Automata Theory and Formal Languages

Summer Term 2025

6th Lecture

Nondeterministic Finite Automata

Prof. Dr. Matthias Mnich *Institute for Algorithms and Complexity*

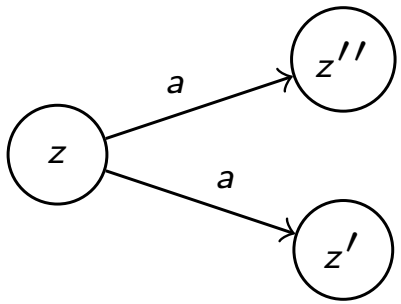
Non-deterministic Finite Automata

Non-deterministic finite automata

A Non-deterministic finite automaton (NFA) is an extension of DFA which allows its states to transit to multiples states on a single input character.

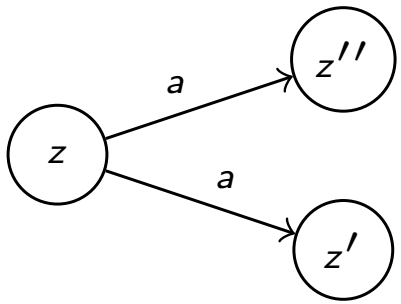
Non-deterministic finite automata

A Non-deterministic finite automaton (NFA) is an extension of DFA which allows its states to transit to multiples states on a single input character.



Non-deterministic finite automata

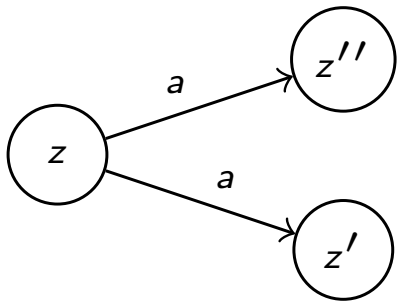
A Non-deterministic finite automaton (NFA) is an extension of DFA which allows its states to transit to multiples states on a single input character.



- An NFA accepts if at least one state sequence exist, which terminates in an accepting state.

Non-deterministic finite automata

A Non-deterministic finite automaton (NFA) is an extension of DFA which allows its states to transit to multiples states on a single input character.



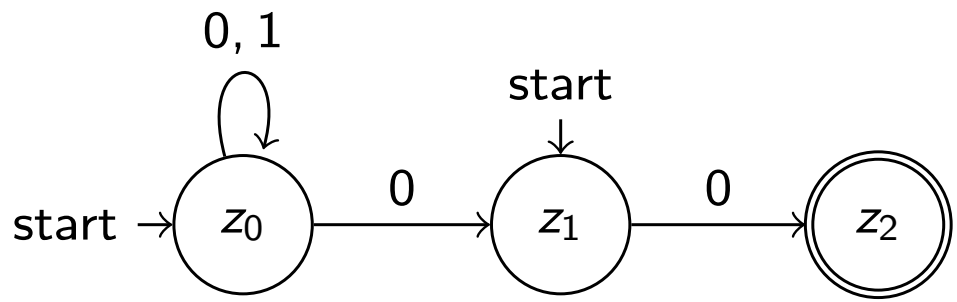
- An NFA accepts if at least one state sequence exist, which terminates in an accepting state.
- An NFA can have multiple initial states.

Example of non-deterministic finite automaton

An NFA over the alphabet $\{0, 1\}$

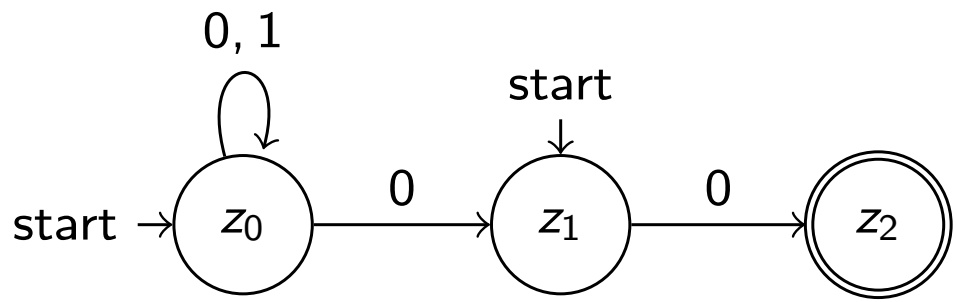
Example of non-deterministic finite automaton

An NFA over the alphabet $\{0, 1\}$



Example of non-deterministic finite automaton

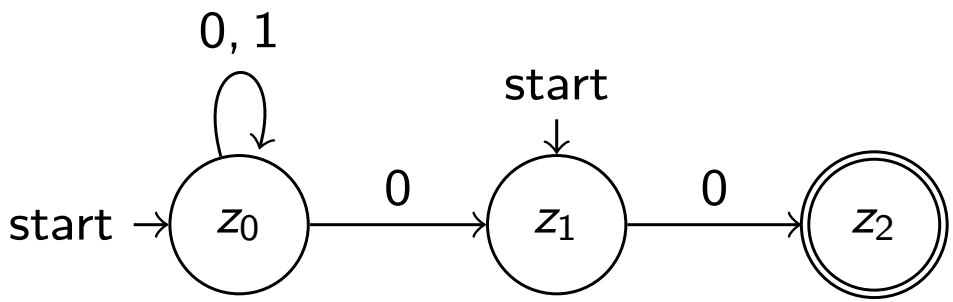
An NFA over the alphabet $\{0, 1\}$



Which strings does this automaton accept?

Example of non-deterministic finite automaton

An NFA over the alphabet $\{0, 1\}$



Which strings does this automaton accept?

It accepts strings ending with 00 or the length one string 0.

Recall: Power sets

For a set X, its *power set* is family

$$\mathcal{P}(X) := \{U \mid U \subsetneq X\}$$

of all its subsets.

Recall: Power sets

For a set X, its *power set* is family

$$\mathcal{P}(X) := \{U \mid U \subsetneq X\}$$

of all its subsets.

$$\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$$

Recall: Power sets

For a set X, its *power set* is family

$$\mathcal{P}(X) := \{U \mid U \subsetneq X\}$$

of all its subsets.

$$\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$$

Remark

- $\emptyset \in \mathcal{P}(X)$
- $X \in \mathcal{P}(X)$
- The power set has $|\mathcal{P}(X)| = 2^{|X|}$ elements.

NFAs: Formal definition

An NFA M is a 5 tuple

$$M = (Z, \Sigma, \delta, S, E)$$

where

NFAs: Formal definition

An NFA M is a 5 tuple

$$M = (Z, \Sigma, \delta, S, E)$$

where

- Z: Finite set of *states*

NFAs: Formal definition

An NFA M is a 5 tuple

$$M = (Z, \Sigma, \delta, S, E)$$

where

- Z: Finite set of *states*
- Σ: Input alphabet $Z \cap \Sigma = \emptyset$

NFAs: Formal definition

An NFA M is a 5 tuple

$$M = (Z, \Sigma, \delta, S, E)$$

where

- Z: Finite set of *states*
- Σ: Input alphabet $Z \cap \Sigma = \emptyset$
- $S \subset Z$: Set of initial states

NFAs: Formal definition

An NFA M is a 5 tuple

$$M = (Z, \Sigma, \delta, S, E)$$

where

- Z: Finite set of *states*
- Σ: Input alphabet $Z \cap \Sigma = \emptyset$
- $S \subset Z$: Set of initial states
- $E \subset Z$: Set of final / accepting states

NFAs: Formal definition

An NFA M is a 5 tuple

$$M = (Z, \Sigma, \delta, S, E)$$

where

- Z: Finite set of *states*
- Σ: Input alphabet $Z \cap \Sigma = \emptyset$
- $S \subset Z$: Set of initial states
- $E \subset Z$: Set of final / accepting states
- δ: $Z \times \Sigma \to \mathcal{P}(Z)$: State transition function

Power Set Construction

Extended transition function $\hat{\delta}$

δ can again be generalized from characters to strings:

$$\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$$

with

Power Set Construction

Extended transition function $\hat{\delta}$

δ can again be generalized from characters to strings:

$$\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$$

with

$$\hat{\delta}(Z', \varepsilon) = Z' \quad \text{for all} \quad Z' \subset Z$$

$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$$

Power Set Construction

Extended transition function $\hat{\delta}$

δ can again be generalized from characters to strings:

$$\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$$

with

$$\hat{\delta}(Z', \varepsilon) = Z' \quad \text{for all} \quad Z' \subset Z$$

$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$$

all the states that can be reached from states in Z' by following ax

Power Set Construction

Extended transition function $\hat{\delta}$

δ can again be generalized from characters to strings:

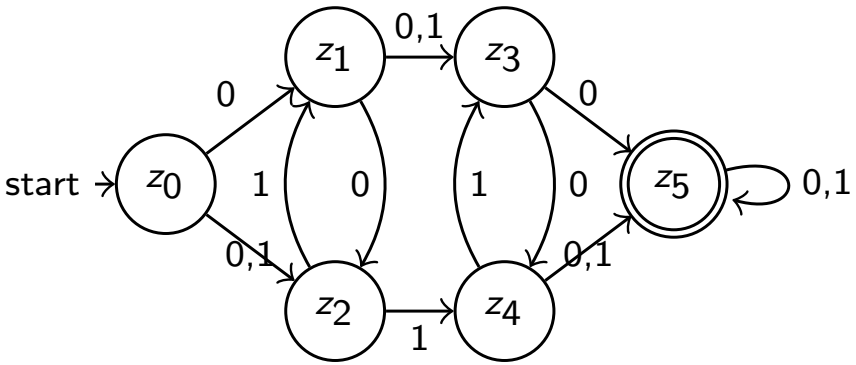
$$\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$$

with

$$\hat{\delta}(Z', \varepsilon) = Z' \quad \text{for all} \quad Z' \subset Z$$

$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$$

all the states that can be reached from states in Z' by following ax



Power Set Construction

Extended transition function $\hat{\delta}$

δ can again be generalized from characters to strings:

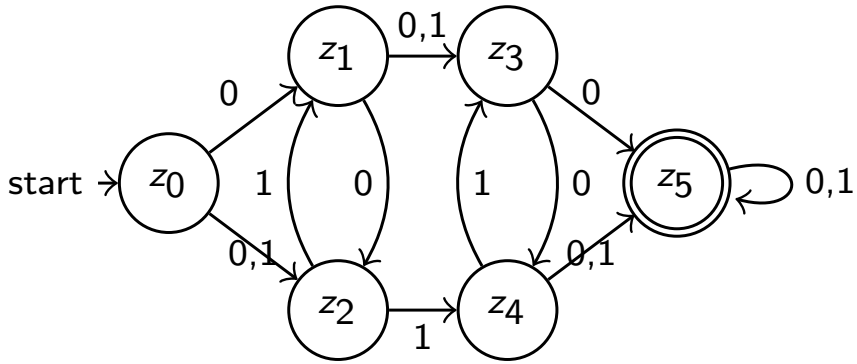
$$\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$$

with

$$\hat{\delta}(Z', \varepsilon) = Z' \quad \text{for all} \quad Z' \subset Z$$

$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$$

all the states that can be reached from states in Z' by following ax



$$\hat{\delta}(\{z_0, z_1\}, 001) = \{z_1, z_3, z_4, z_5\}$$

Power Set Construction

Extended transition function $\hat{\delta}$

δ can again be generalized from characters to strings:

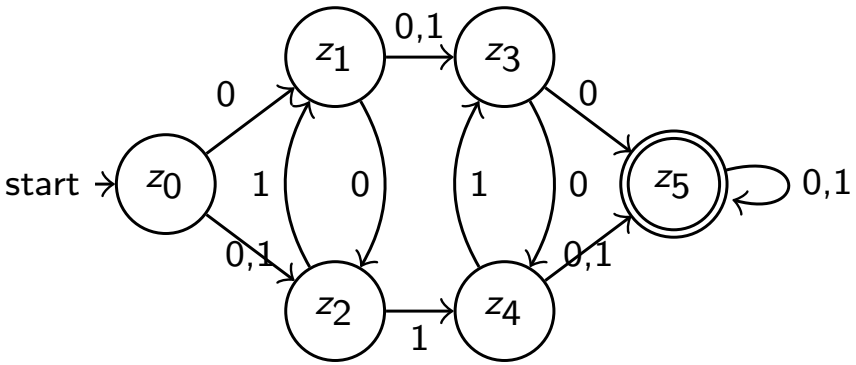
$$\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$$

with

$$\hat{\delta}(Z', \varepsilon) = Z' \quad \text{for all} \quad Z' \subset Z$$

$$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$$

all the states that can be reached from states in Z' by following ax



$$\hat{\delta}(\{z_0, z_1\}, 001) = \{z_1, z_3, z_4, z_5\}$$

$$\hat{\delta}(\{z_2, z_4\}, 110) = \{z_4, z_5\}$$

Equivalence of NFAs and DFAs

For a non-deterministic finite automaton $\mathcal{A} = (Z, \Sigma, \delta, S, E)$, the *language accepted by* $\mathcal{A}$ is

$$L(\mathcal{A}) := \{x \in \Sigma^* \mid \hat{\delta}(S, x) \cap E \neq \emptyset\} \ .$$

Equivalence of NFAs and DFAs

For a non-deterministic finite automaton $\mathcal{A} = (Z, \Sigma, \delta, S, E)$, the *language accepted by* $\mathcal{A}$ is

$$L(\mathcal{A}) := \{x \in \Sigma^* \mid \hat{\delta}(S, x) \cap E \neq \emptyset\} \ .$$

Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

Equivalence of NFAs and DFAs

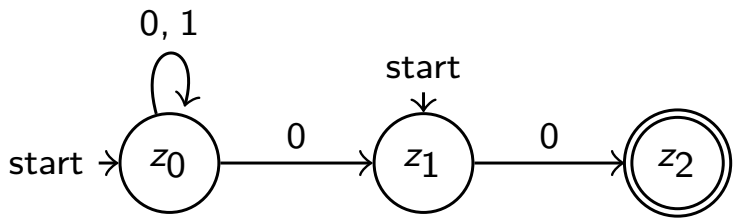
Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

Equivalence of NFAs and DFAs

Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

Proof Idea: Power set Construction

NFA

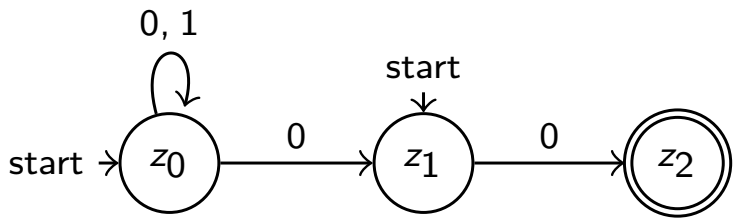


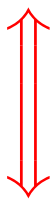
Equivalence of NFAs and DFAs

Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

Proof Idea: Power set Construction

NFA



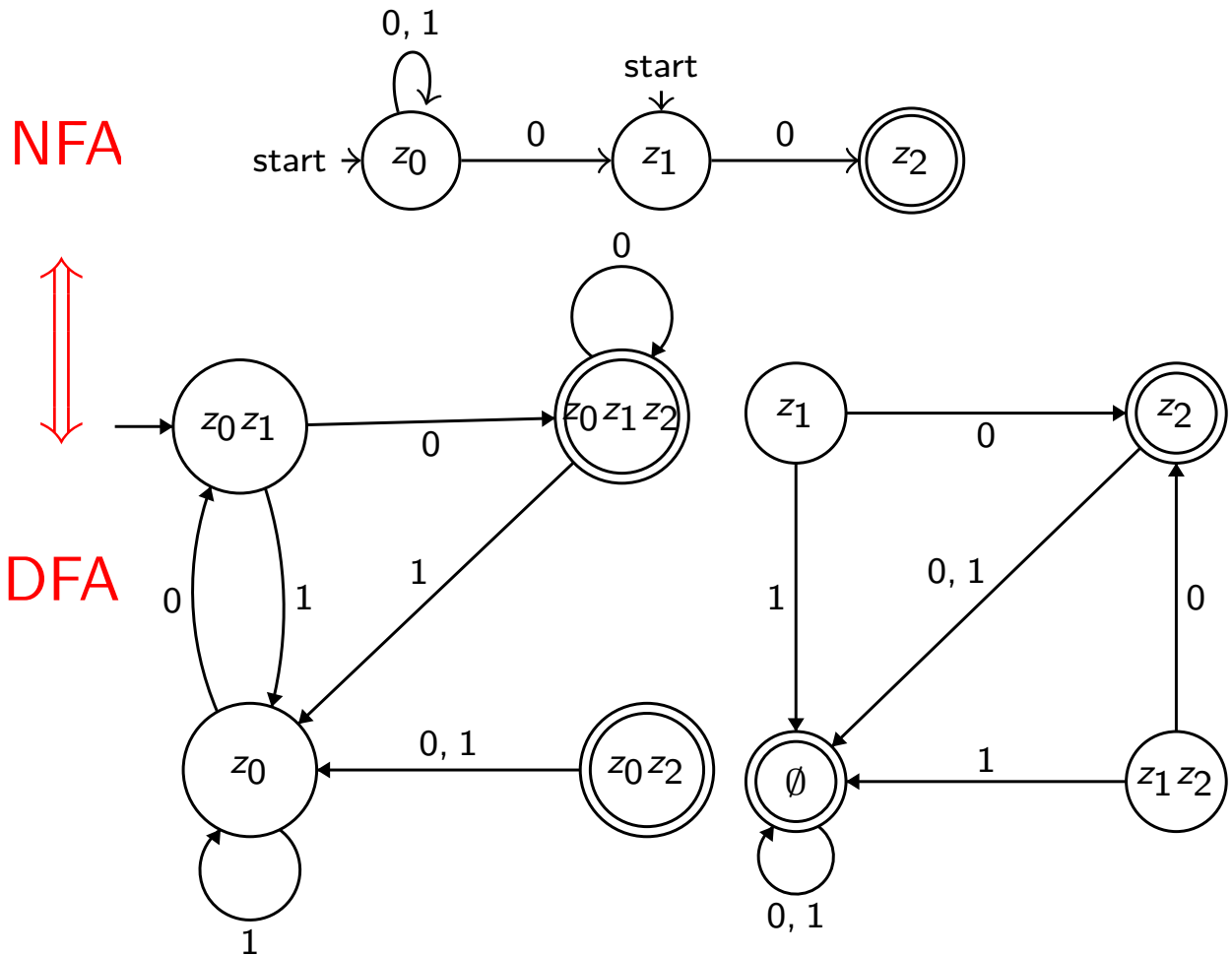


DFA

Equivalence of NFAs and DFAs

Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

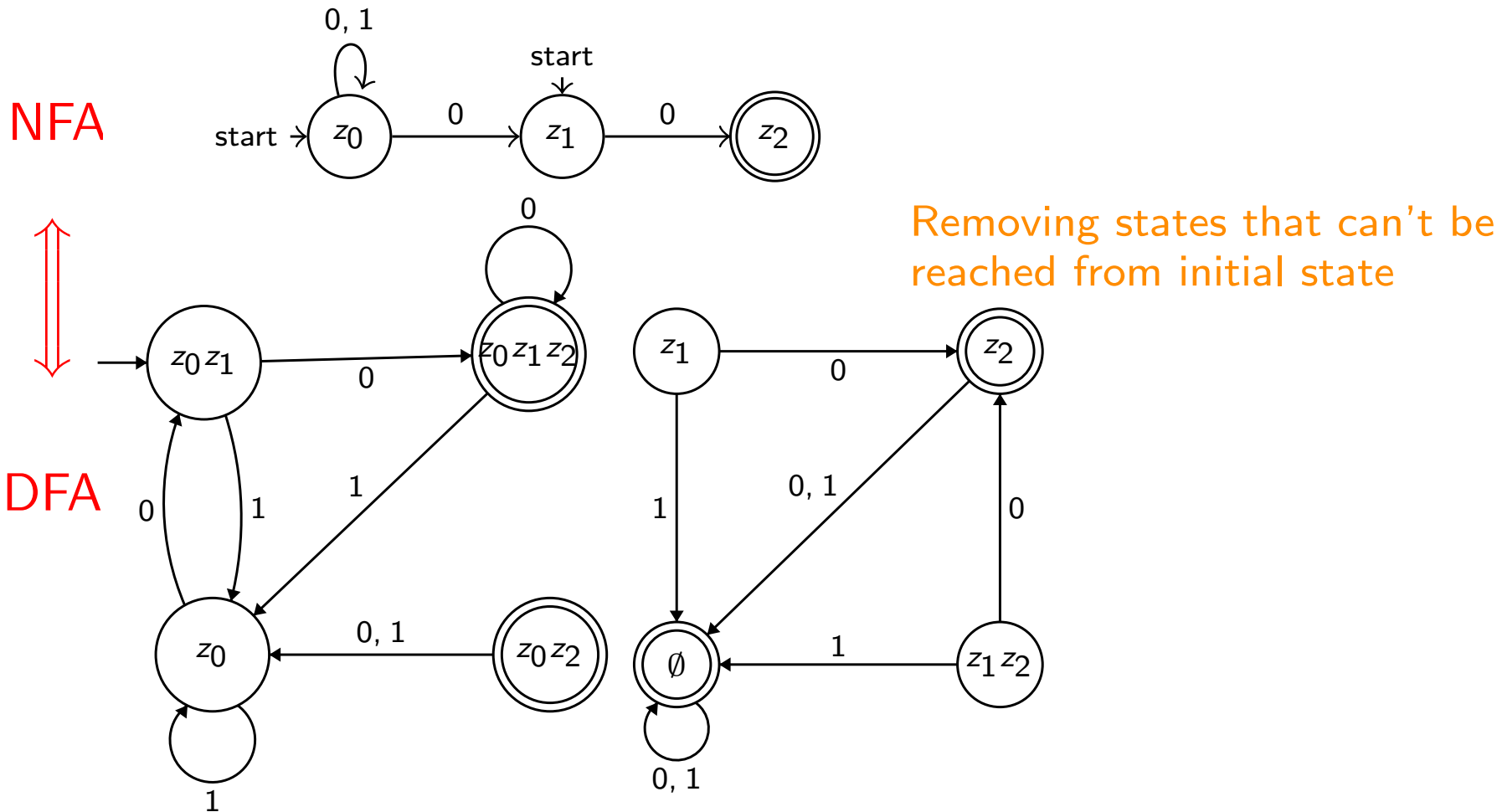
Proof Idea: Power set Construction



Equivalence of NFAs and DFAs

Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

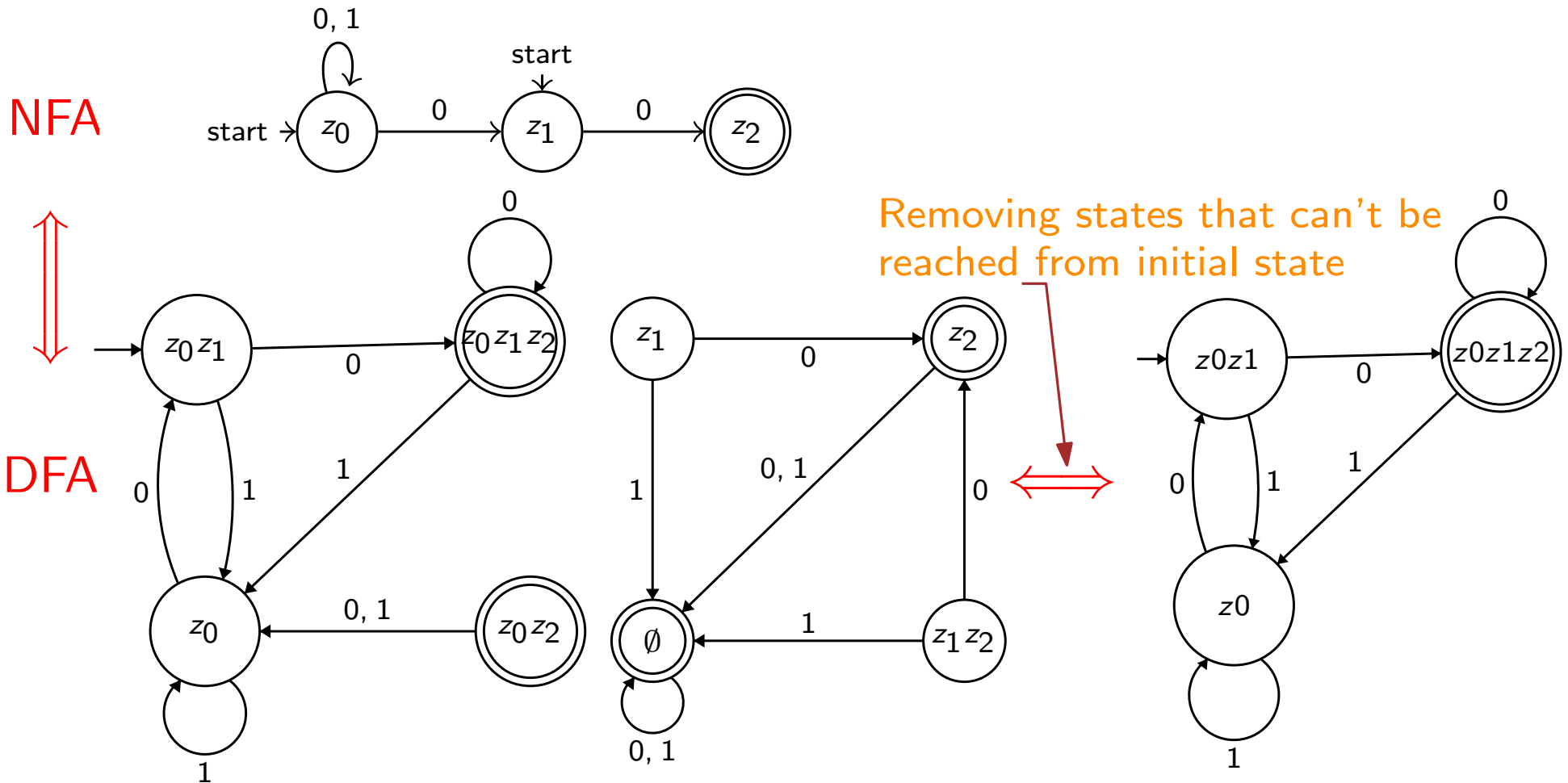
Proof Idea: Power set Construction



Equivalence of NFAs and DFAs

Theorem: Any language that can be represented by an NFA can also be represented by an equivalent DFA.

Proof Idea: Power set Construction



Equivalence of NFAs and DFAs

Proof of Theorem

Equivalence of NFAs and DFAs

Proof of Theorem

Let $M = (Z, \Sigma, \delta, S, E)$ be an NFA.

Equivalence of NFAs and DFAs

Proof of Theorem

Let $M = (Z, \Sigma, \delta, S, E)$ be an NFA.

Construct the power automaton of M, i.e. $M' = (\mathcal{P}(Z), \Sigma, \delta', Q_0, E')$ with

Equivalence of NFAs and DFAs

Proof of Theorem

Let $M = (Z, \Sigma, \delta, S, E)$ be an NFA.

Construct the power automaton of M, i.e. $M' = (\mathcal{P}(Z), \Sigma, \delta', Q_0, E')$ with

$$\delta'(Z', a) = \bigcup_{z \in Z'} \delta(z, a) = \hat{\delta}(Z', a), \;\; Z' \in \mathcal{P}(Z)$$

$$Z_0' = S$$

$$E' = \{Z' \in \mathcal{P}(Z) \mid Z' \cap E \neq \emptyset\}$$

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(M)$

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(M)$

$\Leftrightarrow$ $\exists$ sequence of subsets $Z'_1, \ldots, Z'_n$ of Z with
$\hat{\delta}(S, a_1) = Z'_1, \hat{\delta}(Z'_1, a_2) = Z'_2, \ldots, \hat{\delta}(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \cap E \neq \emptyset$

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(M)$

$\Leftrightarrow$ $\exists$ sequence of subsets $Z'_1, \ldots, Z'_n$ of Z with
$\hat{\delta}(S, a_1) = Z'_1, \hat{\delta}(Z'_1, a_2) = Z'_2, \ldots, \hat{\delta}(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \cap E \neq \emptyset$

$\Leftrightarrow$ $\exists$ sequence $Z'_1, \ldots, Z'_n$ of states of M' with
$\delta'(Z'_0, a_1) = Z'_1, \delta'(Z'_1, a_2) = Z'_2, \ldots, \delta'(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \in E'$

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(M)$

$\Leftrightarrow$ $\exists$ sequence of subsets $Z'_1, \ldots, Z'_n$ of Z with
$\hat{\delta}(S, a_1) = Z'_1, \hat{\delta}(Z'_1, a_2) = Z'_2, \ldots, \hat{\delta}(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \cap E \neq \emptyset$

$\Leftrightarrow$ $\exists$ sequence $Z'_1, \ldots, Z'_n$ of states of M' with
$\delta'(Z'_0, a_1) = Z'_1, \delta'(Z'_1, a_2) = Z'_2, \ldots, \delta'(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \in E'$

$\Leftrightarrow$ $\hat{\delta'}(Z'_0, x) \in E'$

Equivalence of NFAs and DFAs

Proof of Theorem (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(M)$

$\Leftrightarrow$ $\exists$ sequence of subsets $Z'_1, \ldots, Z'_n$ of Z with
$\hat{\delta}(S, a_1) = Z'_1, \hat{\delta}(Z'_1, a_2) = Z'_2, \ldots, \hat{\delta}(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \cap E \neq \emptyset$

$\Leftrightarrow$ $\exists$ sequence $Z'_1, \ldots, Z'_n$ of states of M' with
$\delta'(Z'_0, a_1) = Z'_1, \delta'(Z'_1, a_2) = Z'_2, \ldots, \delta'(Z'_{n-1}, a_n) = Z'_n$ and $Z'_n \in E'$

$\Leftrightarrow$ $\hat{\delta'}(Z'_0, x) \in E'$

$\Leftrightarrow$ $x \in L(M')$

Equivalence of DFA, NFA, Regular Grammars and Regular Expressions

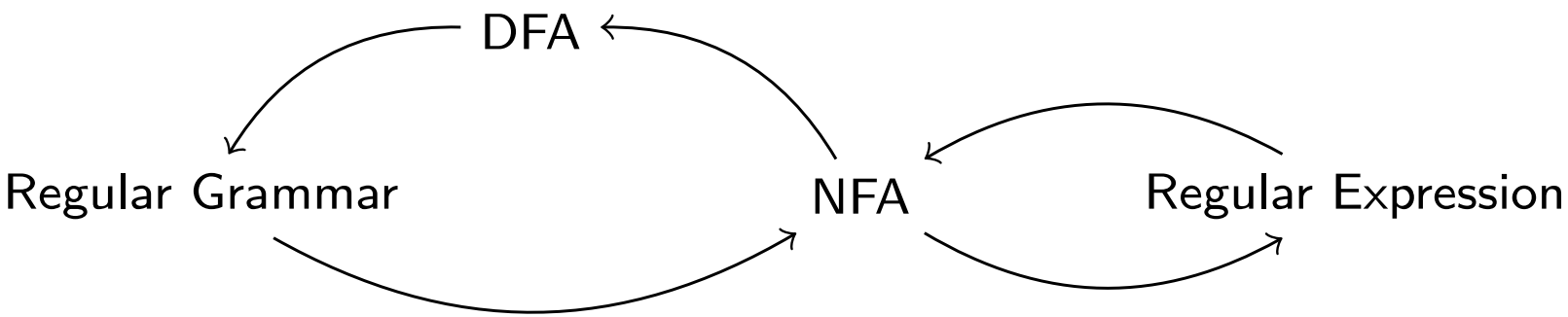
Expressiveness of D/NFAs and regular gramma

Theorem: DFA, NFA, regular grammars and regular expressions are equally expressive, meaning that they produce/recognize the same collection of languages.

Expressiveness of D/NFAs and regular gramma

Theorem: DFA, NFA, regular grammars and regular expressions are equally expressive, meaning that they produce/recognize the same collection of languages.

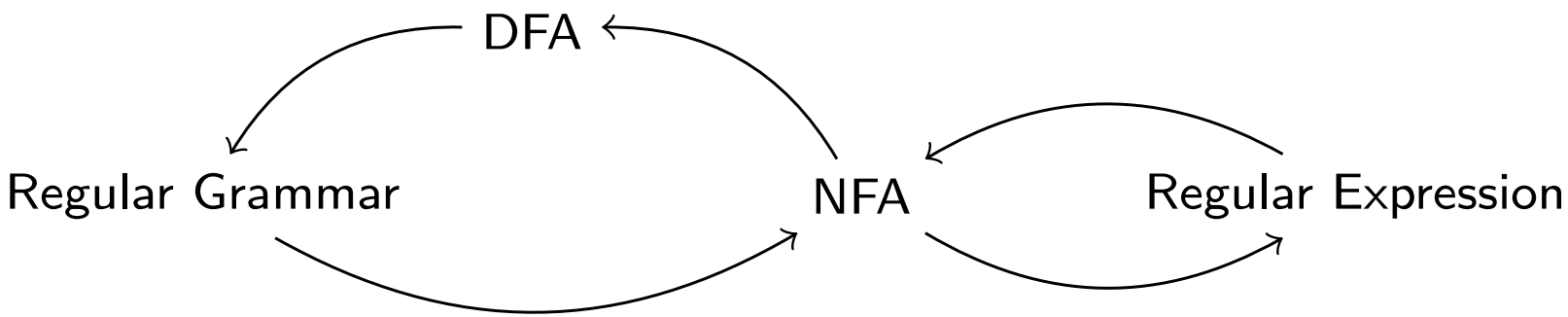
Technique of the proof



Expressiveness of D/NFAs and regular gramma

Theorem: DFA, NFA, regular grammars and regular expressions are equally expressive, meaning that they produce/recognize the same collection of languages.

Technique of the proof

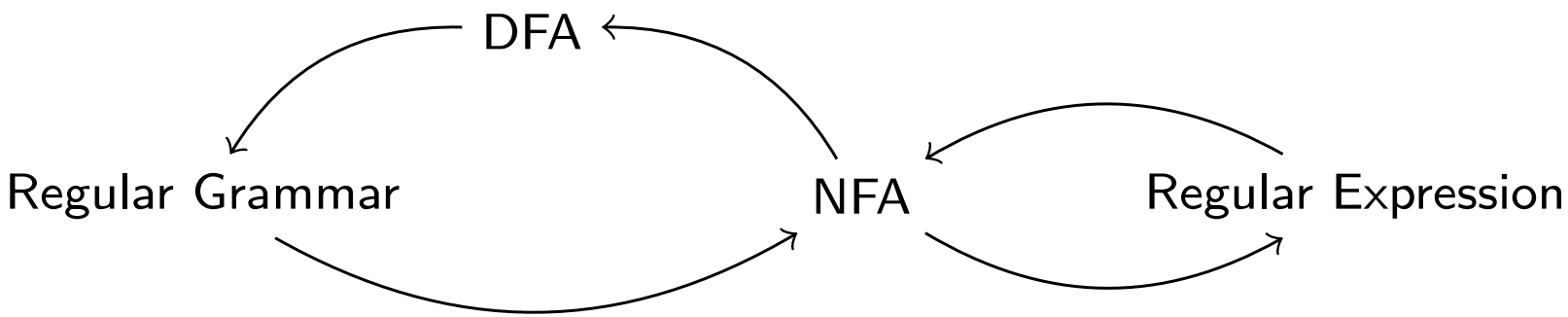


Already proved: NFA $\rightarrow$ DFA and DFA $\rightarrow$ Regular Grammar.

Expressiveness of D/NFAs and regular gramma

Theorem: DFA, NFA, regular grammars and regular expressions are equally expressive, meaning that they produce/recognize the same collection of languages.

Technique of the proof



Already proved: NFA $\rightarrow$ DFA and DFA $\rightarrow$ Regular Grammar.

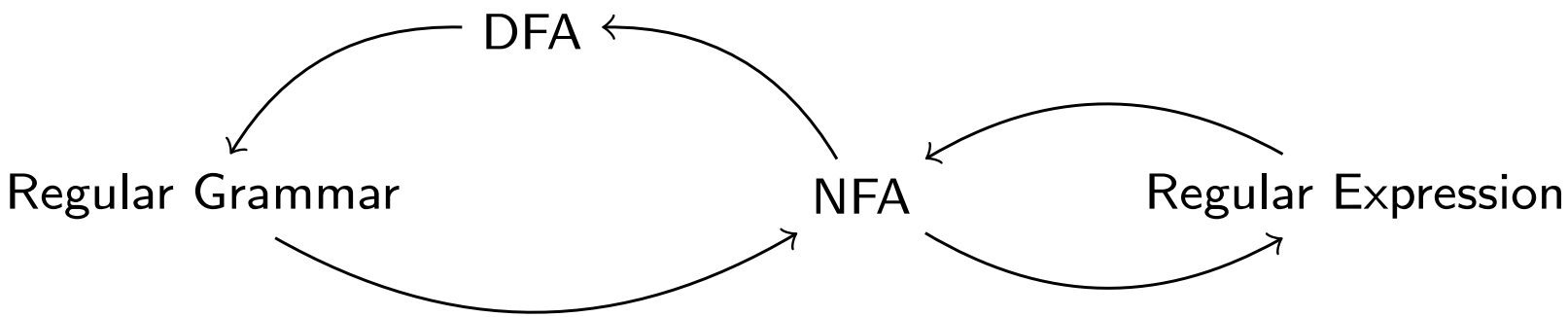
It remains to show that

- Regular Grammar $\rightarrow$ NFA

Expressiveness of D/NFAs and regular gramma

Theorem: DFA, NFA, regular grammars and regular expressions are equally expressive, meaning that they produce/recognize the same collection of languages.

Technique of the proof



Already proved: NFA $\rightarrow$ DFA and DFA $\rightarrow$ Regular Grammar.

It remains to show that

- Regular Grammar $\rightarrow$ NFA
- NFA $\leftrightarrow$ Regular expression

Regular Grammar $\rightarrow$ NFA

Theorem: For any regular grammar $G = (V, \Sigma, P, S)$ there exists an NFA $\mathcal{A} = (Z, \Sigma, \delta, S', E)$ with $L(\mathcal{A}) = L(G)$.

Regular Grammar $\rightarrow$ NFA

Theorem: For any regular grammar $G = (V, \Sigma, P, S)$ there exists an NFA $\mathcal{A} = (Z, \Sigma, \delta, S', E)$ with $L(\mathcal{A}) = L(G)$.

Grammar	Languages	Production rules (constraints)
Type 3	Regular	$A \rightarrow \mathrm{a}$, $A \rightarrow \mathrm{a}B$

Regular Grammar $\rightarrow$ NFA

Theorem: For any regular grammar $G = (V, \Sigma, P, S)$ there exists an NFA $\mathcal{A} = (Z, \Sigma, \delta, S', E)$ with $L(\mathcal{A}) = L(G)$.

Grammar	Languages	Production rules (constraints)
Type 3	Regular	$A \rightarrow$ a, $A \rightarrow$ aB

Proof

$Z = V \cup \{X\}, X \notin V$

$S' = \{S\}$

$$E = \begin{cases} \{S, X\} & \text{if } (S \rightarrow \varepsilon) \in P \\ \{X\} & \text{else} \end{cases}$$

$B \in \delta(A, a)$ if $(A \rightarrow aB) \in P$

$X \in \delta(A, a)$ if $(A \rightarrow a) \in P$

Regular Grammar $\rightarrow$ NFA

Theorem: For any regular grammar $G = (V, \Sigma, P, S)$ there exists an NFA $\mathcal{A} = (Z, \Sigma, \delta, S', E)$ with $L(\mathcal{A}) = L(G)$.

Proof

$Z = V \cup \{X\}, X \notin V$

$S' = \{S\}$

$$E = \begin{cases} \{S, X\} & \text{if } (S \rightarrow \varepsilon) \in P \\ \{X\} & \text{else} \end{cases}$$

$B \in \delta(A, a)$ if $(A \rightarrow aB) \in P$

$X \in \delta(A, a)$ if $(A \rightarrow a) \in P$

Example

$S \rightarrow \mathrm{a}A \mid \mathrm{b}B$

$A \rightarrow \mathrm{a}B \mid \mathrm{a}C$

$B \rightarrow \mathrm{b}A \mid \mathrm{b}C$

$C \rightarrow \mathrm{c}C$

$A \rightarrow \mathrm{a}$

$B \rightarrow \mathrm{b}$

$C \rightarrow \mathrm{c}$

$S \rightarrow \varepsilon$

Regular Grammar $\rightarrow$ NFA

Theorem: For any regular grammar $G = (V, \Sigma, P, S)$ there exists an NFA $\mathcal{A} = (Z, \Sigma, \delta, S', E)$ with $L(\mathcal{A}) = L(G)$.

Proof

$$Z = V \cup \{X\}, X \notin V$$

$$S' = \{S\}$$

$$E = \begin{cases} \{S, X\} & \text{if } (S \rightarrow \varepsilon) \in P \\ \{X\} & \text{else} \end{cases}$$

$$B \in \delta(A, a) \text{ if } (A \rightarrow aB) \in P$$

$$X \in \delta(A, a) \text{ if } (A \rightarrow a) \in P$$

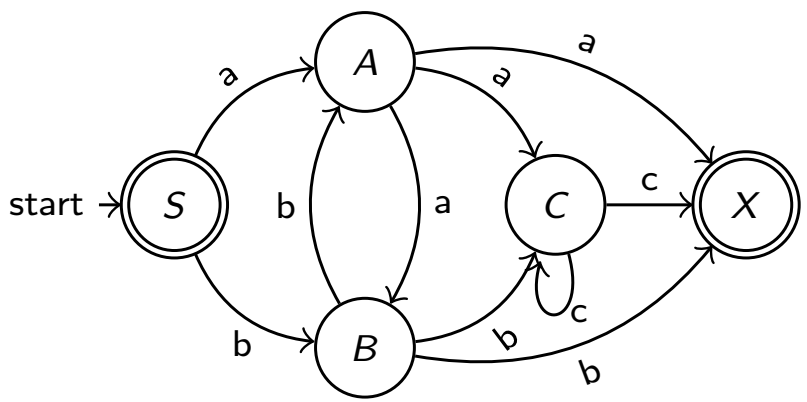
Example

$S \rightarrow \mathrm{a}A \mid \mathrm{b}B$ $\quad$ $A \rightarrow \mathrm{a}$

$A \rightarrow \mathrm{a}B \mid \mathrm{a}C$ $\quad$ $B \rightarrow \mathrm{b}$

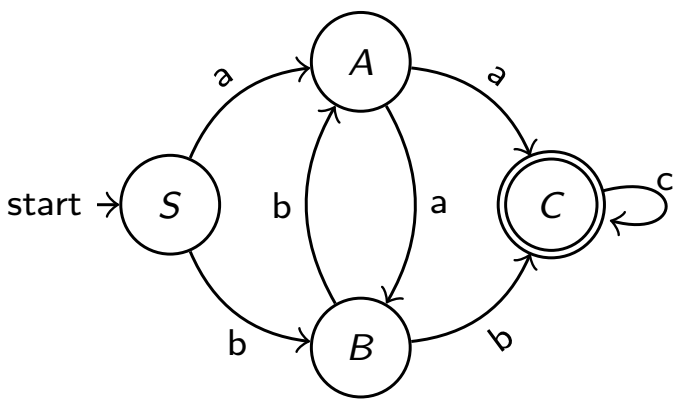
$B \rightarrow \mathrm{b}A \mid \mathrm{b}C$ $\quad$ $C \rightarrow \mathrm{c}$

$C \rightarrow \mathrm{c}C$ $\quad$ $S \rightarrow \varepsilon$



Regular Grammar $\rightarrow$ NFA

Theorem: For any regular grammar $G = (V, \Sigma, P, S)$ there exists an NFA $\mathcal{A} = (Z, \Sigma, \delta, S', E)$ with $L(\mathcal{A}) = L(G)$.



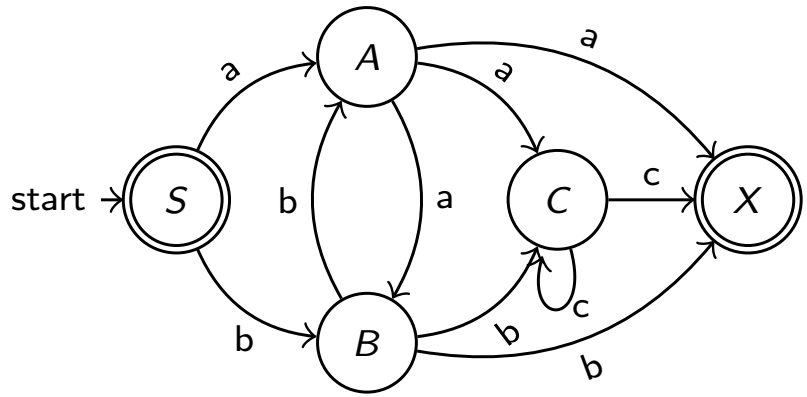
Example

$S \rightarrow aA \mid bB$ $\quad A \rightarrow a$

$A \rightarrow aB \mid aC$ $\quad B \rightarrow b$

$B \rightarrow bA \mid bC$ $\quad C \rightarrow c$

$C \rightarrow cC$ $\quad S \rightarrow \varepsilon$



Regular Grammar $\rightarrow$ NFA

Proof (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

Regular Grammar $\rightarrow$ NFA

Proof (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(G)$

Regular Grammar $\rightarrow$ NFA

Proof (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(G)$

$\Leftrightarrow$ $\exists$ sequence of variables $A_1, \ldots, A_{n-1}$ with
$S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \cdots \Rightarrow a_1 a_2 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$

Regular Grammar $\rightarrow$ NFA

Proof (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(G)$

$\Leftrightarrow$ $\exists$ sequence of variables $A_1, \ldots, A_{n-1}$ with
$S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \cdots \Rightarrow a_1 a_2 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$

$\Leftrightarrow$ $\exists$ sequence of states $A_1, \ldots, A_{n-1}$ with
$\delta(S, a_1) \ni A_1, \delta(A_1, a_2) \ni A_2, \ldots, \delta(A_{n-1}, a_n) \ni X$

Regular Grammar $\rightarrow$ NFA

Proof (cont'd)

Now for a string $x = a_1 a_2 \ldots a_n \in \Sigma^*$ we have

$x \in L(G)$

$\Leftrightarrow$ $\exists$ sequence of variables $A_1, \ldots, A_{n-1}$ with
$S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \cdots \Rightarrow a_1 a_2 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$

$\Leftrightarrow$ $\exists$ sequence of states $A_1, \ldots, A_{n-1}$ with
$\delta(S, a_1) \ni A_1, \delta(A_1, a_2) \ni A_2, \ldots, \delta(A_{n-1}, a_n) \ni X$

$\Leftrightarrow$ $x \in L(M)$

Regular Expressions $\rightarrow$ NFA

Theorem: For any regular expression R, the language $L(R)$ can be recognized by a NFA.

Regular Expressions $\rightarrow$ NFA

Theorem: For any regular expression R, the language $L(R)$ can be recognized by a NFA.

Recall: Regular expressions

0^*10^*

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

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

$01+10$

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

Regular Expressions $\rightarrow$ NFA

Theorem: For any regular expression R, the language $L(R)$ can be recognized by a NFA.

Recall: Regular expressions

0^*10^*

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

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

$01+10$

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

$\emptyset$, ε and each $a \in \Sigma$ are regular expressions with $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$

If R_1 and R_2 are regular expressions then the following are also regular expressions:

- $(R_1 + R_2)$ with $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $(R_1 R_2)$ with $L(R_1 R_2) = L(R_1)L(R_2)$
- (R_1^*) with $L(R_1^*) = L(R_1)^*$

Regular Expressions $\rightarrow$ NFA

Theorem: For any regular expression R, the language $L(R)$ can be recognized by a NFA.

Recall: Regular expressions

0^*10^*

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

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

$01+10$

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

$\emptyset$, ε and each $a \in \Sigma$ are regular expressions with $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$

If R_1 and R_2 are regular expressions then the following are also regular expressions:

- $(R_1 + R_2)$ with $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $(R_1 R_2)$ with $L(R_1 R_2) = L(R_1)L(R_2)$
- (R_1^*) with $L(R_1^*) = L(R_1)^*$

Proof of Theorem

Let R be a regular expression with $L = L(R)$. For $R = \emptyset$, $R = \varepsilon$, and $R = a$, the following DFAs recognize $L(R)$.

Regular Expressions $\rightarrow$ NFA

Theorem: For any regular expression R, the language $L(R)$ can be recognized by a NFA.

Recall: Regular expressions

0^*10^*

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

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

$01+10$

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

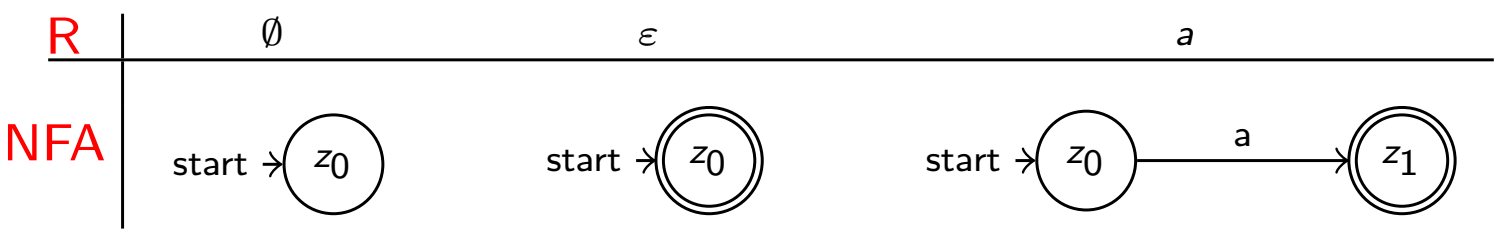
$\emptyset$, ε and each $a \in \Sigma$ are regular expressions with $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$

If R_1 and R_2 are regular expressions then the following are also regular expressions:

- $(R_1 + R_2)$ with $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $(R_1 R_2)$ with $L(R_1 R_2) = L(R_1)L(R_2)$
- (R_1^*) with $L(R_1^*) = L(R_1)^*$

Proof of Theorem

Let R be a regular expression with $L = L(R)$. For $R = \emptyset$, $R = \varepsilon$, and $R = a$, the following DFAs recognize $L(R)$.



Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

Assume that R_1 and R_2 are regular expressions and $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$ and $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ be NFAs with $L(M_1) = L(R_1)$ and $L(M_2) = LR_2)$.

Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

Assume that R_1 and R_2 are regular expressions and $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$ and $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ be NFAs with $L(M_1) = L(R_1)$ and $L(M_2) = LR_2)$.

We will in the following show that $R = R_1 + R_2$, $R = R_1 R_2$, and $R = R_1^*$ can be generated by NFAs.

Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

Assume that R_1 and R_2 are regular expressions and $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$ and $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ be NFAs with $L(M_1) = L(R_1)$ and $L(M_2) = LR_2)$.

We will in the following show that $R = R_1 + R_2$, $R = R_1 R_2$, and $R = R_1^*$ can be generated by NFAs.

$R = R_1 + R_2$

We construct the union automaton
$M = (Z_1 \cup Z_2, \Sigma, \delta_1 \cup \delta_2, S_1 \cup S_2, E_1 \cup E_2)$
and then have $L(M) = L(R)$.

Regular Expressions $\rightarrow$ NFA

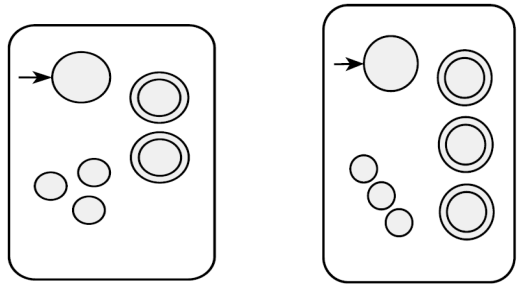
Proof (cont'd)

Assume that R_1 and R_2 are regular expressions and $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$ and $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ be NFAs with $L(M_1) = L(R_1)$ and $L(M_2) = LR_2)$.

We will in the following show that $R = R_1 + R_2$, $R = R_1 R_2$, and $R = R_1^*$ can be generated by NFAs.

$R = R_1 + R_2$

We construct the union automaton $M = (Z_1 \cup Z_2, \Sigma, \delta_1 \cup \delta_2, S_1 \cup S_2, E_1 \cup E_2)$ and then have $L(M) = L(R)$.



M_1 M_2

Regular Expressions $\rightarrow$ NFA

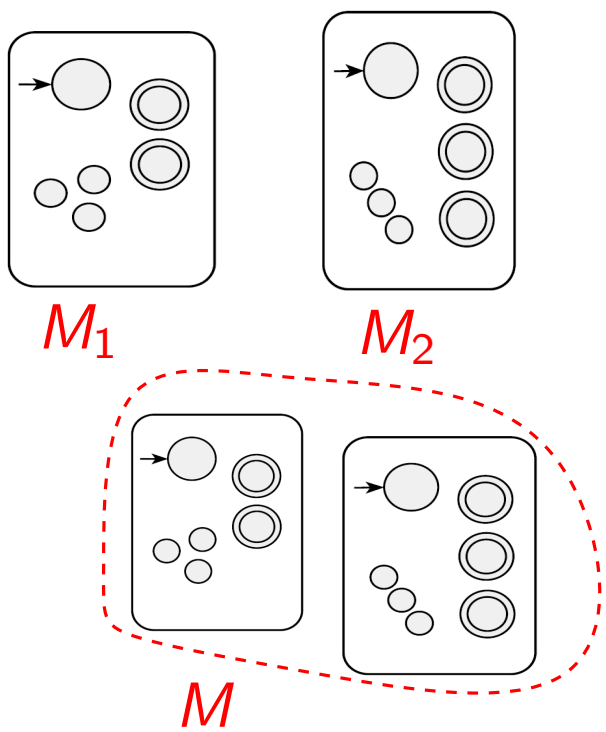
Proof (cont'd)

Assume that R_1 and R_2 are regular expressions and $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$ and $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ be NFAs with $L(M_1) = L(R_1)$ and $L(M_2) = LR_2)$.

We will in the following show that $R = R_1 + R_2$, $R = R_1 R_2$, and $R = R_1^*$ can be generated by NFAs.

$R = R_1 + R_2$

We construct the union automaton $M = (Z_1 \cup Z_2, \Sigma, \delta_1 \cup \delta_2, S_1 \cup S_2, E_1 \cup E_2)$ and then have $L(M) = L(R)$.



Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2

Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2
- If $\varepsilon \in L(M_1)$, we include initial states of M_2 to initial states of M.

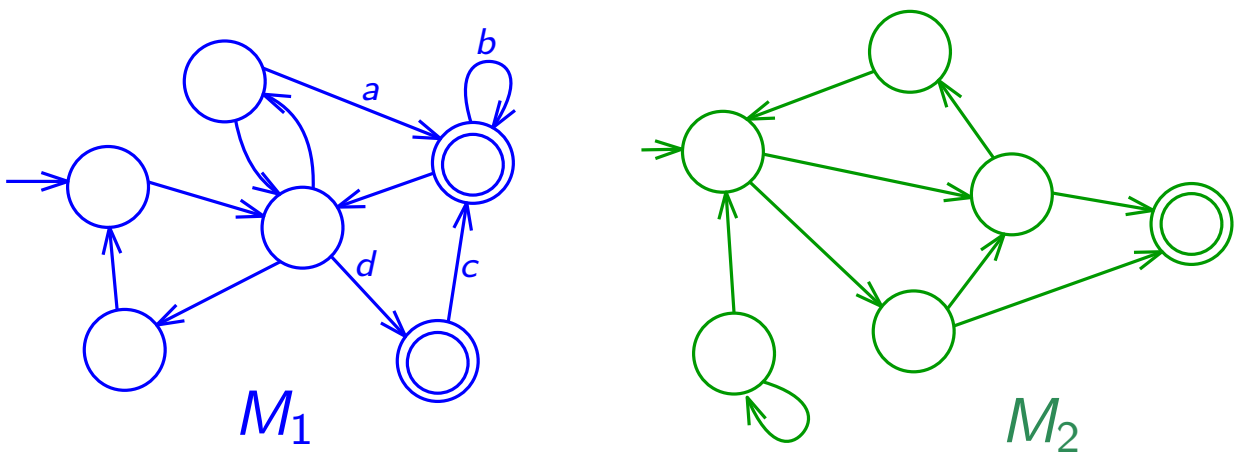
Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2
- If $\varepsilon \in L(M_1)$, we include initial states of M_2 to initial states of M.



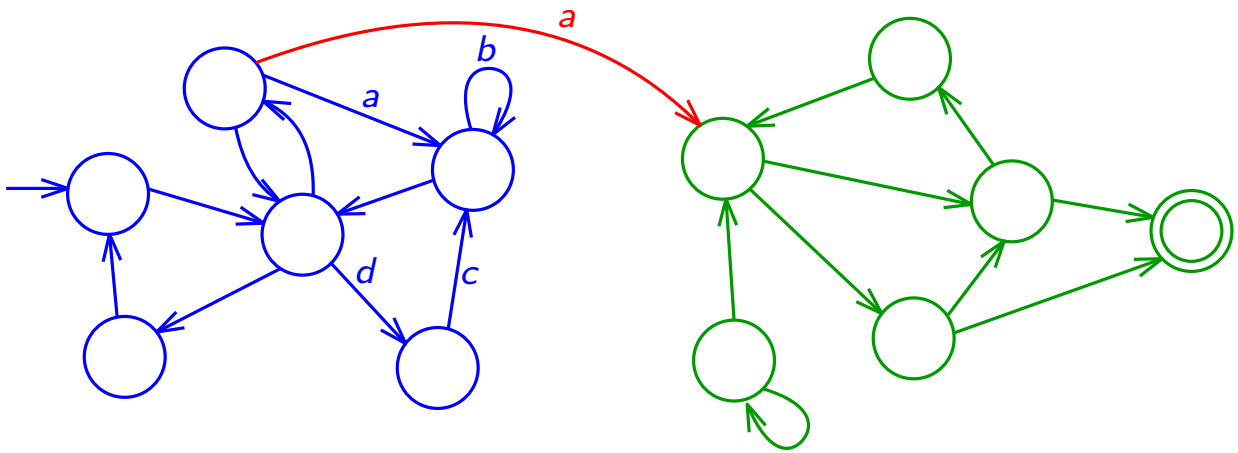
Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2
- If $\varepsilon \in L(M_1)$, we include initial states of M_2 to initial states of M.



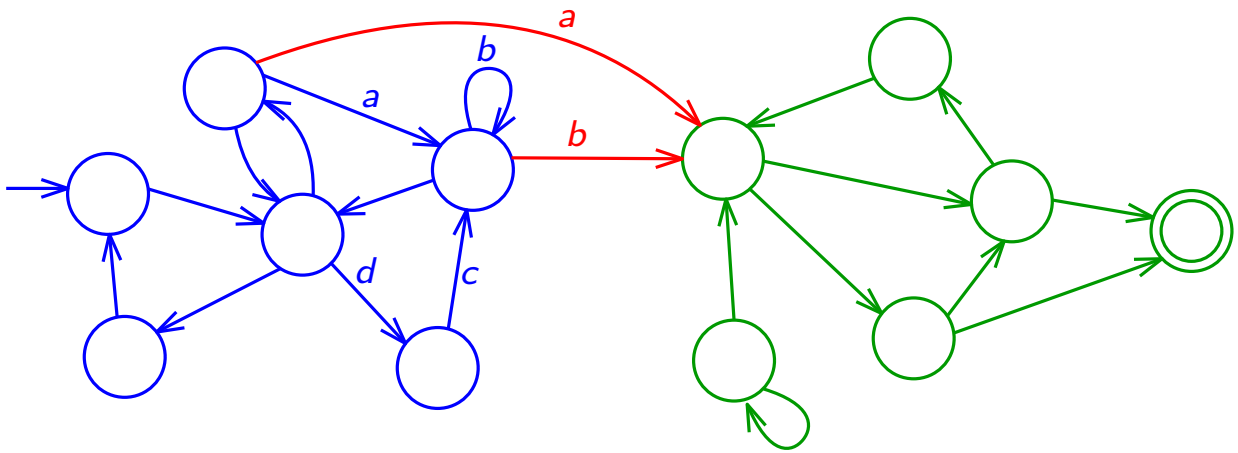
Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2
- If $\varepsilon \in L(M_1)$, we include initial states of M_2 to initial states of M.



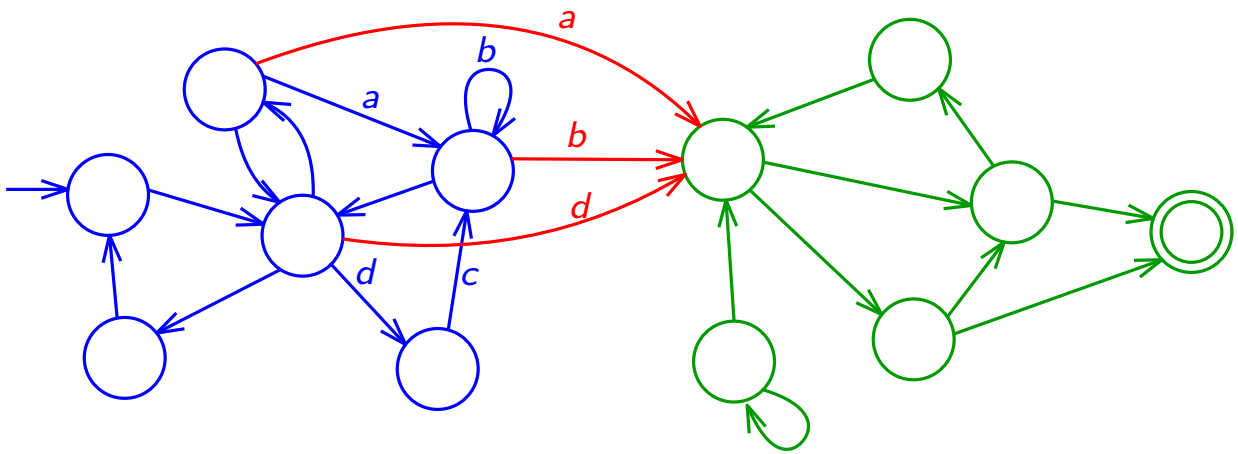
Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2
- If $\varepsilon \in L(M_1)$, we include initial states of M_2 to initial states of M.



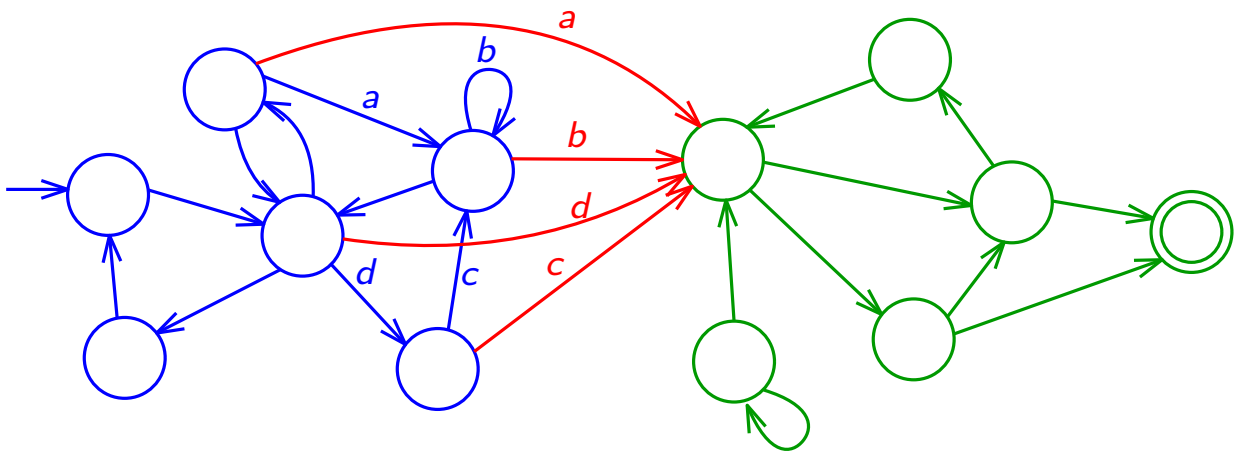
Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1 R_2$

The concatenation can be accepted by an NFA M obtained by connecting M_1 and M_2 in series. To this end, in M_1 we take those states that have an arrow to a final state and connect them with similar transitions to all initial states of M_2.

- Initial states of M = Initial states of M_1
  Final states of M = Final states of M_2
- If $\varepsilon \in L(M_1)$, we include initial states of M_2 to initial states of M.



M

Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

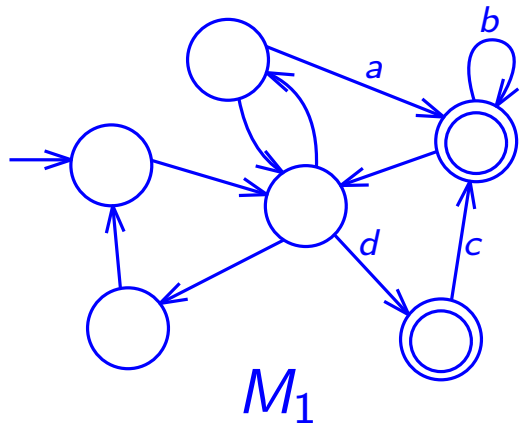
$R = R_1^*$

Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:



M_1

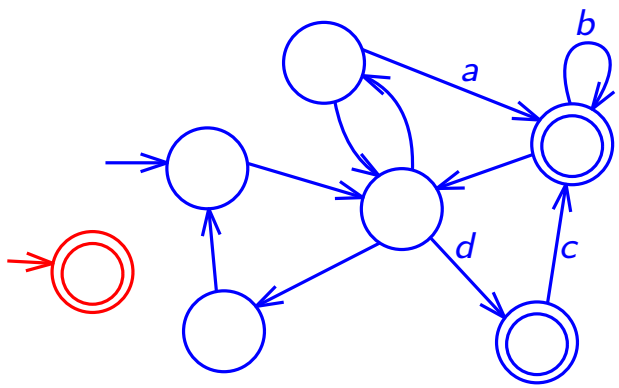
Regular Expressions $\rightarrow$ NFA

Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:

First, we add a new state $\tilde{z}$ which is start and initial state but has no further connections. This ensured that the empty string can be derived.



Regular Expressions $\rightarrow$ NFA

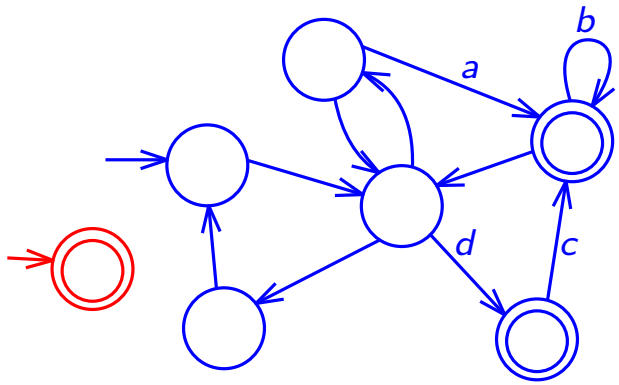
Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:

First, we add a new state $\tilde{z}$ which is start and initial state but has no further connections. This ensured that the empty string can be derived.

Then we take those states in M_1 that have an arrow to a final state and connect them with similar transitions to all initial states of M_1. The resulting NFA accepts the same strings as the regular expression $R = R_1^*$ generates.



Regular Expressions $\rightarrow$ NFA

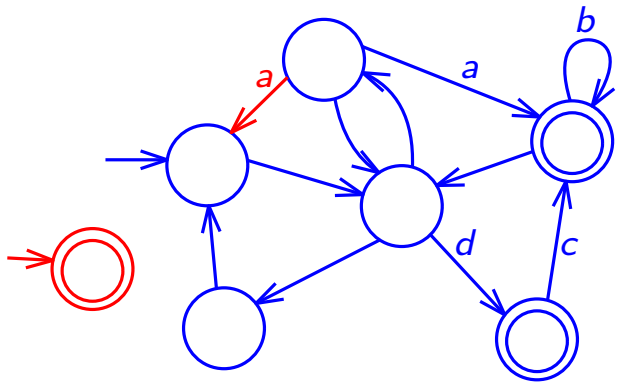
Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:

First, we add a new state $\tilde{z}$ which is start and initial state but has no further connections. This ensured that the empty string can be derived.

Then we take those states in M_1 that have an arrow to a final state and connect them with similar transitions to all initial states of M_1. The resulting NFA accepts the same strings as the regular expression $R = R_1^*$ generates.



Regular Expressions $\rightarrow$ NFA

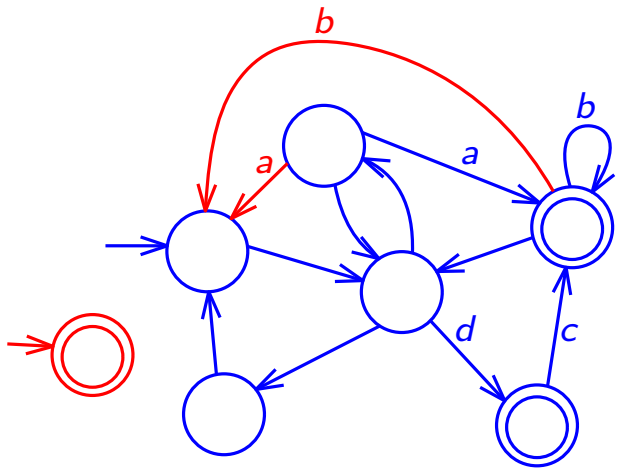
Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:

First, we add a new state $\tilde{z}$ which is start and initial state but has no further connections. This ensured that the empty string can be derived.

Then we take those states in M_1 that have an arrow to a final state and connect them with similar transitions to all initial states of M_1. The resulting NFA accepts the same strings as the regular expression $R = R_1^*$ generates.



Regular Expressions $\rightarrow$ NFA

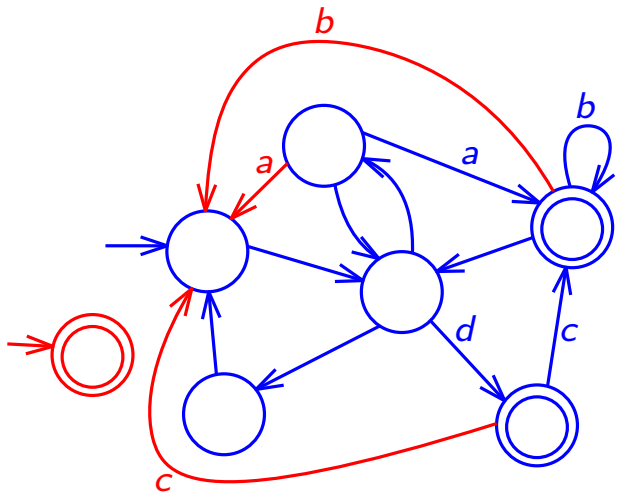
Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:

First, we add a new state $\tilde{z}$ which is start and initial state but has no further connections. This ensured that the empty string can be derived.

Then we take those states in M_1 that have an arrow to a final state and connect them with similar transitions to all initial states of M_1. The resulting NFA accepts the same strings as the regular expression $R = R_1^*$ generates.



Regular Expressions $\rightarrow$ NFA

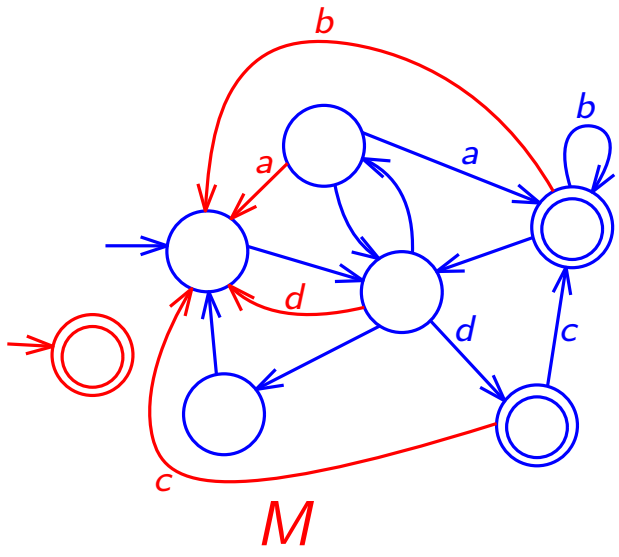
Proof (cont'd)

$R = R_1^*$

We take M_1 and modify it in the following way:

First, we add a new state $\tilde{z}$ which is start and initial state but has no further connections. This ensured that the empty string can be derived.

Then we take those states in M_1 that have an arrow to a final state and connect them with similar transitions to all initial states of M_1. The resulting NFA accepts the same strings as the regular expression $R = R_1^*$ generates.



Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

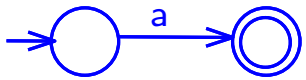
Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

a

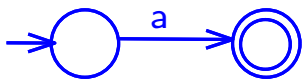
Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

a 

Constructing a DFA for a regular expression

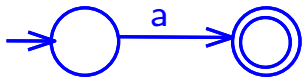
Construct a DFA recognizing $(a+b)^*aba$.

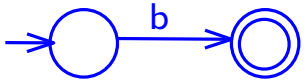
a 

b

Constructing a DFA for a regular expression

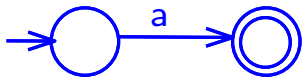
Construct a DFA recognizing $(a+b)^*aba$.

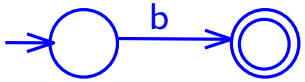
a 

b 

Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

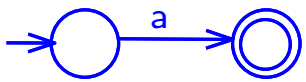
a 

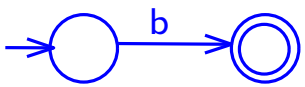
b 

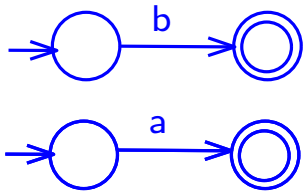
a+b

Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

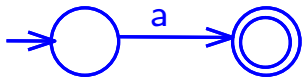
a 

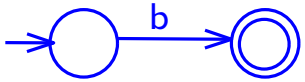
b 

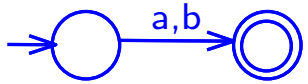
a+b 

Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

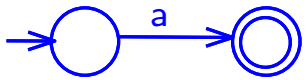
a 

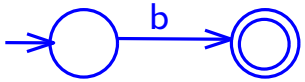
b 

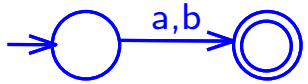
a+b 

Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

a 

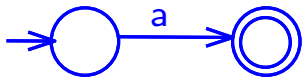
b 

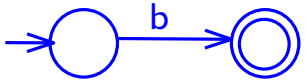
a+b 

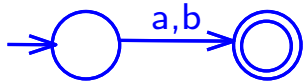
$(a+b)^*$

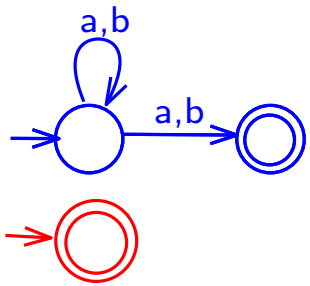
Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

a 

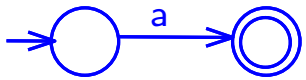
b 

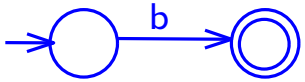
a+b 

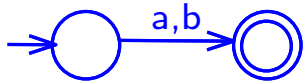
$(a+b)^*$ 

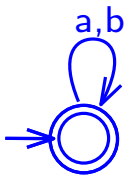
Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

a 

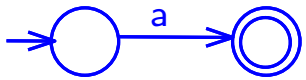
b 

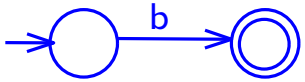
a+b 

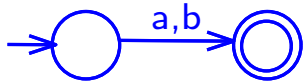
$(a+b)^*$ 

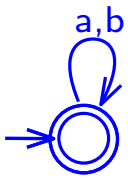
Constructing a DFA for a regular expression

Construct a DFA recognizing $(a+b)^*aba$.

a 

b 

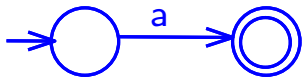
a+b 

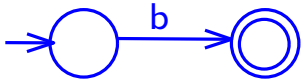
$(a+b)^*$ 

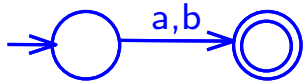
aba

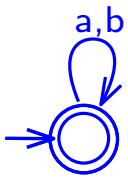
Constructing a DFA for a regular expression

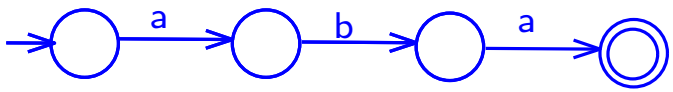
Construct a DFA recognizing $(a+b)^*aba$.

a 

b 

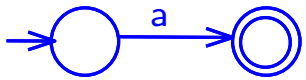
a+b 

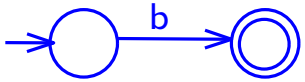
$(a+b)^*$ 

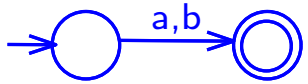
aba 

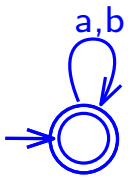
Constructing a DFA for a regular expression

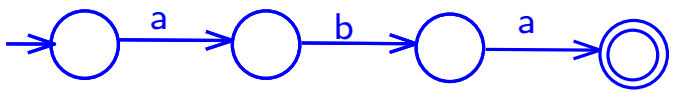
Construct a DFA recognizing $(a+b)^*aba$.

a 

b 

a+b 

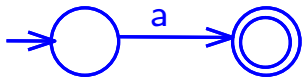
$(a+b)^*$ 

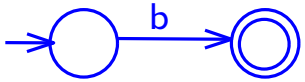
aba 

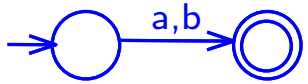
$(a+b)^*aba$

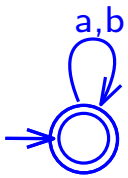
Constructing a DFA for a regular expression

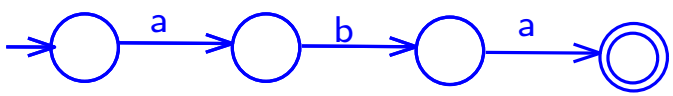
Construct a DFA recognizing $(a+b)^*aba$.

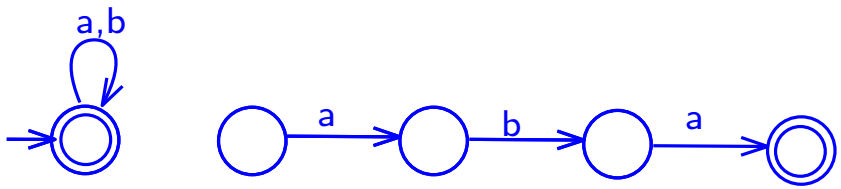
a 

b 

a+b 

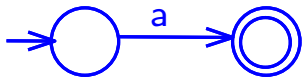
$(a+b)^*$ 

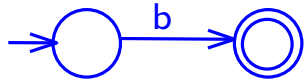
aba 

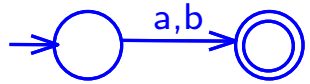
$(a+b)^*aba$ 

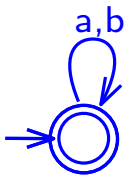
Constructing a DFA for a regular expression

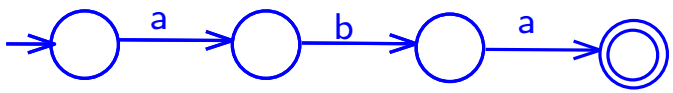
Construct a DFA recognizing $(a+b)^*aba$.

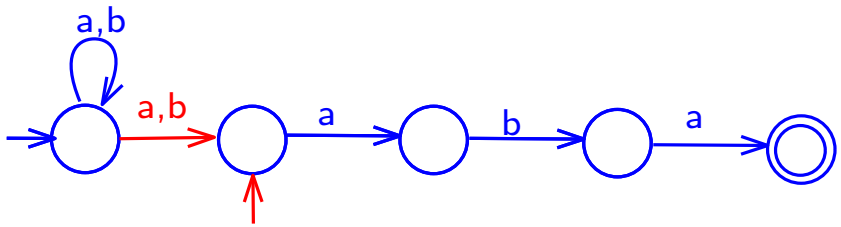
a 

b 

a+b 

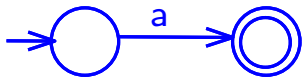
$(a+b)^*$ 

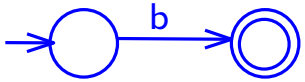
aba 

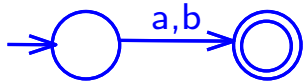
$(a+b)^*aba$ 

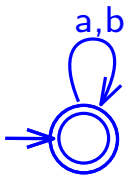
Constructing a DFA for a regular expression

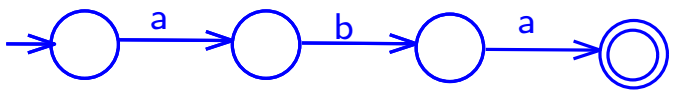
Construct a DFA recognizing $(a+b)^*aba$.

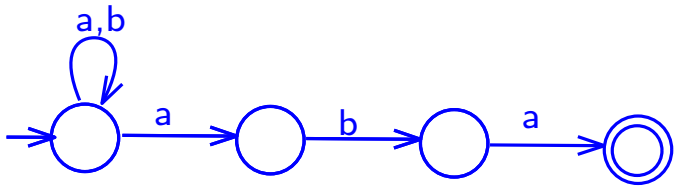
a 

b 

a+b 

$(a+b)^*$ 

aba 

$(a+b)^*aba$ 

NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

For $i, j \in \{1, \ldots, n\}$ and $k \in \{0, 1, \ldots, n\}$ we define languages $L_{i,j}^k$ and show by induction on k that they can be generated by regular expressions.

NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

For $i, j \in \{1, \ldots, n\}$ and $k \in \{0, 1, \ldots, n\}$ we define languages $L_{i,j}^k$ and show by induction on k that they can be generated by regular expressions.

$$L_{i,j}^k := \{x \in \Sigma^* \mid \text{on input } x \ M \text{ starts in } z_i \text{ and ends in } z_j \ (\hat{\delta}(z_i, x) = z_j)$$
$$\text{while the intermediate states (despite } z_i, z_j) \text{ have index} \leq k\}$$

NFA $\rightarrow$ Regular Expressions

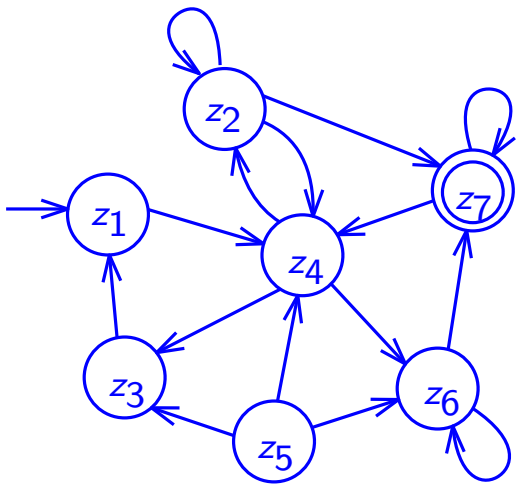
Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

For $i, j \in \{1, \ldots, n\}$ and $k \in \{0, 1, \ldots, n\}$ we define languages $L_{i,j}^k$ and show by induction on k that they can be generated by regular expressions.

$$L_{i,j}^k := \{x \in \Sigma^* \mid \text{on input } x \ M \text{ starts in } z_i \text{ and ends in } z_j \ (\hat{\delta}(z_i, x) = z_j)$$
$$\text{while the intermediate states (despite } z_i, z_j) \text{ have index} \leq k\}$$



NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

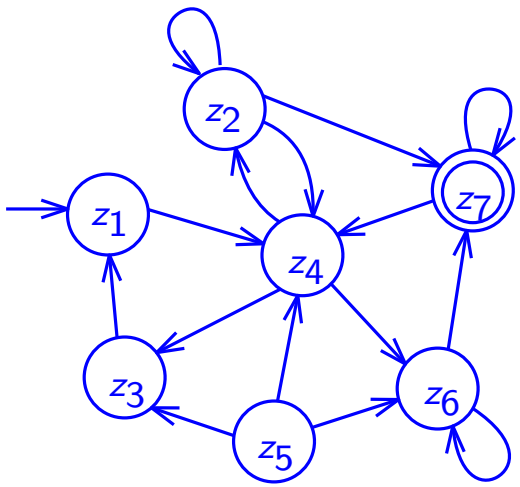
Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

For $i, j \in \{1, \ldots, n\}$ and $k \in \{0, 1, \ldots, n\}$ we define languages $L^k_{i,j}$ and show by induction on k that they can be generated by regular expressions.

$$L^k_{i,j} := \{x \in \Sigma^* \mid \text{on input } x \text{ } M \text{ starts in } z_i \text{ and ends in } z_j \ (\hat{\delta}(z_i, x) = z_j)$$
$$\text{while the intermediate states (despite } z_i, z_j) \text{ have index} \leq k\}$$

$L^5_{3,6}$?



NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

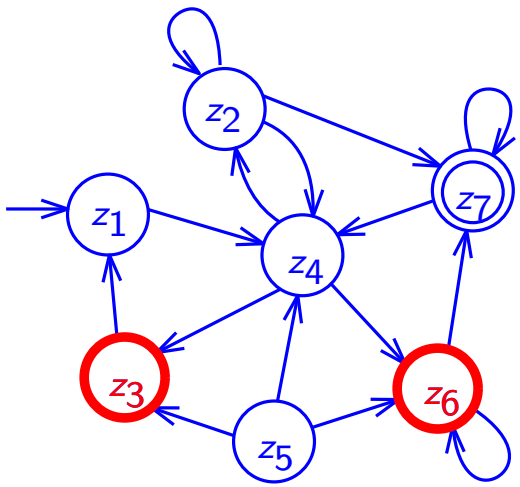
Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

For $i, j \in \{1, \ldots, n\}$ and $k \in \{0, 1, \ldots, n\}$ we define languages $L^k_{i,j}$ and show by induction on k that they can be generated by regular expressions.

$$L^k_{i,j} := \{x \in \Sigma^* \mid \text{on input } x \ M \text{ starts in } z_i \text{ and ends in } z_j \ (\hat{\delta}(z_i, x) = z_j)$$
$$\text{while the intermediate states (despite } z_i, z_j) \text{ have index} \leq k\}$$

$L^5_{3,6}$?



NFA $\rightarrow$ Regular Expressions

Theorem: Any language which can be recognized by an NFA can be described by a regular expression.

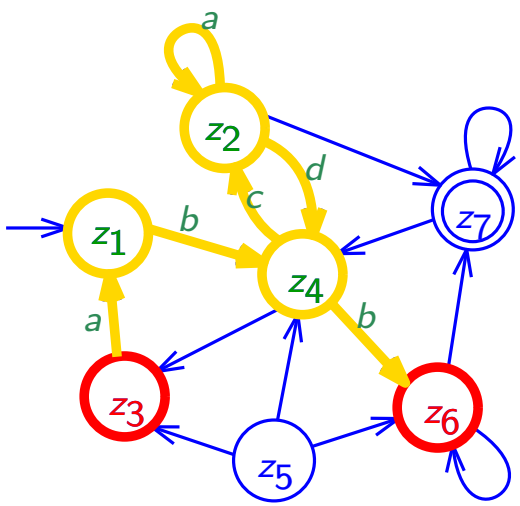
Proof

Let M be a DFA with $L = L(M)$ and let the states be $Z = \{z_1, \ldots, z_n\}$ where z_1 is the initial state.

For $i, j \in \{1, \ldots, n\}$ and $k \in \{0, 1, \ldots, n\}$ we define languages $L^k_{i,j}$ and show by induction on k that they can be generated by regular expressions.

$$L^k_{i,j} := \{x \in \Sigma^* \mid \text{on input } x \text{ } M \text{ starts in } z_i \text{ and ends in } z_j \ (\hat{\delta}(z_i, x) = z_j)$$
$$\text{while the intermediate states (despite } z_i, z_j) \text{ have index} \leq k\}$$

$L^5_{3,6}$?



NFA $\longrightarrow$ Regular Expressions

Proof (cont'd)

Base of induction: For $k = 0$ we have

$$i \neq j: \quad L^0_{i,j} = \{a \in \Sigma \mid \delta(z_i, a) = z_j\}$$

NFA $\longrightarrow$ Regular Expressions

Proof (cont'd)

Base of induction: For $k = 0$ we have

$$i \neq j: \quad L^0_{i,j} = \{a \in \Sigma \mid \delta(z_i, a) = z_j\}$$

$$L^0_{i,i} = \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(z_i, a) = z_i\}$$

NFA $\longrightarrow$ Regular Expressions

Proof (cont'd)

Base of induction: For $k = 0$ we have

$$i \neq j: \quad L^0_{i,j} = \{a \in \Sigma \mid \delta(z_i, a) = z_j\}$$

$$L^0_{i,i} = \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(z_i, a) = z_i\}$$

Thus $L^0_{i,j}$ is finite and can be generated by a regular expression (i.e. just a sequence of $+$ expressions).

NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

Base of induction: For $k = 0$ we have

$$i \neq j: \quad L^0_{i,j} = \{a \in \Sigma \mid \delta(z_i, a) = z_j\}$$

$$L^0_{i,i} = \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(z_i, a) = z_i\}$$

Thus $L^0_{i,j}$ is finite and can be generated by a regular expression (i.e. just a sequence of $+$ expressions).

Induction step: Let $k \geq 1$ and $L^k_{i,j}$ be expressible by a regular expression ($L^k_{i,j} = L(R^k_{i,j})$).

NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

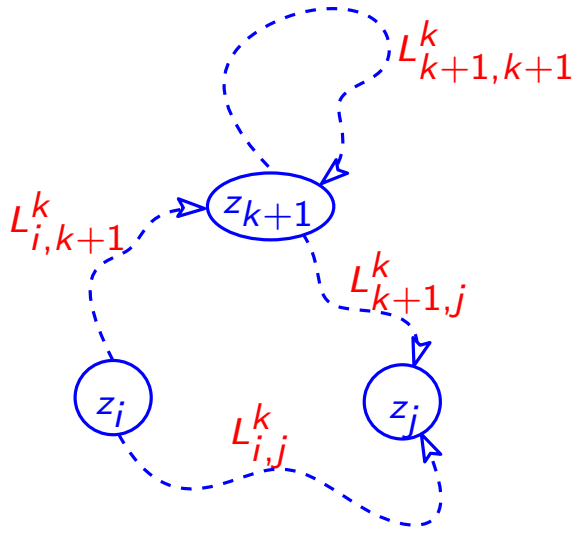
To use the induction step, we need to express $L_{i,j}^{k+1}$ in terms of those with smaller upper indices.

NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

To use the induction step, we need to express $L_{i,j}^{k+1}$ in terms of those with smaller upper indices.

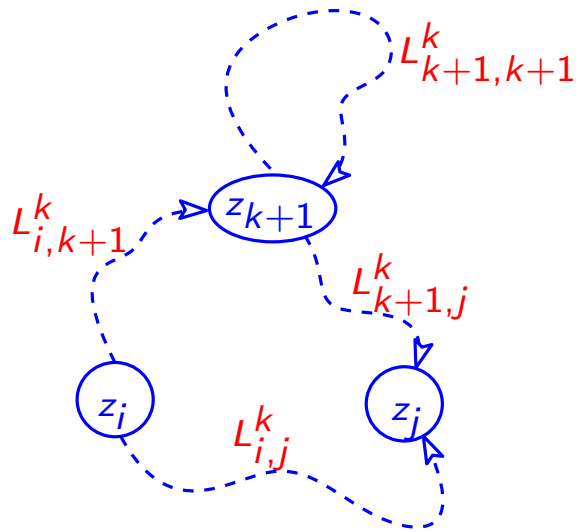
$$L_{i,j}^{k+1} = L_{i,j}^{k} \cup L_{i,k+1}^{k} (L_{k+1,k+1}^{k})^* L_{k+1,j}^{k}$$



NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

To use the induction step, we need to express $L_{i,j}^{k+1}$ in terms of those with smaller upper indices.



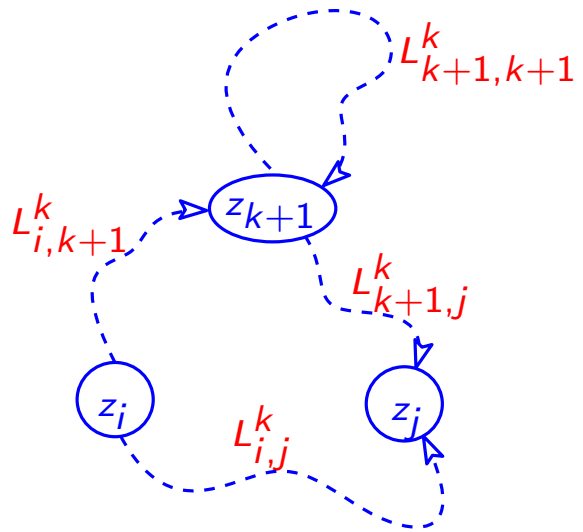
$$L_{i,j}^{k+1} = L_{i,j}^{k} \cup L_{i,k+1}^{k} (L_{k+1,k+1}^{k})^* L_{k+1,j}^{k}$$

Explanation: If z_{k+1} is not needed to travel from z_i to z_j we have $L_{i,j}^k$. Otherwise we run trough z_{k+1} one or multiple times, but we start and stop at z_{k+1} for the substring in the middle.

NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

To use the induction step, we need to express $L_{i,j}^{k+1}$ in terms of those with smaller upper indices.



$$L_{i,j}^{k+1} = L_{i,j}^{k} \cup L_{i,k+1}^{k}(L_{k+1,k+1}^{k})^* L_{k+1,j}^{k}$$

Explanation: If z_{k+1} is not needed to travel from z_i to z_j we have $L_{i,j}^k$. Otherwise we run trough z_{k+1} one or multiple times, but we start and stop at z_{k+1} for the substring in the middle.

By the induction hypothesis, $L_{i,j}^k$, $L_{i,k+1}^k$, $L_{k+1,k+1}^k$ and $L_{k+1,j}^k$ can be generated by the regular expression and thus $L_{i,j}^{k+1}$ can be generated by

$$R_{i,j}^{k+1} = R_{i,j}^{k} + R_{i,k+1}^{k}(R_{k+1,k+1}^{k})^* R_{k+1,j}^{k}.$$

NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

Finally we have

$$L(M) = \bigcup_{z_i \in E} L^n_{1,i}$$

NFA $\rightarrow$ Regular Expressions

Proof (cont'd)

Finally we have

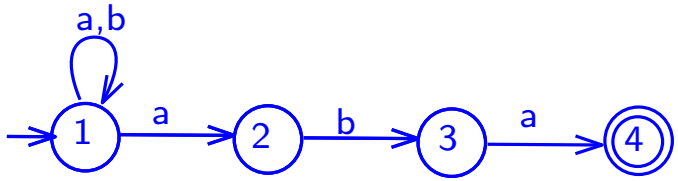
$$L(M) = \bigcup_{z_i \in E} L^n_{1,i}$$

If $i_1, i_2, \ldots, i_m$ are the indices of the final states we have

$$R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$$

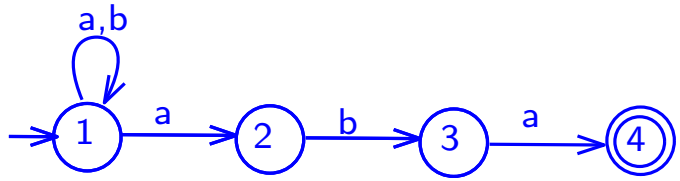
with $L(R) = L(M)$, which completes the proof.

NFA $\longrightarrow$ Regular Expressions



M

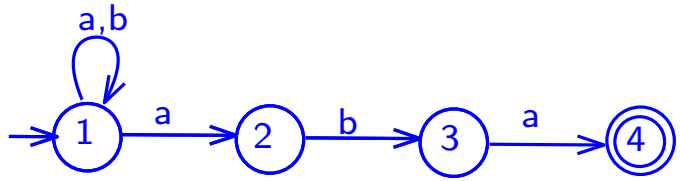
NFA $\rightarrow$ Regular Expressions



M

We need to find $R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$ with $i_1, i_2, \ldots, i_m$ being the final states.

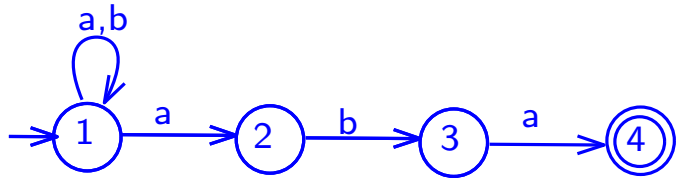
NFA $\rightarrow$ Regular Expressions



M

We need to find $R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$ with $i_1, i_2, \ldots, i_m$ being the final states. Here $R = R^3_{1,4}$.

NFA $\rightarrow$ Regular Expressions

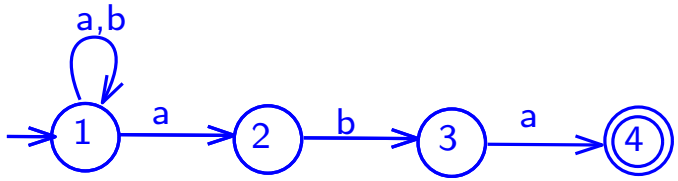


M

We need to find $R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$ with $i_1, i_2, \ldots, i_m$ being the final states. Here $R = R^3_{1,4}$.

Remeber: $R^{k+1}_{i,j} = R^k_{i,j} + R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$.

NFA $\rightarrow$ Regular Expressions



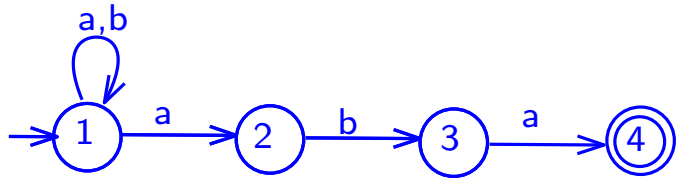
M

We need to find $R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$ with $i_1, i_2, \ldots, i_m$ being the final states. Here $R = R^3_{1,4}$.

Remeber: $R^{k+1}_{i,j} = R^k_{i,j} + R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$.

$R^0_{1,1} = (\varepsilon + a + b)$

NFA $\rightarrow$ Regular Expressions



M

We need to find $R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$ with $i_1, i_2, \ldots, i_m$ being the final states. Here $R = R^3_{1,4}$.

Remeber: $R^{k+1}_{i,j} = R^k_{i,j} + R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$.

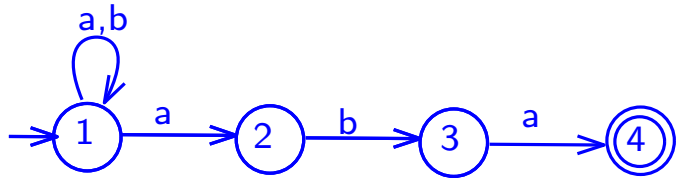
$R^0_{1,1} = (\varepsilon + a + b)$

$R^1_{1,1} = R^0_{1,1} + R^0_{1,1}(R^0_{1,1})^* R^0_{1,1} = (a+b)^*$

$R^1_{1,2} = (a+b)^* a$

$R^2_{1,3} = (a+b)^* ab$

NFA $\rightarrow$ Regular Expressions



M

We need to find $R = (R^n_{1,i_1} + R^n_{1,i_2} + \cdots + R^n_{1,i_m})$ with $i_1, i_2, \ldots, i_m$ being the final states. Here $R = R^3_{1,4}$.

Remeber: $R^{k+1}_{i,j} = R^k_{i,j} + R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$.

$R^0_{1,1} = (\varepsilon + a + b)$

$R^1_{1,1} = R^0_{1,1} + R^0_{1,1}(R^0_{1,1})^* R^0_{1,1} = (a+b)^*$

$R^1_{1,2} = (a+b)^* a$

$R^2_{1,3} = (a+b)^* ab$

$R^3_{1,4} = (a+b)^* aba$

Summary

The following are equally expressive in the sense that they all produce/recognize regular languages:

- Regular Grammars
- NFAs (Nondeterministic Finite Automata)
- DFAs (Deterministic Finite Automata)
- Regular Expressions