Automata Theory and Formal Languages

Summer Term 2025

8th Lecture

ε-FA

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

Regular Languages

Summary of Previous Lecture

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

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

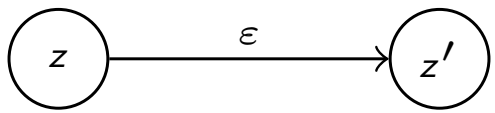
ε-FA

ε-finite automata

An ε-FA is an NFA allowing ε-transitions, that is transitions in which no symbol is read.

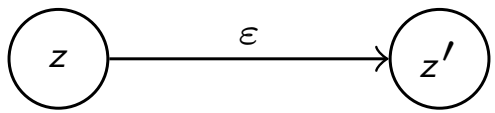
ε-finite automata

An ε-FA is an NFA allowing ε-transitions, that is transitions in which no symbol is read.



ε-finite automata

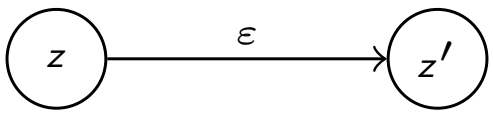
An ε-FA is an NFA allowing ε-transitions, that is transitions in which no symbol is read.



When in state z, the machine has two options: remaining in z, or moving to z' without reading any letter from the input.

ε-finite automata

An ε-FA is an NFA allowing ε-transitions, that is transitions in which no symbol is read.

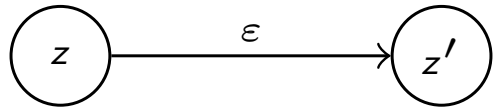


When in state z, the machine has two options: remaining in z, or moving to z' without reading any letter from the input.

Motivation: ε-transitions simplify automaton construction.

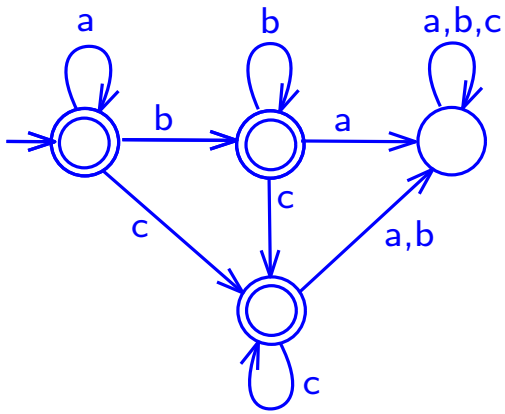
ε-finite automata

An ε-FA is an NFA allowing ε-transitions, that is transitions in which no symbol is read.



When in state z, the machine has two options: remaining in z, or moving to z' without reading any letter from the input.

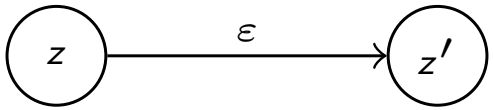
Motivation: ε-transitions simplify automaton construction.



A DFA recognizing a*b*c*

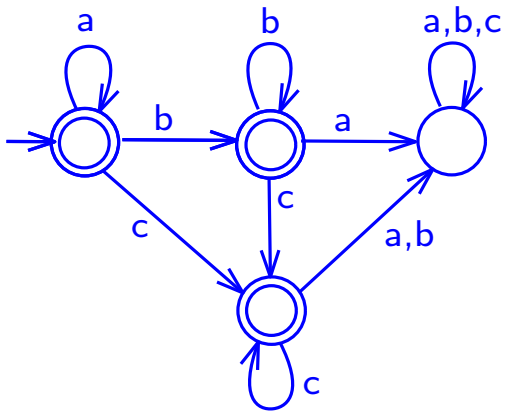
ε-finite automata

An ε-FA is an NFA allowing ε-transitions, that is transitions in which no symbol is read.

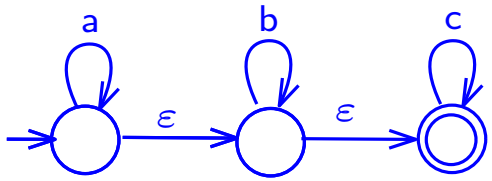


When in state z, the machine has two options: remaining in z, or moving to z' without reading any letter from the input.

Motivation: ε-transitions simplify automaton construction.



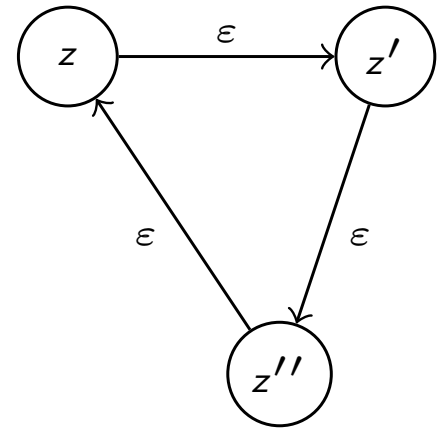
A DFA recognizing a*b*c*



An ε-FA recognizing a*b*c*

Formal definition of ε-finite automata

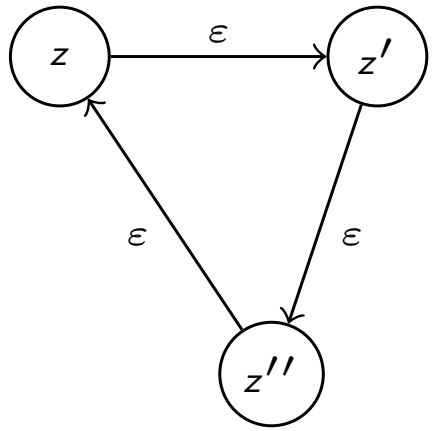
ε-cycle



Formal definition of ε-finite automata

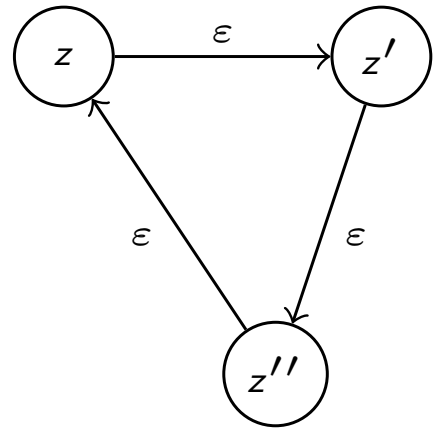
ε-cycle

Such an ε-cycle can be replaced by a single state



Formal definition of ε-finite automata

ε-cycle



Such an ε-cycle can be replaced by a single state

Definition

An ε-FA (or an NFA with ε-transitions) M is a 5 tuple

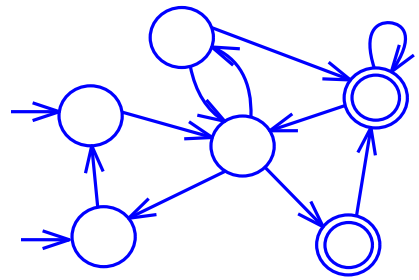
$$M = (Z, \Sigma, \delta, S, E)$$

where

- δ: $Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ Transition function that has no ε-cycles
- Z, Σ, S, E are as NFAs

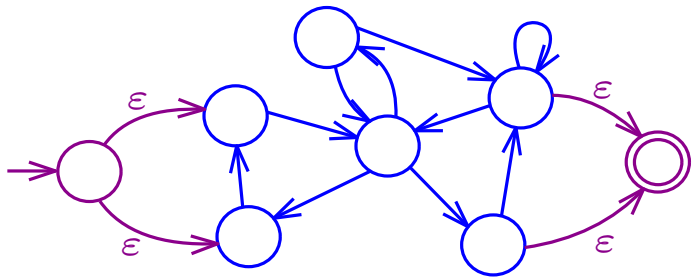
Regular operations with ε-FA

- Multiple starting/final states
  $\rightarrow$ single starting/final state



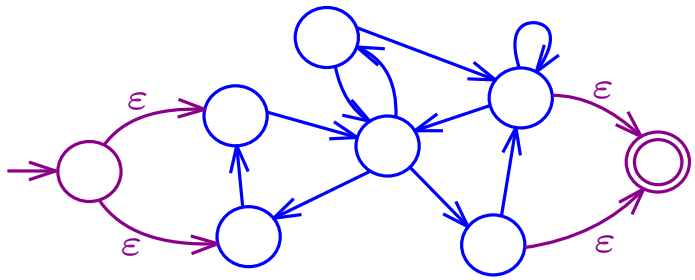
Regular operations with ε-FA

- Multiple starting/final states
  $\rightarrow$ single starting/final state



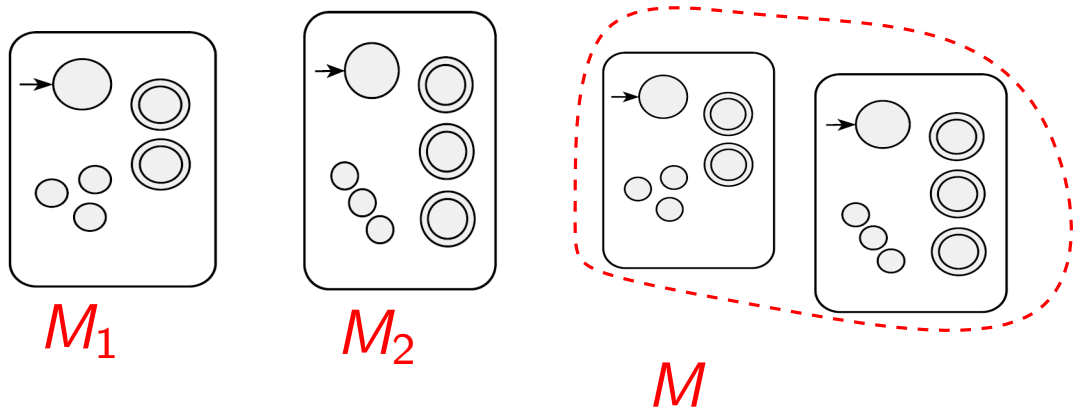
Regular operations with ε-FA

- Multiple starting/final states
  $\rightarrow$ single starting/final state



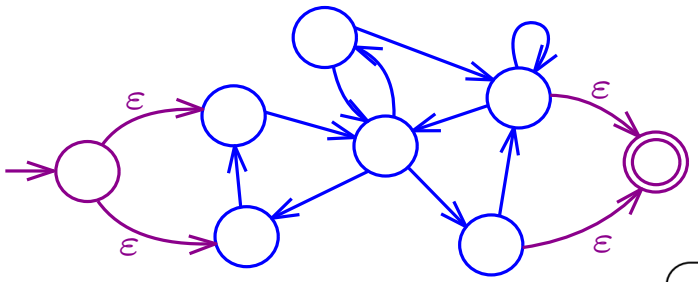
- Union of Regular Languges

$$L(M) = L(M_1) \cup L(M_2)$$



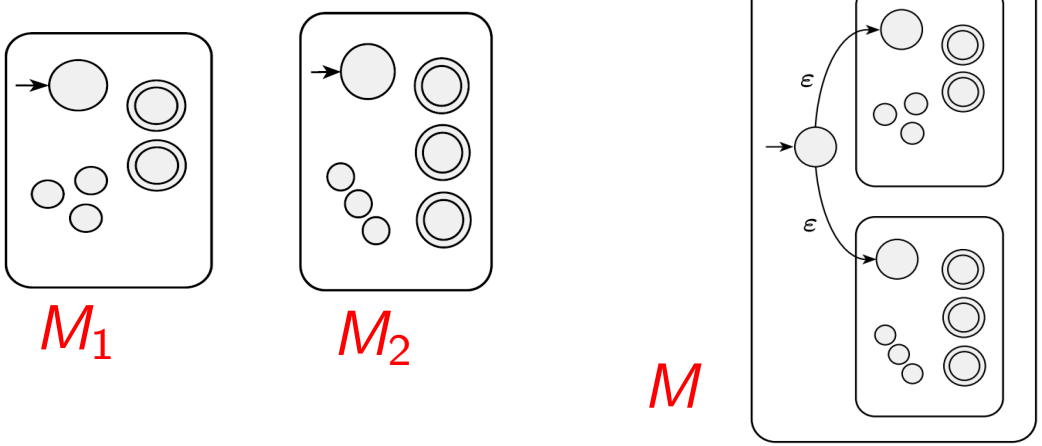
Regular operations with ε-FA

- Multiple starting/final states
  $\rightarrow$ single starting/final state



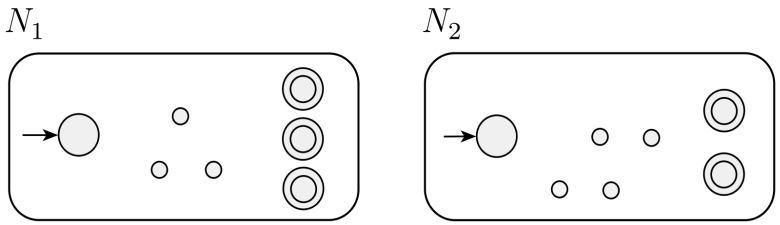
- Union of Regular Languges

$$L(M) = L(M_1) \cup L(M_2)$$



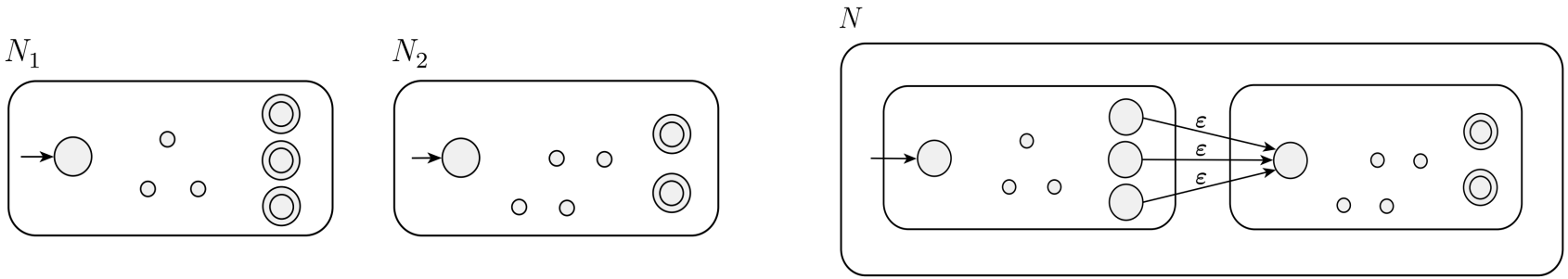
Regular operations with ε-FA

- Concatenation of Regular Languges



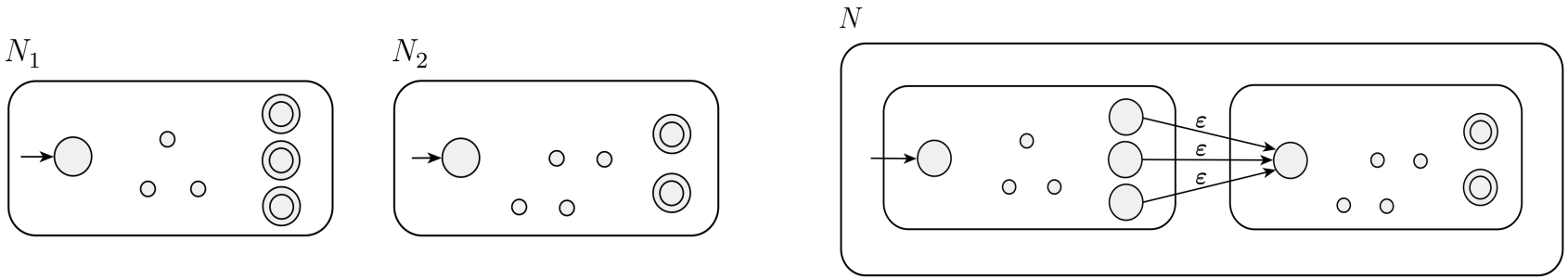
Regular operations with ε-FA

- Concatenation of Regular Languges



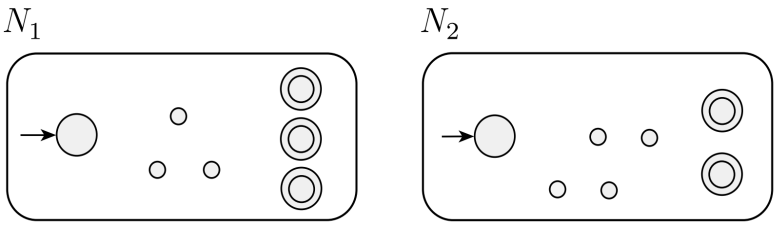
Regular operations with ε-FA

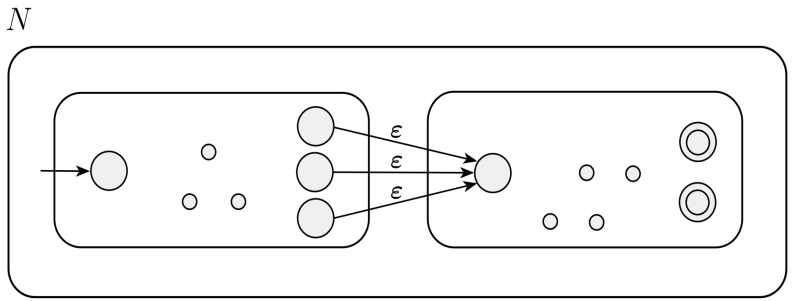
- Concatenation of Regular Languges $L(N) = L(N_1)L(N_2)$



Regular operations with ε-FA

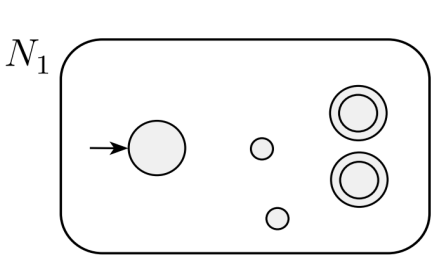
- Concatenation of Regular Languges $L(N) = L(N_1)L(N_2)$

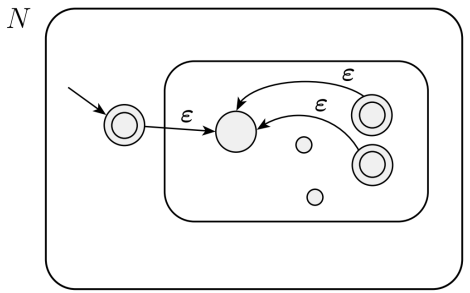




- Kleene Star of Regular Languges

$L(N) = L(N_1)^*$



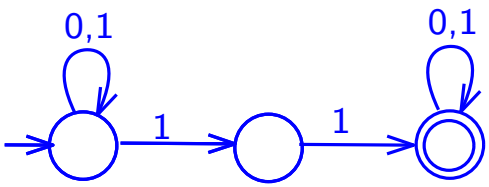


Regular Expressions and ε-FA

Construct an NFA and an ε-FA recognizing the language of binary string containg either 101 or 11 as substring

Regular Expressions and ε-FA

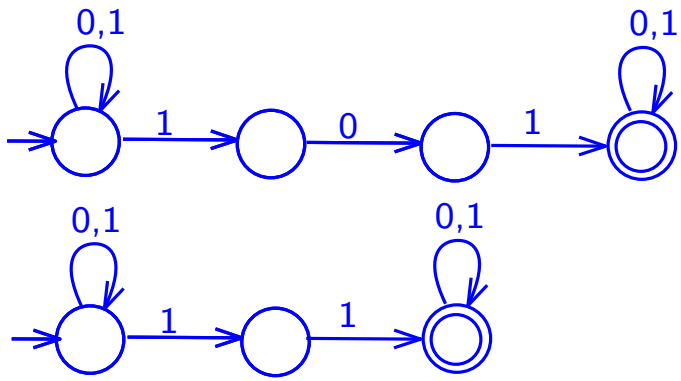
Construct an NFA and an ε-FA recognizing the language of binary string containg either 101 or 11 as substring



Regular Expressions and ε-FA

Construct an NFA and an ε-FA recognizing the language of binary string containg either 101 or 11 as substring

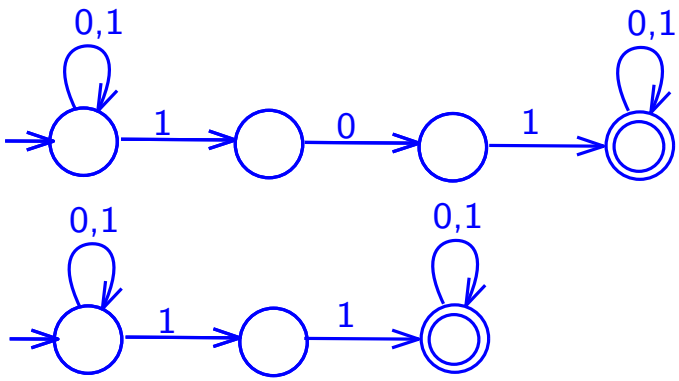
NFA



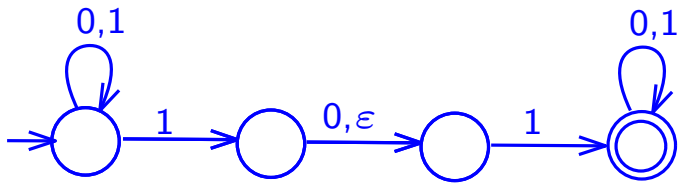
Regular Expressions and ε-FA

Construct an NFA and an ε-FA recognizing the language of binary string containg either 101 or 11 as substring

NFA



ε-FA



Regular Expressions and ε-FA

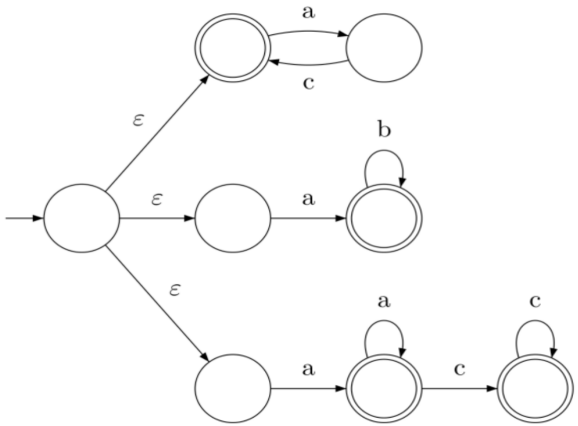
Construct an ε-FA for the regular expression

$$(ab^* + aa^*c^* + (ac)^*)(a + b)^*baba \ .$$

Regular Expressions and ε-FA

Construct an ε-FA for the regular expression

$$(ab^* + aa^*c^* + (ac)^*)(a + b)^*baba \ .$$

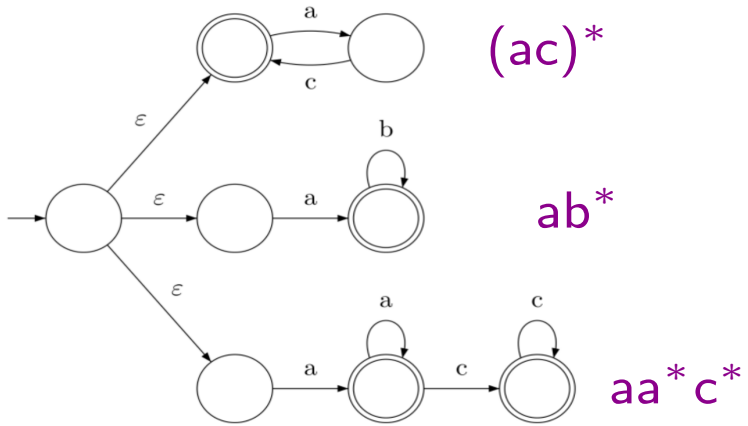


ab*+aa*c*+(ac)*

Regular Expressions and ε-FA

Construct an ε-FA for the regular expression

$$(ab^* + aa^*c^* + (ac)^*)(a+b)^*baba \ .$$

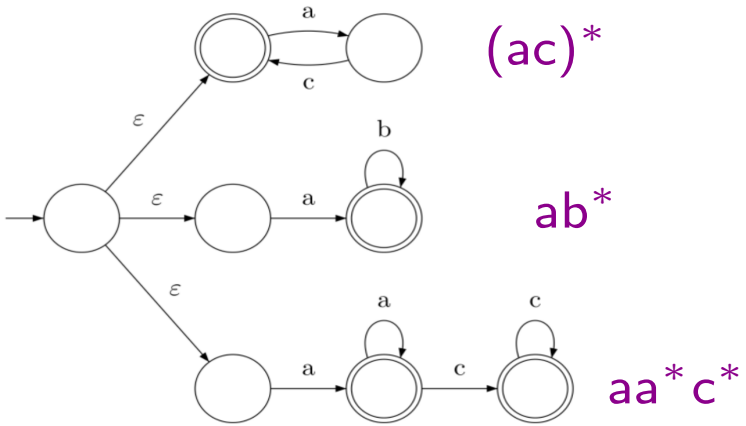


ab*+aa*c*+(ac)*

Regular Expressions and ε-FA

Construct an ε-FA for the regular expression

$$(ab^* + aa^*c^* + (ac)^*)(a+b)^*baba \ .$$

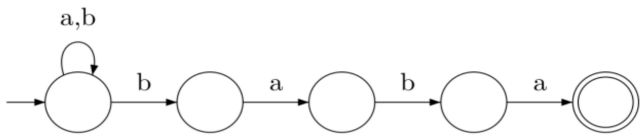


(ac)*

ab*

aa*c*

ab*+aa*c*+(ac)*

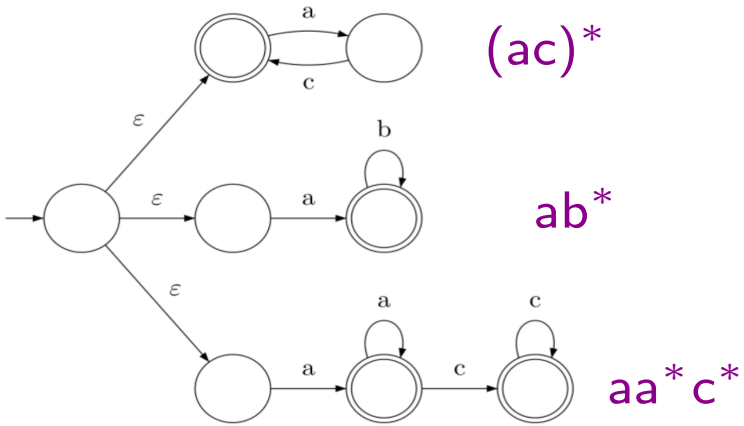


(a+b)*baba

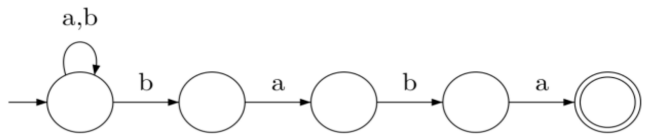
Regular Expressions and ε-FA

Construct an ε-FA for the regular expression

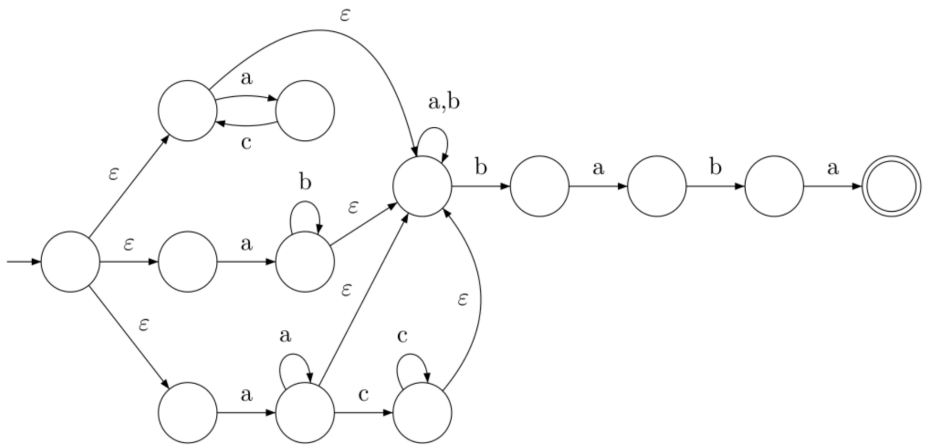
$$(ab^* + aa^*c^* + (ac)^*)(a + b)^*baba \ .$$



ab*+aa*c*+(ac)*



(a+b)*baba



(ab*+aa*c*+(ac)*)(a+b)*baba

Regular Expressions and ε-FA

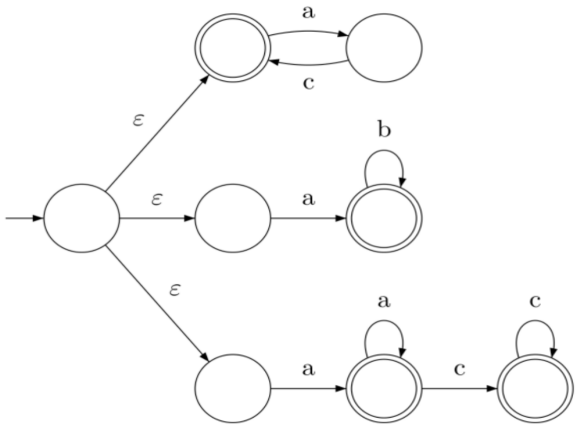
Construct an ε-FA for the regular expression

$$(ab^*+aa^*c^*+(ac)^*)^*$$

Regular Expressions and ε-FA

Construct an ε-FA for the regular expression

$$(ab^*+aa^*c^*+(ac)^*)^*$$

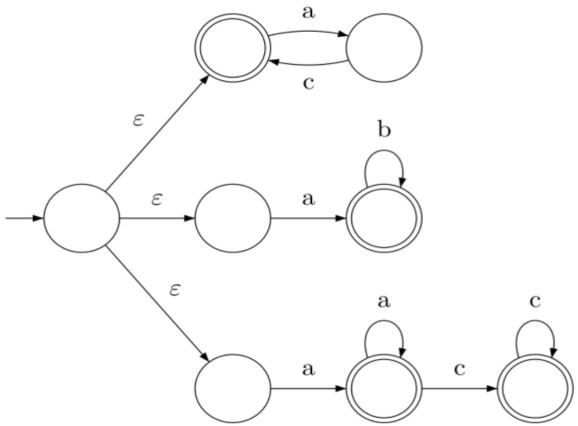


$ab^*+aa^*c^*+(ac)^*$

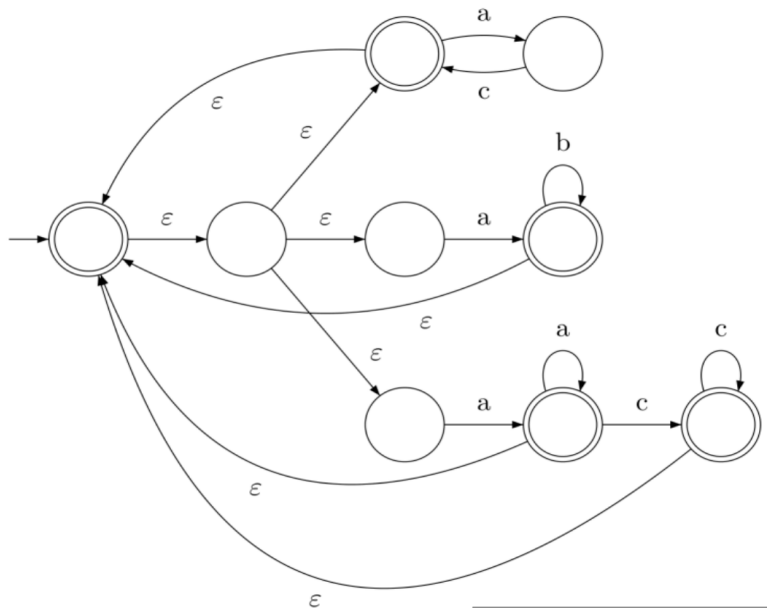
Regular Expressions and ε-FA

Construct an ε-FA for the regular expression

$$(ab^*+aa^*c^*+(ac)^*)^*$$



$ab^*+aa^*c^*+(ac)^*$



$(ab^*+aa^*c^*+(ac)^*)^*$

Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Obviously, ε-closure can be extended to any set of states.

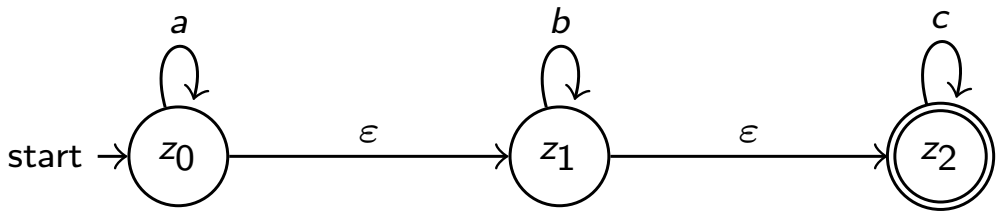
Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Obviously, ε-closure can be extended to any set of states.

Example



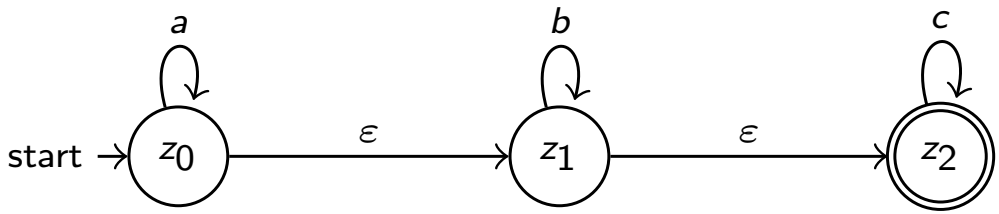
Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Obviously, ε-closure can be extended to any set of states.

Example



$$\varepsilon\text{-closure}(z_0) = \{z_0, z_1, z_2\}$$

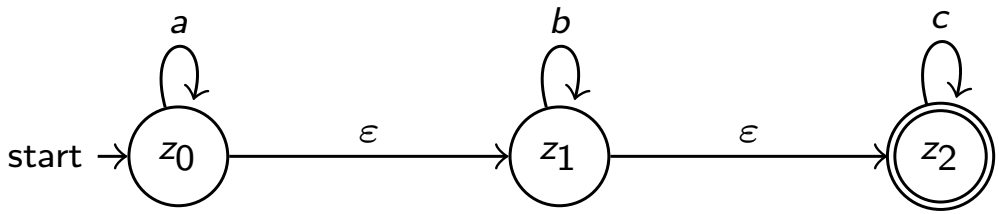
Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Obviously, ε-closure can be extended to any set of states.

Example



$$\varepsilon\text{-closure}(z_0) = \{z_0, z_1, z_2\}$$

every state always has an ε-transition to itself

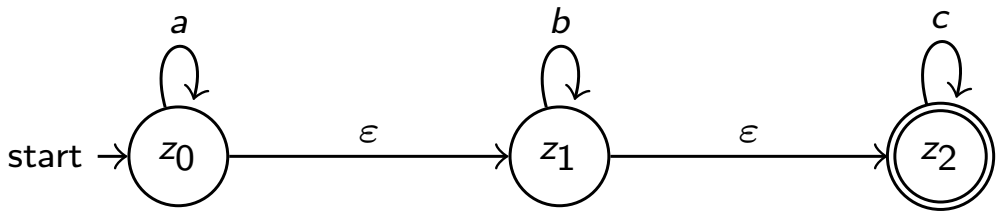
Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Obviously, ε-closure can be extended to any set of states.

Example



$$\varepsilon\text{-closure}(z_0) = \{z_0, z_1, z_2\}$$

every state always has an ε-transition to itself

$$\varepsilon\text{-closure}(z_1) = \{z_1, z_2\}$$

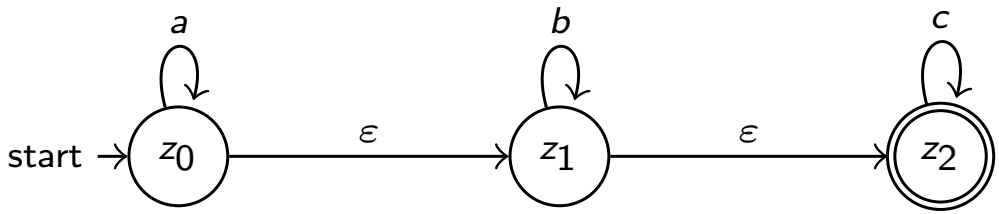
Converting ε-FA to NFA

ε-closure

Let z be a state of an ε-FA. Then the ε-closure of z is defined as the set of states reachable from z through ε-transitions.

Obviously, ε-closure can be extended to any set of states.

Example



$$\varepsilon\text{-closure}(z_0) = \{z_0, z_1, z_2\}$$

every state always has an ε-transition to itself

$$\varepsilon\text{-closure}(z_1) = \{z_1, z_2\}$$

$$\varepsilon\text{-closure}(z_2) = \{z_2\}$$

Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$

Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

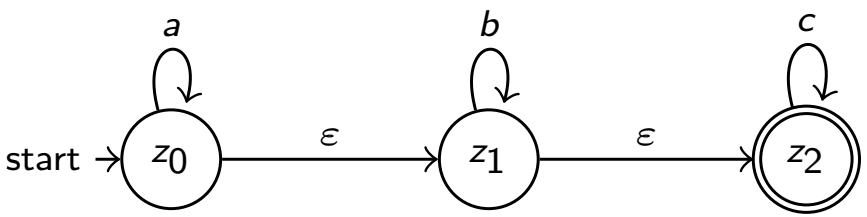
Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

Example



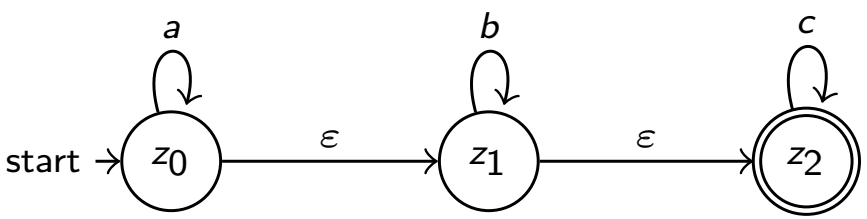
Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

Example



$$\delta'(z_0, \varepsilon) = \{z_0, z_1, z_2\}$$
$$\delta'(z_1, \varepsilon) = \{z_1, z_2\}$$
$$\delta'(z_2, \varepsilon) = \{z_2\}$$

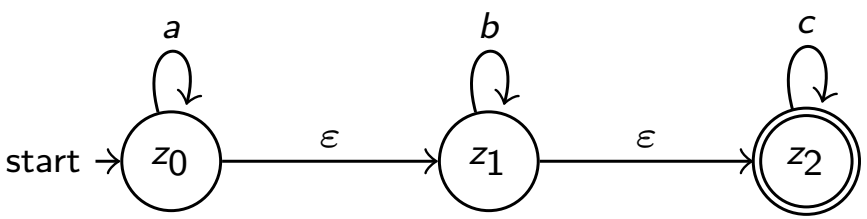
Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

Example



$$\delta'(z_0, \varepsilon) = \{z_0, z_1, z_2\}$$
$$\delta'(z_1, \varepsilon) = \{z_1, z_2\}$$
$$\delta'(z_2, \varepsilon) = \{z_2\}$$

$\delta'(z_0, a) = \{z_0, z_1, z_2\}$

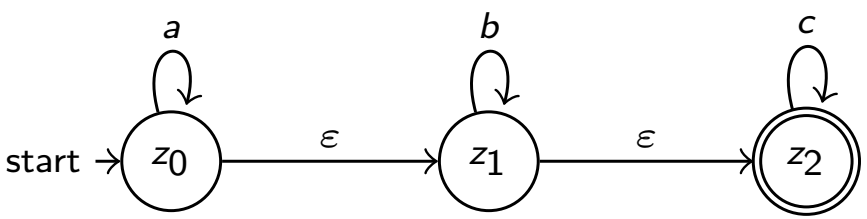
Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

Example



$$\delta'(z_0, \varepsilon) = \{z_0, z_1, z_2\}$$
$$\delta'(z_1, \varepsilon) = \{z_1, z_2\}$$
$$\delta'(z_2, \varepsilon) = \{z_2\}$$

$$\delta'(z_0, a) = \{z_0, z_1, z_2\}$$
$$\delta'(z_0, b) = \{z_1, z_2\}$$
$$\delta'(z_0, c) = \{z_2\}$$

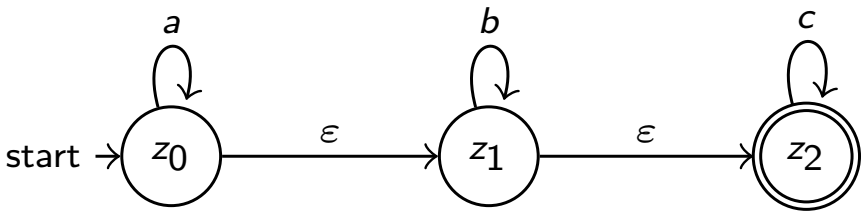
Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

Example



$$\delta'(z_0, \varepsilon) = \{z_0, z_1, z_2\}$$
$$\delta'(z_1, \varepsilon) = \{z_1, z_2\}$$
$$\delta'(z_2, \varepsilon) = \{z_2\}$$

$$\delta'(z_0, a) = \{z_0, z_1, z_2\} \qquad \delta'(z_1, a) = \emptyset$$
$$\delta'(z_0, b) = \{z_1, z_2\} \qquad \delta'(z_1, b) = \{z_1, z_2\}$$
$$\delta'(z_0, c) = \{z_2\} \qquad \delta'(z_1, c) = \{z_2\}$$

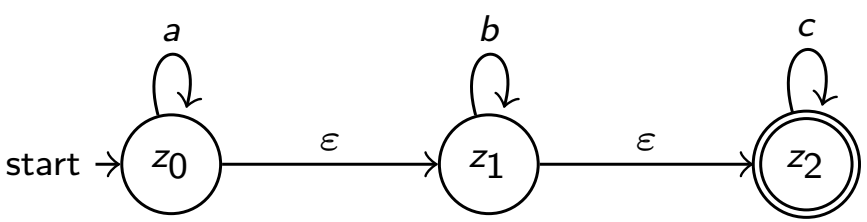
Converting ε-FA to NFA

Definition

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA. Define the function $\delta' : Z \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Z)$ by:

- $\delta'(z, \varepsilon) := \varepsilon\text{-closure}(z)$
- if $\varepsilon\text{-closure}(z) = \{q_1, \ldots, q_r\}$, then
$$\delta'(z, a) := \varepsilon\text{-closure}\big(\delta(q_1, a)\big) \cup \cdots \cup \varepsilon\text{-closure}\big(\delta(q_r, a)\big).$$

Example



$$\delta'(z_0, \varepsilon) = \{z_0, z_1, z_2\}$$
$$\delta'(z_1, \varepsilon) = \{z_1, z_2\}$$
$$\delta'(z_2, \varepsilon) = \{z_2\}$$

$$\delta'(z_0, a) = \{z_0, z_1, z_2\} \qquad \delta'(z_1, a) = \emptyset \qquad \delta'(z_2, a) = \emptyset$$
$$\delta'(z_0, b) = \{z_1, z_2\} \qquad \delta'(z_1, b) = \{z_1, z_2\} \qquad \delta'(z_2, b) = \emptyset$$
$$\delta'(z_0, c) = \{z_2\} \qquad \delta'(z_1, c) = \{z_2\} \qquad \delta'(z_2, c) = \{z_2\}$$

Converting ε-FA to NFA

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA and

$$E' = \{z \in Z \mid \varepsilon\text{-closure}(z_1) \cap E \neq \emptyset\}.$$

Converting ε-FA to NFA

Let $M = (Z, \Sigma, \delta, S, E)$ be an ε-FA and

$$E' = \{z \in Z \mid \varepsilon\text{-closure}(z_1) \cap E \neq \emptyset\}.$$

Then $M' = (Z, \Sigma, \delta', S, E')$ is an equivalent NFA with M.

Theorem

Each ε-FA can be converted into an equivalent NFA.

Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε-transitions yet

Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε-transitions yet

2. All states that can reach a final state over ε-transitions become final states

Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε-transitions yet

2. All states that can reach a final state over ε-transitions become final states

3. Remove all ε-transitions

Converting ε-FA to NFA

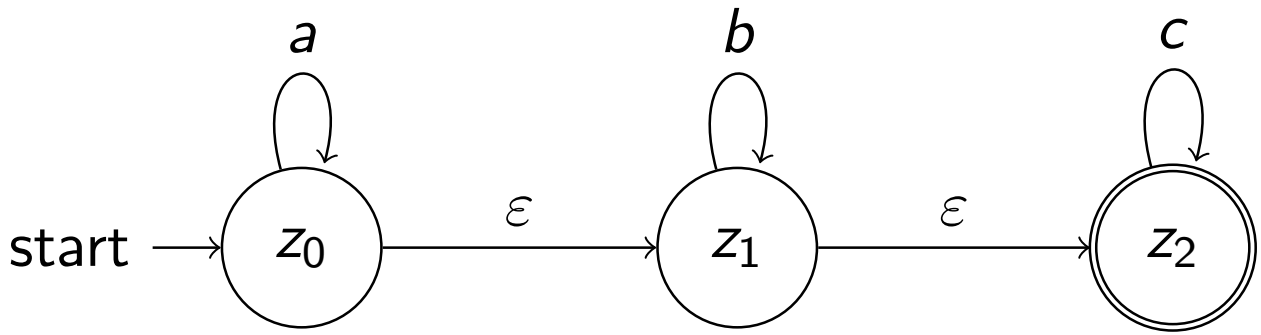
Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε-transitions yet

2. All states that can reach a final state over ε-transitions become final states

3. Remove all ε-transitions

Example



Converting ε-FA to NFA

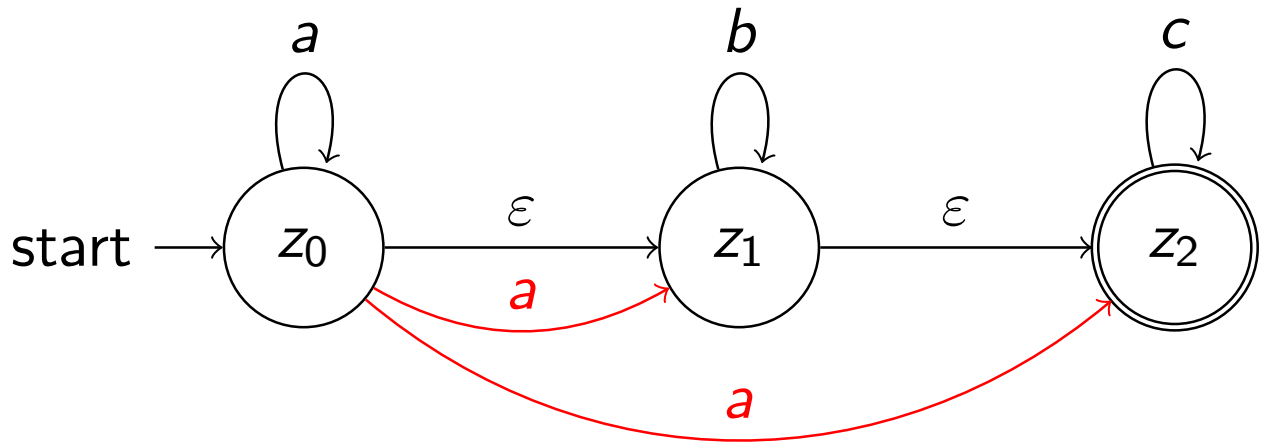
Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε transitions yet.

2. All states that can reach a final state over ε transitions become final states

3. Remove all ε transitions

Example

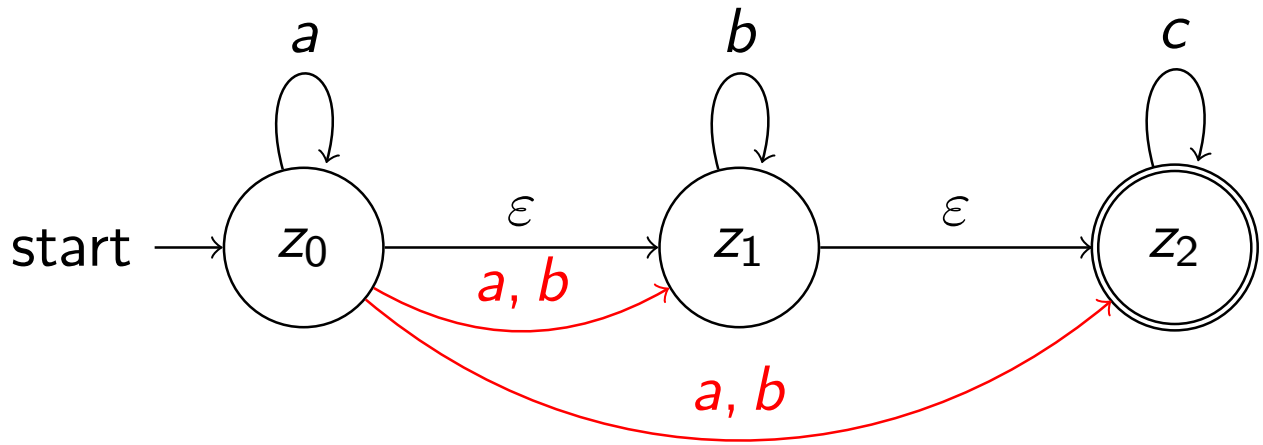


Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε transitions yet.
2. All states that can reach a final state over ε transitions become final states
3. Remove all ε transitions

Example

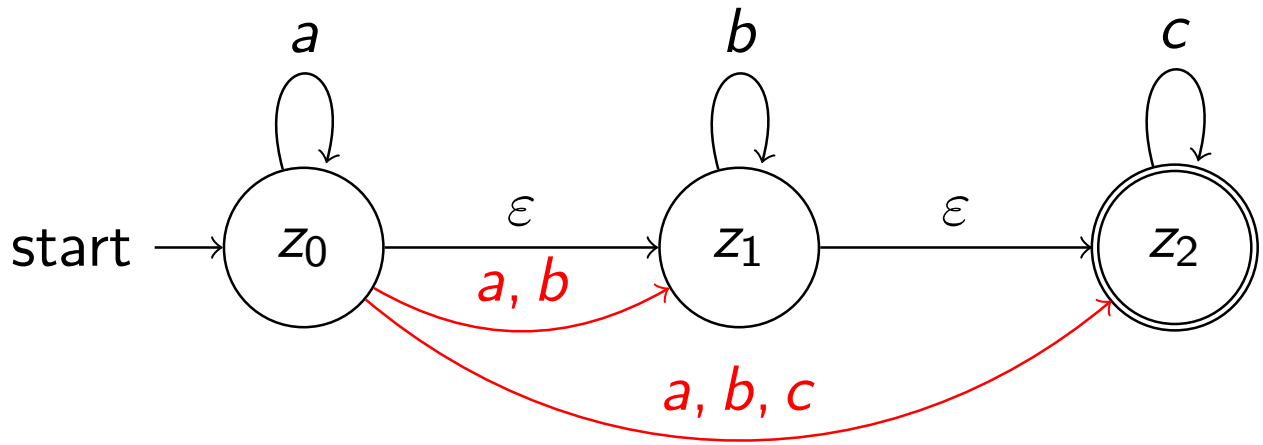


Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε transitions yet.
2. All states that can reach a final state over ε transitions become final states
3. Remove all ε transitions

Example

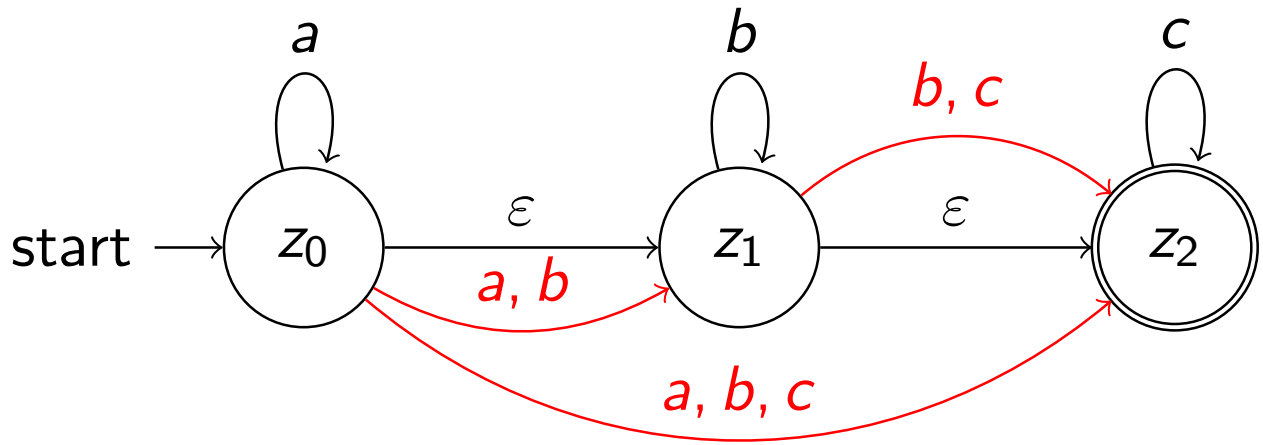


Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε transitions yet.
2. All states that can reach a final state over ε transitions become final states
3. Remove all ε transitions

Example

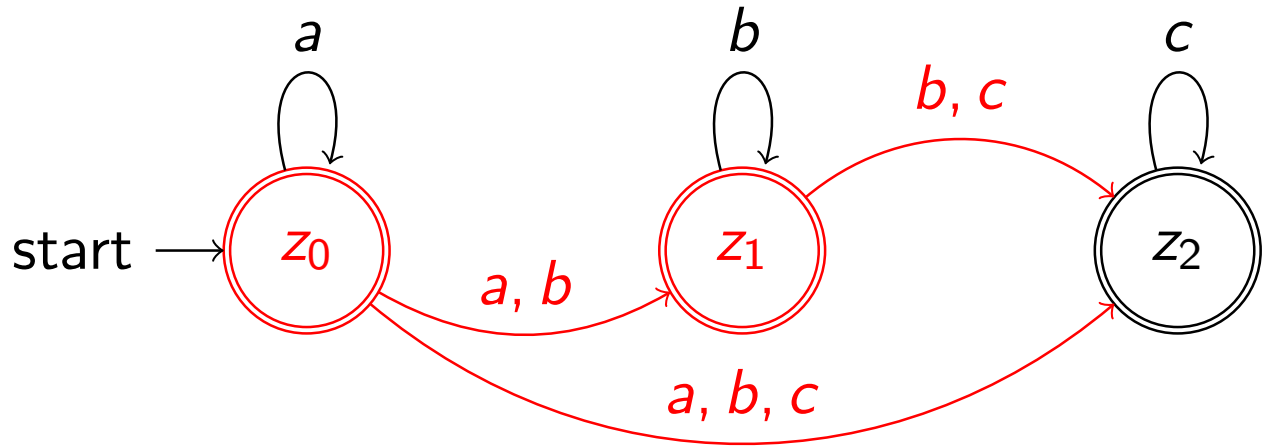


Converting ε-FA to NFA

Algorithm for converting ε-FA into NFA

1. Add arrows representing δ'-transitions. Do not remove ε transitions yet.
2. All states that can reach a final state over ε transitions become final states
3. Remove all ε transitions

Example



Regular Languages

Summary

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

- Regular Expressions
- Regular Grammars
- DFAs
- NFAs
- ε-FAs

State Minimization

Equivalence of Automata

Suppose two descriptions of a regular language are given. How can we answer the question of whether the two descriptions actually define the same language or not?

Equivalence of Automata

Suppose two descriptions of a regular language are given. How can we answer the question of whether the two descriptions actually define the same language or not?

Here we discuss how to test whether two descriptors for regular languages are equivalent in the sense that they define the same language.

Equivalence of Automata

Suppose two descriptions of a regular language are given. How can we answer the question of whether the two descriptions actually define the same language or not?

Here we discuss how to test whether two descriptors for regular languages are equivalent in the sense that they define the same language.

An important consequence of this test is that there is a way to minimize a DFA, i.e., we can take any DFA and find an equivalent DFA that has minimum number of states.

Equivalence of Automata

Suppose two descriptions of a regular language are given. How can we answer the question of whether the two descriptions actually define the same language or not?

Here we discuss how to test whether two descriptors for regular languages are equivalent in the sense that they define the same language.

An important consequence of this test is that there is a way to minimize a DFA, i.e., we can take any DFA and find an equivalent DFA that has minimum number of states.

The minimal automaton is unique (despite trivial modifications such as renaming states).

Testing Equivalence of States

Equivalent States

Given a DFA, we say that states p and q are equivalent if for all input strings w:

$$\hat{\delta}(p, w) \text{ is an accepting state} \iff \hat{\delta}(q, w) \text{ is an accepting state}$$

Testing Equivalence of States

Equivalent States

Given a DFA, we say that states p and q are equivalent if for all input strings w:

$$\hat{\delta}(p, w) \text{ is an accepting state} \iff \hat{\delta}(q, w) \text{ is an accepting state}$$

This defines an equivalence relation on the states of M, which partitions them into the so-called Nerode classes.

Testing Equivalence of States

Equivalent States

Given a DFA, we say that states p and q are equivalent if for all input strings w:

$$\hat{\delta}(p, w) \text{ is an accepting state} \iff \hat{\delta}(q, w) \text{ is an accepting state}$$

This defines an equivalence relation on the states of M, which partitions them into the so-called Nerode classes.

Remark

It's not required that $\hat{\delta}(p, w)$ and $\hat{\delta}(q, w)$ to be the same state; it suffices for either both to be accepting or both to be non-accepting.

Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not

Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

Testing Equivalence of States

Observation

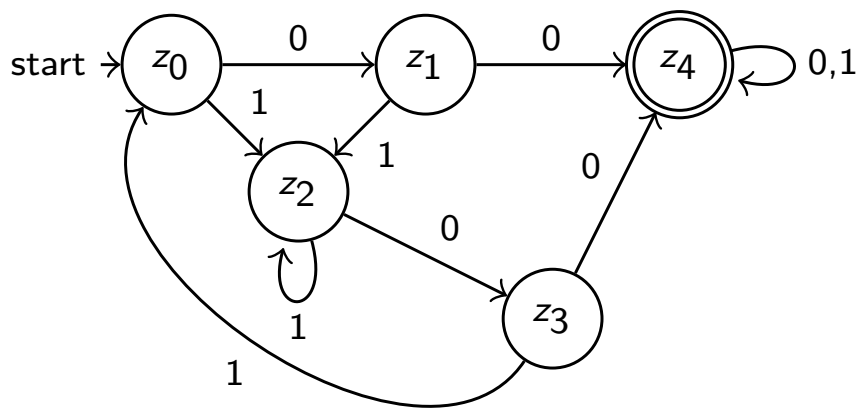
- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

Example



Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

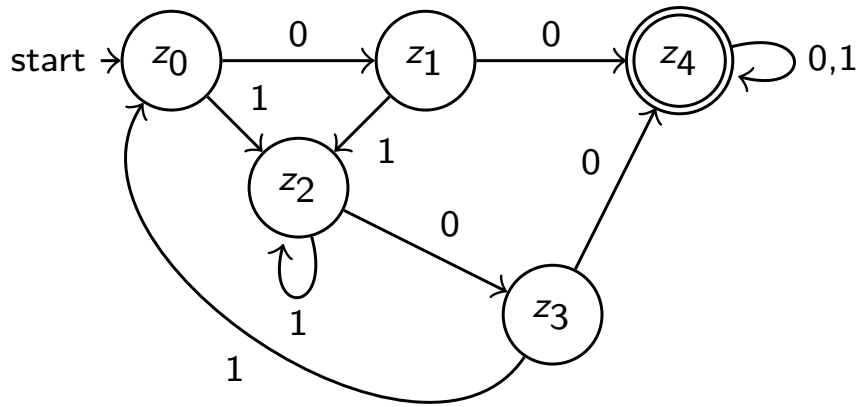
  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

Example

- z_2 and z_4 are obviously not equivalent: one is accepting and the other is not



Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

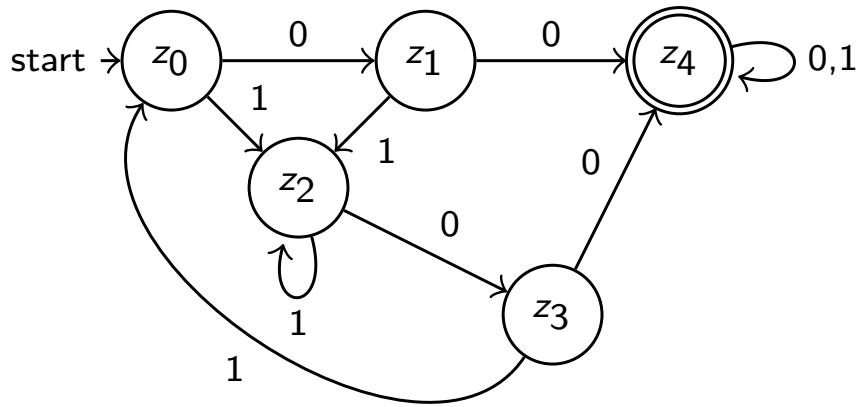
  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

Example

- z_2 and z_4 are obviously not equivalent: one is accepting and the other is not
- z_1 and z_3 are equivalent:



Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

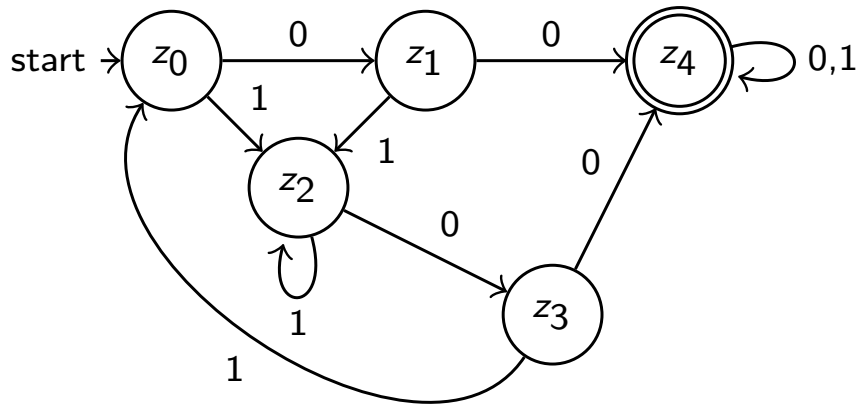
- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

Example

- z_2 and z_4 are obviously not equivalent: one is accepting and the other is not
- z_1 and z_3 are equivalent:

  $\hat{\delta}(z_1, 0) = \hat{\delta}(z_3, 0) = z_4$



Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

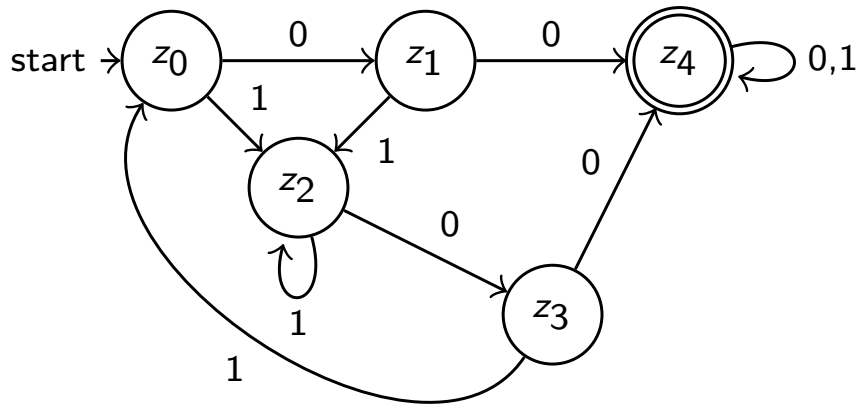
Example

- z_2 and z_4 are obviously not equivalent: one is accepting and the other is not
- z_1 and z_3 are equivalent:

$\hat{\delta}(z_1, 0) = \hat{\delta}(z_3, 0) = z_4$

$\hat{\delta}(z_1, 11) = \hat{\delta}(z_3, 11) = z_2$

$\hat{\delta}(z_1, 10) = z_3, \ \hat{\delta}(z_3, 10) = z_1$



Testing Equivalence of States

Observation

- If p is an accepting state and q not, then they are not equivalent.

  ($\hat{\delta}(p, \varepsilon)$ is an accepting state, but $\hat{\delta}(q, \varepsilon)$ is not)

- If $\delta(p', a) = p$, $\delta(q', a) = q$ for some letter a, and p, q are not equivalent, then p', q' are also not equivalent.

  ($\exists w$, $\hat{\delta}(p, w)$ is an accepting state but $\hat{\delta}(q, w)$ is not
  $\Rightarrow$ $\hat{\delta}(p', aw)$ is an accepting state but $\hat{\delta}(q', aw)$ is not)

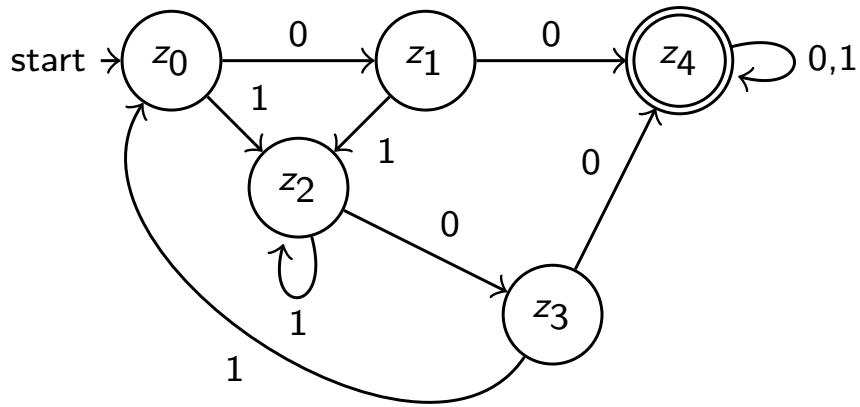
Example

- z_2 and z_4 are obviously not equivalent: one is accepting and the other is not
- z_1 and z_3 are equivalent:

$\hat{\delta}(z_1, 0) = \hat{\delta}(z_3, 0) = z_4$

$\hat{\delta}(z_1, 11) = \hat{\delta}(z_3, 11) = z_2$

$\hat{\delta}(z_1, 10) = z_3, \ \hat{\delta}(z_3, 10) = z_1$



From these it follows that for every string w, $\hat{\delta}(z_1, w) = z_4 \Leftrightarrow \hat{\delta}(z_3, w) = z_4$. Why?

State Minimization Algorithm

Input: a DFA M

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.

3. Tag all pairs $\{z, z'\}$ where $z \in E$ and $z' \notin E$.

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.

3. Tag all pairs $\{z, z'\}$ where $z \in E$ and $z' \notin E$.

4. For each untagged pairs $\{z, z'\}$ and each $a \in \Sigma$ check if

$$\{\delta(z, a), \delta(z', a)\}$$

is already tagged. If yes, tag $\{z, z'\}$.

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.

3. Tag all pairs $\{z, z'\}$ where $z \in E$ and $z' \notin E$.

4. For each untagged pairs $\{z, z'\}$ and each $a \in \Sigma$ check if

$$\{\delta(z, a), \delta(z', a)\}$$

is already tagged. If yes, tag $\{z, z'\}$. Repeat this until nothing changes anymore.

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.

3. Tag all pairs $\{z, z'\}$ where $z \in E$ and $z' \notin E$.

4. For each untagged pairs $\{z, z'\}$ and each $a \in \Sigma$ check if

$$\{\delta(z, a), \delta(z', a)\}$$

is already tagged. If yes, tag $\{z, z'\}$. Repeat this until nothing changes anymore.

5. Merge all untagged pairs of states.

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.
2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.
3. Tag all pairs $\{z, z'\}$ where $z \in E$ and $z' \notin E$.
4. For each untagged pairs $\{z, z'\}$ and each $a \in \Sigma$ check if

$$\{\delta(z, a), \delta(z', a)\}$$

is already tagged. If yes, tag $\{z, z'\}$. Repeat this until nothing changes anymore.

5. Merge all untagged pairs of states. $\leftarrow$ equiavalent states

State Minimization Algorithm

Input: a DFA M

1. Remove all states that cannot be reached from the initial state.

2. Create an empty table of all state pairs $\{z, z'\}$ where $z, z' \in Z$ and $z \neq z'$.

3. Tag all pairs $\{z, z'\}$ where $z \in E$ and $z' \notin E$.

4. For each untagged pairs $\{z, z'\}$ and each $a \in \Sigma$ check if

$$\{\delta(z, a), \delta(z', a)\}$$

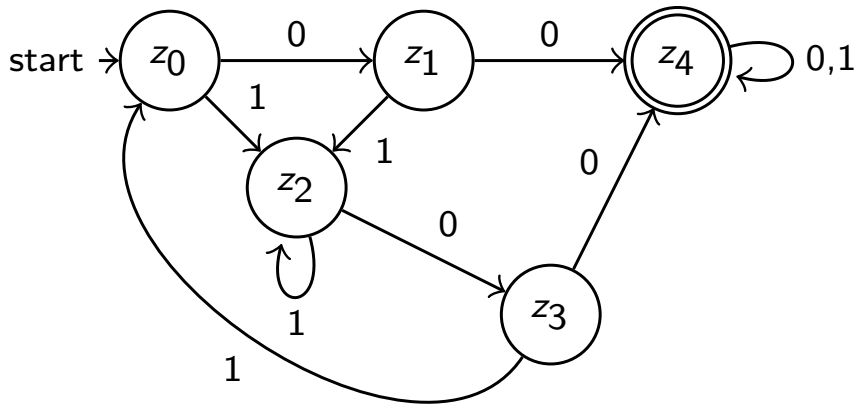
is already tagged. If yes, tag $\{z, z'\}$. Repeat this until nothing changes anymore.

5. Merge all untagged pairs of states. $\leftarrow$ equiavalent states

Output: the minimal automaton M'

State Minimization Algorithm

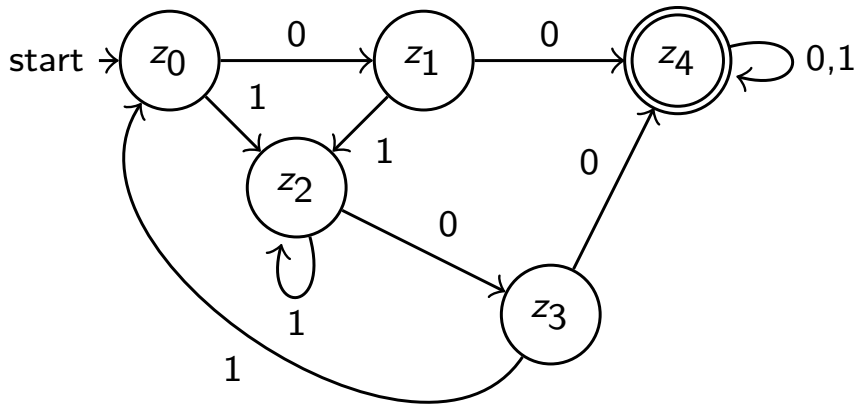
Example



State Minimization Algorithm

Example

Step 1: skip



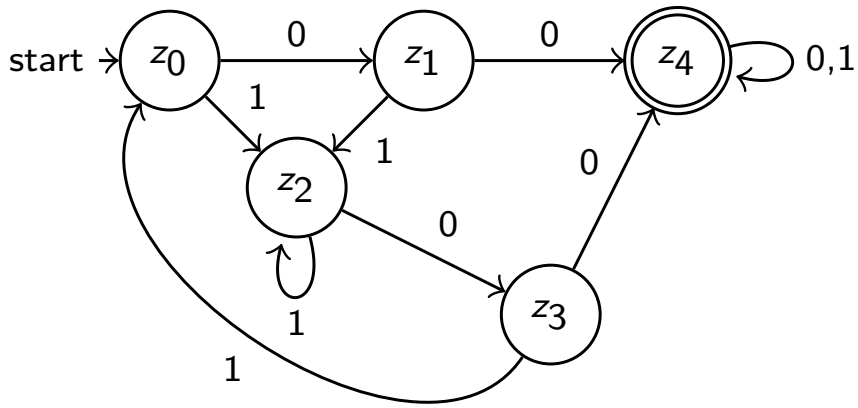
State Minimization Algorithm

Example

Step 1: skip

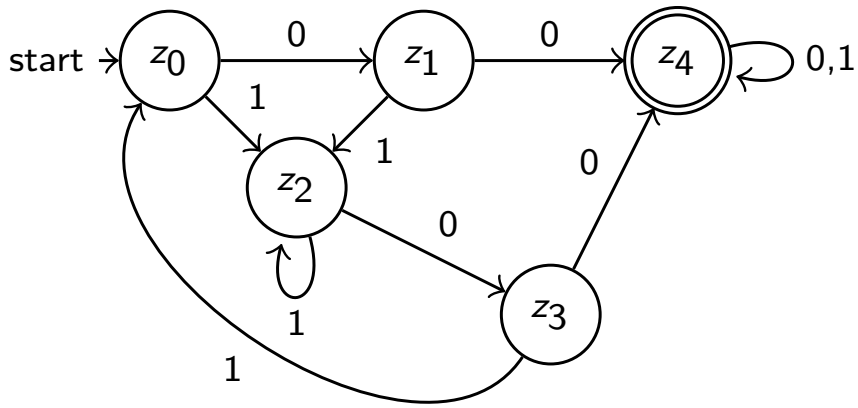
Step 2:

	z_0	z_1	z_2	z_3
z_4				
z_3				
z_2				
z_1				



State Minimization Algorithm

Example



Step 1: skip

Step 2:

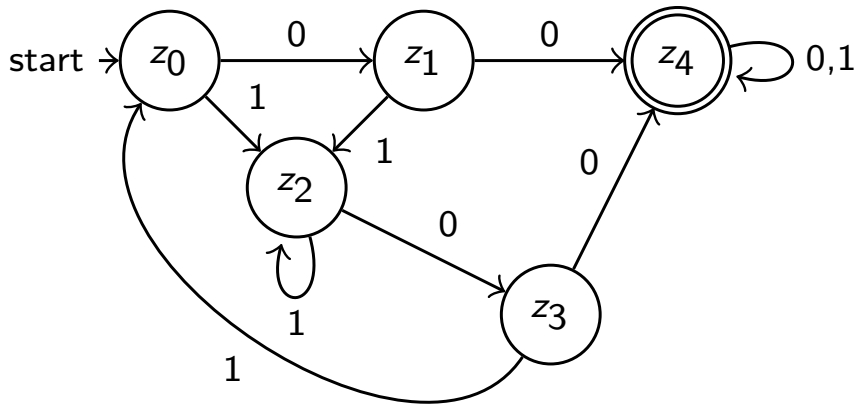
	z_0	z_1	z_2	z_3
z_4				
z_3				
z_2				
z_1				

Step 3:

	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3				
z_2				
z_1				

State Minimization Algorithm

Example



Step 1: skip

Step 2:

	z_0	z_1	z_2	z_3
z_4				
z_3				
z_2				
z_1				

Step 3:

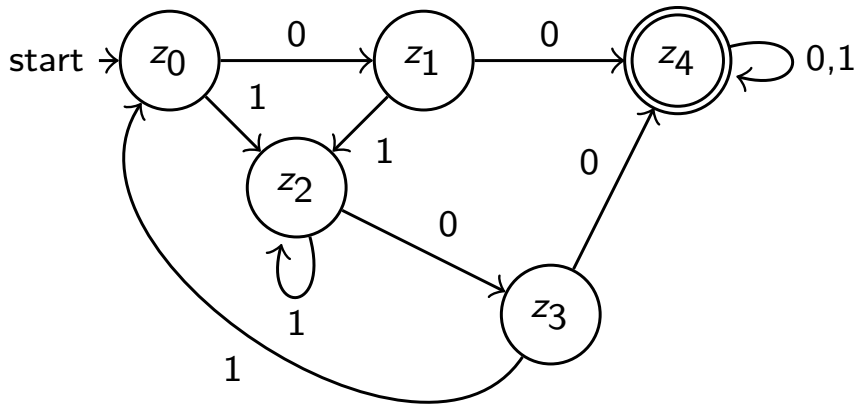
	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3				
z_2				
z_1				

Step 4-5:

	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3	X		X	
z_2		X		
z_1	X			

State Minimization Algorithm

Example



Step 1: skip

Step 2:

	z_0	z_1	z_2	z_3
z_4				
z_3				
z_2				
z_1				

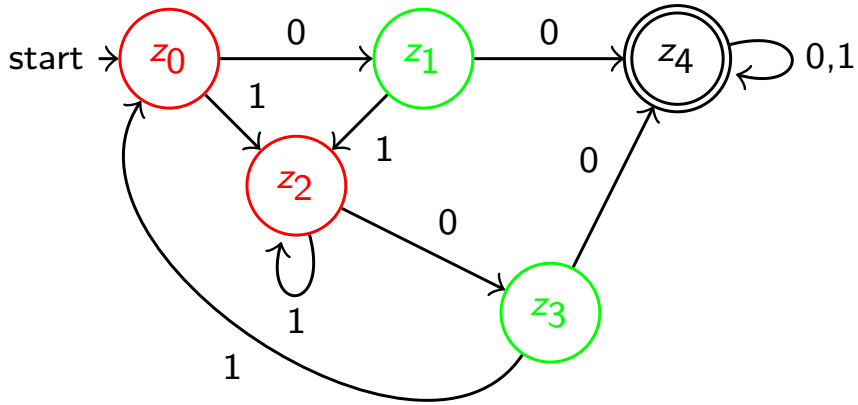
Step 3:

	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3				
z_2				
z_1				

Step 4-5:

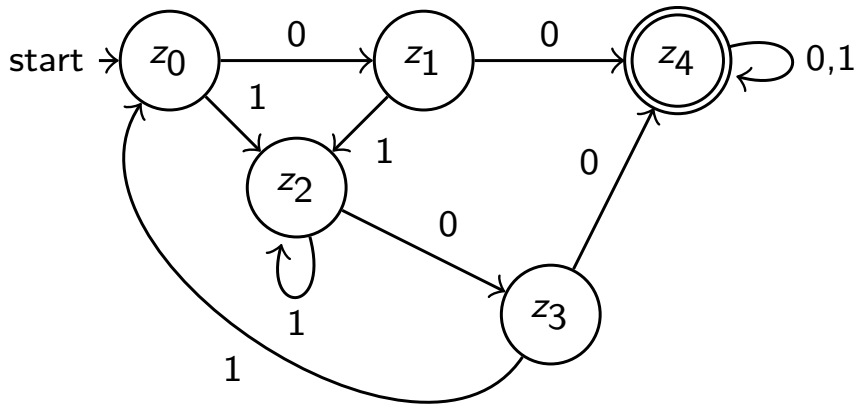
	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3	X		X	
z_2		X		
z_1	X			

Step 6: The pairs z_0, z_2 and z_1, z_3 are equivalent and can be merged.



State Minimization Algorithm

Example



Step 1: skip

Step 2:

	z_0	z_1	z_2	z_3
z_4				
z_3				
z_2				
z_1				

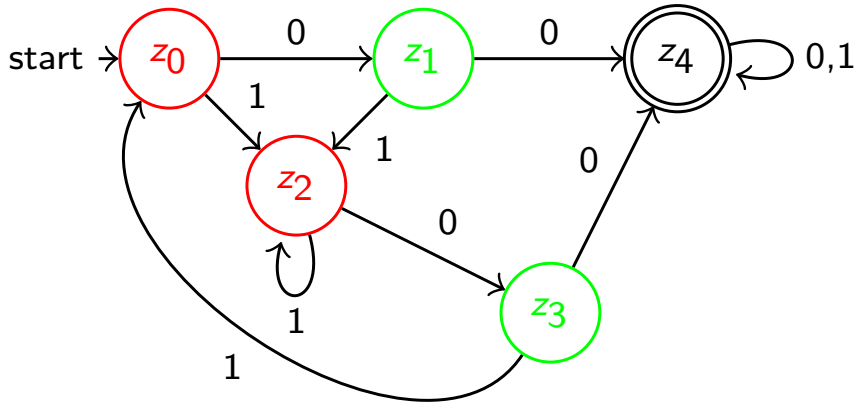
Step 3:

	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3				
z_2				
z_1				

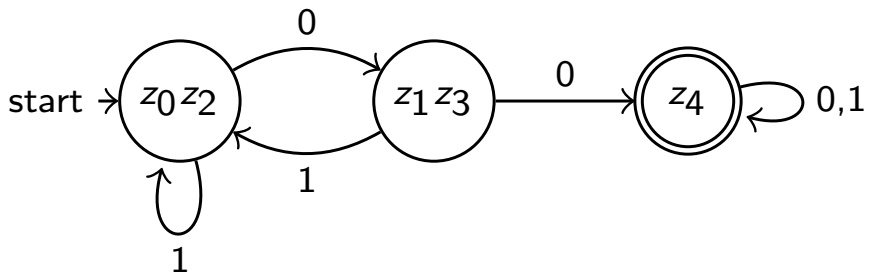
Step 4-5:

	z_0	z_1	z_2	z_3
z_4	X	X	X	X
z_3	X		X	
z_2		X		
z_1	X			

Step 6: The pairs z_0, z_2 and z_1, z_3 are equivalent and can be merged.



Output:



Closure Properties

Theorem

The regular languages are closed under the following operations

- union $(O \cup K)$
- intersection $(O \cap K)$
- complement $(\overline{O})$
- product (OK)
- star (O^*)

Closure Properties

Theorem

The regular languages are closed under the following operations

- union $(O \cup K)$
- intersection $(O \cap K)$
- complement $(\overline{O})$
- product (OK)
- star (O^*)

Proof

The closure under union, product, and star has already been proved. We will now prove it for complement and intersection.

Closure Properties

Complement:

Let $M = (Z, \Sigma, \delta, z_0, E)$ be an automaton with $O = L(M)$ and a total transition function.

Closure Properties

Complement:

Let $M = (Z, \Sigma, \delta, z_0, E)$ be an automaton with $O = L(M)$ and a total transition function.

Then $M' = (Z, \Sigma, \delta, z_0, Z \setminus E)$ accepts the complement of M ($L(M') = \overline{O}$). One thus just has to change the accepting and non-accepting state.

Closure Properties

Complement:

Let $M = (Z, \Sigma, \delta, z_0, E)$ be an automaton with $O = L(M)$ and a total transition function.

Then $M' = (Z, \Sigma, \delta, z_0, Z \setminus E)$ accepts the complement of M ($L(M') = \overline{O}$). One thus just has to change the accepting and non-accepting state.

Intersection:

Due to the rules of de Morgan we have

$$O \cap K = \overline{\overline{O} \cup \overline{K}}.$$

Closure Properties

Complement:

Let $M = (Z, \Sigma, \delta, z_0, E)$ be an automaton with $O = L(M)$ and a total transition function.

Then $M' = (Z, \Sigma, \delta, z_0, Z \setminus E)$ accepts the complement of M ($L(M') = \overline{O}$). One thus just has to change the accepting and non-accepting state.

Intersection:

Due to the rules of de Morgan we have

$$O \cap K = \overline{\overline{O} \cup \overline{K}}.$$

Since the regular languages are closed under union and complement, they are thus also closed under intersection.

Intersection - alternative method

Product Construction

Let $M_1 = (Z, \Sigma, \delta_1, z_0, E)$ and $M_2 = (Q, \Sigma, \delta_2, q_0, F)$ be DFAs recognizing L_1 and L_2, respectively.

Intersection - alternative method

Product Construction

Let $M_1 = (Z, \Sigma, \delta_1, z_0, E)$ and $M_2 = (Q, \Sigma, \delta_2, q_0, F)$ be DFAs recognizing L_1 and L_2, respectively.

Consider

$$M = (Z \times Q, \Sigma, \delta, (z_0, q_0), E \times F)$$

with

$$\delta((z, q), a) = (\delta_1(z, a), \delta_2(q, a)).$$

Intersection - alternative method

Product Construction

Let $M_1 = (Z, \Sigma, \delta_1, z_0, E)$ and $M_2 = (Q, \Sigma, \delta_2, q_0, F)$ be DFAs recognizing L_1 and L_2, respectively.

Consider

$$M = (Z \times Q, \Sigma, \delta, (z_0, q_0), E \times F)$$

with

$$\delta((z, q), a) = (\delta_1(z, a), \delta_2(q, a)).$$

Then M recognizes $L_1 \cap L_2$ (prove this as an exercise).

Product Construction

Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Product Construction

Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Observe that $L = L_0 \cap L_1$ with

$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$

Product Construction

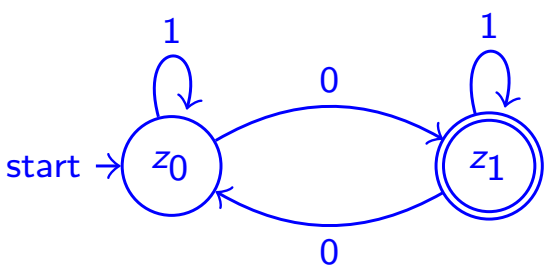
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Observe that $L = L_0 \cap L_1$ with

$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



Product Construction

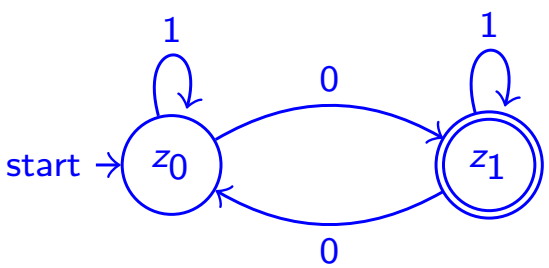
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Observe that $L = L_0 \cap L_1$ with

$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



recognizes L_0

Product Construction

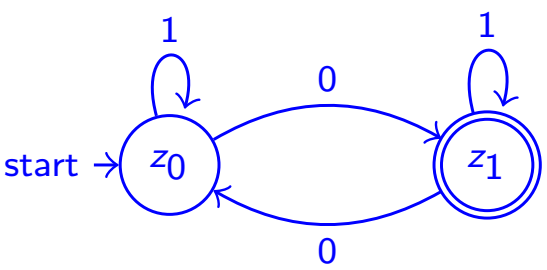
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

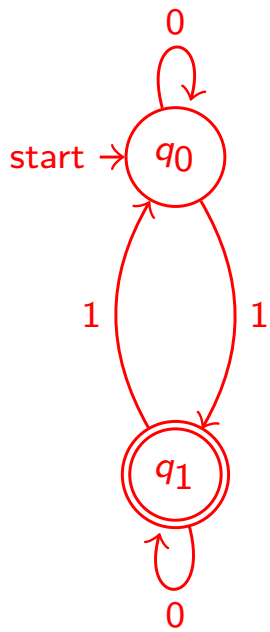
Observe that $L = L_0 \cap L_1$ with

$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



recognizes L_0



Product Construction

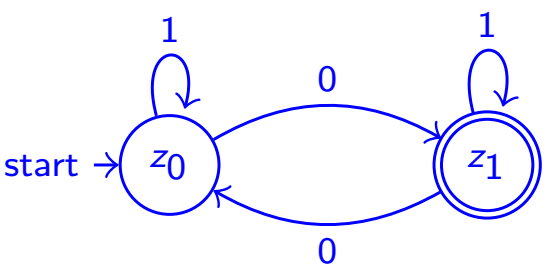
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

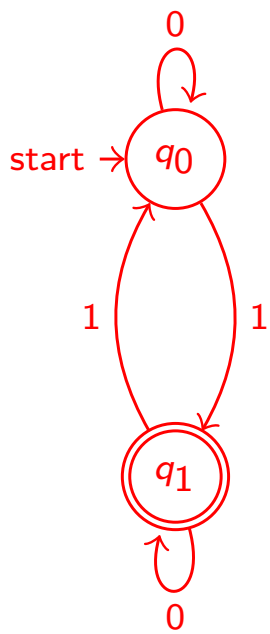
Observe that $L = L_0 \cap L_1$ with

$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



recognizes L_0



recognizes L_1

Product Construction

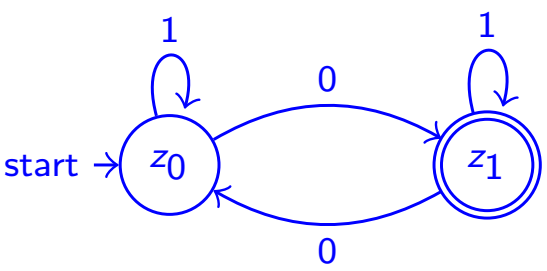
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Observe that $L = L_0 \cap L_1$ with

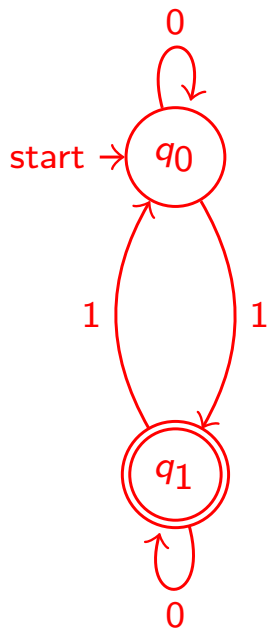
$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



recognizes L_0

$\times$



recognizes L_1

Product Construction

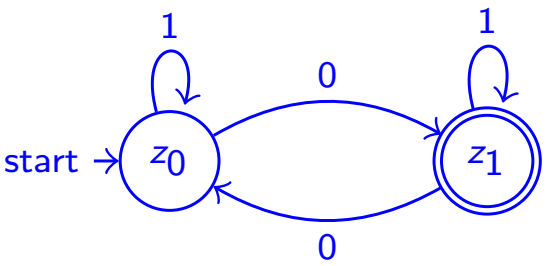
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Observe that $L = L_0 \cap L_1$ with

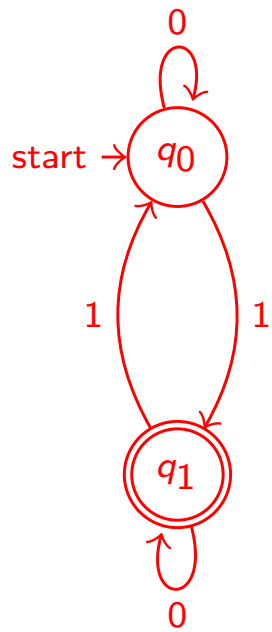
$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



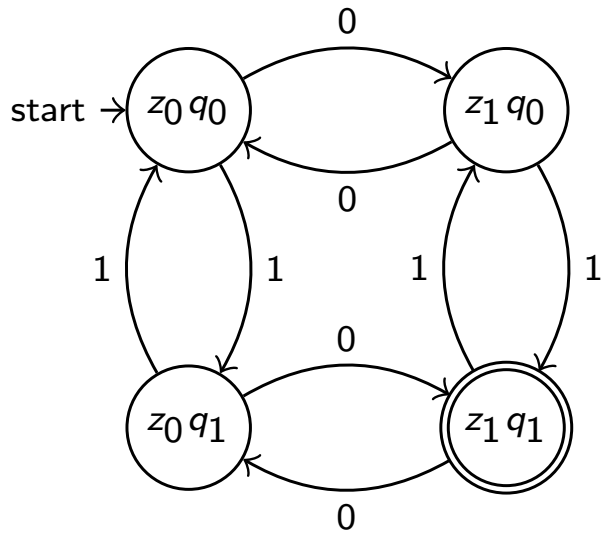
recognizes L_0

$\times$



recognizes L_1

$=$



Product Construction

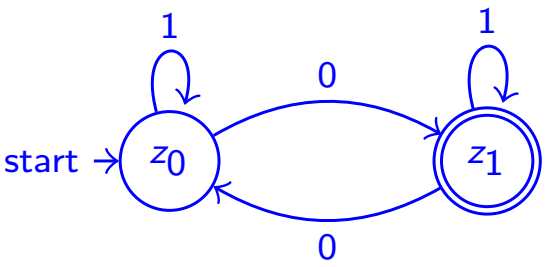
Example

Construct a DFA recognozing the language L of all binary strings with an odd number of 0's and odd number of 1's.

Observe that $L = L_0 \cap L_1$ with

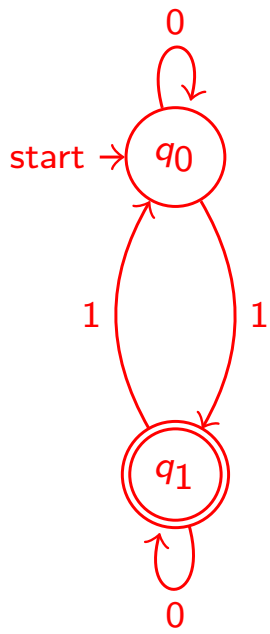
$$L_0 = \{x \in \{0,1\}^* : |x|_0 \text{ is odd}\}$$

$$L_1 = \{x \in \{0,1\}^* : |x|_1 \text{ is odd}\}$$



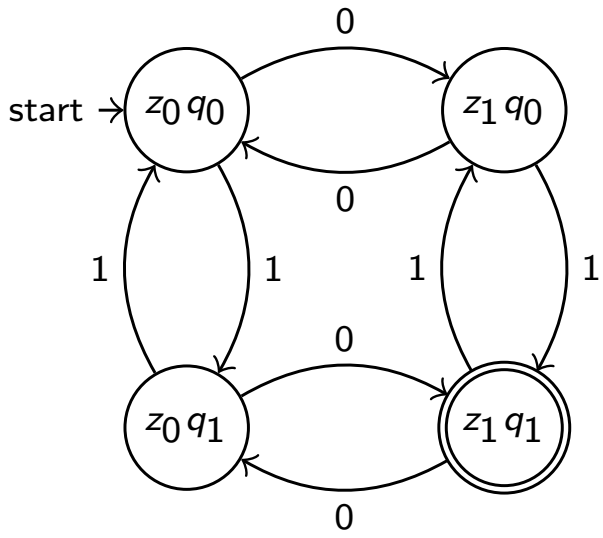
recognizes L_0

$\times$



recognizes L_1

$=$



recognizes L