Automata and Grammars - A Brief Summary

TIN071

Marta Vomlelová

marta@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~marta/brief.pdf

September 26, 2024

Chomsky Hierarchy - Automata, Languages, Grammars

Turing Machines multitape, nondeterministic	$\mathcal{L}_0$ recursively enumerable ↔	Type 0 Grammars
Linear Bounded Automata	$\mathcal{L}_1$ context sensitive ↔	Context Sensitive Grammars monotone grammars
Pushdown Automata	$\mathcal{L}_2$ context free ↔	Context Free Grammars
Finite Automata DFA, NFA, ϵNFA	$\mathcal{L}_3$ regular languages ↔	Regular (right linear) Grammars

Turing Machine

Definition (Turing Machine)

Turing Machine (TM) is the 7–tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ with the components:

- Q The finite set of **states** of the finite control.
- Σ The finite set of **input symbols**.
- Γ The complete set of **tape symbols**. Always $\Gamma \supseteq \Sigma$, $Q \cap \Gamma = \emptyset$.
- δ The partial **transition function** $(Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
  $\delta(q, X) = (p, Y, D)$, where:
  - q $\in (Q - F)$ is the current state.
  - X $\in \Gamma$ is the current tape symbol.
  - p is the next state, $p \in Q$.
  - Y $\in \Gamma$ is written in the cell being scanned, replacing anything there.
  - D $\in \{L, R\}$ is a **direction** in which the head moves (left, right).
- q_0 $\in Q$ is the **start state**.
- B $\in \Gamma \setminus \Sigma$. It appears initially in all but the finite number of initial cells that hold input symbols.
- F $\subseteq Q$ The set of **final** or **accepting** states.

- Note there are no transitions for accepting states.

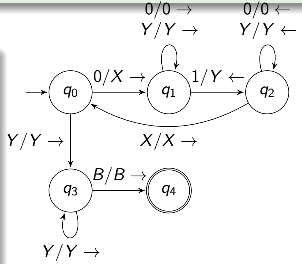
Example

A TM $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1,X,Y,B\}, \delta, q_0, B, \{q_4\})$ with δ in the table accepts the language $\{0^n1^n; n \geq 1\}$.

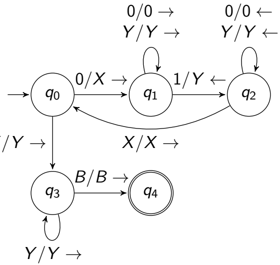
State	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

Definition (Transition diagram)

A **transition diagram** consists of a set of nodes corresponding to the states of the TM. Any arc $q \rightarrow p$ is labeled the list of items X/YD for all $\delta(q, X) = (p, Y, D)$, $D \in \{\leftarrow, \rightarrow\}$.
We assume that the blank symbol is B unless we state otherwise.



A TM for $\{0^n1^n; n \geq 1\}$



Word 0011

$q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash$

$\vdash q_2 X0Y1 \vdash Xq_0 0Y1 \vdash XXq_1 Y1 \vdash XXYq_1 1 \vdash$

$\vdash XXq_2 YY \vdash Xq_2 XYY \vdash XXq_0 YY \vdash XXYq_3 Y \vdash$

$\vdash XXYYq_3 B \vdash XXYYBq_4 B$

Word 0010

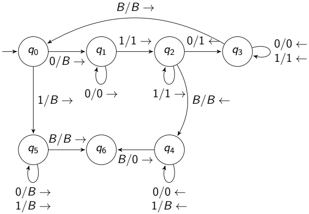
$q_0 0010 \vdash Xq_1 010 \vdash X0q_1 10 \vdash Xq_2 0Y0 \vdash q_2 X0Y0 \vdash Xq_0 0Y0 \vdash XXq_1 Y0 \vdash$

$\vdash XXYq_1 0 \vdash XXY0q_1 B$ ends up with a failure since there is no instruction for $q_1, 0$.

A TM with 'Output'

A TM that computes **monus, proper substraction** $m \dot{-} n = max(m - n, 0)$.

- $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B)$, accepting set omited (TM used for output, not acceptance).
- Start tape $0^m 1 0^n$.
- M halts with the tape $0^{m \dot{-} n}$ surrounded by blanks.
- Find leftmost 0, replace it by a blank.
- Search right, looking for a 1; continue, find 0 and replace it by 1.
- Return left.
- End if no 0 found, either left or right;
  - right: replace all 1 by B.
  - left: $m < n$: replace all 1 and 0 by B, leave the tape blank.



Compare Turing Machines and Finite Automata

- Consider a Turing machine that never writes and always moves to the right.
- Is it a finite automaton or is there some difference?

! Turing Machine does not allow transitions from the final (accepting) states.

Practicals

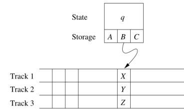
1. Design following finite automata or Turing machines. Describe them as a graph or a table.
   a) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists k \in \mathbb{N}_0)|w|_a = 3k\}$
   b) $L = \{w \mid w \in \{a,b\}^* \;\&\; [(\exists k \in \mathbb{N}_0)|w|_a = 3k \vee (\exists \ell \in \mathbb{N}_0)|w|_a = 2\ell]\}$
   c) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists k \in \mathbb{N}_0)|w|_a = 3k \;\&\; (\exists \ell \in \mathbb{N}_0)|w|_a = 2\ell\}$
   d) $L = \{w \mid w \in \{a,b\}^* \;\&\; [(\exists k \in \mathbb{N}_0)|w|_a = 3k \vee (\exists \ell \in \mathbb{N}_0)|w|_b = 2\ell]\}$
   e) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists k \in \mathbb{N}_0)|w|_a = 3k \;\&\; (\exists \ell \in \mathbb{N}_0)|w|_b = 2\ell\}$
   f) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists k \in \mathbb{N}_0)|w|_a = 3k \;\&\; (\forall \ell \in \mathbb{N}_0)|w|_b \neq 2\ell\}$
   g) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists k \in \mathbb{N}_0)|w|_a = 2k\}$
2. Design an automaton accepting words that contain a given substring:
   a) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists u \in \{a,b\}^*)w = abba.u\}$ Starts with *abba*.
   b) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists u \in \{a,b\}^*)w = u.abba\}$ Ends with *abba*.
   c) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists u, v \in \{a,b\}^*)w = u.abba.v\}$ Has *abba* as a substring.
   d) $L = \{w \mid w \in \{a,b\}^* \;\&\; (\exists u \in \{a,b\}^*)w = u.ab \;\&\; (\exists k \in \mathbb{N}_0)|w| = 3k+1\}$
   e) $L = \{w \mid w \in \{a,b\}^* \;\&\; [(\exists u \in \{a,b\}^*)w = u.ab \;\vee\; (\exists k \in \mathbb{N}_0)|w| = 3k+1]\}$.
3. For a number in the binary encoding, design an automaton accepting:
   a) $L = \{w \mid w \in \{0,1\}^* \;\&\; (\exists k \in \mathbb{N}_0)w = 3k\}$.

Storage in the FA unit

- Storage in the State
- Consider state as a tuple
- $M = (\{q_0, q_1\} \times \{0, 1, B\}, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, \{[q_1, B]\})$
- $L(M) = (01^* + 10^*)$,

δ	0	1	B
$\rightarrow [q_0, B]$	$([q_1, 0], 0, R)$	$([q_1, 1], 1, R)$	
$[q_1, 0]$		$([q_1, 0], 1, R)$	$([q_1, B], B, R)$
$[q_1, 1]$	$([q_1, 1], 0, R)$		$([q_1, B], B, R)$
$*[q_1, B]$			

Multiple Tracks



- $L_{wcw} = \{wcw | w \in (\mathbf{0} + \mathbf{1})^+\}$,
- $M = (\{q_0, \dots, q_9\} \times \{0, 1, B\}, \{[B, 0], [B, 1], [B, c]\}, \{B, *\} \times \{0, 1, B, c\}, \delta, [q_1, B], [B, B], \{[q_9, B]\})$
- δ is defined as $(a, b \in \{0, 1\})$:
  - $\delta([q_1, B], [B, a]) = ([q_2, a], [*, a], R)$ picks up the symbol a
  - $\delta([q_2, a], [B, b]) = ([q_2, a], [B, b], R)$ move right, look for c,
  - $\delta([q_2, a], [B, c]) = ([q_3, a], [B, c], R)$ continue right, the state changed,
  - $\delta([q_3, a], [*, b]) = ([q_3, a], [*, b], R)$ continue right,
  - $\delta([q_3, a], [B, a]) = ([q_4, B], [*, a], L)$ check correct, drop memory and go left,
  - $\delta([q_4, B], [*, a]) = ([q_4, B], [*, a], L)$ go left,
  - $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$ c found, continue left,
  - decide whether all inputs left and right are checked, branch adequately
  - $\delta([q_5, B], [B, a]) = ([q_6, B], [B, a], L)$ left symbol unchecked,
  - $\delta([q_6, B], [B, a]) = ([q_6, B], [B, a], L)$ proceed left,
  - $\delta([q_6, B], [*, a]) = ([q_1, B], [*, a], R)$ start again,
  - $\delta([q_5, B], [*, a]) = ([q_7, B], [*, a], R)$ symbol left from c checked, go right,
  - $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$ proceed right,
  - $\delta([q_8, B], [*, a]) = ([q_8, B], [*, a], R)$ proceed right,
  - $\delta([q_8, B], [B, B]) = ([q_8, B], [B, B], R)$ accept.

Practicals

11. Find a Turing machine accepting the language
a) $L = \{wcw^R \mid w \in \{0,1\}^*\}$
b) $L = \{a^i b^i c^i \mid i = 0, 1, 2, \ldots\}$
c) $L = \{a^i b^j c^k \mid i, j, k = 0, 1, 2, \ldots \& i \leq j \leq k\}$

Multi-tape Turing Machine

Definition (Multi-tape Turing Machine)

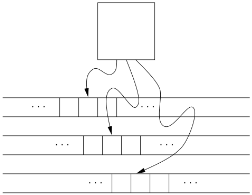
Initial position

- input on the first tape, other tapes completely blank
- first head on the first input letter, other heads anywhere
- initial state

One step of a multi-tape TM

- new state
- each tape writes its new symbol
- each head independently moves left, right, or does not move.

Multitape TM



Theorem (Multitape TM)

Any language accepted by a Multitape TM can be accepted also with some (standard) Turing machine.

Decidable (Recursive) Languages

Definition (Semi-Decidable, Recursively enumerable languages (RE))

The set of languages that we can accept by a Turing machine.

Definition

Turing Machine halts A TM **halts** if it enters a state q, scanning a tape symbol X, and there is no move in this situation, i.e., $\delta(q, X)$ is undefined.

- We assume that a TM always halt when it is in an accepting state.
- We can require that a TM halts even if it does not accept only for **recursive** languages, a proper subset of recursively enumerable languages.

Definition (Decidable, Recursive languages (R))

The language $L \subseteq \Sigma^*$ is called **decidable** if there exists a Turing machine that halts on any string $\in \Sigma^*$ and accepts the language L.

A Language That Is Not Recursively Enumerable

We construct the language consisting of pairs (M, w) such that:

- M is a TM (binary coded) with input alphabet $\{0, 1\}$,
- w is a string of 0's and 1's, and
- M accepts input w.

Our plan:

- Encode TM's by binary code regardless of how many states the TM has.
- Treat TM as a binary string.
- If a string is not well formed, think of it as a TM with no moves. Therefore, every binary string represents some TM.
- **Diagonalization language L_d**;
  $L_d = \{w;$ TM represented as w that **does not accept** $w\}$.
- There does not exists a TM recognizing the language L_d. Running it on its own code leads to the paradox.

Codes for Strings, Turing Machines

We call w_i the i–th string, where ϵ is the first string, 0 the second, 1 the third, 00 the fourth and so on.
Strings are ordered by length, equal length are ordered lexicographically.

- To represent a TM $M = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ as a binary string, we must first assign integers to the states, tape symbols, and directions L, R.
- Assume:
  - Start state is always q_1.
  - Always is q_2 the only accepting state (we do not need more, TM halts).
  - First symbol is always 0, the second 1, the third B, the blank. Other tape symbols can be assigned arbitrarily.
  - Direction L is 1, direction R is 2.
- One transaction $\delta(q_i, X_j) = (q_k, X_l, D_m)$ is coded: $0^i10^j10^k10^l10^m$. Notice all $i, j, k, l, m \geq 1$ so no substring 11 occurs here.
- The entire TM consists of all the codes for transaction in some order, separated by pair of 1's: $C_1 11 C_2 11 \ldots C_{n-1} 11 C_n$.

TM encoding example

Turing Machine

$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$

δ	0	1	B
$\to q_1$		$(q_3, 0, R)$	
$*q_2$			
q_3	$(q_1, 1, R)$	$(q_2, 0, R)$	$(q_3, 1, L)$.

- The code for transitions

C_1	C_2	C_3	C_4
0100100010100	0001010100100	00010010010100	0001000100010010

- The overall TM code:
  01001000101001100010101001001100010010010100110001000100010010.

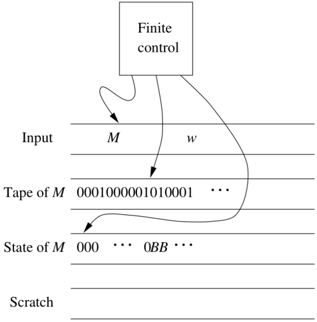
The Universal Language

Definition (The Universal Language)

We define L_u, the **universal language**, the set of binary strings that encode a pair (M, w), such that M is a TM and $w \in L(M)$, that is
$L_u = \{(M, w) : \text{TM M accepts } w\}$.

Theorem (The Universal Turing Machine)

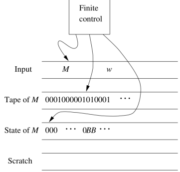
There is a TM U, called the **universal Turing machine**, *such that $L_u = L(U)$.*



We describe U as a mutlitape Turing machine.

- Transactions of M are stored initially on the first tape, along with the string w, separated by 111.
- Second tape holds the simulated tape of M, using format as code of M, i.e. symbols 0^i separated by 1's.
- Third tape holds the state of M represented by i 0's.

Operations of the Universal Turing Machine



The operation of U can be summarized as follows:

- Examine the input whether the code for M is legitimate; if not, U halts without accepting.
- Initialize the second tape with w in its encoded form: 10 for 0 in w, 100 for 1; blanks are left blank and replaced with 1000 only 'on demand'.
- Place 0, the start state of M, on the third tape. Move the head on the second tape to the first simulated cell.
- To simulate a move of M
  - Search on the first tape for a proper transition $0^i10^j10^k10^l10^m$, 0^i on tape 3, 0^j on tape 2.
  - Change the content of tape 3 to 0^k.
  - Replace 0^j on tape 2 by 0^l. Use scratch tape to manage the spacing.
  - Move the head on tape 2 to the position of the next 1 to the left or right, depending on m.
- If M has no transition that matches the simulated state and tape symbol, halt.
- If M enters its accepting state, then U accepts. □

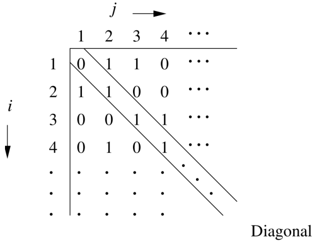
Definition (The Diagonal Language)

The Diagonal Language L_d is defined as
$L_d = \{w;$ TM represented as w that does not accept $w\}$.

$$L_d = \{w; diagonal(w) = 0\}$$

Theorem

L_d is not recursively enumerable language. That is, there is no TM that accepts L_d.



Proof.

- Assume L_d is RE, $L_d = L(M)$ for some TM M.
- It has the language $\{0, 1\}$, so it is in the list in the figure: 'Does TM M_i accept input string w_j?'
- There is at least one code for it, say i, $M = M_i$.
- Is $w_i \in L_d$
  - 'Yes' imply $diagonal(w_i) = 0$, therefore $w_i \notin L(M_i)$. Contradiction $L(M_i) = L_d$.
  - 'No' imply $diagonal(w_i) = 1$, therefore $w_i \in L(M_i)$. Contradiction $L(M_i) = L_d$.

Therefore, such M does not exist. L_d is not RE. □

Recursive Languages

Definition (TM halts)

TM **halts** iff it enters a state q, reading X, and there is no instruction for this situation, that is $\delta(q, X)$ is undefined.

- We assume TM halts in any accepting state $q \in F$,
- We are not sure whether it accepts until TM halts.

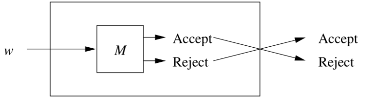
Definition (Recursive languages, Decidable problems)

- We say that a TM M **decides a language** L iff $L = L(M)$ and for any $w \in \Sigma^*$ the TM with the input w halts.
- For a computational problem with yes/no answer, we say it is a **decidable problem** iff there exists a computer program that always halts and gives the correct answer.
- **Recursive languages** are such languages, for those there exists a TM M that decides the language.

Complements of Recursive and RE languages

Theorem

If L is a recursive language, so is $\overline{L}$.



Proof.

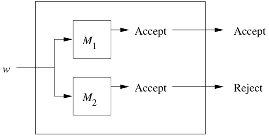
- $L = L(M)$ for some TM M that always halts.
- We construct TM $\overline{M}$ such that $\overline{L} = L(\overline{M})$.
- Accepting states of M are non-accepting in $\overline{M}$ without any transaction out of them.
- $\overline{M}$ has a new accepting state r; no transition from r.
- For each non–accepting state of M and each tape symbol such that M has no transition, add a transition to the accepting state r.
- Since M is guaranteed to halt, $\overline{M}$ is also guaranteed to halt.
- $\overline{M}$ accepts $\overline{L}$.

□

$L\&\overline{L} \in RE \Rightarrow L, \overline{L}$ is recursive

Theorem (Post Theorem)

A language L is recursive iff both L and $\overline{L}$ (the complement) are recursively enumerable.



Proof:

- We have TM $L = L(M_1)$ and $\overline{L} = L(M_2)$.
- for the word w we simulate both M_1 and M_2 (two tapes, states with two components).
- If any M_i accepts, M halts and answers.
- Languages are complementary, one of M_i's always halts, therefore L is recursive. □

Theorem

If L is recursive, so is also $\overline{L}$.

Undecidability of the Universal Language

Theorem (Undecidability of the Universal Language)

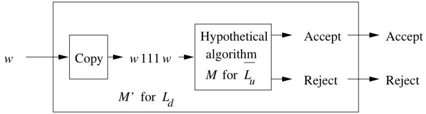
L_u is RE but not recursive.

Proof.

- We have proved that L_u is RE.
- Suppose L_u were recursive.
- Then, $\overline{L_u}$ would also be recursive.
- If we have TM to accept $\overline{L_u}$, then we can costruct a TM to accept L_d (see right).
- Since we already know L_d is not RE, $\overline{L_u}$ is not RE and L_u is not recursive.

□

Modification of TM for $\overline{L_u}$ to TM for L_d:



- Given string w, change it to $w111w$ (2–tapes, convert to 1–tape).
- Simulate M on the new input. Accept iff M accepts.
- Choose i s.t. $w_i = w$. Previous line accepts $\overline{L_u}$, that is cases where M_i does not accept w_i, that is the language L_d.

Further reading

- These slides: https://ktiml.mff.cuni.cz/~marta/brief.pdf
- Literature: J.E. Hopcroft, R. Motwani, J.D. Ullman: *Introduction to Automata Theory, Languages, and Computations*, Addison–Wesley

Let us have a 10 minutes break.

Chomsky Hierarchy

Automata		Grammars
Turing Machines multitape, nondeterministic	$\xleftrightarrow{\mathcal{L}_0}$	Type 0 grammars
Linear bounded automata	$\xleftrightarrow{\mathcal{L}_1}$	Context sensitive grammars Monotone grammars
Pushdown automata	$\xleftrightarrow{\mathcal{L}_2}$	Context free grammars
Finite automata DFA, NFA, ϵNFA	$\xleftrightarrow{\mathcal{L}_3}$	Regular (right linear) grammars

Theorem: $\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$.

Palindrome example

- A string the same forward and backward, like `otto` or `Madam, I'm Adam`.
- w is a palindrome iff $w = w^R$.
- The language L_{pal} of palindromes is not a regular language.
  - We use the pumping lemma.
  - If L_{pal} is a regular language, let n be the asssociated constant, and consider: $w = 0^n 1 0^n$.
  - For regular L, we can break $w = xyz$ such that y consists of one or more 0's from the first group. Thus, xz would be also in L_{pal} if L_{pal} were regular.
- A context-free grammar (right) consists of one or more variables, that represent classes of strings, i.e., languages.

A context-free grammar for palindromes

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

Definition (Grammar)

A Grammar $G = (V, T, P, S)$ consists of

- Finite set of **terminal symbols** (**terminals**) T, like $\{0, 1\}$ in the previous example.
- Finite set of **variables** V (**nonterminals**,**syntactic categories**), like $\{P\}$ in the previous example.
- **Start symbol** S is a variable that represents the language being defined. P in the previous example.
- Finite set of **rules** (**productions**) P that represent the recursive definition of the language. Each has the form:
  - $\alpha A\beta \to \omega$, $A \in V, \alpha, \beta, \omega \in (V \cup T)^*$

    notice the left side (head) contains at least one variable.

  The **head** - the left side, the production symbol $\to$, the **body** - the right side.

Definition (Context free grammar CFG)

Context free grammar (CFG) je $G = (V, T, P, S)$ has only productions of the form

$A \to \alpha$, $A \in V, \alpha \in (V \cup T)^*$.

Chomsky hierarchy

- Grammar types according to productions allowed.
- **Type 0** (recursively enumerable languages $\mathcal{L}_0$)
  general rules $\alpha \to \beta$, $\alpha, \beta \in (V \cup T)^*, \alpha$ contains at least one variable
- **Type 1** (context sensitive languages $\mathcal{L}_1$)
  - productions of the form $\alpha A \beta \to \alpha \omega \beta$
    $A \in V, \alpha, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$
  - with only exception $S \to \epsilon$, then S does not appear at the right side of any production
- **Type 2** (context free languages $\mathcal{L}_2$)
  productions of the form $A \to \omega, A \in V, \omega \in (V \cup T)^*$
- **Type 3** (regular (right linear) languages $\mathcal{L}_3$)
  productions of the form $A \to \omega B, A \to \omega, A, B \in V, \omega \in T^*$

Practicals

9. Find grammars that generate following languages.

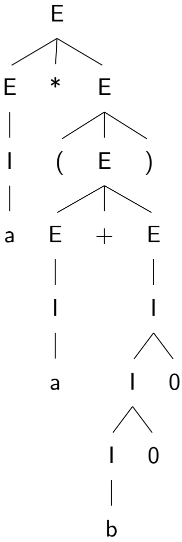
a) $L = \{ww^R \mid w \in \{a, b\}^*\}$
b) $L = \{a^i b^i \mid i = 0, 1, 2, \ldots\}$
c) $L = \{a^i b^j \mid i, j = 0, 1, 2, \ldots\}$
d) $L = \{a^i a^i b^j \mid i, j = 0, 1, 2, \ldots\}$
e) $L = \{a^i b^j a^i \mid i, j = 0, 1, 2, \ldots\}$
f) $L = \{a^i b^i a^j \mid i, j = 0, 1, 2, \ldots\}$
g) $L = \{a^i b^j a^k \mid i, j, k \in \mathbb{N}\}$
h) $L = \{a^{2i} \mid i = 0, 1, 2, \ldots\}$
i) $L = \{a^{3i} \mid i = 0, 1, 2, \ldots\}$
k) *$L = \{a^i b^i c^i \mid i = 0, 1, 2, \ldots\}$
l) *$L = \{a^i b^j c^k \mid i, j, k \in \mathbb{N} \& i \leq j \leq k\}$
m) HTML syntax analyzer

<p>	</p>
<a>	
<table>	</table>
<tr>	</tr>
<td>	</td>

without any attributes.

n) proper parenthesis, that is the same number of the left and the right ones, never more right ones than left. The word '()(())' is in the language.
o) arithmetic expression for $a, +, *, (,)$.

Parse Tree



CFG for simple expressions

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$

5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Normal Forms for Context-Free Grammars

- Chomsky Normal Form: all production are of the form $A \to BC$ or $A \to a$, A, B, C where are variables, a is a terminal.

  With an additional rule $S \to \epsilon$ to generate the empty string with the condition that the S is the start symbol that does not appear in the body of any rule.
- Every CFL is generated by a CFG in Chomsky Normal Form.

To get there, we perform simplifications

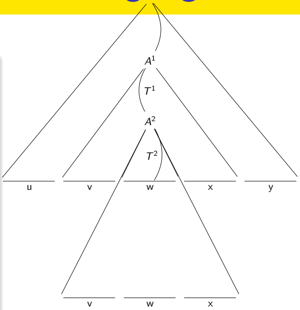
- Eliminate *useless symbols*
- eliminate *ϵ-productions* $A \to \epsilon$ for some variable A
- eliminate *unit productions* $A \to B$ for variables A, B.

Pumping Lemma for Context Free Languages

Theorem (Pumping Lemma for Context Free Languages)

Let L be a CFL. Then there exists a constant $n \in \mathbb{N}$ such that any $z \in L, |z| > n$ can be written $z = uvwxy$ subject to:

- $|vwx| \leq n$.
- $vx \neq \epsilon$.
- $\forall i \geq 0,\ uv^iwx^iy \in L$.



Proof Idea:

- take the parse tree for z
- find the longest path
- there must be two equal variables
- these variables define two subtrees
- the subtrees define partition of $z = uvwxy$
- we can move the tree T^1 $(i > 1)$
- or replace T^1 by T^2 $(i = 0)$

Proof: $|z| > p : z = uvwxy, |vwx| \leq q, vx \neq \epsilon, \forall i \geq 0 uv^iwx^iy \in L$

- we take the grammar in Chomsky NF (for $L = \{\epsilon\}$ and $\emptyset$ aside).
- Let $|V| = k$. We set $n = 2^k$.
- For $z \in L$,$|z| \geq n$, the parse tree has a path z of length $> k$

  we denote the terminal of the longest path t
- At least two of the last k variables on the path to t are equal
- we take the couple A^1, A^2 closest to t (it defines subtrees T^1, T^2)
- the path from A^1 to t is the longest in T^1 and the length is maximally $k + 1$

  the yield of T^1 is no longer than 2^k (so $|vwx| \leq n$)
- there are two paths from A^1 (ChNF), one to T^2 other to the rest of vx

  ChNF not nullable, so $vx \neq \epsilon$
- derivation of the word $(A^1 \Rightarrow^* vA^2x, A^2 \Rightarrow^* w)$

  $S \Rightarrow^* uA^1y \Rightarrow^* uvA^2xy \Rightarrow^* uvwxy$
- if we move A^2 to A^1 $(i = 0)$

  $S \Rightarrow^* uA^2y \Rightarrow^* uwy$
- if we move A^1 to A^2 $(i = 2, 3, \ldots)$

  $S \Rightarrow^* uA^1y \Rightarrow^* uvA^1xy \Rightarrow^* uvvA^2xxy \Rightarrow^* uvvwxxy$

□

Applications of the Pumping Lemma for CFL's

"Adversary game" as for regular languages:

- Pick a language L that is not CFL.
- Our 'adversary' gets to pick n, which we do not know.
- We get to pick z, and we may use n as a parameter.
- Our adversary gets to break z into $uvwxy$, subject $|vwx| \leq n$ and $vx \neq \epsilon$.
- We 'win' the game, if by picking i and showing $uv^i wx^i y$ is not in L.

Lemma (Not CFL)

Following languages are not CFL:

- $\{0^i 1^i 2^i | i \geq 1\}$
- $\{0^i 1^j 2^i 3^j | i \geq 1 \& j \geq 1\}$
- $\{ww | w \textit{ is in } \{0,1\}^*\}$

Pumping Lemma Usage

Example (non CFL)

Following language is not CFL

- $\{0^i 1^i 2^i | i \geq 1\}$

- assume it were CFL
- we get n from the Pumping Lemma
- then $|0^n 1^n 2^n| > n$
- the middle part vwx is not longer then n
- we pump at most two different symbols
- the equality of symbols is violated – CONTRADICTION.

Example (not a CFL)

Following language is not CFL

- $\{0^i 1^j 2^k | 0 \leq i \leq j \leq k\}$

- assume it were CFL
- we get n from the Pumping Lemma
- then $|0^n 1^n 2^n| > n$
- the middle part vwx is not longer then n
- we pump at most two different symbols
- in the case of a (or b), pump up – CONTRADICTION $i \leq j$ (or $j \leq k$)
- if c (or b), pump down – CONTRADICTION $j \leq k$ (or $i \leq j$)

Pumping lemma example

Example (non context free language)

The following language is not context free:

- $\{0^i 1^j 2^i 3^j | i, j \geq 1\}$

- proof by contradiction: assume it is CFL
- we take n from the Pumping lemma
- then $|0^n 1^n 2^n 3^n| > p$
- the middle section must not be longer than n
- it always covers one or two different symbols
- the equality of 0's and 2's or 1's and 3's is violated – CONTRADICTION

Example (non context free language)

The following language is not context free:

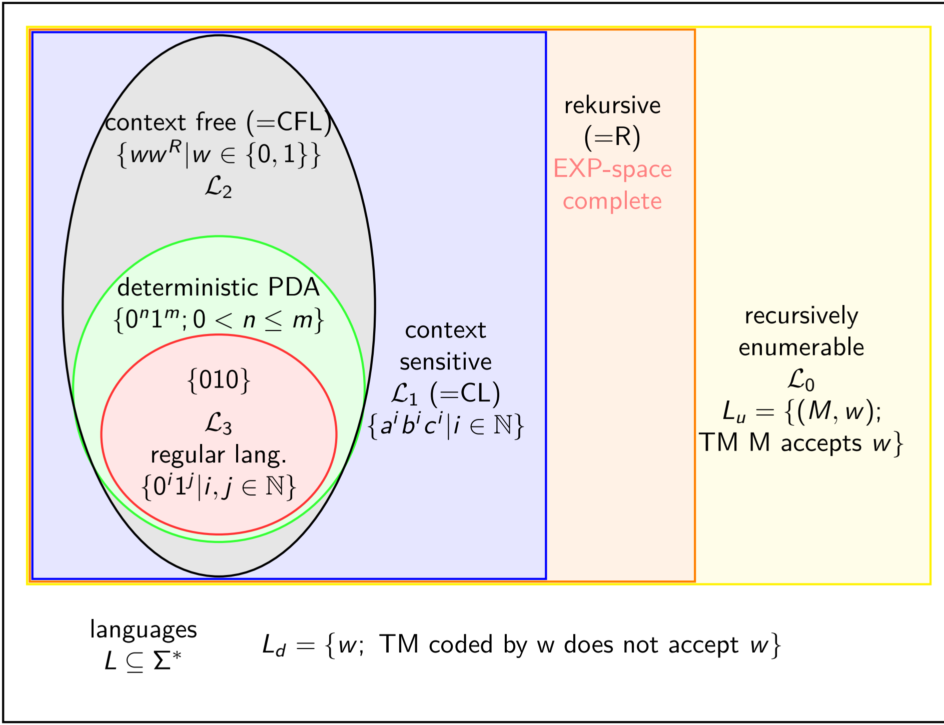
- $\{ww | w \text{ is in } \{0, 1\}^*\}$

- proof by contradiction: assume it is CFL
- we take n from the Pumping lemma
- then $|0^n 1^n 0^n 1^n| > n$
- the inner section must not be longer than q
- it always covers one or two different symbols
- the equality of 0's and 1's is violated – CONTRADICTION

Practicals Pumping lemma

10. Are following languages context-free?

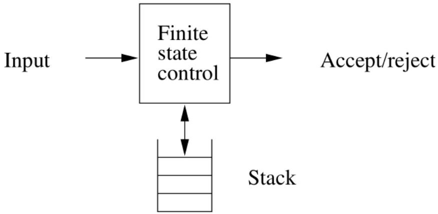
a) $L = \{ww \mid w \in \{a, b\}^*\}$
b) $L = \{a^i b^i \mid i = 0, 1, 2, \ldots\}$
c) $L = \{a^i b^j a^i \mid i, j = 0, 1, 2, \ldots\}$
d) $L = \{a^i b^j a^k \mid i, j, k = 0, 1, 2, \ldots\}$
e) $L = \{ww^R \mid w \in \{a, b\}^*\}$
f) $L = \{ww^R \mid w \in \{a, b\}^* \& |w|_a = |w|_b\}$
g) $L = \{a^{i^2} \mid i = 0, 1, 2, \ldots\}$
h) $L = \{a^{i^2+i+1} \mid i = 0, 1, 2, \ldots\}$


context free (=CFL)
$\{ww^R | w \in \{0,1\}\}$
$\mathcal{L}_2$
deterministic PDA
$\{0^n 1^m; 0 < n \leq m\}$
$\{010\}$
$\mathcal{L}_3$
regular lang.
$\{0^i 1^j | i, j \in \mathbb{N}\}$
context
sensitive
$\mathcal{L}_1$ (=CL)
$\{a^i b^i c^i | i \in \mathbb{N}\}$
rekursive
(=R)
EXP-space
complete
recursively
enumerable
$\mathcal{L}_0$
$L_u = \{(M, w);$
TM M accepts $w\}$
languages
$L \subseteq \Sigma^*$
$L_d = \{w;$ TM coded by w does not accept $w\}$

Pushdown Automata

If you want to decide: More a or b in any sequence $\in \{a, b\}^*$.

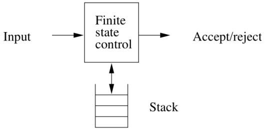
- Pushdown automata is an extension of the ϵ–NFA.
- The additional feature is the stack. **Stack** can be read, pushed, and popped only at the top.
- It can remember an infinite amount of information.
- Pushdown automata define context-free languages.
- Deterministic pushdown automata accept only a proper subset of the CFL's.



A pushdown automaton.

In one transition, the pushdown automaton:

- Consumes from the input zero or one symbol. (ϵ transitions for zero input.)
- Goes to a new state.
- Replaces the symbol at the top of the stack by any string (ϵ corresponds to pop, replace top symbol, push more symbols).



Example

PDA for the language ww^R: $L_{wwr} = \{ww^R | w \in (\mathbf{0} + \mathbf{1})^*\}$.

A PDA accepting L_{wwr}:

- Start q_0 represents a guess that we have not yet seen the middle.
- At any time, non–deterministically guess
  - Stay q_0 (not yet in the middle).
  - Spontaneously go to state q_1 (we have seen the middle).
- In q_0, read the input symbol and push it onto the stack.
- In q_1, compare the input symbol with the one on top of the stack. If they match, consume the input symbol and pop the stack.
- If we empty the stack, w accept the input that was read up to this point.

Pushdown Automata (PDA)

Definition (Pushdown Automata)

A pushdown automaton (PDA) is $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where

- Q A finite set of states.
- Σ A finite set of input symbols.
- Γ A finite stack alphabet.
- δ The transition function. $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P_{FIN}(Q \times \Gamma^*)$, $(q, a, X) = (p, \gamma)$ where p is the new state and γ a string of stack symbols that replace X on top of the stack.
- q_0 The start state.
- Z_0 The start symbol. The only symbol on the stack at the beginning.
- F The set of accepting (final) states.

PDA for L_{wwr}

Example (PDA for L_{wwr})

PDA for L_{wwr} can be described $P = (\{q_0, q_1, q_2, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\}\})$ where δ is defined:

$\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$ $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$	Push the input on stack, leave the start symbol there.
$\delta(q_0, 0, 0) = \{q_0, 00\}$ $\delta(q_0, 0, 1) = \{q_0, 01\}$ $\delta(q_0, 1, 0) = \{q_0, 10\}$ $\delta(q_0, 1, 1) = \{q_0, 11\}$	Stay in q_0, read the input and push it onto stack.
$\delta(q_0, \epsilon, Z_0) = \{q_1, Z_0\}$ $\delta(q_0, \epsilon, 0) = \{q_1, 0\}$ $\delta(q_0, \epsilon, 1) = \{q_1, 1\}$	Spontaneous transition to q_1, no change on stack.
$\delta(q_1, 0, 0) = \{q_1, \epsilon\}$ $\delta(q_1, 1, 1) = \{q_1, \epsilon\}$	State q_1 matches the input and the stack symbols.
$\delta(q_1, \epsilon, Z_0) = \{q_2, Z_0\}$	We have found ww^R and go to the accepting state.

A Graphical Notation for PDA's

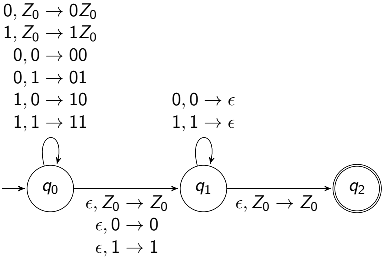
Definition (Transition diagram for PDA)

A transition diagram for PDA contains:

- The nodes correspond to the states of the PDA.
- The first arrow indicates the start state, and doubly circled states are accepting.
- The arc correspond to transitions of the PDA. An arc labeled $a, X \rightarrow \alpha$ from state q to p means that $\delta(q, a, X) \ni (p, \alpha)$.
- Conventionally, the start stack symbol is Z_0.

Labels:

input_symbol, stack_symbol → string_to_push



Definition (PDA configuration)

We represent the configuration of a PDA by a triple (q, w, γ), where

- q is the state
- w is the remaining input and
- γ is the stack contents (top on the left).

Such a tripple is called an **instantaneous description (ID)** of the pushdown automaton.

Definition ($\vdash, \vdash^*$ Sequences of instantaneous descriptions)

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Define $\vdash_P$ or just $\vdash$ as follows. Suppose $\delta(q, a, X) \ni (p, \alpha)$. Then for all strings $w \in \Sigma^*$ and $\beta \in \Gamma^*$:

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta).$$

We also use the symbol $\vdash_P^*$ or $\vdash^*$ to represent zero or more moves of the PDA, i.e.

- $I \vdash^* I$ for any ID I
- $I \vdash^* J$ if there exists some ID K such that $I \vdash K$ and $K \vdash^* J$.

ID's of the PDA on input 1111

$$
\begin{tikzcd}[column sep=tiny]
 & & & & & & & (q_0, 1111, Z_0) \arrow[dll] \arrow[dr] & \\
 & & & & & (q_0, 111, 1Z_0) \arrow[dll] \arrow[dr] & & & (q_1, 1111, Z_0) \arrow[d] \\
 & & & (q_0, 11, 11Z_0) \arrow[dll] \arrow[dr] & & & (q_1, 111, 1Z_0) \arrow[d] & & (q_2, 1111, Z_0) \\
 & (q_0, 1, 111Z_0) \arrow[dl] \arrow[dr] & & & (q_1, 11, 11Z_0) \arrow[d] & & (q_1, 11, Z_0) \arrow[d] & & \\
(q_0, \epsilon, 1111Z_0) \arrow[d] & & (q_1, 1, 111Z_0) \arrow[d] & & (q_1, 1, 1Z_0) \arrow[d] & & (q_2, 11, Z_0) & & \\
(q_1, \epsilon, 1111Z_0) & & (q_1, \epsilon, 11Z_0) & & (q_1, \epsilon, Z_0) \arrow[d] & & & & \\
 & & & & \boxed{(q_2, \epsilon, Z_0)} & & & &
\end{tikzcd}
$$

The Languages of a PDA

Definition (PDA language accepted by final state)

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Then $L(P)$, the **language accepted by F by final state**, is $\{w | (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \alpha)$ for some $q \in F$ and any stack string $\alpha\}$.

Example

The PDA example for L_{wwr} accepts the language.

- (IF) For any $x = ww^R$, we have a accepting computation

  $$(q_0, ww^R, Z_0) \vdash^* (q_0, w^R, w^R Z_0) \vdash (q_1, w^R, w^R Z_0) \vdash^* (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0).$$

- (Only If)
  - The only way to enter q_2 is from q_1 and Z_0 at the top of the stack.
  - Any accepting computation strats in q_0, changes to q_1 and never returns to q_0.
  - We prove $(q_0, x, Z_0) \vdash^* (q_1, \epsilon, Z_0)$ exactly for the strings of the form $x = ww^R$. Proof by induction on $|x|$ in the book p.235.

Practicals Pushdown automata

8. Desing pushdown automata for the following languages:

a) $L_1 = \{w2w^R \mid w \in \{0,1\}^*\}$
b) $L_2 = \{ww^R \mid w \in \{0,1\}^*\}$
c) $L_3 = \{w \mid w \in \{0,1\}^* \& |w|_0 = |w|_1\}$
d) $L_4 = \{u2v \mid u, v \in \{0,1\}^* \& |u| \neq |v|\}$
e) $L_i = \{u2v \mid u, v \in \{0,1\}^* \& u[i] \neq v[i]\}$
f) $L_5 = \{u2v \mid u, v \in \{0,1\}^* \& u^R \neq v\}$
g) $L_6 = \{a^i b^j c^{i+j} \mid i, j = 0, 1, 2, \ldots\}$

Further reading

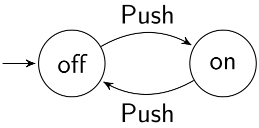
- These slides: https://ktiml.mff.cuni.cz/~marta/brief.pdf
- Literature: J.E. Hopcroft, R. Motwani, J.D. Ullman: *Introduction to Automata Theory, Languages, and Computations*, Addison–Wesley

Thank you for attention.

Finite Automata

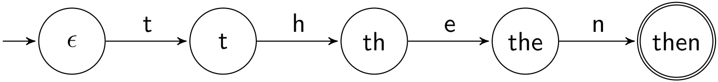
- Software for designing and checking the behavior of digital circuits.

A Finite automaton modeling an on/off switch.



- Lexical analyzer, web page analyzer.

A finite automaton modeling recognition of `then`.



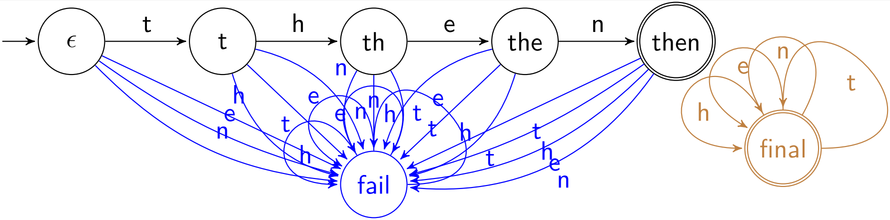
Definition (Deterministic Finite Automata)

A **deterministic finite automation (DFA)** $A = (Q, \Sigma, \delta, q_0, F)$ consists of:

- A finite set of **states**, often denoted Q.
- A finite set of **input symbols**, denoted Σ.
- A **transition function** $Q \times \Sigma \to Q$, denoted δ, represented by arcs.
- A **start state** $q_0 \in Q$.
- A **set accepting states** (final states) $F \subseteq Q$.

Convention: If some transitions are missing, we add a new state *fail* and make the transition δ total by adding edges to *fail* for any 'undefined' pair q, s.

If the set F is empty, we add to F and Q a new state *final*, with no transitions from other states, just 'staying in final' for any $s \in \Sigma$: $\delta(final, s) = final$.

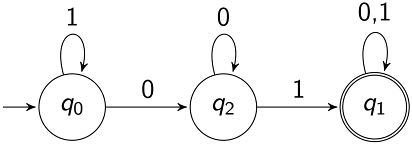


Deterministic Finite Automata Description

Example

An automaton A that accepts $L = \{x01y : x, y \in \{0,1\}^*\}$.

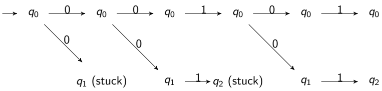
- State diagram (graph) Automaton $A = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_1\})$.



- table
  - rows: states + transitions
  - columns: letters from the input alphabet Σ

δ	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

- State tree
  - nodes = states
  - edges = transitions
  - *only reachable states*
  - we need it only for nondeterministic FA.

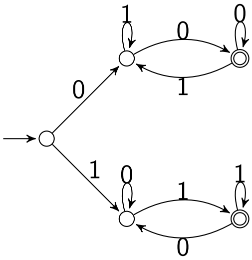


Finite Automata, Regular Languages

Definition (Word, language, Σ^*)

- $\Sigma^* = \{\epsilon\} \cup \Sigma \cup \Sigma.\Sigma \cup \ldots \Sigma^n \cup \ldots$
- a word is an element $w \in \Sigma^*$, a language is a subset $L \subseteq \Sigma^*$.

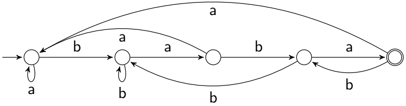
- **Deterministic Finite Automaton (DFA)** $A = (Q, \Sigma, \delta, q_0, F)$.
- **Language accepted (recognized) by a DFA** $A = (Q, \Sigma, \delta, q_0, F)$ is the language $L(A) = \{w | w \in \Sigma^* \ \& \ \delta^*(q_0, w) \in F\}$.
- Language L is **recognizable** by a DFA, if there exists DFA A such that $L = L(A)$.
- The class of languages recognizable by a DFA $\mathcal{F}$ is called **regular languages**.



Regular Languages Examples

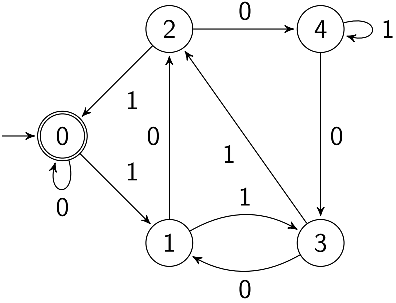
Example (Regular Language)

- $L = \{w \mid w = ubaba, w \in \{a, b\}^*, u \in \{a, b\}^*\}$.



Example (Regular Language)

- $L = \{w | w \in \{0, 1\}^* \& w$ binary encoding of a number dividible by $5\}$.



Example (A language that is not regular)

- $L = \{0^n 1^n | w \in \{0, 1\}^*, n \geq 1\}$ is not regular.

Moore Machine Example

Example (Tennis Game Score)

A machine calculates the tenis score.

- Input alphabet: ID of the player who scored a point
- Output alphabet & states: the score ($Q = Y$ and $\mu(q) = q$)

The language accepted by this automaton is the set of correct sequences A's and B's.

State/output	A	B
00:00	15:00	00:15
15:00	30:00	15:15
15:15	30:15	15:30
00:15	15:15	00:30
30:00	40:00	30:15
30:15	40:15	30:30
30:30	40:30	30:40
15:30	30:30	15:40
00:30	15:30	00:40
40:00	A	40:15
40:15	A	40:30
40:30	A	deuce
30:40	deuce	B
15:40	30:40	B
00:40	15:00	B
deuce	A:40	40:B
A:40	A	deuce
40:B	deuce	B
A	15:00	00:15
B	15:00	00:15

Nondeterministic Finite Automata

Definition (Nondeterministic Finite Automata)

A **nondeterministic finite automation (NFA)** $A = (Q, \Sigma, \delta, q_0, F)$ consists of:

- A finite set of **states**, often denoted Q.
- A finite set of **input symbols**, denoted Σ.
- A **transition function** $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ returns a subset of Q.
- A **start state** $q_0 \in Q$.
- A **set accepting states** (final states) $F \subseteq Q$.

Example

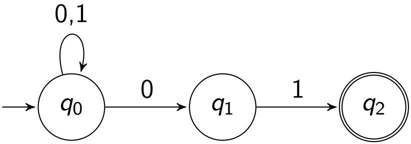
The NFA from previous slide is $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$.

δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

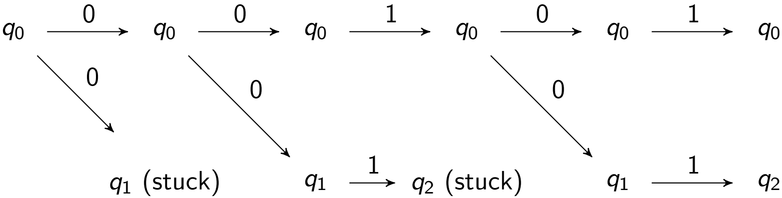
Nondeterministic Finite Automata (NFA)

A NFA can be in serveral states at once. It has an ability to 'guess' something about input.

A NFA accepting all strings that end in 01.



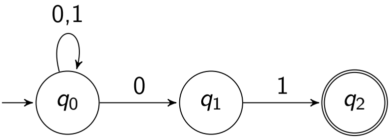
NFA processes input 00101.



Definition (Extended Transition Function to Strings)

If δ is our transition function, then the **extended transition function δ^***, $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ takes a state q and a string w and returns a set of states $\subseteq Q$ and is defined by induction:

> $\delta^*(q, \epsilon) = \{q\}$.
> Let $w = ax$, $a \in \Sigma, x \in \Sigma^*$, suppose $\delta^*(q, x) = \{p_1, \ldots, p_k\}$. Let $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$. Then $\delta^*(q, ax) = \{r_1, r_2, \ldots, r_m\}$.
> First compute $\delta^*(q, x)$ and then follow any transition from any of these states that is labeled a.



$\delta^*(q_0, \epsilon)$	$=$	$=\{q_0\}$
$\delta^*(q_0, 0)$	$= \delta(q_0, 0)$	$=\{q_0, q_1\}$
$\delta^*(q_0, 00)$	$=\delta(q_0, 0) \cup \delta(q_1, 0)$	$=\{q_0, q_1\}$
$\delta^*(q_0, 001)$	$=\delta(q_0, 1) \cup \delta(q_1, 1)$	$=\{q_0, q_2\}$
$\delta^*(q_0, 0010)$	$=\delta(q_0, 0) \cup \delta(q_2, 0)$	$=\{q_0, q_1\}$
$\delta^*(q_0, 00101)$	$=\delta(q_0, 1) \cup \delta(q_1, 1)$	$=\{q_0, q_2\}$

The Language of an NFA

Definition (Language of an NFA)

If $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA, then

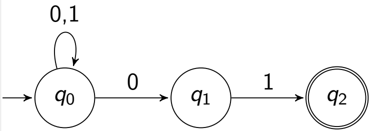
$$L(A) = \{w | \delta^*(q_0, w) \cap F \neq \emptyset\}$$

is the language accepted by NFA A.
That is, $L(A)$ is the set of strings $w \in \Sigma^*$ such that $\delta^*(q_0, w)$ contains at least one accepting state.

Example

The NFA from previous slide accepts the language $L = \{w | w \text{ ends in } 01\}$. The proof is a mutual induction:

- $\delta^*(q_0, w)$ contains q_0 for every w.
- $\delta^*(q_0, w)$ contains q_1 iff w ends in 0.
- $\delta^*(q_0, w)$ contains q_2 iff w ends in 01.



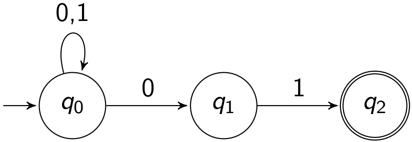
Equivalence of Deterministic and Nondeterministic Finite Automata

Definition (Subset Construction)

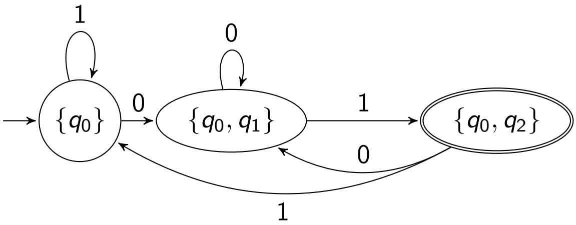
The subset construction starts from an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$. Its goal is the description of an DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(N) = L(D)$.

- Q_D is the set of subsets of Q_N, $Q_D = \mathcal{P}(Q_N)$ (the power set).
  - Inaccesible states can be thrown away so the number of states may be smaller.
- $F_D = \{S : S \in \mathcal{P}(Q_N) \ \& \ S \cap F_N \neq \emptyset\}$, i.e. S include at least one accepting state of N.
- For each $S \subseteq Q_N$ and for each input symbol $a \in \Sigma$,

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a).$$



	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



Theorem (DFA for any NFA)

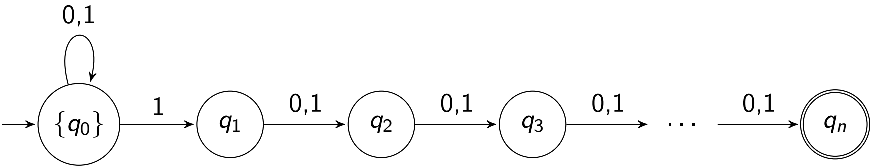
If $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ is the DFA constructed from NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ by subset construction, then $L(N) = L(D)$.

Proof.

By induction we prove: $\delta_D^*(\{q_0\}, w) = \delta_N^*(q_0, w)$. $\square$

Example (A Bad Case for the Subset Construction)

A bad case for the subset construction is a language $L(N)$ of all strings of 0's and 1's such that the nth symbol from the end is 1. Intuitively, a DFA must remember the last n symbols it has read.



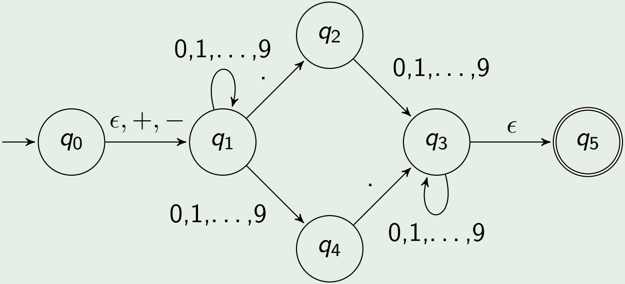
Text search applications.

Finite Automata With ϵ–Transitions

- The new feature is that we allow a transition on ϵ, the empty string, that is without reading any input symbol.

Example (ϵ transition NFA)

(1) Any optional $+$ or - sign,
(2) a string of digits,
(3) A decimal point, and
(4) another string of digits. At least one of strings (2) and (4) must be nonempty.



ϵ-NFA

Definition (ϵ-NFA)

ϵ-NFA is $E = (Q, \Sigma, \delta, q_0, F)$, where all components have their same interpretation as for NFA, except that δ is now a function that takes arguments $Q \times (\Sigma \cup \{\epsilon\})$. We require $\epsilon \notin \Sigma$, so no confusion results.

Example

Previous ϵ-NFA is: $E = (\{q_0, q_1, \ldots, q_5\}, \{., +, -, 0, 1, \ldots, 9\}, \delta, q_0, \{q_5\})$, where

δ is:

	ϵ	+,-	.	0,1,...,9
q_0	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
q_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Extended Transitions and Languages for ϵ-NFA's

Definition

Suppose that $E = (Q, \Sigma, \delta, q_0, F)$ is an ϵ-NFA. We define δ^* as follows:

- $\delta^*(q, \epsilon) = \epsilon CLOSE(q)$.
- Suppose $w = va$ where $a \in \Sigma, v \in \Sigma^*$.
  - Let $\delta^*(q, v) = \{p_1, \ldots, p_k\}$.
  - Let $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, \ldots, r_m\}$.
  - Then $\delta^*(q, w) = \epsilon CLOSE(\{r_1, \ldots, r_m\})$.

Example

$$\begin{array}{rcl}
\delta^*(q_0, \epsilon) = & \epsilon CLOSE(q_0) & = \{q_0, q_1\} \\
\delta^*(q_0, 5) = & \epsilon CLOSE(\bigcup_{q \in \delta^*(q,\epsilon)} \delta(q, 5)) = \epsilon CLOSE(\delta(q_0, 5) \cup \delta(q_1, 5)) & = \{q_1, q_4\} \\
\delta^*(q_0, 5.) = & \epsilon CLOSE(\delta(q_1, .) \cup \delta(q_4, .)) & = \{q_2, q_3, q_5 \\
\delta^*(q_0, 5.6) = & \epsilon CLOSE(\delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 6)) & = \{q_3, q_5\}
\end{array}$$

Definition (Eliminating ϵ-Transition)

Given any ϵ-NFA $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$, we define a DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ that accepts the same language as E.

$Q_D \subseteq \mathcal{P}(Q_E)$, $\forall S \subseteq Q_E : \epsilon CLOSE(S) \in Q_D$. Note that $\emptyset$ may be in Q_D.

$q_D = \epsilon CLOSE(q_0)$.

$F_D = \{S | S \text{ is in } Q_D \text{ and } S \cap F_E \neq \emptyset\}$.

For $S \subseteq Q_D, a \in \Sigma$ define $\delta_D(S, a) = \epsilon CLOSE(\bigcup_{p \in S} \delta(p, a))$.

Theorem (Eliminating ϵ-Transition)

A language L is accepted by some ϵ-NFA if and only if L is regular.

Regular Expressions (RegE)

Definition (Regular Expression (RegE), value of a RegE $L(\alpha)$)

Regular expressions $\alpha, \beta \in RegE(\Sigma)$ over a finite non–empty alphabet $\Sigma = \{x_1, x_2, \dots, x_n\}$ and their value $L(\alpha)$ is defined by induction:

- Basis:

expression α	for	value $L(\alpha) \equiv [\alpha]$
ϵ	empty string	$L(\epsilon) = \{\epsilon\}$
$\emptyset$	empty expression	$L(\emptyset) = \{\} \equiv \emptyset$
$\mathbf{a}$	$a \in \Sigma$	$L(\mathbf{a}) = \{a\}$.

- Induction:

expression	value	remark
$\alpha + \beta$	$L(\alpha + \beta) = L(\alpha) \cup L(\beta)$	
$\alpha\beta$	$L(\alpha\beta) = L(\alpha)L(\beta)$	. may be used
α^*	$L(\alpha^*) = L(\alpha)^*$	
(α)	$L((\alpha)) = L(\alpha)$	brackets do not change the value.

The class of regular expressions over Σ: $RegE(\Sigma)$ is the smallest class closed under operations above.

Examples, Precedence

Example (Regular Expressions)

The language of alternating 0's and 1's may be written:

either $(\mathbf{01})^* + (\mathbf{10})^* + \mathbf{1}(\mathbf{01})^* + \mathbf{0}(\mathbf{10})^*$

or $(\epsilon + \mathbf{1})(\mathbf{01})^*(\epsilon + \mathbf{0})$.

The language $L((\mathbf{0}^*\mathbf{1}\mathbf{0}^*\mathbf{1}\mathbf{0}^*\mathbf{1})^*\mathbf{0}^*) = \{w | w \in \{0,1\}^*, |w|_1 = 3k, k \geq 0\}$.

Definition (Precedence)

The star $*$is the operator with highest precedence, then concatenation ., the lowest precedence has the union $+$.

Theorem (Kleene Theorem (variant))

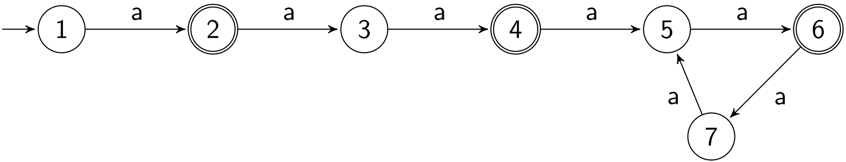
Any language recognizable by a DFA can be expressed by a regular expression. Any language of a regular expression can be recognized by a ϵ-NFA (therefore also a DFA).

Practicals

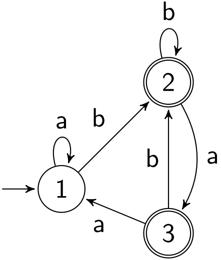
4. Find regular expressions representing languages over $\Sigma = \{a, b\}$:
   a) words with a substring *abba*
   b) words with prefix *abb* and sufix *bbaa*
   c) words w where $|w|_a = 3 * k$
   d) words starting and ending with the same pair of symbols
   e) words not having *aa* as a substring.
5. Construct finite automata accepting languages described by the following regular expressions.
   a) $\mathbf{ab} + \mathbf{ba}$
   b) $\mathbf{a}^2 + \mathbf{b}^2 + \mathbf{ab}$
   c) $\mathbf{a} + \mathbf{b}^*$
   d) $(\mathbf{ab} + \mathbf{c})^*$
   e) $((\mathbf{ab} + \mathbf{c})^+\mathbf{a}(\mathbf{bc})^* + \mathbf{b})^*$
   f) $((\mathbf{ab} + \mathbf{c})^*\mathbf{a}(\mathbf{bc})^* + \mathbf{b})^*$
   g) $(\mathbf{01}^* + \mathbf{101})^*\mathbf{0}^*\mathbf{1}$
   h) $(\mathbf{01})^*\mathbf{11}(\mathbf{01})^*(\mathbf{0} + \mathbf{1})^*\mathbf{00}$

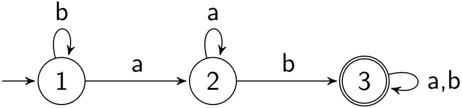
6. Construct regular expressions for languages accepted by the following automata.

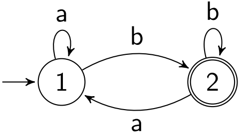
a)

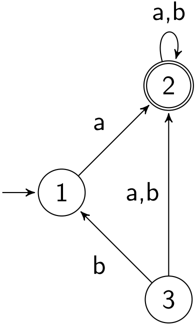


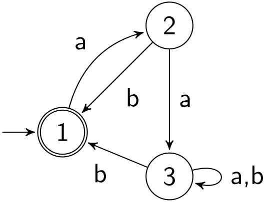
b)






a
b
b
1
2
a


a,b
2
a
1
a,b
b
3


2
a
b
a
1
b
3
a,b

Pumping Lemma For Regular Languages

Is a given language regular?

YES Construct an automaton.

NO Find the contradiction with the Pumping Lemma.

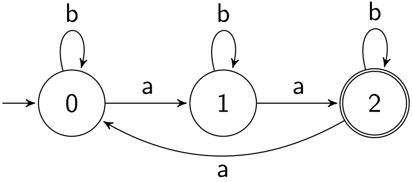
Theorem (Pumping Lemma For Regular Languages)

Let L be a regular language. Then there exists a constant $n \in \mathbb{N}$ (which depends on L) such that for every string $w \in L$ such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

- $y \neq \epsilon$.
- $|xy| \leq n$.
- *For all* $k \geq 0$, *the string* $xy^k z$ *is also in* L.

Example

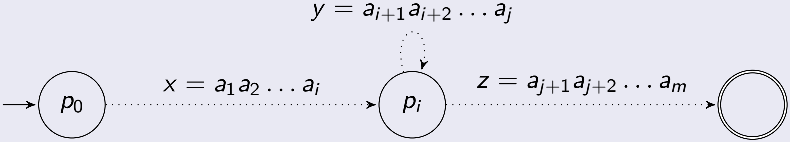
- $abbbba = a(b)bbba$;
  $\forall i \geq 1; a(b)^i bbba \in L(A)$.
- $aaaaba = (aaa)aba$;
  $\forall i \geq 1; (aaa)^i aba \in L(A)$.



Proof of the Pumping Lemma For Regular Languages

Proof.

- Suppose L is regular, then $L = L(A)$ for some DFA A with n states.
- Take any string $w = a_1 a_2 \dots a_m \in L$ of length $m \geq n$, $a_i \in \Sigma$.
- Define $\forall i \; p_i = \delta^*(q_0, a_1 a_2 \dots a_i)$. Note $p_0 = q_0$.
- We have $n + 1$ p_i's and n states, therefore there are i, j such that $0 \leq i < j \leq n : p_i = p_j$.
- Define: $x = a_1 a_2 \dots a_i$, $y = a_{i+1} a_{i+2} \dots a_j$, $z = a_{j+1} a_{j+2} \dots a_m$. Note $w = xyz$.



- The loop above p_i can be repeated any number of times and the input is also accepted.

□

Applications of the Pumping Lemma

Example (The Pumping Lemma as an Adversarial Game)

The language $L_{eq} = \{w; |w|_0 = |w|_1\}$ of all strings with an equal number of 0's and 1's is not regular language.

Proof.

- Suppose it is regular. Take n from the pumping lemma.
- Pick $w = 0^n 1^n \in L_{eq}$.
- Break $w = xyz$ as in the pumping lemma, $y \neq \epsilon$, $|xy| \leq n$.
- Since $|xy| \leq n$ and it comes at front of w, it consists only of 0's. The pumping lemma says: $xy \in L_{eq}$ (for $k = 0$). However, it has less 0's and the same amount of 1's as w, so one of them must not be in L_{eq}.

$\square$

Example

The language $L = \{0^i 1^i; i \geq 0\}$ is not regular.

Applications of the Pumping Lemma 2

Example

The language L_{pr} of all strings of 1's whose length is a prime is not a regular language.

Proof.

- Suppose it were. Take a constant n from the pumping lemma. Consider some prime $p \geq n+2$, let $w = 1^p$.
- Break $w = xyz$ by the pumping lemma, let $|y| = m$. Then $|xz| = p - m$.
- $xy^{p-m}z \in L_{pr}$ by pumping lemma, but $|xy^{p-m}z| = |xz| + (p-m)|y| = p - m + (p-m)m = (m+1)(p-m)$ that is not a prime (none of two factors are 1).

$\square$

Example (Non–regular language that can be 'pumped')

The language $L = \{u | u = a^+ b^i c^i \vee u = b^i c^j\}$ is not regular (Myhill–Nerode theorem), but the first symbol can be always pumped.

Practicals Pumping Lemma For Regular Languages

7. Are following languages regular?

a) $L = \{ww \mid w \in \{a, b\}^*\}$
b) $L = \{ww \mid w \in \{a\}^*\}$
c) $L = \{a^i b^j \mid i, j = 0, 1, 2, \ldots\}$
d) $L = \{a^i a^i b^j \mid i, j = 0, 1, 2, \ldots\}$
e) $L = \{a^i b^j a^i \mid i, j = 0, 1, 2, \ldots\}$
f) $L = \{a^i b^i a^j \mid i, j = 0, 1, 2, \ldots\}$
g) $L = \{a^i b^j c^k \mid i, j, k = 0, 1, 2, \ldots\}$
h) $L = \{ww^R \mid w \in \{a, b\}^*\}$
i) $L = \{ww^R \mid w \in \{a, b\}^* \;\&\; |w|_a = |w|_b\}$
j) $L = \{a^{2i} \mid i = 0, 1, 2, \ldots\}$
k) $L = \{a^{i^2} \mid i = 0, 1, 2, \ldots\}$
l) $L = \{a^{3i} \mid i = 0, 1, 2, \ldots\}$
m) $L = \{a^{i^3} \mid i = 0, 1, 2, \ldots\}$
n) $L = \{a^{3^i} \mid i = 0, 1, 2, \ldots\}$
o) $L = \{a^p \mid p \text{ is a prime number}\}$

Lexical Analysis

- Lexical analyzer scans the source program and recognizes all *tokens* (keywords, identifiers, and many others).
- We specify RE and code as below.
- RE are converted to ϵ-NFA, accepting states distinguish which token was recognized.
- If more than one token is recognized at once, by convention the top-listed RE wins (e.g. `else` may be reserved or identifier, first list reserved words).

Example (A sample of `lex` input)

regular expression	action when found
else	`{return(ELSE);}`
`[A-Za-z][A-Za-z0-9]*`	`{code to enter the found identifier in the symbol table; return(ID); }`
>=	`{return(GE);}`
=	`{return(ASGN);}`

Finding Patterns in Text

- Static text is usually indexed, other methods used.
- RE are useful for the search in dynamic (new) text as daily news.

Example (Search for streets in addresses on the web)

Street identification	`Streen|St\.|Avenue|Ave\.|Road|Rd\`
the name before	`'[A-Z][a-z]*( [A-Z][a-z]*)*'`
house number	`[0-9]+[A-Z]?`
all together	`'[0-9]+[A-Z]? [A-Z][a-z]*( [A-Z][a-z]*)*` `Streen|St\.|Avenue|Ave\.|Road|Rd\.  '`

We are missing:

- Bouleward, Place, Way
- Streets without any identifier (almost all Czech streets)
- Street names with numbers.
- ...