Finite Automata and Monadic Second Order Logic

Sam van Gool

Monday, April 29, 2019

Logic and Computation, Master's course, Utrecht University

Central question in theory of computation

What are the fundamental capabilities

and limitations of computers?

Three types of sub-questions, plus a bonus question

- Complexity theory
  - How *hard* is a computational problem?
  - How much *time*? How much *memory space*?

Three types of sub-questions, plus a bonus question

- Complexity theory
  - How *hard* is a computational problem?
  - How much *time*? How much *memory space*?
- Computability theory
  - Is the answer to a problem *computable*, or not?

Three types of sub-questions, plus a bonus question

- Complexity theory
  - How *hard* is a computational problem?
  - How much *time*? How much *memory space*?
- Computability theory
  - Is the answer to a problem *computable*, or not?
- Automata theory
  - What is a *minimal* model for computation?

Three types of sub-questions, plus a bonus question

- Complexity theory
  - How *hard* is a computational problem?
  - How much *time*? How much *memory space*?
- Computability theory
  - Is the answer to a problem *computable*, or not?
- Automata theory
  - What is a *minimal* model for computation?
- For each of the above, what is the connection to *logic*?

This week

- Today: Automata and their connection to monadic second order logic.
- Wednesday: Turing machines, complexity classes, and their connection to logic.

This week

- Today: Automata and their connection to monadic second order logic.
- Wednesday: Turing machines, complexity classes, and their connection to logic.
- **Reminder!** Top-3 topic preferences due on Blackboard on Wednesday at 5pm.
- **Reminder!** Research question & choice of peer reviewer on Friday at 5pm.

Overview for today

Deterministic Finite Automata

Regular languages and non-determinism

Monadic Second Order Logic

The NFA-MSO Connection

Deterministic Finite Automata

What is a computer?

- ‘New Computer’ (external video link)

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

- Initial input
- Ability to read from memory
- Ability to write to memory
- Final (yes/no) output
- Internal states

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

- Initial input
- Ability to read from memory
- Ability to write to memory
- Final (yes/no) output
- Internal states

An automaton has:

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

- Initial input
- Ability to read from memory
- Ability to write to memory
- Final (yes/no) output
- Internal states

An automaton has:

- Initial input

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

- Initial input
- Ability to read from memory
- Ability to write to memory
- Final (yes/no) output
- Internal states

An automaton has:

- Initial input
- Ability to read from memory:

  once and in one direction

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

- Initial input
- Ability to read from memory
- Ability to write to memory
- Final (yes/no) output
- Internal states

An automaton has:

- Initial input
- Ability to read from memory:

  once and in one direction

- ~~Ability to write to memory~~

What is a computer?

- ‘New Computer’ (external video link)

> What are the essential components of a computer?

A computer (Turing machine) has:

- Initial input
- Ability to read from memory
- Ability to write to memory
- Final (yes/no) output
- Internal states

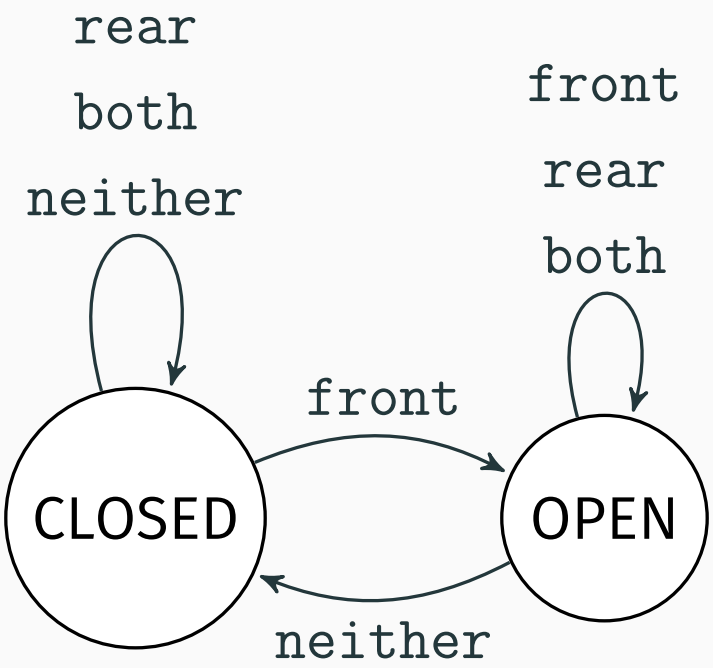
An automaton has:

- Initial input
- Ability to read from memory:

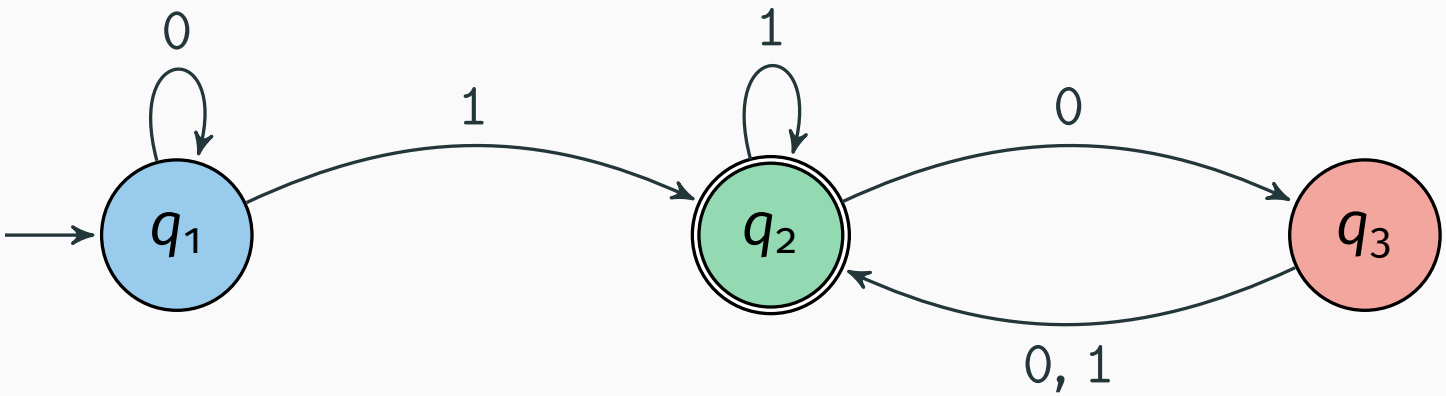
  once and in one direction

- ~~Ability to write to memory~~
- Final (yes/no) output
- Internal states

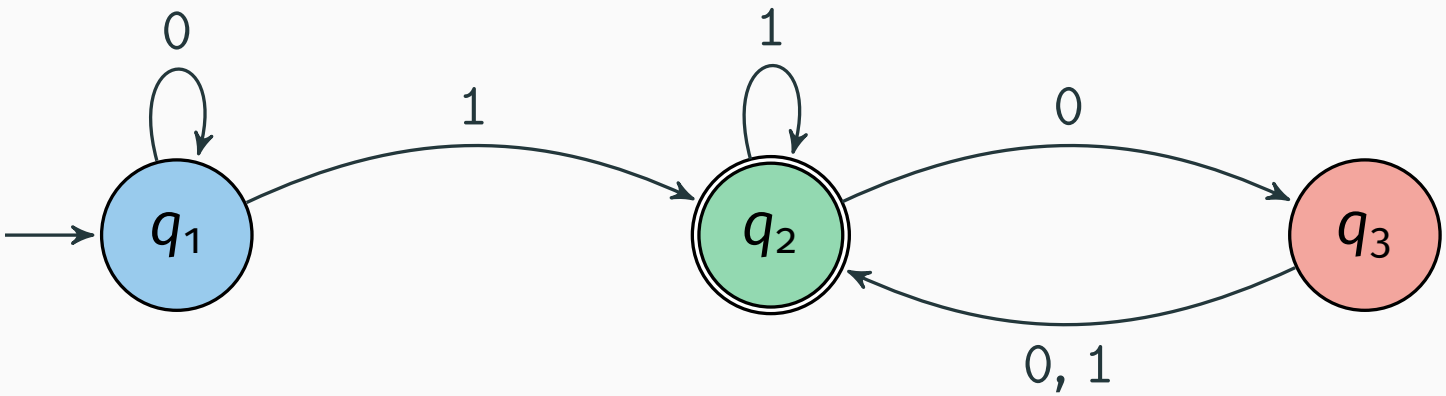
Automatic door automaton



An automaton reading binary strings

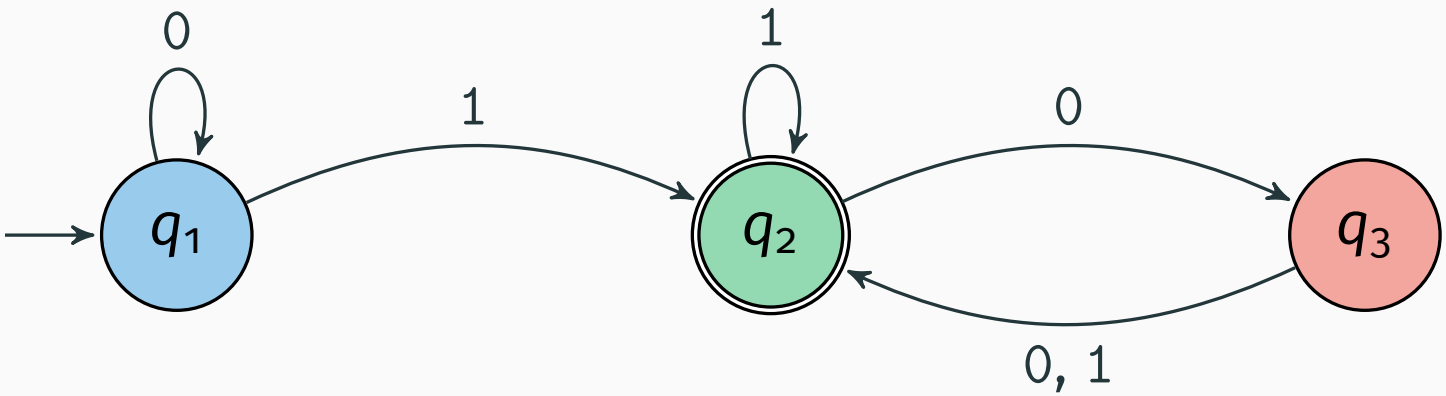


An automaton reading binary strings



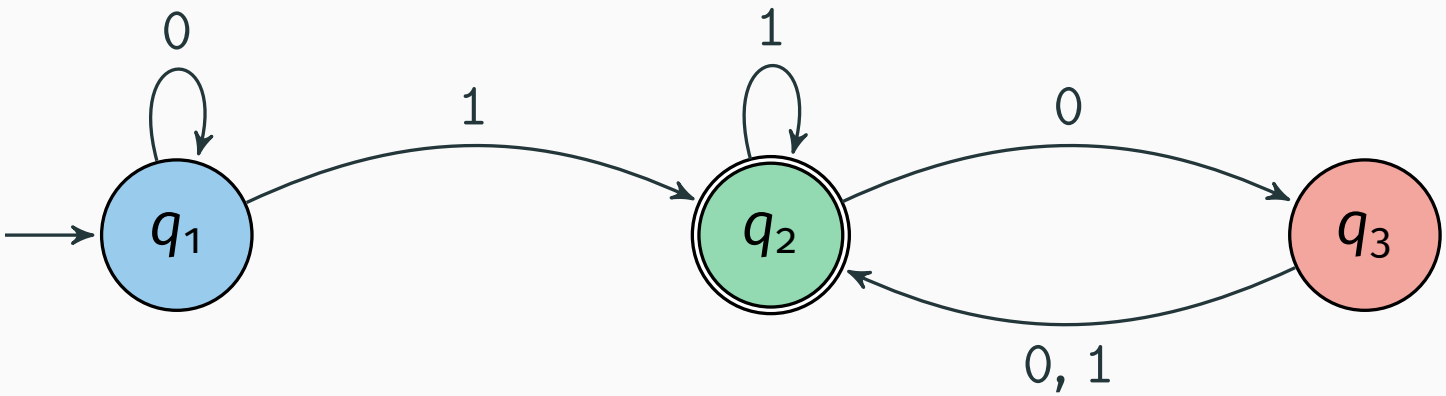
Input	Computation	Last state	Accepted?
000			

An automaton reading binary strings



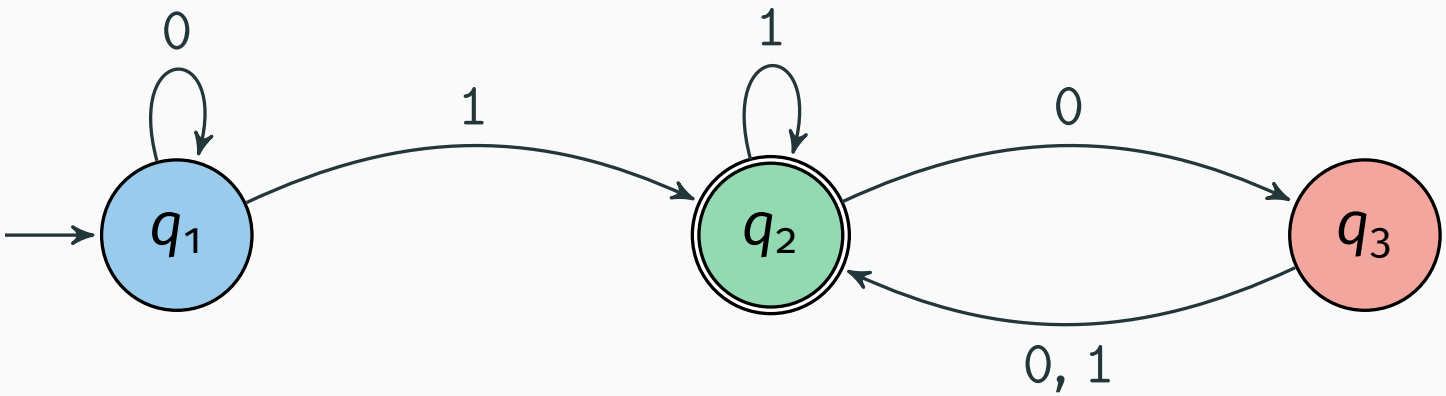
Input	Computation	Last state	Accepted?
000	$q_1 {}^0 q_1 {}^0 q_1 {}^0 q_1$	q_1	

An automaton reading binary strings



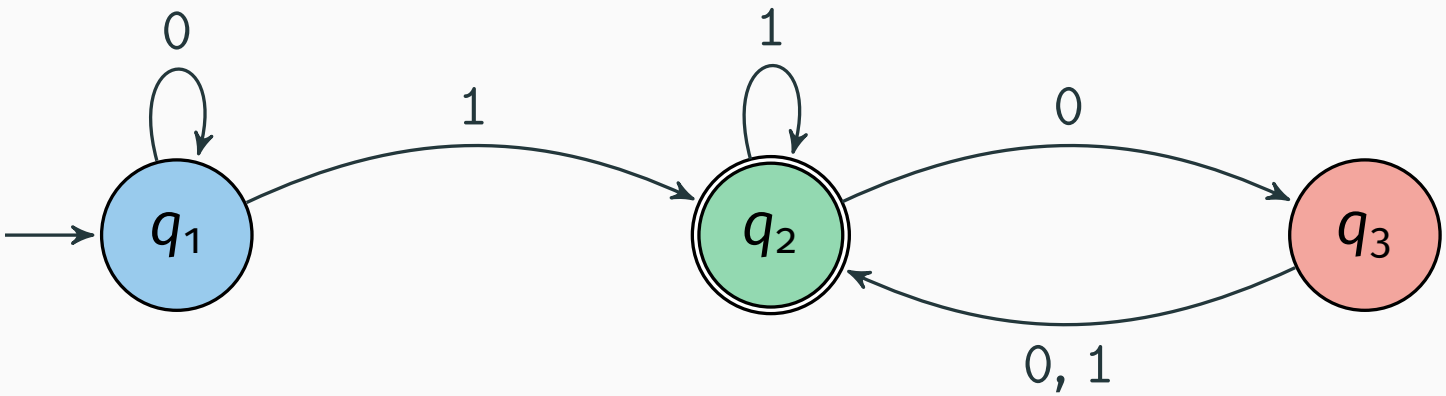
Input	Computation	Last state	Accepted?
000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
0010010			

An automaton reading binary strings



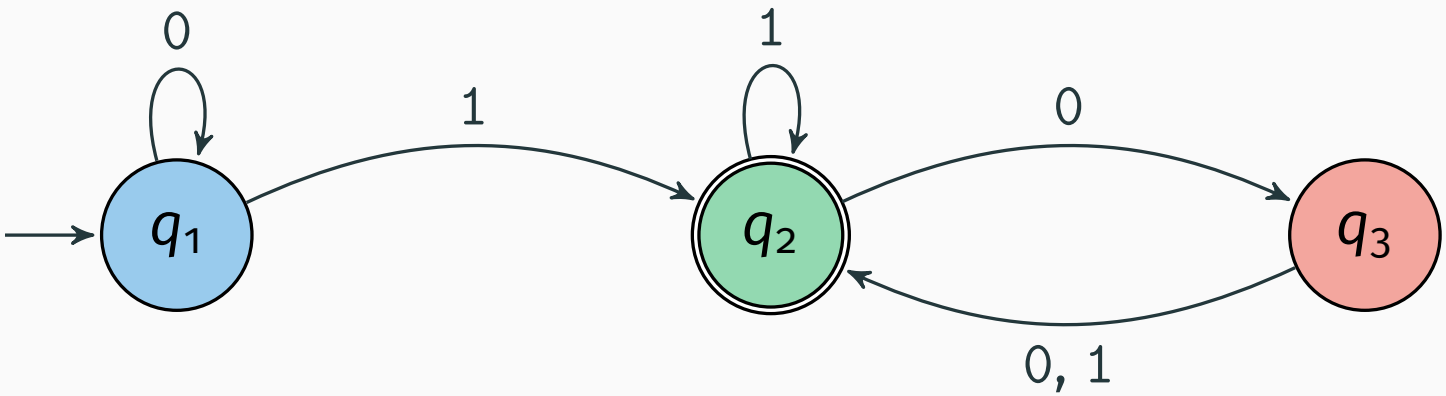
Input	Computation	Last state	Accepted?
000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
0010010	$q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_2{}^1 q_2{}^0 q_3$	q_3	

An automaton reading binary strings



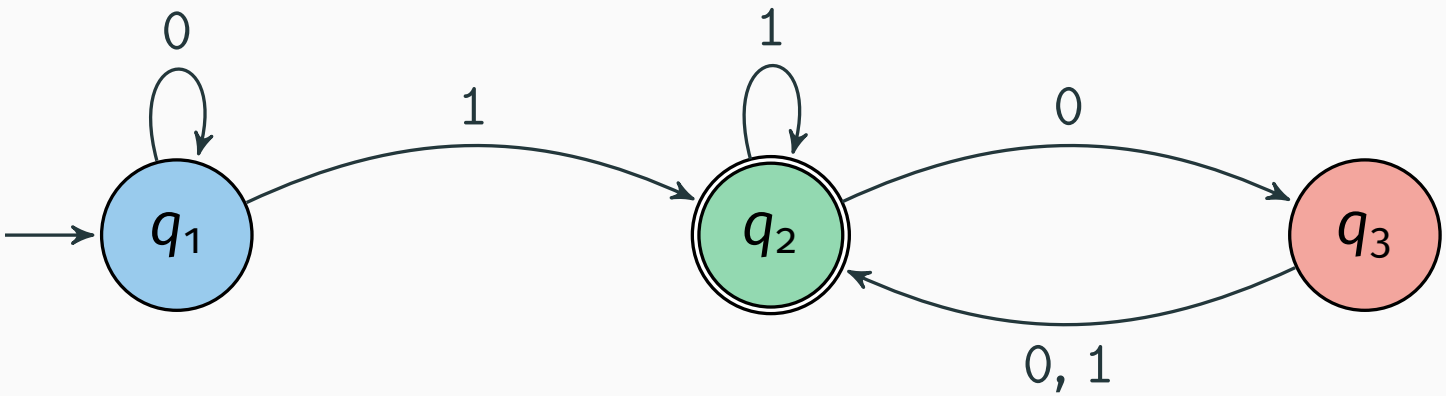
Input	Computation	Last state	Accepted?
000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
0010010	$q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_2{}^1 q_2{}^0 q_3$	q_3	No
100			

An automaton reading binary strings



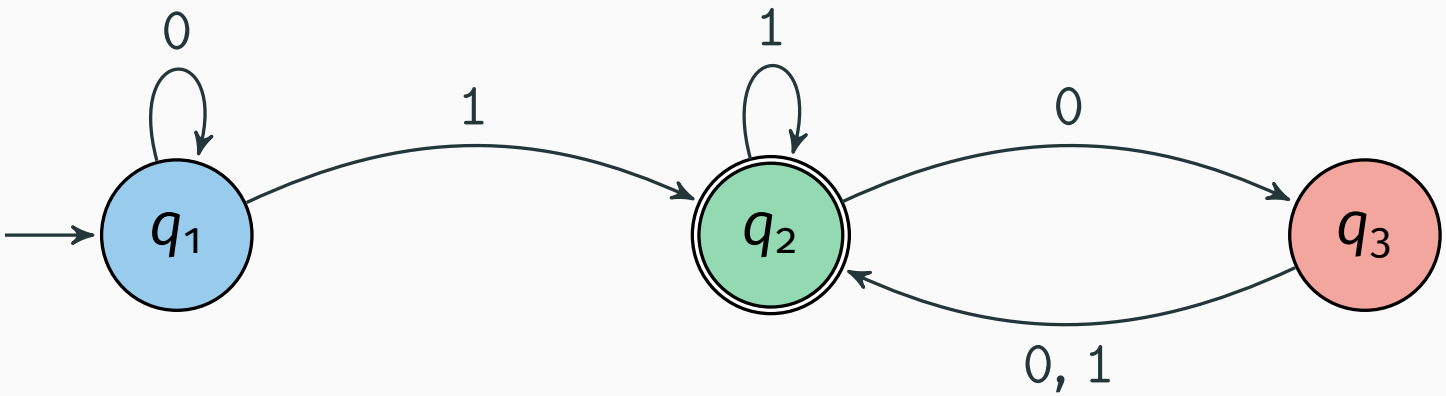
Input	Computation	Last state	Accepted?
000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
0010010	$q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_2{}^1 q_2{}^0 q_3$	q_3	No
100	$q_1{}^1 q_2{}^0 q_3{}^0 q_2$	q_2	

An automaton reading binary strings



Input	Computation	Last state	Accepted?
000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
0010010	$q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_2{}^1 q_2{}^0 q_3$	q_3	No
100	$q_1{}^1 q_2{}^0 q_3{}^0 q_2$	q_2	Yes

An automaton reading binary strings



Input	Computation	Last state	Accepted?
000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
0010010	$q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_2{}^1 q_2{}^0 q_3$	q_3	No
100	$q_1{}^1 q_2{}^0 q_3{}^0 q_2$	q_2	Yes

Question

Which strings are accepted by this automaton?

Words and languages

Definition

- An *alphabet* is a finite set, Σ.
- A Σ-*word* is a finite sequence of elements from Σ.
- For any word w, $|w|$ denotes the *length* of w.
- The *empty sequence* also counts as a word, and is denoted by ϵ.
- Σ^* denotes the (infinite) set of all Σ-words.
- A Σ-*language* is a subset of Σ^*, that is, a set of Σ-words.

Words and languages

Definition

- An *alphabet* is a finite set, Σ.
- A Σ-*word* is a finite sequence of elements from Σ.
- For any word w, $|w|$ denotes the *length* of w.
- The *empty sequence* also counts as a word, and is denoted by ϵ.
- Σ^* denotes the (infinite) set of all Σ-words.
- A Σ-*language* is a subset of Σ^*, that is, a set of Σ-words.
- Idea: a language *is* a computational problem, namely: to determine whether or not a given input word belongs to it.

Words and languages

Examples

- $\Sigma = \{u, t, r, e, c, h, U, T, R, E, C, H\}$. Three examples of (distinct!) Σ-words are: Utrecht, UTRECHT, utrecht.

Examples

- $\Sigma = \{u, t, r, e, c, h, U, T, R, E, C, H\}$. Three examples of (distinct!) Σ-words are: Utrecht, UTRECHT, utrecht. Three more Σ-words: church, Ruth, eUrHcccHreCCCCCh.
  - $>$ Is there a shortest Σ-word? If yes, what is it?

Examples

- $\Sigma = \{u, t, r, e, c, h, U, T, R, E, C, H\}$. Three examples of (distinct!) Σ-words are: Utrecht, UTRECHT, utrecht. Three more Σ-words: church, Ruth, eUrHcccHreCCCCCh.
  - > Is there a shortest Σ-word? If yes, what is it?
  - > Is there a longest Σ-word?

Examples

- $\Sigma = \{u, t, r, e, c, h, U, T, R, E, C, H\}$. Three examples of (distinct!) Σ-words are: Utrecht, UTRECHT, utrecht. Three more Σ-words: church, Ruth, eUrHcccHreCCCCCh.
  - $>$ Is there a shortest Σ-word? If yes, what is it?
  - $>$ Is there a longest Σ-word?
- Two examples of Σ-*languages*:
  - $L = \{w \in \Sigma^* \mid w \text{ is an existing English word}\}$.
  - $L = \{w \in \Sigma^* \mid w \text{ starts with 'e'}\}$.

Examples

- $\Sigma = \{u, t, r, e, c, h, U, T, R, E, C, H\}$. Three examples of (distinct!) Σ-words are: Utrecht, UTRECHT, utrecht. Three more Σ-words: church, Ruth, eUrHcccHreCCCCCh.
  - > Is there a shortest Σ-word? If yes, what is it?
  - > Is there a longest Σ-word?
- Two examples of Σ-*languages*:
  - $L = \{w \in \Sigma^* \mid w \text{ is an existing English word}\}$.
  - $L = \{w \in \Sigma^* \mid w \text{ starts with 'e'}\}$.
- $\Sigma = \{\texttt{front}, \texttt{rear}, \texttt{both}, \texttt{neither}\}$. Two examples of Σ-words are: (`front`, `front`, `both`, `neither`), and (`neither`, `neither`, `neither`, `neither`, `neither`)
  The second word can be briefly denoted as: $\texttt{neither}^5$.

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta\colon Q \times \Sigma \to Q$ is a transition function,

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta\colon Q \times \Sigma \to Q$ is a transition function,
  - q_0 is an element of Q, the start state,

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta\colon Q \times \Sigma \to Q$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta\colon Q \times \Sigma \to Q$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.
- $q \xrightarrow{a} q'$ means: q is a state in Q, a is a letter in Σ, and $\delta(q, a) = q'$.

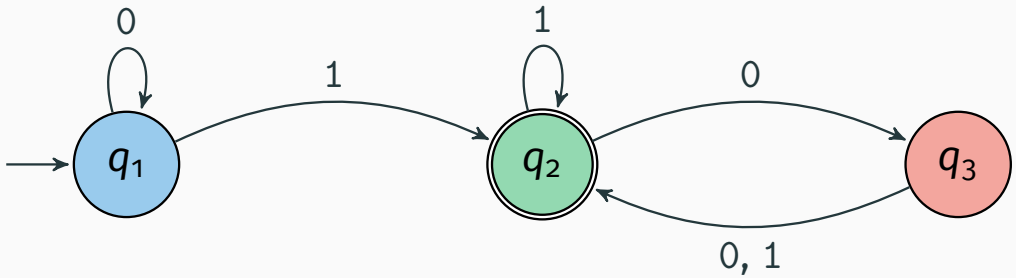
Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta\colon Q \times \Sigma \to Q$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.
- $q \xrightarrow{a} q'$ means: q is a state in Q, a is a letter in Σ, and $\delta(q, a) = q'$.
- Let $w = a_1 \dots a_n \in \Sigma^*$ be a Σ-word of length n. Then:
  - a computation of an automaton $\mathcal{A}$ on w is a sequence $q_0 \dots q_n$ of states such that $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$;
  - a computation is accepting if the last state, q_n, is in F;
  - $\mathcal{A}$ accepts w if it has an accepting computation on w.

Definition of deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta\colon Q \times \Sigma \to Q$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.
- $q \xrightarrow{a} q'$ means: q is a state in Q, a is a letter in Σ, and $\delta(q, a) = q'$.
- Let $w = a_1 \dots a_n \in \Sigma^*$ be a Σ-word of length n. Then:
  - a computation of an automaton $\mathcal{A}$ on w is a sequence $q_0 \dots q_n$ of states such that $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$;
  - a computation is accepting if the last state, q_n, is in F;
  - $\mathcal{A}$ accepts w if it has an accepting computation on w.
- The language recognized by $\mathcal{A}$ is $L_{\mathcal{A}} := \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$.

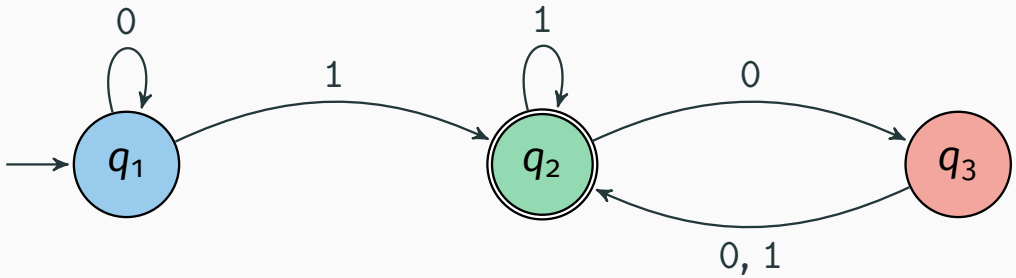
An automaton reading binary strings



For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

> $Q =$

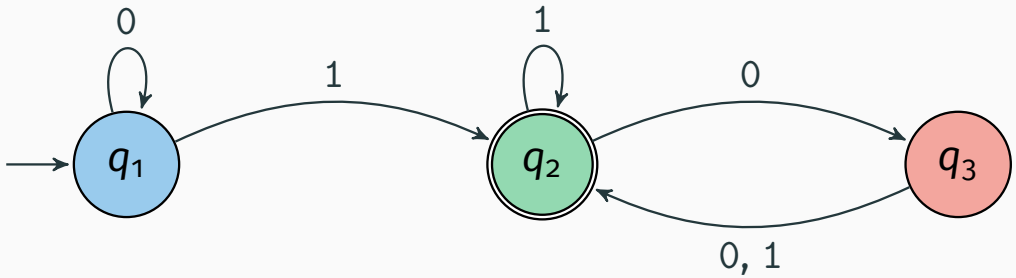
An automaton reading binary strings



For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_1, q_2, q_3\}$.
- $\Sigma =$

An automaton reading binary strings



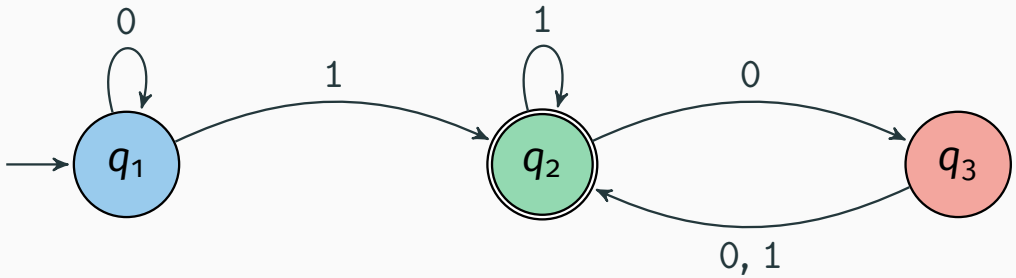
For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

> $Q = \{q_1, q_2, q_3\}$.

> $\Sigma = \{0, 1\}$.

> $\delta =$

An automaton reading binary strings



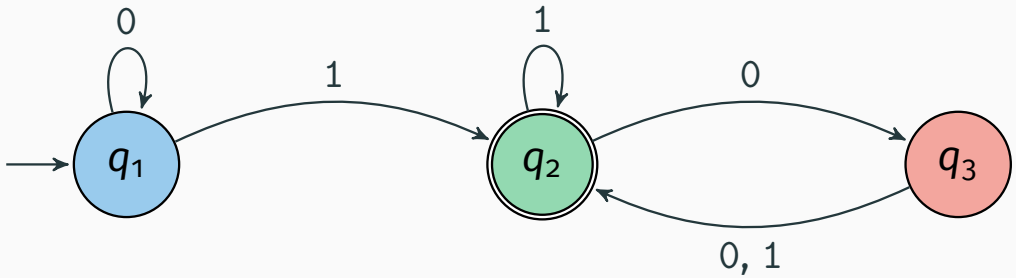
For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_1, q_2, q_3\}$.
- $\Sigma = \{0, 1\}$.
- $\delta =$

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- $q_0 =$

An automaton reading binary strings



For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

> $Q = \{q_1, q_2, q_3\}$.

> $\Sigma = \{0, 1\}$.

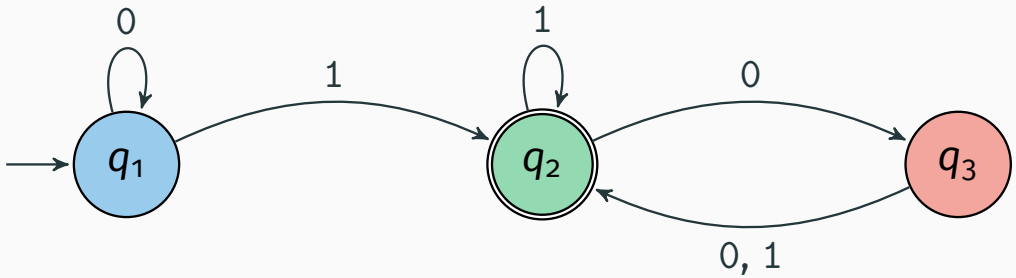
> $\delta =$

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

> $q_0 = q_1$.

> $F =$

An automaton reading binary strings



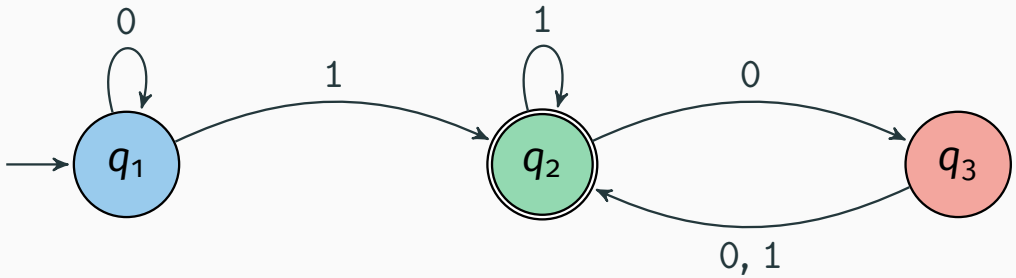
For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_1, q_2, q_3\}$.
- $\Sigma = \{0, 1\}$.
- $\delta =$

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- $q_0 = q_1$.
- $F = \{q_2\}$.

An automaton reading binary strings



For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_1, q_2, q_3\}$.
- $\Sigma = \{0, 1\}$.
- $\delta =$

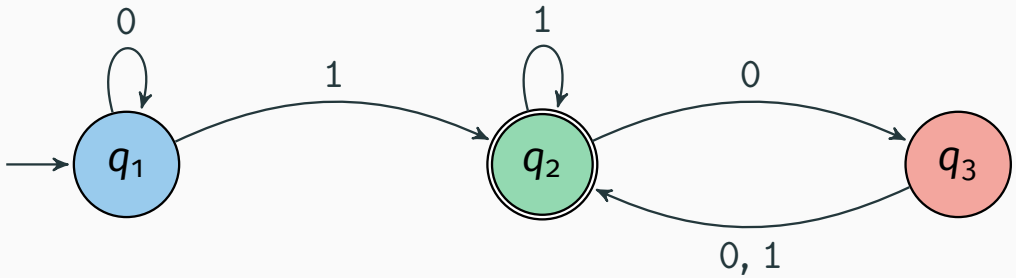
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- $q_0 = q_1$.
- $F = \{q_2\}$.

Question

What is $L_{\mathcal{A}}$?

An automaton reading binary strings



For this automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$:

- $Q = \{q_1, q_2, q_3\}$.
- $\Sigma = \{0, 1\}$.
- $\delta =$

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- $q_0 = q_1$.
- $F = \{q_2\}$.

Question

What is $L_{\mathcal{A}}$?

$L_{\mathcal{A}} = \{w \in \Sigma^* : w$ contains a 1, and an even number of 0's after its last 1 $\}$.

Regular languages and non-determinism

Regular languages

- A language L is called regular if there exists a DFA that recognizes it.

Regular languages

Examples

- The language $\{w \in \Sigma^* : w$ contains a 1, and an even number of 0's after its first 1 $\}$.

Regular languages

Examples

- The language $\{w \in \Sigma^* : w$ contains a 1, and an even number of 0's after its first 1 $\}$.
- Any finite language.

Regular languages

Examples

- The language $\{w \in \Sigma^* : w$ contains a 1, and an even number of 0's after its first 1 $\}$.
- Any finite language.
- The language of words containing a particular substring.

Regular languages

Examples

- The language $\{w \in \Sigma^* : w$ contains a 1, and an even number of 0's after its first 1 $\}$.
- Any finite language.
- The language of words containing a particular substring.
- The language of 'valid passwords': alphanumeric strings containing at least one lower case, one upper case, and one number.

Regular languages

Examples

- The language $\{w \in \Sigma^* : w$ contains a 1, and an even number of 0's after its first 1 $\}$.
- Any finite language.
- The language of words containing a particular substring.
- The language of 'valid passwords': alphanumeric strings containing at least one lower case, one upper case, and one number.
- For any regular expression R, the set of strings which match R.

A nonregular language

Proposition

The following language L is not regular:

$$L := \{w \in \{0,1\}^* \ : \ w \text{ contains as many } 0\text{'s as } 1\text{'s}\}$$

A nonregular language

Proposition

The following language L is not regular:

$$L := \{w \in \{0,1\}^* \ : \ w \text{ contains as many } 0\text{'s as } 1\text{'s}\}$$

Proof.

Suppose, towards a contradiction, that $\mathcal{A}$ were a DFA recognizing this language L. Let p be the number of states in $\mathcal{A}$. When $\mathcal{A}$ reads the word 0^p1^p, there must be a state q which is visited twice in the first half of the computation. (Why?)

A nonregular language

Proposition

The following language L is not regular:

$$L := \{w \in \{0,1\}^* \ : \ w \text{ contains as many } 0\text{'s as } 1\text{'s}\}$$

Proof.

Suppose, towards a contradiction, that $\mathcal{A}$ were a DFA recognizing this language L. Let p be the number of states in $\mathcal{A}$. When $\mathcal{A}$ reads the word 0^p1^p, there must be a state q which is visited twice in the first half of the computation. (Why?) Say $\mathcal{A}$ is in state q both after reading 0^k and also after reading 0^{k+r}, where $0 \leq k < p$ and $r > 0$. But then, when $\mathcal{A}$ reads $0^{p+r}1^p$, it will end up in the same last state as when it reads 0^p1^p. This is impossible. (Why?) □

Definition

Let L be a language. A number $p > 0$ is called a pumping length for L if any word $w \in L$ can be factored into three pieces, $w = axb$, with $x \neq \epsilon$ and ax of length $\leq p$, such that $ax^n b \in L$ for all n.

Nonregular languages and the pumping lemma

Definition

Let L be a language. A number $p > 0$ is called a pumping length for L if any word $w \in L$ can be factored into three pieces, $w = axb$, with $x \neq \epsilon$ and ax of length $\leq p$, such that $ax^n b \in L$ for all n.

Lemma (Pumping Lemma)

Any regular language has a pumping length.

Nonregular languages and the pumping lemma

Definition

Let L be a language. A number $p > 0$ is called a pumping length for L if any word $w \in L$ can be factored into three pieces, $w = axb$, with $x \neq \epsilon$ and ax of length $\leq p$, such that $ax^n b \in L$ for all n.

Lemma (Pumping Lemma)

Any regular language has a pumping length.

This lemma can be used to show that a language is *not* regular: exhibit, for every number $p > 0$, a word w_p in the language that can not be 'pumped'.

Operations on languages

Let Σ be a finite alphabet, and let L_1 and L_2 be Σ-languages.

- The union of L_1 and L_2 is the language

$$L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}.$$

Operations on languages

Let Σ be a finite alphabet, and let L_1 and L_2 be Σ-languages.

- The union of L_1 and L_2 is the language

$$L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}.$$

- The intersection of L_1 and L_2 is the language

$$L_1 \cap L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}.$$

Operations on languages

Let Σ be a finite alphabet, and let L_1 and L_2 be Σ-languages.

- The union of L_1 and L_2 is the language

$$L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}.$$

- The intersection of L_1 and L_2 is the language

$$L_1 \cap L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}.$$

- The complement of L_1 is the language

$$\Sigma^* \setminus L_1 := \{w \in \Sigma^* \mid \text{ not } w \in L_1\}.$$

Operations on languages

Let Σ be a finite alphabet, and let L_1 and L_2 be Σ-languages.

- The union of L_1 and L_2 is the language

$$L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}.$$

- The intersection of L_1 and L_2 is the language

$$L_1 \cap L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}.$$

- The complement of L_1 is the language

$$\Sigma^* \setminus L_1 := \{w \in \Sigma^* \mid \text{ not } w \in L_1\}.$$

- The concatenation of L_1 and L_2 is the language

$$L_1 \circ L_2 := \{w \in \Sigma^* \mid \text{ there exist } w_1 \in L_1 \text{ and } w_2 \in L_2 \text{ such that } w = w_1 w_2\}.$$

Operations on languages

Let Σ be a finite alphabet, and let L_1 and L_2 be Σ-languages.

- The union of L_1 and L_2 is the language

$$L_1 \cup L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}.$$

- The intersection of L_1 and L_2 is the language

$$L_1 \cap L_2 := \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}.$$

- The complement of L_1 is the language

$$\Sigma^* \setminus L_1 := \{w \in \Sigma^* \mid \text{ not } w \in L_1\}.$$

- The concatenation of L_1 and L_2 is the language

$$L_1 \circ L_2 := \{w \in \Sigma^* \mid \text{ there exist } w_1 \in L_1 \text{ and } w_2 \in L_2 \text{ such that } w = w_1 w_2\}.$$

Question. Suppose L_1 and L_2 are regular. Are $L_1 \cup L_2$, $L_1 \cap L_2$, $\Sigma^* \setminus L_1$, and $L_1 \circ L_2$ also regular?

Union of regular languages

Proposition

The union of two regular languages is regular.

Union of regular languages

Proposition

The union of two regular languages is regular.

Proof (sketch).

Let L_1 and L_2 be regular Σ-languages. Choose two DFA's, $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognizing L_1 and L_2, respectively.

Union of regular languages

Proposition

The union of two regular languages is regular.

Proof (sketch).

Let L_1 and L_2 be regular Σ-languages. Choose two DFA's, $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognizing L_1 and L_2, respectively. We define a new automaton, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ which recognizes the languages $L_1 \cup L_2$:

Union of regular languages

Proposition

The union of two regular languages is regular.

Proof (sketch).

Let L_1 and L_2 be regular Σ-languages. Choose two DFA's, $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognizing L_1 and L_2, respectively. We define a new automaton, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ which recognizes the languages $L_1 \cup L_2$:

- $Q = Q_1 \times Q_2$;
- $\delta(\langle q_1, q_2 \rangle, a) := \langle \delta_1(q_1, a), \delta_2(q_2, a) \rangle$;
- $q_0 := \langle q_1, q_2 \rangle$;
- $F := (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

Exercise. Check that, indeed, $L_{\mathcal{A}} = L_1 \cup L_2$. □

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (attempt).

Let L_1 and L_2 be regular Σ-languages. Choose two DFA's, $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognizing L_1 and L_2, respectively. We define a new automaton, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ which recognizes the languages $L_1 \circ L_2$:

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (attempt).

Let L_1 and L_2 be regular Σ-languages. Choose two DFA's, $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognizing L_1 and L_2, respectively. We define a new automaton, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ which recognizes the languages $L_1 \circ L_2$:

?

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (attempt).

Let L_1 and L_2 be regular Σ-languages. Choose two DFA's, $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, recognizing L_1 and L_2, respectively. We define a new automaton, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ which recognizes the languages $L_1 \circ L_2$:

?

We need non-determinism.

Example of a non-deterministic finite automaton

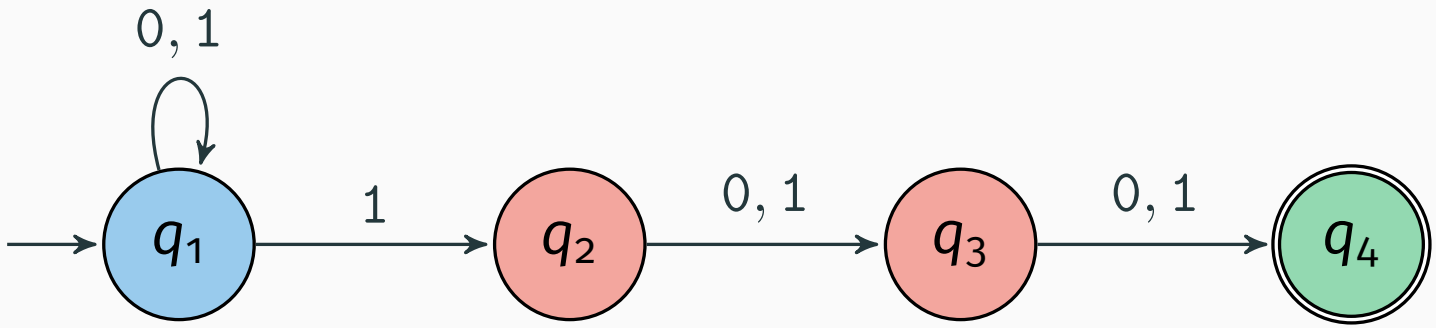
- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

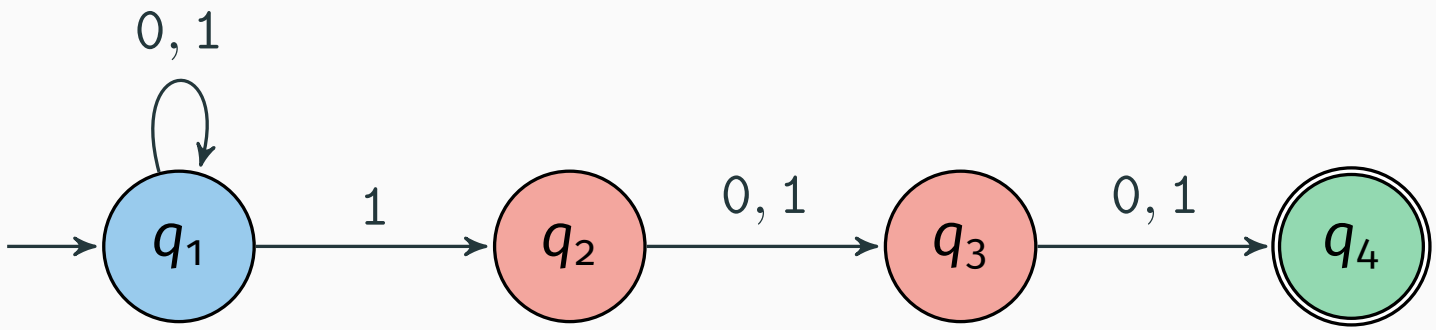
> Is there an automaton that recognizes L?



Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

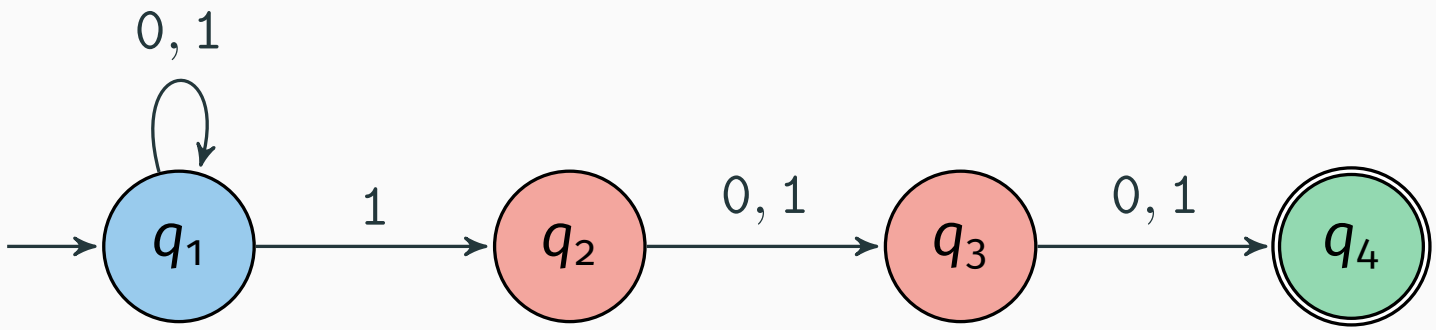


Input	Computations	Last state	Accepted?

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

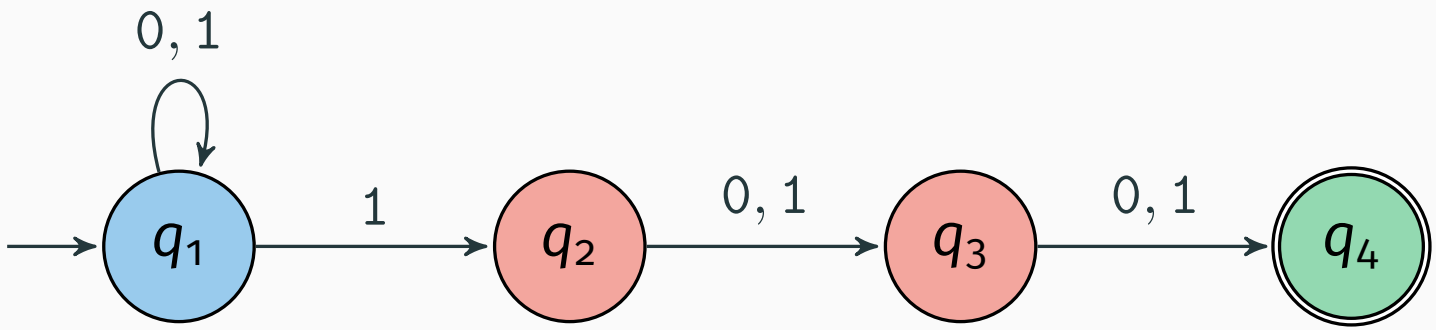


Input	Computations	Last state	Accepted?
000100			

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

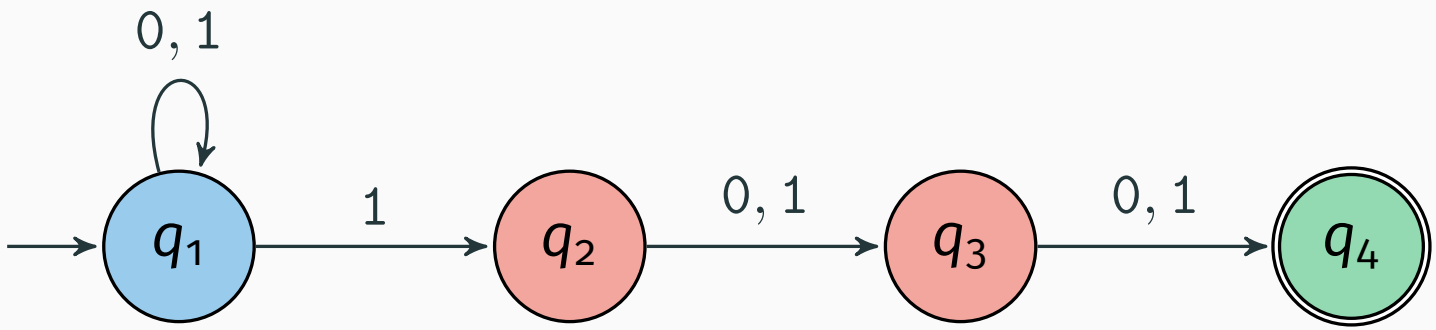


Input	Computations	Last state	Accepted?
000100	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

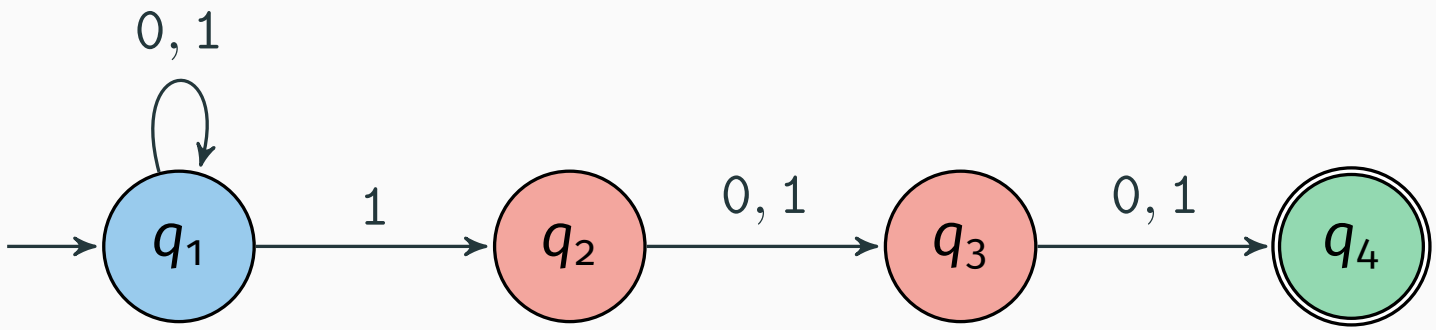


Input	Computations	Last state	Accepted?
000100	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes
0000			

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

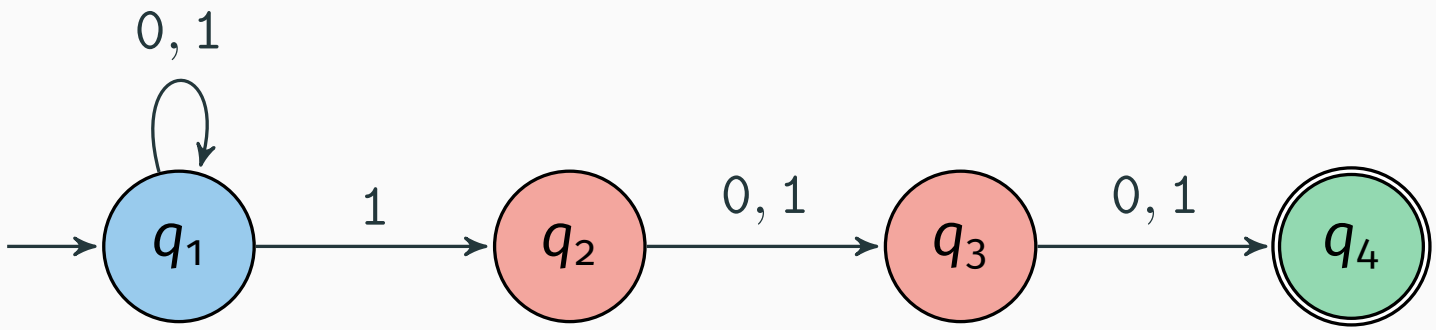


Input	Computations	Last state	Accepted?
000100	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes
0000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

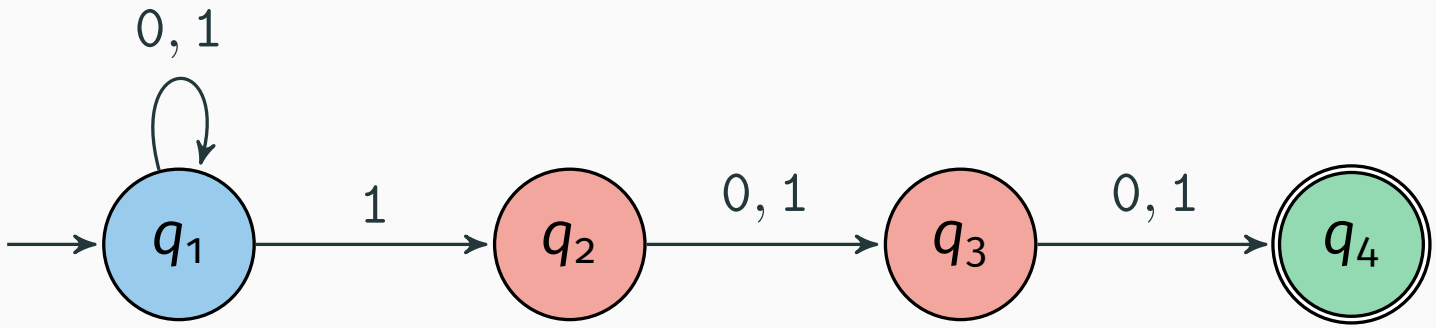


Input	Computations	Last state	Accepted?
000100	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes
0000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
1100			

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?

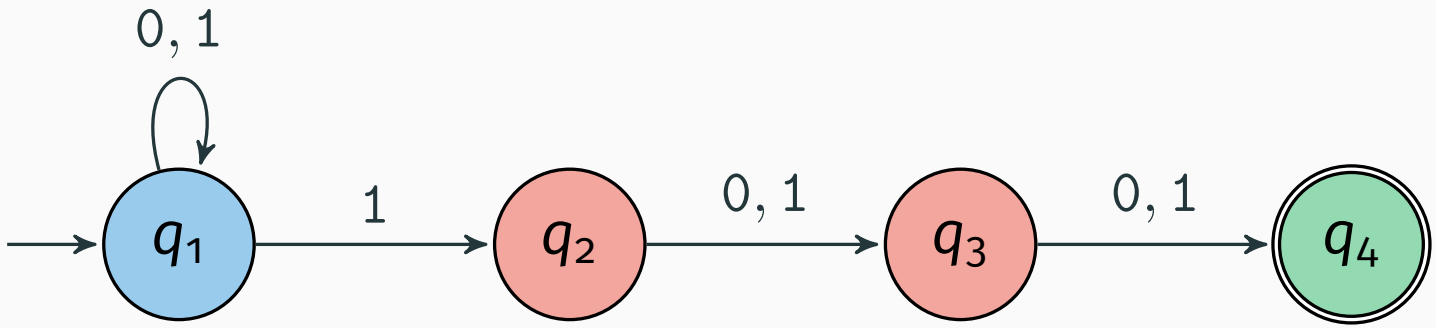


Input	Computations	Last state	Accepted?
000100	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes
0000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
1100	$q_1{}^1 q_2{}^1 q_3{}^0 q_4{}^0$??	??	

Example of a non-deterministic finite automaton

- Consider the language $L := \{w \in \{0,1\}^* :$ the third position from the end of w exists, and is a $1\}$.

> Is there an automaton that recognizes L?



Input	Computations	Last state	Accepted?
000100	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes
0000	$q_1{}^0 q_1{}^0 q_1{}^0 q_1{}^0 q_1$	q_1	No
1100	$q_1{}^1 q_2{}^1 q_3{}^0 q_4{}^0$??	??	
	$q_1{}^1 q_1{}^1 q_2{}^0 q_3{}^0 q_4$	q_4	Yes

Definition of non-deterministic finite automata

- A non-deterministic finite automaton (NFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \colon Q \times \Sigma_{(\epsilon)} \to \mathcal{P}(Q)$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.

Definition of non-deterministic finite automata

- A non-deterministic finite automaton (NFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \colon Q \times \Sigma_{(\epsilon)} \to \mathcal{P}(Q)$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.
- $q \xrightarrow{a} q'$ means: q is a state in Q, a is a letter in Σ, and $q' \in \delta(q, a)$.

Definition of non-deterministic finite automata

- A non-deterministic finite automaton (NFA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - $\delta \colon Q \times \Sigma_{(\epsilon)} \to \mathcal{P}(Q)$ is a transition function,
  - q_0 is an element of Q, the start state,
  - F is a subset of Q, the set of accepting states.
- $q \xrightarrow{a} q'$ means: q is a state in Q, a is a letter in Σ, and $q' \in \delta(q, a)$.
- Let $w = a_1 \dots a_n \in \Sigma^*$ be a Σ-word of length n. Then:
  - a computation of an automaton $\mathcal{A}$ on w is a sequence $q_0 \dots q_n$ of states such that $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$;
  - a computation is accepting if the last state, q_n, is in F;
  - $\mathcal{A}$ accepts w if it has some accepting computation on w.
- The language recognized by $\mathcal{A}$ is $L_{\mathcal{A}} := \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$.

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (2nd attempt).

Given two DFA's $\mathcal{A}_1$ and $\mathcal{A}_2$, build one NFA $\mathcal{A}$ by allowing transitions from any final state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$. □

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (2nd attempt).

Given two DFA's $\mathcal{A}_1$ and $\mathcal{A}_2$, build one NFA $\mathcal{A}$ by allowing transitions from any final state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$. □

But, two issues:

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (2nd attempt).

Given two DFA's $\mathcal{A}_1$ and $\mathcal{A}_2$, build one NFA $\mathcal{A}$ by allowing transitions from any final state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$. □

But, two issues:

- What should we label these transitions with?

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (2nd attempt).

Given two DFA's $\mathcal{A}_1$ and $\mathcal{A}_2$, build one NFA $\mathcal{A}$ by allowing transitions from any final state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$. □

But, two issues:

- What should we label these transitions with? $\rightarrow$ 'ϵ'

Concatenation of regular languages

Proposition

The concatenation of two regular languages is regular.

Proof (2nd attempt).

Given two DFA's $\mathcal{A}_1$ and $\mathcal{A}_2$, build one NFA $\mathcal{A}$ by allowing transitions from any final state of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$. □

But, two issues:

- What should we label these transitions with? $\rightarrow$ 'ϵ'
- Can we now conclude the concatenation of the two languages is regular?

Determinization

Proposition

Any non-deterministic finite automaton can be simulated by a deterministic finite automaton.

Determinization

Proposition

Any non-deterministic finite automaton can be simulated by a deterministic finite automaton.

Proof (sketch).

Given an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, we define a DFA $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ which simulates $\mathcal{A}$, as follows:

- $Q' := \mathcal{P}(Q)$,
- $\delta'(U, a) := \{q' \in Q \mid \text{ there is } q \in U \text{ with } q' \in \delta(q, a)\}$,
- $q_0' := \{q_0\}$,
- $F' := \{U \in Q' \mid \text{ there is } q_f \in U \text{ with } q_f \in F\}$.

Exercise. Check that $\mathcal{A}'$ is deterministic, and $L_{\mathcal{A}'} = L_{\mathcal{A}}$. □

Theorem

A language L can be recognized by an NFA if, and only if, L can be recognized by a DFA.

Monadic Second Order Logic

Logic as a descriptive formalism

- So far, we described languages informally, and then gave implementations for them, using automata.

Logic as a descriptive formalism

- So far, we described languages informally, and then gave implementations for them, using automata.
- We will now use logic to give formal descriptions of (regular) languages.

Logic as a descriptive formalism

- So far, we described languages informally, and then gave implementations for them, using automata.
- We will now use logic to give formal descriptions of (regular) languages.
- We will show that being able to give a logic description of a language is equivalent to being able to give an automaton implementation.

Logic as a descriptive formalism

- So far, we described languages informally, and then gave implementations for them, using automata.
- We will now use logic to give formal descriptions of (regular) languages.
- We will show that being able to give a logic description of a language is equivalent to being able to give an automaton implementation.
- This use of logic is a bit different from its applications as a formalization of ‘reasoning’.

Logic as a descriptive formalism

- So far, we described languages informally, and then gave implementations for them, using automata.
- We will now use logic to give formal descriptions of (regular) languages.
- We will show that being able to give a logic description of a language is equivalent to being able to give an automaton implementation.
- This use of logic is a bit different from its applications as a formalization of ‘reasoning’.
- Expressibility in a logic = Computability by a machine.

Logic on words

- Let $w = a_1 \dots a_n$ be a Σ-word.
- For any number $i \in \{1, \dots, n\}$ and letter $a \in \Sigma$, we say a letter predicate $\mathtt{a}$ is true at i in w if $a_i = a$.

Logic on words

- Let $w = a_1 \dots a_n$ be a Σ-word.
- For any number $i \in \{1, \dots, n\}$ and letter $a \in \Sigma$, we say a letter predicate $\mathtt{a}$ is true at i in w if $a_i = a$.
- For example, let $w = 10010$.

> Where in w is the letter predicate $\mathtt{0}$ true?

Logic on words

- Let $w = a_1 \dots a_n$ be a Σ-word.
- For any number $i \in \{1, \dots, n\}$ and letter $a \in \Sigma$, we say a letter predicate $\mathtt{a}$ is true at i in w if $a_i = a$.
- For example, let $w = 10010$.

> Where in w is the letter predicate $\mathtt{0}$ true?

- We may now use logic formulas to describe properties of words (with valuations).

Logic on words

- Let $w = a_1 \dots a_n$ be a Σ-word.
- For any number $i \in \{1, \dots, n\}$ and letter $a \in \Sigma$, we say a letter predicate $\mathtt{a}$ is true at i in w if $a_i = a$.
- For example, let $w = 10010$.

> Where in w is the letter predicate $\mathtt{0}$ true?

- We may now use logic formulas to describe properties of words (with valuations).
- For example, the formula $\mathtt{0}(x) \wedge \exists y(y > x \wedge \mathtt{1}(y))$ says:

Logic on words

- Let $w = a_1 \dots a_n$ be a Σ-word.
- For any number $i \in \{1, \dots, n\}$ and letter $a \in \Sigma$, we say a letter predicate $\mathtt{a}$ is true at i in w if $a_i = a$.
- For example, let $w = 10010$.

> Where in w is the letter predicate $\mathtt{0}$ true?

- We may now use logic formulas to describe properties of words (with valuations).
- For example, the formula $\mathtt{0}(x) \wedge \exists y(y > x \wedge \mathtt{1}(y))$ says: 'there is a 0 at the position of variable x and a 1 at some later position.'

> Where in w is this formula true?

Examples of logic descriptions of languages

> $L = \{w \in \{0,1\}^* \ : \ \text{the string } 11 \text{ is a substring of } w\}$.

Examples of logic descriptions of languages

> $L = \{w \in \{0,1\}^* \ : \ \text{the string } 11 \text{ is a substring of } w\}$.

- Logic description:

$$\exists x \exists y [\mathtt{S}(x,y) \wedge \mathtt{1}(x) \wedge \mathtt{1}(y)].$$

Examples of logic descriptions of languages

> $L = \{w \in \{0,1\}^* \ : \ \text{the string } 11 \text{ is a substring of } w\}$.

- Logic description:

$$\exists x \exists y [\mathtt{S}(x,y) \wedge \mathtt{1}(x) \wedge \mathtt{1}(y)].$$

> $L = \{w \in \{\texttt{front}, \texttt{rear}, \texttt{neither}, \texttt{both}\}^* \ :$
> there is a '$\texttt{neither}$' after every occurrence of '$\texttt{front}$' in $w\}$.

Examples of logic descriptions of languages

> $L = \{w \in \{0,1\}^* \ : \ \text{the string } 11 \text{ is a substring of } w\}$.

- Logic description:

$$\exists x \exists y [\mathtt{S}(x,y) \wedge \mathtt{1}(x) \wedge \mathtt{1}(y)].$$

> $L = \{w \in \{\mathtt{front}, \mathtt{rear}, \mathtt{neither}, \mathtt{both}\}^* \ :$
> there is a '$\mathtt{neither}$' after every occurrence of '$\mathtt{front}$' in $w\}$.

- Logic description:

$$\forall x [\mathtt{front}(x) \rightarrow \exists y (y > x \wedge \mathtt{neither}(y))].$$

Examples of logic descriptions of languages

> $L = \{w \in \{0,1\}^* \ : \ \text{the string } 11 \text{ is a substring of } w\}$.

- Logic description:

$$\exists x \exists y [\mathtt{S}(x,y) \wedge \mathtt{1}(x) \wedge \mathtt{1}(y)].$$

> $L = \{w \in \{\mathtt{front}, \mathtt{rear}, \mathtt{neither}, \mathtt{both}\}^* \ :$
> there is a '$\mathtt{neither}$' after every occurrence of '$\mathtt{front}$' in $w\}$.

- Logic description:

$$\forall x [\mathtt{front}(x) \rightarrow \exists y (y > x \wedge \mathtt{neither}(y))].$$

> $L = \{w \in \{0,1\}^* \ : \ \text{the length of } w \text{ is even}\}$.

Examples of logic descriptions of languages

> $L = \{w \in \{0,1\}^* \ : \ \text{the string } 11 \text{ is a substring of } w\}$.

- Logic description:

$$\exists x \exists y [\mathtt{S}(x,y) \wedge \mathtt{1}(x) \wedge \mathtt{1}(y)].$$

> $L = \{w \in \{\mathtt{front}, \mathtt{rear}, \mathtt{neither}, \mathtt{both}\}^* \ :$
> there is a '$\mathtt{neither}$' after every occurrence of '$\mathtt{front}$' in $w\}$.

- Logic description:

$$\forall x [\mathtt{front}(x) \rightarrow \exists y (y > x \wedge \mathtt{neither}(y))].$$

> $L = \{w \in \{0,1\}^* \ : \ \text{the length of } w \text{ is even}\}$.

- Logic description:

$$\exists P [P(\mathtt{first}) \wedge \neg P(\mathtt{last}) \wedge \forall x \forall y \, \mathtt{S}(x,y) \rightarrow (P(x) \leftrightarrow \neg P(y))] \vee \mathtt{empty}.$$

Monadic Second Order Logic: syntax

- Basic propositional connectives: $\vee$, $\wedge$, $\neg$, $\ldots$;
- Quantification over first-order variables x, y, $\ldots$ and monadic second-order variables P, Q, $\ldots$;
- Atomic formulas: $x < y$, $P \subseteq Q$, $P(x)$, $\mathrm{a}(x)$ for $a \in \Sigma$.

Monadic Second Order Logic: syntax

- Basic propositional connectives: $\vee$, $\wedge$, $\neg$, $\ldots$;
- Quantification over first-order variables x, y, $\ldots$ and monadic second-order variables P, Q, $\ldots$;
- Atomic formulas: $x < y$, $P \subseteq Q$, $P(x)$, $\mathrm{a}(x)$ for $a \in \Sigma$.

The syntax for First Order (FO) Logic is obtained by disallowing second order variables and second order quantifiers.

Monadic Second Order Logic: semantics

- View a word $w = a_1 \dots a_n$ as a *structure* W, i.e.,

Monadic Second Order Logic: semantics

- View a word $w = a_1 \dots a_n$ as a *structure* W, i.e.,
  - The underlying set of W is $|\mathbf{w}| = \{1, \dots, n\}$.
  - The natural linear order $<^W$ interprets the binary predicate $<$.
  - For every letter $a \in \Sigma$, $\mathtt{a}^W$ is the set of positions i where $a_i = a$.

Monadic Second Order Logic: semantics

- View a word $w = a_1 \dots a_n$ as a *structure* W, i.e.,
  - The underlying set of W is $|\mathbf{w}| = \{1, \dots, n\}$.
  - The natural linear order $<^W$ interprets the binary predicate $<$.
  - For every letter $a \in \Sigma$, $\mathtt{a}^W$ is the set of positions i where $a_i = a$.
- Inductively define the satisfaction relation, $\models$, in the usual way, where each monadic second-order variable is interpreted as a subset of $\{1, \dots, n\}$.

Monadic Second Order Logic: semantics

- View a word $w = a_1 \dots a_n$ as a *structure* W, i.e.,
  - The underlying set of W is $|\mathbf{w}| = \{1, \dots, n\}$.
  - The natural linear order $<^W$ interprets the binary predicate $<$.
  - For every letter $a \in \Sigma$, $\mathtt{a}^W$ is the set of positions i where $a_i = a$.
- Inductively define the satisfaction relation, $\models$, in the usual way, where each monadic second-order variable is interpreted as a subset of $\{1, \dots, n\}$.
- Thus, given a word w, a formula φ with free first- and second-order variables in a set $\mathrm{Var} = \mathrm{Var}_1 \cup \mathrm{Var}_2$, and a valuation V of Var in $|\mathbf{w}|$, we write: $W, V \models \varphi$.

Monadic Second Order Logic: semantics

- View a word $w = a_1 \dots a_n$ as a *structure* W, i.e.,
  - The underlying set of W is $|\mathbf{w}| = \{1, \dots, n\}$.
  - The natural linear order $<^W$ interprets the binary predicate $<$.
  - For every letter $a \in \Sigma$, $\mathtt{a}^W$ is the set of positions i where $a_i = a$.
- Inductively define the satisfaction relation, $\models$, in the usual way, where each monadic second-order variable is interpreted as a subset of $\{1, \dots, n\}$.
- Thus, given a word w, a formula φ with free first- and second-order variables in a set $\mathrm{Var} = \mathrm{Var}_1 \cup \mathrm{Var}_2$, and a valuation V of Var in $|\mathbf{w}|$, we write: $W, V \models \varphi$.
- In particular, any MSO-*sentence* φ defines a language:

$$L_\varphi := \{w \in \Sigma^* \mid W \models \varphi\}.$$

Monadic Second Order Logic: semantics

- View a word $w = a_1 \dots a_n$ as a *structure* W, i.e.,
  - The underlying set of W is $|\mathbf{w}| = \{1, \dots, n\}$.
  - The natural linear order $<^W$ interprets the binary predicate $<$.
  - For every letter $a \in \Sigma$, $\mathtt{a}^W$ is the set of positions i where $a_i = a$.
- Inductively define the satisfaction relation, $\models$, in the usual way, where each monadic second-order variable is interpreted as a subset of $\{1, \dots, n\}$.
- Thus, given a word w, a formula φ with free first- and second-order variables in a set $\mathrm{Var} = \mathrm{Var}_1 \cup \mathrm{Var}_2$, and a valuation V of Var in $|\mathbf{w}|$, we write: $W, V \models \varphi$.
- In particular, any MSO-*sentence* φ defines a language:

$$L_\varphi := \{w \in \Sigma^* \mid W \models \varphi\}.$$

- **Exercise.** Shortcuts such as `first`, `last`, `empty`, $\mathtt{S}(x, y)$ ('immediate successor'), ... are definable.

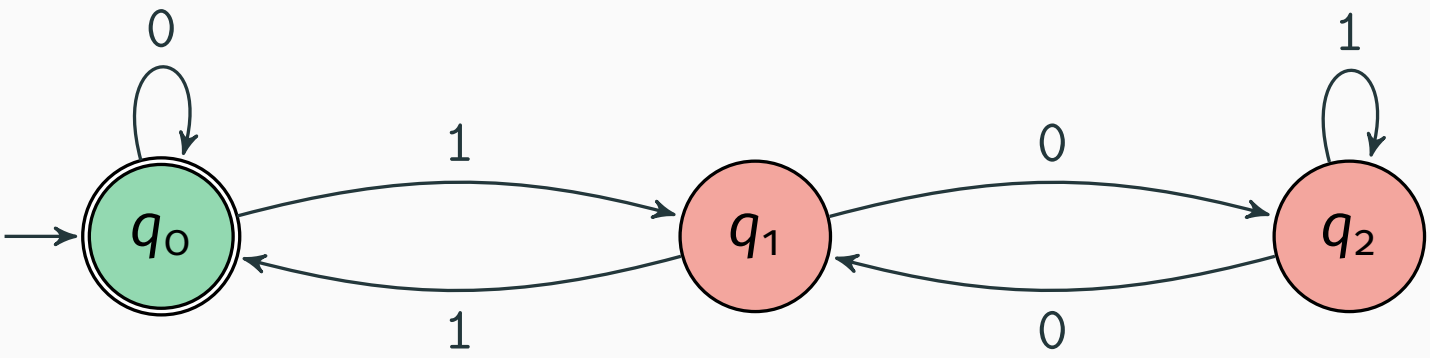
The NFA-MSO Connection

Theorem (Büchi, 1960)

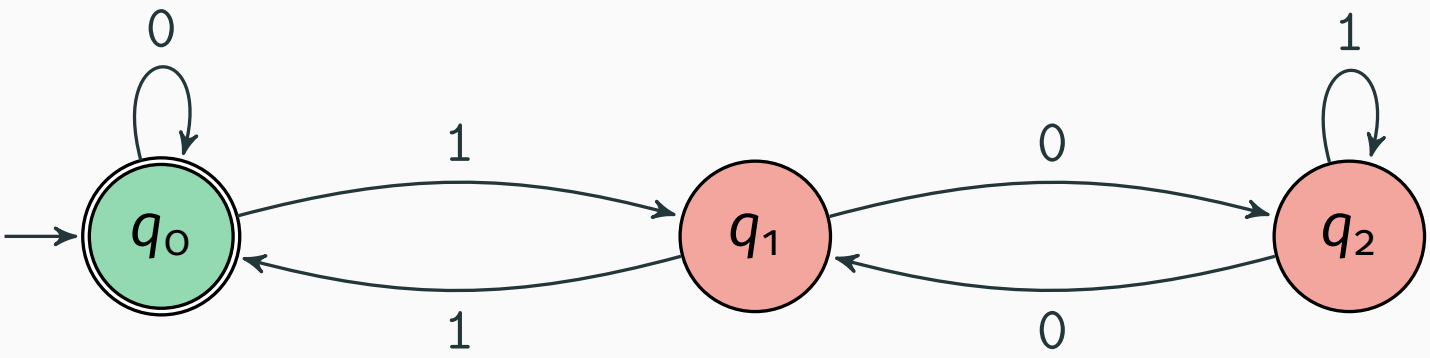
A language is regular if, and only if, it is definable in monadic second order logic:

$$REG = MSO.$$

From an automaton to an MSO-sentence, example

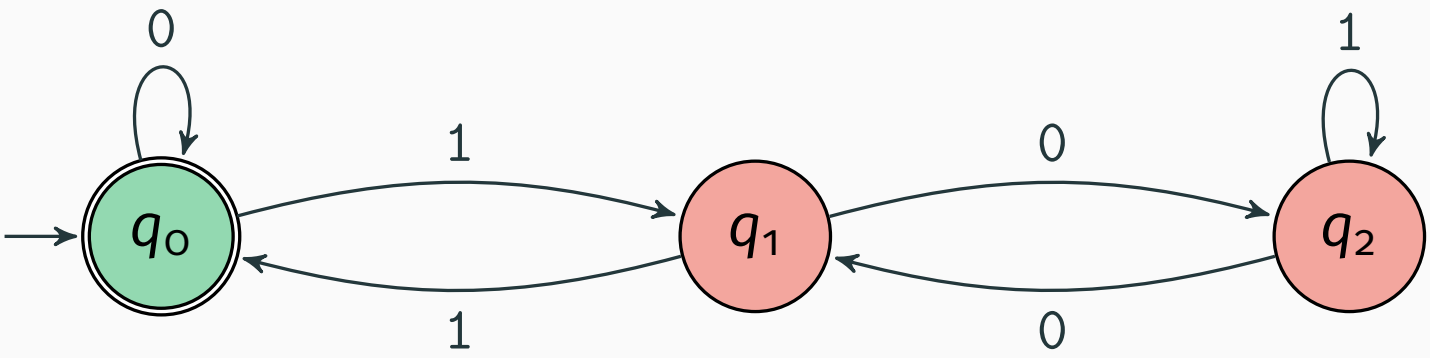


From an automaton to an MSO-sentence, example



> What is the language recognized by this automaton?

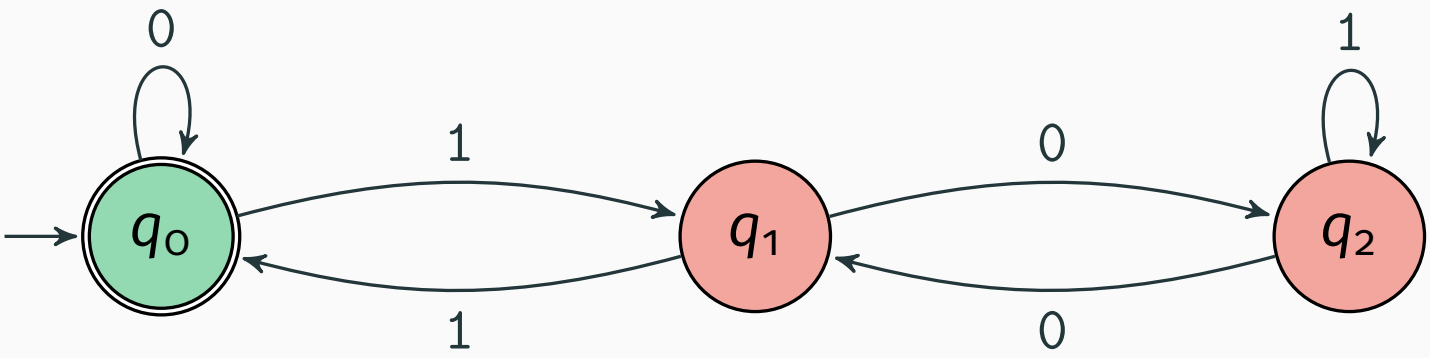
From an automaton to an MSO-sentence, example



> What is the language recognized by this automaton?

- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.

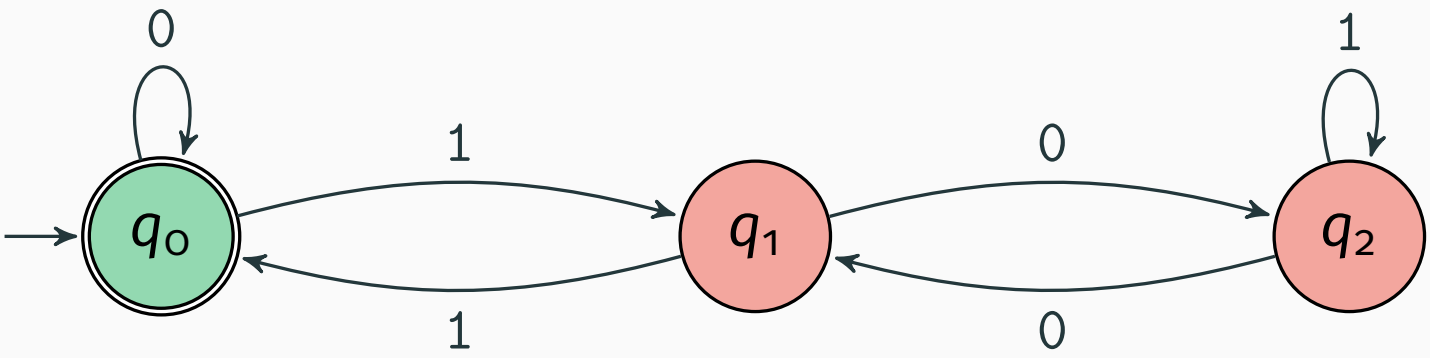
From an automaton to an MSO-sentence, example



> What is the language recognized by this automaton?

- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.
- An MSO sentence describing the behavior of $\mathcal{A}$:

From an automaton to an MSO-sentence, example

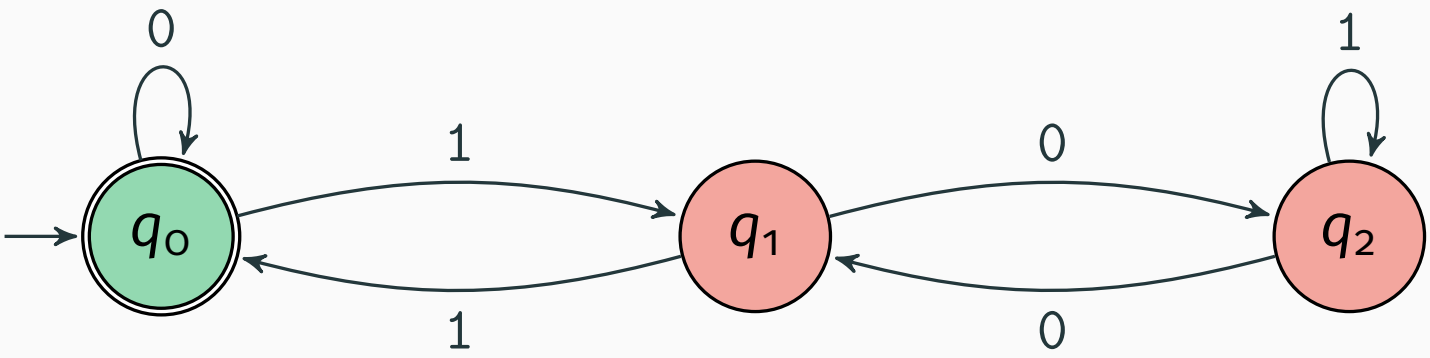


> What is the language recognized by this automaton?

- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.
- An MSO sentence describing the behavior of $\mathcal{A}$:

$$\exists Q_0 \exists Q_1 \exists Q_2 \Big[\qquad \wedge \qquad \wedge \qquad \Big].$$

From an automaton to an MSO-sentence, example



> What is the language recognized by this automaton?

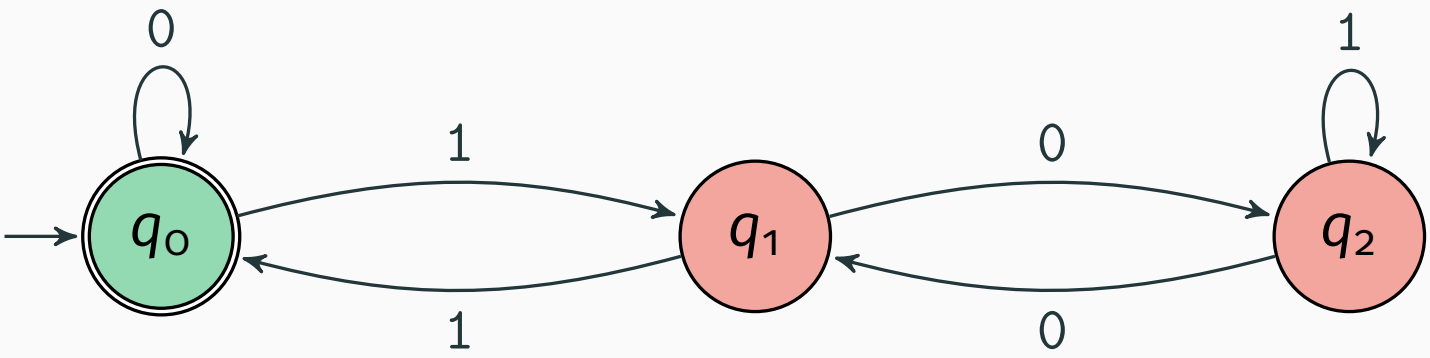
- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.
- An MSO sentence describing the behavior of $\mathcal{A}$:

$$\exists Q_0 \exists Q_1 \exists Q_2 \Big[Q_0(\texttt{first}) \wedge Q_0(\texttt{last}) \wedge$$

$$\wedge$$

$$\Big].$$

From an automaton to an MSO-sentence, example

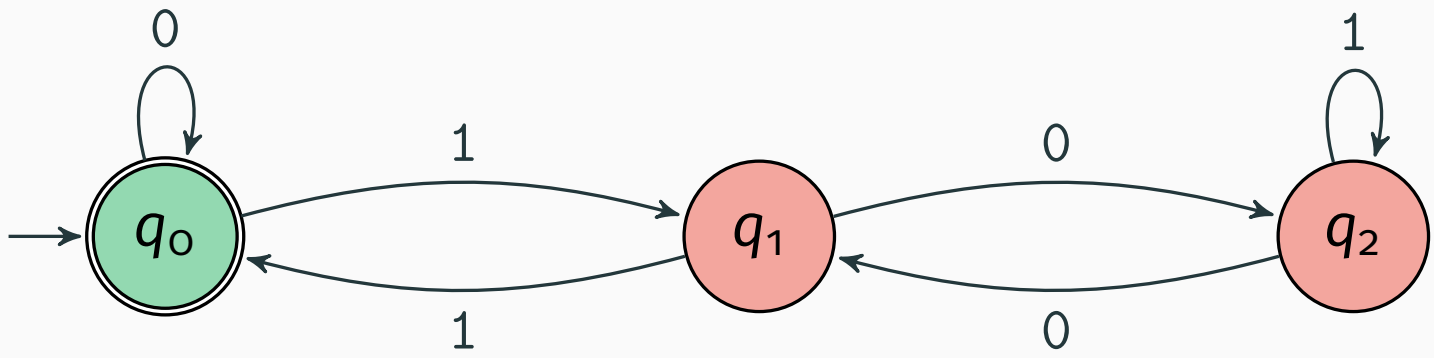


> What is the language recognized by this automaton?

- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.
- An MSO sentence describing the behavior of $\mathcal{A}$:

$$\exists Q_0 \exists Q_1 \exists Q_2 \Big[Q_0(\texttt{first}) \wedge Q_0(\texttt{last}) \wedge$$
$$\forall x[(Q_0(x) \vee Q_1(x) \vee Q_2(x)) \wedge \neg(Q_0(x) \wedge Q_1(x)) \wedge \ldots)] \wedge$$
$$\Big].$$

From an automaton to an MSO-sentence, example

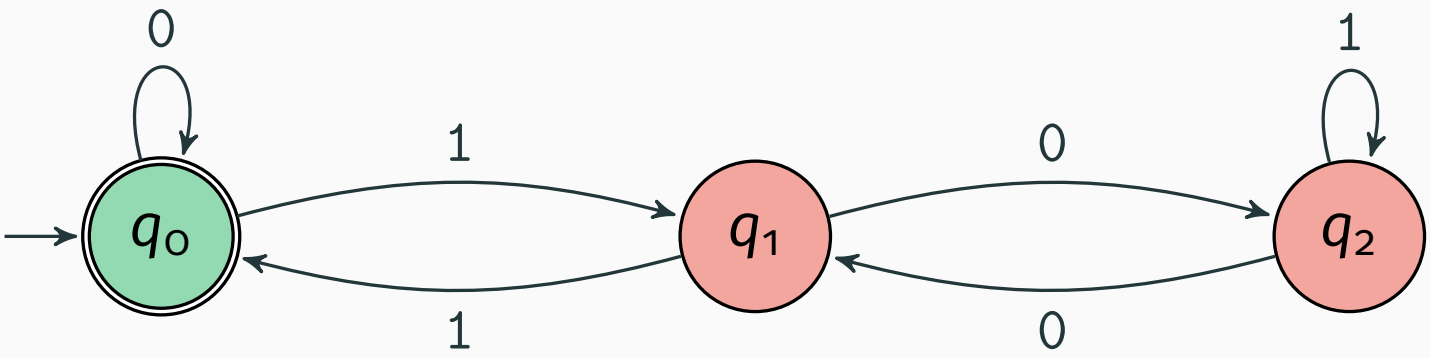


> What is the language recognized by this automaton?

- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.
- An MSO sentence describing the behavior of $\mathcal{A}$:

$$\exists Q_0 \exists Q_1 \exists Q_2 \Big[Q_0(\texttt{first}) \wedge Q_0(\texttt{last}) \wedge$$
$$\forall x[(Q_0(x) \vee Q_1(x) \vee Q_2(x)) \wedge \neg(Q_0(x) \wedge Q_1(x)) \wedge \ldots)] \wedge$$
$$\forall x[0(x) \wedge Q_0(x) \rightarrow Q_0(\mathrm{S}x)] \qquad \Big].$$

From an automaton to an MSO-sentence, example



> What is the language recognized by this automaton?

- The $\{0, 1\}$-words which represent a number in binary that is divisible by 3.
- An MSO sentence describing the behavior of $\mathcal{A}$:

$$\exists Q_0 \exists Q_1 \exists Q_2 \big[Q_0(\texttt{first}) \wedge Q_0(\texttt{last}) \wedge$$
$$\forall x[(Q_0(x) \vee Q_1(x) \vee Q_2(x)) \wedge \neg(Q_0(x) \wedge Q_1(x)) \wedge \ldots)] \wedge$$
$$\forall x[0(x) \wedge Q_0(x) \rightarrow Q_0(\mathrm{S}x)] \wedge [1(x) \wedge Q_0(x) \rightarrow Q_1(\mathrm{S}x)] \wedge \ldots \big].$$

From an automaton to an MSO-sentence

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA.
- We describe an MSO-sentence, $\varphi_{\mathcal{A}}$, which defines the regular language $L_{\mathcal{A}}$.

From an automaton to an MSO-sentence

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA.
- We describe an MSO-sentence, $\varphi_{\mathcal{A}}$, which defines the regular language $L_{\mathcal{A}}$.
- $\varphi_{\mathcal{A}}$ is the sentence $\exists Q_1 \dots Q_k(\varphi_{boundary} \wedge \varphi_{part} \wedge \varphi_{trans})$, where

From an automaton to an MSO-sentence

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA.
- We describe an MSO-sentence, $\varphi_{\mathcal{A}}$, which defines the regular language $L_{\mathcal{A}}$.
- $\varphi_{\mathcal{A}}$ is the sentence $\exists Q_1 \dots Q_k(\varphi_{boundary} \wedge \varphi_{part} \wedge \varphi_{trans})$, where
  - k is the number of states of $\mathcal{A}$;

From an automaton to an MSO-sentence

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA.
- We describe an MSO-sentence, $\varphi_{\mathcal{A}}$, which defines the regular language $L_{\mathcal{A}}$.
- $\varphi_{\mathcal{A}}$ is the sentence $\exists Q_1 \dots Q_k(\varphi_{boundary} \wedge \varphi_{part} \wedge \varphi_{trans})$, where
  - k is the number of states of $\mathcal{A}$;
  - $\varphi_{boundary}$ is the formula $Q_1(\texttt{first}) \wedge \bigvee_{q_i \in F} Q_i(\texttt{last})$;
  - φ_{part} is the formula $\forall x \left[\bigvee_{i=1}^{k}(Q_i(x) \wedge \bigwedge_{j \neq i} Q_j(x)\right]$;

From an automaton to an MSO-sentence

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA.
- We describe an MSO-sentence, $\varphi_{\mathcal{A}}$, which defines the regular language $L_{\mathcal{A}}$.
- $\varphi_{\mathcal{A}}$ is the sentence $\exists Q_1 \dots Q_k(\varphi_{boundary} \wedge \varphi_{part} \wedge \varphi_{trans})$, where
  - k is the number of states of $\mathcal{A}$;
  - $\varphi_{boundary}$ is the formula $Q_1(\texttt{first}) \wedge \bigvee_{q_i \in F} Q_i(\texttt{last})$;
  - φ_{part} is the formula $\forall x \left[\bigvee_{i=1}^{k}(Q_i(x) \wedge \bigwedge_{j \neq i} Q_j(x)\right]$;
  - φ_{trans} is the formula

$$\forall x \forall y \left[\bigwedge_{a \in \Sigma, q_i \in Q} S(x,y) \wedge \mathtt{a}(x) \wedge Q_i(x) \rightarrow Q_j(x)\right],$$

    where each j is chosen such that $\delta(q_i, a) = q_j$.

From an automaton to an MSO-sentence

- Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA.
- We describe an MSO-sentence, $\varphi_{\mathcal{A}}$, which defines the regular language $L_{\mathcal{A}}$.
- $\varphi_{\mathcal{A}}$ is the sentence $\exists Q_1 \dots Q_k(\varphi_{boundary} \wedge \varphi_{part} \wedge \varphi_{trans})$, where
  - k is the number of states of $\mathcal{A}$;
  - $\varphi_{boundary}$ is the formula $Q_1(\texttt{first}) \wedge \bigvee_{q_i \in F} Q_i(\texttt{last})$;
  - φ_{part} is the formula $\forall x \left[\bigvee_{i=1}^{k}(Q_i(x) \wedge \bigwedge_{j \neq i} Q_j(x)\right]$;
  - φ_{trans} is the formula

$$\forall x \forall y \left[\bigwedge_{a \in \Sigma, q_i \in Q} S(x,y) \wedge \mathtt{a}(x) \wedge Q_i(x) \rightarrow Q_j(x)\right],$$

    where each j is chosen such that $\delta(q_i, a) = q_j$.

> **Exercise.** Prove that $\varphi_{\mathcal{A}}$ defines $L_{\mathcal{A}}$.

From an MSO-sentence to an automaton, I

- The proof is by induction on the complexity of the MSO-formula.
- There is a subtlety: an MSO-formula recognizes a language of words *with valuations*.
- If $\mathrm{Var} = \{x_1, \ldots, x_k, P_1, \ldots, P_m\}$ and φ is a formula with free variables in Var, then we need both a word w and a valuation V of Var into the domain of W, before we can determine whether or not φ is true.
- A Σ-word with valuation may be viewed as a $\Sigma \times \mathbf{2}^{\mathrm{Var}}$-word.

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, $\mathbf{n}$, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, $\mathbf{n}$, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.
- Equivalently, it specifies a $\{0, 1\}$-word of length n.

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, **n**, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.
- Equivalently, it specifies a $\{0, 1\}$-word of length n.
- For example: $n = 6$, $V(P) = \{2, 4, 5\}$:

$$\{1, 2, 3, 4, 5, 6\}$$

corresponds to bit-sequence

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, **n**, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.
- Equivalently, it specifies a $\{0, 1\}$-word of length n.
- For example: $n = 6$, $V(P) = \{2, 4, 5\}$:

$$\{1, 2, 3, 4, 5, 6\}$$

corresponds to bit-sequence

$$0\ 1\ 0\ 1\ 1\ 0.$$

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, $\mathbf{n}$, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.
- Equivalently, it specifies a $\{0, 1\}$-word of length n.
- For example: $n = 6$, $V(P) = \{2, 4, 5\}$:

$$\{1, 2, 3, 4, 5, 6\}$$

corresponds to bit-sequence

$$0\ 1\ 0\ 1\ 1\ 0.$$

- If $\Sigma = \{a, b\}$, the word $w = aaabbb$ with valuation $V(P) = \{2, 4, 5\}$ is encoded as the $\Sigma \times \mathbf{2}$-word:

$$(a, 0), (a, 1), (a, 0), (b, 1), (b, 1), (b, 0).$$

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, $\mathbf{n}$, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.
- Equivalently, it specifies a $\{0, 1\}$-word of length n.
- For example: $n = 6$, $V(P) = \{2, 4, 5\}$:

$$\{1, 2, 3, 4, 5, 6\}$$

corresponds to bit-sequence

$$0\ 1\ 0\ 1\ 1\ 0.$$

- If $\Sigma = \{a, b\}$, the word $w = aaabbb$ with valuation $V(P) = \{2, 4, 5\}$ is encoded as the $\Sigma \times \mathbf{2}$-word:

$$(a, 0), (a, 1), (a, 0), (b, 1), (b, 1), (b, 0).$$

- We need one bit per variable at each position.

Encoding valuations as bit-sequences

- Let w be a Σ-word of length n.
- A valuation of a monadic second order variable P into the domain, $\mathbf{n}$, of W, picks a subset $V(P)$ of $\{1, \ldots, n\}$.
- Equivalently, it specifies a $\{0, 1\}$-word of length n.
- For example: $n = 6$, $V(P) = \{2, 4, 5\}$:

$$\{1, 2, 3, 4, 5, 6\}$$

corresponds to bit-sequence

$$0\ 1\ 0\ 1\ 1\ 0.$$

- If $\Sigma = \{a, b\}$, the word $w = aaabbb$ with valuation $V(P) = \{2, 4, 5\}$ is encoded as the $\Sigma \times \mathbf{2}$-word:

$$(a, 0), (a, 1), (a, 0), (b, 1), (b, 1), (b, 0).$$

- We need one bit per variable at each position.

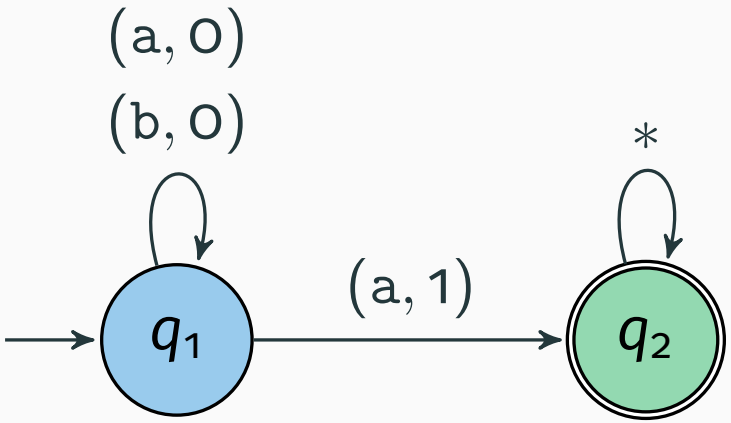
> What happens to first-order variables?

From an MSO-sentence to an automaton, II

- One may now directly describe automata recognizing the atomic formulae $x < y$, $P \subseteq Q$, $\mathtt{a}(x)$.

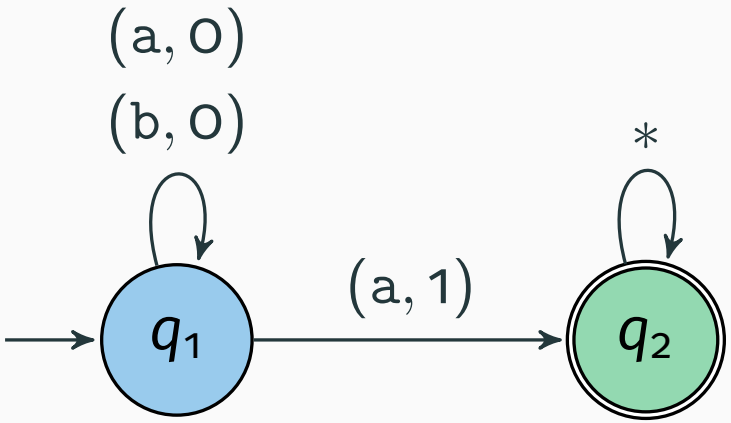
From an MSO-sentence to an automaton, II

- One may now directly describe automata recognizing the atomic formulae $x < y$, $P \subseteq Q$, $\mathrm{a}(x)$.
- For example, an NFA for checking $\mathrm{a}(x)$ when $\Sigma = \{a, b\}$:



From an MSO-sentence to an automaton, II

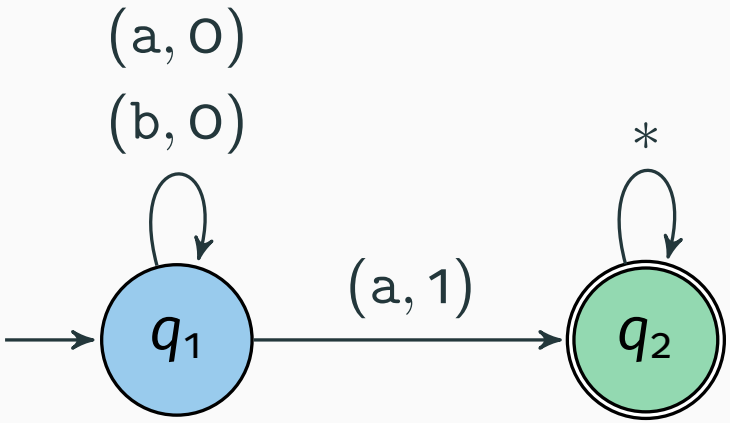
- One may now directly describe automata recognizing the atomic formulae $x < y$, $P \subseteq Q$, $\mathrm{a}(x)$.
- For example, an NFA for checking $\mathrm{a}(x)$ when $\Sigma = \{a, b\}$:



> **Exercise.** Give automata which check $P \subseteq Q$ and $x < y$.

From an MSO-sentence to an automaton, II

- One may now directly describe automata recognizing the atomic formulae $x < y$, $P \subseteq Q$, $\mathrm{a}(x)$.
- For example, an NFA for checking $\mathrm{a}(x)$ when $\Sigma = \{a, b\}$:



> **Exercise.** Give automata which check $P \subseteq Q$ and $x < y$.

- Assume given an automaton for φ_1 and φ_2.

> How to get an automaton for $\varphi_1 \vee \varphi_2$? And for $\neg\varphi_1$?

From an MSO-sentence to an automaton: existential quantifier

- Let $\varphi(P)$ be an MSO-formula with one free variable, P, over an alphabet Σ.

From an MSO-sentence to an automaton: existential quantifier

- Let $\varphi(P)$ be an MSO-formula with one free variable, P, over an alphabet Σ.
- Suppose that $\mathcal{A} = (Q, \Sigma \times \mathbf{2}, \delta, q_0, F)$ is an NFA such that, for any Σ-word w and subset X of the domain of W, $W, (P \mapsto X) \models \varphi(P)$ if, and only if, $\mathcal{A}$ accepts the $\Sigma \times \mathbf{2}$-word w^X.

From an MSO-sentence to an automaton: existential quantifier

- Let $\varphi(P)$ be an MSO-formula with one free variable, P, over an alphabet Σ.
- Suppose that $\mathcal{A} = (Q, \Sigma \times \mathbf{2}, \delta, q_0, F)$ is an NFA such that, for any Σ-word w and subset X of the domain of W, $W, (P \mapsto X) \models \varphi(P)$ if, and only if, $\mathcal{A}$ accepts the $\Sigma \times \mathbf{2}$-word w^X.
- Define the NFA $\mathcal{A}' := (Q, \Sigma, \delta', q_0, F)$, where

$$\delta'(q, a) := \delta(q, (a, 0)) \cup \delta(q, (a, 1)).$$

> **Exercise.** Show that $\mathcal{A}'$ accepts an input word w if, and only if, $W \models \exists P \varphi(P)$.

Theorem (Büchi, 1960)

A language is regular if, and only if, it is definable in monadic second order logic:

$$REG = MSO.$$

Summary

- Automata give a first (incomplete) approximation of a model of computation.
- Regular languages are the languages recognized by automata.
- Non-determinism adds flexibility to automata, while not extending the class of regular languages.
- Logic provides an alternative, descriptive formalism for languages.
- Monadic Second Order logic captures exactly the regular languages.

On Wednesday

- Turing machines: motivation and definition.
- Undecidable problems.
- Definition of main complexity classes (P and NP).
- Connection with logic (Fagin's theorem).
- Look at chapters 3 & 4 in Sipser.
- **Reminder!** Top-3 topic preferences due on Blackboard on Wednesday at 5pm.
- **Reminder!** Research question & choice of peer reviewer on Friday at 5pm.

Reference

M. Sipser, *Introduction to the Theory of Computation*, 3rd edition, Cengage Learning (2013).