Regular Expressions

Regular Expressions

Regular expressions describing languages over an alphabet Σ:

- $\emptyset$, ε, a (where $a \in \Sigma$) are regular expressions:

  $\emptyset$ ... denotes the empty language
  ε ... denotes the language $\{\varepsilon\}$
  a ... denotes the language $\{a\}$

- If α, β are regular expressions then also $(\alpha + \beta)$, $(\alpha \cdot \beta)$, (α^*) are regular expressions:

  $(\alpha + \beta)$ ... denotes the union of languages denoted α and β
  $(\alpha \cdot \beta)$ ... denotes the concatenation of languages denoted α and β
  (α^*) ... denotes the iteration of a language denoted α

- There are no other regular expressions except those defined in the two points mentioned above.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, $\mathtt{0}$ and $\mathtt{1}$ are regular expressions.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.
- Since $(0 + 1)$ and (0^*) are regular expressions, $((0 + 1) \cdot (0^*))$ is also a regular expression.

Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

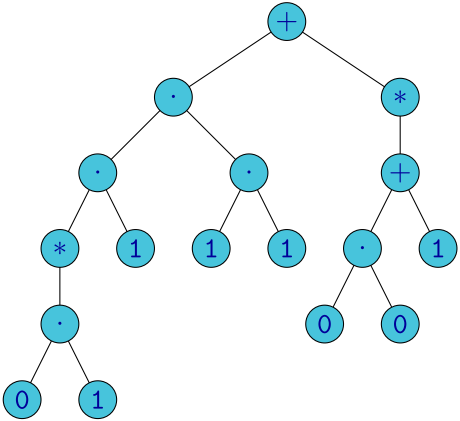
- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.
- Since $(0 + 1)$ and (0^*) are regular expressions, $((0 + 1) \cdot (0^*))$ is also a regular expression.

Remark: If α is a regular expression, by $\mathcal{L}(\alpha)$ we denote the language defined by the regular expression α.

$$\mathcal{L}(((0 + 1) \cdot (0^*))) = \{0,\ 1,\ 00,\ 10,\ 000,\ 100,\ 0000,\ 1000,\ 00000,\ \ldots\}$$

Regular Expressions

The structure of a regular expression can be represented by an abstract syntax tree:



$$(((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*))$$

Regular Expressions

The formal definition of semantics of regular expressions:

- $\mathcal{L}(\emptyset) = \emptyset$
- $\mathcal{L}(\varepsilon) = \{\varepsilon\}$
- $\mathcal{L}(a) = \{a\}$
- $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$
- $\mathcal{L}(\alpha \cdot \beta) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta)$
- $\mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$

Regular Expressions

To make regular expressions more lucid and succinct, we use the following conventions:

- The outward pair of parentheses can be omitted.
- We can omit parentheses that are superflous due to associativity of operations of union ($+$) and concatenation ($\cdot$).
- We can omit parentheses that are superflous due to the defined priority of operators (iteration (*) has the highest priority, concatenation ($\cdot$) has lower priority, and union ($+$) has the lowest priority).
- A dot denoting concatenation can be omitted.

Example: Instead of

$$(((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*))$$

we usually write

$$(01)^*111 + (00 + 1)^*$$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$ ... the language containing the only word $\mathtt{a}$

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$ ... the language containing the only word $\mathtt{a}$

$\mathtt{ab}$ ... the language containing the only word $\mathtt{ab}$

Regular Expressions

Examples: In all examples $\Sigma = \{a, b\}$.

a ... the language containing the only word a

ab ... the language containing the only word ab

$a + b$ ... the language containing two words a and b

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$ ... the language containing the only word $\mathtt{a}$

$\mathtt{ab}$ ... the language containing the only word $\mathtt{ab}$

$\mathtt{a} + \mathtt{b}$ ... the language containing two words $\mathtt{a}$ and $\mathtt{b}$

$\mathtt{a}^*$ ... the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, ...

Regular Expressions

Examples: In all examples $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

$\mathtt{a}$ ... the language containing the only word $\mathtt{a}$

$\mathtt{ab}$ ... the language containing the only word $\mathtt{ab}$

$\mathtt{a} + \mathtt{b}$ ... the language containing two words $\mathtt{a}$ and $\mathtt{b}$

$\mathtt{a}^*$ ... the language containing words ε, $\mathtt{a}$, $\mathtt{aa}$, $\mathtt{aaa}$, ...

$(\mathtt{ab})^*$ ... the language containing words ε, $\mathtt{ab}$, $\mathtt{abab}$, $\mathtt{ababab}$, ...

Regular Expressions

Examples: In all examples $\Sigma = \{a, b\}$.

a ... the language containing the only word a

ab ... the language containing the only word ab

$a + b$ ... the language containing two words a and b

a^* ... the language containing words ε, a, aa, aaa, ...

$(ab)^*$ ... the language containing words ε, ab, $abab$, $ababab$, ...

$(a + b)^*$ ... the language containing all words over the alphabet $\{a, b\}$

Regular Expressions

Examples: In all examples $\Sigma = \{a, b\}$.

a ... the language containing the only word a

ab ... the language containing the only word ab

$a + b$ ... the language containing two words a and b

a^* ... the language containing words ε, a, aa, aaa, ...

$(ab)^*$ ... the language containing words ε, ab, $abab$, $ababab$, ...

$(a + b)^*$ ... the language containing all words over the alphabet $\{a, b\}$

$(a + b)^*aa$ ... the language containing all words ending with aa

Regular Expressions

Examples: In all examples $\Sigma = \{a, b\}$.

a ... the language containing the only word a

ab ... the language containing the only word ab

$a + b$ ... the language containing two words a and b

a^* ... the language containing words ε, a, aa, aaa, ...

$(ab)^*$ ... the language containing words ε, ab, $abab$, $ababab$, ...

$(a + b)^*$ ... the language containing all words over the alphabet $\{a, b\}$

$(a + b)^*aa$ ... the language containing all words ending with aa

$(ab)^*bbb(ab)^*$ ... the language containing all words that contain a subword bbb preceded and followed by an arbitrary number of copies of the word ab

Regular Expressions

$(a + b)^*aa + (ab)^*bbb(ab)^*$ ... the language containing all words that either end with aa or contain a subwords bbb preceded and followed with some arbitrary number of words ab

Regular Expressions

$(a + b)^* aa + (ab)^* bbb(ab)^*$ ... the language containing all words that either end with aa or contain a subwords bbb preceded and followed with some arbitrary number of words ab

$(a + b)^* b(a + b)^*$ ... the language of all words that contain at least one occurrence of symbol b

Regular Expressions

$(\mathtt{a} + \mathtt{b})^*\mathtt{aa} + (\mathtt{ab})^*\mathtt{bbb}(\mathtt{ab})^*$ ... the language containing all words that either end with aa or contain a subwords bbb preceded and followed with some arbitrary number of words ab

$(\mathtt{a} + \mathtt{b})^*\mathtt{b}(\mathtt{a} + \mathtt{b})^*$ ... the language of all words that contain at least one occurrence of symbol b

$\mathtt{a}^*(\mathtt{ba}^*\mathtt{ba}^*)^*$ ... the language containg all words with an even number of occurrences of symbol b

Transformation of a Regular Expression to a Finite Automaton

Proposition

Every language that can be represented by a regular expression is regular (i.e., it is accepted by some finite automaton).

Proof: It is sufficient to show how to construct for a given regular expression α a finite automaton accepting the language $\mathcal{L}(\alpha)$.

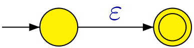
The construction is recursive and proceeds by the structure of the expression α:

- If α is a elementary expression (i.e., $\emptyset$, ε or a):
  - We construct the corresponding automaton directly.
- If α is of the form $(\beta + \gamma)$, $(\beta \cdot \gamma)$ or (β^*):
  - We construct automata accepting languages $\mathcal{L}(\beta)$ and $\mathcal{L}(\gamma)$ recursively.
  - Using these two automata, we construct the automaton accepting the language $\mathcal{L}(\alpha)$.

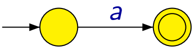
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

$\emptyset$



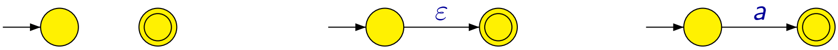
ε



a

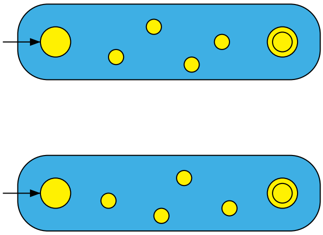
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:



$\emptyset$ ε a

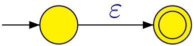
The construction for the union:



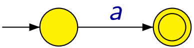
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

$\emptyset$

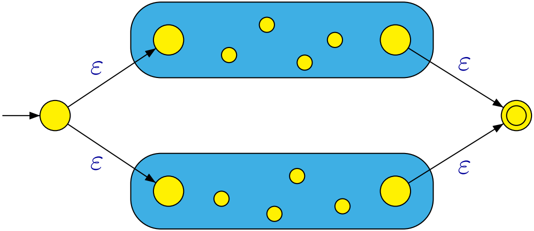


ε



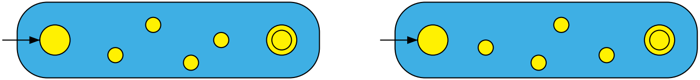
a

The construction for the union:



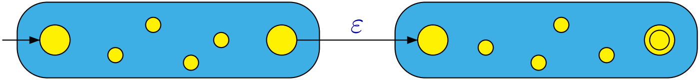
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:



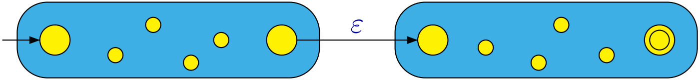
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

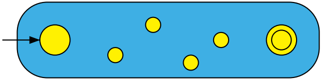


Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

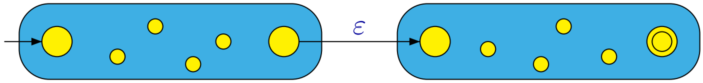


The construction for the iteration:

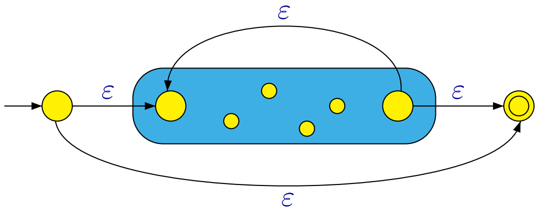


Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:



The construction for the iteration:

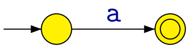


Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:

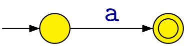
Transformation of a Regular Expression to a Finite Automaton

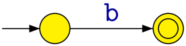
Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:



Transformation of a Regular Expression to a Finite Automaton

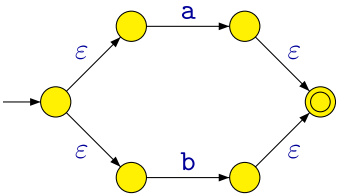
Example: The construction of an automaton for expression $((\mathtt{a} + \mathtt{b}) \cdot \mathtt{b})^*$:





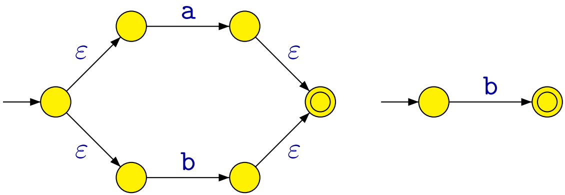
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a} + \mathtt{b}) \cdot \mathtt{b})^*$:



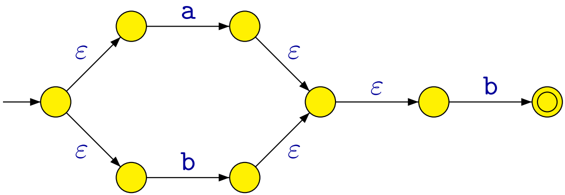
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:



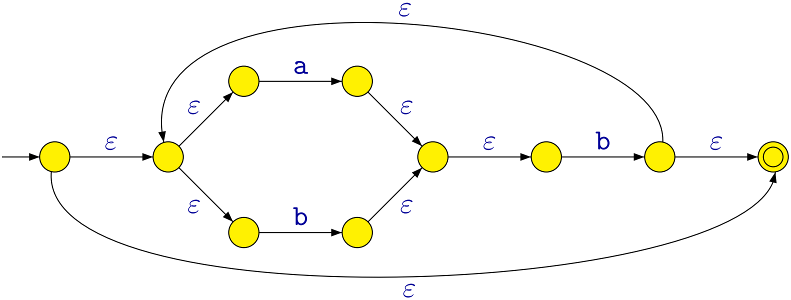
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a} + \mathtt{b}) \cdot \mathtt{b})^*$:



Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((\mathtt{a}+\mathtt{b})\cdot\mathtt{b})^*$:



Transformation of a Regular Expression to a Finite Automaton

If an expression α consists of n symbols (not counting parenthesis) then the resulting automaton has:

- at most $2n$ states,
- at most $4n$ transitions.

Remark: By transforming the generalized nondeterministic automaton into a deterministic one, the number of states can grow exponentially, i.e., the resulting automaton can have up to $2^{2n} = 4^n$ states.

Transformation of an Automaton to a Regular Expression

Proposition

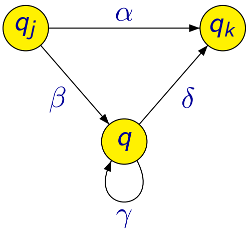
Every regular language can be represented by some regular expression.

Proof: It is sufficient to show how to construct for a given finite automaton $\mathcal{A}$ a regular expression α such that $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A})$.

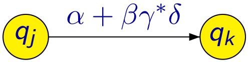
- We modify $\mathcal{A}$ in such a way that ensures it has exactly one initial and exactly one accepting state.
- Its states will be removed one by one.
- Its transitions will be labelled with regular expressions.
- The resulting automaton will have only two states – the initial and the accepting, and only one transition labelled with the resulting regular expression.

Transformation of an Automaton to a Regular Expression

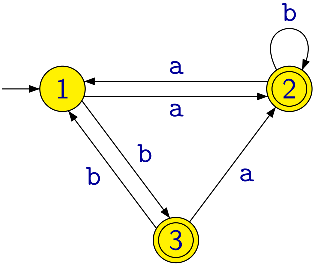
The main idea: If a state q is removed, for every pair of remaining states q_j, q_k we extend the label on a transition from q_j to q_k by a regular expression representing paths from q_j to q_k going through q.



After removing of the state q:

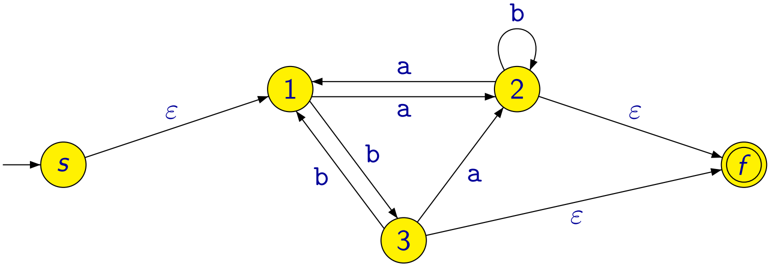


Example:



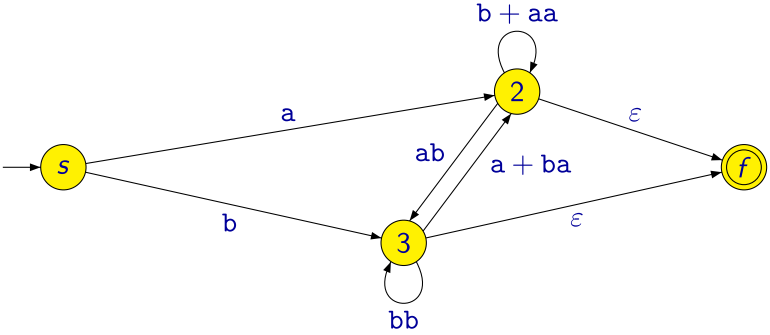
Transformation of an Automaton to a Regular Expression

Example:



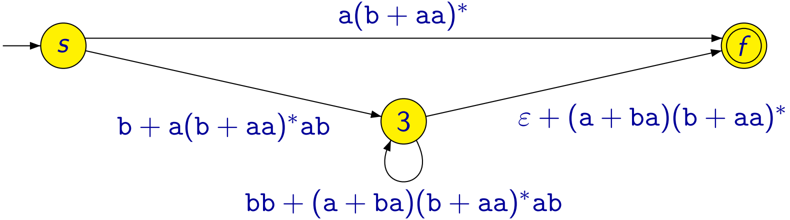
Transformation of an Automaton to a Regular Expression

Example:



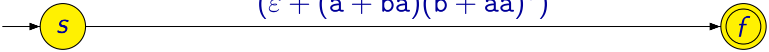
Transformation of an Automaton to a Regular Expression

Example:



Transformation of an Automaton to a Regular Expression

Example:



$$\mathtt{a}(\mathtt{b}+\mathtt{aa})^*+$$
$$(\mathtt{b}+\mathtt{a}(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})$$
$$(\mathtt{bb}+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})^*$$
$$(\varepsilon+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*)$$

Equivalence of Finite Automata and Regular Expressions

Theorem

A language is regular iff it can be represented by a regular expression.

Nonregular Languages

Nonregular Languages

Not all languages are regular.

There are languages for which there exist no finite automata accepting them.

Examples of nonregular languages:

- $L_1 = \{a^n b^n \mid n \geq 0\}$
- $L_2 = \{ww \mid w \in \{a, b\}^*\}$
- $L_3 = \{ww^R \mid w \in \{a, b\}^*\}$

Remark: The existence of nonregular languages is already apparent from the fact that there are only countably many (nonisomorphic) automata working over some alphabet Σ but there are uncountably many languages over the alphabet Σ.

Nonregular Languages

How to prove that some language L is not regular?

A language is not regular if there is no automaton (i.e., it is not possible to construct an automaton) accepting the language.

But how to prove that something does not exist?

Nonregular Languages

How to prove that some language L is not regular?

A language is not regular if there is no automaton (i.e., it is not possible to construct an automaton) accepting the language.

But how to prove that something does not exist?

The answer: By contradiction.

E.g., we can assume there is some automaton $\mathcal{A}$ accepting the language L, and show that this assumption leads to a contradiction.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.
Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Let $|Q| = n$.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Let $|Q| = n$.

Consider word $z = a^n b^n$.

Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.

Let us assume there exists a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ such that $\mathcal{L}(\mathcal{A}) = L$.

Let $|Q| = n$.

Consider word $z = a^n b^n$.

Since $z \in L$, there must be an accepting computation of the automaton $\mathcal{A}$

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

where q_0 is an initial state, and $q_{2n} \in F$.

Nonregular Languages

Consider now the first $n+1$ states of the computation

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n+1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

Nonregular Languages

Consider now the first $n+1$ states of the computation

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n+1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

It is an application of so called **pigeonhole principle**.

Pigeonhole principle

If we have $n+1$ pigeons in n holes then there is at least one hole containing at least two pigeons.

Nonregular Languages

Consider now the first $n+1$ states of the computation

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

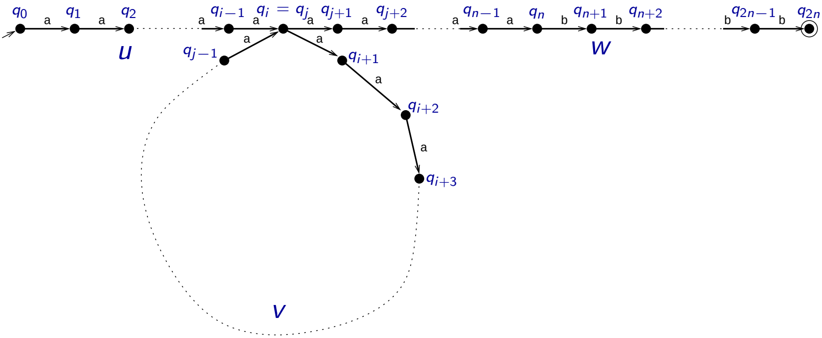
It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n+1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

I.e., there are indexes i, j such that $0 \leq i < j \leq n$ and

$$q_i = q_j$$

which means that the automaton $\mathcal{A}$ must go through a cycle when reading the symbols a in the word $z = a^n b^n$.



The word $z = a^n b^n$ can be divided into three parts u, v, w such that $z = uvw$:

$$u = a^i \qquad v = a^{j-i} \qquad w = a^{n-j} b^n$$

Nonregular Languages

For the words $u = a^i$, $v = a^{j-i}$, and $w = a^{n-j}b^n$ we have

$$q_0 \xrightarrow{u} q_i \qquad q_i \xrightarrow{v} q_j \qquad q_j \xrightarrow{w} q_{2n}$$

Let r be the length of the word v, i.e., $r = j - i$ (obviously $r > 0$, due to $i < j$).

Since $q_i = q_j$, the automaton accepts word $uw = a^{n-r}b^n$ that does not belong to L:

$$q_0 \xrightarrow{u} q_i \xrightarrow{w} q_{2n}$$

The word $uvvw = a^{n+r}b^n$, that also does not belong to L, is accepted too:

$$q_0 \xrightarrow{u} q_i \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{w} q_{2n}$$

Nonregular Languages

Similarly we can show that every word of the form $uvvvv \cdots vvw$, i.e., of the form $uv^k w$ for some $k \geq 0$, is accepted by the automaton $\mathcal{A}$:

$$q_0 \xrightarrow{u} q_i \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{v} \cdots \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{w} q_{2n}$$

A word of the form $uv^k w$ looks as follows: $a^{n-r+rk} b^n$.

Since $r > 0$, the following equivalence holds only for $k = 1$:

$$n - r + rk = n$$

This means that if $k \neq 1$ then $uv^k w$ does not belong to the language L. However, the automaton $\mathcal{A}$ accepts each such word, which is a contradiction with the assumption that $\mathcal{L}(\mathcal{A}) = \{a^n b^n \mid n \geq 0\}$.

Pumping Lemma

Let us assume that language L is accepted by some particular automaton $\mathcal{A}$, i.e., $L = \mathcal{L}(\mathcal{A})$.

Let us consider some arbitrary word $z \in L$ where $z = a_1 a_2 \cdots a_k$.

Since automaton $\mathcal{A}$ accepts word z, there must be some accepting computation of the automaton, i.e., a sequence of states:

$$q_0,\ q_1,\ q_2,\ \ldots,\ q_{k-1},\ q_k$$

of length $k + 1$ where

- q_0 is an initial state
- $q_{i-1} \xrightarrow{a_i} q_i$ for each $i \in \{1, 2, \ldots, k\}$
- q_k is an accepting state

Pumping Lemma

Let us assume that $\mathcal{A}$ has n states (i.e., $|Q| = n$), and that $|z| \geq n$.

Since $|z| = k$, the computation of automaton $\mathcal{A}$ over word z forms a sequence, whose length is at least $n+1$, that contains at most n different states:

$$q_0,\ q_1,\ q_2,\ \ldots,\ q_{k-1},\ q_k$$

It follows that there must be at least one state q that occurs at least twice in this sequence (recall the *pigeonhole principle*).

Pumping Lemma

Let us say that the repeated state occurs on positions i and j, i.e., $q_i = q_j$ where $i < j$.

$$q_0, \cdots, q_i, \cdots, q_j, \cdots, q_k$$

Remark: It is obvious that in fact we can find i and j such that $i < j \leq n$.

The word z can be divided into three parts:

$$\underbrace{a_1 \cdots a_i}_{u} \quad \underbrace{a_{i+1} \cdots a_j}_{v} \quad \underbrace{a_{j+1} \cdots a_k}_{w}$$

- $q_0 \xrightarrow{u} q_i$
- $q_i \xrightarrow{v} q_j$ (and so also $q_i \xrightarrow{v} q_i$ since $q_j = q_i$)
- $q_j \xrightarrow{w} q_k$ (and so also $q_i \xrightarrow{w} q_k$ since $q_j = q_i$)

Pumping Lemma

Consider now words:

$$\underbrace{a_1 \cdots a_i}_{u} \quad \underbrace{a_{j+1} \cdots a_k}_{w}$$

$$\underbrace{a_1 \cdots a_i}_{u} \quad \underbrace{a_{i+1} \cdots a_j}_{v} \quad \underbrace{a_{i+1} \cdots a_j}_{v} \quad \underbrace{a_{j+1} \cdots a_k}_{w}$$

$$\underbrace{a_1 \cdots a_i}_{u} \quad \underbrace{a_{i+1} \cdots a_j}_{v} \quad \underbrace{a_{i+1} \cdots a_j}_{v} \quad \underbrace{a_{i+1} \cdots a_j}_{v} \quad \underbrace{a_{j+1} \cdots a_k}_{w}$$

$$\cdots$$

It is obvious that A accepts all of them because

- $q_0 \xrightarrow{u} q_i$
- $q_i \xrightarrow{v} q_i$
- $q_i \xrightarrow{w} q_k$ where $q_k \in F$

Pumping Lemma

Pumping Lemma

If language L is regular then there exists $n \in \mathbb{N}$ such that every word $z \in L$ such that $|z| \geq n$ can be divided into subwords u, v, w such that $z = uvw$, $|uv| \leq n$, $|v| \geq 1$, and for every $i \geq 0$ it holds that $uv^i w \in L$.

Formally:

If L is regular then

$$(\exists n \in \mathbb{N})(\forall z \in L \text{ s.t. } |z| \geq n)(\exists u, v, w \text{ s.t. } z = uvw, |uv| \leq n, |v| \geq 1)$$
$$(\forall i \geq 0) : uv^i w \in L$$

Pumping Lemma

We can take the contrapositive of the pumping lemma. ($A \Rightarrow B$ is equivalent to $\neg B \Rightarrow \neg A$.)

If

$$(\forall n \in \mathbb{N})(\exists z \in L \text{ s.t. } |z| \geq n)(\forall u, v, w \text{ s.t. } z = uvw,\ |uv| \leq n,\ |v| \geq 1)$$
$$(\exists i \geq 0) : uv^i w \notin L,$$

then L is not regular.

So if we want to show that a language L is not regular, it is sufficient to show that L satisfies this condition.

Example: Let us consider laguage $L = \{a^i b^i \mid i \geq 0\}$.

- Let us assume that L is accepted by some automaton with n states.
- Let us consider word $z = a^n b^n$.
- Let us consider all possibilities how z can be divided into three subwords u, v, w satisfying conditions $|uv| \leq n$ and $|v| \geq 1$.

  It is obvious that words u and v contain only symbols a. For every particular division there are some j and k such that $j + k \leq n$, $k \geq 1$, and
  - $u = a^j$
  - $v = a^k$
  - $w = a^{n-(j+k)} b^n$
- If we choose $i = 0$, we obtain $uv^i w = uw = a^{n-k} b^n$. Since $n - k < n$, we have $uv^i w \notin L$.

Remark: Proving that some first order logic formula with alternating universal and existential quantifiers can be viewed as game played by two players, Player A and Player B.

Player A chooses values of variables bound by existential quantifiers and Player B values of variables bound by universal quantifiers.

If we want to refute the given claim, it is sufficient to find a winning strategy for Player B.

If L is regular then
$(\exists n \in \mathbb{N})(\forall z \in L \text{ s.t. } |z| \geq n)(\exists u, v, w \text{ s.t. } z = uvw,\ |uv| \leq n,\ |v| \geq 1)$
$(\forall i \geq 0) : uv^i w \in L$.

The game for Pumping Lemma looks as follows:

1. Player A chooses some $n \in \mathbb{N}$.
2. Player B chooses a word z such that $z \in L$ and $|z| \geq n$.
3. Player A chooses words u, v, w such that $z = uvw$, $|uv| \leq n$, $|v| \geq 1$.
4. Player B chooses $i \geq 0$.
5. If $uv^i w \in L$ then Player A wins. If $uv^i w \notin L$ then Player B wins.

If Player B has a winning strategy in this game then L is not regular.

Pumping Lemma

Example: $L = \{a^i b^i \mid i \geq 0\}$

1. Player A chooses $n > 0$.
2. Player B chooses $z = a^n b^n$.
3. Player A chooses words u, v, w such that $z = uvw$, $|uv| \leq n$, $|v| \geq 1$.
4. Player B chooses $i = 0$.
5. Player B wins, since no matter what Player A does, we always have $uv^i w \notin L$ because a non-empty word z occurs in the part of word z consisting only of symbols a, and when we omit it, we obtain a word of the form $a^k b^n$ where $k < n$, which does not belong to L.