NFA, DFA, and regular expressions

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

November 12, 2008

Outline

Regular operations

Last time, we defined 3 regular operations:

- Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$,
- Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$,
- Star: $A^* = \{x_1 x_2 \cdots x_k | k \geq 0 \text{ and each } x_i \in A\}$,

Regular operations

Last time, we defined 3 regular operations:

- Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$,
- Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$,
- Star: $A^* = \{x_1 x_2 \cdots x_k | k \geq 0 \text{ and each } x_i \in A\}$,

and proved the following theorem.

Theorem 1

The class of regular languages is closed under the union operation.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?
  - For string $w \in A_1 \circ A_2$, there exists a pair x and y such that $w = xy$ and $x \in A_1$ and $y \in A_2$.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?
  - For string $w \in A_1 \circ A_2$, there exists a pair x and y such that $w = xy$ and $x \in A_1$ and $y \in A_2$.
  - To construct a finite automaton M for $A_1 \circ A_2$ from M_1 and M_2 that recognize A_1 and A_2 we need to simulate M_1 to the end of x and start simulating M_2 right after that.

What about other operations?

- We prove Theorem 1 by simulating two finite automata with one finite automaton.
- This approach cannot be used directly to prove that the set of regular languages is closed under concatenation. Why?
  - For string $w \in A_1 \circ A_2$, there exists a pair x and y such that $w = xy$ and $x \in A_1$ and $y \in A_2$.
  - To construct a finite automaton M for $A_1 \circ A_2$ from M_1 and M_2 that recognize A_1 and A_2 we need to simulate M_1 to the end of x and start simulating M_2 right after that. **And it is hard to "tell" where x ends.**

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?

A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?
  - Ummm..

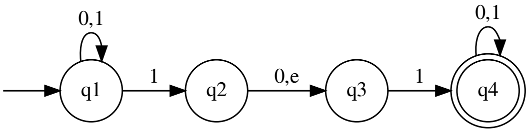
A machine that always guesses correctly

- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?
  - Ummm.. it definitely can,

A machine that always guesses correctly

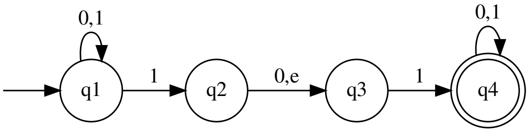
- Suppose that our machine can guess where x ends.
- It can
  - simulate M_1 on the input string up to the end of x,
  - jump to the start state in M_2 right after x ends, and
  - accept string $w = xy$ when the machine stops at some accept state in M_2.
- Can machine guess?
  - Maybe?
  - But guess correctly?
  - Ummm.. it definitely can, in theory.

Example: Nondeterministic Finite Automaton N_1



Note: e in the figure is ε.

Differences



- Duplicate symbols
- Missing symbols
- Empty string: ε

Deterministic and Nondeterministic Finite Automata

- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.

Deterministic and Nondeterministic Finite Automata

- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.
- Computation where each next step is fully determined is called **deterministic** computation.

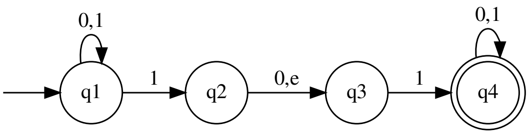
Deterministic and Nondeterministic Finite Automata

- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.
- Computation where each next step is fully determined is called **deterministic** computation.
- On the other hand, in **nondeterministic** computation, many choices may exist.

Deterministic and Nondeterministic Finite Automata

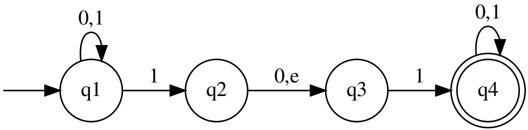
- Previously, we only consider finite automata whose next states are **determined** by their input alphabet and their current states.
- Computation where each next step is fully determined is called **deterministic** computation.
- On the other hand, in **nondeterministic** computation, many choices may exist.
- Therefore, we have deterministic finite automata (**DFA**) and nondeterministic finite automata (**NFA**).

How does N_1 compute?



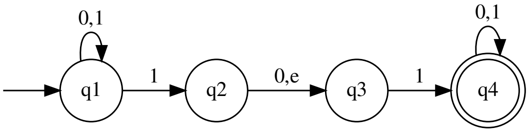
At any point where there are many choices for the next step, the machine **splits** itself into many copies and follow all possible steps in parallel.

How does N_1 compute?



At any point where there are many choices for the next step, the machine **splits** itself into many copies and follow all possible steps in parallel.
Think about *Kage Bunshin no Jutsu!*.

How does N_1 compute?



At any point where there are many choices for the next step, the machine **splits** itself into many copies and follow all possible steps in parallel.

Think about *Kage Bunshin no Jutsu!*.

See simulation.

Rules for computation of nondeterministic finite automata

- If there are many choices, split.

Rules for computation of nondeterministic finite automata

- If there are many choices, split.
- Copies die if they can't move according to the input.

Rules for computation of nondeterministic finite automata

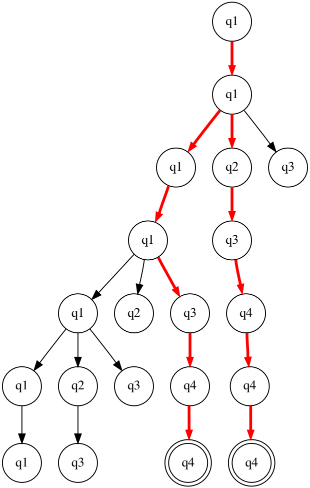
- If there are many choices, split.
- Copies die if they can't move according to the input.
- When to accept a string:

Rules for computation of nondeterministic finite automata

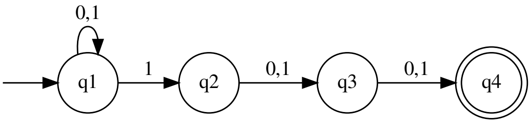
- If there are many choices, split.
- Copies die if they can't move according to the input.
- When to accept a string:
    - At the end of the input, if **any** of the copies is in an accept state, it **accept** the input.

N_1 on 010110

N_1 on 010110

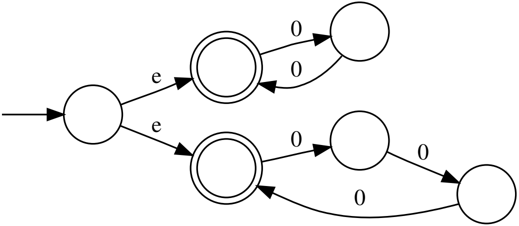


NFA N_2: what are the strings accepted by N_2?



NFA N_3: what are the strings accepted by N_3?

Let $\{0\}$ be the alphabet for N_3.



Formal definition of NFAs: δ

- The transition function δ:
  - takes

Formal definition of NFAs: δ

- The transition function δ:
  - takes the current state and a symbol in

Formal definition of NFAs: δ

- The transition function δ:
  - takes the current state and a symbol in $\Sigma \cup \{\varepsilon\}$, and

Formal definition of NFAs: δ

- The transition function δ:
  - takes the current state and a symbol in $\Sigma \cup \{\varepsilon\}$, and
  - outputs

Formal definition of NFAs: δ

- The transition function δ:
  - takes the current state and a symbol in $\Sigma \cup \{\varepsilon\}$, and
  - outputs a subset of states Q.

Formal definition of NFAs: δ

- The transition function δ:
  - takes the current state and a symbol in $\Sigma \cup \{\varepsilon\}$, and
  - outputs a subset of states Q.
- Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
- Denote by $\mathcal{P}(Q)$ the **power set** of set Q.

Formal definition of NFAs: δ

- The transition function δ:
  - takes the current state and a symbol in $\Sigma \cup \{\varepsilon\}$, and
  - outputs a subset of states Q.
- Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
- Denote by $\mathcal{P}(Q)$ the **power set** of set Q.
- We have that the state transition δ for an NFA is a function from

Formal definition of NFAs: δ

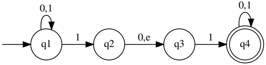
- The transition function δ:
  - takes the current state and a symbol in $\Sigma \cup \{\varepsilon\}$, and
  - outputs a subset of states Q.
- Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
- Denote by $\mathcal{P}(Q)$ the **power set** of set Q.
- We have that the state transition δ for an NFA is a function from $Q \times \Sigma_\varepsilon$ to $\mathcal{P}(Q)$.

Definition [NFA]

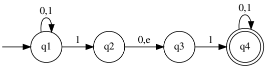
A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Example: N_1



Example: N_1



N_1 is $(Q, \Sigma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0, 1\}$,
3. δ is defined as

	0	1	$\emptyset$
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Formal definition of computation of NFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let w be a string over alphabet Σ.
We say that N **accepts** w if we can write $w = w_1 w_2 \cdots w_n$ where each w_i is a member of Σ_ε and

Formal definition of computation of NFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let w be a string over alphabet Σ.
We say that *N* **accepts** *w* if we can write $w = w_1 w_2 \cdots w_n$ where each w_i is a member of Σ_ε and there exists a sequence of states $r_0, r_1, \ldots, r_n$ in Q such that

Formal definition of computation of NFAs

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let w be a string over alphabet Σ.

We say that **N accepts w** if we can write $w = w_1 w_2 \cdots w_n$ where each w_i is a member of Σ_ε and there exists a sequence of states $r_0, r_1, \ldots, r_n$ in Q such that

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Are NFAs more powerful than DFAs?

- With the power of nondeterminism, NFAs seem to be more powerful.

Are NFAs more powerful than DFAs?

- With the power of nondeterminism, NFAs seem to be more powerful.
- In fact, DFAs and NFAs recognize the **same** class of languages!

Are NFAs more powerful than DFAs?

- With the power of nondeterminism, NFAs seem to be more powerful.
- In fact, DFAs and NFAs recognize the **same** class of languages!
- We say that two machines are *equivalent* if they recognize the same language.

Proving equivalence

Two directions:

- Given a DFA, construct an NFA recognizing the same language.

Proving equivalence

Two directions:

- Given a DFA, construct an NFA recognizing the same language.
  - Easy! DFA is also an NFA.

Proving equivalence

Two directions:

- Given a DFA, construct an NFA recognizing the same language.
  - Easy! DFA is also an NFA.
- Given an NFA, construct a DFA recognizing the same language.

Proving equivalence

Two directions:

- Given a DFA, construct an NFA recognizing the same language.
  - Easy! DFA is also an NFA.
- Given an NFA, construct a DFA recognizing the same language.
  - No that easy.

Main theorem

Theorem 2

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Main theorem

Theorem 2

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

How can we prove that?

Main theorem

Theorem 2

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

How can we prove that?

- Recall "reader as automaton"?

Main theorem

Theorem 2

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

How can we prove that?

- Recall "reader as automaton"?
- Given an NFA N, think of a DFA M as a manager who operates N.

Main theorem

Theorem 2

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

How can we prove that?

- Recall "reader as automaton"?
- Given an NFA N, think of a DFA M as a manager who operates N.
- What does M have to **remember** in order to simulate N correctly?

Proof of Theorem 2

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA.

Proof of Theorem 2

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. We shall construct a DFA $M = (Q', \Sigma, \delta', q'_0, F')$ recognizing the same language.

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

1. $Q' = \mathcal{P}(Q)$,

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

1. $Q' = \mathcal{P}(Q)$,
2. If N is at state $q \in Q$ and receives input symbol $a \in \Sigma$, N may moves to any states in $\delta(q, a)$.

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

1. $Q' = \mathcal{P}(Q)$,
2. If N is at state $q \in Q$ and receives input symbol $a \in \Sigma$, N may moves to any states in $\delta(q, a)$.
   Now M pretends to be on many states in N, i.e., M's state is some $R \subseteq Q$.

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

1. $Q' = \mathcal{P}(Q)$,
2. If N is at state $q \in Q$ and receives input symbol $a \in \Sigma$, N may moves to any states in $\delta(q, a)$.
   Now M pretends to be on many states in N, i.e., M's state is some $R \subseteq Q$. Given a as a input, its possible next state is

$$\delta'(R, a) = \bigcup_{q \in R} \delta(q, a)$$

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

1. $Q' = \mathcal{P}(Q)$,
2. If N is at state $q \in Q$ and receives input symbol $a \in \Sigma$, N may moves to any states in $\delta(q, a)$.
   Now M pretends to be on many states in N, i.e., M's state is some $R \subseteq Q$. Given a as a input, its possible next state is

$$\delta'(R, a) = \bigcup_{q \in R} \delta(q, a)$$

3. $q_0' = \{q_0\}$, and

Proof of Theorem 2: easy case

Easy case: N has no ε arrows.

1. $Q' = \mathcal{P}(Q)$,
2. If N is at state $q \in Q$ and receives input symbol $a \in \Sigma$, N may moves to any states in $\delta(q, a)$.
   Now M pretends to be on many states in N, i.e., M's state is some $R \subseteq Q$. Given a as a input, its possible next state is
   $$\delta'(R, a) = \bigcup_{q \in R} \delta(q, a)$$
3. $q_0' = \{q_0\}$, and
4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$.

Proof of Theorem 2: dealing with ε (1)

- It's time to deal with ε.

Proof of Theorem 2: dealing with ε (1)

- It's time to deal with ε.
- With that symbol on a arrow, N can move freely on that arrow (without taking any input).

Proof of Theorem 2: dealing with ε (1)

- It's time to deal with ε.
- With that symbol on a arrow, N can move freely on that arrow (without taking any input).
- Define, for state $q \in Q$, $D(q)$ to be a set of states in Q that can be reached from q by traveling along 0 or more ε arrows.

Proof of Theorem 2: dealing with ε (1)

- It's time to deal with ε.
- With that symbol on a arrow, N can move freely on that arrow (without taking any input).
- Define, for state $q \in Q$, $D(q)$ to be a set of states in Q that can be reached from q by traveling along 0 or more ε arrows.
- If N is at q, it can move freely to any states in $D(q)$.

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.
- M simulates many copies of N at states in R.

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.
- M simulates many copies of N at states in R.
- Define $E(R)$ to be a collection of states reachable from any states in R by traveling along 0 or more ε arrows.

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.
- M simulates many copies of N at states in R.
- Define $E(R)$ to be a collection of states reachable from any states in R by traveling along 0 or more ε arrows.
- Formally, $E(R) = \{q|$ q can be reached from states in R by traveling along 0 or more ε arrows $\}$

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.
- M simulates many copies of N at states in R.
- Define $E(R)$ to be a collection of states reachable from any states in R by traveling along 0 or more ε arrows.
- Formally, $E(R) = \{q|$ q can be reached from states in R by traveling along 0 or more ε arrows $\}$
- Combining these moves to the previous construction of δ':

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.
- M simulates many copies of N at states in R.
- Define $E(R)$ to be a collection of states reachable from any states in R by traveling along 0 or more ε arrows.
- Formally, $E(R) = \{q|\ q$ can be reached from states in R by traveling along 0 or more ε arrows $\}$
- Combining these moves to the previous construction of δ':

$$\delta'(R, a) = \bigcup_{q \in R} E(\delta(q, a)).$$

Proof of Theorem 2: dealing with ε (2)

- Consiser M at state $R \in Q'$.
- M simulates many copies of N at states in R.
- Define $E(R)$ to be a collection of states reachable from any states in R by traveling along 0 or more ε arrows.
- Formally, $E(R) = \{q|\ q$ can be reached from states in R by traveling along 0 or more ε arrows $\}$
- Combining these moves to the previous construction of δ':

$$\delta'(R, a) = \bigcup_{q \in R} E(\delta(q, a)).$$

- Fix start states $q_0' = E(\{q_0\})$.

Finishing the proof

- At any point on the computation of M, the state of M is the set of all possible states that N can be in at that point.

Finishing the proof

- At any point on the computation of M, the state of M is the set of all possible states that N can be in at that point.
- M correctly simulates N.

Finishing the proof

- At any point on the computation of M, the state of M is the set of all possible states that N can be in at that point.
- M correctly simulates N.
- Thus, our proof is complete.

Note on the correctness proof

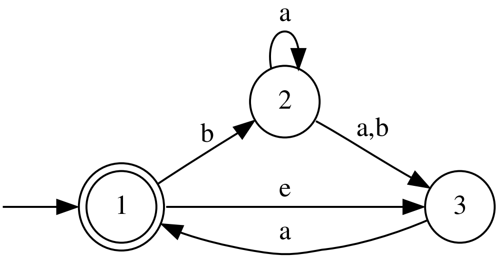
- Our previous proof of Theorem 2 is quite short and does not give out all the details.
- This is okay for now, since our construction is simple enough so that it is quite obvious that it is correct.
- For more complicated constructions, we need to be more formal.

A more general definition of regular languages

Corollary 3

A language is regular iff some nondeterministic finite automaton recognizes it.

Example



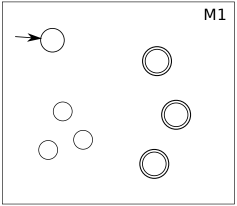
Closure under the regular operations

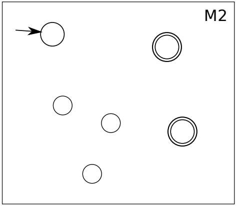
- Using NFA-DFA equivalence, it is much easy to prove that the set of regular languages is closed under the regular operations.

Closure under the regular operations

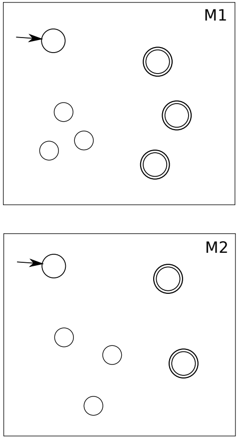
- Using NFA-DFA equivalence, it is much easy to prove that the set of regular languages is closed under the regular operations.
- We'll look at each operation.

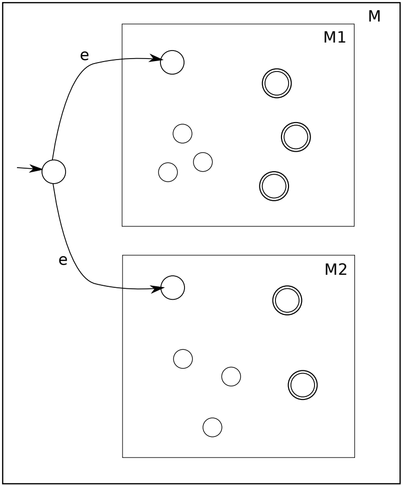
Union



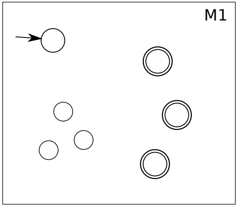


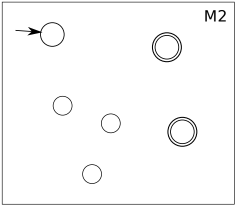
Union



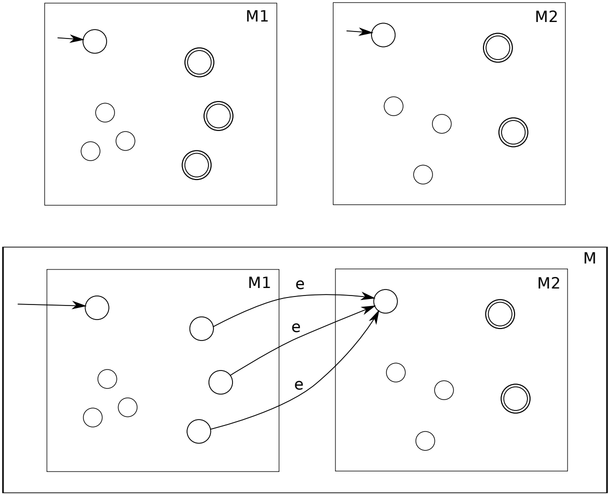


Concatenation

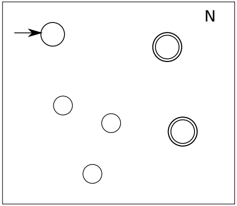




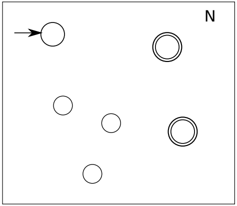
Concatenation

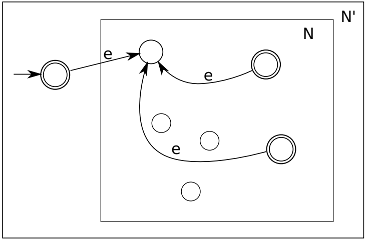


Star



Star





Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- $A_1 = \{w|\ w \text{ contains even number of 1's }\}$.

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- $A_1 = \{w|\ w \text{ contains even number of 1's }\}$.
  Find an FA M_1 that recognizes A_1.

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- $A_1 = \{w |\ w$ contains even number of 1's $\}$.
  Find an FA M_1 that recognizes A_1.
- $A_2 = \{w |\ w$ contains odd number of 0's $\}$.

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- $A_1 = \{w|\ w \text{ contains even number of 1's }\}$.
  Find an FA M_1 that recognizes A_1.
- $A_2 = \{w|\ w \text{ contains odd number of 0's }\}$.
  Find an FA M_2 that recognizes A_2.

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- $A_1 = \{w \mid w \text{ contains even number of 1's }\}$.
  Find an FA M_1 that recognizes A_1.
- $A_2 = \{w \mid w \text{ contains odd number of 0's }\}$.
  Find an FA M_2 that recognizes A_2.
- Construct an NFA N_1 recognizing $A_1 \cup A_2$.
- Construct an NFA N_2 recognizing $A_1 \circ A_2$.
- Construct an NFA N_3 recognizing $(A_1^*) \circ A_2$.

Building up patterns

- Regular operations can be used to build expressions describing languages.

Building up patterns

- Regular operations can be used to build expressions describing languages.
- For example,

$$(\{0\} \cup \{1\}) \circ \{0\}^*,$$

Building up patterns

- Regular operations can be used to build expressions describing languages.
- For example,

$$(\{0\} \cup \{1\}) \circ \{0\}^*,$$

or in a shorter form $(0 \cup 1)0^*$

Building up patterns

- Regular operations can be used to build expressions describing languages.
- For example,

$$(\{0\} \cup \{1\}) \circ \{0\}^*,$$

  or in a shorter form $(0 \cup 1)0^*$
- This is called a **regular expression**.
- Note: 0 denotes $\{0\}$, 1 denotes $\{1\}$, and $\circ$ is omitted.

Another examples

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

Another examples

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

- All possible strings (including ε).

Another examples

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

- All possible strings (including ε).
- If $\Sigma = \{0, 1\}$, we can write Σ for $(0 \cup 1)$, and write Σ^* for any strings from alphabet Σ.

Definition [regular expression]

R is a **regular expression** if R is

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

Definition [regular expression]

R is a **regular expression** if R is

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

This is an **inductive definition**.

Precedence

Operations are performed in this order:

- $*$
- $\circ$
- $\cup$

Shorthands

- RR^*

Shorthands

- RR^* can be written as R^+
- $RRRR$

Shorthands

- RR^* can be written as R^+
- $RRRR$ can be written as R^4, in general R^k is the concatenation of R to itself for k times.

Examples of regular expressions

1. 0^*10^*.

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$
6. $01 \cup 10$

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$
6. $01 \cup 10$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$
6. $01 \cup 10$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
8. $1^*\emptyset$

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$
6. $01 \cup 10$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
8. $1^*\emptyset = \emptyset$
9. $\emptyset^*$

Examples of regular expressions

1. 0^*10^*.
2. $\Sigma^*1\Sigma^*$.
3. $\Sigma^*001\Sigma^*$.
4. $(01^+)^*$
5. $(\Sigma\Sigma)^*$
6. $01 \cup 10$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$
8. $1^*\emptyset = \emptyset$
9. $\emptyset^* = \{\varepsilon\}$

Properties

- $R \cup \emptyset$

Properties

- $R \cup \emptyset = R$

Properties

- $R \cup \emptyset = R$
- $R \circ \varepsilon$

Properties

- $R \cup \emptyset = R$
- $R \circ \varepsilon = R$

Properties

- $R \cup \emptyset = R$
- $R \circ \varepsilon = R$
- $R \cup \varepsilon$

Properties

- $R \cup \emptyset = R$
- $R \circ \varepsilon = R$
- $R \cup \varepsilon$ might not equal R.

Properties

- $R \cup \emptyset = R$
- $R \circ \varepsilon = R$
- $R \cup \varepsilon$ might not equal R.
- $R \circ \emptyset$

Properties

- $R \cup \emptyset = R$
- $R \circ \varepsilon = R$
- $R \cup \varepsilon$ might not equal R.
- $R \circ \emptyset$ might not equal R.

Equivalence

Theorem 4

A language is regular iff some regular expression describes it.

Equivalence

Theorem 4

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular.
- If a language is regular, then it can be described by a regular expression.

Equivalence

Theorem 4

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular.
- If a language is regular, then it can be described by a regular expression.

Today we'll prove only the first direction.

Practice

- Let alphabet $\Sigma = \{0, 1\}$.

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- Find an FA M_1 that recognizes 01^+

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- Find an FA M_1 that recognizes 01^+
- Find an FA M_2 that recognizes $(10)^*$

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- Find an FA M_1 that recognizes 01^+
- Find an FA M_2 that recognizes $(10)^*$
- Find an FA M_3 that recognizes $(01^+) \cup (10)^*$

Practice

- Let alphabet $\Sigma = \{0, 1\}$.
- Find an FA M_1 that recognizes 01^+
- Find an FA M_2 that recognizes $(10)^*$
- Find an FA M_3 that recognizes $(01^+) \cup (10)^*$
- Find an FA M_4 that recognizes $1^+ \circ ((01^+) \cup (10)^*)$

A regular expression describes a regular language

Lemma 5

If a language is described by a regular expression, then it is regular.

A regular expression describes a regular language

Lemma 5

If a language is described by a regular expression, then it is regular.

To prove this, we'll look at how we a regular expression is constructed.

Rule 1

R is a **regular expression** if R is a for some $a \in \Sigma$.

Rule 1

R is a **regular expression** if R is a for some $a \in \Sigma$.

What is a DFA that recognizes R?

Rule 2

R is a **regular expression** if R is ε.

Rule 2

R is a **regular expression** if R is ε.

What is a DFA that recognizes R?

Rule 3

R is a **regular expression** if R is $\emptyset$.

Rule 3

R is a **regular expression** if R is $\emptyset$.

What is a DFA that recognizes R?

Rule 4

R is a **regular expression** if R is $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions.

Rule 4

R is a **regular expression** if R is $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions.

Given an NFA N_1 and N_2 that recognize R_1 and R_2, what is a NFA that recognizes R?

Rule 5

R is a **regular expression** if R is $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions.

Rule 5

R is a **regular expression** if R is $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions.

Given an NFA N_1 and N_2 that recognize R_1 and R_2, what is a NFA that recognizes R?

Rule 6

R is a **regular expression** if R is (R_1^*) where R_1 is a regular expression.

Rule 6

R is a **regular expression** if R is (R_1^*) where R_1 is a regular expression.

Given an NFA N_1 that recognizes R_1, what is a NFA that recognizes R?

Proof by Structural Induction

- In proving Lemma 5, we show how to construct an NFA of a regular expression given NFAs of its subexpressions.

Proof by Structural Induction

- In proving Lemma 5, we show how to construct an NFA of a regular expression given NFAs of its subexpressions.
- The proof is again

Proof by Structural Induction

- In proving Lemma 5, we show how to construct an NFA of a regular expression given NFAs of its subexpressions.
- The proof is again an inductive proof.

Proof by Structural Induction

- In proving Lemma 5, we show how to construct an NFA of a regular expression given NFAs of its subexpressions.
- The proof is again an inductive proof.
- Sometimes, this kind of inductive proofs is called structural induction.

Proof by Structural Induction

- In proving Lemma 5, we show how to construct an NFA of a regular expression given NFAs of its subexpressions.
- The proof is again an inductive proof.
- Sometimes, this kind of inductive proofs is called structural induction.
  - Inductive Hypothesis (when considering a regular expression R): Assume that for all smaller regular expressions R', the language described by R' can be recognized by some NFA N'.

Practice

1. Find an NFA recognizing $(01 \cup 0)^*$.

Practice

1. Find an NFA recognizing $(01 \cup 0)^*$.
   Try to build an NFA using the construction discussed in class.

Practice

1. Find an NFA recognizing $(01 \cup 0)^*$.
   Try to build an NFA using the construction discussed in class.
2. Find an NFA recognizing $(0 \cup 1)^* 010$.