

# Základy informatiky

## Von Neumannův model počítače

Teoretické základy konstrukce současných počítačů položil John von Neumann (1903-1957), jeden z největších matematiků minulého století. Tento návrh je znám jako *von Neumannova koncepce počítače*, která umožňuje vytvořit univerzální počítač pro široké využití, v této koncepci shrnul práce svých předchůdců (Babbage, Turing) a vytvořil nadčasový model počítače. JVN většinu bodů své koncepce nevymyslel, ale vytvořil ucelený koncept, který se ukázal funkční a který umožnil hromadnou výrobu počítačů, které se pro různé úlohy lišily pouze programovým vybavením.

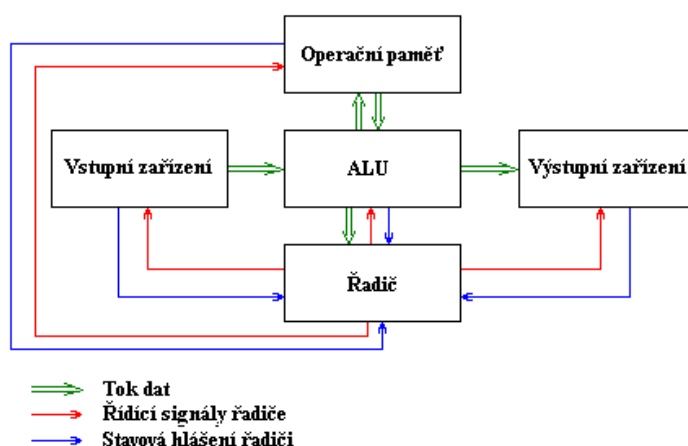
Podle Neumannovy koncepce se počítač skládá z několika základních částí: *paměti*, *procesoru* (tenkrát řadiče a aritmetické jednotky), *vstupních a výstupních zařízení*. Pro reprezentaci všech údajů se používá dvojková soustava, tedy pouze nuly a jedničky. Dnes jsou tyto body zcela samozřejmé, *ve své době však byly zcela převratné*. Podle von Neumannovy koncepce byl v roce 1951 vyroben elektronkový počítač Edvac, který byl již poměrně univerzálně využitelný.

### Body John von Neumannovy koncepce:

- Počítač bude využívat dvojkovou soustavu (tedy jen 0 a 1) – bude digitální. Tento bod JVN nevymyslel, ale důsledně použil.
- Počítač (jeho technická část) bude univerzální a pro různá využití se budou používat různé programy řídící jeho činnost.
- Počítač bude mít procesor (původně řadič a aritmetickou jednotku) vykonávající postupně jednotlivé, v programu zadané operace a dále (operační) paměť, ve které budou data a programy v okamžiku, kdy s nimi bude procesor pracovat.
- Tato paměť bude společná pro programy a data. Neboli, určitá část kódu v paměti může být interpretována jako program a jiná jako data.
- Také bude mít počítač vstupní zařízení pro zadávání programů nebo dat a pro jejich dlouhodobější uchování (z počátku děrné štítky, dnes disky) a výstupní zařízení, která nám umožní vidět výsledky výpočtu.

**Von Neumannova architektura** je v informatice označení pro jednoduché schéma počítače, které používá jednu sběrnici, na kterou jsou připojeny všechny aktivní prvky (procesor, paměť, vstupy a výstupy). Von Neumannovo schéma bylo navrženo roku 1945 americkým matematikem (narozeným v Maďarsku) Johnem von Neumannem jako model samočinného počítače. Tento model s jistými výjimkami zůstal zachován dodnes.

Podle tohoto schématu se počítač **skládá z pěti hlavních modulů**:



- **Operační paměť:** slouží k uchování zpracovávaného programu, zpracovávaných dat a výsledků výpočtu
- **ALU – Arithmetic-logic Unit (aritmetikologická jednotka):** jednotka provádějící veškeré aritmetické výpočty a logické operace. Obsahuje sčítačky, násobičky (pro aritmetické výpočty) a komparátory (pro porovnávání)
- **Řadič:** řídicí jednotka, která řídí činnost všech částí počítače. Toto řízení je prováděno pomocí **řídicích signálů**, které jsou zasílány jednotlivým modulům. Reakce na řídicí signály, stavy jednotlivých modulů jsou naopak zasílány zpět řadiči pomocí **stavových hlášení**
- **Vstupní zařízení:** zařízení určená pro vstup programu a dat
- **Výstupní zařízení:** zařízení určená pro výstup výsledků, které program zpracoval

Ve von Neumannově schématu je možné ještě vyznačit **dva další moduly vzniklé spojením předcházejících modulů**:

- **Procesor:** Řadič + ALU
- **CPU - Central Processor Unit (centrální procesorová jednotka):** Procesor + Operační paměť

### Princip činnosti počítače podle von Neumannova schématu

1. Do operační paměti se pomocí vstupních zařízení přes ALU umístí program, který bude provádět výpočet.
2. Stejným způsobem se do operační paměti umístí data, která bude program zpracovávat
3. Proběhne vlastní výpočet, jehož jednotlivé kroky provádí ALU. Tato jednotka je v průběhu výpočtu spolu s ostatními moduly řízena řadičem počítače. Mezivýsledky výpočtu jsou ukládány do operační paměti.
4. Po skončení výpočtu jsou výsledky poslány přes ALU na výstupní zařízení.

### Základní odlišnosti dnešních počítačů od von Neumannova schématu

- Podle von Neumannova schématu počítač pracuje vždy nad jedním programem. Toto vede k velmi špatnému využití strojového času. Je tedy obvyklé, že počítač zpracovává paralelně více programů zároveň - tzv. *multitasking*
- Počítač může disponovat i více než jedním procesorem
- Počítač podle von Neumannova schématu pracoval pouze v tzv. diskrétním režimu.
- Existují vstupní /výstupní zařízení I/O devices, která umožňují jak vstup, tak výstup dat (programu)
- Program se do paměti nemusí zavést celý, ale je možné zavést pouze jeho část a ostatní části zavádět až v případě potřeby

Sestává z:

- řídicí jednotky (control unit), která načítá instrukce programu
- výpočetní jednotky (processing unit), která je zpracovává; zahrnuje aritmeticko-logickou jednotku (arithmetic-logic unit, ALU) a nejrychlejší a zároveň nejmenší paměť, registry
- paměti, ve které jsou společně (v jednom sdíleném adresním prostoru) uložena data a spustitelný kód
- vstupů a výstupů (inputs, outputs) – ty dnes bývají součástí společného adresního prostoru (jsou tzv. memory-mapped)

Čistá von Neumannova architektura trpí výrazným výkonnostním nedostatkem, který se nazývá *Von Neumann bottleneck* (úzké hrdlo). Tím je neschopnost paralelně a nezávisle načítat data a program. Protože k oběma se přistupuje přes jednu sdílenou sběrnici (bus), musíme proto načítat postupně. Procesory dnes počítají podstatně rychleji, než jim je paměť schopna dodávat potravu, byly by proto nuceny čekat.

Komplexnější Harvardský model využívá dva oddělené adresní prostory, čímž se tomuto problému vyhýbá za cenu složitosti. V praxi se používá kombinace obou přístupů, upravená Harvardská architektura (modified Harvard architecture), kdy procesor z jednotného logického adresního prostoru načítá data a instrukce do dvou nezávislých vyrovnávacích pamětí (data cache, instruction cache) a pracuje s nimi dál odděleně.

## Reprezentace čísel v počítači

Vše uložené v počítači jsou jen nuly a jedničky. Počítače používají tzv. **dvojkovou soustavu**, jsou to digitální zařízení. Tento způsob se technicky dobře realizuje pomocí elektrického signálu (není napětí = 0; je napětí = 1) i pomocí mechanických prostředků (0 = není prohlubeň; 1 = je prohlubeň nebo výstupek). Jedna nula nebo jednička (něco je nebo není) je také **nejmenší jednotka informace**, která říká, který ze dvou stejně pravděpodobných stavů nastal.

### BITY A JEJICH POČET

Této *nejmenší jednotce informace* (je nebo není) se říká **1 bit** (značka malé b). Tvůrci počítačů počítali, kolik různých znaků lze zakódovat pomocí kolika (jak dlouhého řetězce) nul a jedniček. Pokud by byla k dispozici pouze jedna nula či jednička, mohli byste zakódovat jen dva znaky. Např. písmeno A by bylo 0, B pak 1. BABA = 1010 => pro praktickou potřebu málo.

### BAJT

Kolik nul a jedniček na jeden znak je tedy potřeba, aby bylo možné zakódovat celou abecedu, malá i velká písmena, číslice a ještě zbude rezerva? Tvůrci počítačů se shodli na 8 bitech (00000001). Počet různých kombinací z 8 nul a jedniček: 256 ( $2^8$  -> obecně platí, že kombinací zakódovaných dvěma znaky je  $2^N$ , kde N je počet bitů). Této kombinaci 8 nul a jedniček dali jméno **bajt** (z anglického byte) se značkou velké B. *Jeden bajt je tedy řetězec osmi bytů.*

Protože bajtů se do počítače vejde hodně, používají se násobné jednotky:

**1 kilobajt (KB)** je 1 024 bajtů (B)

**1 megabajt (MB)** je 1 048 576 B, tedy 1 024 KB

**1 gigabajt (GB)** je 1 073 741 824 B, tedy 1 048 576 KB, tedy 1 024 MB

**1 terabajt (TB)** je 1 073 741 824 KB, tedy 1 048 576 MB, tedy 1 024 GB

Pro jednoduchost stačí většinou uvažovat zaokrouhlené hodnoty, např. KB jako tisíc bajtů, MB jako milion bajtů a GB jako miliardu bajtů nebo tisíc MB.

ALU se nejlépe pracuje s čísly ve dvojkové (binární) soustavě s omezenou, pevnou délkou. Tuto posloupnost jedniček a nul nazýváme slovo (word).

Rozsah hodnot závisí na délce slova (počtu bitů/bajtů) exponenciálně: historické 8bitové slovo pojme  $2^8 = 256$  různých hodnot (v přirozených číslech tedy rozsah 0—255), současné 64bitové unese  $2^{64}$  = hodně moc :-)

Číslo 45 zapsané jako 8bitové slovo vypadá takto: 001011012, číslo 93: 010111012.

### VYJÁDŘENÍ ZÁPORNÝCH ČÍSEL

S celými čísly se díky metodě dvojkového doplňku (two's complement) počítá stejně jako s přirozenými, jen využijeme **nejvyšší (první) bit jako znaménko** (nula je plus a jednička minus). Tento systém má jednu nulu a jedno tak trochu „divné“ číslo.

Stále s 8 bity: nejmenší nezáporné číslo je 000000002 = 0, nejvyšší je 011111112 (127; máme o jeden bit méně prostoru). Záporná čísla začínají opět od „nuly“; ovšem už s nastaveným znaménkem: 100000002 a končí hodnotou 111111112. Mezi nejmenším a největším záporným číslem opět můžeme rozdělit 128 hodnot, jako jsme to udělali u kladných (0—127). Končíme -1, a proto začínáme -128.

### NEGACE A SČÍTÁNÍ (ODČÍTÁNÍ?)

Člověk si opatří opak čísla 45 převrácením bitů a *přičtením jedničky*. V binárním zápise inverze (doplňek) cifer 001011012 vyjde 110100102 a po inkrementaci získáme výsledek 110100112 = -45.

Na negaci -128 nemáme místo; při přičítání jedničky k inverzi 011111112 dochází k přetečení, kterého když si nevšimneme, získáváme opět -128.

Součet 93 a -45 získáme klasickým sčítáním pod sebou, jako známe pro desítkovou soustavu, jen přenášíme dvojku místo desítky.

01011101 <sub>2</sub>	93
11010011 <sub>2</sub>	-45
00110000 <sub>2</sub>	48

### ČÍSLA S PLOVOUCÍ ŘÁDOVOU ČÁRKOU (FLOATING-POINT NUMBERS)

Hodnoty příliš velké nebo neceločíselné se ukládají způsobem, který kopíruje tzv. vědeckou notaci. Například palec v centimetrech (2,54) by se zapsal jako  $254 \cdot 10^{-2}$ . Při ukládání v počítači stačí rozdělit přidělené místo (např. 64 bitů) na dvě (nestejné) části, do první (větší) zapsat mantisu 254 a do menší části za ní exponent -2.

## DIGITALIZACE PODROBĚJI, POTŘEBNÝ POČET BITŮ

Digitalizace = vzorkování původního (analogového) signálu a jeho záznam pomocí toku nul a jedniček.

V počítačových zařízeních se bity používají k záznamu všeho – textů, zvuků i videa. Jak již bylo zmíněno, zápisem pomocí dvou znaků o délce  $N$  pozic můžeme zakódovat  $2^N$  možností, kde  $N$  je počet bitů. Třemi bity můžeme zapsat  $2^3$ , tedy 8 možností, čtyřmi bity pak 16 možností, pěti 32 možností atd. Ve výpočetní technice se nejčastěji setkáme s těmito hodnotami:

**8 bitů** (tedy 1 B) umožňuje záznam  $2^8 = 256$  možností. Jeden B se dlouho používal pro záznam písmen v tzv. fontech True type, dnes používaná typová písma Open type většinou využívají 3 bajty na znak.

**16 bitů** (tedy 2B) umožňuje záznam  $2^{16} = 65\,536$  možností. Dnes se tato hodnota používá jen výjimečně.

**24 bitů** (tedy 3B) umožňuje záznam  $2^{24} = 16,8$  milionů možností. 24bitová barevná hloubka se často používá při záznamu obrázků i snímků videa.

**32 bitů** (4 B) nabízí cca 4,3 miliardy možností, používá se nejčastěji pro **kódování barev**.

## DVOJKOVÁ, DESÍTKOVÁ A ŠESTNÁCTKOVÁ SOUSTAVA

Způsob kódování čísel pomocí dvou znaků se v matematice nazývá dvojková nebo také binární soustava. Při práci se setkáme se zápisem i v šestnáctkové (hexadecimální soustavě).

V obvyklé **desítkové soustavě** se čísla zapisují pomocí deseti číslic (0..9). Např. číslo 1 623 je součtem  $1 \times 1\,000 + 6 \times 100 + 2 \times 10 + 3 \times 1$ . Číslo se tedy skládá z mocnin (řádů) deseti násobených jejich pozičními hodnotami.

$10^3$	$10^2$	$10^1$	$10^0$
1	6	2	3
1000	100	10	1

Ve **dvojkové soustavě** bude stejné číslo vyjádřeno pomocí mocnin čísla 2 násobených opět jejich pozičními hodnotami, ty však nyní mohou nabývat pouze stavů, 0 a 1. Číslo 1 623 je vyjádřeno takto:

$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
1	1	0	0	1	0	1	0	1	1
512	256	128	64	32	16	8	4	2	1
$512 + 256 + 0 + 0 + 32 + 16 + 0 + 4 + 2 + 1 = 1623$									

V **šestnáctkové soustavě** se používá 16 znaků, číslic 0..9 a písmena A..10, B..11, C..12, D..13, E..14, F..15. Číslo 1 623 je vyjádřeno takto:

$16^2$	$16^1$	$16^0$
6	5	7
1536	80	7
$1536 + 80 + 7 = 1623$		

Neboli  $11001010111_2$  v dvojkové soustavě je  $1623_{10}$  v desítkové soustavě a  $657_{16}$  v šestnáctkové soustavě.

**Převod čísla z desítkové do dvojkové soustavy** není složitý: stále číslo dělíme dvěma a zbytek po dělení zapisujeme zprava jako vyjádření čísla ve dvojkové soustavě. Např.  $1623:2=811$ , zbytek 1 (zapíšeme 1),  $811:2=405$ , zbytek 1 (zapíšeme 1);  $405:2=202$ , zbytek 1 (111),  $202:2=101$ , zbytek 0 (0111),  $101:2=50$ , zbytek 1 (10111);  $50:2=25$ , zbytek 0 (010111),  $25:2=12$ , zbytek 1 (1010111),  $12:2=6$ , zbytek 0 (01010111),  $6:2=3$ , zbytek 0 (001010111),  $3:2=1$ , zbytek 1 (1001010111),  $1:2=0$ , zbytek 1 (11001010111),  **$1623_{10} = 11001010111_2$** .

**Převod čísla z desítkové do šestnáctkové soustavy** probíhá stejně jako u dvojkové, pouze samozřejmě původní číslo dělíme 16 a opět zapisujeme zbytek po dělení zprava.

## Reprezentace znaků v počítači, znakové sady a jejich kódování

Se znaky (písmeny, číslicemi, interpunkcí, mezerami, řídicími znaky atd.) pracujeme jako s čísly, každému přiřadíme systematicky jednu pozici v potřebném rozsahu. Např. americký standard ASCII si vystačil se 128 pozicemi (7 bity, osmý se používal při přenosu dat pro kontrolu), ale neobsahoval diakritiku. Jazyky s ideogramy naopak potřebují uložit tisíce znaků. Evropané se v mnoha svých standardech vešli buď do 7, nebo do 8 bitů.

**Znakovými sadami myslíme přiřazení čísel znakům**, např. v ASCII se „e“ skrývá pod číslem 101, mezera má číslo 32, rovnítko 61 a zalomení řádku 10. V našem prostředí se používalo mnoho sad, ale stále přežívá microsoftův CP1250 (code page) a v mailech standard ISO-8859-2. Celosvětově se ale prosazuje obří **znaková sada Unicode** (nadmnožina ASCII), která zatím zahrnuje přes 110 000 znaků ze všech možných „abeced“.

Kódováním dáváme číslu, které zastupuje znak, formu. 7- a 8bitové znakové sady stačí zakódovat do sekvence bajtů. Text v Unicode může sestávat z latinky spolu s rozsypaným čajem a výslovností v IPA. Existují kódování s pevným, nebo proměnlivým počtem bajtů. **Zastaralé kódování UCS-2** používá 16 bitů, zaznamenává proto pouze podmnožinu 65 536 znaků. **Populární kódování UTF-8** zapisuje každý znak 1 bajtem, pokud má znak číselnou hodnotu do 127 (a je v tomto kompatibilní s ASCII), anebo až 4 bajty, kde první má vždy hodnotu větší než 128. České znaky s háčky a čárkami se vejdou do 2 bajtů, valná většina čínštiny do 3, ve 4 jsou jen rarity.

# Algoritmy a datové struktury (pole, spojový seznam, fronta, zásobník)

## ALGORITMUS

Algoritmus je schematický postup (přesný návod) pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků. Ačkoliv se dnes tento pojem používá především v informatice a přírodních vědách obecně, tak je jeho působnost daleko širší (kuchyňské recepty, návody a postupy...).

### Způsob zápisu algoritmu:

- *lineárním vyjádřením* (pomocí přirozeného jazyka doplněného klíčovými slovy – tj. metajazyk nebo pseudokód; zápisem v některém z programovacích jazyků)
- *grafickým zápisem* (v podobě vývojového diagramu; v podobě strukturogramu)

### Vlastnosti algoritmů

- **Konečnost (finitnost)** – algoritmus má konečné množství kroků.
- **Obecnost (univerzálnost)** – algoritmus řeší všechny úlohy daného typu.
- **Elementárnost** – algoritmus se skládá z konečného počtu jednoduchých (elementárních) kroků
- **Určitost (determinovanost)** – všechny kroky algoritmu jsou přesně definovány, v každé situaci musí být naprosto zřejmé, co a jak se má provést, pro stejné vstupy dostaneme pokaždé stejné výsledky
- **Korektnost** – algoritmus skončí pro libovolná (korektní) data správným výsledkem v konečném množství kroků.
- **Výstup (resultativnost)** – algoritmus má alespoň jeden výstup, vede od zpracování hodnot k výstupu

## DĚLENÍ ALGORITMŮ

### Rekurzivní a iterativní algoritmy

**Iterativní algoritmus** je takový, který spočívá v opakování určité své části (bloku).

**Rekurzivní algoritmus** naproti tomu opakuje kód prostřednictvím volání sebe sama (obvykle na podproblémech menší velikosti). Každý rekurzivní algoritmus lze převést do iterativní podoby. Samotný převod často řeší automaticky kompilátor nebo virtuální stroj daného programovacího jazyka.

Výhoda rekurzivních algoritmů je v jejich snadno čitelném a kompaktním zápisu. Nevýhodou je spotřeba dodatečných systémových prostředků pro udržení jednotlivých rekurzivních volání.

- **Iterativní** – *bubble sort, insertion sort*
- **Rekurzivní** – *merge sort, quicksort*

### Deterministické a nedeterministické algoritmy

**Deterministický** je takový algoritmus, který má v každém svém kroku právě jednu možnost, jak pokračovat. **Nedeterministický** jich má více. Příkladem může být *deterministický a nedeterministický automat*.

### Sériové, paralelní a distribuované algoritmy

**Sériový algoritmus** vykonává všechny kroky v sérii (jeden po druhém), **paralelní algoritmus** tyto kroky vykonává zároveň (ve více vláknech) a **distribuovaný algoritmus** kroky vykonává zároveň na více strojích.

## SLOŽITOST ALGORITMU

Pod složitostí algoritmu rozumíme dobu provádění algoritmu => časovou složitost a rozsah použité operační paměti – paměťovou náročnost (složitost).

### SLOŽITOST ALGORITMU × SLOŽITOST PROBLÉMU

**Složitostí algoritmu** rozumíme složitost konkrétní instance zadaného algoritmu (implementovaného v nějakém programovacím jazyce) Algoritmy pracující s lepší než exponenciální/faktoriálovou složitostí označujeme jako efektivní.

**Složitostí problému** rozumíme složitost optimálního algoritmu konkrétně řešícího zadaný problém.

## Asymptotická složitost algoritmu

- snaha vyjádřit složitost jako funkci vstupu
- říct, jak roste složitost algoritmu vzhledem k rostoucímu vstupu
- složitost algoritmu udává, jak je daný algoritmus rychlý (kolik provede elementárních operací) vzhledem k množině vstupních dat
- můžeme uvažovat:
  - složitost v nejlepším případě
  - složitost v průměrném případě
  - složitost v nejhorším případě
  - amortizovanou složitost: určuje časovou složitost algoritmu v sekvenci nejhorších možných vstupních dat – nevyužívá pravděpodobnosti, a proto je zaručená

*Asymptotická složitost* algoritmu charakterizuje počet provedených operací v závislosti na velikosti dat. Například pokud procházíme pole, pak složitost bude lineární (na každý prvek připadá konstantní množství operací), pokud jej ovšem řadíme například *bubble sortem*, pak složitost bude kvadratická (na  $n$  prvků bude připadat  $n^2$  operací).

**ČASOVÁ SLOŽITOST:** funkce, která každé množině vstupních dat přiřazuje počet operací vykonaných při výpočtu podle daného algoritmu. Kolik času potřebuje program na své vykonání.

**PROSTOROVÁ SLOŽITOST:** závislost paměťových nároků algoritmu na vstupních datech; kolik paměti (úložného prostoru) program ke svému běhu potřebuje.

## **ZÁKLADNÍ SLOŽITOSTI**

název	složitost	příklad
konstantní	$O(1) = c$	pole – $n$ -tý prvek
logaritmická	$O(\log n)$	binární vyhledávání
lineární	$O(n)$	hledání v neseřazených posl.
lineárnělogaritmická	$O(n \cdot \log n)$	
kvadratická	$O(n^2)$	
kubická	$O(n^3)$	
obecná polynomiální	$O(n^c)$	
exponenciální	$O(c^n)$	
faktoriálová	$O(n!)$	

## Třída složitosti

*Třída složitosti* stanovuje obtížnost rozhodnutelnosti daného problému na *Turingově stroji*.

**Třída P** – obsahuje problémy rozhodnutelné v polynomiálním čase.

- Má daný *graf* kostru o velikosti maximálně  $k$ ?
- Existuje v daném acyklickém grafu mezi uzly  $a$  a  $b$  cesta, jejíž délka je nejvýše  $k$ ?

**Třída NP** – obsahuje problémy, které jsou rozhodnutelné pomocí nedeterministického Turingova stroje v polynomiálním čase – tzn. jsme schopni ověřit jejich řešení v polynomiálním čase.

- Lze daný graf obarvit maximálně  $k$  barvami?
- Existuje v daném grafu hamiltonovská kružnice?
- Existuje v daném grafu klika o alespoň  $k$  vrcholech?

## vyhledávání, řazení;

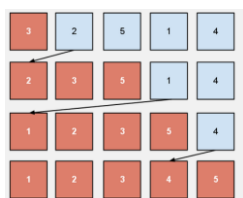
u řadicích (třídících) algoritmů (zápisy z Pythonu); např. seznam (9,8,3,5) se má seřadit podle velikosti

[8, 2, 7, 14, 3, 1]  
[2, 7, 8, 3, 1, 14]  
[2, 7, 3, 1, 8, 14]  
[2, 3, 1, 7, 8, 14]  
[2, 1, 3, 7, 8, 14]  
[1, 2, 3, 7, 8, 14]

**Bublíkové algoritmy (bubble sort)** – „probublávání vyšších hodnot nahoru“; srovnávání a prohazování sousedů, opakovaně prochází program (nesprávně prohazuje, jinak je nechá být); po  $i$  iteracích je nejvyšších  $i$  členů na svém místě; řazení posledních částí seznamu je rychlejší, třídí se jen část seznamu; kvadratická složitost

**Řazení výběrem (select sort)** – projdeme dosud nezařazenou část pole a vybereme nejmenší prvek a snažíme se jej dát na správnou pozici; nejmenší prvek zařadíme na aktuální pozici (výměnou); porovnávání s celým programem a zařazujeme; kvadratická složitost

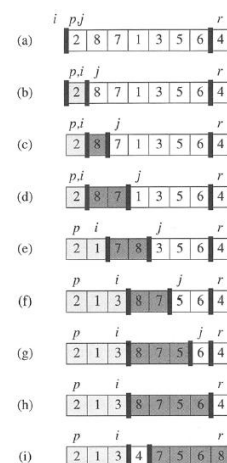
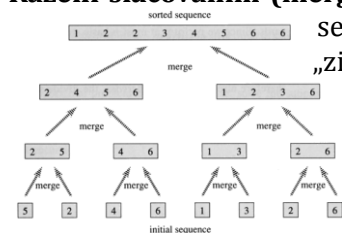
Find Smallest 21 73 32 10 42  
Swap with First 10 73 32 21 42  
Find Smallest 10 21 32 73 42  
Swap with First 10 21 32 73 42



**Řazení vkládáním (insert sort)** – prefix pole udržujeme seřazený, každá další hodnota zařazena tam, kam patří (porovnávám ho se sousedy a probublávám); neúčinné moc dlouhé; kvadratická složitost, závisí na vstupu – rychlé, pokud už máme část seřazenou

**Quicksort** – rekurzivní algoritmus, vybereme „pivota“ (ústřední prvek) a pole rozdělíme na dvě části (tj. větší než pivot a menší než pivot); obě části se pak seřazují nezávisle; poté vybíráme dalšího pivota; pokud vybereme špatného pivota, je postup pomalý jako minulé přístupy (obr. č. 4 vpravo) ->>

**Řazení slučováním (merge sort)** – rekurzivní algoritmus, rozdělíme pole na poloviny a ty seřadíme (merge sort), po seřazení je zařadíme do jednoho pole – „zipování“; vždy efektivní



**Radix sort** – předtím se porovnávaly dvě hodnoty, u RS aplikujeme doplňující předpoklady (jde jen o číslo) -> aplikovatelné na krátká čísla

329 720 720 329  
457 355 329 355  
657 436 436 436  
839 457 839 457  
436 657 355 657  
720 329 457 720  
355 839 657 839

## DATOVÉ STRUKTURY

Datová struktura je konkrétní způsob uložení dat v paměti počítače. Kvůli různým technickým omezením není možné vytvořit jednu univerzální a za všech okolností efektivní datovou strukturu. Proto bylo navrženo mnoho rozličných datových struktur, které jsou používány tam, kde se zdají být nejvhodnější.

Požadavky na datové struktury jsou často protichůdné. Někdy je prioritou rychlost přístupu k datům, jindy je přednější maximální úspora paměti. Volba té či oné datové struktury by měla být vždy učiněna s přihlédnutím k účelu, k němuž má být struktura nejčastěji použita.

Operace na datových strukturách – přístup k  $n$ -tému prvku, vložení prvku, odstranění  $n$ -tého prvku.

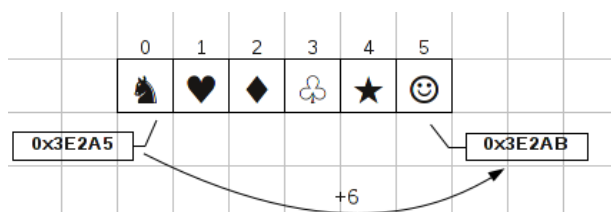
### Indexování:

- klíč – jednoznačný identifikátor záznamu
- pro rychlejší vyhledávání nad klíčem vybudovat index (index = seřazená posloupnost hodnot pro jeden klíč, s ukazateli na záznam)
- indexování indexu – rychlé vyhledávání
- příkl. korpus



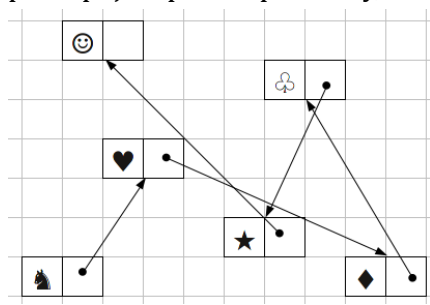
## Pole (array)

- uložení  $n$  hodnot v **souvislé paměťové oblasti**, souvislá lineární posloupnost homogenních prvků v paměti, jejichž pravidelné uspořádání zaručuje rychlý přístup ke každému z nich
- **indexování číslem** ( $\rightarrow$  adresou v paměti) - každý prvek je jednoznačně určen adresou prvního prvku pole a dalším přirozeným číslem (tzv. *indexem*) označujícím vzdálenost požadovaného prvku od začátku pole. Absolutní adresa prvku je pak vypočtena jako součet těchto dvou čísel.
- **prostorová složitost**:  $O(c \cdot n)$ , kde  $n$  je počet záznamů,  $c$  velikost jednoho záznamu (konstanta)
- **přístup k  $n$ -tému prvku**:  $O(1)$  (adresování v paměti) – probíhá v konstantním čase
- **vložení prvku**:  $O(n)$  (realokace)
- **odstranění prvku**:  $O(n)$  (realokace)
- rychlejší přístup, pomalejší modifikace



## Seznam (list)

- uložení  $n$  hodnot v **nesouvislé paměťové oblasti**
- **pojme libovolné množství prvků (omezené množstvím dostupné paměti)**
- s každou hodnotou je asociovaný **ukazatel na následníka**
- **prostorová složitost**:  $O(c \cdot n)$ , kde  $n$  je počet záznamů,  $c$  velikost jednoho záznamu (konstanta) – ale: záznam zde musí obsahovat i ukazatel na následníka  $\rightarrow$  vyšší paměťové nároky než pole
- **časová náročnost** některých operací je vyšší než u pole (vyhledávání, vkládání, mazání), neznáme přesná umístění všech prvků (každý může být na libovolném místě, není řazen lineárně), seznam je tak často nutné procházet
- **přístup k  $n$ -tému prvku**:  $O(n)$  (lineární průchod seznamem)
- **vložení prvku**:  $O(1)$  (přepojení ukazatelů)
- **odstranění prvku**:  $O(1)$  (přepojení ukazatelů)
- pomalejší přístup, rychlejší modifikace
- **seznam v Pythonu**: seznam prvků oddělených čárkami a uzavřený do hranatých závorek; k prvkům přistupujeme přes např. indexy, seznamy lze modifikovat (na rozdíl od řetězců)



## Pole vs. seznam:

- Pole: rychlejší přístup, pomalejší modifikace
- Seznam: pomalejší přístup, rychlejší modifikace
- Vhodnost použití závisí na datech



## Hashování

- v předchozích příkladech bylo indexem vždy číslo
- můžeme ovšem chtít indexovat např. řetězcem („ke každému slovu přiřadit četnost“)
- nutná transformace nečíselné hodnoty na číslo – vytvoření tzv. *hashe* pomocí *hashovací funkce*

### Hashovací funkce

$h : D \rightarrow N$ , kde  $D$  je doména dat; požadavky:

- rychlá
- deterministická (pro stejná data vždy stejný výsledek)
- uniformní (výstupní hodnoty mají přibližně stejnou pravděpodobnost, všechny mají stejnou velikost)
- výstup volitelné délky
- obtížná reverzibilita (použití: hesla)
- minimální změna vstupu vyvolá velkou změnu výstupu

Hashovací funkci nazýváme perfektní, je-li její zúžení na konkrétních vstupních datech injektivní.

- obecně je hashovací funkce vždy neinjektivní a vznikají kolize
- důležitý požadavek: neměnnost vstupních dat po vytvoření hashe (*immutability*)

**Hashovací tabulka** – je datová struktura, která slouží k ukládání dvojic klíč-hodnota. Hashovací tabulka kombinuje výhody vyhledávání pomocí indexu (složitost  $O(1)$ ) a procházení seznamu (nízké nároky na paměť).

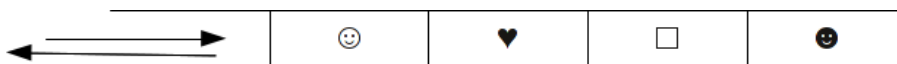
## Fronta

- datová struktura typu **FIFO** (First In First Out)
- slouží k ukládání a výběru dat takovým způsobem, aby prvek, který byl uložen jako první, byl také jako první vybrán, uchovává prvky ve stejném pořadí, v jakém byly vloženy
- implementace pomocí pole nebo spojového seznamu
- Python: `list.append()`, `list.pop(0)`, ale pomale (collections.deque)!
- př. použití: prohledávání stromu do šířky

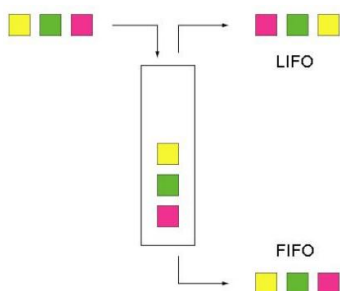


## Zásobník

- datová struktura typu LIFO (Last In First Out)
- poslední vložený prvek jde na výstup jako první, předposlední jako druhý a tak dále, uchovává prvky v opačném pořadí, než v jakém byly do zásobníku vloženy
- implementace pomocí pole nebo spojového seznamu
- využívá se především pro dočasné ukládání dat v průběhu výpočtu
- Python: `list.append()`, `list.pop()`
- př. použití: prohledávání stromu do hloubky



### LIFO vs. FIFO

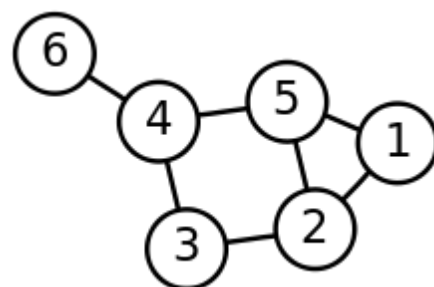


# Grafy, stromy, vlastnosti grafů, základní grafové algoritmy, prohledávání stromu do hloubky, do šířky

## Grafy a jejich teorie

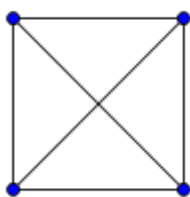
- teorie grafů zkoumá vlastnosti struktur, které se nazývají grafy
- graf je uspořádaná dvojice množiny vrcholů  $V$  a množiny hran  $H$   
 $V = \{1, 2, 3, 4, 5, 6\}$

$$E = \{\{1,2\}, \{1,5\}, \{2,3\}, \{2,5\}, \{3,4\}, \{4,5\}, \{4,6\}\}$$

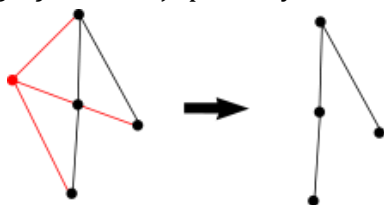


## Základní pojmy

- stupeň vrcholu** – udává počet hran, které z vrcholu vycházejí; u orientovaného grafu se rozlišují na vstupní a výstupní
- úplný graf** – graf, ve kterém jsou každé dva vrcholy spojené hranou



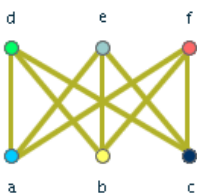
- podgraf** – obsahuje pouze vybrané vrcholy a hrany z daného grafu



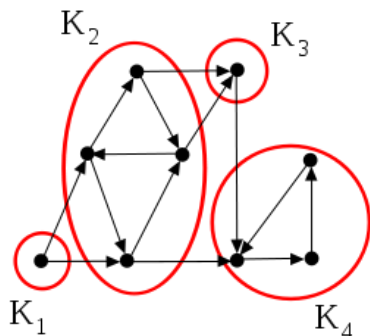
- isomorfismus** – dva grafy, jejichž hrany a vrcholy sobě odpovídají, ale mohou mít rozdílné označení a rozmístění bijekce:

Dva grafy  $G=(V,E)$  a  $G'=(V',E')$  jsou isomorfní, jestliže existuje bijektivní zobrazení

$f: V \rightarrow V'$  tak, že platí:  $\{x,y\} \in E$ , právě když  $\{f(x), f(y)\} \in E'$ .

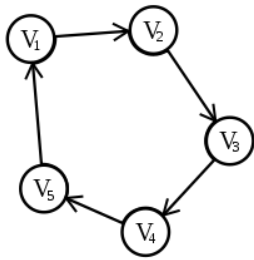


- silné souvislé komponenty** - maximální podgraf orientovaného grafu, mezi každými dvěma vrcholy existuje cesta tam i zpět

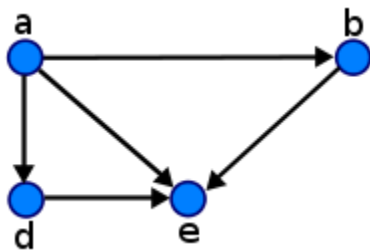


## Typy grafů

- **kružnice** – graf, který má stejný počet vrcholů a hran a každý vrchol je stupně 2, graf tvoří uzavřený okruh
  - kružnice se může vyskytovat i jako podgraf, v tomto případě se graf nazývá cyklický graf (v opačném případě se jedná o strom, viz dále)



- **cesta** – kružnice s jednou chybějící hranou
- **souvislý graf** – pro každé dva vrcholy  $x, y$  platí, že existuje alespoň jedna cesta z  $x$  do  $y$
- **strom** – jednoduchý souvislý graf bez kružnic
- **cyklus v grafu** – podgraf, který je kružnicí
- **cesta v grafu** – podgraf, který je cestou
- **kostra grafu** – strom spojující všechny vrcholy
- **orientovaný graf** – graf, jehož hrany jsou orientovány v určitém směru jako uspořádaná dvojice



$$V = \{a, b, d, e\}$$

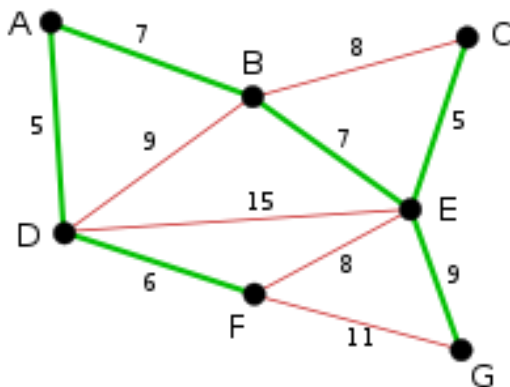
$$E = \{(a,b), (a,d), (a,e), (b,e), (d,e)\}$$

- **ohodnocený graf** – graf, jehož hrany jsou ohodnoceny jako vzdálenost z vrcholu  $x$  do vrcholu  $y$

## Základní grafové algoritmy

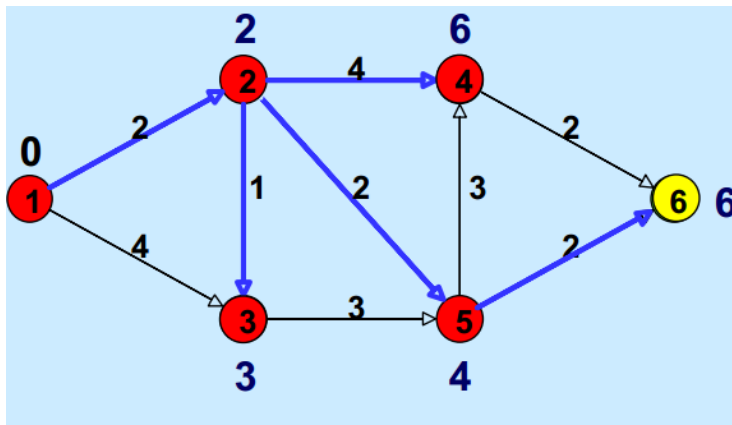
### Kruskalův algoritmus

- slouží k vyhledání minimální kostry ohodnoceného grafu
- postupně vybíráme nejkratší hrany, aniž by došlo k vytvoření kružnice
- celková váha (součet délek) hran grafu je minimální



## Dijkstrův algoritmus

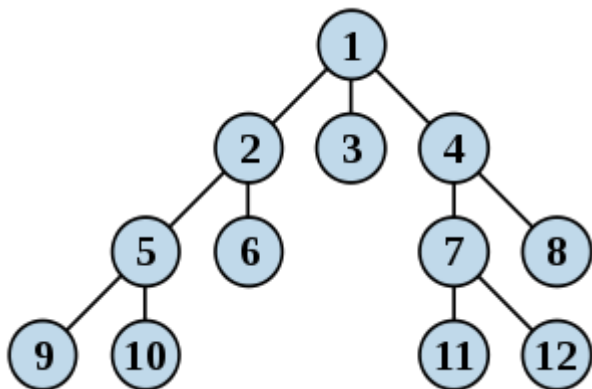
- slouží k nalezení nejkratší cesty v grafu
- pro každý vrchol si pamatuje délku nejkratší cesty, kterou se k němu dá dostat
- prochází množinu nenavštívených uzlů, dokud není prázdná



## Prohledávání grafu

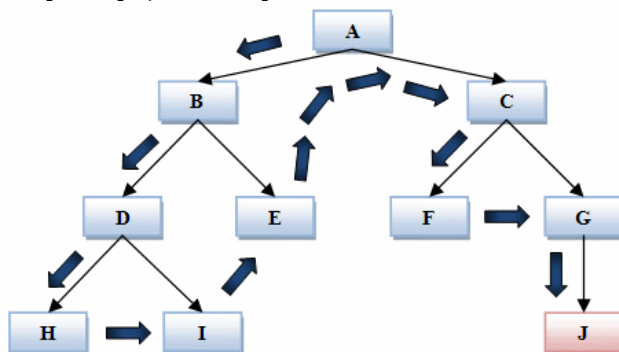
### Prohledávání grafu do šířky

- BFS = Breadth First Search
- Prochází graf po vrstvách



### Prohledávání grafu do hloubky

- DFS = Depth First Search
- postupuje dál od počátečního uzlu



## ***Základní pojmy z algoritmické složitosti (časová a prostorová složitost algoritmu, asymptotická složitost a třídy složitosti, složitost algoritmu a složitost problému)***

Algoritmická složitost vyjadřuje výkon, případně náročnost programu/algoritmu. Jde o matematickou funkci představující závislost sledovaného parametru na množství vstupních dat. Rozlišujeme

- **časovou složitost** – kolik času potřebuje program na své vykonání, spotřeba času v závislosti na vstupních datech, je obvykle kritičtější (prostor si lze koupit, čas nikoliv)
- **paměťovou složitost** – prostorová, kolik paměti (úložného prostoru) program ke svému běhu spotřebuje

Rozlišujeme **třídy složitosti** – není podstatná přesná konkrétní složitost, stačí charakterizovat třídu. Počítačové algoritmy klasifikujeme na zákl. **asymptotické složitosti**. Zjišťuje, jakým způsobem se bude chování algoritmu měnit na v závislosti na změně velikosti vstupních dat. Zapisuje se pomocí Omikron notace. Můžeme uvažovat složitost v nejlepším případě, průměrném případě, nejhorším případě nebo amortizovanou složitost. V mnoha případech (ale ne vždy) nás zajímá hlavně složitost v nejhorším případě. Základní složitosti:

seřazené od „nejmenší“, s užitím tzv. O-notace

název	složitost	$n = 10^2, c = 10$	$n = 10^6, c = 10$
konstantní	$O(1) = c$	10	10
logaritmická	$O(\log n)$	6,64	19,9
lineární	$O(n)$	100	1 000 000
lineárnělogaritmická	$O(n \cdot \log n)$	664	19 900 000
kvadratická	$O(n^2)$	10 000	$10^{12}$
kubická	$O(n^3)$	1 000 000	$10^{18}$
obecná polynomiální	$O(n^c)$	$10^{20}$	$10^{600}$
exponenciální	$O(c^n)$	$10^{100}$	$10^{1000000}$
faktoriálová	$O(n!)$	moc	strašně moc:)

**Složitost algoritmu** – složitost konkrétní instance zadaného algoritmu (implementovatelného v nějakém programovacím jazyce) Algoritmy pracující s lepší než exponenciální/faktoriálovou složitostí označujeme jako efektivní.

**Složitost problému** – složitost optimálního algoritmu korektně řešícího zadaný problém.

**Amortizovaná složitost** – průměrný čas potřebný pro vykonání určité operace v sekvenci operací v nejhorším případě. Na rozdíl od časové složitosti v průměrném případě nevyužívá pravděpodobnosti.

### **Vztah asymptotické složitosti a výkonu HW**

- uvažujme jeden krok výpočtu jako 1 s, 10 min odpovídá 600 krokům
- lineární algoritmus složitosti  $3 \cdot n$ : 10 min stačí na vykonání pro vstup velikosti 200
- exponenciální algoritmus  $2n$ : 10 min stačí na vykonání pro vstup velikosti 9 ( $2^9 = 512$ ,  $2^{10} = 1024$ )
- uvažujme dvakrát výkonnější HW: jeden krok výpočtu = 0,5 s, 10 min odpovídá 1 200 krokům
- lineární algoritmus složitosti  $3 \cdot n$ : 10 min stačí na vykonání pro vstup velikosti 400
- exponenciální algoritmus  $2n$ : 10 min stačí na vykonání pro vstup velikosti 10 ( $2^{11} = 2048$ )

# Počítačová lingvistika, korpusová lingvistika

## Významné úkoly v NLP:

- analýza přirozeného jazyka – morfologická, syntaktická, sémantická
- generování přirozeného jazyka
- syntéza a rozpoznávání řeči
- strojový překlad (Machine translation)
- odpovídání na otázky (Question answering)
- získávání informací (Information retrieval)
- korektura textu (Spell-checking, Grammar checking)
- extrakce informací (Information extraction)
- výtah z textu (Text summarization)
- určení typu dokumentu (Text Classification/Clustering)

## Formální jazyky vs. přirozené jazyky

---

**Struktura jazyka** zahrnuje informace o:

- co jsou slova (z jakých znaků, jaké slovní tvary a jejich složky)
- jak se slova (větné složky) kombinují do vět
- co slova označují, jaké jsou jejich lexikální významy
- jak se význam věty skládá z významů slov a slovních spojení (větných složek)

**Zpracování jazyka dále potřebuje:**

- obecnou (encyklopedickou) znalost světa (ontologie)
- inferenční mechanismus
- znalost komunikační situace

**Roviny analýzy jazyka** – fonetická, morfologická, syntaktická, sémantická, pragmatická; kontextová, znalost zákl. ontologie, jazykové metaznalosti

## PŘIROZENÝ JAZYK VS. FORMÁLNÍ JAZYK

**přirozený jazyk:** přirozený jazyk je jazyk, který je mluvený nebo napsaný lidmi pro univerzální komunikaci

**formální jazyk:** množina konečných slov (tj. slov konečné délky) nad určitou abecedou. Místo výrazu "slovo" se někdy užívá výraz "řetězec". Definice pojmu *formální jazyk* se může měnit podle toho, v jakém kontextu a v jakém vědním oboru jej používáme.

Pro potřeby matematické teorie jazyků je třeba definovat formální jazyk abstraktně jako matematický model. Využití formálních jazyků: v oblasti programovacích jazyků k definici syntaxe programovacího jazyka; formální jazyky jsou základním kamenem teoretické informatiky. Teorie formálních gramatik se využívá například v oblasti tvorby překladačů a kompilátorů.

## ALGEBRA NA MNOŽINĚ SLOV NAD DANOU ABECEDOU

*abeceda*

- konečná množina symbolů (znaků, písmen)
- značíme ji (množinu)  $\Sigma$  [sigma]
- prvky abecedy = znaky abecedy (případně taky písmena nebo symboly)
- příkladem abecedy je třeba množina {a, b}, slovem nad touto abecedou je např. *ababa* (abeceda má jen dvě písmena)
- další příklad abecedy: množina číslic {0, 1, ..., 9}, nebo prázdná množina  $\emptyset$

*jazyk*

- jazyk nad abecedou  $\Sigma$  je libovolná množina slov nad  $\Sigma$  (jazyky nad  $\Sigma$  jsou tedy právě podmnožiny  $\Sigma^*$ )
- např. {10, 1, 011101} je jazyk nad abecedou {0, 1}

*slovo (řetězec)*

- libovolná konečná posloupnost prvků abecedy  $\Sigma$
- značíme  $w, v$
- např.  $w = aabab$

### délka slova

- počet členů posloupnosti  $w$
- značíme  $\#w, |v|$
- počet výskytů znaku  $a$  ve slově  $w$  značíme  $\#_a(w)$
- počet znaků  $b$  ve slově  $abaaba$ :  **$\#b(abaaba) = 2$**

### prázdné slovo

- prázdná posloupnost znaků, která má nulovou délku
- značíme  $\varepsilon$

$\Sigma^*$  = množina všech slov nad abecedou (tj. i prázdné slovo)

$\Sigma^+$  = množina všech neprázdných slov nad abecedou

$$\{a\}^* = \{\varepsilon, a, aa, aaa, aaaa, \dots\}$$

$$\{a\}^+ = \{a\}^* \setminus \{\varepsilon\}$$

$$\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\}$$

### zrcadlový obraz (reverze) slova

- zapsáním znaků v opačném pořadí
- značíme  $w^R$
- $\Sigma = \{a, b, c\}$ ,  $w = abcabc$ ;  $w^R = cbacba$
- $w = \text{automobil}$ ;  $w^R = \text{libomotua}$

### zřetězení (konkatenace) slov

- binární operace, spojení dvou nebo více slov za sebou
- $u.v = uv$
- např. zřetězením slova  $abba$  .  $bba = abbabba$
- $\Sigma = \{a, b, c\}$ ,  $w_1 = acc$ ,  $w_2 = abab$ ;  $w_1.w_2 = w_3 = abcabc$
- znak zřetězení  $(.)$  se v zápisu většinou vynechává

### $i$ -tá mocnina slova

- unární operace, „ $i$ -tá“ mocnina slova
- $u^0 = \varepsilon$
- $u^{i+1} = u.u^i$
- $(abc)^3 = abcabcabc$

## OPERACE NAD JAZYKY

důrazné odlišení operací nad slovy a operací nad jazyky! (zřetězení, mocnina v obou příkladech stejné)

### jazyk nad abecedou $\Sigma$

- libovolná množina  $L$  jeho slov, tj. libovolná  $L \subseteq \Sigma^*$
- pokud je  $\Sigma$  neprázdnou konečnou množinou, je  $\Sigma^*$  nekonečná (přesněji *nespočetná*) a také jazyků je nekonečně mnoho (*nespočetně* mnoho)

### zřetězení (konkatenace)

- zřetězení jazyků  $K$  a  $L$  je  $KL = \{w_1 w_2 ; w_1 \in K, w_2 \in L\}$
- asociativní operace
- pro libovolný jazyk  $L$  definujeme mocniny  $L^k$  pro  $k = 0, 1, 2, 3, \dots$  takto:  $L^0 = \{\varepsilon\}$ ,  $L^1 = L$ ,
- $L^k = LL \dots L$  ( $k$ -krát)

### iterace

- iterace jazyka  $L$  je  $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots$
- **Iterace jazyka  $L$  je jazyk  $L^* = \bigcup_{i=0}^{\infty} L^i$ .**

### komplement (doplňěk)

- komplement jazyka  $L$  je  $\text{co}(L) = \{w; w \in L\}$

### sjednocení jazyků - $K$ a $L$ je $K \cup L = \{w; w \in K \text{ nebo } w \in L\}$

- do sjednocení padnou všechna slova, která náleží alespoň do jednoho ze sjednocovaných jazyků

### průnik jazyků - $K$ a $L$ je $K \cap L = \{w; w \in K \text{ a zároveň } w \in L\}$

příklad:

$$L_1 = \{a, ab, aab\}, L_2 = \{w \in \Sigma^* ; \#_a(w) = 3\}, L_3 = \{a^i b^i ; i \in \mathbb{N}\}$$

$$L_1 \cap L_3 =$$

$$L_2 \cap L_3 =$$



## GRAMATIKA

- model, který generuje jazyk; systém pravidel
- gramatika  $G$  je čtveřice  $(N, \Sigma, P, S)$ , kde:
  - $N$  – neprázdná konečná množina neterminálů
  - $\Sigma$  – konečná množina terminálů –  $N \cap \Sigma = \emptyset$
  - $P$  – konečná množina pravidel
  - pravidlo  $(\alpha, \beta)$  - zapisujeme ve tvaru  $\alpha \rightarrow \beta$  (čteme:  $\alpha$  přepiš na  $\beta$ )
  - $S$  – počáteční neterminál (kořen gramatiky)

## Pojmy

- **Terminál:** prvotní symbol daného jazyka; symbol z množiny symbolů jazyka, který už v procesu dané analýzy nenahrazujeme; jde o cílový termín, který je součástí nějakého řetězce jazyka (a, b, c)
- **Neterminál:** prvotní symbol daného jazyka; symbol definovaný pomocí pravidel jazyka (A, B, C)
- **Začáteční symbol:** jeden z neterminálů, ze kterého začíná generování vět jazyka, případně analýza věty
- **Přepisovací pravidla:** jsou základem pro generování vět jazyka; pravidla přepisujeme tak dlouho, dokud nedostaneme řetězec terminálů

## Příklad:

Mějme gramatiku  $(N, \Sigma, P, S)$ , kde:

$\Sigma = \{a, b\}$

$N = \{S, A\}$

$P = \{S \rightarrow A, A \rightarrow AA, A \rightarrow a\}$

Příklady odvození z této gramatiky jsou:

$S \Rightarrow A \Rightarrow a$

$S \Rightarrow A \Rightarrow AA \Rightarrow aA \Rightarrow aAA \Rightarrow aaA \Rightarrow aaa$

# Chomského hierarchie jazyků, gramatiky, automaty

## CHOMSKÉHO HIERARCHIE

- podle Noama Chomského
- gramatiky rozděleny do 4 skupin (typů) na základě omezení na tvar pravidel
- gramatika vyššího typu je vždy současně i gramatikou nižšího typu
- mechanismus vyčerpávajícím způsobem popisující jazyk
- matematický model gramatiky
- nástroj, který lze aplikovat například v lingvistice
- gramatika = systém pravidel; jazyk = množina řetězců

### TYP 0 (FÁZOVÉ GRAMATIKY)

Obsahuje pravidla v nejobecnějším tvaru, libovolná gramatika, neklade žádná omezení na tvar pravidel, povoluje přepis řetězců na řetězce, neomezené prepisovací systémy. Rekurzivně vyčíslitelné, ekvivalentní síle Turingova stroje.

### TYP 1 (KONTEXTOVÉ GRAMATIKY)

Pro každé pravidlo  $\alpha \rightarrow \beta$  platí  $|\alpha| \leq |\beta|$ , výjimku tvoří  $S \rightarrow \epsilon$ , pokud se  $S$  nevyskytuje na pravé straně žádného pravidla. V kontextových pravidlech lze neterminální symbol nahradit řetězcem pouze tehdy, je-li jeho pravým kontextem řetězec  $\beta$  a levým kontextem řetězec  $\alpha$ . Nepřipouštějí, aby neterminální symbol byl nahrazen prázdným řetězcem. Při generování věty nemůže dojít ke zkracování generovaných řetězců. Více termů na levé straně (kontext neterminálu), na levé straně se počet termů zmenšuje ( $ASB \rightarrow AAaBB$ ). Umí  $a^n b^n c^n$ .

### TYP 2 (BEZKONTEXTOVÉ GRAMATIKY)

Každé pravidlo je tvaru  $A \rightarrow \alpha$ , kde  $|\alpha| \geq 1$ , výjimku tvoří  $S \rightarrow \epsilon$ , pokud se  $S$  nevyskytuje na pravé straně žádného pravidla. Nahrazení neterminálního symbolu  $A$  řetězcem lze provést bez ohledu na okolí, v němž by se neterminální symbol mohl vyskytovat. Jsou nejzajímavější pro popis syntaktické stavby přirozených jazyků. Neterminál lze přepsat na cokoliv ( $S \rightarrow aSb$ ), jsou ekvivalentní síle zásobníkových automatů, umí  $a^n b^n$ , neumí  $a^n b^n c^n$ .

### TYP 3 (REGULÁRNÍ GRAMATIKY)

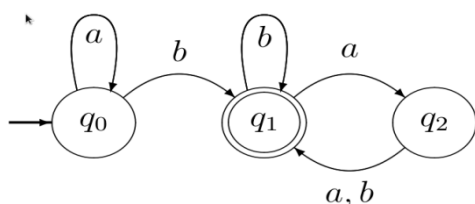
Každé pravidlo je tvaru  $A \rightarrow aB$  nebo  $A \rightarrow a$ , výjimku tvoří  $S \rightarrow \epsilon$ , pokud se  $S$  nevyskytuje na pravé straně žádného pravidla. Tzn. přepisujeme **neterminál**  $\rightarrow$  **terminál**[**neterminál**] ( $S \rightarrow aS$ ,  $S \rightarrow b$ ). Jsou ekvivalentní síle konečných automatů, neumí  $a^n b^n$ . Jediný možný neterminální symbol na pravé straně pravidla stojí zcela vpravo.

pozn.

Nejčastěji se pracuje s regulárními a bezkontextovými gramatikami a jazyky. Tyto typy gramatik jsou efektivně zpracovatelné na počítačích a jako základ se používají i při zpracování textů v přirozeném jazyce. Přirozený jazyk byl dlouho pokládán za bezkontextový, nyní je prokázáno, že obsahuje kontextové prvky.

## AUTOMATY

- abstraktní model charakterizující jazyky
- stavové systémy, které čtou po znacích slovo na vstupu, s přečtením každého znaku změni stav a na konci rozhodnou, zda slovo je akceptováno či nikoliv
- nejjednodušší je *konečný automat*



Automat  $A$  nad abecedou  $= \{a, b\}$  je znázorěn

stavovým diagramem vlevo. Jeho součástí jsou:

$Q = \{q_0, q_1, q_2\} \dots$  množina stavů

$\Sigma = \{a, b\} \dots$  (vstupní abeceda); symbol  $\epsilon$  značí prázdné slovo

$\delta : Q \times \Sigma \rightarrow Q \dots$  přechodová funkce, kde

(viz stavový diagram vpravo):  $\delta(q_0, a) = q_0$ ,  $\delta(q_0, b) = q_1$ ,

$\delta(q_1, a) = q_2$ ,  $\delta(q_1, b) = q_1$ ,  $\delta(q_2, a) = \delta(q_2, b) = q_1$

$q_0 \dots$  počáteční stav

automatu

$F = \{q_1\} \dots$  množina přijímacích stavů automatu (může jich

být více)

příklad:

$w_1 = ababab$

$(q_0, ababab) \vdash$

zkráceně (v řeči stavů):

$w_2 = abba$

$(q_0, abba) \vdash$

pozn.

Je-li  $A$  konečný automat nad abecedou  $\Sigma$ , pak jazykem přijímaným automatem  $A$  míníme množinu  $L(A)$  všech slov přijímaných automatem  $A$ , tj.  $L(A) = \{w \in \Sigma^*; A \text{ přijímá slovo } w\}$ .

## ANALÝZA ČINNOSTI KONEČNÉHO AUTOMATU

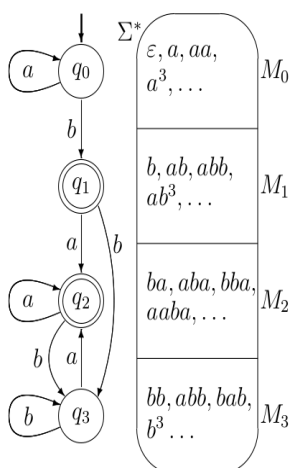
Je dán konečný deterministický automat  $A$  nad abecedou  $\Sigma$  s množinou stavů  $Q$ , přechodovou funkcí  $\delta: Q \times \Sigma \rightarrow Q$  a množinou přijímacích stavů  $F \subseteq Q$ .

Jazyk přijímaný automatem  $A$  je  $L(A)$ .

Iterace přechodové funkce (označovaná  $\delta^*$ ) působí na množině všech konfigurací jako  $(\text{stav}, \text{slovo}) \vdash^* (\text{stav}', \text{slovo}')$ .

Takový převod nazýváme **dílčím výpočtem automatu  $A$** .

Speciálně pro libovolné slovo  $w \in \Sigma^*$  má výpočet automatu  $A$  na slově  $w$  tvar  $(q_0, w) \vdash^* (q_i, \varepsilon)$ ; řekneme, že výpočet skončil ve stavu  $q_i$ .



Zřejmě platí:  $w \in L(A)$  právě když výpočet automatu  $A$  na slově  $w$  skončí v přijímacím stavu.

Příklad: Ukázkový automat  $A$  nad abecedou  $\Sigma = \{a, b\}$  je znázorněn stavovým diagramem vlevo.

(1) Ukažme si nejdříve dílčí výpočet našeho automatu na slově  $w = aababba$  odstartovaný ze stavu  $q_1$  mající 4 kroky:

$(q_1, aababba) \vdash$

Zkrácená forma:  $(q_1, aababba) \vdash^*$

## PŘEHLED MALÝCH AUTOMATŮ NAD ABECEDOU $\Sigma = \{A, B\}$

Existují právě dva jednostavové automaty nad  $\Sigma$ .

První z nich přijímá všechna slova jazyka  $\Sigma^*$

a druhý z nich žádné.

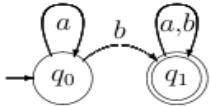
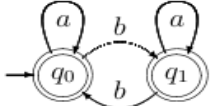
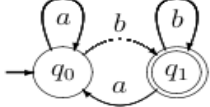
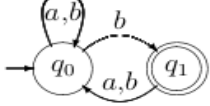
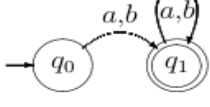
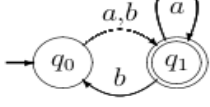

Dvojstavových automatů nad  $\Sigma$  se stavy  $q_0, q_1$  je celá řada, ovšem všechny ty, pro něž je  $F = \{q_0, q_1\}$  přijímají všechna slova jazyka  $\Sigma^*$ , zatímco ty, pro něž je  $F = \emptyset$  nepřijímají žádné slovo.

Dále zvláštní skupinu tvoří ty automaty, jejichž přechodová funkce dává  $\delta(q_0, a) = \delta(q_0, b) = q_0$ .

V nich je totiž stav  $q_1$  nedosažitelný, takže se redukuji na jednostavové (viz úvahy výše).

Tabulka níže zachycuje všechny "podstatné" dvoustavové automaty. Ostatní se z nich získají výměnou rolí znaků  $a$  a  $b$  či změnou přijímacího stavu z  $q_1$  na  $q_0$ :



AUTOMAT $\mathcal{A}$	JAZYK $L(\mathcal{A})$
	všechna slova obsahující aspoň jeden symbol $b$ , tj. $\{w \in \Sigma^*; \#_b(w) \geq 1\}$
	všechna slova obsahující lichý počet symbolů $b$ , tj. $\{w \in \Sigma^*; \#_b(w) \text{ je liché}\}$
	všechna slova mající příponu $b$ , tj. $\{ub; u \in \Sigma^*\}$
	všechna slova mající na konci lichý počet symbolů $b$ , tj. $\{b^n; n \text{ liché}\} \cup \{uab^n; u \in \Sigma^* \wedge n \text{ liché}\}$
	všechna slova kromě $\varepsilon$ , tj. $\Sigma^* - \{\varepsilon\}$
	$\{b^n; n \text{ liché}\} \cup \{uab^n; u \in \Sigma^* \wedge n \text{ sudé, včetně } n=0\}$
	všechna slova liché délky, tj. $\{w \in \Sigma^*; \#(w) \text{ je liché číslo}\}$

## Regulární výrazy a jejich využití ve zpracování přirozeného jazyka

### Regulární výrazy a jejich využití ve zpracování přirozeného jazyka

- řetězce popisující celou množinu řetězců (regulární jazyk)
- představil je americký matematik Stephen Cole Kleene
- metajazyk, jazyk specifikující jazyk
- regulární výrazy jsou dalším způsobem formální reprezentace regulárních jazyků – umožňují jej popsat jako výsledek kompozice několika jednoduchých operací nad jazyky (tedy nerekurzivní popis na rozdíl od gramatik a konečných automatů)
- skládají se z literálů textu, které se mají shodovat, a speciálních znaků, které nejsou součástí
- hledaného textu (sloužících pro popis alternativ, množin, počtů výskytů a přepínačů)
- **formální definice:**
  - 1)  $a$  je regulární výraz pro libovolný znakový literál (znak abecedy)  $a$ , popisující právě text  $a$ .
  - 2)  $\varepsilon$  je regulární výraz pro prázdný řetězec.
  - 3) Znak pro prázdnou množinu označuje prázdný jazyk.
  - 4) Pokud  $A$  a  $B$  jsou regulární výrazy, je  $AB$  regulární výraz, popisující zřetězení textů popsaných výrazy  $A$  a  $B$ .
  - 5) Pokud  $A$  a  $B$  jsou regulární výrazy, je  $A + B$  regulární výraz, popisující buď text popsaný výrazem  $A$ , nebo text, popsaný výrazem  $B$ .
  - 6) Pokud  $A$  je regulární výraz, pak  $A^*$  je regulární výraz, popisující libovolný počet opakování (včetně žádného opakování) textů popsaných výrazem  $A$ .
  - 7) Pokud  $A$  je regulární výraz, je  $(A)$  regulární výraz popisující stejný jazyk. (Závorky slouží pouze pro vyjasnění priorit.)
- **využití:** formální popis jevů přirozeného jazyka
  - vyhledávání textu
  - manipulace s textem
  - pokud chce uživatel v textu vyhledat nějaký řetězec, který nezná přesně nebo který může mít více podob, může zadat regulární výraz, který postihne všechny požadované varianty; program tak nalezne všechny části textu, které danému výrazu odpovídají
- každý znak vyhoví sám sobě + metaznaky (zástupné znaky) mají speciální význam
- kvantifikátory:
  - $*$  = libovolný počet opakování předchozího znaku
  - $+$  = libovolný počet opakování předchozího znaku  $> 0$

- ? = žádný nebo jeden výskyt předchozího znaku/výrazu
  - {n} = nkrát, {n, m} = n–mkrát, {n,} = n–∞krát
  - hladové × s ? – minimální počet znaků, které je třeba zachytit, aby došlo ke shodě s regulárním výrazem
  - . = libovolný znak
  - ^ = začátek řetězce / negace (někdy !)
  - [^a-z0-9] = libovolný znak kromě znaků v rozsahu a–z, 0–9
  - \$ = konec řetězce
  - | = logické or: nebo (disjunkce)
  - [] = výběr jednoho znaku z výčtu
  - ()
  - \
  - **Příklady:**
- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>a+</li> <li>a*</li> <li>o?kov</li> <li>tel(efon)?</li> <li>telef(on ax)</li> <li>[0-9]  [1-9][0-9]</li> <li>\d{2}</li> <li>[0-9a-fA-F]  [1-9a-fA-F][0-9a-fA-F]+</li> <li>(19 20)\d{2}</li> <li>\d{2,6}</li> <li>[^ „,]+</li> <li>^p.*</li> <li>\\d+0\$</li> <li>a+b</li> <li>a\\+b</li> <li>h?rozin(k c).*</li> <li>.+nést</li> <li>.+in[kg]</li> <li>[bB]ože</li> <li>les.*</li> </ul> | <ul style="list-style-type: none"> <li>sekvence písmen a (1 a více znaků)</li> <li>sekvence písmen a (0 a více znaků)</li> <li>okov či kov</li> <li>tel či telefon</li> <li>telefon či telefax</li> <li>čísla 0 až 99</li> <li>sekvence dvou číslic desítkové soustavy (00, 01, ..., 98, 99)</li> <li>hexadecimální čísla</li> <li>letopočty 1900–2099</li> <li>sekvence dvou až šesti číslic</li> <li>neprázdná sekvence znaků, mezi nimiž nesmí být mezera ( ), čárka (,) či tečka (.)</li> <li>řetězec, který začíná písmenem P, za nímž následuje libovolný (i nulový) počet libovolných znaků</li> <li>řetězec, který končí znakem 0 (nula), kterému předchází minimálně jedna číslice</li> <li>ab, aab, aaab atd.</li> <li>a+b</li> <li>hrozinka, rozinka + v dalších pádech a v plurálu</li> <li>donést, odnést, přinést, zanést, vynést, povznést, ...</li> <li>mítink, brífing, leasing, sing, ...</li> <li>bože, Bože</li> <li>les, lesk, lesklý, lesní, lest, ...</li> </ul> |
|--|---|

## Automatická morfologická analýza přirozeného jazyka

**Fonetická analýza** jazyka postihuje vztahy mezi zvuky používanými v (mluveném) jazyce, jejich skládání do slabik a slov. Foném – nejmenší jednotka jazyka, které může odlišit význam nadřazených jednotek (*kosit/nosit* – odlišení dvou slov), často odpovídá znaku, vždy ale označuje zvuk.

**Morfologická analýza** jazyka – interní struktura slov, skládání slov z menších jednotek. Morfém – nejmenší jednotka, která může nést význam. Např. slovo *pří-lež-it-ost-n-ými* (*pří* je prefix označující blízkost, *lež* – lexikální kořen slova *ležet*, *it* – adjektivní derivační sufix, *ost* – substantivní derivační sufix, *n* – adjektivní derivační sufix, *ými* – gramatický afix instrumentálu plurálu).

Morfologická analýza klasifikuje (značuje, *tag*) slovní tvary jednotlivých kategorií (**Part of Speech/PoS tags**). **Kategorie pro účely** analýzy můžeme dělit na dvě skupiny:

- **lexikální kategorie** – pojmenovávají věci, akce, myšlenky... → substantiva, verba, adjektiva, adverbia, ...
- **gramatické kategorie** – vyjadřují vztahy mezi ostatními větnými členy – předložky, spojky, částice, anglické členy

Jazyky můžeme rozdělit na:

- **jazyky s jednoduchou morfologií** – např. angličtina, několik desítek kategorií
- **jazyky s bohatou morfologií** – hierarchický systém, kde vedle základních slovních druhů určujeme nejružnější subklasifikace (pád, číslo, rod, osoba, druhy příslovčí, ...) - celkově tisíce značek

Morfologická analýza provádí

- **rozpoznávání slovních tvarů** - nástroj se nazývá **morfologický analyzátor** (Part-of-Speech/PoS tagger)
- **lemmatizaci** – přiřazuje k rozpoznaným slovním tvarům základní tvar (lemma)
- charakterizuje morfo-syntaktické vlastnosti nalezených slovních tvarů
- kvalita morfologické analýzy ovlivňuje všechny následující analytické roviny

Úkol morfologické analýzy zahrnuje 3 podúkoly:

- vypsát všechny možné analýzy – klasický **morfologický analyzátor**
- vybrat jednu nejpravděpodobnější analýzu – značkovač (**tagger**)
- analýzy pro neznámé slovo podle koncovky – **guesser**

#### BRILLŮV ZNAČKOVAČ

- učí se podle trénovacích dat
- 1. přiřadí nejčastější značku
- 2. zkontroluj, kde jsou chyby (podle trénovacích dat)
- 3. ohodnot' pravidla pro opravu chyb → vyber nejlepší → oprav zpětně chybné značky
- 4. opakuj, dokud se daří odvozovat dobrá pravidla
- používá **učení založené na transformacích** (transformation-based learning)
- analogie – malování obrazu: nejprve pozadí a pak přes něj stále drobnější detaily
- značkuje 36 různých PoS značek, úspěšnost přes 90 %

#### ALGORITMICKÝ POPIS ČESKÉ FORMÁLNÍ MORFOLOGIE

V češtině nestačí pravidla podle obecných morfémů – je potřebné mít lexikon, který ke každému kmenu obsahuje jeho přiřazení ke vzoru.

**Morfologické (tvaroslovné) paradigma** – soubor tvarů ohebného slova vyjadřující systém jeho mluvnických kategorií.

**Vzor** – reprezentace tvaroslovného paradigmatu paradigmatickým určení konkrétního slova.

#### Algoritmický popis:

- definice koncovkových množin
- definice vzorů prostřednictvím vzorových slov rozdělených na
  - neměnnou část vzorového slova – **kmenový základ**
  - proměnlivé části vzorového slova – **intersegmenty**
  - **koncovkové množiny** obsahující utříděné seznamy všech přípustných koncovek vzorového slova spolu s jejich gramatickými významy

**Popis vzoru** – formální pravidlo, které specifikuje přípustné kombinace těchto komponent (segmentů) ohebného slova.

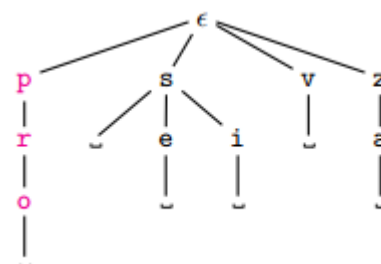
#### SEGMENTACE SLOVA PRO POTŘEBY ALGORITMICKÉHO POPISU

- segmentace od začátku slova
- a) segmenty se snadno formalizovatelným výskytem vázaným gramaticky:
  - – negativní prefix *ne-*
  - – superlativní prefix *nej-*
  - – futurální slovesný prefix *po-*
- b) segmenty s nesnadno formalizovatelným výskytem vázaným sémanticky:
  - – prefixy
  - – první členy kompozit
  - – prefixy *ni-*, *ně-* zájmen neurčitých a záporných
- segmentace od konce slova
- a) rozdělení slovního tvaru na **kmen** a **koncovku**
- b) další segmentace kmene na **kmenový základ** a **intersegment**

#### EFEKTIVNÍ IMPLEMENTACE MORFOLOGICKÉHO LEXIKONU – TRIE

Struktura trie:

- uspořádaný strom nad danou abecedou A
- v každém uzlu je různé písmeno z abecedy A
- klíč je v trie uložen jako cesta od kořene
- výhody:
  - sdílení společných prefixů
  - v každém případě nalezení nejdelšího shodného prefixu



## JINÁ EFEKTIVNÍ IMPLEMENTACE ML – KONEČNÝ AUTOMAT

- použití mírně pozměněných volně dostupných knihoven pro práci s KA od Jana Daciuka – FSA library
- vstupní data se generují ze slovníku ajky převedeného do tvaru “slovo<TAB>lemma<TAB>značka” (cca 33 mil. řádků)
- data se dále upravují pro KA – slovo+zkr.lemma+značky
- v lemmatu – 1. písmeno je počet znaků, které se odtrhnou jako předpona, 2. písmeno je počet znaků, které se trhají od konce a ostatní znaky se přidají
- tím se sníží počet řádků na 6.7 mil. řádků, ze kterých se přímo generuje (a minimalizuje) konečný automat
- výsledný slovník má 4.3MB
- rychlost je cca o 1/4 lepší než u trie, velikost řádově srovnatelná

## ČESKÉ MORFOLOGICKÉ ANALYZÁTORY

### Ajka

- Radek Sedláček, FI MU Brno
- <http://nlp.fi.muni.cz/projekty/ajka/>
- značky jsou řetězce dvojic **atribut-hodnota**
- napsaný v C
- využívá struktury **trie**
- 390 000 základních tvarů, 6 300 000 různých slovních tvarů, 15 000 různých značek, slovník 3.13MB
- rychlost analýzy – cca 18 000 slov/s
- v současnosti nový nástroj majka od Pavla Smerka, na principu konečných automatů, s novým mechanismem vzorů

### Pražský morfologický analyzátor

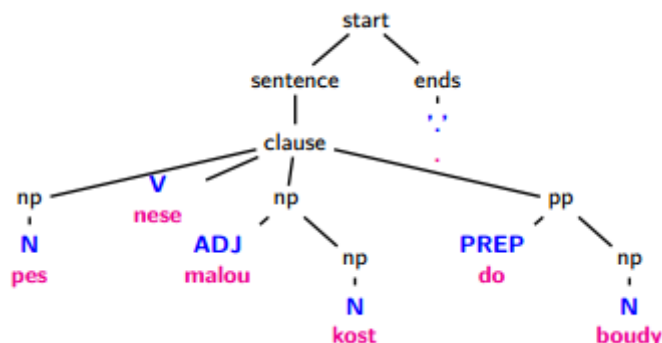
- Barbora Hladká, Jan Hajič a jeho tým, UFAL MFF UK Praha
- <http://ufal.mff.cuni.cz/czech-tagging/>
- používá **poziční značky**
- “free” část napsaná v Perlu, menší slovník (cca 76 000 základních tvarů, 6 000 koncovek)



# Automatická syntaktická analýza přirozeného jazyka

**Syntaktická analýza** – struktura větných frází, popisuje, jak vypadá gramaticky správná věta, většinou pomocí pravidel gramatiky. **Syntaktický analyzátor** – nástroj, který analyzuje vstup na základě gramatiky a na výstup dává různé info, např. derivační stromy.

**Syntax** – charakterizace dobře utvořených kombinací slovních tvarů do věty nebo fráze. Syntaktická analýza se provádí pomocí gramatických pravidel, výstup ze syntaktické analýzy (např. derivační strom) tvoří často vstup pro analýzu sémantickou.



## ZÁKLADNÍ TERMÍNY

- **fráze (phrase)** – jednotka jazyka větší než slovo, ale menší než věta např. jmenná fráze, slovesná fráze, adjektivní fráze nebo příslovečná fráze
- **lexikální symbol, lexikální kategorie (lexical category)** – tzv. pre-terminál speciální neterminál gramatiky, který se přímo přepisuje na terminálový řetězec znaků, tj. pravidla tvaru  $X \rightarrow w$ , označuje všechny slova, která odpovídají určitému lexikálnímu symbolu (všechna podstatná jména, přídavná jména, ...)

$N \rightarrow \text{pes} \mid \text{člověk} \mid \text{dům} \dots$

$V \rightarrow \text{nese} \mid \text{chodit} \mid \text{psal} \dots$

$ADJ \rightarrow \dots$

$PREP \rightarrow \dots$

- **frázová kategorie (phrasal category)** – neterminální symbol gramatiky, který nevyjadřuje lexikální kategorii
- **větný člen (constituent)** – lexikální nebo frázová kategorie
- **větná struktura (sentence structure)** – strukturovaný popis větných členů
  - povrchová struktura (surface structure) - derivační/složkový strom jako výsledek bezkontextové (CF) analýzy
  - závislostní struktura (dependency structure) – zobrazuje závislosti mezi větnými členy
  - hloubková struktura (deep structure) – sémantická interpretace fráze. Popisuje role větných členů (agens, patiens, donor, cause, ...)

## SLOŽKOVÝ A ZÁVISLOSTNÍ PŘÍSTUP

- dva základní způsoby zadávání gramatik

### Složkový přístup

- skupiny slov tvoří větné jednotky, které jsou označovány jako fráze, a jako větné členy (složky, constituents) formují větu
- např. podstatné jméno – součást jmenné fráze (noun phrase – NP), jmenná fráze spolu s předložkou – tvoří předložkovou frázi (prepositional phrase – PP)
- syntaktická struktura věty je zachycována jako složkový strom

### Závislostní přístup

- jeden člen vazby je označován jako řídící, druhý jako závislý
- např. přídavné jméno závisí na řídícím podstatném jménu
- syntaktická struktura věty je zachycována pomocí závislostního stromu:
  - uzly odpovídají elementárním jednotkám vstupu (často slovům)
  - hrany označují vztahy závislosti mezi elementárními jednotkami
- závislost není relací mezi jednotlivými slovy, ale obecně relací mezi jedním slovem a frází řízenou druhým slovem. např. vazba mezi konkrétním slovesem a podmětem nebo vazba mezi slovesem a předmětem věty

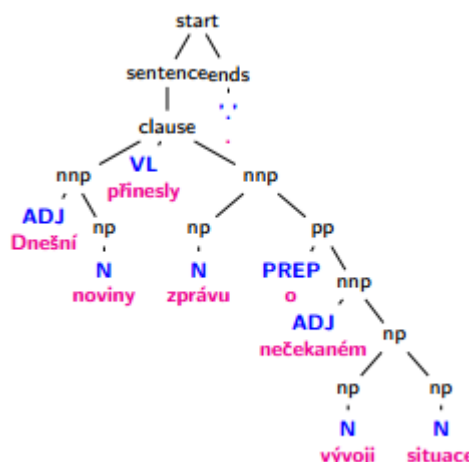
- technicky vzato, závislostní relace je vztahem mezi uzly a podstromy (uzlem a všemi uzly, které na tomto uzlu závisí)

Jen zřídka se používá čistě složkový či striktně závislostní přístup. Ve složkovém jsou závislosti zpravidla vyjádřeny přidáním označení, která složka je řídící pro danou frázi. Závislostní strom bývá doplněn o informaci určující lineární precedenci. Je možné pak mezi těmito přístupy výsledek převádět.

## UZLY SYNTAKTICKÉHO STROMU

Označení uzlu (název neterminálu) podle zvoleného přístupu reprezentuje:

- **gramatická role (gramatická funkce)**
  - charakterizují vztahy mezi větnými složkami na povrchové úrovni
  - určujeme, zda daný větný člen je NP v roli podmětu, NP v roli předmětu, ADVP určující lokaci atd.
  - v češtině (a jazycích se systémem gramatických pádů) pomáhá k určení gramatické role právě informace o pádu
  - ovšem přiřazení gramatických rolí ke gramatickým pádům a naopak není zdaleka jednoznačné.
- **tematická role** (též hlubkový/sémantický pád)
  - na rozdíl od gramatické role se jedná o sémantickou kategorii
  - určujeme např.:
    - Agens – kdo je životným původcem nějaké cílevědomé činnosti
    - Patiens – co hraje roli entity, na kterou se působí
    - Donor – osoba, která dává
    - Cause – entita, která způsobuje, že je něco děláno



## PŘÍZNAKY A PŘÍZNAKOVÉ STRUKTURY

Informace v uzlu syntaktického stromu:

- příznaky/rysy (features) – zaznamenávají syntaktické nebo sémantické informace o slovu nebo frázi.
- např. test na shodu:

Malý Petr přišel domů.

podmět (Petr) je ve shodě s přísudkem (přišel) v čísle a rodě, přídavné jméno (malý) a podstatné jméno (Petr) se shodují v pádě, čísle a rodě

$S(n, g) \rightarrow NP(\_, n, g) VP(n, g)$

$NP(c, n, g) \rightarrow ADJ(c, n, g) N(c, n, g)$

- gramatické znaky (slovní druh, gramatický pád, rod, číslo, osoba, ...) je výhodné začlenit do gramatiky ve formě dvojic atribut–hodnota
- potom je možné **zobecňovat**, např. vyjádřit shodu v pádě, čísle a rodě výhradně pomocí atributů
- aplikace – v mnoha gramatických formalismech
- jazykové objekty jsou zde modelovány jako **příznakové struktury** (feature structures), tedy právě matice dvojic atribut–hodnota.
- u složitějších struktur – nestačí pak běžné porovnání instanciace jde oběma směry → použije se **unifikace**

## MOŽNOSTI ZADÁVÁNÍ GRAMATIK

- nejčastější formát specifikace gramatik – **produkční pravidla** gramatika se skládá z pravidel generujících správně utvořené řetězce
- **cíl analyzátoru** – najít odvození vstupního řetězce ze zadaného neterminálu (označovaného obvykle velkým písmenem S
- z anglického sentence – věta) na základě daných pravidel pokud je tohoto cíle dosaženo, vstup je akceptován a je mu přiřazena odpovídající struktura
- v minulosti rovněž populární – **přechodové sítě** (transition networks) přechody sítě = lingvistické jednotky, uzly sítě = stavy analyzátoru v procesu analýzy vstupu. Přechody jsou označeny symboly definujícími, za jakých podmínek se analyzátor může přesunout z jednoho stavu do stavu druhého.
- **rozsířené přechodové sítě** (ATN – Augmented TN) jsou doplněny o podmínky a procedury – ekvivalentní deklarativním gramatikám

## CHOMSKÉHO TEORIE SYNTAXE

- 50. léta 20. stol – Noam Chomsky vytvořil formální teorii syntaxe
- jedna ze základních tezí – autonomie syntaxe  $\Leftarrow$  k ověření syntaktické správnosti věty nepotřebujeme znát její význam

Colorless green ideas sleep furiously.

vs.

Furiously sleep ideas green colorless.

- syntaktické principy mají univerzální platnost pro různé přirozené jazyky
- znalost jazyka = gramatika
- Noam Chomsky, Aspects of the Theory of Syntax, 1965 – standardní teorie syntaxe – transformační generativní gramatika (TGG)
- snaží se řešit i zachycení sémantických vztahů v hloubkové struktuře
- postupně se vyvinula:
  - v rozšířenou standardní teorii (1968)
  - později tzv. Government & Binding Theory (teorie nadřazení a vázání, 1981), která zakládá na pojmu univerzální gramatiky
  - 90. léta – teorie minimalismu (snaha po úspornosti popisného aparátu)

Chomského předpoklady o rozumu:

- rozum má vrozenou strukturu
- rozum je modulární
- rozum obsahuje speciální modul pro jazyk, porozumění jazyku je oddělitelné od jiných aktivit
- syntaxe je formální, nezávislá na významu a komunikačních funkcích
- znalost jazyka je modulární, obsahuje moduly pro jednotlivé fáze analýzy jazyka

### Základní části standardní teorie:

- bazová komponenta
  - bezkontextová pravidla a schémata pravidel generují základní strukturu větných členů
  - lexikon popisuje lexikální kategorie a syntaktické rysy lexikálních položek

#### pravidla:

$S \rightarrow NP VP$   
 $NP \rightarrow (D) A^* N PP^*$   
 $VP \rightarrow V (NP) (PP)$   
 $PP \rightarrow P NP$

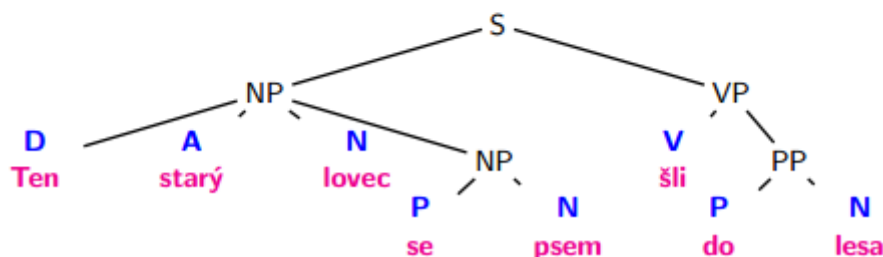
#### lexikon:

D: ten, ta  
A: velký, hnědý, starý  
N: pták, pes, lovec, já, lesa  
V: loví, jí, šli  
P: se, do

#### věta:

Ten starý lovec se psem šli do lesa.

#### syntaktický strom:



- transformační pravidla – vložení, smazání, přesun, změna-rysu, kopie-rysu; transformace převádí hloubkové struktury na struktury povrchové
- transformace:
  - obligatorní – např. přesun slovesné koncovky za sloveso
  - fakultativní – např. pasivizace, tvorba otázek, negace (změna významu)
- pravidla bazové komponenty – popisují strom hloubkové struktury v obvyklém pořadí
- transformace umožňují jeho změny na různé povrchové varianty (trpný rod, otázka, ...)
- stopa (trace) – ukazuje, kde byl prvek před přemístěním

## VÝCHODISKA SYNTAKTICKÉ ANALÝZY

### Návrh podkladů a datových struktur

- syntaktický (odvozovací, derivační) frázový strom – kompletní hierarchický popis struktury věty
- úkol syntaktické analýzy = pro danou gramatiku a daný vstup (větu) dát všechny odvozovací stromy
- existují techniky pro kompaktní uložení lesa takových stromů (chart parsing)
- jelikož se zabýváme výhradně syntaktickou strukturou a nevylučujeme a priori derivační stromy s absurdní interpretací, má většina vět mnoho různých syntaktických stromů

Automatická analýza syntaxe musí vždy projít třemi fázemi:

1. musí být zvolena notace pro zápis gramatiky – gramatický formalismus
2. musí být ve zvoleném formalismu napsána gramatika pro každý jazyk, který bude zpracováván
3. musí být vybrán nebo navržen algoritmus, který určí, zda daný vstup odpovídá gramatice, a pokud ano, jaký popis mu odpovídá

## ALGORITMY SYNTAKTICKÉ ANALÝZY

Základní postupy pro syntaktickou analýzu obecných bezkontextových gramatik

- obecná CFG – rozsáhlá, (silně) víceznačná, s  $\epsilon$ -pravidly
- všechny uvedené algoritmy pracují s polynomiální časovou a prostorovou složitostí
- algoritmus CKY – Cocke, Kasami, Younger;
- tabulková (chart) analýza (neplést s LR tabulkou):
  - shora dolů (top-down);
  - zdola nahoru (bottom-up);
  - analýza řízená hlavou pravidla (head-driven);
- Tomitův zobecněný algoritmus LR

Vstupem syntaktické analýzy je řetězec lexikálních kategorií (pretermínálních symbolů, např. ADJ CONJ ADJ N V PREP N.) a bezkontextová gramatika. Výstupem je efektivní reprezentace derivačních stromů.

### ALGORITMUS CKY

- Gramatika musí být v Chomského normální formě.  
CNF (každá CFG jde do ní převést):  $A \rightarrow BC$   
 $D \rightarrow 'd'$
- Pro daný vstup délky  $n$  derivujeme podřetězce symbolů délky  $q$  na pozici  $p$ , značíme  $w_{p,q}$ ,  $1 \leq p, q \leq n$ .
- Derivace řetězců délky 1,  $A \Rightarrow w_{p,1}$ , je prováděno prohledáváním terminálních pravidel
- Derivace delších řetězců  $A \Rightarrow^* w_{p,q}$ ,  $q \geq 2$  vyžaduje aby platilo  $A \Rightarrow BC \Rightarrow^* w_{p,q}$ . Tedy z  $B$  derivujeme řetězec délky  $k$ ,  $1 \leq k \leq q$ , a z  $C$  derivujeme zbytek, řetězec délky  $q - k$ . Tzn.  $B \Rightarrow^* w_{p,k}$  a  $C \Rightarrow^* w_{p+k,q-k}$ . Kratší řetězce máme tedy vždy “předpočítané.”
- složitost CKY je  $O(n^3)$
- příklad:                                      vstupní řetězec                                      vstupní gramatika

a b a a b a

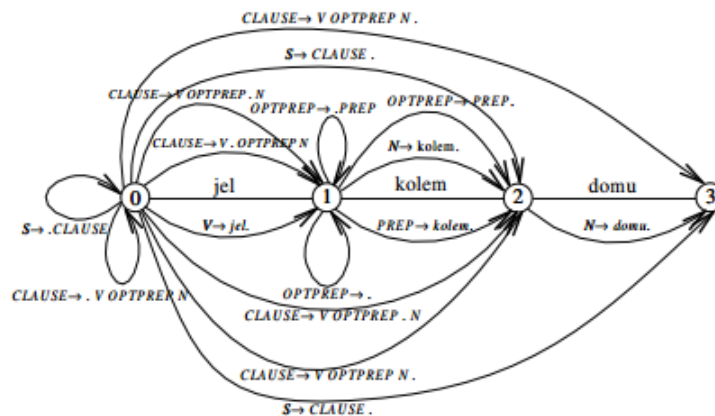
$S \rightarrow AA|BB|AX|BY|a|b$   
 $X \rightarrow SA$   
 $Y \rightarrow SB$   
 $A \rightarrow a$   
 $B \rightarrow b$

$p$  – pozice,  $q$  – délka

$q \backslash p$	1	2	3	4	5	6
1	$S, A$	$S, B$	$S, A$	$S, A$	$S, B$	$S, A$
2	$Y$	$X$	$S, X$	$Y$	$X$	
3	$S$	$\emptyset$	$Y$	$S$		
4	$X$	$S$	$\emptyset$			
5	$\emptyset$	$X$				
6	$S$					

## TABULKOVÉ (CHART) ANALYZÁTORY

- Rozlišujeme tři základní typy tabulkových analyzátorů:
  - shora dolů;
  - zdola nahoru;
  - analýza řízená hlavou pravidla.
- Neklade se žádné omezení na gramatiku.
- Analyzátor typu “chart” v sobě většinou obsahují dvě datové struktury **chart** a **agendu**. Chart a agenda obsahují tzv. hrany
  - Hrana je trojice  $[A \rightarrow \alpha \bullet \beta, i, j]$ , kde:
    - $i, j$  jsou celá čísla,  $0 \leq i \leq j \leq n$  pro  $n$  slov ve vstupní větě
    - a  $A \rightarrow \alpha \beta$  je pravidlem vstupní gramatiky.
  - chart po analýze shora dolů



## TOMITŮV ZOBECNĚNÝ ANALYZÁTOR LR

- generalized LR parser (GLR)
- Masaru Tomita: Efficient parsing for natural language, 1986
- standardní LR tabulka, která může obsahovat konflikty;
- zásobník je reprezentován acyklickým orientovaným grafem (DAG);
- derivační stromy jsou uloženy ve sbaleném “lese” stromů.
- v podstatě stejný, jako algoritmus LR;
- udržujeme si seznam aktivních uzlů zásobníku (grafu);
- akce redukce provádíme vždy před akcemi čtení;
- akci čtení provádíme pro všechny aktivní uzly najednou;
- kde je to možné, tam uzly slučujeme.

## Formální gramatiky pro přirozené jazyky

Jazyk lze chápat jako množinu, v níž je podle Chomského možné přesně specifikovat členství konečným souborem pravidel. Množina jazykových výrazů přirozeného jazyka ale není konečná, nelze podat jejich plný výčet a žádný přirozený jazyk není konečný → potřeba formálních systémů, které umožňují definovat členství v nekonečné množině jazykových výrazů a každému členu této množiny přiřadit jeho strukturní popis pomocí konečného souboru pravidel.

**Formální jazyk** v matematice, logice a informatice označuje množinu konečných slov (tj. slov konečné délky) nad určitou abecedou. Místo výrazu "slovo" se někdy užívá výraz "řetězec". Definice pojmu formální jazyk se může měnit podle toho, v jakém kontextu a v jakém vědním oboru jej používáme.

Příkladem abecedy může být  $\{a, b\}$ , slovem nad touto abecedou je například *ababba*. Příkladem jazyka mohou být slova nad touto abecedou, která obsahují stejný počet symbolů *a* a *b*.

Prázdné slovo (tj. slovo, které se skládá z nulového počtu znaků) se značí  $\epsilon$ ,  $\epsilon$  nebo  $\lambda$ . Ačkoli abeceda je konečná množina a každé slovo je konečná množina, jazyk konečný být nemusí, jelikož délka slov nemusí být shora omezena.

**Abeceda** je obvykle značena symbolem  $\Sigma$ . Zápis  $\Sigma^*$  pak označuje jazyk, obsahující všechna slova nad danou abecedou, včetně prázdného slova. Každý jazyk  $L$  nad určitou abecedou  $\Sigma$  je podmnožinou jazyka  $\Sigma^*$ .

**Příklady formálních jazyků:**

- množina všech slov nad abecedou  $a, b$
- množina  $\{a^n\}$ ,  $n$  je přirozené číslo a  $a^n$  znamená, že *a* se vyskytuje  $n$ -krát za sebou.
- konečné jazyky jako například *a, aa, bba*
- množina všech programů v daném programovacím jazyce
- množina všech slov, nad kterými daný Turingův stroj zastaví.

**Formální jazyk může být definován různými způsoby, například:**

- slova generovaná nějakou formální gramatikou (viz Chomského hierarchie);
- slova vyhovující nějakému regulárnímu výrazu;
- slova akceptovaná nějakým automatem, například Turingovým strojem nebo konečným automatem

**Gramatika – formální systém**, který splňuje kromě zmíněného kritéria tato pravidla:

- je vyjádřena deklarativním formalismem obsahujícím informaci o kombinovatelnosti objektů a vlastnostech výsledného objektu
- transparentně spojuje každý přípustný výraz jazyk (řetězec) s jeho implicitním strukturním popisem
- přímo specifikuje pořadí prvků v řetězu

### GRAMATIKA JAKO REPREZENTACE ZNALOSTÍ

Gramatiky jsou založeny dekompozici syntaktických kategorií na složky známé jako rysy, podporují kompozicionální přístup k významu – každý dobře utvořený výraz jazyka má svůj vlastní význam složený z významů podvýrazů, které jej tvoří. Gramatika je prostředkem pro reprezentaci znalostí o jazyce.

Všechny druhy gramatik užívaných v počítačové lingvistice využívají:

- reprezentaci syntaktických kategorií nebo slovních druhů
- datové typy pro slova (slovní formy, tj. slovník)
- datový typy pro syntaktická (morfologická) pravidla
- datové typy pro syntaktické struktury

Gramatiku pak lze chápat jako užití konkrétních datových typů složených z uvedených tří jednotek.

### FORMÁLNÍ GRAMATIKY

= soubor formálních pravidel, která umožňují generovat nebo rozpoznávat české věty a současně jim přiřazovat popisy jejich struktury

= skládá se z množiny pravidel, pomocí kterých může být každé slovo předepsaným způsobem vygenerováno z předem daného počátečního symbolu. Generování probíhá tak, že vezmeme počáteční symbol, na něj aplikujeme kterékoli z pravidel, na získaný řetězec opět aplikujeme kterékoli z pravidel atd., dokud nevygenerujeme požadované slovo. Pokud je pro každé slovo nejvýše jeden postup generování, gramatika je jednoznačná.

Mějme například abecedu obsahující symboly '*a*' a '*b*', počáteční symbol je '*S*' a pravidla jsou definována takto:

1.  $S \rightarrow aSb$
2.  $S \rightarrow ba$

začneme symbolem „*S*“ a vybereme pravidlo, které budeme aplikovat. Pokud vybereme 1, nahradíme '*S*' řetězcem '*aSb*' a obdržíme tak „*aSb*“. Znovuzvolením 1. pravidla nahradíme '*S*' opět řetězcem '*aSb*' a obdržíme „*aaSbb*“. Tento proces

můžeme opakovat, dokud nejsou všechny symboly našeho slova z abecedy (tj. '*a*' a '*b*'). Abychom tedy vygenerovali slovo, musíme zvolit 2. pravidlo a přepsat '*S*' na '*ba*'. Tím obdržíme „*aababb*“ a jsme hotovi. Jazykem gramatiky jsou všechna slova, která dokážeme vygenerovat:  $\{ba, abab, aababb, aaababbb, \dots\}$

Znaky z abecedy (v našem případě '*a*' a '*b*') se nazývají **terminály**, ostatní znaky (*S*) se nazývají **neterminály**.

Větu lze rozdělit na **podmětovou část (NP1)** a **přísudkovou část (VP)**. To lze zapsat jako pravidlo

**S → NP1 VP**

Gramatika představuje formální prostředek, jímž lze vyjádřit koneční i nekonečné jazyky, sama gramatika je nekonečná. Základní pojmy:

- **abeceda** – neprázdná množina prvků/symbolů abecedy (česká abeceda obsahuje 82 symbolů)
- **řetězec** – slovo, posloupnost symbolů abecedy, prázdný řetězec značíme *e*. Je-li *x* řetězec nad abecedou *T* a *a*  $\in T$ , pak *xa* je řetězec nad *T*, *y* je řetězec nad *T* pouze tehdy, lze-li ho získat aplikací předchozích dvou pravidel.
- **zřetězení (konkatenace)** – připojení jednoho řetězce za druhý
- jazykem může být libovolná podmnožina řetězců nad danou abecedou
- **abeceda N** neterminálních symbolů (syntaktické kategorie) a **abeceda T** terminálních symbolů (slova daného jazyka), sjednocení obou množin – **slovník gramatiky**

Gramatika *G* je uspořádaná čtveřice  $g = \{N, T, P, S\}$ , kde

- *N* je konečná množina neterminálních symbolů – syntaktických kategorií
- *T* – je množina terminálních symbolů – konkrétních slovních tvarů (většinou se označuje řeckým písmenem sigma –  $\Sigma$ )
- *P* – konečná podmnožina kartézského součinu  $(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ ,
- $S \in N$  je vyznačení počáteční symbol gramatiky *G*
- prvek  $(\alpha, \beta)$  množiny *P* nazýváme přepisovacím pravidlem, zapisuje se  $\alpha \rightarrow \beta$ .

Konvence:

- jednotlivé terminály značíme – a, b, c, ...
- řetězce terminálů značíme – u, v, w, ...
- jednotlivé neterminály – A, B, C ... X, Y, Z
- řetězce neterminálů a terminálů –  $\alpha, \beta, \gamma, \dots$
- prázdný řetězec značíme symbolem *e* nebo také  $\epsilon$

**Chomského hierarchie** popisuje čtyři důležité množiny gramatik, z nichž každá je podmnožinou následující (podrobněji viz jiná podotázka):

- Regulární gramatiky
- Bezkontextové gramatiky
- Kontextové gramatiky
- Všechny formální gramatiky

Jádrem gramatiky je konečná množina přepisovacích pravidel, která stanovují možné nahrazení řetězce  $\alpha$  řetězcem  $\beta$ . Dojdeme-li v posloupnosti přímých odvození k řetězci obsahujícímu pouze terminální symboly, pak nelze aplikovat žádné přepisovací pravidlo a proces generování končí.



## Jazykový model, n-gramy

---

**Statistický jazykový model** přiřazuje pravděpodobnost posloupnosti slov pomocí rozložení pravděpodobnosti. Přiřazuje každé posloupnosti pravděpodobnost s jakou je daná posloupnost větou určitého jazyka. Tyto pravděpodobnosti se získávají statistickým zpracováním jazykových dat. Může být unigramový (prosté frekvenční distribuce), bigramový, trigramový, obecně n-gramový (aplikuje podmíněné pravděpodobnosti), cache language model a další.

**N-gramový model** se formálně skládá z podmíněných pravděpodobností  $P(W_n | W_1, \dots, W_{n-1})$ , tzn. pravděpodobností, že se vyskytlo slovo  $W_n$  za předpokladu, že před ním se vyskytla slova  $W_1, \dots, W_{n-1}$ . Souhrn takovýchto pravděpodobností pro všechny možné kombinace slov v korpusu (tento souhrn snadno odvodíme z textu korpusu) se nazývá n-gramový jazykový model.  $W_1, \dots, W_{n-1}$  nemusí být pouze slova – můžeme vytvořit n-gramový model znaků, fonémů, pádů, morfologických značek apod. Využití v rozpoznávání řeči, strojovém překladu, word sense disambiguation atd. S rostoucím  $n$  rychle přibývá různých n-gramů – větší kontext → vyšší úspěšnost, ale zvyšuje se výpočetní náročnost práce s modelem a náchylnost k data sparseness (řídkost dat, znamená nedostatek dat, týká se zejména okrajových jevů).

**Cache language model** – cache = LIFO, využití například při rozpoznávání mluvené řeči, předpokládáme, že nejpravděpodobnější je to slovo, které bylo vyřčeno před chvílí, např. vyprávíme o slonech, samo slovo není tak časté, podle n-gramového modelu by dostaly přednost častější n-gramy, pokud ale slovo vyslovíme, opravíme na „slon“, bude mu dále přisuzována vyšší pravděpodobnost.

**N-gramové jazykové modely** – hledáme další slovo (značku) na základě předchozích, pravděpodobnostní rozložení  $P(w_n | w_1, \dots, w_{n-1})$ , z dat odvodíme pravděpodobnostní rozložení všech možných  $w_n$ ; použití – strojový překlad, morfologické značkování, rozpoznávání řeči...; problémy – pro  $N > 4$  většinou výpočetně nezvládnutelné, *data sparseness* – pro slova, která se vyskytují méně často, není dost dat → špatný model

## Využití statistiky ve zpracování přirozeného jazyka

---

**Statistika** – umožňuje uchopit a popsat velké množství dat → vyvozovat další informace

**Statistické zpracování přirozeného jazyka** – k dispozici velké soubory jazykových dat (korpusy +10mld slov), které umožňují aplikovat statistické metody, predikovat chování přirozeného jazyka a modelovat jej

**Využití statistiky:**

a) **Vyhledávání kolokací** – kolokace = statisticky významné spojení dvou slov v korpusu, lze určit síla kolokace libovolných dvou slov

Možné způsoby: 1. uvažovat při vyhledávání kolokací pouze dvojice složené z určitých slovních druhů  
2. Statistické metriky, např. T-test – testování hypotéz – Nulová hypotéza – pokud je vyvrácena (slova se spolu vyskytují častěji než náhodně), jde o kolokaci

b) **N-gramové jazykové modely** (viz výše); nedostatky jazykových modelů: jazykové modely mohou být velké → výpočetně náročné X malé  $n$  zase neposkytuje dostatečný kontext pro kvalitní odhady a *data sparseness* (řídkost dat) – pro slova a n-gramy, které nejsou tak časté → dostaneme nekvalitní odhady pravděpodobností

Ke zpracování přirozeného jazyka existují dva přístupy:

- **pravidlový** – občas kriticky selhává, jestliže jsme na něco nevymysleli pravidlo, je potřeba mnoho pravidel, těžko je možné vymyslet pravidla na všechno – přirozený jazyk je vágní, mnohoznačný a vyvíjí se, při tomto přístupu se používají formální gramatiky
- **statistický** – selhává často, ale jen málo, je potřeba mnoho dat ke statickému zpracování (čím víc dat, tím lepší výsledky)

# Automatická sémantická analýza přirozeného jazyka

**Sémantická analýza** - význam výrazů přirozeného jazyka a jejich kombinací hodně závisí na zvolené sémantické reprezentaci. Logická analýza věty – strukturní část sémantické analýzy. **Pragmatická analýza** – zkoumá vztah mezi výrazy přirozeného jazyka a kontextem, často se do ní řadí znalost komunikační situace, základní ontologie a jazykových metaznalostí.

Popis a definování významu představuje nejobtížnější oblast v rámci zpracování přirozeného jazyka, zároveň je zvládnutí této problematiky důležitým předpokladem k pokroku v ZPJ i AI, konkrétně v rámci reprezentace znalostí a inferencí.

Pokud se pokoušíme definovat význam slov, užíváme opět jiného jazyka – metajazyka jímž může být týž nebo jiný přirozený jazyk, formální jazyk (vhodný matematický nebo logický kalkul nebo jazyk sémantických rysů), ostenzivní definicí (*to je auto, toto jsou klíče*). Na ostenzi je založeno učení člověka, bylo by tedy vhodné ji přenést také do AI.

Problémy v popisu významu činí také závislost vět na kontextu. Do jisté míry lze slova zkoumat nezávisle na kontextu.

## PROBLÉMY PŘI ANALÝZE PŘIROZENÉHO JAZYKA

- **víceznačnost**
  - lexikální (*stát, žena, hnát*)
  - syntaktická (*Jím špagety s masem/se salátem/s použitím vidličky/se sebezapřením/s přítelem.*)
  - sémantická (*Jeřáb je vysoký. Viděli jsme veliké oko.*)
  - referenční (*Oni přišli pozdě. Můžeš mi půjčit knihu. Ředitel vyhodil dělníka, protože byl agresivní.*)
- **anaforické výrazy** – používají zájmena pro odkazování na objekty zmíněné dříve (*Poté, co se Honza s Marií rozhodli vzít, (oni) vyhledali kněze, aby je oddal.*)
- **indexické výrazy** – odkazují se na údaje v jiných částech promluvy a mimo promluvu (*Já jsem tedy. Proč jsi to udělal.*)
- nejasnost
- nekompozicionalita – *aligátorí boty, červená kniha, bílý trpaslík, dřevěný pes, umělá tráva, velká molekula*
- struktura promluvy
- metonymie – používání jména jedné věci pro označení jiné (*Čtu Shakespeara. Chrysler oznámil rekordní zisk. Ten pstruh na másle u stolu 3 chce další pivo.*)
- metafora – použití slov v přeneseném významu na zákl. podobnosti (*Zkoušel jsem ten proces zabít. Bouře se vzteká.*)

## POPIS VÝZNAMŮ SLOV

Lze je roztrždit do významových tříd či sémantických polí, tzn. nejprve si zavést vhodnou **ontologii** – množinu tříd objektů, která představuje klasifikaci objektů universa U. Podle Aristotela jsou hlavní třídy objektů:

- substance – fyzické objekty
- kvantity – čísla
- vlastnosti
- relace – typicky slovesa
- stavy
- události – stávají se, probíhají → nejčastěji slovesa
- akce – to, co dělají činitelé
- procesy
- situace – soubor okolností, abstrakce úseku světa na určitém místě a v určitém čase
- pozice
- čas
- následek
- plány, záměry

Význam slov můžeme popisovat pomocí synonym (SSJČ), pomocí definic (SSČ) nebo pomocí množiny vybraných primitivních výrazů daného jazyka, případně pomocí speciálního metajazyka: sémantických rysů, komponentová analýza (*muž* = HUM, MASK, ADU). Rysy lze kombinovat a jednomu výrazu jich přiřadit víc.

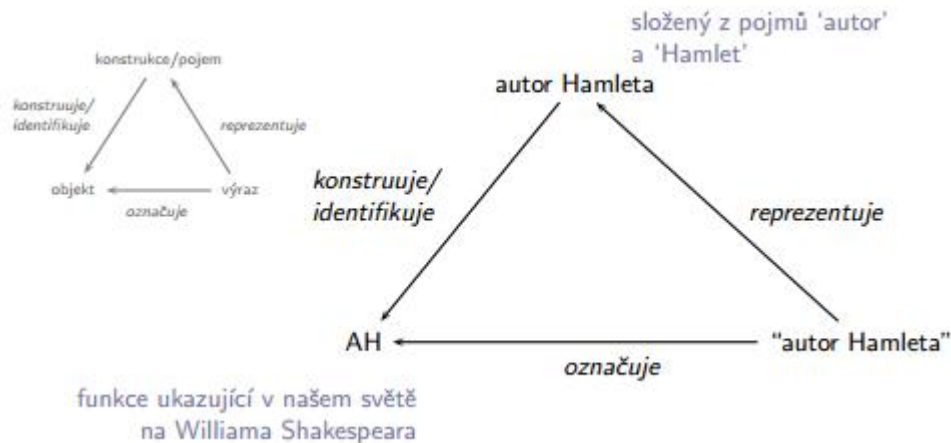
## LOGICKÁ ANALÝZA PŘIROZENÉHO JAZYKA

- analýza významu výrazů (vět) PJ
- přirozený jazyk = nástroj pojmového uchopení reality
- pojem – kritéria/procedury umožňující identifikovat různé konkrétní a abstraktní objekty, např. “planeta” – třída

nebeských těles s určitými charakteristikami – obíhá po oběžné dráze kolem stálice, není zdrojem světla, ...

- pojem  $\neq$  výraz – např. výrazy v různých jazycích často reprezentují stejný pojem (pojem („prvočíslo“)  $\equiv$  pojem („prime number“))
- pojem  $\neq$  představa – představa je subjektivní, pojem je objektivní pojmy mohou identifikovat různé objekty:
  - jedno individuum – individuální pojmy (např. Petr, Pegas, prezident ČR)
  - třídu objektů – vlastnost (např. červený, šelma, hora)
  - n-člennou relaci – vztah (např. otec (někoho), křivdit (někdo někomu))
  - pravdivostní hodnotu – propozice (např. v Brně prší)
  - funkcionální přiřazení – empirické funkce (např. rychlost)
  - číslo – (fyzikální) veličiny (např. rychlost světla)

Vztah pojmu a výrazu – ve zjednodušené podobě pojem odpovídá logické konstrukci



#### OMEZENOST PREDIKÁTOVÉ LOGIKY 1. ŘÁDU

dva omezující rysy:

- nedostatečná expresivita
- extenzionalismus

**Expresivita:** vyjadřovací síla jazyka

„Je-li barva stropu pokoje č. 3 uklidňující, je pokoj č. 3 vhodný pro pacienta X a není vhodný pro pacienta Y.“

**Analýza ve výrokové logice:**  $P \Rightarrow (Q \wedge \neg R)$

- P „Barva stropu pokoje č. 3 je uklidňující.“
- Q „Pokoj č. 3 je vhodný pro pacienta X.“
- R „Pokoj č. 3 je vhodný pro pacienta Y.“

**Analýza v PL1:**  $U(B) \Rightarrow (V(P, X) \wedge \neg V(P, Y))$

- U třída uklidňujících objektů
- B individuum 'barva stropu pokoje č. 3'
- V relace mezi individuy 'být vhodný pro'
- P individuum 'pokoj č. 3'
- X, Y individua 'pacient X' a 'pacient Y'

*Cervená barva je krásnější než hnědá barva. Kostka ~ je červená.*

Analýza v PL1:  $Kr(\check{C}_1, H)$   $\check{C}_2(Ko)$

$\check{C}_1$  individuum 'červená barva'

$\check{C}_2$  vlastnost individuí 'být červený' (třída červených objektů)

nelze vyjádřit  $\check{C}_1 \equiv \check{C}_2$

## EXTENZIONALISMUS PL1

Varšava – hlavní město Polska

- Varšava – **jméno individua**, jasně identifikovatelné a odlišitelné
- hlavní město Polska – **individuová role**, momentálně identifikuje Varšavu, ale dříve to byl i Krakov, 'hlavní město Polska':
  - závisí na světě a čase
  - pochopení významu, ale není vázané na znalost obsahu – tj. **význam na světě a čase nezávisí**

číslo  $X$  je větší než číslo  $Y$  budova  $X$  je větší než budova  $Y$

- matematické větší než – relace dvojic čísel, pevně daná
- empirické větší než – vztah dvou individuí, který se může měnit v čase (otec a syn)

ano V Brně prší

- ano – pravdivostní hodnota true
  - V Brně prší – propozice – označuje pravdivostní hodnotu, která se mění (alespoň) v čase
- i když hodnota někdy závisí na světě a čase, samotný význam na nich nezávisí

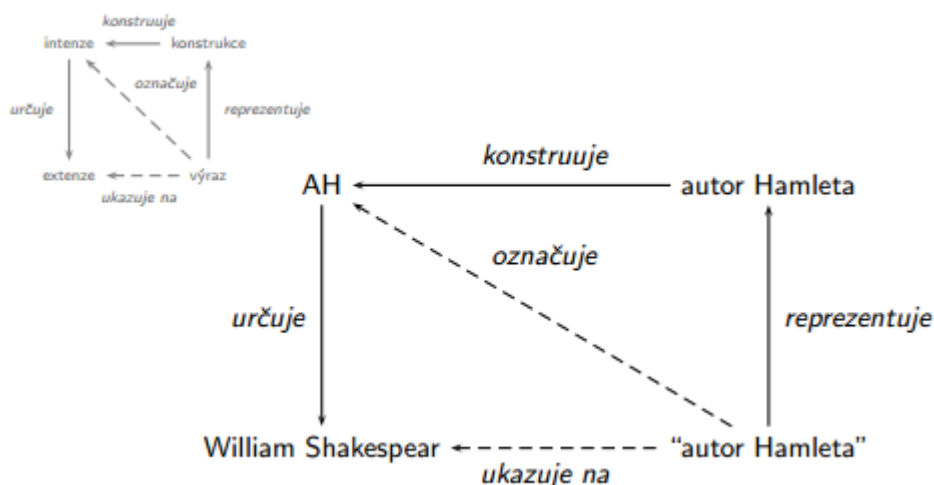
Definujeme:

- intenze – objekty typu funkcí, jejichž hodnoty závisí na světě a čase
- extenze – ostatní objekty (na světě a čase nezávislé)

časté extenze a intenze:

Extenze	Intenze
individua	individuové role
třídy	vlastnosti
relace	vztahy
pravdivostní hodnoty	propozice
funkce	empirické funkce
čísla	veličiny

Rozšířený vztah výrazu a významu u intenzí



## Sémantické reprezentace vět PJ

### SÉMANTICKÉ REPREZENTACE VÝRAZŮ PŘIROZENÉHO JAZYKA

Sémantické reprezentace by měly:

- umožňovat jednoznačné zachycení významů výrazů přirozeného jazyka (PJ)
- umožňovat postižení synonymie výrazů jazyka, tj. situace, kdy různým větám odpovídá jeden význam (např. situaci, kdy různé otázky mají jednu a tutéž odpověď)
- umožňovat přirozené postižení homonymie jazykových výrazů, tj. situaci, kdy jedné větě odpovídá více významů  
→ více sémantických reprezentací

### FORMÁLNÍ APARÁT PRO SR – CHARAKTERISTIKA TIL

**Transparentní intenzionální logika** – logický systém založený na modifikaci typovaného lambda kalkulu = logický aparát umožňující manipulaci s funkcemi. Je založen na principu teorie typů. Je vhodný pro sémantickou analýzu výrazů PJ. Byl navržen speciálně pro zachycení významu výrazů PJ.

Pro TIL je sémantika výrazu dána tím, že způsob, jakým je tento výraz strukturován, zobrazuje strukturu konstrukce, jejímiž složkami nejsou složky jazykového výrazu, nýbrž objekty těmito složkami označené. (Např. ve formálním pojetí je sémantikou výrazu  $3+5$  číslo 8, v pojetí TIL je to určitý způsob, jakým uvedené složky spolupracují na vytvoření objektu.

Neobsahuje žádná „logická slova“ jako jsou výrokové spojky nebo kvantifikátory. TIL aplikována na analýzu přirozeného jazyka se stává sémantikou založenou na pojmu možných světů. Univerzum je chápáno jako množina společná všem možným světům. Vztah označování a vztah vyjadřování je nahrazen jiným schématem.

Základní rysy TIL systému jsou:

- schopnost systematicky překračovat omezení platná v predikátové logice 1. řádu
- intenzionalismus a z něho vyplývající schopnost přesného definování intenzí<sup>1</sup> a zacházení s nimi
- větší expresivní síla
- má rozvětvenou typovou hierarchii
- temporální
- transparentní – nositel významu (konstrukce) není prvek formálního aparátu, tento aparát pouze studuje konstrukce, zachycení intenzionality je přesně popsáno z matematického hlediska

#### Typy v TILu

- základní typy – typová báze =  $\{o, \iota, \tau, \omega\}$ , umožňují přiřadit typ objektům z intenzionální báze jazyka – třída základních vlastností (barvy, rozměry, postoje, ...) popisujících stav světa
  - $o$  (omikron) – pravdivostní hodnoty Pravda (true, T) a Nepravda (false, F), odpovídají běžným logikám
  - $\iota$  (jota) – třída individuí, individua jako numerická identifikace nestrukturované entity
  - $\tau$  (tau) – třída časových okamžiků (časová kontinua), zachycení závislosti na čase, třída reálných čísel
  - $\omega$  (omega) – třída možných světů, zachycení empirické závislosti na stavu světa
- funkcionální typy – funkce nad typovou bází
- typy vyšších řádů

#### Možné světy

- termín možný svět – Gottfried Wilhelm von Leibniz (1646 – 1716, filozof a matematik)
- požadavky na definici “možného světa”:
  - soubor myslitelných faktů
  - je konzistentní a maximální ze všech takových souborů
  - je objektivní (nezávislý na individuálním názoru)
- mezi možnými světy existuje právě jeden aktuální svět – jeho znalost  $\equiv$  vševědoudnost
- možný svět v TILu = rozhodovací systém, pro každý prvek intenzionální báze obsahuje konzistentní přiřazení hodnot, např. realita s 2 objekty a 2 vlastnostmi (9 možných světů):

---

<sup>1</sup>intenze neboli smysl pojmu je jeho obsah, to, co se pojmem míní, jeho hlavní či ústřední význam

	být hubený	být tlustý	
	$\{Laurel, Hardy\}$	$\{Laurel\}$	$\{Hardy\}$
$\{Laurel, Hardy\}$	×	×	×
$\{Laurel\}$	×	×	$w_2$
$\{Hardy\}$	×	$w_4$	×
$\emptyset$	$w_6$	$w_7$	$w_8$

být hubený	... objekt typu $(ol)_{\tau\omega}$ , funkce z možných světů a času do tříd individuí
$w$	... proměnná typu $\omega$ , možný svět
$t$	... proměnná typu $\tau$ , časový okamžik
$[být\ hubený\ w\ t]$	... konstruuje $(ol)$ -objekt, třídu individuí, kteří mají ve světě $w$ a čase $t$ vlastnost být hubený (značíme $být\ hubený_{wt}$ )

## SÉMANTICKÁ ANALÝZA VÝRAZŮ PJ

Cílem je ukázat, jak význam složeného výrazu může být odvozen z významů jeho složek. Znamená to nalézt konstrukci, která je výrazem vyjadřována.

Vlastní sémantická analýza může začínat testováním uzlů syntaktického stromu a rysů v seznamech připojených k uzlům. Dalším krokem je analýza slovesné skupiny ve větě – pro hlavní sloveso se najde jeho typová charakteristika. Dále se hledají adverbia, pokud jsou nalezena připojí se pod slovesnou skupinu a vytvoří se celkový typ slovesné skupiny. Nejobtížnější fází analýzy je analýza jmenných skupin. Ve slovníku se vyhledají typy složek tvořících jmennou skupinu. Od uzlu NP se poté sestavuje výsledný typ celé jmenné skupiny, který byl předikován typovou charakteristikou slovesa získanou v předchozím průběhu analýzy. Následně je vytvořen sémantický strom analyzované věty.

(Velmi zjednodušený postup z Pala: *Počítačové zpracování přirozeného jazyka*. 2000. s. 100)

## Komunikační agenty (chatbots)

- počítačové programy simulující psanou konverzaci s člověkem v přirozeném jazyce
- funkce: ne porozumět textu, ale snaha o vytvoření dojmu, že agent tématu konverzace skutečně rozumí
- snaha o splnění Turingova testu (dvě komory, v jedné je PC, v druhém člověk; soudce má na základě počítačového dialogu rozhodnout, kdo je kdo)
- Loebnerova cena: každoroční soutěž programů umělé inteligence (od roku 1991), založena na Turingově testu
- pro zahrnutí volné chvíle, ale i jako virtuální asistent, průvodce internetovým nakupováním atp.

## ELIZA

- 1964-1966 vyvíjena profesorem Josephem Weizenbaumem působícím na Massachusettském institutu technologie
- parodie psychoterapeuta rogeriánské školy - přeměna pacientových výpovědí do podoby otázek, jež jsou pacientům pokládány
- využívala známé psychologické skutečnosti, že většina lidí hovoří nejraději o sobě
- významné procento testerů považovalo program za člověka -> Eliza se dá považovat za první počítačový program, který prošel Turingovým testem, i když jen částečně
- <http://epanel.cz/eliza/eliza.php>

### FUNGOVÁNÍ ELIZY

- vstupní text je analyzován a vyhledají se v něm klíčová slova
- když je takové slovo nalezeno, je věta převedena podle transformačních pravidel přiřazených k onomu klíčovému slovu a je vygenerován výstupní text
- výstupní text bývá často kombinován s uživatelskou předchozí odpovědí
- určuje nejmenší možný kontext, který je ke klíčovému slovu potřebný

### PROBLÉMY ELIZY

- nalezení nejdůležitějšího klíčového slova
- výběr odpovídajícího transformačního pravidla a transformace samotná
- ustanovení mechanismu, který umožní Elize odpovědět srozumitelně, když se v textu nevyskytuje klíčové slovo

## PARRY

- 1972 vytvořen Kennethem Colbym
- simuluje paranoidního schizofrenika
- vyhledá klíčové slovo ve vstupu a přiřadí mu nějaký vnitřní stav, dle vývoje konverzace např. hněv, strach, nedůvěra
- pokročilejší než Eliza

Oba roboti spolu dokonce vedli několikrát rozhovor

## DALŠÍ CHATBOTI

- **Alice:** třikrát získala Loebnerovu cenu (2000, 2001, 2004). Vytvořena v r. 1995, představuje ženu, komunikuje v angličtině
- **Elbot:** angličtina, němčina, od r. 2000; mnozí jej považují za nejúspěšnějšího chatbota vůbec; úspěch tkví v používání žertů, v nichž naznačuje, že je stroj -> mate
- **Cleverbot:** učí se z předešlých konverzací, které si ukládá do databází, v nich pak vyhledává odpověď na současný vstup uživatele (v databázi vyhledává, jak uživatelé reagovali na Cleverbotův dotaz, a jejich reakci použije jako odpověď na aktuální vstup); vlastně zopakuje to, co na současný vstup odpověděli lidé v dřívějších konverzacích – odpověď tedy nevytváří program, ale lidé.

## PRINCIP CHATBOTŮ

- **Natural language processing** (zpracování přirozeného jazyka): rozkládání vstupní repliky na jednotlivé rozpoznatelné části struktury věty
- **Case-based reasoning:** vyhledávání klíčových frází a slov, které spouštějí kýženou, tedy co nejvíce lidské reakci podobnou odpověď
- **Jazyk AIML (Artificial Intelligence Markup Language)**
- **Vyhledávání podle klíčových slov**
  - Rozpoznává je: používá keyword ranking: cesta, s jejíž pomocí chatbot vybere nejlepší klíčové slovo ve své databázi
  - Nerozpoznává je: řekne, že nerozumí; změní téma, položí další otázku, o které něco ví; metoda vytahování starých témat
  - Mohou ztroskotat na překlepech a chybách -> používají **levensteinovy (editační) vzdálenosti**; metoda, jež dokáže stanovit slovo, které je nejvíce podobné slovu chybnému – určuje podobnost mezi dvěma řetězci znaků
- **Další zpracování uživatelského vstupu:** nezpracovává uživatelský vstup, pouze zjišťuje další informace („řekni mi víc“), používání krátkodobé paměti, citace člověka přeformulují do zjišťovacích otázek; využívání encyklopedických databází (např. na otázku, kdo je Einstein si najdou odpověď); schopnost učit se během konverzace



## Reprezentace znalostí (ontologie, wordnet, rámce)

- hledá způsob vyjádření znalostí počítačově zpracovatelnou formou (za účelem odvozování)
- vyvozování znalostí (reasoning) – zpracovává znalosti uložené v bázi znalostí (knowledge base, KB) a provádí odvození (inference) nových závěrů
  - odpovědi na dotazy
  - zjištění faktů, které vyplývají z faktů a pravidel v KB
  - odvození akcí, které vyplývají z dodaných znalostí
- možnosti: ontologie, wordnet, rámce

### ONTOLOGIE

- explicitní a formalizovaný popis určité problematiky
- je slovníkem, který slouží k uchovávání a předávání znalosti týkající se určité problematiky
- obsahuje **glosář** (definici pojmů) a **tezaurus** (definici vztahů mezi jednotlivými pojmy)
- datový model reprezentující určitou znalost nebo její část

#### TYPY PRVKŮ

- **jedinci** – živý i neživý objekt, abstraktní pojem atp.
- **třídy** – množina jedinců určitého typu nebo další podtřídy
- **atributy** – popisují určitou vlastnost, charakteristiku či parametr jedince, obsahují název a hodnotu a jsou určeny pro uložení určité informace vztahující se k danému jedinci
- **vazby** – jednosměrné nebo obousměrné propojení dvou jedinců

### WORDNET A SÉMANTICKÉ SÍŤ

Standardní způsob organizace lexikálního materiálu ve slovnících je abecední řazení (lexikografické uspořádání). Hledání v takových slovnících je pomalé. Slovníky obsahují prototypické informace založené na genu proximum (nadřazený pojem), neodkazují však k výrazům stejného typu, k hyponymům nebo sourozencům (coordinates).

V poslední době se značná pozornost věnuje lexikální sémantice s cílem vytvořit lexikální zdroje, které by popisovaly jak významy lexikálních jednotek, tak jejich vztahy formálně (algoritmicky) a umožňovaly tak systematické využití v NLP. Dalším směrem je budování počítačových lexikálních databází a elektronických verzí tezaurů, vznik sémantických sítí. Reprezentují faktové znalosti (pojmy + vztahy), znalosti jsou uloženy ve formě grafu. K nejdůležitějším vztahům patří **podtřída** – vztah mezi třídami a **instance** – vztah mezi konkrétním objektem a jeho rodičovskou třídou.

#### WORDNET

Slouží k reprezentaci faktových znalostí – pojmy + vztahy. Autorem je skupina kolem G. A. Millera z Princetonu, vznikl kolem r. 1960 pro reprezentaci významu anglických slov. Znalosti jsou uloženy ve formě grafu. V Evropě vznikl EuroWordNet. Miller usiluje o poznání toho, jak je organizována naše lexikálních paměť a na jakých principech jsou budovány naše mentální slovníky.

WordNet je založen na **psycholingvistických principech**. Ve verzi 3.0 obsahuje databáze 155 287 slov uspořádaných do 117 659 synsetů, čímž je pokryto 206 941 slovních významů (dvojic slovo-smysl). Slovník člení do pěti kategorií: **substantiva, verba, adjektiva, adverbia a funkční slova** (sysnsémantika). Slovní druhy jsou rozlišovány na základě jejich sémantické povahy – substantiva jako tématické hierarchi, sloves na zákl. vztahů vyplývání, adj. a dv. jako n-dimenzionální hyperprostory.

Pokouší se lexikální informace organizovat v termínech slovních významů a nikoli slovních tvarů. Přiřazení forem a významů je víceznačné – některým formám odpovídá více různých významů a zároveň některé významy mohou být vyjádřeny různými formami.

Lexikální paměť lze chápat jako organizovanou stromově, kde zákl. vztahem je **tranzitivní a antisymetrický významový vztah ISA** (is a kind of = je druhu), tzn. vztah hypero/hyponymie vedoucí od specifickému ke generickému = generalizace a specializace. WordNet je lexikální databází s hierarchickou strukturou umožňující prohledávání shora dolů a zdola nahoru stejnou rychlostí.

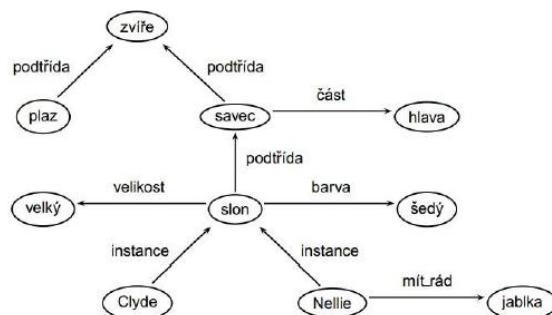
K editaci wordnetů byl na FI MU vyvinut nástroj DEBVisDic. Jako diplomová práce vznik VisualBrowser – nástroj na vizualizaci (sémantických) sítí.

## SÉMANTICKÉ VZTAHY VE WORDNETU

- **hyponymie/hyperonymie** – vztah podřazenosti/nadřazenosti (ISA-vztah), tranzitivní a antisymetrický, generuje hierarchickou (stromovou) reprezentaci pro substantiva
  - uskupují substantiva tak, že tvoří lexikální dědičný systém
  - popis významu substantivních synsetů je obvykle založen na nadřazeném výraz (termu) doplněném o rozlišující příznaky (differentia specifica)
  - synsety (synonimické řady) jsou propojeny ohodnocenými ukazateli (pointry)
  - každé slovo dědí všechny rozlišující příznaky všech svých nadřazených výrazů, např. činnosti, zvíře, výrobek, událost, pocit, jídlo, rostlina, proces, vztah atd. (původně bylo 25 tematických souborů), jestliže platí vlastnost pro třídu, platí i pro všechny její podtřídy a pro všechny její prvky
- funguje mechanismus vzorů a výjimek
  - vzor je hodnota vlastnosti u třídy nebo podtřídy, platí ta, která je blíže objektu
  - výjimka – u konkrétního objektu, odlišná od vzoru
- **synonymie** – nevysvětluje, co jednotlivé významy jsou, ale vyznačují jejich existenci a diferenciaci, významy spojené vztahem synonymie se seskupují do synonymických řad, které jsou zákl. organizačním prvkem sémantické sítě, vynucují si oddělení slovních druhů – nelze volně substituovat jednotky patřící k různým syntaktickým kategoriím
- **antonymie** – centrální organizující vztah pro adjektiva a adverbia
- **meronymie/homonymie** – vztah část – celek, tranzitivní a asymetrický vztah, vede také k budování hierarchických struktur

### Adjektiva – člení se na dvě zákl. třídy

- deskriptivní – jsou organizována v termínech binárních opozic antonymních (*velký: malý*) a podobných významů (synonym)
- relační – *prezidentský, nukleární, zubní* – mají vztah k určitému substantivu nebo jsou s ním nějak spojena, nerozlišují škály, nemají přímá antonyma a nelze je stupňovat
- samostatné stojí skupina referenčně modifikujících adjektiv (*předchozí, údajný*) a adj. označujících barvy



**Slovesa** jsou organizována na základě vztahu vyplývání (*prodávat: platit*) a jeho modifikací: troponymie (*chrápat: spát*) a kauzálních vztahů (*dát: mít*). 15 hlavních tříd, např. slovesa tělesných funkcí, poznání, komunikace, soutěžení, emocí, pohybu a další.

## EUROWORDNET

Samotný WordNet je vytvořen pro angličtinu, EuroWordNet v rámci dvou projektů pokrývá další jazyky (aj, holandština, italština, španělština a francouzština, němčina, čeština, estonština). Princetonsky WordNet rozšiřuje a upravuje.

## ČESKÝ WORDNET

Obsahuje asi 28 tis. synsetů, což pokrývá 47.542 slovních významů (dvojic slovo-smysl). Při jeho vytváření bylo použito výkladového slovníku češtiny (pracovní název pro postupně budovanou lexikální databázi češtiny s důrazem na maximální vnitřní konzistencí, Lingea Lexicon 2.0 (oboustranný aj-čj a čj-aj slovník) a Slovníku českých synonym (Pala, Všianský, 1994). Pomocné zdroje – seznam českých kolokací z textového korpusu ESO (FI MU), gramaticky i strukturálně značkovaný korpus DESAM (FI MU)

## RÁMCE

- varianta sémantických sítí
- všechny informace se ukládají do univerzálních struktur – rámců
- stejně jako sémantické sítě podporují dědičnost
- dobře vyjadřují statické znalosti, tedy nějakou hierarchii pojmů
- rámec obsahuje objekty, sloty a hodnoty

Objekt	Slot	Hodnota
Savec	podtřída	zvíře
	část	hlava
	*má kožich	ano
Slon	podtřída	savec
	*barva	šedá
	*velikost	velký

\* vzorové hodnoty, které mohou měnit hodnoty u podtříd a instancí

## SÉMANTICKÉ SÍTĚ VS. RÁMCE

Sémantické sítě	Rámce
- uzly	- objekty
- spoje	- sloty
- uzel na druhém konci spoje	- hodnota slotu

## Word sense disambiguation

= zjednodušování významů slov, nalezení významu slova v daném kontextu, úkolem je zjistit, jaký význam (z inventáře významů) má slovo ve vstupním textu

Jde o **klasifikační úlohu**

- jednotlivé významy tvoří třídy
- podle kontextu se rozhodujeme, do kterých tříd slovo na vstupu patří
- předpoklady: významy jsou diskrétní a je jich konečný počet, máme nějaký inventář významů
- jde o AI complete problém (těžký problém, k jeho řešení je třeba zvládnout všechny ostatní problémy v dané oblasti)

## ZÁKLADNÍ METODY AUTOMATICKÉ DESAMBIGUACE TEXTŮ

**stochastická (statistická, pravděpodobnostní) desambiguace:** model, který je založen především na pravděpodobnostech přechodu mezi jednotlivými značkami v morfologicky analyzovaném textu. Princip: nejprve ručně (správně) označujeme větší množství textů (řády set tisíc slov), a vznikne tak trénovací korpus. Tagger (statisticky koncipovaný desambiguací program) se poté „naučí“ toto správné značkování, tj. učiní si představu o pravděpodobnostech přechodu mezi jednotlivými značkami a jejich četnostech, kterou uloží do svých vnitřních tabulek. Program pak tyto „naučené“ znalosti aplikuje na nedesambigovaný korpus; v angličtině úspěšnost 97 a 98 %, v ČN 94 % (asi každé 16 slovo je špatně). Hlavní problém: nedostatek trénovacích dat, syntagmatická a slovosledná variabilita textů je příliš velká

**pravidly řízená desambiguace:** Kvůli neúspěšnosti stochastického modelu zahájen vývoj metody disambiguace založené na syntaktických pravidlech. Podstatou je intuitivní formulace celé řady syntaktických pravidel, která odrážejí syntaktické konfigurace češtiny dané jejím vnitřním systémem. Jakmile je formulováno určité pravidlo, které vyplynulo z analýzy obecné chyby, ihned se počítačově implementuje a ověřuje na datech korpusu. Poněvadž tato metoda modeluje jazykový systém, není – na rozdíl od metody stochastické – závislá na trénovacích datech a vlastně je vůbec nepotřebuje. Pokud je možné formulovat nějaké pravidlo se stoprocentní jistotou, budou i data korpusu značkována správně, pokud ovšem není v textu korpusu chyba. Na chyby v textech (např. chybějící slovo či čárka, nesprávná morfologická analýza aj.) je pravidly řízený tagger velmi citlivý, dokáže však některé takové chyby i odhalit. Jelikož je vývoj této metody dosud na počátku, nelze ještě její úspěšnost exaktně kvantifikovat.

**Aplikace WSD:**

- strojový překlad
- inteligentní vyhledávání – information retrieval (IR)
- inteligentní korektor překlepů

Přístupy k WSD jsou **hloubkové** (zahrnují znalosti o jazyce i o světě) a **povrchové** (bez dalších znalostí, počítají s okolím). Hloubkové přístupy potřebují zpracovat a formalizovat příliš mnoho dat, jsou funkční jen v omezených doménách, protože tam je „svět“ malý a experty se vyplatí platit. Povrchové přístupy mají navrch, jsou relativně levné.

Největší slabinou WSD je inventář významů, proto existují jednak snahy vytvořit dobré inventáře a jednak snahy se jim úplně vyhnout (např. projekt Hyperlex). Všechny algoritmy pro WSD pracují s kolokacemi a určitým oknem, ve kterém kolokace sledují. Ono okno může zásadně ovlivňovat průběhy algoritmů. Není žádná doporučená velikost okna. Hlavním důvodem je to, že různá slova mají různý „dopad“ na význam promluvy.

## ALGORITMY WSD

### Algoritmy založené na znalostech

- reprezentant: **Leskův algoritmus** (1986) = slovo  $w$ , jehož okolí sdílí nejvíc slov s definicí (nebo s příklady užití)  $i$  tého významu, má význam  $s_i$ .
  - Pro každý význam si slova  $w$ :
  - nastav váhu  $v$  na 0:  $v(s_i) := 0$
  - najdi množinu slov  $O$  v okolí slova  $w$
  - pro každé slovo  $o_j$  z okolí  $O$
  - pro každý význam  $s_i$

- pokud se  $o_j$  nachází v denici n. př. užití  $D_i$
- přiřti  $v(o)$  k  $v(s_i)$ :  $v(s_i) := v(s_i) + v(o)$
- vyber si s nejvyšším  $v(s_i)$ :  $\text{return } \max(v(s_i))$
- váha slova  $v(o) = \text{idf}_o$
- $\text{idf}_i$  je nižší, čím běžněji se slovo vyskytuje

**Algoritmy strojového učení** – způsobují změnu stavu počítačového systému tak, že zefektivní schopnost přizpůsobení se

- s učitelem – pro zadaný vstup máme i správný výstup (trénovací data)
- bez učitele – pro zadaný vstup neznáme správný výstup
- kombinace – pro část vstupu máme i správný výstup
- typické úlohy strojového učení jsou klasifikační úlohy
- pravidlové – jsou typické pro lidi, intuitivní, příkladem je hra „Myslím si zvíře“ (protihráč se snaží zvíře uhádnout pomocí otázek, na které dostává odpovědi ano/ne – postupná redukce možných správných odpovědí podle toho jaké vlastnosti jsou vybučeny zápornou odpovědí, pokračuje se dokud nezůstane (ideálně) jedno zvíře)
- rozhodovací seznamy – je jednodušší na implementaci

if (zvíře má chobot) then output(slon)

if (zvíře má pruhy) then output(zebra)

if (zvíře má ploutve & zvíře není ryba) then output(žralok)

- rozhodovací stromy – přehlednější i při vyšší složitosti

savec?

žije ve vodě?

žije na souši?

žije v moři? žije v řece? býložravec? masožravec

- matematické
- pravděpodobnostní: naivní Bayesovský
- předpokládá nezávislost znaků, je rychlý
- počítá s podmíněnou pravděpodobností určité vlastnosti (máme černé zvíře → spočítá na kolik procent je možné, že toto zvíře bude kráva, vlk, slon, ...)
- maximální entropie
- podobnost: k-NN ve vektorovém prostoru
- promluvvé
- one sense per discourse
- one sense per collocation (Yarowsky) – závisí na první volbě kolokací, způsobu určení pravděpodobnosti, prahu pro pravděpodobnost a správnosti předpokladu one-sense-per-discourse
- 1. vezmi všechny výskyty slov  $w$  z korpusu včetně jejich kontextů
- 2. pro každý možný význam slova vytvoř malou sadu příkladů (ručně nebo pomocí kolokací)
- 3. vytvoř rozhodovací seznam s pravděpodobnostmi pro další slova vyskytující se v kontextech
- 4. aplikuj tento seznam na celý korpus (s prahem pro pravděpodobnost)
- 5. nově zařazená slova obsahují další slova v kontextech
- 6. algoritmus můžeme upravit pomocí zařazení předpokladu one-sense-per-discourse
- 7. opakuj kroky 3–6
- 8. jakmile množiny přestanou narůstat, zastav
- 9. systém je nyní natrénovaný i na jiný korpus
- redundance atributů

## COMMON SENSE

- sdílená znalost, zahrnuje taková fakta, která zná většina z nás, ale také odvozovací schopnosti, které lidé užívají při aplikaci znalostí
- nemá pevnou hranici, má velký rozsah
- není nutně vědecká znalost (někdy jde i proti ní)
- tvrzení common sense jsou příliš obyčejná, než aby je někdo někde psal
- bez common sense není možné úspěšně modelovat porozumění
- common sense najdeme ve výkladových slovnících, encyklopediích, korpusech, sémantických sítích, specializovaných kolekcích (CyC, OpenMind, ConceptNet)

# Textové korpusy

**Korpusová lingvistika** – nová větev lingvistiky, v níž se pracuje s korpusy uloženými v počítačích. Korpusy jsou v jazykovém inženýrství východiskem pro realistický základní výzkum ve formě relativně blízké přírodním vědám. Umožňují dělat věci, které dříve byly nepředstavitelné pro časovou náročnost a pracnost, systematictější využití statistických a pravděpodobnostních přístupů.

**Korpus** je rozsáhlý vnitřně strukturovaný a ucelený soubor textů daného jazyka elektronicky uložený a zpracováváný. Jsou vytvářeny se zřetelem k zvolenému cíli a vychází z těchto předpokladů:

- jazyková data jsou v korpusu uložena ve své přirozené textové podobě, lze je všestranně a opakovaně zkoumat a vyvozovat z nich příslušné teoretické generalizace
- velký rozsah dat minimalizuje nebezpečí, že by mohlo dojít k převaze jevů okrajových nad základními
- velký rozsah dat je podmínkou reprezentativnosti

**Rozdíl korpus a sbírka textů:** Korpus se skládá ze vzorků (samples), které jsou vybrány dle předem stanovených kritérií tak, aby reprezentovaly jazyk. Korpus je rozsahově i obsahově vymezen a omezen, je uložen v elektronické podobě a obsahuje standardní retence. Tato kritéria sbírka textů splňovat nemusí.

## Co v korpusu nenajdeme

- korpus není elektronická encyklopedie (přesto se spoustu věcí dá zjistit z novinových textů)
- korpus není výkladový slovník (přesto lze významy vyvodit z kontextu)
- korpus není seznam českých slov (nezachycuje izolované jevy)

## Co v korpusu najdeme:

- zjistíme frekvence – např. které varianty jsou používanější (ačkoli, ačkoliv), srovnávat počestěná a cizí slova (talent, nadání), stylistická pozorování (v jakém kontextu se co vyskytuje)

## Word – slovní tvar

**Lemma** – základní tvar pro nějakou skupinu tvarů (nominativ u substantiv, infinitiv u sloves).

**Token** – výskyt slovního tvaru v korpusu, slovní tvar jako takový; textová slova, tj. výskyt slovních tvarů (slovních exemplářů), čísel, zkratk, speciálních znaků a interpunkčních znamének. Základní prvek korpusu (pozice)

**Tokenizace** – rozdělení na slova, rozčlenění textu na základní jednotky určené pro vyhledávání

**Typ:** slovní tvar jako takový

Př.: Token (počet všech unikátů v korpusu, tj. i interpunkce) a type (počet všech různých slov) => např. krásy krásy, jak si vlastně povídáte => token = 7, type = 5)

**Tag** je poziční atribut obsahující informaci o slovním druhu a dalších morfologických charakteristikách daného slova. Pro již zmíněnou konkordační řádku „Mýt v zimě auto, či nemýt.“ je hodnota atributu word na pozici -1 „NNFS6—A—“.

**Tagování** je značkování na úrovni slovních druhů

## BUDOVÁNÍ KORPUSŮ

Zdrojem dat je psaný i mluvený jazyk. Z psaných textů se data získávají třemi způsoby:

- konverzi ze sázecích disket a pásek, které lze získat od většiny nakladatelství
- užitím OCR – závisí na kvalitě scanneru a programového vybavení a typografické složitosti textu
- klasickým opisováním do počítače

## TYPOLOGIE KORPUSŮ

Malé, střední, velké

Korpusy monitorovací, paralelní (dva a více jazyků), studijní, cvičné a testovací

Lingvistické členění: psané (textové), mluvené; synchronní a diachronní

## TYPY KORPUSŮ A STANDARDIZACE

**Elektronické archivy** – volné kolekce různorodých textů, různé formáty a jazyky.

**Vlastní korpusy** – tvoří úplné celky. Většina Evropských jazyků už má dnes svůj vlastní korpus. Existují také korpusy paralelní – dvojjazyčné, obecné a specifické, velké korpusy obsahující subkorpusy jazyka psaného, mluveného, nářečí, synchronní – diachronní aj.

S rostoucím počtem korpusů vzniká potřeba jejich standardizace → vznik **Text Encoding Initiative (TEI)** – vydala doporučení pro společný výměnný formát, zásady kódování, znakové sady, navrhla společný kódovací – značkovací jazyk **Standard Generalized Markup Language (SGML)**.

## OBSAH KORPUSU

- text
- metainformace
- struktura dokumentu – odstavce, nadpisy, verše, věty
- značkování – informace o slovech/pozicích, morfologie, základní tvary, syntaktické vazby, ...

## TOKENIZACE

### Rozdělení textu do pozic

- může silně ovlivnit výsledky dotazování, četnosti i značkování
- token (pozice) = základní prvek korpusu
- většinou slovo, číslo, interpunkce
  - bude-li, don't – 4 možnosti:
    1. |don't|
    2. |don| |'t|
    3. |don| |'| |t|
    4. |do| |n't| – v BNC
  - zkratky (s tečkama?)
  - data
  - desetinná čísla, ...

## VYUŽITÍ KORPUSŮ

- pro lingvisty (výstavba popisů jazyka, tvorba a ověřování teorií, tvorba velkých aplikací založených na korpusových datech – hlavně lexikografové)
- pro další odborníky (literární vědce, sociology, psychology, pedagogy)
- pro studenty i laiky

## ZNAČKOVÁNÍ KORPUSU

### ZNAČKOVACÍ JAZYK

jákýkoli jazyk, který vkládá do textu značky vysvětlující význam nebo vzhled jednotlivých jeho částí. Text obohacený o dodatečné informace (o významu, struktuře, způsobu zobrazení jednotlivých částí textů), se původně používal jen pro formátování textu v nakladatelstvích - dodnes např. formátovací jazyk TeX (formátování knih do tisku). Dalšími jazyky jsou troff, PDF. Nejznámějšími značkovacími jazyky jsou HTML a XML, v nichž je vytvořena většina WWW stránek. Pro potřeby korpusové lingvistiky se používá SGML jazyk, dnes XML. Pro potřeby KL se používal jazyk SGML, dnes XML.

### METODY ZNAČKOVÁNÍ

**vnější anotace:** typ textu, autor, rok (lze pak vybírat texty z určitého roku, texty psané jen ženami apod.)

**vnitřní anotace:** strukturní informace (členění na kapitoly, odstavce, věty, slova) a informace lingvistické, ty jsou nákladné => omezuje se na morfologické značkování jednotlivých slovních tvarů (tagování), přiřazení slovnědruhové charakteristiky a lemmatizaci (přiřazení základního, slovníkového tvaru)

### VÝHODY ANOTOVANÝCH KORPUSŮ

- odrazový můstek pro další lingvistické výzkumy, nedocenitelný test pro lingvistickou teorii
- trénovací data, na nichž se počítačové programy učí automaticky anotovat slova
- poskytuje výchozí statistická data pro pravděpodobnostní zpracování jazyka
- přináší lingvistické interpretace na různých úrovních, slouží k výzkumu jazyka (vytěžování korpusu pro lingvisty)
- slouží pro aplikace NLP – budování modelů jazyka na základě dat získaných z korpusů

### RUČNÍ ZNAČKOVÁNÍ KORPUSU

Značkují je anotátoři, zkušení a vyškolení pracovníci. Ruční značkování korpusu je nákladná věc, využívá se hlavně kvůli tzv. **desambiguaci**, tj. zjednodušování víceznačných jednotek.

Používá se i při **pročišťování korpusu – odstraňování překlepů**.

Využívá se k sestavování **trénovacích dat**, na nichž se pak učí nástroje automaticky anotovat.

## OBECNÉ ZÁSADY VYTÝČENÉ G. LEECHEM, PODLE NICHŽ MAJÍ BÝT ZNAČKY VYTVÁŘENY

- zachovat vratnost anotovaného korpusu do surového stavu (autor značek je interpretem, s nímž nemusí každý potenciální uživatel souhlasit, přičemž by mělo být technicky možné se případně nežádoucí interpretace zbavit a moci pracovat bez ní)
- možnost extrahovat anotace z textu a uložit je zvlášť, aby bylo možné se k nim vrátit (formou nějaké relační databáze, nebo interlineárního formátu)
- anotační schéma by mělo vycházet z teoretických východisek, která by měla být jasně formulovaná a přístupná každému konečnému uživateli korpusu. Mnohé korpusy byly anotovány ručně (existence subjektivních interpretací zaviněných osobou anotátora ve sporných případech). Značkování by pak mělo být doplněno komentáři, z nichž by byl důvod příslušné volby patrný.
- mělo by být jasné, JAK a KDO anotaci provedl (JAK – ručně × automaticky × polobautomaticky, s postkorekcí × bez korekce; KDO – počítačový program, anotátor - člověk)
- uživatel korpusu by si měl být vědom toho, že anotace nejsou nějakou nedotknutelnou neomylnou instancí. Anotace je pouze více či méně užitečným nástrojem. INTERPRETACE.
- anotační schéma by mělo být založeno na široce schvalovaných a teoreticky nezátížených principech. Není na škodu i zjednodušující přístup.
- žádné anotační schéma nemá právo být pokládáno za standardní. Je-li nějaké řešení uznávanější, děje se tak pouze z praktických důvodů.

## LINGVISTICKÉ INFORMACE DVOJÍHO TYPU

**LEMMATIZACÍ** je danému slovu přiřazena informace o jeho základním, slovníkovém tvaru zvaném lemma, popř. o více možných základních tvarech (tj. u slov vícestupňových v rámci jednoho SD nebo v rámci více SD)

**PŘÍRAZENÍ POTENCIÁLNÍCH MORFOLOGICKÝCH INTERPRETACÍ** každé formě, tj. informace o její slovnědruhov příslušnosti a morfologických vlastnostech (např. o rodu, čísle a pádu podstatných a přídavných jmen, zájmen a číslovek, o stupni přídavných jmen a příslovčí, o osobě, čísle, slovesném a jmenném rodu slovesných tvarů atd.). Morfologická interpretace daného slova je formálně vyjádřena morfologickou značkou tvořenou maximálně 15 údaji, z nichž každý je reprezentován jedním znakem na dané pozici, přičemž význam jednotlivých pozic je jednoznačně stanoven.

## ZNAČKOVÁNÍ V ČNK

Celkem 15 pozic, 2 jsou volné pro případ nových kategorií.

- **pozice 1 – slovní druh:** A (adjektivum); C (numerál), D (adverbium), I (interjekce), J (konjugace), N (substantivum), P (pronomen), R (prepozice), T (partikule), X (neznámý), Z (interpunkce)
- **pozice 2 – detailní určení slovního druhu:** slouží především k určení dalších relevantních morfologických kategorií, které jsou uvedeny na dalších pozicích
- **pozice 3 – jmenný rod:** F (femininum), H (femininum nebo neutrum), I (maskulinum inanimatum), M (maskulinum animatum), N (neutrum), Q (femininum singuláru nebo neutrum plurálu), T (maskulinum inanimatum nebo femininum), Y (maskulinum – i. nebo a.), Z („nikoli femininum“)
- **pozice 4 – číslo:** D (duál), P (plurál), S (singulár), W, X (libovolné číslo)
- **pozice 5 – pád:** čísla 1–7
- **pozice 6 – přivlastňovací rod:** rody mužský neživotný a střední se nikdy nevyskytují samostatně; F (femininum), M (maskulinum animatum), X (libovolný), Z („nikoli femininum“)
- **pozice 7 – přivlastňovací číslo:** P (plurál), N (neutrál)
- **pozice 8 – osoba:** 1–3; X
- **pozice 9 – čas:** F (futurum), H (minulost), P (přezens), R (minulý čas), X
- **pozice 10 – grade:** 1–3
- **pozice 11 – negace:** A (afirmativ – bez negativní předpony), N (negace – tvar s negativní předponou „ne-“)
- **pozice 12 – aktivum, pasivum:** A (aktivum), P (pasivum)
- **pozice 13 – nepoužito**
- **pozice 14 – nepoužito**
- **pozice 15 – varianta, stylový příznak apod.**

## ZNAČKOVÁNÍ BRNĚNSKÉHO KORPUSU

### k1 - Substantivum

x	Speciální vzor
P	půl

g	Rod
M	Rod mužský životný
I	Rod mužský neživotný
N	Rod střední
F	Rod ženský
R	Rodina (příjmení)

n	Číslo
S	Číslo jednotné
P	Číslo množné
D	Duál
R	Hromadné ozn. čl. rodiny

c	Pád
1	Pád první
2	Pád druhý
3	Pád třetí
4	Pád čtvrtý
5	Pád pátý
6	Pád šestý
7	Pád sedmý

### Tvary

Značka	Typ tvaru
zS	tvar s příklonným s

### Poznámky

Značka	Poznámka
rD	deverbativum
hF	ženské posesivum
hM	mužské posesivum
hR	rodinné posesivum
tQ	vyjadřuje míru
tA	vyjadřuje zřetel
tL	vyjadřuje místo
tI	vyjadřuje čas
tC	vyjadřuje příčinu
tM	vyjadřuje způsob
tD	příslovce modální
tS	příslovce stavové
hT	zastupuje věc (což)
hP	zastupuje osobu (kdo?)
xU	číslovka základní
xO	číslovka řadová
xR	číslovka druhová

### k2 - Adjektivum

e	Negace
A	Afirmace
N	Negace

g	Rod
M	Rod mužský životný
I	Rod mužský neživotný
N	Rod střední
F	Rod ženský

n	Číslo
S	Číslo jednotné
P	Číslo množné
D	Duál

c	Pád
1	Pád první
2	Pád druhý
3	Pád třetí
4	Pád čtvrtý
5	Pád pátý
6	Pád šestý
7	Pád sedmý

d	Stupeň
1	Positiv
2	Komparativ
3	Superlativ

### Styl

w	Stylistický příznak
A	archaismus
B	hásknický
C	pouze v korpusech
E	expresivně
H	hovorově
K	knižně
O	oblastně
R	řidčeji
Z	zastarale

### k3 - Zájmeno

p	Osoba
1	První osoba
2	Druhá osoba
3	Třetí osoba
X	1., 2. nebo 3.

g	Rod
M	Rod mužský životný
I	Rod mužský neživotný
N	Rod střední
F	Rod ženský

n	Číslo
S	Číslo jednotné
P	Číslo množné
D	Duál

c	Pád
1	Pád první
2	Pád druhý
3	Pád třetí
4	Pád čtvrtý
5	Pád pátý
6	Pád šestý
7	Pád sedmý

### k4 - Číslovka

g	Rod
M	Rod mužský životný
I	Rod mužský neživotný
N	Rod střední
F	Rod ženský

n	Číslo
S	Číslo jednotné
P	Číslo množné
D	Duál

c	Pád
1	Pád první
2	Pád druhý
3	Pád třetí
4	Pád čtvrtý
5	Pád pátý
6	Pád šestý
7	Pád sedmý

### Poznámky-pokr.

Značka	Poznámka
yQ	tázací
yR	vztahné
yN	zápor
yl	neurčitost
yF	reflexivní
xD	ukazovací
xI	vymezovací
xP	osobní
xO	privlastňovací
xU	spojka souřadici
xS	spojka podřadici
c1	předložka s 1. pádem
c2	předložka s 2. pádem
c3	předložka s 3. pádem
c4	předložka s 4. pádem
c5	předložka s 5. pádem
c7	předložka s 7. pádem
aP	dokonavě
ai	nedokonavě
aB	obovově
wh	hovorově

### k5 - Sloveso

e	Negace
A	Afirmace
N	Negace

u	Vid
P	Perfektivum
I	Imperfektivum
B	Obovově

m	Typ (Mód)
F	Indinitiv
I	Indikativ
R	Imperativ
A	Přičestí činné (minulé)
N	Přičestí trpné
S	Přechodník přítomný
D	Přechodník minulé
B	Indikativ futura

p	Osoba
1	První osoba
2	Druhá osoba
3	Třetí osoba

g	Rod
M	Rod mužský životný
I	Rod mužský neživotný
N	Rod střední
F	Rod ženský

n	Číslo
S	Číslo jednotné
P	Číslo množné

### k6 - Příslovce

e	Negace
A	Afirmace
N	Negace

d	Stupeň
1	Positiv
2	Komparativ
3	Superlativ

### k7 - Předložka

### k8 - Spojka

### k9 - Částice

### k0 - Citoslovce

### kA - Zkratka

### kY - by,aby,...

m	Vztah k sl. módu
C	kondicionál

p	Osoba
1	První osoba
2	Druhá osoba
3	Třetí osoba

n	Číslo
S	Číslo jednotné
P	Číslo množné

## ROZDÍL MEZI PRAŽSKOU A BRNĚNSKOU ZNAČKOU

**NNNS4-----A----**: substantivum, obyčejné, neutrum, singulár, akuzativ, afirmace

**k1gNnSc4**: substantivum, neutrum, singulár, akuzativ (nezachycena specifikace a afirmace)

**Výhody brněnského systému**: přehlednější, úspornější, snadno rozšiřitelný

## Známé korpusy

### První korpus - Brown

- americká angličtina (1961)
- Brown University, 1964
- gramatické značkování, 1979
- 500 textů, 1 mil. slov
- W. N. Francis & H. Kučera
  - první statistické charakteristiky angličtiny
  - relativní četnosti slov a slovních druhů

### SUSANNE

- autor Geoffrey Sampson, Sussex University
- kniha English for the Computer, 1995
- část korpusu Brown (1/4)
- nové gramatické značkování
- syntaktické značkování

### British National Corpus

- britská angličtina, 10% mluva
- první velký korpus pro lexikografy
- vydavatelé slovníků (OUP) + univerzity



- 1. verze: 1991–1994, 2. verze: World Edition 2000
- ≈3000 dokumentů, 100 mil. slov
- gramatické značkování automatickým nástrojem

#### **Bank of English**

- britská angličtina
- COBUILD (HarperCollins), University of Birmingham
- 1991, dále rozšiřován
- 2002, ≈450 mil. slov

#### **Další národní korpusy**

- Český národní korpus
  - UČNK, FF UK
  - SYN2000, SYN2005, SYN2010 `a 100 mil. slov
  - Litera, Synek, BMK, ...
- Slovenský, Maďarský, Chorvatský, ...
- Americký

#### **Pražský mluvený korpus, Brněnský mluvený korpus**

##### **Korpusy na FI**

##### **vytvořené na FI, příklady:**

- Desam
  - 1996, ručně značkováný (desambiguovaný)
  - ≈1 mil. slov
- Czes
  - periodika z webu, z let 1996–1998, další el. zdroje, webové zdroje (crawl)
  - ≈465 mil.
- \*TenTen
  - různé jazyky, ve spolupráci s LCL, UK
  - 1–20 mld. pozic
- Chyby
  - práce studentů předmětu Základy odb. stylu s vyznačenými chybami
  - ≈400 tis.

## **ČESKÝ NÁRODNÍ KORPUS**

Vznikl v roce 1994, vytvářen Ústavem Českého národního korpusu na FF UK, vedený Františkem Čermákem; akademický, nekomerční projekt

Části: SYN2000 (100 000 000 slovních tvarů), DIAKORP (1 750 000 tvarů), ORAL-PMK (700 000 tvarů)

#### **Žánrové složení korpusů psaného jazyka ČNK a pojetí synchronního korpusu**

Publicistika: 60 %; po roce 1990

Odborné texty: 25 %; po roce 1989

Krásná literatura: 15 %, po roce 1990 (doplňující kritéria vzhledem k přetiskům a čtení knih starších autorů; do synchronního korpusu se zařazují i současně čtení starší autoři, kteří se narodili roku 1880 a později a knihy publikované od roku 1945)

## **PRAŽSKÝ A BRNĚNSKÝ MLUVENÝ KORPUS**

**Pražský mluvený korpus:** první korpus mluvené češtiny, zachycuje autentickou mluvenou češtinu, hlavně obecnou a tematicky nespécializovanou. Materiály pochází z Prahy a jejího okolí, kde pracuje spousta obyvatel z celé ČR. Praha rovněž ovlivňuje zbytek republiky mediálně, proto v ČR není místo, které by bylo více reprezentativnější. Nahrávky pocházejí z roku 1988-1996.

**Brněnský mluvený korpus:** první mluvený korpus češtiny z oblasti Moravy. Zaznamenává autentickou tematicky nespécializovanou mluvu města Brna. BMK je elektronickým přepisem 250 anonymních magnetofonových nahrávek z let 1994-1999 zachycujících 294 mluvčích.

**Rozdíl mezi PMK a BMK:** BMK se pokouší nahradit tradiční interpunkci interpunkcí pauzovou. Striktně zachycuje simultánnosti dialogických promluv. Pausová interpunkce je náročnější -> difference zápisu u přepisovatelů; zatím bez morfologického značkování. Praha: do jisté míry ukazuje skutečnost.

# *Uložení korpusu v počítači*

---

## FORMÁTY KORPUSŮ

1. archiv/kolekce
  - různé formáty, podle zdroje/typu
1. textové banky
  - jednotný formát a základní struktura
  - dokumenty/texty, základní metainformace
1. vertikální text
2. binární data v aplikaci
  - pomocná data pro rychlejší zpracování
    - – indexy
    - – statistiky

## KÓDOVÁNÍ ZNAKŮ

- 8 bitů  $\approx$  256 znaků
  - ASCII – základ 7 bitů
  - kódování pro češtinu
    - ISO-Latin-2, Windows-1250, 852
- Unicode
  - 32bitů na znak
  - UTF-8
    - 1 až 4 byty na znak
  - UTF-16
    - 2 až 4 byty na znak

## KÓDOVÁNÍ METAINFOREMACÍ

- escape-sekvence – speciální znak mění význam následujících znaků \n, \t, &amp;, <tag>
- SGML – • Standard Generalised Markup Language, ISO 8879:1986(E)
- XML – Extensible Markup Language, W3C, 1998
  - struktura popsána v DTD
  - elementy
    - počáteční, koncová značka
    - <doc>, <head>, </head>, <g/>
  - atributy elementů/značek
    - <doc title="Jak pejsek ..." author="čapek">
    - <head type="main">
  - entity – &gt;, &lt;, &amp;, &acute;

## STANDARDY PRO UKLÁDÁNÍ TEXTŮ

- SGML/XML
- TEI
  - Text Encoding Initiative (1994)
  - TEI Guidelines for Electronic Text Encoding and Interchange
- CES, XCES
  - Corpus Encoding Standard

## VERTIKÁLNÍ TEXT

- jednoduchý formát i jeho zpracování
  - každý token na samostatném řádku
  - struktury formou XML značek
  - značkování odděleno tabulátorem (různé atributy k dané pozici)
- podrobnosti na: <http://nlp.fi.muni.cz/> → Informace pro současné a potenciální spolupracovníky → Textové korpusy → Popis vertikálů

# Indexování a prohledávání korpusu

*Nenašla jsem nic konkrétního v dostupných PLIN materiálech, tak alespoň popíšu, co je INDEX, jak se pomocí něj vyhledává apod. Analogicky by to mohlo fungovat i na korpusech... snad :)*

**Index** – někdy označován jako KEY (klíč), jde o databázovou konstrukci, která usnadňuje vyhledávání a dotazování v databázích; definován výběrem tabulky a konkrétního sloupce, nad kterými se má vyhledávání urychlit. Vytvoření indexu je náročnější na operační paměť a prostor na disku (při tvoření a zápisu indexu je ukládání do databáze mnohem pomalejší, vyhledávání je však mnohonásobně rychlejší).

Index se tvoří mj. **pomocí příkazu jazyku SQL** a datový server do něj poté uloží informace o rozmístění hodnot indexovaných sloupců. Při vyhledávání dotazu pak není tabulka prohledávána podle toho, jak je sestavena, ale pomocí informací, které jsou uloženy v paměťovém prostoru indexu, se rovnou přejde na relevantní řádky. Zjednodušeně **jde o rejstřík** (jako v knize), který sice neurčí přesné místo výskytu dotazu, ale **zmenší prohledávanou oblast** (odkáže na paměťové místo s hledanými daty).

Na první pohled se může zdát, že čím více indexů, tím lepší a rychlejší vyhledávání. Opak je však pravdou, a to kvůli zmiňované potřebě paměťového místa pro indexy. Při vytvoření dostatečného množství indexů by pak mohlo dojít k situaci, že paměť zabraná pro jejich chod je shodná s pamětí zabranou samotnými daty. Druhým uskalím je fakt, že každý **index zpomaluje práci databáze**, protože při jakýchkoliv změnách v databázi musí být provedena i **změna dat odpovídajících indexů** (např. přidání řádků do tabulky apod.).

## DRUHY INDEXŮ:

- **PRIMARY** (primární index) – tvoří ho sloupec (kombinace více sloupců) obsahující primární klíč, v každé tabulce se smí vyskytnout pouze jednou, dle konvence nazývá se ID; jedná se o zvláštní případ druhu klíče **UNIQUE**
- **UNIQUE** (unikátní index) – podobně jako předchozí typ, rozdíl: indexů může být více (příklad: kromě ID záznamu o osobě můžeme požadovat i unikátnost sloupce s loginovým jménem osoby; kombinace rodového a druhového jména zvířat apod.)
- **INDEX** (též **SECONDARY**) – sloupce obsahující sekundární/druhotný klíč (též vedlejší index), umožňuje vyhledávání pomocí dalších sloupců – nejen pomocí primárních nebo unikátních indexů; na rozdíl od předchozích lze vkládat do tabulky záznamy, které nejsou v sekundárním indexu unikátní
- **FULLTEXT** – optimalizace full.textového vyhledávání, způsob provedení záleží na samotném databázovém serveru

## KORPUS Z POHLEDU INFORMATIKY

### KORPUS:

rozsáhlý vnitřně strukturovaný a ucelený soubor textů daného jazyka elektronicky uložený a zpracovaný. Texty psané jedním přirozeným jazykem, jsou strojově čitelné. Datový soubor, elektronická databáze. Pro účely korpusového manažeru je korpus vnímám jako posloupnost pozic.

### TVORBA KORPUSŮ:

pouhým převodem dostupných textů v elektronické podobě; skenování textů a převádění je pomocí programů na rozpoznávání písma (OCR), ruční přepis

### FORMÁTY KORPUSŮ:

- archiv/kolekce (různé formáty, podle zdroje/typu)
- textové banky (jednotný formát a základní struktura; dokumenty/texty, základní metainformace)
- vertikální text
- binární data v aplikaci (pomocné data pro rychlejší zpracování; indexy, statistiky)

### OBSAH KORPUSU:

- text
- metainformace (autor, rok, publikace, pohlaví cílové skupiny)
- struktura dokumentu (odstavce, nadpisy, verše, věty)
- značkování (informace o slovech, morfologie, základní tvary)

### KÓDOVÁNÍ ZNAKŮ

- 8 bitů  $\approx$  256 znaků
  - ASCII – základ 7 bitů
  - **pro češtinu dvě hlavní kódování:** ISO-Latin-2 podle mezinárodní normy ISO [ISO99] a Windows 1250 prosazované společností Microsoft v operačních systémech Windows. Na internetu se setkáme s „kódování“ ASCII, tedy psaní „cestiny“ bez háčků a čárek (což není správný český text)
  - ISO-Latin-2, Windows-1250, 852

- Unicode
  - 32bitů na znak
  - UTF-8
    - 1 až 4 byty na znak
  - UTF-16
    - 2 až 4 byty na znak

### KÓDOVÁNÍ META INFORMACÍ

- escape-sekvence – speciální znak mění význam následujících znaků \n, \t, & amp;, < tag>
- SGML – • Standard Generalised Markup Language, ISO 8879:1986(E)
- XML – Extensible Markup Language, W3C, 1998
  - struktura popsána v DTD
  - elementy
    - počáteční, koncová značka
    - < doc>, < head>, < /head>, < g/>
  - atributy elementů/značek
    - < doc title="Jak pejsek ..." author="čapek">
    - < head type="main">
  - entity – & gt;, & lt;, & amp;, & eacute;

### STANDARDS PRO UKLÁDÁNÍ TEXTŮ

- SGML/XML
- TEI
  - Text Encoding Initiative (1994)
  - TEI Guidelines for Electronic Text Encoding and Interchange
- CES, XCES
  - Corpus Encoding Standard

## STRUKTURA KORPUSU

### LOGICKÁ STRUKTURA KORPUSU

- korpus pro účely korpusového manažeru = množina pozičních atributů, pro každý z nich obsahuje korpus na každé pozici nějakou hodnotu
- pro každý korpus definujeme množinu pozičních atributů, pro každý z nich obsahuje korpus nějakou hodnotu.
- korpus může obsahovat značkování ve formě struktur. Struktury tvoří pozice, jsou jaksi nad nimi
- pro každou strukturu v korpusu jsou definovány intervaly pozic, které struktura „pokrývá“. Některé struktury navíc mohou obsahovat množinu atributů, které jsou analogií pozičních atributů. Každý interval musí obsahovat hodnotu pro každý takový atribut.
- paralelní korpusy jsou tvořeny samostatnými korpusy pro každý z jazyků; jednotlivé korpusy navíc obsahují pro všechny ostatní jazyky zarovnání určující sobě si odpovídající části textu (nejčastěji zarovnávají texty/překlady pouze dvou jazyků)

### POZIČNÍ ATRIBUT

U každé pozice (slovo, číslo, interpunkční znaménko apod.) je v dostupných korpusech přiřazen jeden nebo více pozičních atributů. **Poziční atribut** je textová informace, která se vztahuje k dané pozici. Atribut může obsahovat například informaci o slovním druhu slova, jeho základním tvaru atd. Různé korpusy obsahují různé soubory atributů. V následujícím textu předpokládáme, že pracujete s korpusem SYN2000. V korpusu SYN2000 najdeme poziční atributy následovně: Zobrazení – Atributy; lze vybrat základní slovní tvar (**lemma**), morfologické značky (**tag**), hledané slovo psané malými písmeny (**lc**) a slovní druh (**pos**).

### STRUKTURY

- Pomocí následujících struktur dokážeme pokrýt všemným ožnuti značkování struktury textu, ktré obsahuje jazyk SGML. I když v korpusovém manažeru CQP lze uchovávat hodnotu pro struktury, nelze je používat v dotazech. Systém se omezuje pouze na existenci začátku nebo konce určité struktury. Používá se zápisu ve tvaru SGML, tedy jméno struktury uzavřené v úhlových závorkách, např. < s> jako začátek věty a < /s> jako konec věty.

Typy struktury: plochá, stromová, prázdná. Pomocí nich můžeme pokrýt všechny možnosti značkování struktury textu, které obsahuje jazyk SGML.

- **Plochá struktura:** Struktury nejsou vnořovány, ani se nijak nepřekrývají. Každá struktura je dána svým začátkem a koncem. Mohou korpus rozčleňovat na části, tedy kde jedna struktura končí, tam další začíná – příkladem je doc. Nebo se jednotlivé intervaly vyskytují samostatně v textu – např. head, lang.
- **Stromová struktura:** struktury jsou dány začátkem a koncem, ale navíc lze stejné značky vnořovat do sebe. Například list, q.
- **Prázdná struktura:** značky tvoří pouhé „oddělovače“. Jsou dány jedním údajem: mezi kterými pozicemi se nachází. Příklad jsou značky pre a g.

```
<doc file="S/NWS/1994/Ind94039" id=081>  
  kopcích/kopce/1 Okolo/okolo/7 města/město/1 ./.</s><s>První/první/4  
    korpus/korpus/1  
      bosenskė/bosenský/2 armády/armáda/1 ,./ bránící/bráněný/2
```

Obrázek 2.2: Formát KWIC se zobrazenými pozičními atributy a strukturami.

## STRUKTURNÍ ATRIBUT

**Strukturní atributy** jsou prvky vkládané do korpusu mezi pozice, které korpus strukturují na různě velké souvislé celky (věty, dokumenty, knihy).

Strukturní atributy rozdělují korpus na menší celky, odrážející strukturu původního zdrojového dokumentu. Každý korpus může obsahovat své vlastní strukturní atributy, mezi nejčastější patří např. „opus“ nebo „doc“ (např. „doc.autor“ obsahuje informaci o autorovi původního dokumentu).

**Atributy struktur** k jednotlivým intervalům struktur přidávají další informace. Můžeme je brát stejně jako poziční atributy, tedy hodnotami jsou řetězce znaků, resp. množiny řetězců znaků, lze pro ně definovat dynamické atributy. Žádným způsobem nebudeme omezovat počet různých atributů k jedné struktuře. Každá struktura má samozřejmě svoji vlastní skupinu atributů, jejich jména se případně mohou shodovat.

Vnitřní struktura korpusu

1) atributy poziční

2) atributy strukturní (hranice vět, odstavců)

slovo	lemma	gr.značky	sém.značky
ženu	hnát/žena	k5/k1gFnSc1	HUM+FEM/POHYB
ovce	ovce	k1gFnPc4	ANIM
na	na	k7c4	DIRECT
pastvu	pastva	k1gFnSc4	LOC

## PROČ UCHOVÁVAT KORPUS JAKO DATABÁZI A INDEXOVAT JI?

- morfologické značkování korpusu: zvýšená informační hodnota korpusu
- možnost hledání v korpusu podle morfologických kategorií
- morfologická databáze
- předpoklad pro další stupně analýzy jazyka: syntaktická, sémantická
- předpoklad pro navazující aplikace: Word Sketch Engine
- zapojení do dalších nástrojů pro práci s jazykem: kontrola pravopisu, překladače, slovníky, webové prohlížeče
- možnost adaptace pro jiné slovanské jazyky

# TOKEN, TYPE, REVERZNÍ INDEX, ZÁKLADNÍ OPERACE NAD DATABÁZÍ KORPUSU

**Token:** počet všech znaků v korpusu (i interpunkce)

**Type:** počet všech různých slov

**Reverzní index,** často též označovaný invertovaný soubor (inverted file), obsahuje pro každé slovo (type) z korpusu samostatně seznam pozic, na kterých se dané slovo vyskytuje.

Uvedme si příklad na korpusu KOČKA:

Část textu	Signatura
Kočka na okně	0010 1110 0110
Na okně seděla kočka, byl horký letní den,	0111 1111 0110
na okně seděla kočka a koukala se ven,	1010 1111 0111

Slovo	Seznam pozic
Kočka	1
na	2 14 32 70
okně	3 5 15 33 71
Na	4
seděla	6 16 34 72
kočka	7 17 35 73
,	8 13 22 31 36 58 64 69
byl	9 23
horký	10 24 42
letní	11 25 43
den	12 26 44

Reverzní index tedy obsahuje dva seznamy (slovo – type a seznam pozic). Výsledkem reverzního indexu je seznam pozic přímo uložený v reverzním indexu.

Tab. 3. Část reverzního indexu korpusu KOČKA

Pokud jednotlivé seznamy uspořádáme podle velikosti (a to je výhodné i z hlediska vyhledávání), můžeme ukládat pouze rozdíly mezi po sobě jdoucími prvky seznamu -> **rozdílové reverzní indexy**.

Slovo	Seznam rozdílných pozic
Kočka	1
na	2 12 18 38
okně	3 2 10 18 38
Na	4
seděla	6 10 18 38
kočka	7 10 18 38
,	8 5 9 9 5 22 6 5
byl	9 14
horký	10 14 18
letní	11 14 18
den	12 14 18

Tab. 5. Část reverzního rozdílového indexu korpusu KOČKA

Ještě jednou pro srovnání tabulka 1 a tabulka 5 vedle sebe, kde je vidět, jak se v tabulce 5 ukládají pouze rozdílová čísla (např. řádek 2 -> 70 - 32 = 38 -> do tabulky 5 se uloží jen 38).

Slovo	Seznam pozic
Kočka	1
na	2 14 32 70
okně	3 5 15 33 71
Na	4
seděla	6 16 34 72
kočka	7 17 35 73
,	8 13 22 31 36 58 64 69
byl	9 23
horký	10 24 42
letní	11 25 43
den	12 26 44

Tab. 3. Část reverzního indexu korpusu KOČKA

Slovo	Seznam rozdílných pozic
Kočka	1
na	2 12 18 38
okně	3 2 10 18 38
Na	4
seděla	6 10 18 38
kočka	7 10 18 38
,	8 5 9 9 5 22 6 5
byl	9 14
horký	10 14 18
letní	11 14 18
den	12 14 18

Tab. 5. Část reverzního rozdílového indexu korpusu KOČKA

## PROHLÉDÁVÁNÍ KORPUSU

viz CQL

## Nástroje pro práci s korpusy

Základem jsou obvykle **konkordanční programy** – třídí a počítají objekty nalezené v korpusu (slovní tvary, interpunkce, vyznačují i hranice vět a odstavců, závisí na značkování). Ke korpusu se přidružují nástroje – gramatické analyzátoři – **značkovací programy (taggers)** na různých úrovních (morfologie, syntax, sémantika) – ke každému slovu/slovnímu tvaru přiřazují gramatické značky (slovní druh a gramatické kategorie).

**Korpusové manažery** – program umožňující efektivně pracovat s počítačovým korpusem, tj. vyhledávat podle zadatelných kritérií (slovní tvar, značka, lemma) ve formě KWIC, vyhledané informace třídí a statisticky zpracovává, vytvářet subkorpusy, ukládat získané informace, využívat standardních statistických metod pro vyhledávání kolokací atd. Program musí splňovat dvě základní kritéria, z nichž prvním je dostatečná rychlost při vyhledávání požadovaných lingvistických jevů a druhým uživatelsky příjemné rozhraní. Např. GCQP nebo Bonito, který je vyvíjen a ověřován.

Mají vlastní procesor, uživatelské rozhraní, lze zadávat vyhledávací dotazy, výstupem jsou konkordanční seznamy, výskyty slov a slovních tvarů v kontextech, kolokace, frekvenční údaje, statistické parametry. Nástroje na zpracování korpusů – uložení textu, editace/příprava textu, značkování, rozdělení do pozic (tokenizace), vyhledávání (konkordance), statistiky.

### SYSTÉM MANATEE

- výkonný korpusový manažer
- aplikace provádějící operace nad samotnými korpusy
- neposkytuje uživatelské rozhraní ani neřeší správu uživatelů. K tomuto účelu slouží aplikace Bonito 1 a Bonito 2
- přímo podporuje – uložení textu, vyhledávání (konkordance), • statistiky
- externí nástroje – značkování, rozdělení do pozic
- hlavní zaměření – velké korpusy, rozsáhlé značkování (morfologické, syntaktické, metainformace), návaznost na další aplikace/nástroje (korpusový editor (CED), tvorba slovníků), univerzálnost (různé jazyky, k'odování, systémy značek)

#### Klíčové vlastnosti

- modulární systém
- přístup z různých rozhraní – grafické uživatelské rozhraní (Bonito), aplikační programové rozhraní (API), příkazový řádek
- rozsáhlá data – stovky mld. pozic, neomezeně atributů a metainformací
- rychlost – vyhledávání, statistiky
- multihodnoty – zpracování víceznačných značkování
- dynamické atributy – vyhledávání a statistiky na počítaných datech
- subkorpusy, paralelní korpusy
- silný dotazovací jazyk – dotazy na všechny atributy, metainformace, pozitivní/negativní filtry, regulární výrazy + booleovské operátory
- frekvenční distribuce – víceúrovňová, všechny atributy a metainformace
- kolokace - různé statistické funkce

### KORPUSOVÉ MANAŽERY, KTERÉ POUŽÍVÁ ČNK:

**Bonito 1:** je desktopová aplikace napsaná v jazyce Tcl/Tk. Skládá se ze dvou částí, jménem Bonito 1 se označuje jeho klientská část. Ta komunikuje se serverovou částí ManateeSRV přes TCP/IP. K přenosu dat aplikace implementuje svůj vlastní stavový textový protokol. ManateeSRV se stará o autentizaci a autorizaci a má přístup k databázi. K samotné práci s korpusy využívá knihovnu Manatee, komunikace s ním probíhá přes standardní vstup a výstup.

Výhody a nevýhody vyplývají z podstaty desktopové aplikace. Uživatelské rozhraní je interaktivní, s rychlou odezvou. Bonito 1 má také širokou funkcionalitu, včetně podpory statistik, vytváření subkorpusů, nebo dokonce grafické tvorby dotazů. Nevýhodou je, že aplikace není platformově nezávislá (ale existuje verze pro OS Windows, Linux i Mac OS). Aplikaci je také nutno na klientském počítači nejdříve nainstalovat, z čehož plynou problémy s případným zaváděním nových verzí. Také GUI postupem času zastaralo a jeho vzhled neodpovídá současným požadavkům.

**Bonito 2:** z výše zmíněných důvodů vznikala nová verze klienta - Bonito 2. Bonito 2 je webová aplikace napsaná v jazyce Python. Na klientském počítači je tedy potřeba jen webový prohlížeč. Na straně serveru běží CGI skripty, které obsluhují příchozí HTTP požadavky a následně vygenerují HTTP odpověď - čistou html stránku bez jakékoli klientské logiky. Ke komunikaci s Manatee využívá Bonito 2 rozhraní vygenerované nástrojem SWIG. Struktura aplikace je naznačena na obrázku 2.2. Právě neinteraktivní GUI (při každé akci se musí načíst nová stránka) spolu s menší funkcionalitou způsobily, že uživatelé raději nadále využívají starší Bonito 1.

**Kondor:** neúspěch Bonita 2 dal podnět ke vzniku nového manažeru s pracovním názvem Kondor. Jedná se také o webovou aplikaci. Jako programovací jazyk byl zvolen jazyk Java4, který je standardem pro tvorbu podnikových aplikací. Na rozdíl od svého předchůdce však poskytuje interaktivní GUI podobné desktopové aplikaci, což je umožněno použitím technologie AJAX. Aplikace Kondor je stále ve vývoji, její základní komponenty včetně dotazování a správy uživatelů jsou již implementovány. Chybí však podpora pro výpočet statistických funkcí.

\*AJAX (Asynchronous JavaScript and XML) je obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich opětovného načtení. Z JavaScriptového kódu se na pozadí zašle HTTP dotaz na server a jeho odpověď je opět zpracována JavaScriptem. Na rozdíl od klasických webových aplikací poskytují AJAXové aplikace uživatelsky příjemnější prostředí, ale vyžadují použití moderních webových prohlížečů.

## AUTOMATICKÉ NÁSTROJE PRO ZNAČKOVÁNÍ KORPUSU

**Morfologické analyzátory** = nejznámější morfologické analyzátory (značkovací programy - taggers pro angličtinu) zpracovávají data v korpusu tak, že každému slovnímu tvaru přiřadí jeho gramatickou značku (tag), tj. obvykle symbol slovního druhu (může jich být i víc). Obvykle se značkují vybrané části korpusu v rozsahu do 10 mil. slovních tvarů; vzniklé soubory jsou zhruba třikrát až čtyřikrát větší než původní, což znamená, že při jejich dalším zpracování vznikají časové problémy.

**Probabilistický analyzátor CLAWS** (autor R.Garside z Lancasteru): má vysokou úspěšnost, dosahuje jen 1.7% chyb. Celkově je CLAWS hybridní (vedle stochastického přístupu obsahuje i jednoduchá syntaktická pravidla) a pracuje s anotovaným lexikonem, jehož součástí je i seznam základních anglických idiomů. Značkování se provádí v několika fázích, používá se rovněž Viterbiho algoritmu (zpracovává pravděpodobnosti přechodu mezi větnými složkami). Probabilistický přístup je motivován tím, že je blízký psychologii člověka.

**Analyzátor vytvořený J.Clearem v birminghamském COBUILDU:** využívá pravděpodobnostního přístupu, je velmi robustní a jeho míra úspěšnosti je 95% - autor ji pokládá za dostačující.

**Helsinský analyzátor:** založen na tzv. constraint grammars a je 60krát rychlejší než ostatní (předpokládá ale užití dvoustupňového morfologického analyzátoru Kimmo od Koskenniemiho) - je zatím ze všech zjevně nejúspěšnější, pokud jde o zvládnutí více jazyků (dosud dovede pracovat s 5 jazyky).

**Analyzátor D.Cuttinga et al.** (je v public domain a dostupný v Internetu):

Užívá skrytého Markovova modelu, je jazykově nezávislý, učí se od počátku na menších vzorcích, pracuje s vahami pravděpodobnostního výskytu, pracuje iterativně a ve fázi učení počítá s 18% předem označovaného textu.

Taggit

Morfologické analyzátory Ajka a Majka

## DALŠÍ NÁSTROJE

### WORD SKETCH ENGINE

- webové rozhraní propojené s korpusovým manažerem Bonito 2
- slouží k vyhledávání ve velkých jazykových korpusech, nabízí přístup k 66 jazykovým korpusům v různých grafických soustavách (azbuka, hebrejské písmo) a 42 jazycích
- vznikl díky spolupráci CZPJ a společnosti Lexicography MasterClass ve VB
- slouží k rychlému vyhledání a rozbrazení kontextu zadaného slova (substantiva, adjektiva, slovesa), přičemž se řídí gramatickými charakteristikami, jimiž jsou korpusy doplněny
- výsledkem po zadání slovesného lematu je seznam kontextových slov, které se k zadanému lematu váží, rozříděných do jednotlivých tabulek.
- Sketch Engine je korpusový manažer, který nabízí uživateli kromě standardních funkcí jako je konkordance, třídění a filtrování, také funkci Word Sketch, funkci Tezaurus a Sketch Difference. Sketch Engine zobrazuje výsledky hledání i se statistickými parametry. Konkordance je zobrazení zadaného slova v jeho kontextovém okolí. U každého konkordančního řádku je také uveden textový zdroj, z něhož byla slova a slovní spojení převzata.
- Word Sketch je funkce, díky níž se zobrazuje seznam kolokací zvoleného slova. Seznam kolokací je rozdělen podle syntaktických vztahů do syntaktických kategorií.



## PŘEDZPRACOVÁNÍ KORPUSOVÝCH DAT ZÍSKANÝCH Z INTERNETU

### **JUSTTEXT**

- nástroj, který odstraní nežádoucí obsah
- odstraňuje netextové části webových stránek (html značky, styly, poznámky; negramatické věty: navigace, reklamy, tabulky, příliš krátké úseky)

### **ONION**

- odstranění podobných odstavců
- odstraní duplicitní texty

### **UNITOK**

- tokenizace

### **CHARED**

- detekce znakové sady dokumentu
- rozpoznává kódování textu

## ***DALŠÍ POJMY A OTÁZKY SOUVISEJÍCÍ S KOPRUSY***

### **Využití korpusů při tvorbě slovníků**

Dříve sběr dat založen na lístkových katalogích sestavovaných z příkladů lexikálních jednotek. Korpusy umožňují vyhledávání zadané lexikální jednotky během několika vteřin. To umožňuje rychlé obohacování a rozšiřování slovníků; definice a výklad mohou být co nejbohatší až kompletní; příklady se mohou rychle řadit do smysluplných podskupin; při tvorbě slovníků se používá otevřených monitorovacích korpusů; ohromný význam u terminologických slovníků; souvýskyt + možnost jeho srovnání -> přesná definice termínu. Velké využití u frekvenčních slovníků, různých učebnic, jazykovědných studií, příruček. Na základě PMK vznikl frekvenční slovník pražské mluvené češtiny.

### **KWIC**

KWIC - Keyword in context je konkrétní vyhledané spojení, které odpovídá zadanému dotazu (CQL). Pokud máme např. dotaz [word="auto"][] [word="koně"] (tři slova, z nichž první je „auto“ a poslední je „koně“), pak KWIC může být třeba „auto přejede koně“ nebo „auto, koně“.

### **Konkordanční seznamy a jejich využití**

Konkordance je množina klíčových slov nebo slovních spojení spolu s jejich kontextem vyhledaných jako odpověď na uživatelem zadaný dotaz. Konkordance je zpravidla zobrazována v tabulce se třemi sloupci. Uprostřed jsou vyhledaná spojení, odpovídající dotazu.

Konkordanční seznam je výsledek dotazu na korpus; zobrazuje se ve formátu KWIC, kdy jsou hledaná slova se svými kontexty zobrazena přehledně pod sebou.

### **Frekvenční seznamy a jejich využití**

Frekvenční distribuce“ slouží ke spočítání frekvence slov (lemmat), morfologických značek a jejich poskupností na zadaných pozicích (počítáno od KWIC). Jako výsledek tedy zobrazí tabulku se slovy (pozičními či strukturními atributy), která se nejčastěji vyskytují na daných pozicích.

### **K čemu může sloužit kvantitativní analýza dat získaných z velkých jazykových korpusů?**

NLP: stochastická desambiguace, automatická analýza založená na statistických metodách

Frekvenční seznamy, výzkum kolokací – lexikografie

### **Kolokace a korpusová lingvistika**

Kolokace jsou slova (nebo i jiné poziční atributy), která se statisticky významně často vyskytují v korpusu pohromadě. Statistika „Kolokace“ nám tedy zobrazuje, jaká slova se nejčastěji vyskytují v blízkosti KWIC spolu s hodnotami specializovaných kolokačních funkcí

T-score - míra kontrastu: Vychází ze statistické metody testování hypotéz pomocí tzv. t-testu.

V případě kolokací testujeme, zda zjištěné počty výskytů jednotlivých slov a jejich dvojic odpovídají náhodnému rozložení slov v korpusu. Čím vyšší je hodnota t-score, tím méně je pravděpodobné, že jde o náhodné rozložení slov a a naopak tím pravděpodobnější je, že jde o pevnější, ustálenější kombinace slov, tj. o kolokace.

Mi-score

### **Jaký je rozdíl mezi kognitivně plausibilním a kognitivně neplausibilním systémem?**

Kognitivně plausibilní systém usiluje o vytvoření poznávacího (kognitivního) modelu, pro nějž je relevantní, jakým způsobem člověk řeší nějaký úkol pomocí inteligence, a používá jej jako bázi pro stroj, který má tento problém (úkol) inteligentně řešit. Takový systém často používá úplnou sadu pravidel, které explicitně formulují znalosti, které člověk implicitně používá, ve formě tzv. báze znalostí (knowledge base).

Naopak systémy, které rezignují na kognitivní plausibilitu a jsou tedy kognitivně neplausibilní, se prostě snaží vytvořit model inteligentního chování, aniž by se přihlíželo k tomu, zda systém pracuje stejným způsobem jako člověk (inteligentně). Tyto systémy často používají surová kvantitativní data k vytvoření statistického modelu napodobujícího lidské chování.

### **MRF (machine readable form)**

soubor počítačově čitelných textů, složený ze souvislých textových úseků vybraných dle jistých pravidel tak, aby reprezentovaly jazyk jako celek v celé jeho pestrosti -> obsahuje standardní reference

anotace - ty zahrnují metatextové informace a interpretace jednotek, z nichž je text složen

### **OCR (optical character recognition)**

optické rozpoznávání znaků je metoda, která pomocí scanneru umožňuje digitalizaci tištěných textů, s nimiž pak lze pracovat jako s normálním počítačovým textem. Počítačový program převádí obraz buď automaticky, nebo se musí naučit rozpoznávat znaky. Převedený text je téměř vždy v závislosti na kvalitě předlohy třeba podrobit důkladné korektuře, protože OCR program nerozezná všechna písmena správně.

### TEI (Text encoding initiative)

\*1986. Aktivita sponzorovaná hlavními vědecky orientovanými asociacemi zabývajícími se využitím počítačů v humanitních vědách. ACL (Association for Computational Linguistics), ALLC (the Association for Literary and Linguistic Computing), ACH (the Association for Computers and Humanities). Je sponzorovaná EU a americkou vládou;

Cíl: vytvoření standardní implementace pro operace s počítačově čitelnými texty. Návrhla společný kódovací a značkovací metajazyk SGML – přijat pro svou jednoduchost, jasnost, formální přísnost a mezinárodní uznání; vydala doporučení pro společný výměnný formát, zásady kódování, znakové sady

### EAGLES

dozorčí skupina založená EU, jejím úkolem je sledovat a pomáhat různým evropským iniciativám

jde o vytvoření systému značek, který by na jedné straně zachytil všechny zvláštnosti, potřeby, specifika všech evropských jazyků - na straně druhé zachoval jednotu systému

### DDL (Data Driven Learning)

Výuková metoda studentů, která se zakládá na vyhledávání v korpusu. Otec Tim Johns (univerzita v Birminghamu). Vytvořit takový postup výuky, který byl založený na induktivní metodě, tedy na aktivním vyhledávání jedné jazykové jednotky v korpusu. Studenti tak pomocí korpusu shromáždí data, které jim dávají příležitost objevit příklady užití jazyka. Z nich poté zobecní pravidlo. DDL je možné využít při skupinové nebo individuální výuce i mimo tradiční výuku, např. ve formě e-learningu nebo jako domácí cvičení.

Student je tedy naveden k tomu, aby sám na základě materiálu získaného z korpusu tvořil hypotézy o různých významech lexikálních, gramatických. Rozvíjí deduktivní přístup, což napomáhá dalšímu studiu jazyků.

Hlavní výhodou využití korpusu pro výuku jazyků je práce se skutečným materiálem, reálnými větami a slovními spojeními, které nejsou upravené pro didaktické účely.

### DTD (Document Type Definition)

Definice typu dokumentu; jazyk pro popis struktury XML, případně SGML dokumentu. Omezuje množinu přípustných dokumentů spadajících do daného typu nebo třídy. DTD tak například vymezuje jazyky HTML a XHTML. Struktura třídy nebo typu dokumentu je v DTD popsána pomocí popisu jednotlivých značek (nebo též elementů) a atributů. Popisuje, jak mohou být značky navzájem uspořádány a vnořeny. Vymezuje atributy pro každou značku a typ těchto atributů. Poměrně starý a málo expresivní jazyk. Formální reprezentace, informuje uživatele nebo program o tom, které elementy text obsahuje, jak jsou tyto elementy kombinovány a obsahuje také sadu deklarací entit. Používá se programem SGML parser – kontroluje, zda je text otagován ve formátu kompatibilním s TEI

```
<!ELEMENT clovek (jmeno, adresa*)>
<!ELEMENT jmeno (#PCDATA)>
<!ELEMENT adresa (ulice?, cislo?, mesto)>
<!ELEMENT ulice (#PCDATA)>
<!ELEMENT cislo (#PCDATA)>
<!ELEMENT mesto (#PCDATA)>
```

### SGML (Standard Generalized Markup Language)

Standardní jazyk určený k formálnímu popisu struktury dokumentů. Zcela otevřený standard nezávislý na platformách, výrobcích nebo aplikacích. Univerzální značkovací metajazyk, který umožňuje definovat značkovací jazyky jako své vlastní podmnožiny (umožňuje definování dalších jazyků). **Výhody:** jednoduchý formát, otevřenost (nehrozí znehodnocení investic v důsledku nutnosti přechodu na nové verze), zajímá nás obsah, nikoli forma dokumentů (formu lze určit až v závislosti na požadovaném výstupu). Otaggovaný text = výhody pro prohledávání dokumentů.

**Nevýhody:** není lehké psát parsery (programy, které provádí kontrolu SGML dokument, kontrolují, zda použití elementů v dokumentu odpovídá jejich definici v DTD); WWW nepodporuje SGML; komplikovaný jazyk, obtížně se integruje do jednotlivých aplikací

### PDT (Prague Dependency Treebank; Pražský závislostní korpus)

Pražský závislostní korpus (PDT) je probíhající projekt pro ruční anotaci velkého množství českých textů bohatou lingvistickou informací, sahající od morfologie přes syntax až po sémantiku/pragmatiku a ještě dále.

Představuje druhý největší, ručně označovaný korpus na světě (hned za anglickým Penn Treebank (1992))

Corpora query language (CQL) je dotazovací jazyk, pomocí něhož jsou tvořeny jednotlivé dotazy na korpus. S jeho pomocí se v korpusu vyhledávají slova dle jejich atributů (většinou word, lemma, tag). Jazyk dovoluje i kombinování podmínek pro vyhledávání slova. Při vytváření dotazu se používá regulárních výrazů, které umožňují poměrně velkou flexibilitu v parametrech hledání slova.

Každá pozice (slovo) je ohraničena hranatými závorkami, v nichž se nachází specifikace jednoho nebo více atributů.

[atribut="value"] kde value je regulární výraz

## Vyhledávání podle tvaru slova nebo slovního spojení:

[word="holubí"]

Korpus najde všechny výskyty slova holubí

K vyhledávání více než jednoho slova použít pajpu

[word = „admin|amidst“]

[word = „struggle|battle|fight“]

## Vyhledávání podle lemmatu (základního slovníkového tvaru):

[lemma="holubí"]

Korpus najde: holubího, holubí, holubích

## Vyhledávání podle morfologické značky

Každá značka obsahuje 15 pozic, každá pozice odpovídá jedné morfologické kategorii

[tag="....4.\*"] hledáme-li všechny akuzativy

[tag="ACY5.\*"] hledáme-li všechna adjektiva ve jmenném tvaru mužského rodu životného i neživotného v singuláru

## Vyhledávání spojení:

[lemma="mořsk."][word="ryba|živočich"]

Korpus najde: mořská ryba, mořský živočich

## Kombinace elementů

Výraz vybere všechny tvary slovesa chodit následované dvěma až čtyřmi slovy (započítává se i interpunkce) a zakončené podstatným jménem ve třetím nebo sedmém pádě. Pro zkrácení zápisu se dále využívá značky (?) ekvivalentní zápisu {0,1} a [pos="A"] ekvivalentní [tag="A.\*"].

[lemma="chodit"][] {2,4}[tag="N...3.\*"|tag="N...7.\*"]

Např. pila je buď sloveso nebo podstatné jméno. Dotaz, chceme-li najít pila jako sloveso:

[lemma = „pila“ & tag = „V.“]

## Řetězce elementů:

Která předložka následuje za pila? [lemma = „pila“] [tag „PRP“]

Která předložka následuje za podstatným jménem pila? [lemma = „pila“ & tag = „N.“] [tag „PRP“]

## Vložení slova mezi řetězce:

[lemma=""] [] [lemma = „approach“]

Číslo v hranatých závorkách {x} označuje, kolik nechat volných slov (počet period) mezi slovy učení a nuda: [lemma="učení"] [] {3}[lemma = „nuda“]

Závorky {1,3} určují rozsah – od jedné do tří. Tento dotaz hledá jedno, dvě nebo tři slova mezi slovy patra a koupit:

[lemma="parta"] [] {1,3}[lemma = „koupit“]

### Vyloučení mezi řetězci:

[lemma = „koupit“] [word != „alkohol“]

### Vyhledávání podle interpunkce:

[word=“\,”][word = „which“]

### Najít pozici v korpusu (číslo tokenu):

Najdi pozici 100 a 210: [#100|#210]

### REGULÁRNÍ VÝRAZY, KTERÉ POUŽÍVÁ CQL

- \* = libovolný počet opakování předchozího znaku
- + = libovolný počet opakování předchozího znaku > 0
- ? = žádný nebo jeden výskyt předchozího znaku/výrazu
- {n} = nkrát
- {n, m} = n-mkrát, {n,} = n-∞krát
- hladové × s ? – minimální počet znaků, které je třeba zachytit, aby došlo ke shodě s regulárním výrazem
- . = libovolný znak
- ^ = začátek řetězce / negace (někdy !)
- [^a-z0-9] = libovolný znak kromě znaků v rozsahu a–z, 0–9
- \$ = konec řetězce
- | = logické or: nebo (disjunkce)
- [] = výběr jednoho znaku z výčtu
- ()
- \

### Příklady:

a+	sekvence písmen a (1 a více znaků)
a*	sekvence písmen a (0 a více znaků)
o?kov	okov či kov
tel(efon)?	tel či telefon
telef(on ax)	telefon či telefax
[0-9]  [1-9][0-9]	čísla 0 až 99
\d{2}	sekvence dvou číslic desítkové soustavy (00, 01, ..., 98, 99)
[0-9a-fA-F]  [1-9a-fA-F][0-9a-fA-F]+	hexadecimální čísla
(19 20)\d{2}	letopočty 1900-2099
\d{2,6}	sekvence dvou až šesti číslic
[^ „,“]+	neprázdná sekvence znaků, mezi nimiž nesmí být mezera ( ), čárka (,) či tečka (.)
^P.*	řetězec, který začíná písmenem P, za nímž následuje libovolný (i nulový) počet libovolných znaků
\d+0\$	řetězec, který končí znakem 0 (nula), kterému předchází minimálně jedna číslice
a+b	ab, aab, aaab atd.
a\+b	a+b
h?rozin(k c).*	hrozinka, rozinka + v dalších pádech a v plurálu
.+nést	donést, odnést, přinést, zanést, vynést, povznést, ...
.+in[kg]	mítink, brífing, leasing, sing, ...
[bB]ože	bože, Bože
les.*	les, lesk, lesklý, lesní, lest, ...