Formal Languages in Theory and Practice — day 4

Complexity of Natural Languages
Mildly-context sensitivity
T1 languages

Wiebke Petersen & Kata Balogh

(Heinrich-Heine-Universität Düsseldorf)

ESSLLI 2019
University of Latvia, Riga

Outline

Chomsky Normal Form

Definition

A grammar is in ***Chomsky Normal Form (CNF)*** *if all production rules are of the form*

$$A \rightarrow a \text{ or } A \rightarrow BC$$

where $A, B, C \in T$ and $a \in \Sigma$ (and if necessary $S \rightarrow \epsilon$ in which case S may not occur in any right-hand side of a rule).

Proposition

No node in a derivation tree of a grammar in CNF has more than two daughters.

Proposition

Each context-free language is generated by a grammar in CNF.

Chomsky Normal Form

- Each context-free language is generated by a grammar in CNF.
- given a context-free grammar G_{CF} with $\epsilon \notin L(G_{CF})$
- create an equivalent grammar G_{CNF} in CNF in 3 steps:
  1. Eliminate complex terminal rules.
  2. Eliminate chain rules.
  3. Eliminate $A \rightarrow B_1 B_2 \dots B_n$ $(n > 2)$ rules.

G_{CF} :	G_{CNF} :
$\mathsf{S} \rightarrow \mathsf{ABA} \mid \mathsf{B}$	$\mathsf{S} \rightarrow \mathsf{AZ} \mid \mathsf{YB} \mid \mathsf{b}$
$\mathsf{A} \rightarrow \mathsf{aA} \mid \mathsf{C} \mid \mathsf{a}$	$\mathsf{A} \rightarrow \mathsf{XA} \mid \mathsf{a}$
$\mathsf{B} \rightarrow \mathsf{bB} \mid \mathsf{b}$	$\mathsf{B} \rightarrow \mathsf{YB} \mid \mathsf{b}$
$\mathsf{C} \rightarrow \mathsf{A}$	$\mathsf{X} \rightarrow \mathsf{a}$
	$\mathsf{Y} \rightarrow \mathsf{b}$
	$\mathsf{Z} \rightarrow \mathsf{BA}$

CNF: eliminate complex terminal rules

Aim: Terminals only occur in rules of type $A \rightarrow a$

1. Introduce a new non-terminal X_a for each terminal a occurring in a complex terminal rule.
2. Replace a by X_a in all complex terminal rules.
3. For each X_a add a rule $X_a \rightarrow a$.

$$
\begin{array}{l}
\mathsf{S} \rightarrow \mathsf{ABA} \mid \mathsf{B} \\
\mathsf{A} \rightarrow \mathsf{aA} \mid \mathsf{C} \mid \mathsf{a} \\
\mathsf{B} \rightarrow \mathsf{bB} \mid \mathsf{b} \\
\mathsf{C} \rightarrow \mathsf{A}
\end{array}
\quad \Longrightarrow \quad
\begin{array}{l}
\mathsf{S} \rightarrow \mathsf{ABA} \mid \mathsf{B} \\
\mathsf{A} \rightarrow \mathsf{X}_a\mathsf{A} \mid \mathsf{C} \mid \mathsf{a} \\
\mathsf{B} \rightarrow \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\
\mathsf{C} \rightarrow \mathsf{A} \\
\mathsf{X}_a \rightarrow \mathsf{a} \\
\mathsf{X}_b \rightarrow \mathsf{b}
\end{array}
$$

CNF: eliminate chain rules

Aim: No rules of the form $A \rightarrow B$

- For each circle $A_1 \rightarrow A_2, \ldots, A_{k-1} \rightarrow A_k, A_k \rightarrow A_1$ replace in all rules each A_i by a new non-terminal A' and delete all $A' \rightarrow A'$-rules.
- Remove stepwise all rules $A \rightarrow B$ and add for each $B \rightarrow \beta$ a rule $A \rightarrow \beta$

$$
\begin{array}{l}
\mathsf{S} \rightarrow \mathsf{ABA} \mid \mathsf{B} \\
\mathsf{A} \rightarrow \mathsf{X}_a\mathsf{A} \mid \mathsf{C} \mid \mathsf{a} \\
\mathsf{B} \rightarrow \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\
\mathsf{C} \rightarrow \mathsf{A} \\
\mathsf{X}_a \rightarrow \mathsf{a} \\
\mathsf{X}_b \rightarrow \mathsf{b}
\end{array}
\Longrightarrow
\begin{array}{l}
\mathsf{S} \rightarrow \mathsf{A'BA'} \mid \mathsf{B} \\
\mathsf{A'} \rightarrow \mathsf{X}_a\mathsf{A'} \mid \mathsf{a} \\
\mathsf{B} \rightarrow \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\
\mathsf{X}_a \rightarrow \mathsf{a} \\
\mathsf{X}_b \rightarrow \mathsf{b}
\end{array}
\Longrightarrow
\begin{array}{l}
\mathsf{S} \rightarrow \mathsf{A'BA'} \mid \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\
\mathsf{A'} \rightarrow \mathsf{X}_a\mathsf{A'} \mid \mathsf{a} \\
\mathsf{B} \rightarrow \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\
\mathsf{X}_a \rightarrow \mathsf{a} \\
\mathsf{X}_b \rightarrow \mathsf{b}
\end{array}
$$

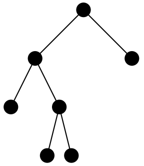
CNF: $A \to B_1 B_2 \ldots B_n$ $(n > 2)$

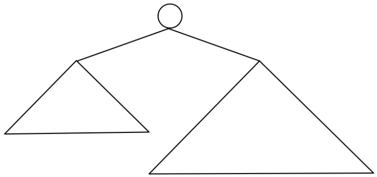
Aim: not more than two non-terminals in one rule's right-hand side

- For each rule of the form $A \to B_1 B_2 \ldots B_n$ introduce a new non-terminal $X_{B_2 \ldots B_n}$.
- Remove the rule and add two new rules:
  - $A \to B_1 X_{B_2 \ldots B_n}$
  - $X_{B_2 \ldots B_n} \to B_2 \ldots B_n$

$$\begin{array}{l} \mathsf{S} \to \mathsf{A'BA'} \mid \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\ \mathsf{A'} \to \mathsf{X}_a\mathsf{A'} \mid \mathsf{a} \\ \mathsf{B} \to \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\ \mathsf{X}_a \to \mathsf{a} \\ \mathsf{X}_b \to \mathsf{b} \end{array} \quad \Longrightarrow \quad \begin{array}{l} \mathsf{S} \to \mathsf{A'X}_{BA'} \mid \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\ \mathsf{A'} \to \mathsf{X}_a\mathsf{A'} \mid \mathsf{a} \\ \mathsf{B} \to \mathsf{X}_b\mathsf{B} \mid \mathsf{b} \\ \mathsf{X}_a \to \mathsf{a} \\ \mathsf{X}_b \to \mathsf{b} \\ \mathsf{X}_{BA'} \to \mathsf{BA'} \end{array}$$

Binary trees





The height of a tree is the maximal number of edges on a downward path from the root to a leaf.

Proposition

If T is an arbitrary binary tree with at least 2^k leafs, then height$(T) \geq k$.

Proof by induction on k. The proposition is true for $k = 0$. Given the proposition is true for some fixed k, let T be a tree with $\geq 2^{k+1}$ leafs. T has two subtrees of which at least one has 2^k leafs and thus a height $\geq k$. It follows that the height of T is $\geq k + 1$.

Corollary

If a context-free grammar is in CNF, then the height of a derivation tree of a word of length $\geq 2^k$ is greater than k (note that the last derivation step is always a unary one).

Pumping lemma for context-free languages

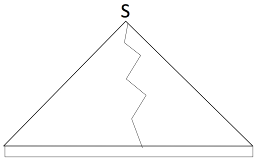
Lemma (Pumping Lemma)

For each context-free language L there exists a $n \in \mathbb{N}$ such that for any $z \in L$: if $|z| \geq n$, then z may be written as $z = uvwxy$ with

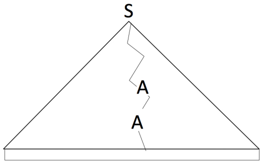
- $u, v, w, x, y \in T^*$,
- $|vwx| \leq n$,
- $vx \neq \epsilon$ *and*
- $uv^iwx^iy \in L$ *for any* $i \geq 0$.

Pumping lemma: proof sketch

Let (N, T, S, R) be a context-free grammar in CNF generating L. Let $k = |N|$ and $n = 2^k$. Be $z \in L$ with $|z| \geq n$.

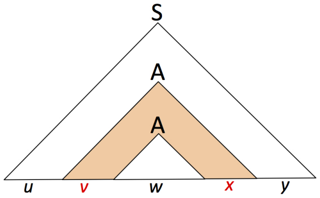


Take a maximal path in the binary part of the derivation tree of z. Because of $|z| \geq 2^k$ the length of the path is $\geq k$.



At least one non-terminal symbol occurs twice on the path.
Starting from the bottom of the path, let A be the first non-terminal occurring twice.

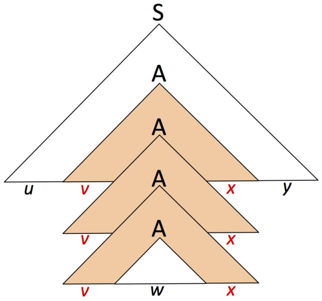
Pumping Lemma: proof sketch

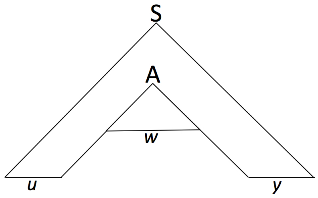


$|vwx| \leq n = 2^k$ (A is chosen such that no non-terminal occurs twice in the trees spanned by the upper of the two A's $\Rightarrow$ height of tree spanned by upper A $\leq k \Rightarrow$ width of tree spanned by upper A $\leq 2^k$)

$vx \neq \epsilon$ (a binary rule $A \to BC$ must have been applied to the upper A)

Pumping Lemma: proof sketch





$uv^iwx^iy \in L$ for any $i \geq 0$.

Pumping Lemma: application

The language $L(a^k b^m c^k d^m)$ is not context-free

- Assume that $L(a^k b^m c^k d^m)$ is context-free then there is a $n \in \mathbb{N}$ as specified by the Pumping Lemma.
- Choose $z = a^n b^n c^n d^n$, and $z = uvwxy$ in accordance with the Pumping Lemma.
- Because of $vwx \leq n$ the string vwx consists either of only a's, of a and b's, only of b's, of b and c's, only of c's,....
- It follows that the pumped word uv^2wx^2y cannot be in L.
- That contradicts the assumption that L is context-free.

Closure properties of context-free languages

	Type3	Type2	Type1	Type0
union	+	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+	+	+	+
Kleene's star	+	+	+	+
homomorphism	+	+	+	+
intersection with a regular language	+	+	+	+

union: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with
$P = P_1 \cup_{\uplus} P_2 \cup \{S \to S_1, S \to S_2\}$

concatenation: $G = (N_1 \uplus N_2 \cup \{S\}, T_1 \cup T_2, S, P)$ with $P = P_1 \cup_{\uplus} P_2 \cup \{S \to S_1 S_2\}$

Kleene's star: $G = (N_1 \cup \{S\}, T_1, S, P)$ with $P = P_1 \cup \{S \to S_1 S, S \to \epsilon\}$

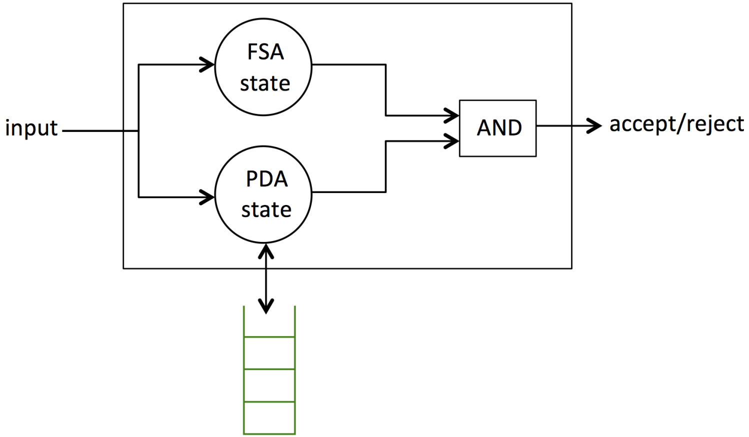
intersection: $L_1 = \{a^k b^* c^k d^*\}$, $L_2 = \{a^* b^m c^* d^m\}$, but $L_1 \cap L_2 = \{a^k b^m c^k d^m\}$

complement: *de Morgan*

Closure properties of context-free languages

CFLs are closed under intersection with a regular language.

the proof is based on running a PDA and an FSA parallel



Are natural languages context-free?

- a long time debate about the context-freeness of natural languages

  Chomsky (1957) : "Of course there are languages (in our general sense) that cannot be described in terms of phrase structure, but I do not know whether or not English is itself literally outside the range of such analysis."

- several wrong arguments, e.g.:

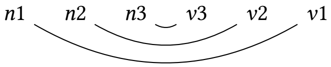
  Bresnan (1987): : "in many cases the number of a verb agrees with that of a noun phrase at some distance from it ... this type of syntactic dependency can extend as memory or patience permits ... the distant type of agreement ... cannot be adequately described even by context-sensitive phrase-structure rules, for the possible context is not correctly describable as a finite string of phrases."

- right proof techniques: pumping lemma and closure properties
- a non context-free phenomenon: **cross-serial dependencies** in Schwyzerdütsch (Schieber 1985)

Are natural languages context-free?

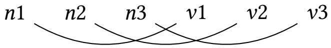
- German: **nested dependency** (subordinate clauses)

(1) ..., daßer die Kinder dem Hans das Haus streichen helfen ließ.
.... that he the children the Hans the house paint help let.
'.... that he he let the children to help Hans to paint the house.'



- Schwyzerdütsch: **cross-serial dependency**

(2) ..., das mer d'chind em Hans es huus lönd hälfe aastriiche.
... that we children.acc the Hans.dat the house.acc let help paint.
'... that we let the children to help Hans to paint the house.'



(3) *mer d'chind de Hans es huus lönd hälfe aastriiche.
we children.acc the Hans.acc the house.acc let help paint.

NL not context-free

- Jan säit das mer d’chind em Hans es huus lönd hälfe aastriiche.
- homomorphism f:
- $f(\text{Schwyzerdütsch}) \cap wa^* b^* xc^* d^* y = wa^m b^n xc^m d^n y$
  - CF languages are closed under intersection with regular languages
  - $wa^* b^* xc^* d^* y$ is regular
  - by Pumping Lemma: $wa^m b^n xc^m d^n y$ is not context-free
- $\Rightarrow$ Schwyzerdütsch is not context-free

Dutch cross dependencies

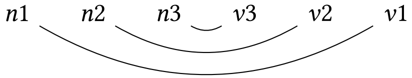
- cross dependencies in Dutch

(4)

dat	Jan	Piet	de	kinderen	zag	helpen	zwemmen.
that	Jan	Piet	the	children	saw	help	swim

‘that Jan saw Piet helping the children to swim.’

  - ▸ no case marking → string can be generated by a CFG



  - ▸ however the linguistic dependencies are not preserved ⇒ structure (predicate-argument relations)
  - ▸ weak generative capacity: preserve the string language
  - ▸ strong generative capacity: preserve the structure

Duplication

- duplication (in morphology): Bambara (spoken in Mali)
  - wulu *'dog'*
    wulu-lela *'dog watcher'*
    wulu-lela-nyinila *'dog watcher hunter'*
    wulu-o-wulu *'whatever dog'*
    wulu-lela-o-wulu-lela *'whatever dog watcher'*
    wulu-lela-nyinila-o-wulu-lela-nyinila *'whatever dog watcher hunter'*
- structure of the form $x = yy \Rightarrow$ not context-free

Context-sensitive languages

- NL $\notin$ CFL (T2)
- context-sensitive? (T1)

Definition

A grammar (N, T, S, R) is **Type1** or **context-sensitive** iff all rules are of the form:

$\gamma A \delta \to \gamma \beta \delta$ with $\gamma, \delta, \beta \in (N \cup T)^*$, $A \in N$ and $\beta \neq \epsilon$;

With the exception that $S \to \epsilon$ is allowed if S does not occur in any rule's right-hand side.

A language generated by a T1 grammar is said to be a **context-sensitive** or **Type1-language**.

- γ and δ can be empty, but β cannot be the empty string; $\beta \neq \epsilon$ (!)
  $\leadsto$ 'non-shrinking' context-sensitive scheme

Example: CS grammar

- consider the language $a^n b^n c^n$
- a context-sensitive grammar generating this language is:
  - $G = (T, N, S, R)$ with
    $T = \{a, b, c\}$
    $N = \{S, A, B, C, T\}$
    $R = \{$S$\rightarrow$ ϵ, S$\rightarrow$ T,
    T$\rightarrow$ aBC, T$\rightarrow$ aTBC,
    (recursively generating $a^n(BC)^n$)
    CB $\rightarrow$ CX, CX $\rightarrow$ BX, BX $\rightarrow$ BC,
    (swapping two non-terminals: $CB \rightarrow BC$)
    aB $\rightarrow$ ab, bB $\rightarrow$ bb, bC $\rightarrow$ bc, cC $\rightarrow$ cc$\}$
    (from $a^n B^n C^n$ to $a^n b^n c^n$)

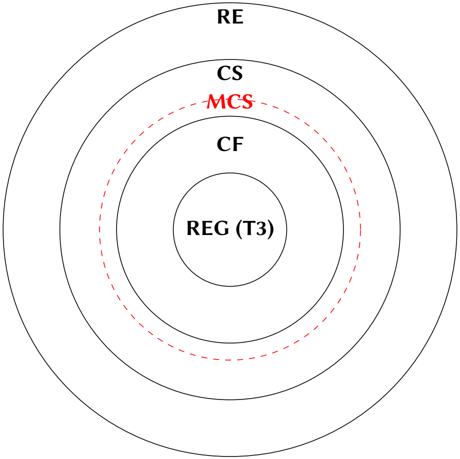
  - S $\Rightarrow$ T $\Rightarrow^*$ aaaaBCBCBCBC $\Rightarrow^*$ aaaaBCBCBCBC $\Rightarrow^*$ aaaaBBCCBCBC $\Rightarrow^*$ aaaaBBCBCCBC $\Rightarrow^*$ aaaaBBBBCCCC $\Rightarrow$ aaaabBBBCCCC $\Rightarrow$ aaaabbBBCCCC $\Rightarrow^*$ aaaabbbbCCCC $\Rightarrow$ aaaabbbbcCCC $\Rightarrow$ aaaabbbbccCC $\Rightarrow^*$ aaaabbbbcccc

Mildly context sensitive languages

- class of NLs is outside of the set of context-free languages
- CFGs are not powerful enough to describe all NL phenomena
- questions: how much context-sensitivity is required?
- for natural languages context-free grammars are just not ‘enough’
  (1) $\{a^n b^n c^n \mid n \geq 0\}$ (multiple agreement)
  (2) $\{a^n b^m c^n d^m \mid m, n \geq 0\}$ (cross-serial dependencies)
  (3) $\{ww \mid w \in \{a, b\}^*\}$ (duplication)
- to describe all NL phenomena we need grammars, that are somewhat richer than CFGs, but more restricted than CSGs
- Aravind Joshi (1985): notion of **mild context-sensitivity**

Mildly context sensitive languages

$$RL \subset CFL \subset MCSL \subset CSL \subset RE$$



Mildly context sensitive languages

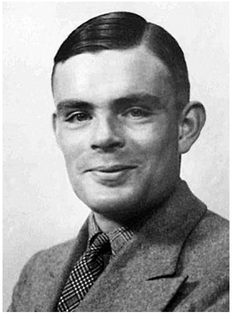
Definition: Mildly context-sensitive language (Joshi, 1985)

1. A set $\mathcal{L}$ of languages is mildly context-sensitive iff
   a. $\mathcal{L}$ contains all context-free languages
   b. $\mathcal{L}$ can describe **cross-serial dependencies**: there is an $n \geq 2$ such that $\{w^k \mid w \in (V_T)^*\} \in L$ for all $k \geq n$
   c. the languages in $\mathcal{L}$ are **polynomially parseable**, i.e., $L \subset$ PTIME
   d. the languages in $\mathcal{L}$ have the constant growth property
2. A formalism F is mildly context-sensitive iff the set $\{L \mid L = L(G)$ for some $G \in F\}$ is mildly context-sensitive.

- mildly context-sensitive grammar formalisms
  - Linear Indexed Grammars (LIGs)
  - Head Grammars (HGs)
  - Combinatory Categorial Grammars (CCGs)
  - Tree Adjoining Grammars (TAGs)
  - Multicomponent TAGs (MCTAGs)
  - Linear Context-Free Rewriting Systems (LCFRSs)

Turing machine

- unrestricted grammars generate Type 0 languages
- Turing machines recognize Type 0 languages


Alan Turing

(1912 – 1952)

Turing machine: an abstract 'computer'

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. [...]
I think that it is agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e. on tape divided into squares."

[Alan Turing: On computable numbers with an application to the Entscheidungsproblem. In: *Proceedings of the London Mathematical Society*, 2, 1936.]

Turing machine

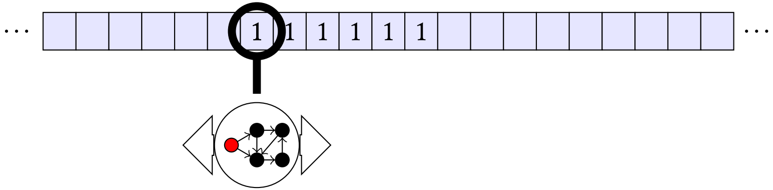
Definition

*A deterministic **Turing machine** (TM) is a tuple* $(Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ *with:*

- Q *is a finite, non-empty set of **states***
- $\Sigma \subset \Gamma$ *is the set of the **input symbols***
- Γ *is the finite, non-empty set of the **tape symbols***
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *is the partial **transition function** with L for left and R for right move.*
- $q_0 \in Q$ *is the **initial state***
- $\Box \in \Gamma \setminus \Sigma$ *is the **blank symbol***
- $F \subseteq Q$ *is the set of **accepting states***

Note: the transition function is partial, i.e. for some state tape-symbol pairs $\delta(q, a)$ is undefined.
Proposition: Every non-deterministic TM can be transformed into an equivalent deterministic TM.

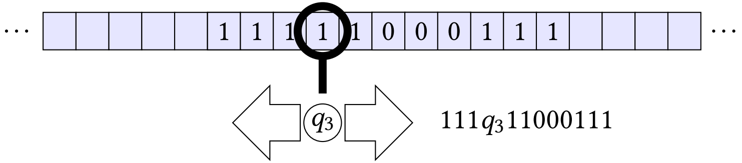
An image of a Turing machine



Start conventions

- The tape of the TM contains the input finite string. All other tape positions are filled by the blank symbol □.
- The (read-write) head of the TM is placed above the left-most input symbol.
- The TM is in the initial state.

Configurations



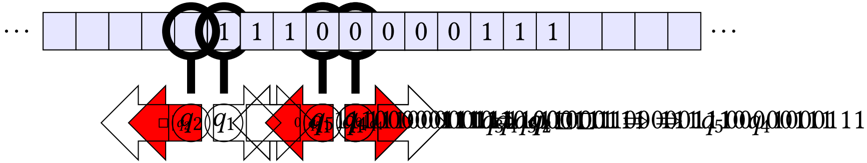
$111q_311000111$

A **configuration** of a TM is a string $\alpha q \beta$, where

- α is the string of symbols to the left of the head starting with the left-most non-blank symbol on the tape
- q is the state the TM is in.
- β is the rest of the string ending with the right-most non-blank symbol on the tape.
- the read-write head is currently scanning the first symbol of β.

$\alpha\beta$ must be finite for any configuration of a TM as every configuration of a TM is reached after a finite number of steps (i.e., the head can only be moved a finite number of positions to the right or to the left from the starting position).

Transitions



Transitions

- $\delta(q, a) = (q', b, R|L)$ specifies that if the TM is in state q and reads an a it can change to state q', write b, and move either one position right (R) or left (L).
- For a right-move transition $\delta(q, a) = (q', b, R)$ we get: $\alpha qa\beta \Rightarrow \alpha bq'\beta$.
- For a left-move transition $\delta(q, a) = (q', b, L)$ we get: $\alpha cqa\beta \Rightarrow \alpha q'cb\beta$

$\Rightarrow^*$ is used as before for the closure of $\Rightarrow$

Language accepted by a TM

Acceptance by final state

A turing machine M **accepts** the language $L(M)$ by final state:
$L(M) = \{w \mid q_0 w \Rightarrow^* C$, where C is a configuration with a final state$\}$

Acceptance by halting

A turing machine M **accepts** the language $H(M)$ by halting:
$H(M) = \{w \mid q_0 w \Rightarrow^* C$, where C is a configuration without possible moves$\}$

Equivalence of acceptance by finite state and by halting

- If $L = L(M)$, then there exists a TM M' with $L = H(M')$.
  (remove all moves from the final state)
- If $L = H(M)$, then there exists a TM M'' with $L = L(M'')$.
  (transition to a new final state from all pairs for which $\delta(q, a)$ is undefined).

Turing machines accept the Type0 languages.

Turing-computable functions

TMs can be seen as acceptors (accepting languages) or as computers (computing functions).

A partial function $f : \Sigma^* \to \Sigma^*$ is Turing-computable if there exists a TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that:

$f(w) = v$ if and only if $q_0 w \Rightarrow^* q_f v$ with $q_f \in F$.

Church's thesis

Every effective computation can be carried out by a Turing machine. Everything that is in some intuitive way computable is Turing-computable.

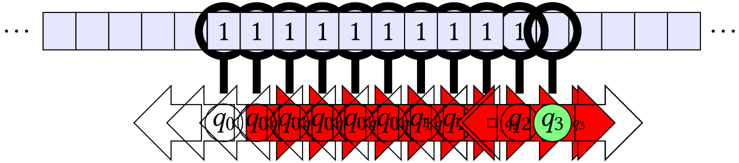
Thus "computable" is equivalent to "Turing-computable".

Example: Turing machine for addition

Note: In this and the following example we violate the restriction that the input string should not contain the blank symbol. Use another symbol to separate the two number representations and write a new addition and substraction machine (the machines even become simpler).

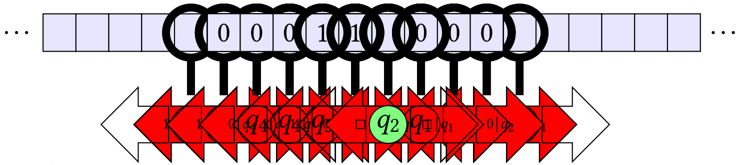
- Take the following Turing machine:
  $M = (\{q_0, q_1, q_2\}, \{1\}, \{1\}, q_0, \delta, \{q_3\})$
  - states: q_0, q_1, q_2
  - alphabet and input alphabet: $\{1\}$
  - final state: q_3
  - transitions: $\delta = \{(q_0, 1) \rightarrow (q_0, 1, R), (q_0, \Box) \rightarrow (q_1, 1, R),$
    $(q_1, 1) \rightarrow (q_1, 1, R), (q_1, \Box) \rightarrow (q_2, \Box, L),$
    $(q_2, 1) \rightarrow (q_3, \Box, R)\}$

Example: Turing machine for addition



	1	$\square$
q_0	$(q_0, 1, R)$	$(q_1, 1, R)$
q_1	$(q_1, 1, R)$	$(q_2, \square, L)$
q_2	$(q_3, \square, R)$	
$\underline{q_3}$		

Turing machine for subtraction



	1	□	0
q_0	$(q_0, 1, R)$	$(q_1, \Box, R)$	
q_1	$(q_1, 1, R)$	$(q_2, 0, L)$	$(q_2, \Box, L)$
q_2		🟩	$(q_3, 0, L)$
q_3	$(q_3, 1, L)$	$(q_4, \Box, L)$	
q_4	$(q_4, 1, L)$	$(q_5, \Box, R)$	$(q_5, 0, R)$
q_5			$(q_0, 0, R)$

TM extensions

multi-track TMs

If a language L is accepted by a TM with any finite number of tracks, then there is a TM with one tape which accepts L.

A multi-track TM consists of a finite number of tapes, called tracks; the head scans all tapes at the same position and moves on all tapes in simultaneously (analogue to 1-tape TM with a tape alphabet of vectors).

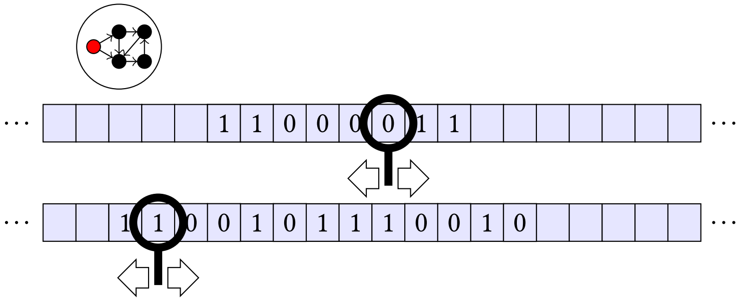
multi-tape TMs

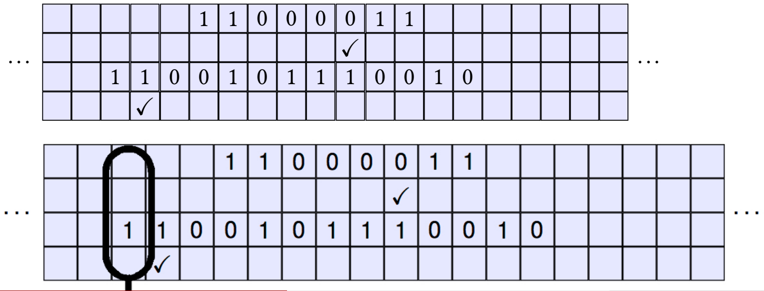
If a language L is accepted by a TM with any finite number of tapes, then there is a TM with one tape which accepts L.

In a multi-tape TM the head can move independently on all tapes.
A 2-tape TM is simulated by a 4-track TM, where

- the 1st track simulates the tape of the 1st TM.
- the 2nd track simulates the position of the head of the 1st TM.
- the 3rd track simulates the tape of the 2nd TM.
- the 4th track simulates the position of the head of the 2nd TM.

multi-tape TM → multi-track TM





Further extensions and restrictions of TMs

Extension: nondeterministic TMs

If a language L is accepted by a nondeterministic Turing machine, then there is a deterministic Turing machine which accepts L.

Restrictions of TMs

- Push-down automata with two stacks have the same expressive power as Turing machines.
- Turing machines with semi-bounded tapes (the tape only grows into one direction) have the same expressive power as Turing machines.

Linearly bounded TMs

Turing machines with a bounded tape the length of which is linearly bounded by the length of the input string are weaker than general Turing machines. They accept languages of Type 1.

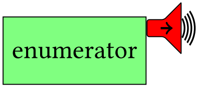
Enumerations

enumerable

A language $L \subseteq \Sigma^*$ is enumerable, if $L = \emptyset$ or there exists a total function $f : \mathbb{N} \to \Sigma^*$ such that $L = \{f(n) | n \in \mathbb{N}\}$.

recursively enumerable

A language $L \subseteq \Sigma^*$ is recursively enumerable, if $L = \emptyset$ or there exists a total computable function $f : \mathbb{N} \to \Sigma^*$ such that $L = \{f(n) | n \in \mathbb{N}\}$



$\ldots, w_{14}, w_{13}, w_{12}, w_{11}, w_{10}, w_9, w_8, w_7, w_6, w_5, w_4, w_3, w_2, w_1$

Proposition

A language is accepted by a Turing machine iff it is recursively enumerable.

Closure properties of recursively enumerable languages

Recursively enumerable languages are closed under

- union (Given two TMs M and M'. For $L(M) \cup L(M')$ construct a 2-tape Turing machine which simulates M and M' on the two tapes.)
- intersection (Similar construction as for union but with $L(M) \cap L(M')$)
- concatenation (Given two TMs M and M'. For $L(M) \frown L(M')$ construct a 2-tape nondeterministic TM which guesses the breakpoint of an input string and then simulates on the first tape M on the first part of the string and on the second tape M' on the second part.)
- Kleene star (Similar to concatenation)

RE is not closed under complement, as we cannot decide whether a running TM will ever halt.

Closure properties of formal language classes

	Type3	Type2	Type1	Type0
union	+	+	+	+
intersection	+	-	+	+
complement	+	-	+	-
concatenation	+	+	+	+
Kleene’s star	+	+	+	+
intersection with a regular language	+	+	+	+