41080 Theory of Computing Science
Week 3 Tutorial Class

Chuanqi Zhang

Centre for Quantum Software and Information
University of Technology Sydney

22nd August, 2024

Outline

- **Tutorial**: how to do the powerset construction for an NFA
- **Review**: regular languages and regular expressions
- **Recipe**: conversion between regular expressions and NFAs

Outline

- **Tutorial**: how to do the powerset construction for an NFA
- **Review**: regular languages and regular expressions
- **Recipe**: conversion between regular expressions and NFAs

Outline

- **Tutorial**: how to do the powerset construction for an NFA
- **Review**: regular languages and regular expressions
- **Recipe**: conversion between regular expressions and NFAs

The relationship between DFAs and NFAs

Definition (DFA)

A deterministic finite automaton (DFA) is a five tuple $(Q, \Sigma, q_0, F, \delta)$:

1. Q: a set of states;
2. Σ: an alphabet set;
3. $q_0 \in Q$: the start state;
4. $F \subseteq Q$: a set of accept states;
5. $\delta : Q \times \Sigma \to Q$: a transition function.

Definition (NFA)

A non-deterministic finite automaton (NFA) is a five tuple $(Q, \Sigma, Q_0, F, \delta)$:

1. A set of states Q;
2. The alphabet Σ;
3. $Q_0 \subseteq Q$: a set of start states;
4. $F \subseteq Q$: a set of accept states;
5. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$: a transition function.

The relationship between DFAs and NFAs

Definition (DFA)

A deterministic finite automaton (DFA) is a five tuple $(Q, \Sigma, q_0, F, \delta)$:

1. Q: a set of states;
2. Σ: an alphabet set;
3. $q_0 \in Q$: the start state;
4. $F \subseteq Q$: a set of accept states;
5. $\delta : Q \times \Sigma \to Q$: a transition function.

Definition (NFA)

A non-deterministic finite automaton (NFA) is a five tuple $(Q, \Sigma, Q_0, F, \delta)$:

1. A set of states Q;
2. The alphabet Σ;
3. $Q_0 \subseteq Q$: a set of start states;
4. $F \subseteq Q$: a set of accept states;
5. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$: a transition function.

The relationship between DFAs and NFAs

Definition (DFA)

A deterministic finite automaton (DFA) is a five tuple $(Q, \Sigma, q_0, F, \delta)$:

1. Q: a set of states;
2. Σ: an alphabet set;
3. $q_0 \in Q$: the start state;
4. $F \subseteq Q$: a set of accept states;
5. $\delta : Q \times \Sigma \rightarrow Q$: a transition function.

Definition (NFA)

A non-deterministic finite automaton (NFA) is a five tuple $(Q, \Sigma, Q_0, F, \delta)$:

1. A set of states Q;
2. The alphabet Σ;
3. $Q_0 \subseteq Q$: a set of start states;
4. $F \subseteq Q$: a set of accept states;
5. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$: a transition function.

The relationship between DFAs and NFAs

DFAs are actually a special form of NFAs.

- This basically means: given an arbitrary DFA, we can always treat it as an NFA.
- So, it is natural to wonder the converse: given an arbitrary NFA, can we always construct a DFA that recognises the same language as the original NFA does?
- The answer is YES, through the powerset construction.

The relationship between DFAs and NFAs

DFAs are actually a special form of NFAs.

- This basically means: given an arbitrary DFA, we can always treat it as an NFA.
- So, it is natural to wonder the converse: given an arbitrary NFA, can we always construct a DFA that recognises the same language as the original NFA does?
- The answer is YES, through the powerset construction.

The relationship between DFAs and NFAs

DFAs are actually a special form of NFAs.

- This basically means: given an arbitrary DFA, we can always treat it as an NFA.
- So, it is natural to wonder the converse: given an arbitrary NFA, can we always construct a DFA that recognises the same language as the original NFA does?
- The answer is YES, through the powerset construction.

UTS:QSI

CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

The relationship between DFAs and NFAs

DFAs are actually a special form of NFAs.

- This basically means: given an arbitrary DFA, we can always treat it as an NFA.
- So, it is natural to wonder the converse: given an arbitrary NFA, can we always construct a DFA that recognises the same language as the original NFA does?
- The answer is YES, through the powerset construction.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$

What's the ε-closure for each state?

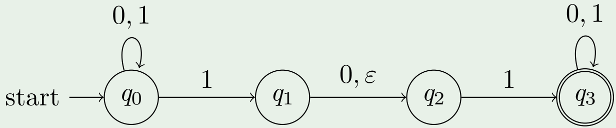
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$



What's the ε-closure for each state?

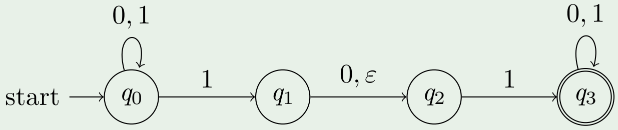
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$



What's the ε-closure for each state?

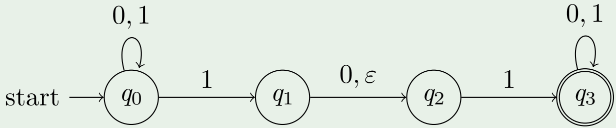
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$



What's the ε-closure for each state?

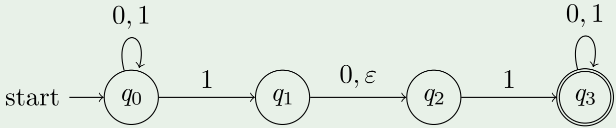
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$



What's the ε-closure for each state?

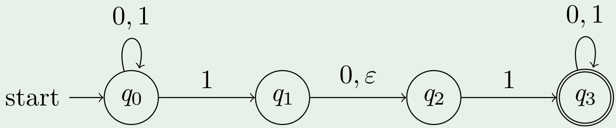
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$



What's the ε-closure for each state?

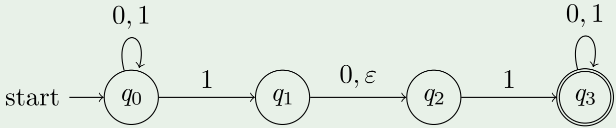
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

Tutorial: powerset construction

Definition (ε-closure)

For any state $q \in Q$, the ε-closure of q is defined as

$$\varepsilon(q) = \{q\} \cup \{q' \in Q : q' \text{ is reachable from } q \text{ by } \varepsilon\text{-transitions.}\}.$$

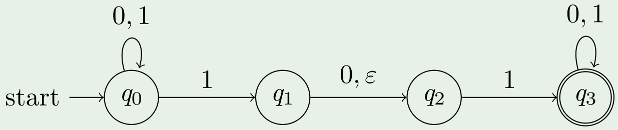


What's the ε-closure for each state?

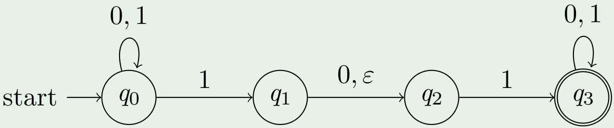
- $\varepsilon(q_0) = \{q_0\}$;
- $\varepsilon(q_1) = \{q_1, q_2\}$;
- $\varepsilon(q_2) = \{q_2\}$;
- $\varepsilon(q_3) = \{q_3\}$.

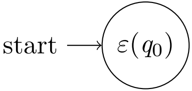
UTS:QSI

Tutorial: powerset construction

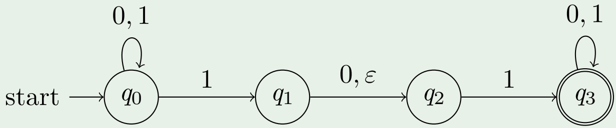


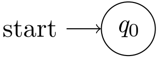
Tutorial: powerset construction



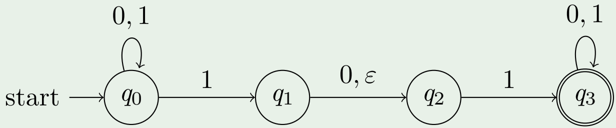


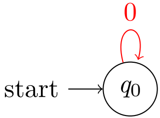
Tutorial: powerset construction



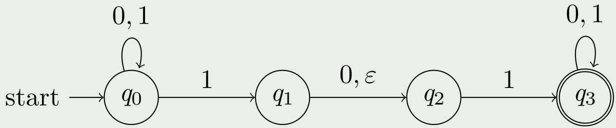


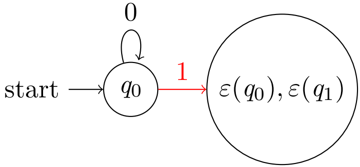
Tutorial: powerset construction



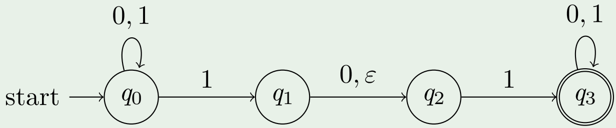


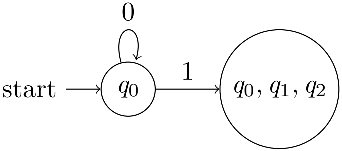
Tutorial: powerset construction



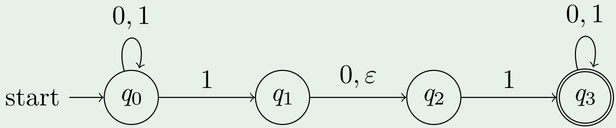


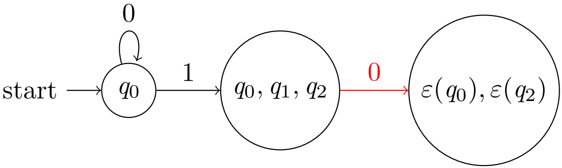
Tutorial: powerset construction



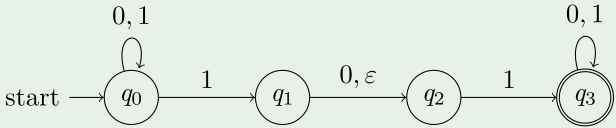


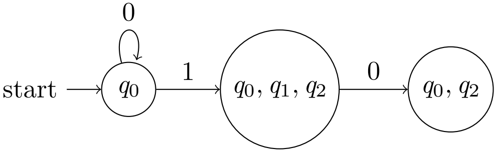
Tutorial: powerset construction



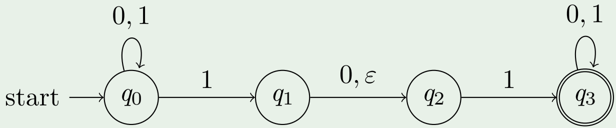


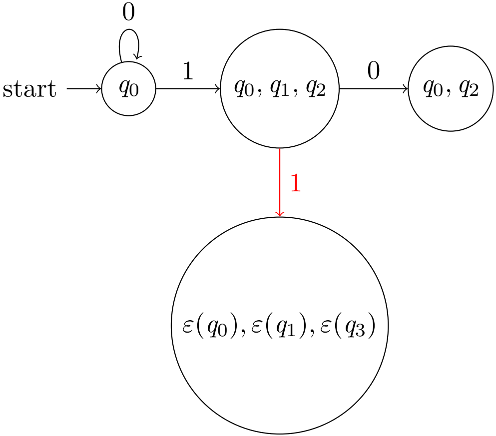
Tutorial: powerset construction



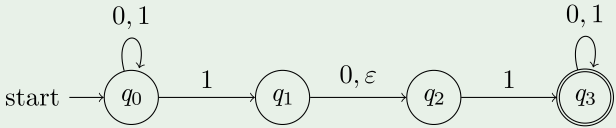


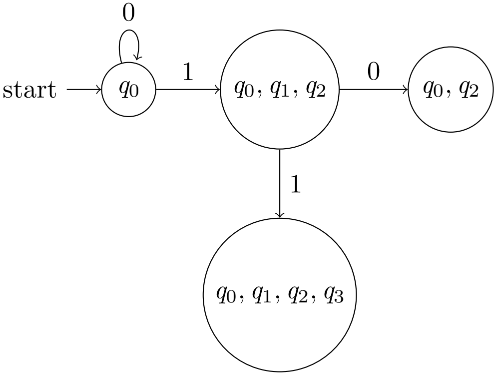
Tutorial: powerset construction



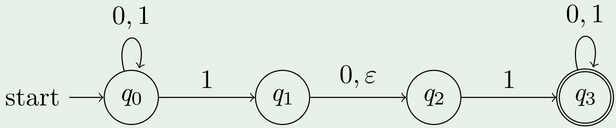


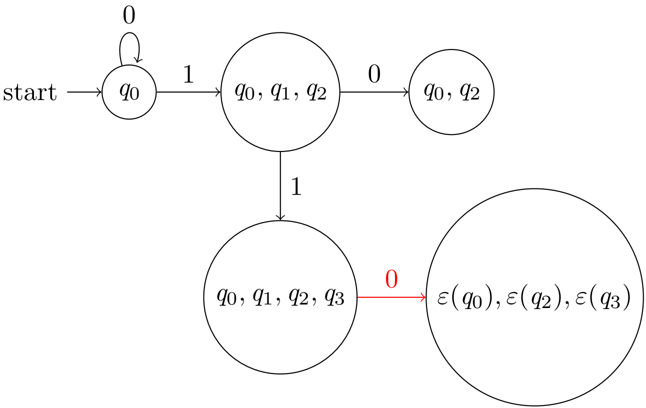
Tutorial: powerset construction



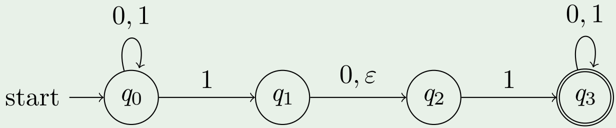


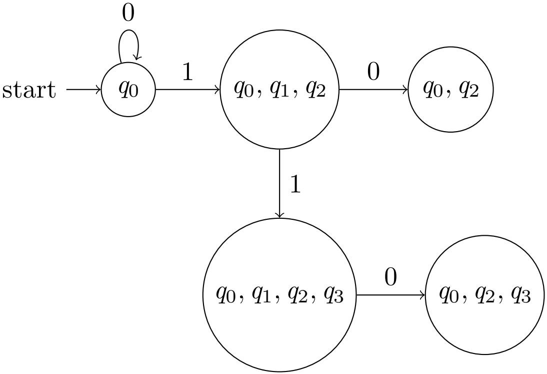
Tutorial: powerset construction



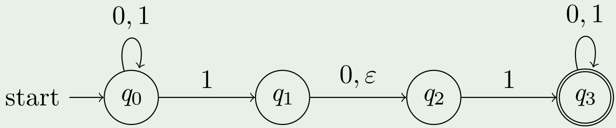


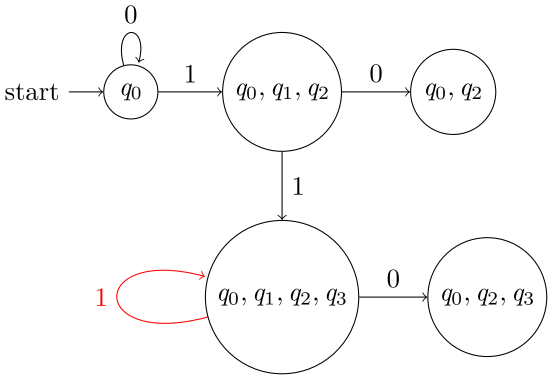
Tutorial: powerset construction



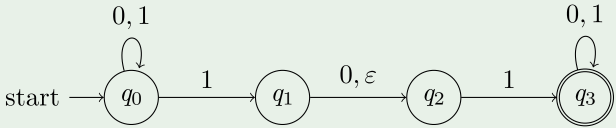


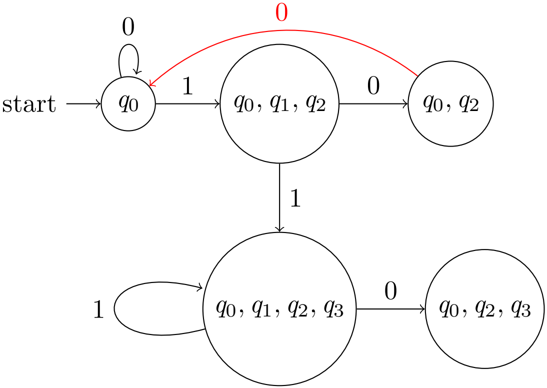
Tutorial: powerset construction



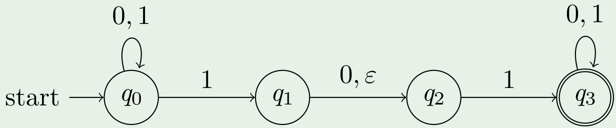


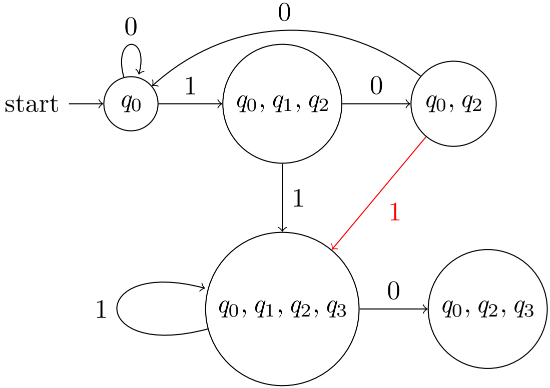
Tutorial: powerset construction



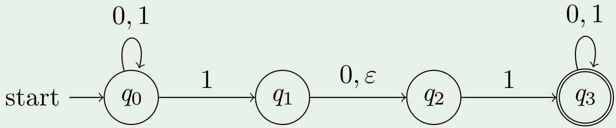


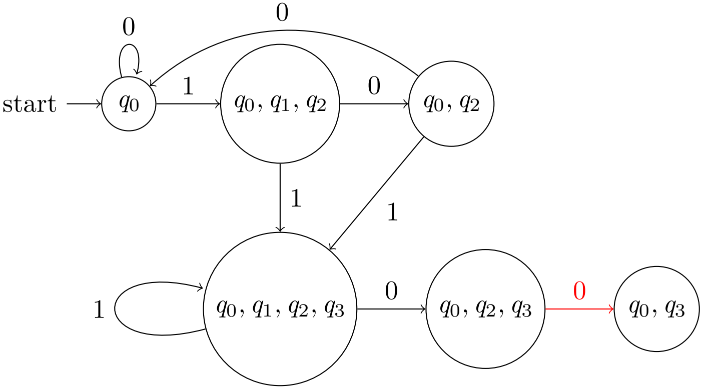
Tutorial: powerset construction



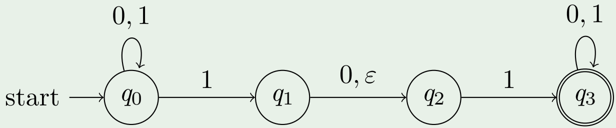


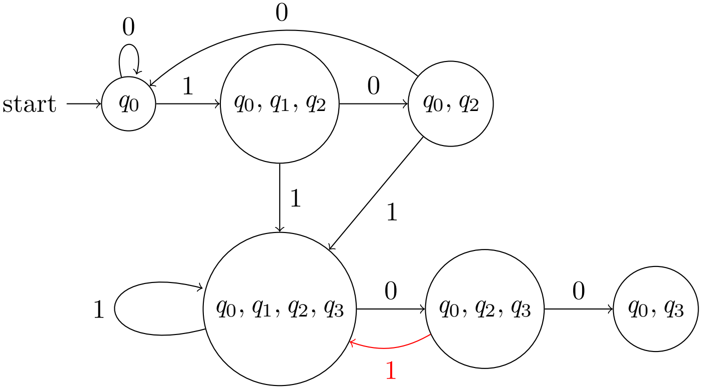
Tutorial: powerset construction



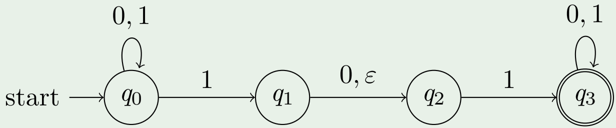


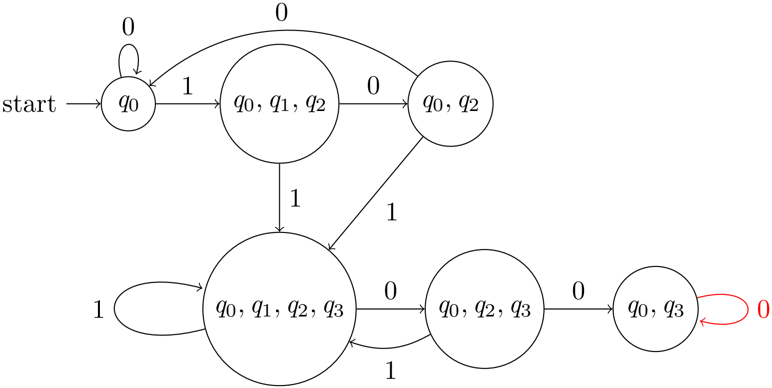
Tutorial: powerset construction



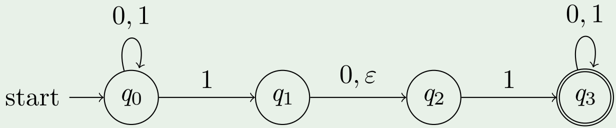


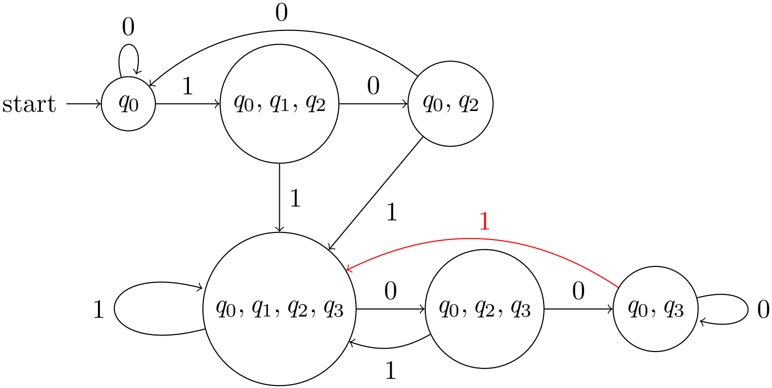
Tutorial: powerset construction



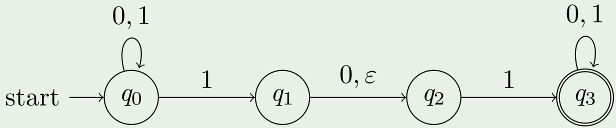


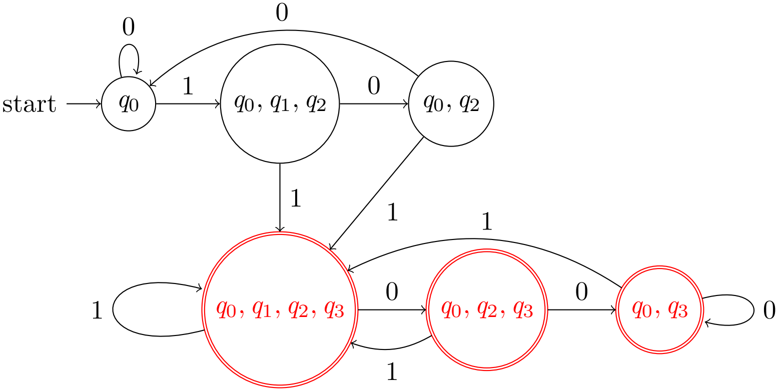
Tutorial: powerset construction



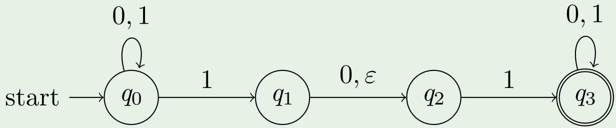


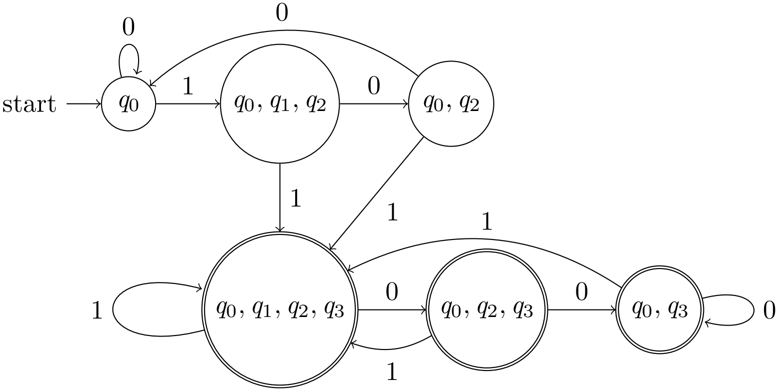
Tutorial: powerset construction





Tutorial: powerset construction





What is a regular language?

- NFAs are as powerful as DFAs.
- The range of languages that can be recognised by all NFAs is the same as that by all DFAs.
- A language can be recognised by a DFA if and only if it can be recognised by an NFA.

Definition (Regular languages)

A language is regular if there exists a DFA (or equivalently, an NFA) that recognises it.

What is a regular language?

- NFAs are as powerful as DFAs.
- The range of languages that can be recognised by all NFAs is the same as that by all DFAs.
- A language can be recognised by a DFA if and only if it can be recognised by an NFA.

Definition (Regular languages)

A language is regular if there exists a DFA (or equivalently, an NFA) that recognises it.

What is a regular language?

- NFAs are as powerful as DFAs.
- The range of languages that can be recognised by all NFAs is the same as that by all DFAs.
- A language can be recognised by a DFA if and only if it can be recognised by an NFA.

Definition (Regular languages)

A language is regular if there exists a DFA (or equivalently, an NFA) that recognises it.

What is a regular language?

- NFAs are as powerful as DFAs.
- The range of languages that can be recognised by all NFAs is the same as that by all DFAs.
- A language can be recognised by a DFA if and only if it can be recognised by an NFA.

Definition (Regular languages)

A language is regular if there exists a DFA (or equivalently, an NFA) that recognises it.

Some facts about regular languages

Theorem (Closure properties)

Regular languages are closed under the following operations:

1. *Union:* $L_1 \cup L_2 = \{w \in \Sigma^* : w \in L_1 \text{ or } w \in L_2\}$.
2. *Intersection:* $L_1 \cap L_2 = \{w \in \Sigma^* : w \in L_1 \text{ and } w \in L_2\}$.
3. *Complement:* $\neg L_1 = \{w \in \Sigma^* : w \notin L_1\}$.
4. *Reverse:* $L_1^R = \{a_k \dots a_1 \in \Sigma^* : a_1 \dots a_k \in L_1 \text{ for each } a_i \in \Sigma\}$.
5. *Concatenation:* $L_1 \circ L_2 = \{w_1 w_2 \in \Sigma^* : w_1 \in L_1 \text{ and } w_2 \in L_2\}$.
6. *Kleene star:* $L_1^* = \{w_1 \dots w_k \in \Sigma^* : w_i \in L_1\} \cup \{\varepsilon\}$.

These three are crucial for understanding the notion of regular expressions!

Some facts about regular languages

Theorem (Closure properties)

Regular languages are closed under the following operations:

1. *Union:* $L_1 \cup L_2 = \{w \in \Sigma^* : w \in L_1 \text{ or } w \in L_2\}$.
2. *Intersection:* $L_1 \cap L_2 = \{w \in \Sigma^* : w \in L_1 \text{ and } w \in L_2\}$.
3. *Complement:* $\neg L_1 = \{w \in \Sigma^* : w \notin L_1\}$.
4. *Reverse:* $L_1^R = \{a_k \dots a_1 \in \Sigma^* : a_1 \dots a_k \in L_1 \text{ for each } a_i \in \Sigma\}$.
5. *Concatenation:* $L_1 \circ L_2 = \{w_1 w_2 \in \Sigma^* : w_1 \in L_1 \text{ and } w_2 \in L_2\}$.
6. *Kleene star:* $L_1^* = \{w_1 \dots w_k \in \Sigma^* : w_i \in L_1\} \cup \{\varepsilon\}$.

These three are crucial for understanding the notion of regular expressions!

UTS:QSI CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

What is a regular expression?

A regular expression is a compact and precise way to describe a regular language.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

Note that $(R_1 R_2)$ refers to the concatenation and $(R_1 + R_2)$ refers to the union.

What is a regular expression?

A regular expression is a compact and precise way to describe a regular language.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

Note that $(R_1 R_2)$ refers to the concatenation and $(R_1 + R_2)$ refers to the union.

What is a regular expression?

A regular expression is a compact and precise way to describe a regular language.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

Note that $(R_1 R_2)$ refers to the concatenation and $(R_1 + R_2)$ refers to the union.

What is a regular expression?

A regular expression is a compact and precise way to describe a regular language.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

Note that $(R_1 R_2)$ refers to the concatenation and $(R_1 + R_2)$ refers to the union.

What is a regular expression?

A regular expression is a compact and precise way to ***describe a regular language.***

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

Note that $(R_1 R_2)$ refers to the concatenation and $(R_1 + R_2)$ refers to the union.

What is a regular expression?

A regular expression is a compact and precise way to describe a regular language.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

Note that $(R_1 R_2)$ refers to the concatenation and $(R_1 + R_2)$ refers to the union.

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

aa

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

aa

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

aa

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

aa

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

aa

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

$\cdots\ a\ \cdots\ a\ \cdots$

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

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

What is a regular expression?

Exercise 1

What does the regular expression $(10^*) + (01^*)$ mean?

The set of strings that either start with 1 and followed by any number of 0s, or start with 0 and followed by any number of 1s.

Exercise 2

What is a regular expression of $L_1 = \{w \in \{a, b\}^* \mid w \text{ contains at least two } a\text{s}\}$?

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

From regular expression to NFA

- Does a regular expression always represent a regular language?
- The answer is YES!
- Why: we can construct an NFA that recognises the language represented by the given regular expression.
- How: follow the recipe for the base case and the inductive case.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

From regular expression to NFA

- Does a regular expression always represent a regular language?
- The answer is YES!
- Why: we can construct an NFA that recognises the language represented by the given regular expression.
- How: follow the recipe for the base case and the inductive case.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

From regular expression to NFA

- Does a regular expression always represent a regular language?
- The answer is YES!
- Why: we can construct an NFA that recognises the language represented by the given regular expression.
- How: follow the recipe for the base case and the inductive case.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

From regular expression to NFA

- Does a regular expression always represent a regular language?
- The answer is YES!
- Why: we can construct an NFA that recognises the language represented by the given regular expression.
- How: follow the recipe for the base case and the inductive case.

Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

From regular expression to NFA

- Does a regular expression always represent a regular language?
- The answer is YES!
- Why: we can construct an NFA that recognises the language represented by the given regular expression.
- How: follow the recipe for the base case and the inductive case.

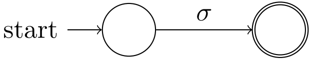
Definition (Regular expressions)

Let Σ be an alphabet. A regular expression is defined inductively as follows:

1. Base case: Any single symbol $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. We also say the empty set $\emptyset$ is a regular expression.
2. Inductive case: If R_1 and R_2 are regular expressions, then $(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regular expressions.

From regular expression to NFA

Base case:



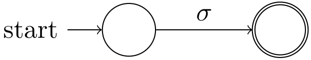
The NFA recognising a single symbol $\sigma \in \Sigma$.

The NFA recognising a single ε.

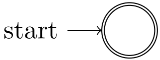
The NFA recognising an empty set $\emptyset$.

From regular expression to NFA

Base case:



The NFA recognising a single symbol $\sigma \in \Sigma$.

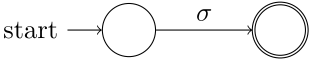


The NFA recognising a single ε.

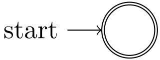
The NFA recognising an empty set $\emptyset$.

From regular expression to NFA

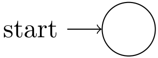
Base case:



The NFA recognising a single symbol $\sigma \in \Sigma$.



The NFA recognising a single ε.



The NFA recognising an empty set $\emptyset$.

From regular expression to NFA

Inductive case:

- How to deal with the star operation on an NFA:
  1. Use ε-transition(s) to connect the accept state(s) to the start state.
  2. Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.
- How to deal with the concatenation of two NFAs:
  1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
  2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).
- How to deal with the union of two NFAs:
  1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

From regular expression to NFA

Inductive case:

- How to deal with the star operation on an NFA:
  1. Use ε-transition(s) to connect the accept state(s) to the start state.
  2. Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.

- How to deal with the concatenation of two NFAs:
  1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
  2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).

- How to deal with the union of two NFAs:
  1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

From regular expression to NFA

Inductive case:

- How to deal with the star operation on an NFA:
  1. Use ε-transition(s) to connect the accept state(s) to the start state.
  2. Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.
- How to deal with the concatenation of two NFAs:
  1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
  2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).
- How to deal with the union of two NFAs:
  1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

From regular expression to NFA

Recipe (how to deal with the star operation)

- Use ε-transition(s) to connect the accept state(s) to the start state.
- Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.

Let's try to construct an NFA recognising $\mathbf{0}^*$!

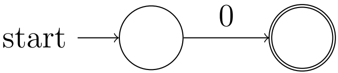
From regular expression to NFA

Recipe (how to deal with the star operation)

1. Use ε-transition(s) to connect the accept state(s) to the start state.
2. Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.

Let's try to construct an NFA recognising $\mathbf{0}^*$!

First, we recall the base case that recognises a single 0.



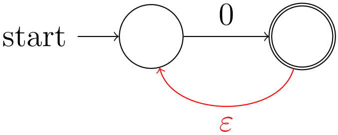
From regular expression to NFA

Recipe (how to deal with the star operation)

1. Use ε-transition(s) to connect the accept state(s) to the start state.
2. Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.

Let's try to construct an NFA recognising $\mathbf{0}^*$!

Then, we follow the recipe of the star operation.



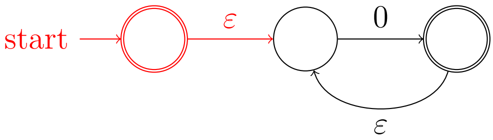
From regular expression to NFA

Recipe (how to deal with the star operation)

1. Use ε-transition(s) to connect the accept state(s) to the start state.
2. Draw a new start state and use an ε-transition to connect it to the original start state. Also make the new start state acceptable.

Let's try to construct an NFA recognising $\mathbf{0}^*$!

Then, we follow the recipe of the star operation.



From regular expression to NFA

Recipe (how to deal with the concatenation operation)

1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).

Let's try to construct an NFA recognising 10*!

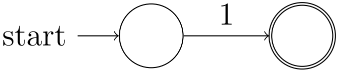
From regular expression to NFA

Recipe (how to deal with the concatenation operation)

1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).

Let's try to construct an NFA recognising 10*!

First, we recall the base case that recognises a single 1.



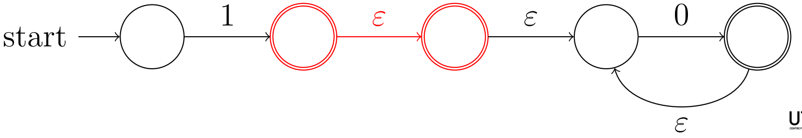
From regular expression to NFA

Recipe (how to deal with the concatenation operation)

1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).

Let's try to construct an NFA recognising 10*!

Then, we follow the recipe of the concatenation operation.



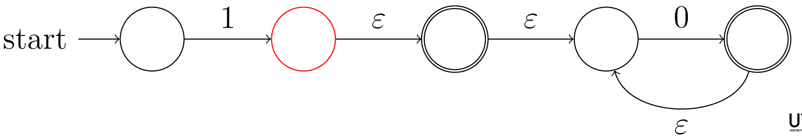
UTS:QSI

Recipe (how to deal with the concatenation operation)

1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).

Let's try to construct an NFA recognising 10*!

Then, we follow the recipe of the concatenation operation.



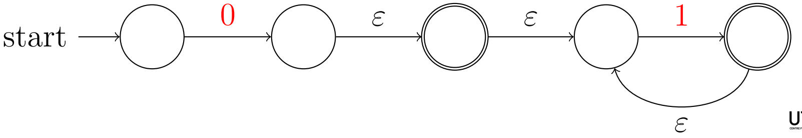
From regular expression to NFA

Recipe (how to deal with the concatenation operation)

1. Use ε-transition(s) to connect the accept state(s) in the first NFA to the start state in the second NFA.
2. Remain the accept state(s) in the second NFA but change the accept state(s) in the first NFA to the general state(s).

Let's try to construct an NFA recognising 10^*!

Similarly, the NFA recognising 01^* would be



From regular expression to NFA

Recipe (how to deal with the union operation)

1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

Let's try to construct an NFA recognising $(\mathbf{10}^*)+(\mathbf{01}^*)$!

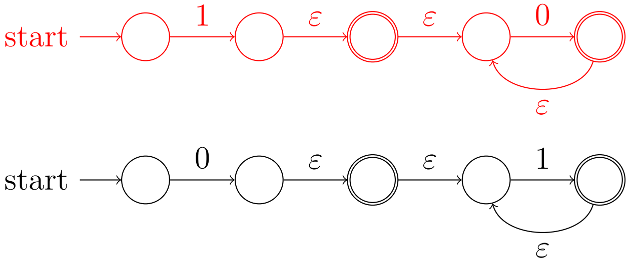
From regular expression to NFA

Recipe (how to deal with the union operation)

1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

Let's try to construct an NFA recognising $(\mathbf{10^*})+(\mathbf{01^*})$!

First, we recall the NFAs that recognise 10^* and 01^*, respectively.



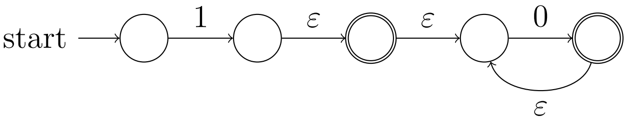
From regular expression to NFA

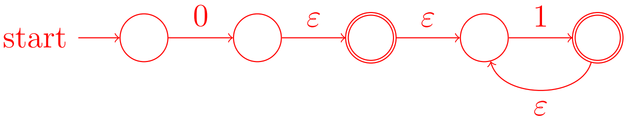
Recipe (how to deal with the union operation)

1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

Let's try to construct an NFA recognising $(\mathbf{10^*})+(\mathbf{01^*})$!

First, we recall the NFAs that recognise 10^* and 01^*, respectively.





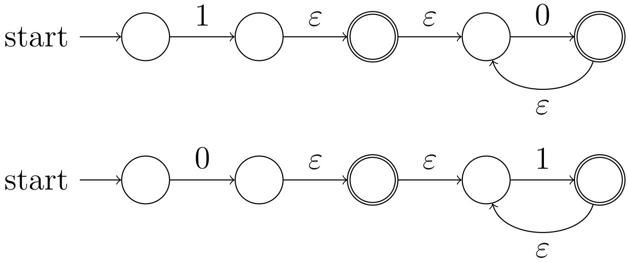
From regular expression to NFA

Recipe (how to deal with the union operation)

1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

Let's try to construct an NFA recognising $(\mathbf{10^*})+(\mathbf{01^*})$!

First, we recall the NFAs that recognise 10^* and 01^*, respectively.



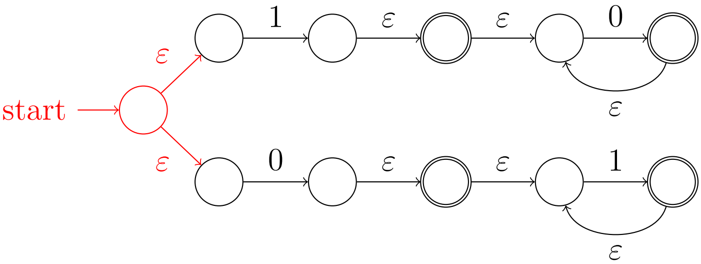
From regular expression to NFA

Recipe (how to deal with the union operation)

1. Draw a new start state and use ε-transitions to connect it to the original start states in the two NFAs.

Let's try to construct an NFA recognising $(\mathbf{10}^*)+(\mathbf{01}^*)$!

Then, we follow the recipe of the union operation.

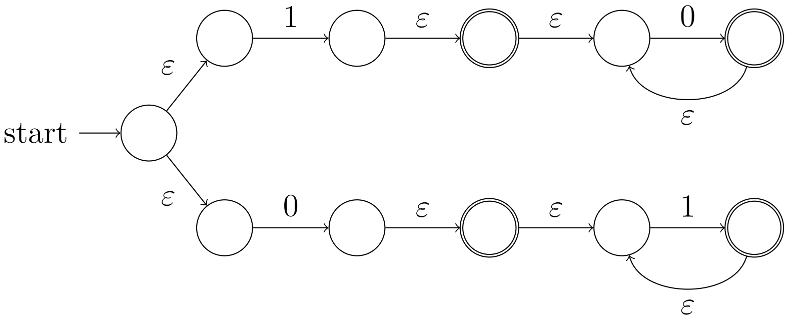


From regular expression to NFA

Exercise

Draw an NFA recognising the language represented by $(10^*)+(01^*)$.

Solution:



From NFA to regular expression

- Given a regular language, can we always find a regular expression that represents?
- The answer is also YES!
- Why: we can construct a generalised NFA that recognises the given language where the arrows can carry regular expressions.
- How: follow the recipe to eliminate the states one-by-one.

From NFA to regular expression

- Given a regular language, can we always find a regular expression that represents?
- The answer is also YES!
- Why: we can construct a generalised NFA that recognises the given language where the arrows can carry regular expressions.
- How: follow the recipe to eliminate the states one-by-one.

From NFA to regular expression

- Given a regular language, can we always find a regular expression that represents?
- The answer is also YES!
- Why: we can construct a generalised NFA that recognises the given language where the arrows can carry regular expressions.
- How: follow the recipe to eliminate the states one-by-one.

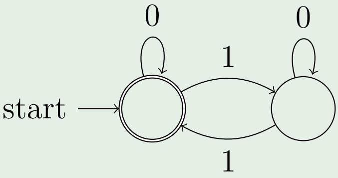
From NFA to regular expression

- Given a regular language, can we always find a regular expression that represents?
- The answer is also YES!
- Why: we can construct a generalised NFA that recognises the given language where the arrows can carry regular expressions.
- How: follow the recipe to eliminate the states one-by-one.

From NFA to regular expression

Exercise

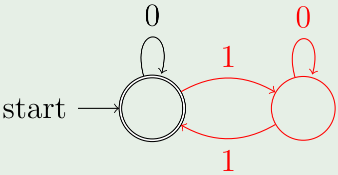
Write a regular expression representing the language recognised by the following NFA.



From NFA to regular expression

Exercise

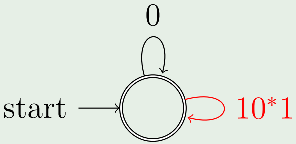
Write a regular expression representing the language recognised by the following NFA.



From NFA to regular expression

Exercise

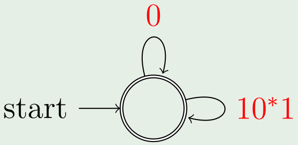
Write a regular expression representing the language recognised by the following NFA.



From NFA to regular expression

Exercise

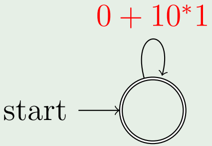
Write a regular expression representing the language recognised by the following NFA.



From NFA to regular expression

Exercise

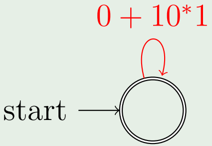
Write a regular expression representing the language recognised by the following NFA.



From NFA to regular expression

Exercise

Write a regular expression representing the language recognised by the following NFA.



Solution: $(0 + 10^*1)^*$.