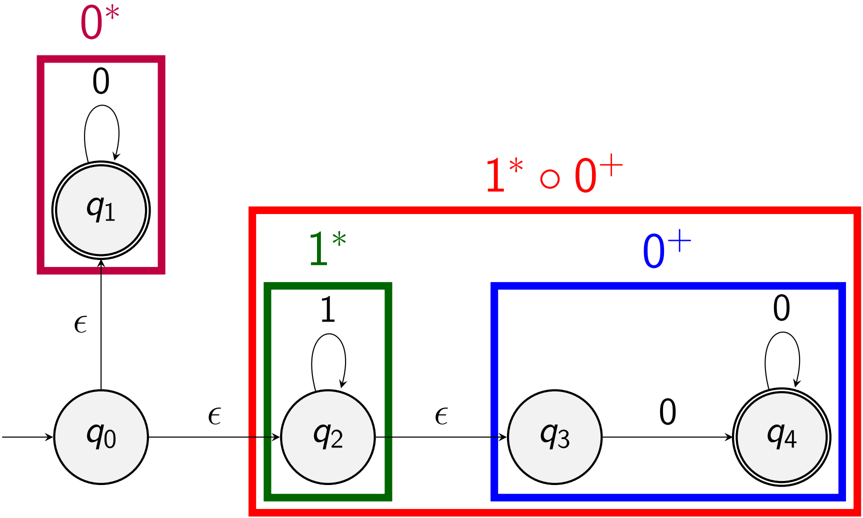
Regex to NFA

Design an NFA with 5 states to recognize $0^* \cup 1^*0^+$



Regex to NFA

Design an NFA with 5 states to recognize $0^* \cup 1^*0^+$



Kleene's Theorem

Kleene's Theorem

Theorem: A language is described by a regular expression if and only if it is regular

Kleene's Theorem

Theorem: A language is described by a regular expression if and only if it is regular

- What are the two directions we must prove?

Kleene's Theorem

Theorem: A language is described by a regular expression if and only if it is regular

- What are the two directions we must prove?
  - ($\Rightarrow$) If a language is described by a regular expression, it is regular

Kleene's Theorem

Theorem: A language is described by a regular expression if and only if it is regular

- What are the two directions we must prove?
  - ($\Rightarrow$) If a language is described by a regular expression, it is regular
  - ($\Leftarrow$) If a language is regular, then it is described by a regular expression

Kleene's Theorem

Theorem: A language is described by a regular expression if and only if it is regular

- What are the two directions we must prove?
  - ($\Rightarrow$) If a language is described by a regular expression, it is regular
  - ($\Leftarrow$) If a language is regular, then it is described by a regular expression
- **Recall:** A language is regular if and only if it is described by a DFA

Kleene's Theorem

Theorem: A language is described by a regular expression if and only if it is regular

- What are the two directions we must prove?
  - ($\Rightarrow$) If a language is described by a regular expression, it is regular
  - ($\Leftarrow$) If a language is regular, then it is described by a regular expression
- **Recall:** A language is regular if and only if it is described by a DFA
  - Or equivalently (and conveniently), an NFA

Kleene's Theorem (Forward Direction)

Kleene's Theorem (Forward Direction)

Claim: If a language L can be described by a regular expression R, then L is regular

Kleene's Theorem (Forward Direction)

Claim: If a language L can be described by a regular expression R, then L is regular

- **Proof Idea:** We will use induction to create an NFA for R

Kleene's Theorem (Forward Direction)

Claim: If a language L can be described by a regular expression R, then L is regular

- **Proof Idea:** We will use induction to create an NFA for R
- Show how to make an NFA for the atomic regular expressions

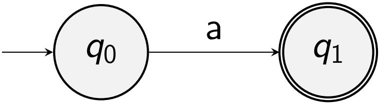
Kleene's Theorem (Forward Direction)

Claim: If a language L can be described by a regular expression R, then L is regular

- **Proof Idea:** We will use induction to create an NFA for R
- Show how to make an NFA for the atomic regular expressions
- For union, concatenation, and star, use induction to make NFAs for the smaller parts of the expression, and then combine them

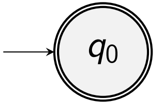
Kleene's theorem (Forward Direction)

Base Case: $R = \sigma \in \Sigma$



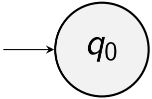
Kleene's theorem (Forward Direction)

Base Case: $R = \epsilon$



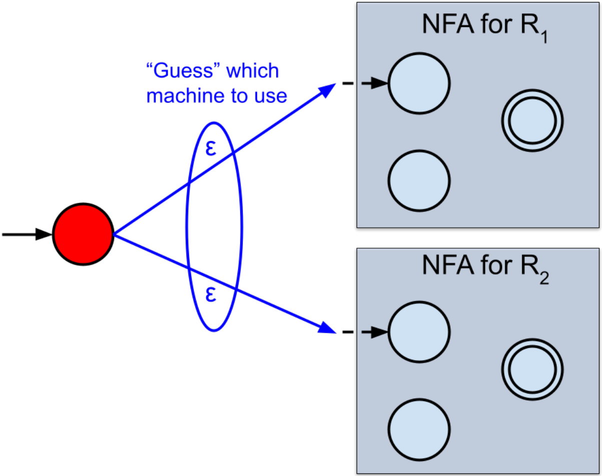
Kleene's theorem (Forward Direction)

Base Case: $R = \emptyset$



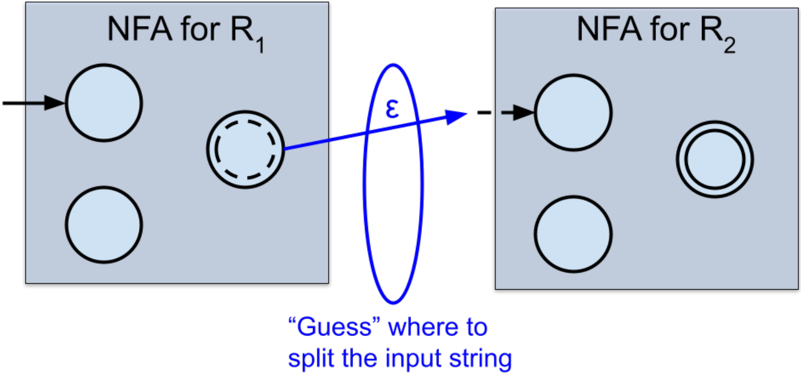
Kleene's theorem (Forward Direction)

Inductive Case: $R = R_1 \cup R_2$



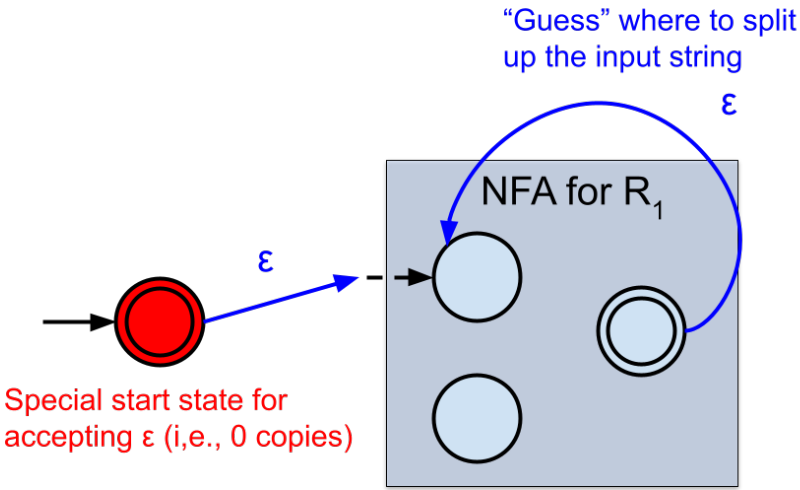
Kleene's theorem (Forward Direction)

Inductive Case: $R = R_1 \circ R_2$



Kleene's theorem (Forward Direction)

Inductive Case: $R = (R_1)^*$



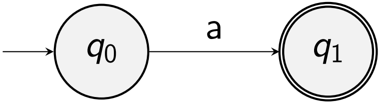
Regex to NFA Conversion Example

Let's make an NFA for $R = ((ab) \cup a)^*$

Regex to NFA Conversion Example

Let's make an NFA for $R = ((ab) \cup a)^*$

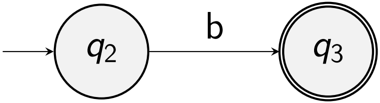
NFA for a



Regex to NFA Conversion Example

Let's make an NFA for $R = ((ab) \cup a)^*$

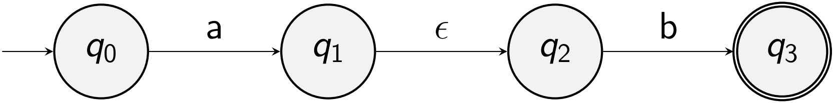
NFA for b



Regex to NFA Conversion Example

Let's make an NFA for $R = ((ab) \cup a)^*$

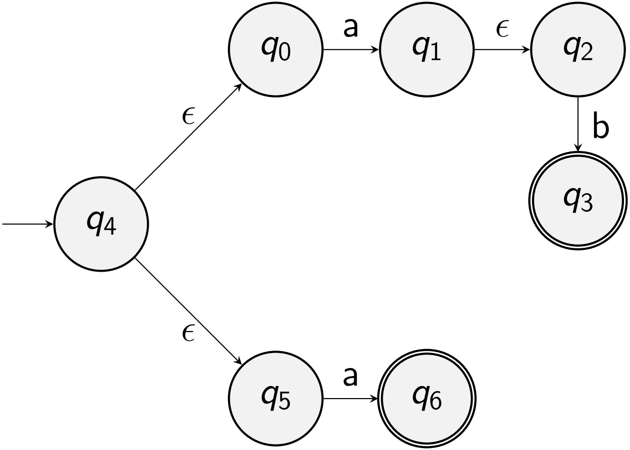
NFA for $ab = a \circ b$



Regex to NFA Conversion Example

Let's make an NFA for $R = ((ab) \cup a)^*$

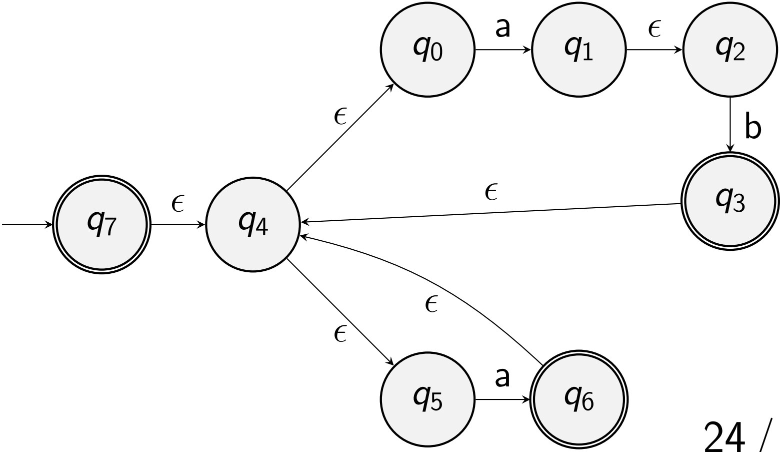
NFA for $ab \cup a$



Regex to NFA Conversion Example

Let's make an NFA for $R = ((ab) \cup a)^*$

NFA for $(ab \cup a)^*$



Kleene's Theorem (backwards direction)

Kleene's Theorem (backwards direction)

Claim: If L is regular, then L can be described by a regular expression

Kleene's Theorem (backwards direction)

Claim: If L is regular, then L can be described by a regular expression

- **Proof Idea**: Convert the DFA for L into a regex

Kleene's Theorem (backwards direction)

Claim: If L is regular, then L can be described by a regular expression

- **Proof Idea**: Convert the DFA for L into a regex
- Extend the DFA so that each transition is a regex

Kleene's Theorem (backwards direction)

Claim: If L is regular, then L can be described by a regular expression

- **Proof Idea**: Convert the DFA for L into a regex
- Extend the DFA so that each transition is a regex
- "Rip" states one at a time, and modify the other transitions to compensate

Kleene's Theorem (backwards direction)

Claim: If L is regular, then L can be described by a regular expression

- **Proof Idea**: Convert the DFA for L into a regex
- Extend the DFA so that each transition is a regex
- "Rip" states one at a time, and modify the other transitions to compensate
- When there's just one transition remaining, we will have the desired regex

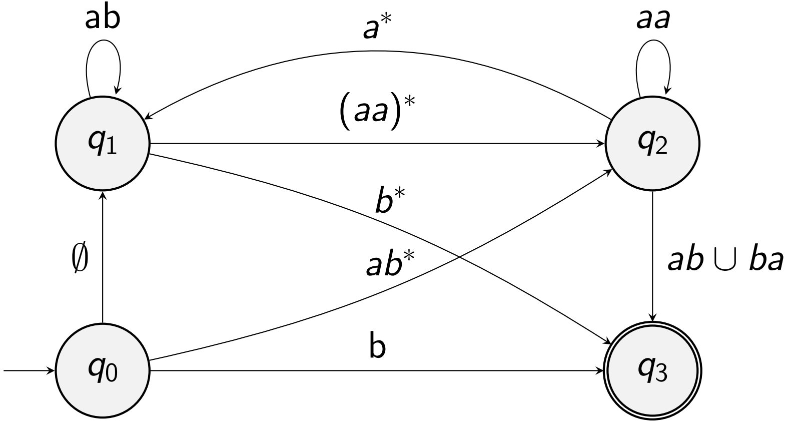
GNFAs

GNFAs

A **Generalized Nondeterministic Finite Automata (GNFA)** is an NFA in which arrows are labelled by regular expressions (rather than symbols)

GNFAs

A **Generalized Nondeterministic Finite Automata (GNFA)** is an NFA in which arrows are labelled by regular expressions (rather than symbols)



GNFAs

For convenience we require GNFAs be in the following special form:

GNFAs

For convenience we require GNFAs be in the following special form:

- The start state q_s has transition arrows going to every other state, but no arrows coming in from any other state

GNFAs

For convenience we require GNFAs be in the following special form:

- The start state q_s has transition arrows going to every other state, but no arrows coming in from any other state
- There is only a single accept state, q_F, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.

GNFAs

For convenience we require GNFAs be in the following special form:

- The start state q_s has transition arrows going to every other state, but no arrows coming in from any other state
- There is only a single accept state, q_F, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state, and also from each state to itself.

DFA to GNFA

To make a DFA onto a GNFA:

DFA to GNFA

To make a DFA onto a GNFA:

1. Create a special start state, with an ϵ transition to the original start state

DFA to GNFA

To make a DFA onto a GNFA:

1. Create a special start state, with an ϵ transition to the original start state
2. Add a special accept state, with ϵ transitions from the original accept states

DFA to GNFA

To make a DFA onto a GNFA:

1. Create a special start state, with an ϵ transition to the original start state
2. Add a special accept state, with ϵ transitions from the original accept states
3. If any transition has multiple symbols, combine them into a union regex

DFA to GNFA

To make a DFA onto a GNFA:

1. Create a special start state, with an ϵ transition to the original start state
2. Add a special accept state, with ϵ transitions from the original accept states
3. If any transition has multiple symbols, combine them into a union regex
4. If any transition between states is missing, ad an $\emptyset$ transition

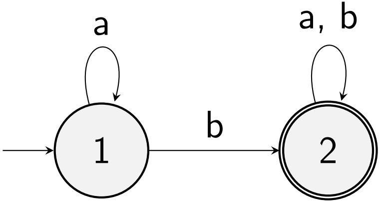
DFA to GNFA

To make a DFA onto a GNFA:

1. Create a special start state, with an ϵ transition to the original start state
2. Add a special accept state, with ϵ transitions from the original accept states
3. If any transition has multiple symbols, combine them into a union regex
4. If any transition between states is missing, ad an $\emptyset$ transition
   - We can omit these when drawing state diagrams

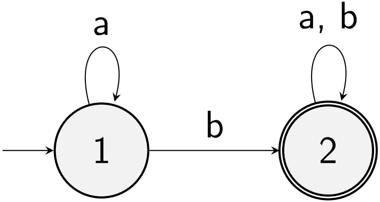
DFA to GNFA

Starting DFA

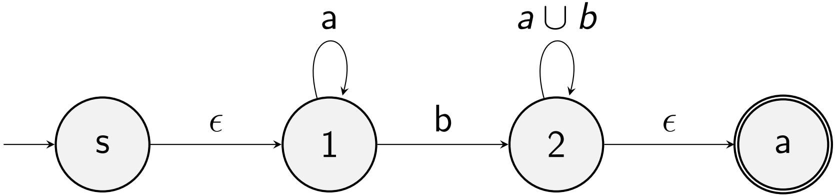


DFA to GNFA

Starting DFA

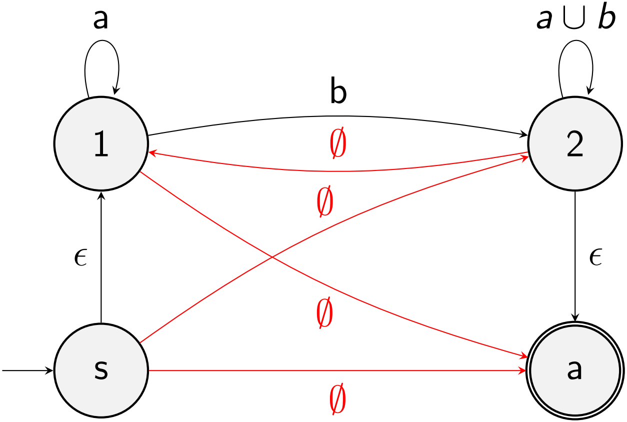


Starting GNFA



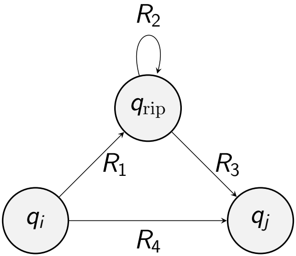
DFA to GNFA

Full starting GNFA



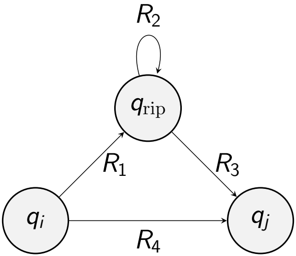
Ripping a state

How could we get from q_i to q_j?



Ripping a state

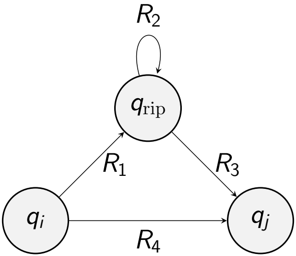
How could we get from q_i to q_j?



1. Go directly from q_i to q_j

Ripping a state

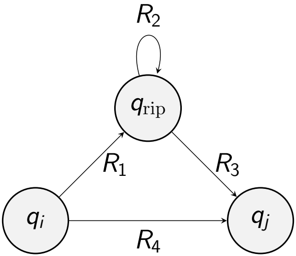
How could we get from q_i to q_j?



1. Go directly from q_i to q_j
2. Go through $q_{\rm rip}$

Ripping a state

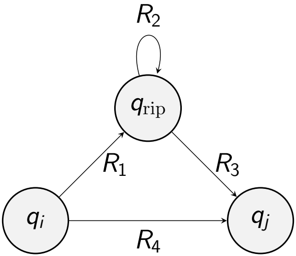
How could we get from q_i to q_j?



1. Go directly from q_i to q_j
2. Go through q_{rip}
   2.1 $q_i \to q_{\mathrm{rip}}$

Ripping a state

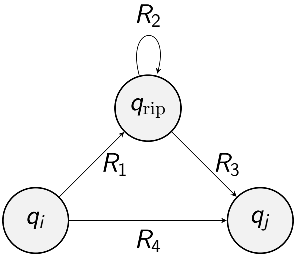
How could we get from q_i to q_j?



1. Go directly from q_i to q_j
2. Go through q_{rip}
   2.1 $q_i \to q_{\text{rip}}$
   2.2 $q_{\text{rip}} \to q_{\text{rip}}$ any number of times

Ripping a state

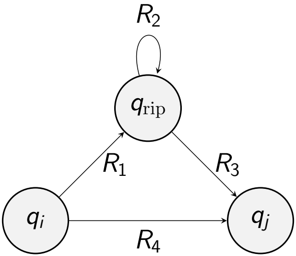
How could we get from q_i to q_j?



1. Go directly from q_i to q_j
2. Go through q_{rip}
   2.1 $q_i \to q_{\text{rip}}$
   2.2 $q_{\text{rip}} \to q_{\text{rip}}$ any number of times
   2.3 $q_{\text{rip}} \to q_j$

Ripping a state

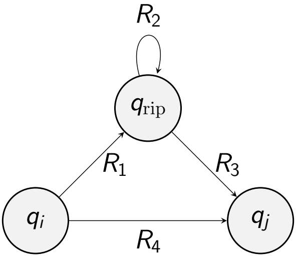
How could we get from q_i to q_j?



1. Go directly from q_i to q_j (R_4)
2. Go through q_{rip}
   2.1 $q_i \to q_{\text{rip}}$
   2.2 $q_{\text{rip}} \to q_{\text{rip}}$ any number of times
   2.3 $q_{\text{rip}} \to q_j$

Ripping a state

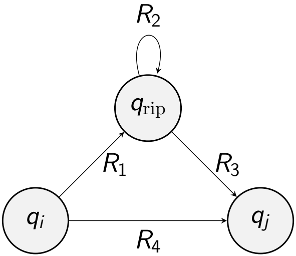
How could we get from q_i to q_j?



1. Go directly from q_i to q_j (R_4)
2. Go through q_{rip}
   2.1 $q_i \to q_{\text{rip}}$ (R_1)
   2.2 $q_{\text{rip}} \to q_{\text{rip}}$ any number of times
   2.3 $q_{\text{rip}} \to q_j$

Ripping a state

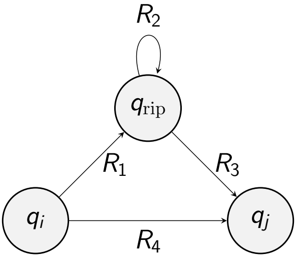
How could we get from q_i to q_j?



1. Go directly from q_i to q_j (R_4)
2. Go through q_{rip}
   2.1 $q_i \to q_{\text{rip}}$ (R_1)
   2.2 $q_{\text{rip}} \to q_{\text{rip}}$ any number of times (R_2^*)
   2.3 $q_{\text{rip}} \to q_j$

Ripping a state

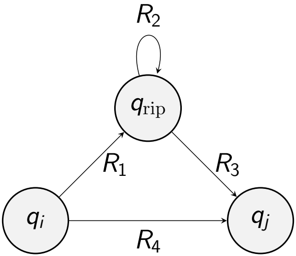
How could we get from q_i to q_j?



1. Go directly from q_i to q_j (R_4)
2. Go through q_{rip}
   2.1 $q_i \to q_{\mathrm{rip}}$ (R_1)
   2.2 $q_{\mathrm{rip}} \to q_{\mathrm{rip}}$ any number of times (R_2^*)
   2.3 $q_{\mathrm{rip}} \to q_j$ (R_3)

Ripping a state

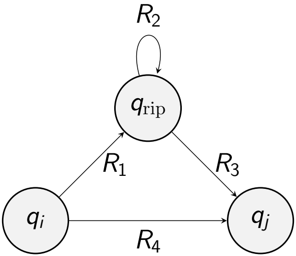
How could we get from q_i to q_j?



1. Go directly from q_i to q_j (R_4)
2. Go through q_{rip} ($R_1 \circ R_2^* \circ R_3$)
   - 2.1 $q_i \rightarrow q_{\mathrm{rip}}$ (R_1)
   - 2.2 $q_{\mathrm{rip}} \rightarrow q_{\mathrm{rip}}$ any number of times (R_2^*)
   - 2.3 $q_{\mathrm{rip}} \rightarrow q_j$ (R_3)

Ripping a state

How could we get from q_i to q_j?



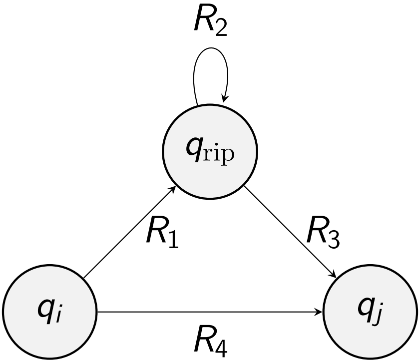
1. Go directly from q_i to q_j (R_4)
2. Go through q_{rip} ($R_1 \circ R_2^* \circ R_3$)
   2.1 $q_i \to q_{\text{rip}}$ (R_1)
   2.2 $q_{\text{rip}} \to q_{\text{rip}}$ any number of times (R_2^*)
   2.3 $q_{\text{rip}} \to q_j$ (R_3)

$R = (R_1 \circ R_2^* \circ R_3) \cup R_4$

Ripping a State

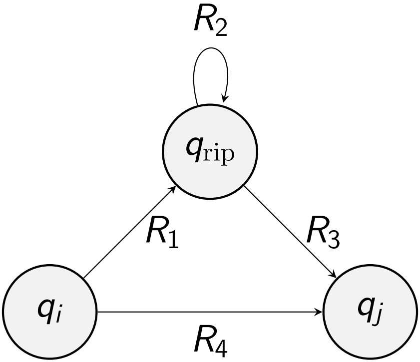
Ripping a State

Before

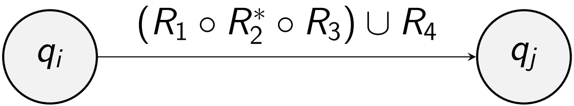


Ripping a State

Before

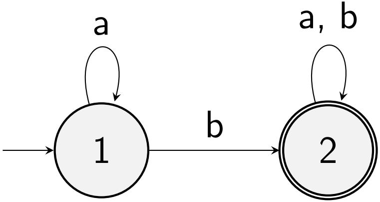


After



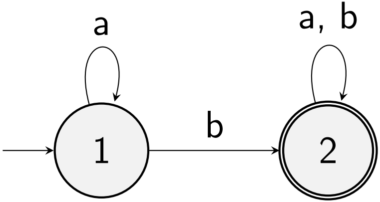
DFA to Regex

Starting DFA

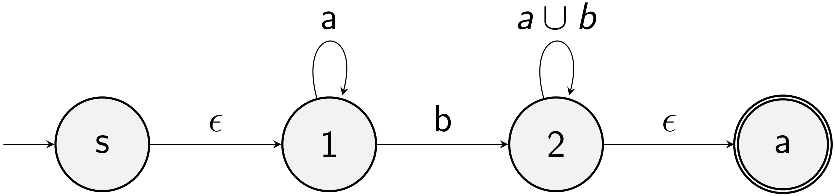


DFA to Regex

Starting DFA

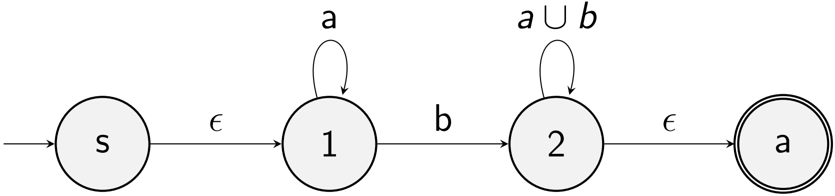


Starting GNFA

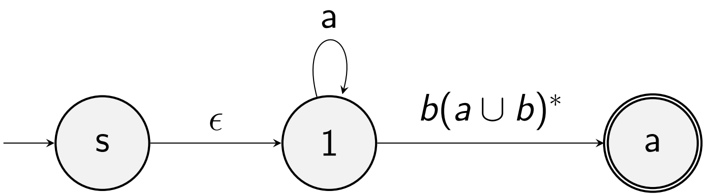


DFA to Regex

Starting GNFA

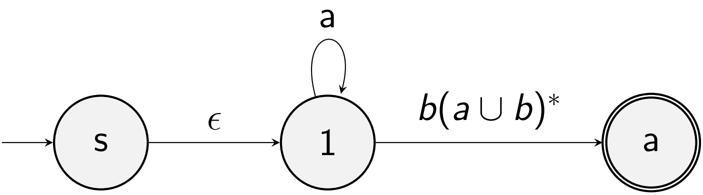


Rip State 2

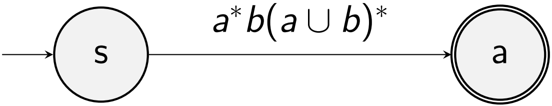


DFA to Regex

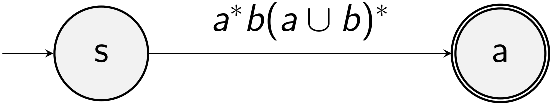
Rip State 2



Rip State 1



DFA to Regex



$$R = a^* b(a \cup B)^*$$

Regular expressions recap

Regular expressions recap

- Regular expressions are equivalent to NFAs

Regular expressions recap

- Regular expressions are equivalent to NFAs
  - Which makes them equivalent to DFAs

Regular expressions recap

- Regular expressions are equivalent to NFAs
  - Which makes them equivalent to DFAs
- DFAs are equivalent to regular expressions

Regu

Regular expressions recap

- Regular expressions are equivalent to NFAs
  - Which makes them equivalent to DFAs
- DFAs are equivalent to regular expressions
- A language is regular if and only if it is described by a regular expression

Regular expressions recap

- Regular expressions are equivalent to NFAs
  - Which makes them equivalent to DFAs
- DFAs are equivalent to regular expressions
- A language is regular if and only if it is described by a regular expression
- To show a language is regular, can use a state machine or a regex

Regular expression closure proofs

Regular expression closure proofs

- Regular expressions characterize the regular languages

Regular expression closure proofs

- Regular expressions characterize the regular languages
- To show a language is regular, it is sometimes more convenient to use a regex than a state machine

Regular expression closure proofs

- Regular expressions characterize the regular languages
- To show a language is regular, it is sometimes more convenient to use a regex than a state machine
- Sometimes, it is easier to write a closure proof using a regex

Regular expression closure proofs

- Regular expressions characterize the regular languages
- To show a language is regular, it is sometimes more convenient to use a regex than a state machine
- Sometimes, it is easier to write a closure proof using a regex
- **Blueprint:** Use an inductive proof

Regular expression closure proofs

- Regular expressions characterize the regular languages
- To show a language is regular, it is sometimes more convenient to use a regex than a state machine
- Sometimes, it is easier to write a closure proof using a regex
- **Blueprint:** Use an inductive proof
  - **Base case:** Show that closure holds for the three atomic regexes ($\emptyset, \epsilon, a$)

Regular expression closure proofs

- Regular expressions characterize the regular languages
- To show a language is regular, it is sometimes more convenient to use a regex than a state machine
- Sometimes, it is easier to write a closure proof using a regex
- **Blueprint:** Use an inductive proof
  - **Base case:** Show that closure holds for the three atomic regexes ($\emptyset, \epsilon, a$)
  - **Inductive case:** show that closure holds for t he three regular operations (union, concatenation, Kleene star)

EVERY-OTHER closure

Claim: If A is regular then $\text{EVERY-OTHER}(A)$ is regular

$$\text{EVERY-OTHER}(A) = \{w = \begin{array}{l} a_1 y_1 \dots a_n y_n | \\ a_1 \dots a_n \in A \\ y_i' s \text{ can be anything}\} \end{array}$$

EVERY-OTHER closure

Claim: If A is regular then $\text{EVERY-OTHER}(A)$ is regular

$$\text{EVERY-OTHER}(A) = \{w = a_1 y_1 \ldots a_n y_n | \; a_1 \ldots a_n \in A \; y_i's \text{ can be anything}\}$$

- Because A is regular, it is described by a regular expression R

EVERY-OTHER closure

Claim: If A is regular then EVERY-OTHER(A) is regular

$$\text{EVERY-OTHER}(A) = \{w = \begin{array}{l} a_1 y_1 \ldots a_n y_n | \\ a_1 \ldots a_n \in A \\ y_i's \text{ can be anything}\} \end{array}$$

- Because A is regular, it is described by a regular expression R
- We will construct a regular expression R' that describes EVERY-OTHER(A)

EVERY-OTHER closure: base case

EVERY-OTHER closure: base case

- $R = \emptyset$

EVERY-OTHER closure: base case

- $R = \emptyset$
  $R' = \emptyset$

EVERY-OTHER closure: base case

- $R = \emptyset$
  $R' = \emptyset$
- $R = \epsilon$

EVERY-OTHER closure: base case

- $R = \emptyset$
  $R' = \emptyset$
- $R = \epsilon$
  $R' = \epsilon$

EVERY-OTHER closure: base case

- $R = \emptyset$
  $R' = \emptyset$
- $R = \epsilon$
  $R' = \epsilon$
- $R = a \in \Sigma$

EVERY-OTHER closure: base case

- $R = \emptyset$
  $R' = \emptyset$
- $R = \epsilon$
  $R' = \epsilon$
- $R = a \in \Sigma$
  $R' = a\Sigma$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for $\text{EVERY-OTHER}(A)$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for $\text{EVERY-OTHER}(A)$

Let A be described by a regex R with size $n + 1$.

- $R = R_1 \cup R_2$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for EVERY-OTHER(A)

Let A be described by a regex R with size $n + 1$.

- $R = R_1 \cup R_2$
  $R' = R_1' \cup R_2'$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for $\text{EVERY-OTHER}(A)$

Let A be described by a regex R with size $n+1$.

- $R = R_1 \cup R_2$
  $R' = R_1' \cup R_2'$
- $R = R_1 \circ R_2$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for EVERY-OTHER(A)

Let A be described by a regex R with size $n + 1$.

- $R = R_1 \cup R_2$
  $R' = R_1' \cup R_2'$
- $R = R_1 \circ R_2$
  $R' = R_1' \circ R_2'$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for $\text{EVERY-OTHER}(A)$

Let A be described by a regex R with size $n + 1$.

- $R = R_1 \cup R_2$
  $R' = R_1' \cup R_2'$
- $R = R_1 \circ R_2$
  $R' = R_1' \circ R_2'$
- $R = (R_1)^*$

EVERY-OTHER closure: inductive case

Assume if A is described by a regex R_i with size $\leq n$, there is a regex R_i' for $\text{EVERY-OTHER}(A)$

Let A be described by a regex R with size $n+1$.

- $R = R_1 \cup R_2$
  $R' = R_1' \cup R_2'$
- $R = R_1 \circ R_2$
  $R' = R_1' \circ R_2'$
- $R = (R_1)^*$
  $R' = (R_1')^*$