Theoretical Computer Science
Regular Languages

Jonas Hübotter

Outline I

Outline II

Regular Expressions
- Definition
- Interlude: Structural Induction
- Regex $\rightarrow$ ϵ-NFA (Kleene)
- DFA/NFA $\rightarrow$ Regex (Kleene)
- Arden's Lemma

Closure Properties

Pumping Lemma

Overview

Representations of regular languages

Overview

Representations of regular languages

- Right-Linear Grammar (RLG)

Overview

Representations of regular languages

- Right-Linear Grammar (RLG)
- Deterministic Finite Automaton (DFA)

Overview

Representations of regular languages

- Right-Linear Grammar (RLG)
- Deterministic Finite Automaton (DFA)
- Nondeterministic Finite Automaton (NFA)

Overview

Representations of regular languages

- Right-Linear Grammar (RLG)
- Deterministic Finite Automaton (DFA)
- Nondeterministic Finite Automaton (NFA)
- ϵ-NFA

Overview

Representations of regular languages

- Right-Linear Grammar (RLG)
- Deterministic Finite Automaton (DFA)
- Nondeterministic Finite Automaton (NFA)
- ϵ-NFA
- Regular Expression (Regex)

DFA

Definition 1

A deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$

DFA

Definition 1

A deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ consists of

- a finite set of states Q

DFA

Definition 1

A deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ consists of

- a finite set of states Q;
- a (finite) alphabet Σ

DFA

Definition 1

A deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ consists of

- a finite set of states Q;
- a (finite) alphabet Σ;
- a total transition function $\delta : Q \times \Sigma \to Q$

DFA

Definition 1

A deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ consists of

- a finite set of states Q;
- a (finite) alphabet Σ;
- a total transition function $\delta : Q \times \Sigma \to Q$;
- an initial state $q_0 \in Q$

DFA

Definition 1

A deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ consists of

- a finite set of states Q;
- a (finite) alphabet Σ;
- a total transition function $\delta : Q \times \Sigma \to Q$;
- an initial state $q_0 \in Q$; and
- a set of terminal (accepting) states $F \subseteq Q$.

DFA

Definition 2

The induced transition function $\hat{\delta}$ of a DFA M is defined by

DFA

Definition 2

The induced transition function $\hat{\delta}$ of a DFA M is defined by

$$\hat{\delta}(q, \epsilon) = q$$

DFA

Definition 2

The induced transition function $\hat{\delta}$ of a DFA M is defined by

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, aw) = \hat{\delta}(\delta(q, a), w), a \in \Sigma, w \in \Sigma^*.$$

DFA

Definition 2

The induced transition function $\hat{\delta}$ of a DFA M is defined by

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, aw) &= \hat{\delta}(\delta(q, a), w), a \in \Sigma, w \in \Sigma^*. \end{aligned}$$

The language accepted by M is $L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$.

NFA

Definition 3

A nondeterministic finite automaton (NFA) $N = (Q, \Sigma, \delta, q_0, F)$

NFA

Definition 3

A nondeterministic finite automaton (NFA) $N = (Q, \Sigma, \delta, q_0, F)$ consists of

- Q, Σ, q_0, F as defined for DFAs

NFA

Definition 3

A nondeterministic finite automaton (NFA) $N = (Q, \Sigma, \delta, q_0, F)$ consists of

- Q, Σ, q_0, F as defined for DFAs; and
- a (partial) transition function $\delta : Q \times \Sigma \to 2^Q$.

NFA

Definition 4

The induced transition function $\hat{\hat{\delta}}$ of a NFA N is defined analogously to $\hat{\delta}$ where

NFA

Definition 4

The induced transition function $\hat{\bar{\delta}}$ of a NFA N is defined analogously to $\hat{\delta}$ where

$$\bar{\delta} : 2^Q \times \Sigma \to 2^Q, (S, a) \mapsto \bigcup_{q \in S} \delta(q, a).$$

NFA

Definition 4

The induced transition function $\hat{\bar{\delta}}$ of a NFA N is defined analogously to $\hat{\delta}$ where

$$\bar{\delta} : 2^Q \times \Sigma \to 2^Q, (S, a) \mapsto \bigcup_{q \in S} \delta(q, a).$$

The language accepted by N is
$L(N) = \{w \in \Sigma^* \mid \hat{\bar{\delta}}(\{q_0\}, w) \cap F \neq \emptyset\}$.

NFA $\rightarrow$ DFA (determinization)

Idea

Interpret every reachable subset $S \subseteq 2^Q$ in the NFA N as its own state in the new DFA M.

NFA $\to$ DFA (determinization)

Idea

Interpret every reachable subset $S \subseteq 2^Q$ in the NFA N as its own state in the new DFA M.
Every state S of M where $S \cap F_N \neq \emptyset$ is an accepting state of M.

NFA $\to$ DFA (determinization)

Idea

Interpret every reachable subset $S \subseteq 2^Q$ in the NFA N as its own state in the new DFA M.
Every state S of M where $S \cap F_N \neq \emptyset$ is an accepting state of M.

Worst-case exponential growth!

ϵ-NFA

Definition 5

An ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$ is an NFA with a special symbol $\epsilon \neg \in \Sigma$ where

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q.$$

ϵ-NFA

Definition 5

An ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$ is an NFA with a special symbol $\epsilon \neg \in \Sigma$ where

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q.$$

ϵ-transitions can be executed at any time without reading a symbol.

ϵ-NFA $\to$ NFA

Idea

Given ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$

ϵ-NFA $\rightarrow$ NFA

Idea

Given ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$ construct NFA $N' = (Q, \Sigma, \delta', q_0, F')$

ϵ-NFA $\to$ NFA

Idea

Given ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$ construct NFA $N' = (Q, \Sigma, \delta', q_0, F')$ where

$$\delta' : Q \times \Sigma \to 2^Q : (q, a) \mapsto \bigcup_{i,j \geq 0} \hat{\delta}(\{q\}, \epsilon^i a \epsilon^j)$$

ϵ-NFA $\rightarrow$ NFA

Idea

Given ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$ construct NFA $N' = (Q, \Sigma, \delta', q_0, F')$ where

$$\delta' : Q \times \Sigma \rightarrow 2^Q : (q, a) \mapsto \bigcup_{i,j \geq 0} \hat{\delta}(\{q\}, \epsilon^i a \epsilon^j);$$

if $\epsilon \in L(N)$ then $F' = F \cup \{q_0\}$ else $F' = F$.

Product-Construction

Idea

Given DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$

Product-Construction

Idea

Given DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$
the product automaton is $M = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$

Product-Construction

Idea

Given DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ the product automaton is $M = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$ where

$$\delta : (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2$$

Product-Construction

Idea

Given DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ the product automaton is $M = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$ where

$$\begin{aligned}\delta &: (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2 \\ &: ((q_1, q_2), a) \mapsto (\delta_1(q_1, a), \delta_2(q_2, a)).\end{aligned}$$

Product-Construction

Idea

Given DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ the product automaton is $M = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$ where

$$\begin{aligned} \delta &: (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2 \\ &: ((q_1, q_2), a) \mapsto (\delta_1(q_1, a), \delta_2(q_2, a)). \end{aligned}$$

For the product automaton $L(M) = L(M_1) \cap L(M_2)$ holds.

Minimal Automaton

For any regular language L there exists a DFA D of minimal size such that $L(D) = L$.

Minimal Automaton

For any regular language L there exists a DFA D of minimal size such that $L(D) = L$.

Algorithm ($\mathcal{O}(|Q|^2)$ for constant $|\Sigma|$)

1. remove unreachable states from q_0

Minimal Automaton

For any regular language L there exists a DFA D of minimal size such that $L(D) = L$.

Algorithm ($\mathcal{O}(|Q|^2)$ for constant $|\Sigma|$)

1. remove unreachable states from q_0
2. determine equivalent states

Minimal Automaton

For any regular language L there exists a DFA D of minimal size such that $L(D) = L$.

Algorithm ($\mathcal{O}(|Q|^2)$ for constant $|\Sigma|$)

1. remove unreachable states from q_0
2. determine equivalent states
3. merge equivalent states

Equivalent States

Definition 6

States $p, q \in Q$ are

- equivalent if $\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$

Equivalent States

Definition 6

States $p, q \in Q$ are

- equivalent if $\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$;
- distinguishable if they are not equivalent.

Equivalent States

Definition 6

States $p, q \in Q$ are

- equivalent if $\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$;
- distinguishable if they are not equivalent.

Algorithm for finding equivalent states

Idea: mark distinguishable states
step-by-step.

Equivalent States

Definition 6

States $p, q \in Q$ are

- equivalent if $\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$;
- distinguishable if they are not equivalent.

Algorithm for finding equivalent states

Idea: mark distinguishable states
step-by-step.

1. mark all pairs $p, q \in Q$ if $p \in F$
   and $q \in Q \setminus F$

Equivalent States

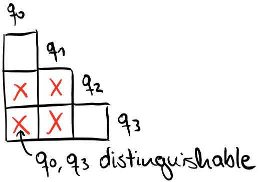
Definition 6

States $p, q \in Q$ are

- equivalent if $\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$;
- distinguishable if they are not equivalent.

Algorithm for finding equivalent states

Idea: mark distinguishable states step-by-step.



1. mark all pairs $p, q \in Q$ if $p \in F$ and $q \in Q \setminus F$
2. while $\exists$ unmarked $\{p, q\}$ and $\exists a \in \Sigma$, if $\{\delta(p, a), \delta(q, a)\}$ is marked, mark $\{p, q\}$

Interlude: Equivalence Relations

Definition 7

A relation $\sim \subseteq A \times A$ is an equivalence relation if

Interlude: Equivalence Relations

Definition 7

A relation $\sim \subseteq A \times A$ is an equivalence relation if

- $\forall a \in A.\ a \sim a$. (reflexivity)

Interlude: Equivalence Relations

Definition 7

A relation $\sim \subseteq A \times A$ is an equivalence relation if

- $\forall a \in A.\ a \sim a$. (reflexivity)
- $\forall a, b \in A.\ a \sim b \implies b \sim a$. (symmetry)

Interlude: Equivalence Relations

Definition 7

A relation $\sim \subseteq A \times A$ is an equivalence relation if

- $\forall a \in A.\ a \sim a.$ (reflexivity)
- $\forall a, b \in A.\ a \sim b \implies b \sim a.$ (symmetry) and
- $\forall a, b, c \in A.\ a \sim b \wedge b \sim c \implies a \sim c.$ (transitivity)

Interlude: Equivalence Relations

Definition 7

A relation $\sim \subseteq A \times A$ is an equivalence relation if

- $\forall a \in A.\ a \sim a$. (reflexivity)
- $\forall a, b \in A.\ a \sim b \implies b \sim a$. (symmetry) and
- $\forall a, b, c \in A.\ a \sim b \land b \sim c \implies a \sim c$. (transitivity)

$[a]_\sim = \{b \mid a \sim b\}$ is called the equivalence class of a under $\sim$.

Interlude: Equivalence Relations

Definition 7

A relation $\sim \subseteq A \times A$ is an equivalence relation if

- $\forall a \in A.\ a \sim a$. (reflexivity)
- $\forall a, b \in A.\ a \sim b \implies b \sim a$. (symmetry) and
- $\forall a, b, c \in A.\ a \sim b \wedge b \sim c \implies a \sim c$. (transitivity)

$[a]_\sim = \{b \mid a \sim b\}$ is called the equivalence class of a under $\sim$.

The set of equivalence classes $A/{\sim} = \{[a]_\sim \mid a \in A\}$ is called the quotient set of $\sim$.

Quotient Automaton

Observation: the equivalence of states defines an equivalence relation.

Quotient Automaton

Observation: the equivalence of states defines an equivalence relation.

We say $p \equiv_M q$ iff p and q are equivalent states in M.

Quotient Automaton

Observation: the equivalence of states defines an equivalence relation.

We say $p \equiv_M q$ iff p and q are equivalent states in M.

Definition 8

The collapsed automaton relative to $\equiv_M$ is called quotient automaton.

Quotient Automaton

Observation: the equivalence of states defines an equivalence relation.

We say $p \equiv_M q$ iff p and q are equivalent states in M.

Definition 8

The collapsed automaton relative to $\equiv_M$ is called quotient automaton.

$$M/{\equiv_M} = (Q/{\equiv_M}, \Sigma, \delta', [q_0]_{\equiv_M}, F/{\equiv_M})$$

Quotient Automaton

Observation: the equivalence of states defines an equivalence relation.

We say $p \equiv_M q$ iff p and q are equivalent states in M.

Definition 8

The collapsed automaton relative to $\equiv_M$ is called quotient automaton.

$$M/{\equiv_M} = (Q/{\equiv_M}, \Sigma, \delta', [q_0]_{\equiv_M}, F/{\equiv_M})$$

with $\delta'([p]_{\equiv_M}, a) = [\delta(p, a)]_{\equiv_M}$ for $p \in Q, a \in \Sigma$.

Canonical Minimal Automaton

Definition 9

The canonical minimal automaton M_L is a unique minimal automaton for any regular language L.

Canonical Minimal Automaton

Definition 9

The canonical minimal automaton M_L is a unique minimal automaton for any regular language L.

$$M_L = (\Sigma^*/{\equiv_L}, \Sigma, \delta_L, [\epsilon]_{\equiv_L}, F_L)$$

Canonical Minimal Automaton

Definition 9

The canonical minimal automaton M_L is a unique minimal automaton for any regular language L.

$$M_L = (\Sigma^*/\!\equiv_L, \Sigma, \delta_L, [\epsilon]_{\equiv_L}, F_L)$$

with

$$\delta_L([w]_{\equiv_L}, a) = [wa]_{\equiv_L}$$

Canonical Minimal Automaton

Definition 9

The canonical minimal automaton M_L is a unique minimal automaton for any regular language L.

$$M_L = (\Sigma^*/{\equiv_L}, \Sigma, \delta_L, [\epsilon]_{\equiv_L}, F_L)$$

with

$$\begin{aligned} \delta_L([w]_{\equiv_L}, a) &= [wa]_{\equiv_L} \\ F_L &= \{[w]_{\equiv_L} \mid w \in L\} \end{aligned}$$

Canonical Minimal Automaton

Definition 9

The canonical minimal automaton M_L is a unique minimal automaton for any regular language L.

$$M_L = (\Sigma^*/{\equiv_L}, \Sigma, \delta_L, [\epsilon]_{\equiv_L}, F_L)$$

with

$$\begin{aligned} \delta_L([w]_{\equiv_L}, a) &= [wa]_{\equiv_L} \\ F_L &= \{[w]_{\equiv_L} \mid w \in L\} \end{aligned}$$

It follows that $\hat{\delta}([\epsilon]_{\equiv_L}, w) = [w]_{\equiv_L}$ for $w \in \Sigma^*$,

Canonical Minimal Automaton

Definition 9

The canonical minimal automaton M_L is a unique minimal automaton for any regular language L.

$$M_L = (\Sigma^*/{\equiv_L}, \Sigma, \delta_L, [\epsilon]_{\equiv_L}, F_L)$$

with

$$\begin{aligned} \delta_L([w]_{\equiv_L}, a) &= [wa]_{\equiv_L} \\ F_L &= \{[w]_{\equiv_L} \mid w \in L\} \end{aligned}$$

It follows that $\hat{\delta}([\epsilon]_{\equiv_L}, w) = [w]_{\equiv_L}$ for $w \in \Sigma^*$, hence $L(M_L) = L$.

Theorem of Mihill-Nerode

Theorem 10 (Theorem of Mihill-Nerode)

$L \subseteq \Sigma^*$ is regular $\iff$ $\equiv_L$ has finitely many equivalence classes.

DFA $\rightarrow$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$

DFA $\rightarrow$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ define RLG $G = (Q, \Sigma, P, q_0)$

DFA $\to$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ define RLG $G = (Q, \Sigma, P, q_0)$ with productions P:

DFA $\to$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ define RLG $G = (Q, \Sigma, P, q_0)$ with productions P:

- $(q_1 \to aq_2) \in P$ iff $\delta(q_1, a) = q_2$

DFA $\to$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ define RLG $G = (Q, \Sigma, P, q_0)$ with productions P:

- $(q_1 \to aq_2) \in P$ iff $\delta(q_1, a) = q_2$;
- $(q_1 \to a) \in P$ iff $\delta(q_1, a) \in F$

DFA $\rightarrow$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ define RLG $G = (Q, \Sigma, P, q_0)$ with productions P:

- $(q_1 \rightarrow aq_2) \in P$ iff $\delta(q_1, a) = q_2$;
- $(q_1 \rightarrow a) \in P$ iff $\delta(q_1, a) \in F$; and
- $(q_0 \rightarrow \epsilon) \in P$ iff $q_0 \in F$.

DFA $\rightarrow$ RLG

Idea

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$ define RLG $G = (Q, \Sigma, P, q_0)$ with productions P:

- $(q_1 \rightarrow aq_2) \in P$ iff $\delta(q_1, a) = q_2$;
- $(q_1 \rightarrow a) \in P$ iff $\delta(q_1, a) \in F$; and
- $(q_0 \rightarrow \epsilon) \in P$ iff $q_0 \in F$.

Then, $L(G) = L(M)$.

RLG $\to$ NFA

Idea

Given RLG $G = (V, \Sigma, P, S)$ without the production $S \to \epsilon$

RLG $\rightarrow$ NFA

Idea

Given RLG $G = (V, \Sigma, P, S)$ without the production $S \rightarrow \epsilon$, define the NFA $N = (V \cup \{q_f\}, \Sigma, \delta, S, \{q_f\})$

RLG $\to$ NFA

Idea

Given RLG $G = (V, \Sigma, P, S)$ without the production $S \to \epsilon$, define the NFA $N = (V \cup \{q_f\}, \Sigma, \delta, S, \{q_f\})$ with:

- $Y \in \delta(X, a)$ iff $(X \to aY) \in P$

RLG $\to$ NFA

Idea

Given RLG $G = (V, \Sigma, P, S)$ without the production $S \to \epsilon$, define the NFA $N = (V \cup \{q_f\}, \Sigma, \delta, S, \{q_f\})$ with:

- $Y \in \delta(X, a)$ iff $(X \to aY) \in P$; and
- $q_f \in \delta(X, a)$ iff $(X \to a) \in P$.

RLG $\to$ NFA

Idea

Given RLG $G = (V, \Sigma, P, S)$ without the production $S \to \epsilon$, define the NFA $N = (V \cup \{q_f\}, \Sigma, \delta, S, \{q_f\})$ with:

- $Y \in \delta(X, a)$ iff $(X \to aY) \in P$; and
- $q_f \in \delta(X, a)$ iff $(X \to a) \in P$.

Then, $L(N) = L(G)$.

Regular Expressions

Syntax

- $\emptyset$ is a regular expression

Regular Expressions

Syntax

- $\emptyset$ is a regular expression;
- ϵ is a regular expression

Regular Expressions

Syntax

- $\emptyset$ is a regular expression;
- ϵ is a regular expression;
- $\forall a \in \Sigma$, a is a regular expression

Regular Expressions

Syntax

- $\emptyset$ is a regular expression;
- ϵ is a regular expression;
- $\forall a \in \Sigma$, a is a regular expression; and
- given regular expressions α, β, the following are regular expressions:

Regular Expressions

Syntax

- $\emptyset$ is a regular expression;
- ϵ is a regular expression;
- $\forall a \in \Sigma$, a is a regular expression; and
- given regular expressions α, β, the following are regular expressions:
  - $\alpha\beta$ (concatenation)

Regular Expressions

Syntax

- $\emptyset$ is a regular expression;
- ϵ is a regular expression;
- $\forall a \in \Sigma$, a is a regular expression; and
- given regular expressions α, β, the following are regular expressions:
    - $\alpha\beta$ (concatenation);
    - $\alpha|\beta$ (disjunction)

Regular Expressions

Syntax

- $\emptyset$ is a regular expression;
- ϵ is a regular expression;
- $\forall a \in \Sigma$, a is a regular expression; and
- given regular expressions α, β, the following are regular expressions:
  - $\alpha\beta$ (concatenation);
  - $\alpha|\beta$ (disjunction); and
  - α^* (repetition).

Regular Expressions

Semantics

- $L(\emptyset) = \emptyset$

Regular Expressions

Semantics

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$

Regular Expressions

Semantics

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$;
- $L(a) = \{a\}$

Regular Expressions

Semantics

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$;
- $L(a) = \{a\}$;
- $L(\alpha\beta) = L(\alpha)L(\beta)$

Regular Expressions

Semantics

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$;
- $L(a) = \{a\}$;
- $L(\alpha\beta) = L(\alpha)L(\beta)$;
- $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$

Regular Expressions

Semantics

- $L(\emptyset) = \emptyset$;
- $L(\epsilon) = \{\epsilon\}$;
- $L(a) = \{a\}$;
- $L(\alpha\beta) = L(\alpha)L(\beta)$;
- $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$; and
- $L(\alpha^*) = L(\alpha)^*$.

Interlude: Structural Induction

To prove a statement P for an object γ that is defined inductively, we use structural induction.

Interlude: Structural Induction

To prove a statement P for an object γ that is defined inductively, we use structural induction.

Let γ be defined by base cases $\alpha_1, \ldots, \alpha_k$ and inductive cases $\beta_1, \ldots, \beta_l$ with assumptions $a_{i1}, \ldots, a_{im_i}$ for $i \in \{1, \ldots, l\}$.

Interlude: Structural Induction

To prove a statement P for an object γ that is defined inductively, we use structural induction.

Let γ be defined by base cases $\alpha_1, \ldots, \alpha_k$ and inductive cases $\beta_1, \ldots, \beta_l$ with assumptions $a_{i1}, \ldots, a_{im_i}$ for $i \in \{1, \ldots, l\}$.

To prove P for all γ, prove:

- $P(\alpha_i)$ for $i \in \{1, \ldots, k\}$

Interlude: Structural Induction

To prove a statement P for an object γ that is defined inductively, we use structural induction.

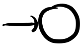
Let γ be defined by base cases $\alpha_1, \ldots, \alpha_k$ and inductive cases $\beta_1, \ldots, \beta_l$ with assumptions $a_{i1}, \ldots, a_{im_i}$ for $i \in \{1, \ldots, l\}$.

To prove P for all γ, prove:

- $P(\alpha_i)$ for $i \in \{1, \ldots, k\}$; and
- $P(a_{i1}) \wedge \cdots \wedge P(a_{im_i}) \implies P(\beta_i)$ for $i \in \{1, \ldots, l\}$.

Regex $\to$ ϵ-NFA (Kleene)

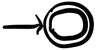
$\emptyset$



Regex $\to$ ϵ-NFA (Kleene)

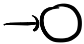
$\emptyset$

ϵ

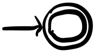


Regex → ϵ-NFA (Kleene)

$\emptyset$



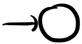
ϵ



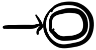
a

Regex $\to$ ϵ-NFA (Kleene)

$\emptyset$

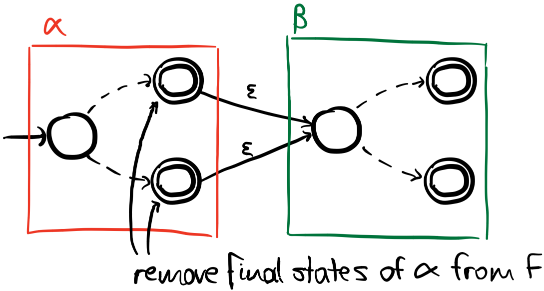


ϵ



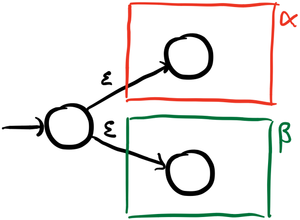
a

$\alpha\beta$



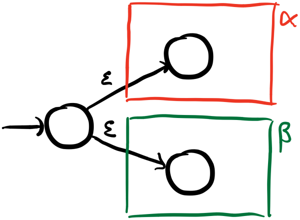
Regex $\to$ ϵ-NFA (Kleene)

$\alpha \mid \beta$

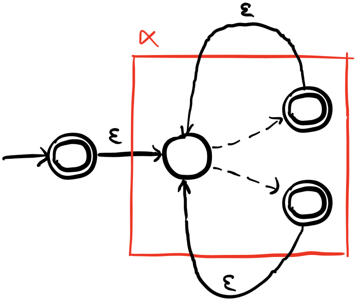


Regex → ϵ-NFA (Kleene)

$\alpha \mid \beta$



α^*



DFA/NFA $\to$ Regex (Kleene)

Given $M = (Q, \Sigma, \delta, q_1, F)$ with $Q = \{q_1, \ldots, q_n\}$ define

$$R_{ij}^k = \{w \in \Sigma^* \mid \text{input } w \text{ transitions from } q_i \text{ to } q_j \text{ and all states in between have an index } \leq k\}.$$

DFA/NFA $\to$ Regex (Kleene)

Given $M = (Q, \Sigma, \delta, q_1, F)$ with $Q = \{q_1, \ldots, q_n\}$ define

$$R_{ij}^k = \{w \in \Sigma^* \mid \text{input } w \text{ transitions from } q_i \text{ to } q_j \text{ and all states in between have an index } \leq k\}.$$

Idea: for all $i, j \in [n]$ and $k \in [n]_0$
a regex α_{ij}^k can be constructed with $L(\alpha_{ij}^k) = R_{ij}^k$.

DFA/NFA $\to$ Regex (Kleene)

Induction over k.

- $k = 0$:

DFA/NFA $\to$ Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R^0_{ij} = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

DFA/NFA $\to$ Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^0 = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

DFA/NFA $\to$ Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^{0} = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^{0} = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

where $\{a_1, \ldots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

DFA/NFA $\rightarrow$ Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^0 = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

  where $\{a_1, \ldots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.
- $k \implies k + 1$:

DFA/NFA → Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^0 = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

where $\{a_1, \ldots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

- $k \implies k+1$:

$$R_{ij}^{k+1} = R_{ij}^k$$

DFA/NFA $\rightarrow$ Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^0 = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

where $\{a_1, \ldots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

- $k \implies k + 1$:

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$

New paths using q_{k+1} in terms of the already built subpaths.

DFA/NFA $\to$ Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^0 = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

  where $\{a_1, \ldots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

- $k \implies k+1$:

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$
$$\alpha_{ij}^{k+1} = a_{ij}^k \mid a_{i(k+1)}^k (a_{(k+1)(k+1)}^k)^* a_{(k+1)j}^k.$$

New paths using q_{k+1} in terms of the already built subpaths.

DFA/NFA → Regex (Kleene)

Induction over k.

- $k = 0$: Let

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & i = j \end{cases}$$

$$\alpha_{ij}^0 = \begin{cases} a_1 \mid \cdots \mid a_l & i \neq j \\ a_1 \mid \cdots \mid a_l \mid \epsilon & i = j \end{cases}$$

where $\{a_1, \ldots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

- $k \implies k+1$:

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$
$$\alpha_{ij}^{k+1} = a_{ij}^k \mid a_{i(k+1)}^k (a_{(k+1)(k+1)}^k)^* a_{(k+1)j}^k.$$

New paths using q_{k+1} in terms of the already built subpaths.

We now have, $L(M) = L(\alpha_{1i_1}^n \mid \cdots \mid \alpha_{1i_r}^n)$ where $\{q_{i_1}, \ldots, q_{i_r}\} = F$.

Arden's Lemma

Theorem 11 (Arden's Lemma for regular languages)

Let A, B, X be regular languages and $\epsilon \notin A$, then:

$$X = AX \cup B \implies X = A^* B.$$

Arden's Lemma

Theorem 11 (Arden's Lemma for regular languages)

Let A, B, X be regular languages and $\epsilon \notin A$, then:

$$X = AX \cup B \implies X = A^*B.$$

Theorem 12 (Arden's Lemma for regular expressions)

Let α, β, X be regular expressions and $\epsilon \notin L(\alpha)$, then:

$$X = \alpha X \mid \beta \implies X = \alpha^*\beta.$$

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

- $R_1 R_2$

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

- $R_1 R_2$;
- $R_1 \cup R_2$

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

- $R_1 R_2$;
- $R_1 \cup R_2$;
- R^*

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

- $R_1 R_2$;
- $R_1 \cup R_2$;
- R^*;
- $\bar{R}$

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

- $R_1 R_2$;
- $R_1 \cup R_2$;
- R^*;
- $\bar{R}$;
- $R_1 \cap R_2$

Closure Properties

Theorem 13

Given the regular languages R, R_1, R_2, then the following are also regular languages:

- $R_1 R_2$;
- $R_1 \cup R_2$;
- R^*;
- $\bar{R}$;
- $R_1 \cap R_2$; *and*
- $R_1 \setminus R_2$.

Pumping Lemma

Lemma 14 (Pumping Lemma for regular languages)

Let $R \subseteq \Sigma^*$ *be regular.*

Pumping Lemma

Lemma 14 (Pumping Lemma for regular languages)

Let $R \subseteq \Sigma^*$ be regular. Then there exists some $n > 0$ such that every $z \in R$ with $|z| \geq n$ can be decomposed into $z = uvw$

Pumping Lemma

Lemma 14 (Pumping Lemma for regular languages)

Let $R \subseteq \Sigma^*$ be regular. Then there exists some $n > 0$ such that every $z \in R$ with $|z| \geq n$ can be decomposed into $z = uvw$ such that

- $v \neq \epsilon$

Pumping Lemma

Lemma 14 (Pumping Lemma for regular languages)

Let $R \subseteq \Sigma^*$ be regular. Then there exists some $n > 0$ such that every $z \in R$ with $|z| \geq n$ can be decomposed into $z = uvw$ such that

- $v \neq \epsilon$;
- $|uv| \leq n$

Pumping Lemma

Lemma 14 (Pumping Lemma for regular languages)

Let $R \subseteq \Sigma^*$ be regular. Then there exists some $n > 0$ such that every $z \in R$ with $|z| \geq n$ can be decomposed into $z = uvw$ such that

- $v \neq \epsilon$;
- $|uv| \leq n$; *and*
- $\forall i \geq 0.\ uv^i w \in R$.

Pumping Lemma

Lemma 14 (Pumping Lemma for regular languages)

Let $R \subseteq \Sigma^*$ be regular. Then there exists some $n > 0$ such that every $z \in R$ with $|z| \geq n$ can be decomposed into $z = uvw$ such that

- $v \neq \epsilon$;
- $|uv| \leq n$; *and*
- $\forall i \geq 0.\ uv^i w \in R$.

A necessary condition for regular languages.

Pumping Lemma

Example 15 (proof structure)

Assume L is regular.
Let $n > 0$ be a Pumping Lemma number.

Pumping Lemma

Example 15 (proof structure)

Assume L is regular.
Let $n > 0$ be a Pumping Lemma number.
Choose $z \in L$ with $|z| \geq n$.
Define $z = uvw$ with $v \neq \epsilon$ and $|uv| \leq n$.

Pumping Lemma

Example 15 (proof structure)

Assume L is regular.
Let $n > 0$ be a Pumping Lemma number.
Choose $z \in L$ with $|z| \geq n$.
Define $z = uvw$ with $v \neq \epsilon$ and $|uv| \leq n$.
Then, $\forall i \geq 0.\ uv^i w \in L$.

Pumping Lemma

Example 15 (proof structure)

Assume L is regular.
Let $n > 0$ be a Pumping Lemma number.
Choose $z \in L$ with $|z| \geq n$.
Define $z = uvw$ with $v \neq \epsilon$ and $|uv| \leq n$.
Then, $\forall i \geq 0.\ uv^i w \in L$.
Now, use the last statement to find a contradiction.