VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

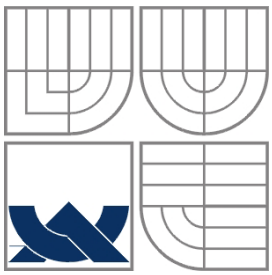
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYNTAKTICKÝ ANALYZÁTOR PRO ČESKÝ JAZYK

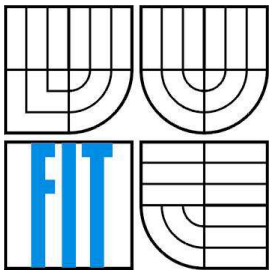
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. VOJTĚCH BENEŠ
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYNTAKTICKÝ ANALYZÁTOR PRO ČESKÝ JAZYK

SYNTACTIC ANALYZER FOR CZECH LANGUAGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. VOJTĚCH BENEŠ
AUTHOR

VEDOUCÍ PRÁCE — ING. JAN KOUŘIL
SUPERVISOR

BRNO 2014

Abstrakt

Diplomová práce popisuje teoretický návrh a vytvoření syntaktického analyzátoru pro český jazyk pracujícího s frázovým přístupem ke stavbě věty. Využívaná frázová syntaxe je založena na slovních druzích, které jsou sdružovány do větších slovních celků – frází. Implementovaný program pracuje s manuálně sestaveným anotovaným vzorkem dat (korpusem češtiny), na základě kterého za běhu vytvoří pravděpodobnostní bezkontextovou gramatiku (strojové učení). Syntaktický analyzátor, jehož jádrem je rozšířený CKY algoritmus, poté pro zadanou českou větu rozhodne, zda-li patří do jazyka generovaného vytvořenou gramatikou, a v kladném případě vrátí nejpravděpodobnější derivační strom této věty. Tento výsledek je následně porovnán s očekávaným řešením, čímž je vyhodnocena úspěšnost syntaktické analýzy.

Abstract

Master's thesis describes theoretical basics, solution design, and implementation of constituency (phrasal) parser for Czech language, which is based on a part of speech association into phrases. Created program works with manually built and annotated Czech sample corpus to generate probabilistic context free grammar within runtime machine learning. Parser implementation, based on extended CKY algorithm, then for the input Czech sentence decides if the sentence can be generated by the created grammar and for the positive cases constructs the most probable derivation tree. This result is then compared with the expected parse to evaluate constituency parser success rate.

Klíčová slova

Zpracování přirozeného jazyka, pravděpodobnostní bezkontextová gramatika, PBKG, český jazyk, korpus, frázový syntax, rozšířený CKY algoritmus, syntaktický analyzátor

Keywords

Natural language processing, NLP, probabilistic context free grammar, PCFG, Czech language, corpus, phrasal syntax, extended CKY algorithm, constituency parsing, parser

Citace

Beneš Vojtěch: Syntaktický analyzátor pro český jazyk, diplomová práce, Brno, FIT VUT v Brně, 2014

Syntaktický analyzátor pro český jazyk

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kouřila. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Vojtěch Beneš
28. 5. 2014

Poděkování

Rád bych poděkoval svému vedoucímu práce, Ing. Janu Kouřilovi, za odborné konzultace, poskytnutí materiálů k vypracování této diplomové práce a neméně také za trpělivost, kterou se mnou při vypracování práce měl.

Obsah

1 Úvod

Zpracování přirozeného jazyka je zejména v posledním době čím dál tím více se rozvíjející obor, kterého využívá, i když třeba nevědomky, stále větší množství běžné lidské populace. Dnešní chytré telefony zvládají stále náročnější aplikace tohoto typu - od již nějakou dobu známého hlasového vytáčení telefonního čísla, přes spouštění aplikací a zjednodušeného vyhledávání na internetu, až po zapisování poznámek. Jedná se tedy o interakci mezi člověkem a strojem a zejména o strojové zpracování mluveného slova, jehož využití lze nalézt v různých automatizovaných činnostech – titulkování, přepisování hovorů, přednášek, nahrávek či třeba diktování textu.

Kvalita zpracování přirozeného jazyka pomocí existujících systémů se samozřejmě liší jazyk od jazyka. Angličtina, jako jeden z nejpoužívanějších a nejrozšířenějších jazyků, je v tomto směru podstatně dále, než čeština, která stojí relativně na začátku. Hlavním rozdílem mezi těmito dvěma jazyky je z hlediska strojového zpracování jejich odlišná struktura. Angličtina, patřící mezi germánské jazyky, se vyznačuje pevným slovosledem, který má poměrně striktní pravidla řazení jednotlivých větných členů, což slouží k vyjádření vztahů mezi nimi. Naproti tomu má čeština volný slovosled, kde není pořadí větných členů pevně dáno a jejich vztah závisí hlavně na zapojení do kontextu, což značně zvyšuje obtížnost strojové zpracovatelnosti. Volný slovosled je charakteristický zejména pro slovanské a ugrofinské jazyky.

Cílem této diplomové práce je vytvoření syntaktického analyzátoru pro český jazyk pracujícího s frázovým přístupem ke stavbě věty. Využívaná frázová syntaxe je založena na slovních druzích, které jsou sdružovány do větších slovních celků – frází. Implementovaný program pracuje s manuálně sestaveným anotovaným vzorkem dat (korpusem češtiny), na základě kterého za běhu vytvoří pravděpodobnostní bezkontextovou gramatiku (strojové učení). Syntaktický analyzátor, jehož jádrem je rozšířený CKY algoritmus, poté pro zadanou českou větu rozhodne, zda-li patří do jazyka generovaného vytvořenou gramatikou, a v kladném případě vrátí nejpravděpodobnější derivační strom této věty. Tento výsledek je následně porovnán s očekávaným řešením, čímž je vyhodnocena úspěšnost syntaktické analýzy.

Následující dvě kapitoly obsahují základní pojmy a teorii jak z oboru teoretické informatiky (kapitola 2), tak ze zpracování přirozeného jazyka potřebné k uvedení a pochopení řešené problematiky (kapitola 3). Kapitola 4 rozebírá různé druhy algoritmů syntaktické analýzy a jejich varianty, zejména potom CKY algoritmus, který rozšíří o práci s pravděpodobnostní bezkontextovou gramatikou. Její součástí je také vysvětlení způsobu vyhodnocování úspěšnosti syntaktických analyzátorů. Kapitola 5 poté na příkladech detailně a prakticky demonstruje výpočet pravděpodobnosti věty či průběh algoritmu syntaktické analýzy i s jeho následným vyhodnocením. Kapitola 6 se již zabývá požadavky na vytvoření aplikace, jejím návrhem a vlastní implementací. Obsahuje také popis testování programu a jeho výsledky. V závěru je práce zhodnocena a dále je konzultován další možný vývoj projektu.

V rámci semestrálního projektu, předcházejícímu této diplomové práci, byly zpracovány teoretické základy z oboru teoretické informatiky a zpracování přirozeného jazyka (kapitoly 2 a 3), které byly později podle potřeby rozšířeny. Dále také obsahoval praktičtější seznámení s korpusovými daty a pravděpodobnostními bezkontextovými gramatikami (podkapitoly 5.1 a 5.2).

2 Pojmy z teorie informatiky

V této kapitole se seznámíme s důležitými pojmy vhodnými pro uvedení do řešené problematiky. Předpokládáme zde již předchozí kontakt čtenáře s úvodem do teorie grafů a s oborem teoretické informatiky, příp. teorie překladačů, a znalost základních pojmů z tohoto oboru (symbol, abeceda, řetězec, věta, jazyk, derivace apod.). Od gramatiky a bezkontextové gramatiky se dostaneme k definici pravděpodobnostní bezkontextové gramatiky, dále zavedeme pojmy jako derivační strom či Chomského normální forma a uvedeme také některé algoritmy pro transformace bezkontextových gramatik.

Tato kapitola je volně převzata z [1] s výjimkou části týkající se samotné pravděpodobnostní bezkontextové gramatiky, která byla volně převzata z [2] a [3].

2.1 Gramatika

Gramatika je využívána jako nejznámější a nejpoužívanější prostředek pro reprezentaci jazyka, který je na rozdíl od jiných způsobů reprezentace jazyka (výčet všech vět jazyka, obvyklé matematické prostředky) použitelný nejen pro rozsáhlé konečné jazyky, ale i pro jazyky nekonečné.

Gramatika splňuje základní požadavek konečnosti reprezentace, pro kterou využívá dvě konečné disjunktní abecedy:

- množinu N nonterminálních symbolů,
- množinu T terminálních symbolů.

Nonterminální symboly, krátce *nonterminály* či *neterminály*, plní roli pomocných proměnných označujících určité syntaktické celky – syntaktické kategorie. Množina *terminálních symbolů*, zkráceně *terminálů*, je identická s abecedou jazyka. Sjednocení obou množin, tj. $N \cup T$, nazýváme *slovníkem gramatiky*.

Definice 2.1 *Gramatika* G je čtveřice $G = (N, T, R, S)$, kde

- N je konečná množina nonterminálních symbolů (abeceda nonterminálů),
- T je konečná množina terminálních symbolů (abeceda terminálů), $N \cap T = \varnothing$,
- R je konečná podmnožina kartézského součinu $(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ (množina přepisovacích pravidel),
- $S \in N$ je výchozí (také počáteční) symbol gramatiky.

Prvek (α, β) množiny R nazýváme *přepisovacím pravidlem* (krátce *pravidlem*), zapisujeme jej ve tvaru $\alpha \to \beta$. Řetězec α, respektive β, nazýváme *levou*, respektive *pravou*, *stranou* přepisovacího pravidla.

2.2 Bezkontextová gramatika

Bezkontextová gramatika (context free grammar, CFG) obsahuje pravidla ve tvaru:

$$A \to \gamma, A \in N, \gamma \in (N \cup T)^*.$$

Bezkontextové gramatiky dostaly své jméno podle toho, že substituci pravé strany γ pravidla za nonterminál A lze provádět bez ohledu na kontext, ve kterém je nonterminál A uložen (na rozdíl

od kontextových gramatik). Bezkontextové gramatiky se také nazývají gramatiky *typu 2*, a to podle zařazení v Chomského klasifikaci gramatik[1].

Bezkontextové gramatiky (opět na rozdíl od kontextových) smí obsahovat pravidla ve tvaru $A \to \varepsilon$, tzv. e*psilon pravidla*. Gramatiku lze ovšem transformovat tak, že obsahuje nejvýše jedno pravidlo s prázdným řetězcem na pravé straně ve tvaru $S \to \varepsilon$. V takovém případě se nesmí výchozí symbol S objevit na žádné pravé straně přepisovacího pravidla gramatiky.

2.3 Pravděpodobnostní bezkontextová gramatika

V této části práce uvedeme pouze samotnou definici *pravděpodobnostní bezkontextové gramatiky* (probabilistic context free grammar, PCFG), její význam a vztah ke zpracování přirozeného jazyka rozebereme společně s demonstračními příklady v dalších částech tohoto textu.

Definice 2.2 *Pravděpodobnostní bezkontextová gramatika* G je pětice $G = (N, T, R, S, P)$, kde

- N je konečná množina nonterminálních symbolů,
- T je konečná množina terminálních symbolů, $N \cap T = \varnothing$,
- R je konečná podmnožina přepisovacích pravidel ve tvaru $A \to \gamma$, kde $A \in N, \gamma \in (N \cup T)^*$,
- $S \in N$ je počáteční symbol gramatiky.
- P je funkce pravděpodobnosti, pro kterou platí:
  - $P : R \to [0,1]$,
  - $\forall A \in N, \sum_{A \to \gamma \in R} P(A \to \gamma) = 1$.

V pravděpodobnostní bezkontextové gramatice tedy pro každé přepisovací pravidlo existuje pravděpodobnost aplikace tohoto pravidla při vytváření derivačního stromu.

2.4 Derivační strom

Derivační nebo *syntaktický strom* je kořenový, uzlově ohodnocený strom, který slouží pro grafické vyjádření struktury věty (její derivace).

Definice 2.3 Nechť $G = (N, T, R, S)$ je gramatika. Řetězec $\alpha \in (N \cup T)^*$ nazýváme *větnou formou*, jestliže platí $S \Rightarrow^* \alpha$, tj řetězec α je generovatelný z výchozího symbolu S. Větná forma, který obsahuje pouze terminální symboly, se nazývá *věta.*

Definice 2.4 Nechť δ je věta nebo větná forma generovaná v gramatice $G = (N, T, R, S)$ a nechť $S = \nu_0 \Rightarrow \nu_1 \Rightarrow \nu_2 \Rightarrow ... \Rightarrow \nu_k = \delta$ její derivace v G. *Derivační strom* příslušející této derivaci je strom s těmito vlastnostmi:

(1) Uzly derivačního stromu jsou (ohodnoceny) symboly z množiny $N \cup T$; kořen stromu je označen počátečním symbolem S.

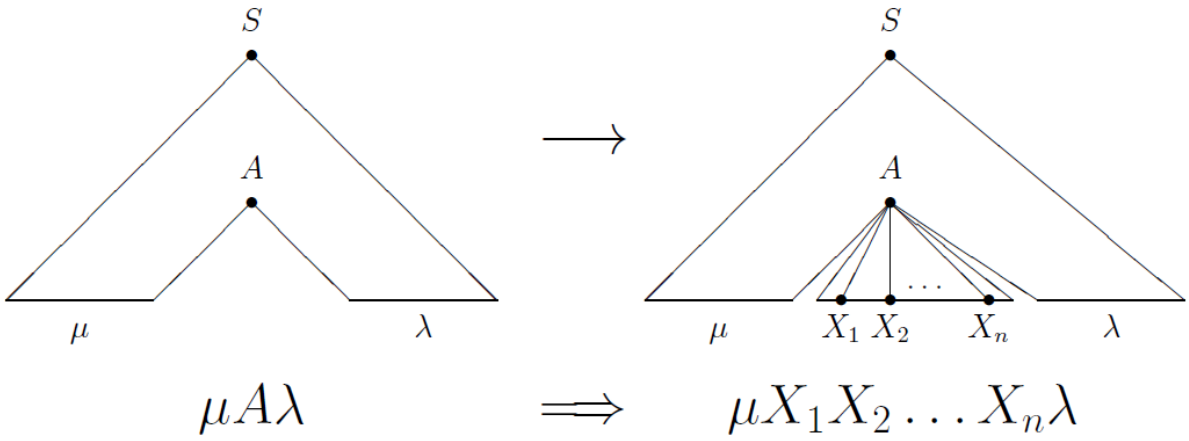
[1] http://cs.wikipedia.org/wiki/Chomského_hierarchie

(2) Přímé derivaci $\nu_{i-1} \Rightarrow \nu_1$, $i = 0,1,...,k$, kde

- $\nu_{i-1} = \mu A \lambda$, $\mu, \lambda \in (N \cup T)^*$, $A \in N$,
- $\nu_i = \mu \alpha \lambda$ a
- $A \to \alpha$, $\alpha = X_1 ... X_n$ je pravidlo z R,

odpovídá právě n hran $(A, X_j), j = 1,...,n$ vycházejících z uzlu A, jež jsou uspořádány zleva doprava v pořadí $(A, X_1), (A, X_2), ..., (A, X_n)$.

(3) Označení koncových uzlů derivačního stromu vytváří zleva doprava větnou formu nebo větu δ (plyne z (1) a (2)).



Obrázek 2.1: Konstrukce derivačního stromu. Převzato z [1].

Při konstrukci derivačního stromu k dané derivaci opakovaně aplikujeme bod (2) z definice 2.4. Tuto aplikaci ilustruje obrázek 2.1.

2.5 Víceznačnost gramatik

Definice 2.5 Nechť G je gramatika. Říkáme, že věta w generovaná gramatikou G je *víceznačná* (někdy také uváděno jako mnohoznačná, v angličtině ambiguous), existují-li alespoň dva různé derivační stromy s koncovými uzly tvořící větu w. Gramatika G je víceznačná, jestliže generuje alespoň jednu víceznačnou větu. V opačném případě mluvíme o jednoznačné gramatice.

Je také vhodné si povšimnout, že definujeme víceznačnou gramatiku, nikoli víceznačný jazyk. V mnoha případech lze vhodnými transformacemi víceznačné gramatiky odstranit víceznačnost vět, aniž se samozřejmě změní jazyk generovaný získanou jednoznačnou gramatikou. Existují však jazyky, které nelze generovat jednoznačnou gramatikou. Takové jazyky jsou pak nazývány jazyky s *inherentní víceznačností*.

Typickým příkladem víceznačné gramatiky uváděným například ve [12] je bezkontextová gramatika reprezentující syntaktickou konstrukci mnoha programovacích jazyků **`if then else`** uvedená v tabulce 2.1. Počáteční symbol gramatiky S je nonterminál, ostatní symboly jsou terminály.

1: $S \to$ **`if`** b **`then`** S **`else`** S
2: $S \to$ **`if`** b **`then`** S
3: $S \to a$

Tabulka 2.1: Víceznačná bezkontextová gramatika. Převzato z [12].

Pro vstupní řetězec uvedený níže potom gramatika vygeneruje dva různé derivační stromy, protože nelze jednoznačně určit, ke které konstrukci **`if then`** se váže jediná konstrukce **`else`.**

`if` b **`then if`** b **`then`** a **`else`** a

2.6 Chomského normální forma

Definice 2.6 Gramatika $G = (N, T, R, S)$ je v *Chomského normální formě (CNF)*, jestliže každé pravidlo z R má jeden z těchto tvarů:

- $A \to BC, A, B, C \in N$ nebo
- $A \to a, a \in T$ nebo
- je-li $\varepsilon \in L(G)$, pak $S \to \varepsilon$ je pravidlo z R a výchozí symbol S se neobjev na pravé straně žádného pravidla.

Chomského normální forma gramatiky je prostředkem zjednodušujícím tvar reprezentace bezkontextového jazyka. Transformace do této formy, jejíž algoritmus je popsán níže, nijak nemění vyjadřovací schopnost bezkontextové gramatiky – gramatika generuje stejný jazyk ovšem s odlišnými derivačními stromy. Binarizace přepisovacích pravidel je však rozhodující pro kubickou časovou složitost překladu založeného na bezkontextové gramatice.

Algoritmus 2.1 Převod do Chomského normální formy.
Vstup: Vlastní gramatika $G = (N, T, R, S)$ bez jednoduchých pravidel.
Výstup: Gramatika $G' = (N', T, R', S')$ v CNF taková, že $L(G) = L(G')$.
Metoda: Z gramatiky G získáme ekvivalentní gramatiku G' v CNF takto:

(1) Množina pravidel R' obsahuje všechna pravidla tvaru $A \to a$ z R.
(2) Množina pravidel R' obsahuje všechna pravidla tvaru $A \to BC$ z R.
(3) Je-li pravidlo $S \to \varepsilon$ v R, pak $S \to \varepsilon$ je také v R'.
(4) Pro každé pravidlo tvaru $A \to X_1 ... X_k$, kde $k > 2$ z R přidej k R' tuto množinu pravidel. Symbolem X_i' značíme nonterminál X_i, je-li $X_i \in N$, nebo nový nonterminál, je-li $X_i \in T$:

$$A \to X_1' \langle X_2 ... X_k \rangle$$
$$\langle X_2 ... X_k \rangle \to X_2' \langle X_3 ... X_k \rangle$$
$$\vdots$$
$$\langle X_{k-1} X_k \rangle \to X_{k-1}' X_k',$$

kde každý symbol $\langle X_i ... X_k \rangle$ značí nový nonterminální symbol.

(5) Pro každé pravidlo tvaru $A \to X_1 X_2$, kde některý ze symbolů X_1 nebo X_2 leží v T přidej k R' pravidlo $A \to X_1' X_2'$.
(6) Pro každý nový nonterminál tvaru a' přidej k R' pravidlo $a' \to a$. Výsledná gramatika je $G' = (N', T, R', S')$; množina N' obsahuje všechny nonterminály tvaru $\langle X_i ... X_k \rangle$ a a'.

2.6.1 Transformace bezkontextových gramatik

Algoritmus 2.1 pro převod bezkontextové gramatiky do Chomského normální formy očekává na vstupu *vlastní gramatiku* bez jednoduchých pravidel. Vlastní gramatikou se nazývá gramatika bez zbytečných symbolů, ε-pravidel a bez cyklů. Vlastní gramatiky bez jednoduchých pravidel docílíme aplikací algoritmů pro transformace bezkontextových gramatik. Pro stručnost dále uvedeme pouze potřebné definice a algoritmy, které budou prakticky použity později v tomto textu. Kompletní teorie týkající se tohoto tématu je uvedena v [1].

Definice 2.7 Říkáme, že gramatika $G = (N, T, R, S)$ je *gramatikou bez ε-pravidel*, jestliže buď P neobsahuje žádné ε-pravidlo, nebo, případě $\varepsilon \in L(G)$, existuje jediné ε-pravidlo tvaru $S \to \varepsilon$ a výchozí symbol S se nevyskytuje na pravé straně žádného pravidla.

Algoritmus 2.2 Transformace na gramatiku bez ε-pravidel.
Vstup: Gramatika $G = (N, T, R, S)$.
Výstup: Ekvivalentní gramatika $G' = (N', T, R', S')$ bez ε-pravidel.
Metoda:

(1) Sestroj $N_\varepsilon = \{A \mid A \in N \wedge A \Rightarrow^* \varepsilon\}$.

(2) Nechť P' je množina pravidel, kterou konstruujeme takto:

a) Jestliže $A \to \alpha_0 B_1 \alpha_1 B_2 \ldots B_k \alpha_k$ je v P, $k > 0$ a každé B_i je v N_ε, $1 \le i \le k$, avšak žádný ze symbolů řetězců α_j není v N_ε, $0 \le j \le k$, pak k P' přidej všechna nová pravidla tvaru

$$A \to \alpha_0 X_1 \alpha_1 X_2 \ldots X_k \alpha_k,$$

kde X_i je buď B_i, nebo ε. Nepřidávej ε-pravidlo $A \to \varepsilon$, které se objeví, jsou-li všechna $\alpha_i = \varepsilon$.

b) Jestliže $S \in N_\varepsilon$, pak k P' přidej pravidla

$$S' \to \varepsilon \mid S.$$

S' je nový výchozí symbol. Polož $N' = N \cup \{S'\}$. Jestliže $S \notin N_\varepsilon$, pak $N' = N$ a $S' = S$.

c) Výsledná gramatika má tvar $G' = (N', T, R', S')$.

Definice 2.8 Přepisovací pravidlo tvaru $A \to B$, $A, B \in N$ se nazývá *jednoduché přepisovací pravidlo.*

Algoritmus 2.3 Odstranění jednoduchých pravidel.
Vstup: Gramatika G bez ε-pravidel.
Výstup: Ekvivalentní gramatika G' bez jednoduchých pravidel.
Metoda:

(1) Pro každé $A \in N$ sestroj množinu $N_A = \{B \mid A \Rightarrow^* B\}$ takto:

a) $N_0 = \{A\}, i = 1$.

b) $N_i = \{C \mid (B \to C) \in P \wedge B \in N_{i-1}\} \cup N_{i-1}$.

c) Jestliže $N_i \neq N_{i-1}$, polož $i = i + 1$ a opakuj krok b). V opačném případě je $N_A = N_i$.

(2) Sestroj P' takto: Jestliže $B \to \alpha$ je v P a není jednoduchým pravidlem, pak pro všechna A, pro něž $B \in N_A$, přidej k P' pravidla $A \to \alpha$.

(3) Výsledná gramatika je $G = (N, T, P', S)$.

3 Zpracování přirozeného jazyka

V této kapitole se budeme zabývat základy zpracování přirozeného jazyka. Nadefinujeme frázovou gramatiku, vysvětlíme si pojem korpus a představíme si dva základní pohledy na lingvistickou strukturu věty.

3.1 Frázová gramatika

V této části uvedeme podle [3] definici frázové gramatiky vzhledem ke zpracování přirozeného jazyka. Nové pojmy, jako např. fráze, jsou vysvětleny a dány do významového kontextu s frázovou gramatikou v podkapitole 3.3.2. Jednoduchá frázová gramatika je uvedena v tabulce 5.1.

Definice 3.1 Frázová gramatika je šestice $G = (T, C, N, S, L, R)$, kde

- T je konečná množina terminálních symbolů,
- C je konečná množina předterminálních symbolů,
- N je konečná množina nonterminálních symbolů,
- $S \in N$ je počátečná symbol,
- L je konečná množina přepisovacích pravidel ve tvaru $X \to x, X \in C, x \in T$ nazývaná lexikon neboli *slovník*,
- R je konečná množina pravidel ve tvaru $X \to \gamma, X \in N, \gamma \in (N \cup C)^*$ nazývaná gramatikou.

3.2 Korpus

Tato podkapitola byla převzata z [4].

Korpusem se v současnosti rozumí rozsáhlý vnitřně strukturovaný a ucelený soubor textů daného jazyka elektronicky uložený a zpracovávaný. Texty jsou v korpusu strukturovány a organizovány se zřetelem k využití pro určený cíl, vůči němuž je potom korpus považován za reprezentativní.

Vzhledem k účelu použití rozlišujeme rozdílné typy korpusů. Podle zdroje textů mohou být korpusy psaného nebo mluveného jazyka všeobecné nebo specializované na určitý styl, publicistický nebo odborný. Většina korpusů s ohledem na svou reprezentativnost obsahuje v různém poměru zástupce všech možných kategorií textů. Podle uložených dat mohou korpusy obsahovat pouze holé texty nebo texty určitým způsobem označkované (anotované). Značkované korpusy samozřejmě poskytují více informací o jazyku, a proto je snaha korpusy značkovat. To lze provádět buď ručně, což je ale velice nákladné, nebo automaticky (strojově), což může někdy znamenat zanesení jisté míry nepřesností do značkování. Proto se také mnoho výzkumů v korpusové lingvistice zabývá právě automatickým značkováním textů.

Korpusová data jsou využitelná pro odborníky v řadě oborů – psychologie, sociologie, lexikografie, lingvistika, překladatelství (strojový překlad), umělá inteligence (porozumění v přirozeném jazyce, reprezentace znalostí) aj.

Některé používané korpusy, jako např. PDT (viz podkapitola 3.3.1 níže), mohou mít podle [5] zavedeno rozdělení korpusových dat do následujících skupin:

- *Trénovací data* (train) – tvoří přibližně 80 % celkového množství dat. Slouží k libovolnému použití.

- *Vývojová testovací data* (dtest) – asi 10 %. Používají se k prověřování hypotéz nebo vytvořených nástrojů.
- *Evaluační testovací data* (etest) – asi 10 %. Uživatel by se na ně nikdy neměl dívat, protože jsou určena výhradně pro vyhodnocení. I tak by se měla používat s rozvahou, protože závěry získané z opakovaného testování by mohly vést ke změně původní hypotézy či nástroje, čímž by evaluační testovací data začala sloužit jako vývojová testovací data.

Poměrové rozdělení uvedených skupin je spíše doporučené, často záleží na očekávaném použití korpusu. Různé zdroje uvádějí jiná doporučená dělení, např. v poměru 6:2:2 nebo 4:3:3, některé dokonce pracují pouze se dvěma množinami dat (nejčastěji trénovací a vyhodnocovací), a to např. v poměru 7:3 či 8:2.

Treebank [8] je textový korpus, jehož každá věta byla analyzována, což znamená, že byla anotována syntaktickou strukturou. Ty bývají často reprezentovány ve formě stromů, odtud tedy plyne název treebank. Často se vytvářejí na základě již morfologicky anotovaného korpusu (označené slovní druhy – part-of-speech tags), někdy jsou také vylepšeny sémantickou případně jinou lingvistickou informací.

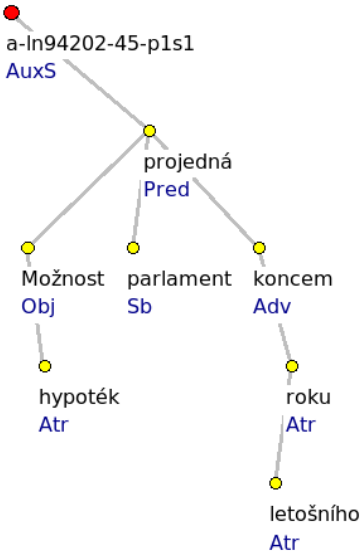
3.3 Pohledy na syntaktickou strukturu věty

Tato podkapitola byla volně převzata z [3], [5] a [6].

Momentálně rozlišujeme dva základní přístupy k syntaktické (lingvistické) struktuře věty – *závislostní* (dependency view) a *frázový* (phrasal view). Od každého z těchto pohledů se odvíjejí celé vytvořené korpusy a způsoby anotace korpusových dat.

3.3.1 Závislostní přístup

Základní myšlenkou této syntaktické struktury je závislostní vztah mezi jednotlivými slovy věty, tj. mezi větnými členy. Hlavním větným členem je nejčastěji sloveso (přísudek), na kterém závisí všechny ostatní větné členy (podmět, předmět, přívlastek, příslovečné určení). Tyto závislosti mezi jednotlivými slovy pak větu přirozeně uspořádají do syntaktického stromu. Vytvořený kořenový strom (acyklický orientovaný graf s kořenovým vrcholem) má přesně tolik vrcholů, kolik je slov ve větě (na rozdíl od frázového stromu, kde slova ve větě představují jen listy), případně je přidán pomocný kořenový uzel reprezentující celou větu. Příklad takového stromu je uveden na obrázku 3.1. Vztahy mezi slovy jsou reprezentovány hranami grafu věty, kdy každá hrana vyjadřuje závislost jednoho slova na druhém. Z logiky věci lze také pozorovat značnou podobnost se stromem vytvářeným při větném rozboru.



Obrázek 3.1: Závislostní syntaktický strom pro větu „*Možnost hypoték parlament projedná koncem letošního roku.*“, převzato ze vzorových vizualizovaných dat pro PDT 2.0[2].

V závislostní syntaxi mohou mezi větnými členy ve větě existovat tyto dva základní typy vztahů:

- *závislost* (determinace) – jedno slovo nějakým způsobem (významově) určuje druhé,
- *koordinace* (několikanásobný větný člen), *apozice* (přístavek), *parenteze* (vsuvka) – slova jsou na stejné úrovni.

Tradiční závislostní syntax také rozlišuje dvě úrovně větné skladby:

- *úroveň věty jednoduché* – do syntaktických vztahů vstupují větné členy, které nejčastěji odpovídají právě jednomu slovu,
- *úroveň souvětí* – do syntaktických vztahů vstupují celé věty.

Závislostní přístup pro jazyk český je rozvíjen na Ústavu formální a aplikované lingvistiky (ÚFAL) Univerzity Karlovy, kde byl také vytvořen Pražský závislostní korpus (Prague Dependency Treebank, PDT), jehož zdrojová data jsou nezkrácené články z novin a časopisů z počátku 90. let 20. století (Lidové noviny, Mladá fronta Dnes, Českomoravský Profit a Vesmír). Plná verze dat sestává z 7110 ručně anotovaných textových dokumentů, obsahujících celkem 115844 vět s 1957247 slovními jednotkami (slovy, čísly, interpunkcí) [5].

3.3.2 Frázový přístup

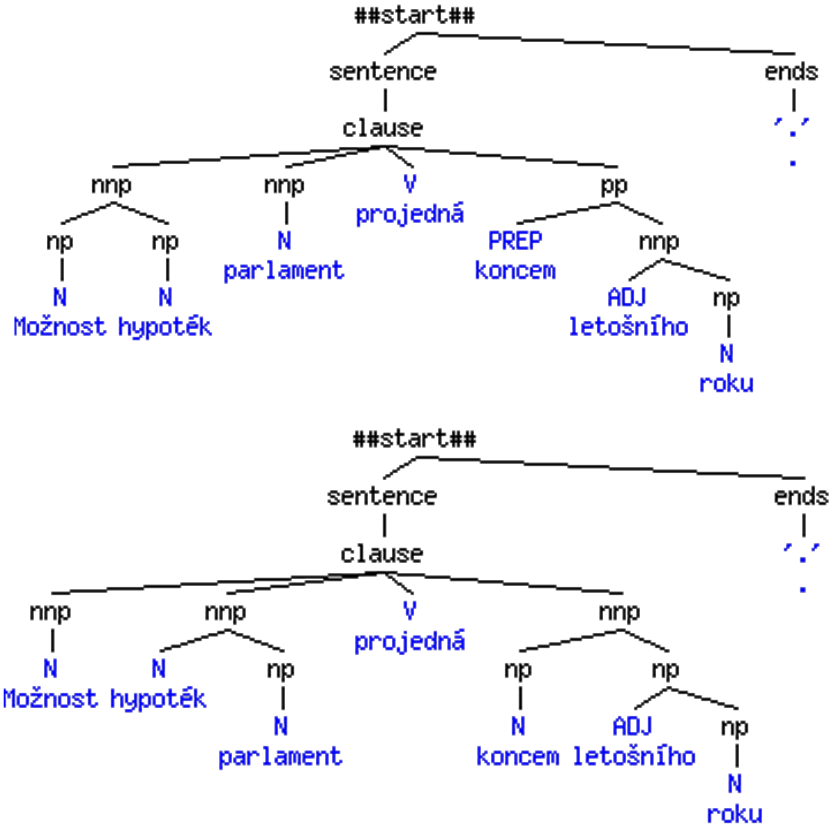
Frázový přístup se větou zabývá z hlediska slovních druhů a je na rozdíl od závislostního pohledu schopen syntakticky zpracovávat i větší větné celky než jen slova. Slova jsou sdružována do frází (constituent), které se chovají jako jednotky, jež se mohou vyskytovat v různých částech věty. Ukázka této vlastnosti je uvedena v příkladu 3.1, kde jsou v prvních dvou větách označené fráze zaměněny, ve třetí větě už ale při roztržení jedné z frází věta ztrácí význam. Fráze bývají typicky řízeny určitým slovním druhem, podle kterého mají i svůj název, např. slovesné, jmenné, předložkové či příslovečné fráze.

[2] http://ufal.mff.cuni.cz/pdt2.0/visual-data/sample/index.htm

```
Petr jel [do školy] [svým novým autem].
Petr jel [svým novým autem] [do školy].
 * Petr jel svým novým do školy autem.
```

Příklad 3.1: Chování frází jako jednotek s různým výskytem ve větě. Pro český jazyk upraveno z [3].

V porovnání se závislostní syntaxí, kde pro zadanou větu existuje vždy pouze jeden strom, se pro složkový přístup vytváří stromů znatelně více, které jsou navíc mnohem bohatší a jejich podoba odpovídá konstrukci derivačního stromu z bezkontextové gramatiky. Slova věty (terminální symboly) jsou reprezentována listy stromu, uzly stromu popisují neterminální symboly gramatiky (frázové struktury). Příklady dvou různých syntaktických stromů pro stejnou větu jsou uvedeny na obrázku 3.2, kde můžeme pozorovat jejich odlišnou strukturu.



Obrázek 3.2: Frázové syntaktické stromy pro větu „*Možnost hypoték parlament projedná koncem letošního roku.*“, vytvořeno pomocí online nástroje wwwsynt[3].

Frázový přístup pro jazyk český je rozvíjen v Centru zpracování přirozeného jazyka na Fakultě informatiky Masarykovy univerzity v Brně. Zde byl také vytvořen Brněnský frázový korpus (Brno Phrasal Treebank), jehož hlavním zdrojem vět je výše zmíněný PDT. Korpus se celkově skládá z 86058 tokenů (slovních jednotek) a 6162 syntakticky označených vět. Kromě stromů ve frázovém formalismu zdrojová data korpusu dále obsahují informace o zdroji textu a lemmatizované[4] a morfologicky označené texty. V centru jsou také vyvíjeny syntaktické analyzátory, jejichž nejvýznamnějším představitelem je analyzátor `synt` [7].

[3] http://nlp.fi.muni.cz/projekty/wwwsynt/query.cgi

[4] Lemmatizace – hromadné sdružování různě skloňovaných nebo časovaných forem slova, aby mohly být zpracovány jako jeden prvek.

3.3.3 Vzájemný vztah závislostní a frázové syntaktické struktury

Závislostní gramatiky se vyznačují řídícím vztahem mezi jednotlivými větnými členy, standardní bezkontextové gramatiky ale běžně žádnou takovou vazbu nemají. Avšak moderní lingvistické teorie a všechny moderní statistické analyzátory pro zavedení této vlastnosti používají frázová ručně psaná tzv. řídicí pravidla. Potom tedy je řídicím členem neboli hlavou jmenné fráze podstatné jméno, číslovka nebo přídavné jméno, slovesné fráze pravidelné či modální sloveso, apod. Tato pravidla mohou být tudíž použita pro vytvoření závislostního stromu z frázového syntaktického stromu.

4 Syntaktická analýza a její algoritmy

Tato kapitola byla volně převzata z [3] a [9], vyjma částí explicitně uvedených jinak.

Algoritmy pro bezkontextovou syntaktickou analýzu jsou základem téměř pro všechny přístupy syntaktické analýzy k vytváření hierarchických frázových struktur. Existuje velké množství algoritmů lišících se v různých aspektech. Některé algoritmy například analyzují veškeré možnosti pro dané vstupy, jiné se zase zabývají pouze analýzou částí, které mohou pozitivním způsobem přispět k syntaktické analýze, a zpracování zbylých částí (nedůležitých pro úspěšné dokončení) se snaží různými způsoby v průběhu algoritmu omezit.

Hlavní společnou vlastností algoritmů využívaných při zpracování přirozeného jazyka je jejich efektivní práce s mnohoznačností, kdy nevytvářejí stejné syntaktické podstromy identickým způsobem více než jednou. Toho v průběhu algoritmu dosahují ukládáním analyzovaných derivací do tabulky či grafu (chart), ze které je následně výhodnější data získat, než je znovu přepočítávat. Zpracování daného podproblému pouze jednou a následné znovupoužití uloženého výsledku je jedním ze základních principů *dynamického programování*, které zadaný problém řeší jako složené řešení jednotlivých podproblémů. Kromě dále uvedených algoritmů syntaktické analýzy můžeme uvést i příklady z jiných odvětví informatiky – výpočet *n*-tého členu Fibonacciho posloupnosti, nalezení řešení hlavolamu Hanojské věže, Dijkstrův algoritmus pro nalezení nejkratší cesty v grafu aj.

V následujících odstavcích uvedeme základní rozdělení algoritmů syntaktické analýzy zejména v kontextu zpracování přirozeného jazyka (tedy pro další účely této práce). Následně (i vzhledem k uvedeným rozdělením) popíšeme méně či více podrobně nejznámější algoritmy syntaktické analýzy, ze kterých v dnešní době vychází většina implementovaných algoritmů. Budeme se také zabývat řešením víceznačnosti bezkontextových gramatik a vyhodnocením úspěšnosti syntaktické analýzy.

4.1 Rozdělení algoritmů syntaktické analýzy

4.1.1 Obecné rozdělení podle způsobu konstrukce derivačního stromu

Podle [1] lze algoritmy syntaktické analýzy rozdělit podle způsobu, kterým je konstruována derivace věty, tj. vytvářen derivační strom (definice 2.4), do těchto dvou základních skupin:

- syntaktická analýza *shora dolů* (top-down parsing),
- syntaktická analýza *zdola nahoru* (bottom-up parsing).

Při syntaktické analýze shora dolů začínáme derivační strom budovat od výchozího symbolu (kořene derivačního stromu) a postupnými derivacemi dojdeme k terminálním symbolům, které tvoří analyzovanou větu (koncovým uzlům derivačního stromu). Problém tohoto typu analyzátorů spočívá ve správnosti volby přímých derivací, tj. pořadí používání přepisovacích pravidel.

Při syntaktické analýze zdola nahoru začínáme derivační strom budovat od koncových uzlů a postupnými přímými redukcemi dojdeme ke kořenu (výchozímu symbolu gramatiky). Základním problémem této třídy syntaktických analyzátorů je hledání prvního podřetězce věty (v dalších krocích větných forem), který může být redukován k jistému nonterminálu – kořenu podstromu derivačního stromu.

4.1.2 Rozdělení podle způsobu ukládání analyzovaných derivací do tabulky

Podmnožinu tabulkových syntaktických analyzátorů (chart parser) lze podle [9] dále rozdělit na základě způsobu ukládání analyzovaných derivací do tabulky na:

- aktivní tabulkový syntaktický analyzátor (*active chart parser*),
- pasivní tabulkový syntaktický analyzátor (*passive chart parser*).

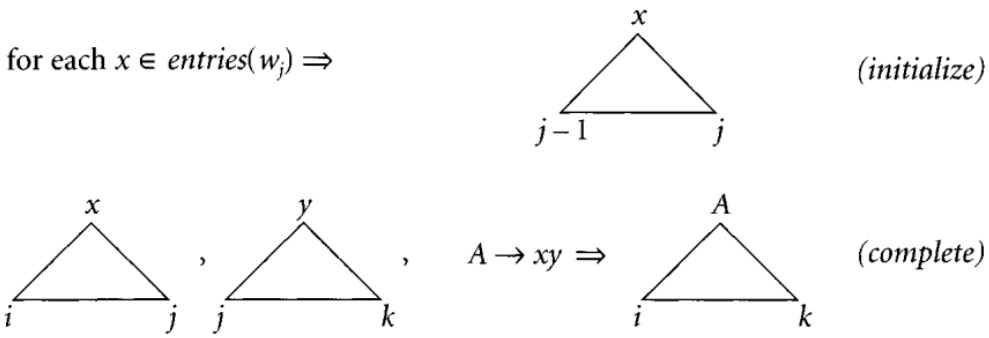
Hlavní rozdíl mezi těmito dvěma přístupy spočívá v tom, že při pasivní syntaktické analýze je do tabulky derivační podstrom (nonterminál či níže také fráze) uložen až při zpracování a dokončené derivaci všech jeho uzlů. Naproti tomu při aktivní syntaktické analýze je derivační podstrom do tabulky zaznamenán již po zpracování dílčích uzlů (typicky probíhá zleva doprava), kdy zbylé uzly podstromu bývají ještě zpracovávány.

4.2 CKY algoritmus

CKY algoritmus (Cocke – Kasami – Younger), někdy také uváděn jako CYK algoritmus, je základní tabulkový bezkontextový algoritmus pro syntaktickou analýzu bezkontextových gramatik. Vzhledem k rozdělení algoritmů uvedeného v podkapitole 4.1 jej můžeme označit jako pasivní algoritmus syntaktické analýzy zpracovávající vstupní řetězec od terminálních symbolů až ke kořenovému nonterminálnímu symbolu. Jedná se tedy o syntaktickou analýzu zdola nahoru.

Vstupem tohoto algoritmu je bezkontextová gramatika v Chomského normální formě, kde jednoduše řečeno pravidla gramatiky přepisují nonterminální symboly buď na právě jeden terminální symbol, nebo na právě dva nonterminální symboly. Přesná definice Chomského normální formy je uvedena v definici 2.6. Udávaná složitost tohoto algoritmu vzhledem k počtu slov ve vstupním řetězci n je $O(n^3)$.

Na obrázku 4.1 jsou graficky znázorněny kroky algoritmu, které bývají často označovány jako kroky inicializace (initialize) a hlavní smyčky (complete). Terminální symboly přesně odpovídají jednotlivým slovům w_j vstupního řetězce. Pro vynechání unárních pravidel přepisující terminální symboly (zjednodušení) definujeme také funkci *entries* vracející kategorii slova (v našem kontextu slovní druh). V inicializačním kroku jsou nalezeny lexikální jednotky vstupního řetězce, které jsou uloženy jako jednotlivé fráze (constituents). Následně jsou v hlavní smyčce procesovány větší celky vstupního řetězce, kdy jsou pro každé dvě sousedící fráze a pravidla, která lze aplikovat, vytvářeny nové fráze vyšší úrovně. Smyčku opakujeme tak dlouho, dokud nejsou vytvořeny veškeré možné fráze.



Obrázek 4.1: Kroky CKY algoritmu. Převzato z [9].

Protože se jedná o základní algoritmus, ze kterého vychází většina dále uvedených algoritmů, uvedeme zde i pseudokód tohoto algoritmu.

Algoritmus 4.1 Pseudokód CKY algoritmu

Vstup:

Řetězec S skládající se z n znaků: $w_1 \dots w_n$.

Bezkontextová gramatika v Chomského normální formě obsahující r nonterminálních symbolů $R_1 \dots R_r$, jejichž podmnožina R_S je množina počátečních symbolů.

Pole $P[n, n, r]$ datového typu boolean, inicializované pro všechny prvky hodnotou *false*.

Výstup:

Rozhodnutí, zda-li je vstupní řetězec přijímaný danou bezkontextovou gramatikou, a náleží tedy do generovaného bezkontextového jazyka. Nastává v případě, kdy je stav (fráze) pokrývající celý vstupní řetězec S přiřazen nějakému počátečnímu symbolu z množiny R_S.

Metoda:

```
for each i = 1 to n
  for each rule Rj -> wi
    set P[i,1,j] = true
for each i = 2 to n
  for each j = 1 to n-i+1
    for each k = 1 to i-1
      for each rule RA -> RB RC
        if P[j,k,B] and P[j+k,i+k,C] then
          set P[j,I,A] = true
if any of P[1,n,x] is true (x is iterated over the set s, where s
are all the indices for RS) then
  S is member of language
else
  S is not member of language
```

Pro přehlednost při výkladu bývá algoritmus také často zobrazován formou tabulky (odtud také pojmenování tabulkový algoritmus). Pro vstupní řetězec $S = w_1 w_2 w_3 w_4 w_5$ složený z pěti znaků (slov) je jejich pokrytí v tabulce algoritmu vyjádřeno v tabulce 4.1.

i, j	1	2	3	4	5
1	w_1	w_2	w_3	w_4	w_5
2	$w_1 w_2$	$w_2 w_3$	$w_3 w_4$	$w_4 w_5$	
3	$w_1 w_2 w_3$	$w_2 w_3 w_4$	$w_3 w_4 w_5$		
4	$w_1 w_2 w_3 w_4$	$w_2 w_3 w_4 w_5$			
5	$w_1 w_2 w_3 w_4 w_5$				

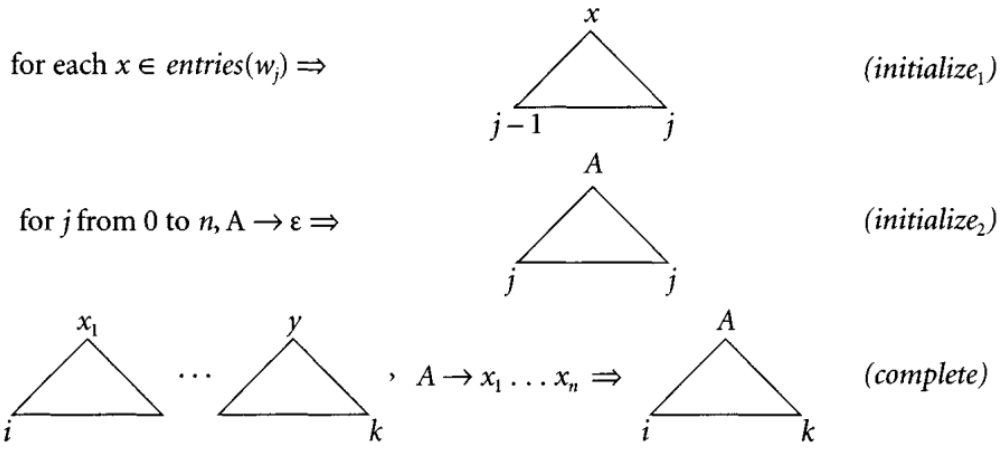
Tabulka 4.1: Pokrytí vstupního řetězce v tabulce (chart) CKY algoritmu.

4.3 Rozšířený CKY algoritmus

Pojem *rozšířený CKY algoritmus* (extended CKY algorithm) je z teoretického pohledu příliš obecný, a proto bývá často také v různých literaturách vykládán jiným způsobem. Jak je z názvu patrné, implementovaná rozšíření vycházejí ze základního CKY algoritmu popsaného výše v podkapitole 4.2.

Autoři v [9] popisují rozšíření algoritmu pro práci s libovolnými bezkontextovými gramatikami bez omezení počtu symbolů na pravé straně pravidel. Gramatika tedy již nemusí být definována v Chomského normální formě. Tato varianta algoritmu bývá také nazývána jako tabulkový pasivní algoritmus syntaktické analýzy pracují zdola nahoru (bottom-up passive chart parsing). Protože gramatika může nyní obsahovat pravidla s prázdnou pravou stranou (epsilon pravidla), je oproti základnímu CKY algoritmu potřeba provést inicializační krok navíc, který vytvoří pro všechny příslušné kategorie vzhledem ke vstupnímu řetězci pravidla s prázdnou pravou stranou, což je znázorněno na obrázku 4.2.

Tato modifikace CKY algoritmu pracuje s konstantní složitostí $n^{\rho+1}$, kde ρ udává maximální počet symbolů na pravé straně pravidla (maximální počet uzlů derivačního podstromu).



Obrázek 4.2: Kroky rozšířeného CKY algoritmu. Převzato z [9].

Autoři v [3] zavádějí do algoritmu, podobně jako výše, gramatická pravidla s prázdnou pravou stranou, zaměřují se však více na nezvýšení složitosti algoritmu, který dále upravují možností přidání unárních pravidel do zpracovávané bezkontextové gramatiky. Na rozdíl od autorů v [9] však zdůrazňují význam binarizace (vstupní gramatika v Chomského normální formě), bez které nelze docílit kubické složitosti algoritmu vzhledem k délce vstupního řetězce. Nevýhodou binarizace ovšem zůstává jiná struktura výsledného derivačního stromu, která kvůli převodu obecné bezkontextové gramatiky do Chomského normální formy (vytvoření nových „hloupých“ pravidel) nemusí odpovídat předpokládané frázové struktuře věty.

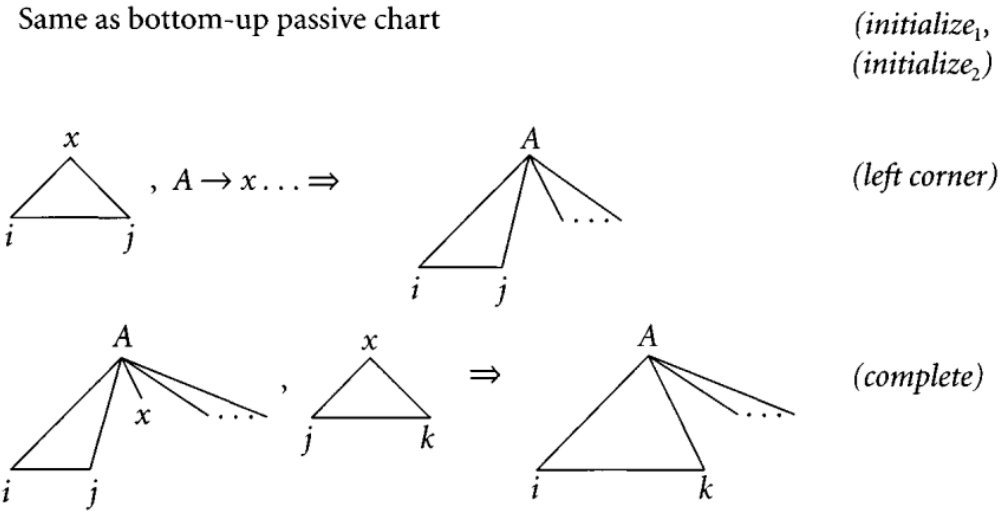
4.4 Aktivní varianta rozšířeného CKY algoritmu

Rozšířený CKY algoritmus popsaný v předchozí podkapitole 4.3 lze také upravit a používat ve variantě aktivního tabulkového analyzátoru. V angličtině bývá uváděn jako *bottom-up left corner*, což by se do češtiny dalo volně přeložit jako syntaktický analyzátor zdola nahoru pracující podle pravidla levé ruky.

Vzhledem k tomu, že obě pasivní varianty CKY algoritmu zaznamenávají do tabulky až kompletní derivované podstromy, může při syntaktické analýze docházet k opakovanému pokusu o

derivaci stejného podstromu bez ohledu na znalost předchozího úspěchu či neúspěchu. Tato vlastnost nemá výrazný dopad na algoritmy pracující s generickými kategoriemi, které zde popisujeme, nicméně opakování stejných kroků může mít mnohem větší dopad v syntaktických analyzátorech využívající pokročilejší gramatické formalismy – kategorie může být rozšířena o zakódované lingvistické informace následně zpracovávané operací sjednocení. Tyto pokročilejší algoritmy již přesahují rozsah této práce.

Na obrázku 4.3 jsou znázorněny kroky tohoto aktivního algoritmu. Inicializační část je shodná s rozšířeným CKY algoritmem z obrázku 4.2. Hlavní smyčka nejdříve zpracuje nejlevější synovský uzel (typicky nejlevější nonterminál pravé strany pravidla) a až následně jsou postupně zpracovávány další synovské uzly v pořadí. Po dokončení zpracování všech uzlů daného podstromu se dostává do stejného stavu jako při pasivní syntaktické analýze.



Obrázek 4.3: Kroky algoritmu syntaktické analýzy bottom-up left corner. Převzato z [9].

4.5 Earleyho syntaktický analyzátor

Převzato z [10]. Zástupcem syntaktické analýzy shora dolů je tzv. *Earleyho analyzátor* (Earley parser), který je založen na tečkové notaci. Časová složitost udávaná pro tento algoritmus je podobně jako pro CKY algoritmus $O(n^3)$. Tento algoritmus je efektivní zejména pro gramatiky s levou rekurzí.

Způsob výpočtu algoritmu je popsaný dále a používá standardně používanou notaci (pro ujasnění) – malá písmena označují terminální symboly, velká písmena nonterminální symboly a řecká písmena reprezentují řetězce složené z nula nebo více terminálních a nonterminálních symbolů

Analyzátor používá *tečkovou notaci* - pravidlo $A \rightarrow B.CD$ s tečkou před C reprezentuje situaci, kdy B již bylo analyzováno a zbývá zanalyzovat CD. Pro každou vstupní pozici analyzovaného řetězce vytvoříme množinu stavů, které reprezentují kombinaci dvou prvků:

(1) Pravidlo s tečkovou notací.
(2) Pozice, na které pravidlo začalo – výchozí stav.

Stav na vstupní pozici k se nazývá $S(k)$. Počáteční stav analýzy je $S(0)$. Analyzátor vykonává iterativně tři typy operací:

(1) *Predikci:* Pro každý stav v $S(k)$ tvaru $(X \rightarrow \alpha.Y\beta, j)$, kde j je výchozí stav, přidej $(Y \rightarrow .\gamma, k)$ do $S(k)$ pro každé pravidlo s Y na levé straně.

(2) *Čtení:* Pokud máme a jako další symbol v analyzovaném řetězci, pak pro každý stav v $S(k)$ ve tvaru $(X \to \alpha.a\beta, j)$ přidáme $(X \to \alpha a.\beta, j)$ do $S(k+1)$.

(3) *Ukončování:* Pro každý stav v $S(k)$ tvaru $(X \to \alpha., j)$ najdeme stavy v $S(j)$ tvaru $(X \to \alpha.X\beta, i)$ a přidáme stavy $(Y \to \alpha X.\beta, i)$ do $S(k)$.

Algoritmus stále přidává nové stavy, dokud lze takové přidat. Důležité také je, že do množiny stavů nepřidáváme duplicitní stavy, ale pouze nové.

4.6 Víceznačnost syntaktické analýzy při zpracování přirozeného jazyka

Víceznačnost gramatik, definovaná v podkapitole 2.5, se při zpracování přirozeného jazyka projevuje možným generováním větných konstrukcí, které nebyly autorem gramatiky zamýšleny. Pravděpodobnost tohoto jevu roste s rozsahem pokrytí jazyka, pro který je gramatika vytvořena. Některé syntaktické systémy používají manuálně vytvořené heuristiky, které část potencionálních mnohoznačností vyřeší, nicméně tato metoda je vhodná spíše pro úzce vymezené oblasti, než pro pokrytí víceznačností v rozsahu celého jazyka. Alternativním přístupem, na který se zaměřuje mnoho výzkumných činností, je automatické učení syntaktického analyzátoru vybrat statisticky nejpoužitelnější derivaci (pravidlo) na základě textu, který byl již předem syntakticky zpracován ručně. Pro tyto účely se tedy používají manuálně anotované korpusy, tzv. *treebank*, které jsme definovali v podkapitole 3.2.

Data z těchto korpusů následně, díky přiřazení statistických preferencí ke gramatice, umožňují syntaktické analýze vyřešit potencionální víceznačnosti. Gramatika může být buď vytvořena manuálně předem, nebo může být vygenerována automaticky z anotovaného korpusu. Druhá uvedená možnost bude pravděpodobně více generická a její nevýhodou bude absence sémantické informace. Naproti tomu první varianta bude zpracována přesněji a bude více podporovat sémantickou interpretaci věty.

Gramatiky definované z anotovaného korpusu jsou vytvářeny definováním pravidel pro každý podstrom anotovaného korpusu hloubky jedna. Pravděpodobnost použití daného pravidla v syntaktické analýze je poté vypočtena na základě frekvence výskytu každého pravidla a normalizované frekvence jeho výskytu tak, že součet pravděpodobností všech pravidel se stejnou levou stranou je jedna. Výsledkem tohoto zpracování je poté *pravděpodobnostní bezkontextová gramatika*, kterou jsme definovali v podkapitole 2.3. Pravděpodobnost vytvořeného derivačního stromu pro daný vstupní řetězec se potom vypočte jako součin pravděpodobností jednotlivých pravidel použitých při konstrukci derivačního stromu. Tento výpočet je demonstrován v příkladu 5.3 v podkapitole 5.2. Syntaktický analyzátor takto vypočte pravděpodobnosti pro všechny nalezené derivační stromy a jako výsledek vrátí ten s nejvyšší pravděpodobností výskytu.

4.7 Rozšířený CKY algoritmus pracující s pravděpodobnostní bezkontextovou gramatikou

Budeme-li na syntaktický analyzátor klást požadavek na potlačení gramatických víceznačností, což je žádoucí, je potřebné výše uvedené algoritmy upravit pro práci s pravděpodobnostními bezkontextovými gramatikami. V této podkapitole uvedeme úpravu rozšířeného CKY algoritmu podle [3], názorný příklad jeho průběhu je poté demonstrován v podkapitole 5.4.

4.7.1 Viterbiho algoritmus

Viterbiho algoritmus je podle [13] algoritmus dynamického programování pro hledání/nalezení nejpravděpodobnější posloupnosti skrytých stavů – nazývané Viterbiho cesta. Jeho výsledkem je posloupnost pozorovaných událostí, především v kontextu skrytých Markovových modelů (není součástí tohoto textu). Používá se také pro hledání nejpravděpodobnějšího vysvětlení určitého pozorování, v našem případě tedy pro nalezení nejpravděpodobnějšího bezkontextového derivačního stromu vstupního řetězce.

Samotný Viterbiho algoritmus je popsán níže, jeho uváděná složitost je $O\left(T \times |S|^2\right)$.

Algoritmus 4.2 Viterbiho algoritmus

Předpokládejme, že je dán skrytý Markovův model se stavovým prostorem S, pravděpodobnostmi π_i začátku ve stavu i (počáteční pravděpodobnosti), pravděpodobnostmi $a_{i,j}$ pro přechod ze stavu i do stavu j (přechodové pravděpodobnosti). Pokud pozorujeme výstupní posloupnost $y_1, \ldots, y_T$, pak nejpravděpodobnější posloupnost stavů $x_1, \ldots, x_T$, která produkuje pozorovaný výstup, je dána rekurentními vztahy:

$$V_{1,k} = P(y_1 \mid k) \cdot \pi_k$$
$$V_{t,k} = P(y_1 \mid k) \cdot \max_{x \in S} \left(a_{x,k} \cdot V_{t-1,x}\right),$$

kde $V_{t,k}$ je pravděpodobnost nejpravděpodobnější posloupnosti stavů odpovědné za prvních t pozorování, jejíž koncový stav je k. Pro získání Viterbiho cesty lze používat zpětné ukazatele, které zachycují, jaký stav x byl použit ve druhé rovnici. Nechť $Ptr(k,t)$ je funkce, která vrací hodnotu x použitou pro výpočet $V_{t,k}$ pokud $t > 1$, nebo k pokud $t = 1$. Pak:

$$x_T = \arg\max_{x \in S} \left(V_{T,x}\right),$$
$$x_{t-1} = Ptr\left(x_t, T\right).$$

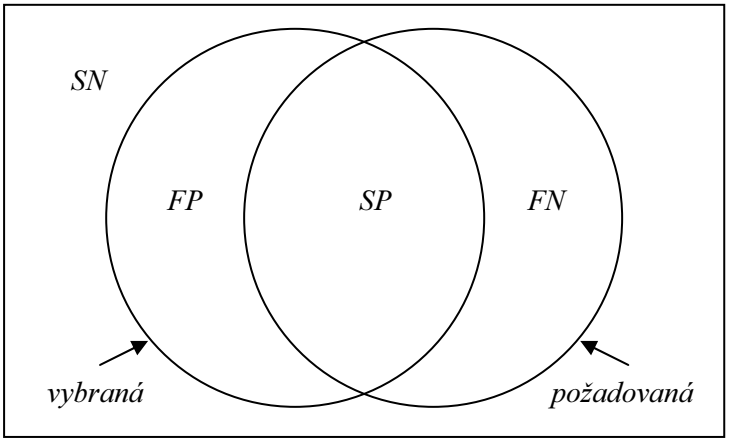
Pro zpracování pravděpodobnostních bezkontextových gramatik je tedy vhodné do rozšířeného CKY algoritmu zapracovat výše uvedený Viterbiho algoritmus, díky kterému budeme schopni vyhledat nejpravděpodobnější derivační strom vstupního řetězce. Pseudokód funkce takového rozšířeného CKY algoritmu zpracovávajícího i unární pravidla může potom podle [3] vypadat následovně:

```
function CKY(words, grammar) returns [most_probable_parse,prob]
  score = new double[#(words)+1][#(words)+1][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#nonterms]]
  // inicializace
  for i=0; i<#(words); i++
    for A in nonterms
      if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i])
    // unární pravidla
    boolean added = true
    while added
      added = false
      for A, B in nonterms
        if score[i][i+1][B] > 0 && A->B in grammar
          prob = P(A->B)*score[i][i+1][B]
          if prob > score[i][i+1][A]
            score[i][i+1][A] = prob
            back[i][i+1][A] = B
            added = true
  // hlavní smyčka
  for span = 2 to #(words)
    for begin = 0 to #(words)- span
      end = begin + span
      for split = begin+1 to end-1
        for A,B,C in nonterms
          prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
          if prob > score[begin][end][A]
          score[begin]end][A] = prob
          back[begin][end][A] = new Triple(split,B,C)
      // unární pravidla
      boolean added = true
      while added
        added = false
        for A, B in nonterms
          prob = P(A->B)*score[begin][end][B];
          if prob > score[begin][end][A]
            score[begin][end][A] = prob
            back[begin][end][A] = B
            added = true
  return buildTree(score, back)
```

4.8 Vyhodnocení úspěšnosti syntaktické analýzy

Statistické modely pro zpracování přirozeného jazyka, popisované zejména v této kapitole, využívají podle [2] pro měření úspěšnosti a vyhodnocení pojmy *precision* a *recall*, které vycházejí z binárního klasifikačního systému. V češtině existují pro tyto pojmy překlady *senzitivita* a *specificita*, které se

ale uvádějí spíše ve spojitosti s lékařským oborem, proto dále v textu zůstaneme pro tyto pojmy u používání anglického názvosloví. Při řešení problému máme často v rámci nějaké množiny stavů definovánu sadu *požadovaných* (cílových) stavů (např. věta, kde mají slova určitý význam/kategorii) a výstupem našeho systému je potom sada *vybraných* stavů (např. věta se slovy, které mají přiřazeny význam/kategorii na základě syntaktické analýzy). Graficky lze tuto situaci znázornit Vénnovým diagramem na obrázku 4.4.



Obrázek 4.4: Vénnův diagram zobrazující binární klasifikaci. Převzato z [2].

Výsledek binární klasifikace lze také pro požadované a vybrané stavy zapsat do 2x2 kontingenční tabulky, kde hodnoty v tabulce udávají frekvenci nebo počet stavů v jednotlivých (disjunktních) podmnožinách dané množiny stavů:

		Požadované stavy (skutečná hodnota)	
		Pozitivní	**Negativní**
Vybrané stavy (přiřazená hodnota)	**Pozitivní**	*Skutečně pozitivní (SP)*	*Falešně pozitivní (FP)*
	Negativní	*Falešně negativní (FN)*	*Skutečně negativní (SN)*

Tabulka 4.2: Kontingenční tabulka rozdělení stavů. Podle [2].

Význam jednotlivých podmnožin je poté vzhledem k výše uvedenému kontextu syntaktické analýzy následující:

- *Skutečně pozitivní (SP, true positive)* – zjištěné případy odpovídají skutečnosti. Syntaktický analyzátor správně určil kategorii slova, přiřazená hodnota tedy odpovídá skutečné hodnotě.
- *Skutečně negativní (SN, true negative)* – zjištěné případy nepřítomnosti odpovídají skutečnosti, což systém také vyhodnotil správně.
- *Falešně pozitivní (FP, false positive)* – falešný poplach, někdy bývá také označován jako *Chyba I typu*. Špatně určené kategorie syntaktickým analyzátorem.
- *Falešně negativní (FN, false negative)* – odpovídá přehlédnutému výskytu, jinak také *Chyba II typu*. Kategorie, které nebyly přiřazeny syntaktickým analyzátorem.

Následně na základě uvedeného rozdělení definujeme parametry precision a recall. *Precision P* je definován jako poměr správně vyhodnocených stavů vůči všem stavům, které systém ohodnotil (poměr vybraných prvků, které jsou správně). *Recall R* podobně udává poměr správně vyhodnocených stavů, avšak vůči všem požadovaným stavům (poměr správných prvků, které jsou vybrány). Vzorce pro výpočet obou parametrů jsou tedy následující:

$$P = \frac{SP}{SP + FP} \qquad R = \frac{SP}{SP + FN}$$

Vyhodnocovat úspěšnost analýzy na základě dvou parametrů není zrovna žádoucí případ, zvláště když v některých případech je možné hodnotu jednoho parametru zvyšovat na úkor druhého. Proto tyto dvě hodnoty kombinujeme do tzv. parametru *F measure* (někde uváděn také jako *F score*), který je definován:

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}},$$

kde P a R označují precision a recall definované výše, α je faktor, který určuje váhy těchto dvou parametrů. Hodnota $\alpha = 0{,}5$ bývá často používána pro jejich vzájemné vyvážení a vzorec je poté možno zjednodušit na:

$$F = \frac{2PR}{(R+P)}.$$

Pro vyhodnocení úspěšnosti vytvoření derivačního stromu pomocí syntaktického analyzátoru využíváme tzv. *PARSEVAL metriky* (zkrácené parser evaluation). Při vyhodnocování statistických analyzátorů pro zpracování přirozeného jazyka se typicky porovnávají derivační strom vytvořený syntaktickým analyzátorem s tzv. *gold standard tree*, což je předpokládaný derivační strom, který měl analýzou vzniknout, a považujeme jej za správný (nejčastěji ručně anotovaný). Samotné derivační stromy lze také zapisovat v závorkové notaci, která je pro demonstrační účely vyhodnocování přehlednější. Parametr presicion je poté udáván jako počet závorek z vytvořeného stromu, které se napárují na závorky ze správného stromu, recall vypočítává, kolik závorek správného stromu je obsaženo ve stromu vytvořeném. Hodnotit můžeme také přesnost přiřazení kategorií jednotlivým slovům zpracovávané věty. Příklad výpočtu úspěšnosti syntaktické analýzy je uveden níže v textu v podkapitole 5.5.

4.9 Algoritmy syntaktické analýzy pracující se závislostním přístupem

Algoritmy uvedené v předchozích podkapitolách se primárně zabývají frázovým přístupem ke stavbě věty. Závislostní syntaxe, popsána v podkapitole 3.3.1, definuje syntaktickou strukturu věty složenou z lexikálních prvků vázaných binárními asymetrickými relacemi – závislostmi. Relace závislosti určuje nadřízený a podřízený (závislý) prvek, tento formalismus však není v základu pro bezkontextové gramatiky definován. Moderní lingvistické teorie ovšem závislostní syntax přizpůsobují pro práci v oboru bezkontextových gramatik, a to definicí tzv. řídících pravidel podobně jako ve frázové syntaxi.

K závislostní syntaktické analýze lze přistupovat následujícími metodami:

- Dynamické programování – podobný přístup jako výše uvedený CKY algoritmus, pracuje s lexikalizovanou pravděpodobnostní bezkontextovou gramatikou.

- Grafové algoritmy – prakticky řeší problém nalezení maximální kostry v orientovaném grafu, reprezentantem této skupiny syntaktických analyzátorů je MSTParser[5].
- Syntaktická analýza založená na splnění podmínek (constraint satisfaction) – jsou vyloučeny hrany, které nesplňují předem definované podmínky.
- Deterministická syntaktická analýza – zde ve významu syntaktické analýzy, která neprovádí backtracking. Zástupcem této skupiny je MaltParser[6], který popíšeme v následující podkapitole.

4.9.1 MaltParser

Podle [11] lze MaltParser charakterizovat jako datově řízený syntaktický analyzátor-generátor, který při syntaktické analýze pracuje s anotovaným korpusem dat (treebank – viz podkapitola 3.2). MaltParser implementuje indukční závislostní syntaktický analyzátor, kde je syntaktická analýza vstupního řetězce tvořena derivacemi závislostní syntaxe a je používáno indukční strojové učení při rozhodování v nedeterministických (mnohoznačných) bodech analýzy. Metodika syntaktické analýzy je založena na třech základních komponentách:

- deterministický algoritmus syntaktické analýzy pro tvorbu značených závislostních grafů,
- modely založené na historii pro predikci další akce analyzátoru v nedeterministických bodech analýzy,
- diskriminační učení pro mapování historie na kroky analyzátoru.

Z praktického hlediska provádí syntaktický analyzátor podle [3] posloupnost kroků zdola nahoru, které si můžeme představit jako „shift“ a „reduce“ operace ve standardních algoritmech syntaktické analýzy, kde je však „reduce“ specializovaná akce pro vytvoření závislosti s řídícím prvkem na levé nebo na pravé straně. Samotný syntaktický analyzátor je poté tvořen:

- zásobníkem σ zapisovaným shora doprava, který je inicializován $ROOT$ symbolem,
- bufferem β zapisovaným shora doleva, který je inicializován vstupním řetězcem,
- množinou závislostí A, která začíná algoritmus prázdná,
- množinou prováděných akcí.

Základní přechodově založený závislostní syntaktický analyzátor vypadá následovně:

- Inicializace: $\sigma = [ROOT], \beta = w_1, \ldots, w_n, A = \varnothing$
- Povolené akce:
  (1) *Shift* $\quad \sigma, w_i \mid \beta, A \rightarrow \sigma \mid w_i, \beta, A$
  (2) $Left - Arc_r \quad \sigma \mid w_i, w_j \mid \beta, A \rightarrow \sigma, w_j \mid \beta, A \cup \{r(w_j, w_i)\}$
  (3) $Right - Arc_r \quad \sigma \mid w_i, w_j \mid \beta, A \rightarrow \sigma, w_j \mid \beta, A \cup \{r(w_i, w_j)\}$
- Ukončení: $\beta = \varnothing$

Výběr prováděné akce v daném kroku syntaktické analýzy je předvídán diskriminačním klasifikátorem nad množinou povolených akcí.

Výkonnostně se MaltParser řadí mezi nejlépe hodnocené systémy syntaktické analýzy, pracuje s velmi rychlou lineární časovou složitostí.

[5] http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html
[6] http://www.maltparser.org/

5 Analytický rozbor

V této kapitole budou na ilustračních příkladech prakticky vysvětleny a použity pojmy uvedené dříve v tomto textu. První část se bude detailněji zabývat korpusy, konkrétně formáty dat a rovinami anotace, z nichž podrobněji představíme morfologickou rovinu dat. V druhé části si na jednoduché pravděpodobnostní bezkontextové gramatice ukážeme výpočet pravděpodobnosti derivačního stromu a dané věty. Následně také podle algoritmů definovaných v podkapitole 2.6.1 provedeme transformaci gramatiky do Chomského normální formy, detailněji popíšeme a vysvětlíme průběh rozšířeného CKY algoritmu a v poslední části této kapitoly budeme demonstrovat způsob výpočtu vyhodnocení úspěšnosti syntaktické analýzy. Příklady popisované v druhé části kapitoly (5.2 a dále) jsou podrobně komentované příklady z [3].

5.1 Práce s korpusy

5.1.1 Roviny anotace

Anotační roviny velice úzce souvisí se zpracováním zdrojových dat korpusu. Data v PDT 2.0 jsou anotována na třech rovinách (morfologická, analytická, tektogramatická) a na jedné neanotační rovině (slovní), která reprezentuje čistý text. Roviny v [5] po sobě následují v tomto pořadí:

Slovní rovina

Jak již bylo řečeno výše, data na slovní rovině reprezentují „surový“ text, který je pomocí značek rozdělen do dokumentů a odstavců. Jsou zde rozlišeny slovní jednotky (slova, čísla, interpunkce), které jsou opatřeny jednoznačnými identifikátory.

Hlavním problémem na této rovině je zajištění správného rozlišení slovních jednotek (tokenů), např. rozeznání spojovníku a pomlčky pro správnou identifikaci tokenu.

Morfologická rovina

Na morfologické rovině je posloupnost slovních jednotek rozdělena do vět. Anotace na této rovině spočívá v přiřazení několika atributů tokenům ze slovní roviny, z nichž nejdůležitější jsou morfologické *lemma* a *tag*. V angličtině se tento postup nazývá *Part-of-Speech tagging*, zkráceně také POS tagging.

Lemma reprezentuje základní tvar slovní jednotky a odpovídá jednoznačnému klíči příslušného záznamu v morfologickém slovníku. Atribut *tag* obsahuje morfologickou značku, která má 15 pozic vyjadřujících slovní druh a hodnoty ostatních morfologických kategorií dané slovní jednotky.

Na úrovni této roviny můžeme řešit problémy normalizace, kdy chceme, aby některé různé slovní jednotky vyjadřovaly stejný prvek, např. U.S.A. a USA, nebo stemmingu, kdy pro nás může být výhodné, aby tokeny se stejným kořenem slova (slova příbuzná) opět určovaly stejný prvek.

Uvedeme zde příklady dvou různých přístupů ke značkování morfologických tagů. První z nich, používaný v PDT 2.0 či v morfologickém analyzátoru Morče[7], tvoří řetězec 15 znaků, které kódují jednotlivé morfologické kategorie (slovní druh, mluvnické kategorie pro jména a slovesa). Příklad takového značkování je uveden v příkladu 5.1.

[7] http://ufal.mff.cuni.cz/morce/download.php

```
diplomaty     NNMP4-----A----
přistavuje    VB-S---3P-AA---
```

Příklad 5.1: Morfologické tagy prvního typu, převzato ze vzorových dat pro PDT 2.0 [5].

Druhý přístup je používán např. v morfologickém analyzátoru ajka vyvinutého na FI MU. Zde je každá kategorie reprezentována dvojicí znaků, kdy první z nich určuje kategorii a druhý kóduje její hodnotu. Příklad 5.2 ukazuje tento typ přístupu k morfologickému značkování.

```
diplomaty     k1gMnPc4
přistavuje    k5eAaImIp3nS
```

Příklad 5.2: Morfologické tagy druhého typu, vytvořeno pomocí online nástroje[8].

V obou uvedených typech morfologických značek můžeme bez předchozího vysvětlení rozpoznat po krátkém rozboru základní mluvnické kategorie. Slovo `diplomaty` z předchozích příkladů lze tedy identifikovat jako podstatné jméno (`N`, `k1`), mužského rodu (`M`, `gM`), množného čísla (`P`, `nP`), 4.pádu (`4`, `c4`). Podobně také pro slovo `přistavuje` – sloveso (`V`, `k5`), jednotného čísla (`S`, `nS`), 3.osoby (`3`, `p3`).

Analytická rovina

V PDT 2.0 je věta na analytické rovině reprezentována orientovaným stromem s kořenem, ohodnocenými hranami a uzly. Každý prvek morfologické roviny odpovídá právě jednomu uzlu stromu a závislostní vztah mezi dvěma slovními jednotkami je vyjádřen hranou mezi příslušnými dvěma uzly, což jasně definuje, že se jedná o závislostní přístup ke zpracování přirozeného jazyka. Typ vztahu je dán funkčním ohodnocením hrany – většina z nich reprezentuje závislostní vztah, ostatní odrážejí různé další lingvistické či technické jevy (koordinaci, apozici, interpunkci apod.). Zaznamenáno je i lineární uspořádání uzlů, které odpovídá pořadí slovních jednotek ve větě, což umožňuje přehledné grafické zobrazení stromu.

Tektogramatická rovina

Tektogramatická reprezentace věty zachycuje informace z oblastí tektogramatické struktury a funktorů, aktuálního členění a koreference. Zde strom zachycuje hloubkovou strukturu věty. Uzly zastupují pouze plnovýznamová slova, hrany stromu reprezentují vztah mezi uzly.

Pro účely toho projektu jsou opravdu důležité jen první dvě roviny anotace, zejména potom ta morfologická, která poskytuje základ pro správnou syntaktickou analýzu.

5.1.2 Formát dat

Tato podkapitola je volně převzata z [5].

Formáty dat pro korpusy českého jazyka jsou buď založeny na značkovacích jazycích XML nebo SGML, nebo jsou zdrojová data označkována přímo nimi. Zde uvedeme formáty používané v PDT. Některé z nich jsou využívány i jinými korpusy. Většina formátů je mezi sebou pomocí vhodných nástrojů vzájemně převoditelná.

- *CSTS (Czech sentence tree structure)* – formát založený na SGML, který může reprezentovat jen morfologickou a analytickou anotaci. V dnešní době už je spíše zastaralý.

[8] http://nlp.fi.muni.cz/projekty/wwwajka/WwwAjkaSkripty/morph.cgi?jazyk=0

- *FS (feature structure)* – formát souborů pro reprezentaci stromů, jejichž uzly popisují struktury atribut – hodnota. Podobně jako XML či SGML může být chápán jako „meta formát“.
- *PML (Prague Markup Language)* – formát dat založený na XML, navržený pro reprezentaci bohaté lingvistické anotace textů (morfologické značkování, závislostní stromy apod.). PML je probíhající projekt ve své rané fázi. Jednotlivé oddělené roviny anotace se mohou překrývat a mohou být konzistentně propojeny jak mezi sebou, tak i s dalšími zdroji dat. Každá rovina anotace je popsána v souboru *PML schéma*, který formalizuje abstraktní anotační schéma pro konkrétní rovinu anotace. PML schéma popisuje, které elementy se na dané rovině vyskytují, jak jsou spojovány, vnořovány a strukturovány, hodnoty jakého typu se v nich mohou vyskytovat a jakou roli hrají v anotačním schématu.

5.2 Výpočet pravděpodobnosti věty

Mějme jednoduchou frázovou gramatiku vytvořenou podle definice 3.1, která se skládá ze slovníku a vlastní gramatiky. Slovník je tvořen pravidly, která na pravé straně obsahují pouze terminální symboly, tedy jednotlivá slova jazyka. Tato gramatika je uvedena v tabulce 5.1.

$S \rightarrow NP\ \ VP$	$N \rightarrow people$
$VP \rightarrow V\ \ NP$	$N \rightarrow fish$
$VP \rightarrow V\ \ NP\ \ PP$	$N \rightarrow tanks$
$NP \rightarrow NP\ \ NP$	$N \rightarrow rods$
$NP \rightarrow NP\ \ PP$	$V \rightarrow people$
$NP \rightarrow N$	$V \rightarrow fish$
$NP \rightarrow \varepsilon$	$V \rightarrow tanks$
$PP \rightarrow P\ \ NP$	$P \rightarrow with$

Tabulka 5.1: Frázová gramatika [3].

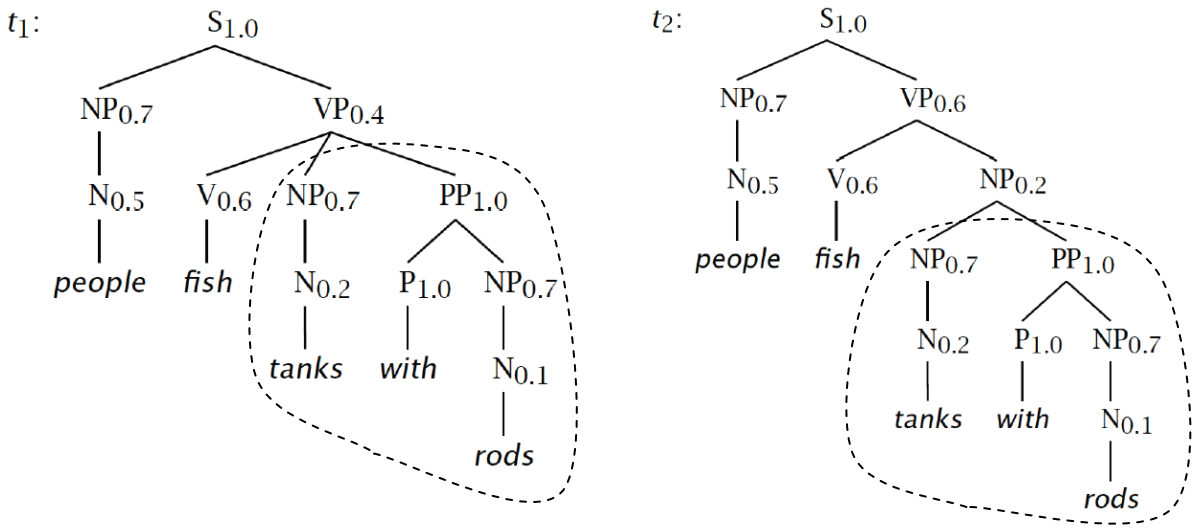
Z uvedené frázové gramatiky vytvoříme doplněním pravděpodobností k jednotlivým pravidlům pravděpodobnostní bezkontextovou gramatiku, která je uvedena v tabulce 5.2. Pro větší jednoznačnost gramatiky jsme také odstranili epsilon pravidlo $NP \rightarrow \varepsilon$. Hodnoty pravděpodobností jednotlivých pravidel slovníku jsou statisticky určeny na základě četnosti výskytu slova (terminálu) v korpusu v daném tvaru a kategorii (předterminálním symbolu) v porovnání s celkovou frekvencí výskytu slov dané kategorie. Hodnoty pravděpodobností přepisovacích pravidel vlastní gramatiky také vycházejí z četnosti použití pravidla při derivaci daného nonterminálu. Proto bývá toto odvětví také nazýváno statistické zpracování přirozeného jazyka.

$S \rightarrow NP\ \ VP$	1,0	$N \rightarrow people$	0,5
$VP \rightarrow V\ \ NP$	0,6	$N \rightarrow fish$	0,2
$VP \rightarrow V\ \ NP\ \ PP$	0,4	$N \rightarrow tanks$	0,2
$NP \rightarrow NP\ \ NP$	0,1	$N \rightarrow rods$	0,1
$NP \rightarrow NP\ \ PP$	0,2	$V \rightarrow people$	0,1
$NP \rightarrow N$	0,7	$V \rightarrow fish$	0,6
$PP \rightarrow P\ \ NP$	1,0	$V \rightarrow tanks$	0,3
		$P \rightarrow with$	1,0

Tabulka 5.2: Pravděpodobnostní bezkontextová gramatika [3].

Nyní můžeme na základě hodnot pravděpodobností jednotlivých pravidel počítat následující hodnoty pravděpodobností vztahující se k větě jako celku:

- $P(t)$ - pravděpodobnost derivačního stromu t, vypočteme jako součin pravděpodobností všech přepisovacích pravidel použitých ke generování daného derivačního stromu,
- $P(s)$ - pravděpodobnost řetězce (věty) s je vyjádřena jako součet pravděpodobností všech derivačních stromů, jejichž konstrukcí dostaneme daný řetězec.



Obrázek 5.1: Derivační stromy pro řetězec $s =$ *people fish tanks with rods*. Převzato z [3].

Pro vzorový řetězec $s =$ *people fish tanks with rods* a jeho dva derivační stromy t_1 a t_2 uvedené na obrázku 5.1 vypočteme pravděpodobnosti $P(s)$, $P(t_1)$ a $P(t_2)$. Postup výpočtu je uveden v příkladu 5.3.

$$s = people\ fish\ tanks\ with\ rods$$
$$P(t_1) = 1{,}0 \cdot 0{,}7 \cdot 0{,}4 \cdot 0{,}5 \cdot 0{,}6 \cdot 0{,}7 \cdot 1{,}0 \cdot 0{,}2 \cdot 1{,}0 \cdot 0{,}7 \cdot 0{,}1 = 0{,}0008232$$
$$P(t_2) = 1{,}0 \cdot 0{,}7 \cdot 0{,}6 \cdot 0{,}5 \cdot 0{,}6 \cdot 0{,}2 \cdot 0{,}7 \cdot 1{,}0 \cdot 0{,}2 \cdot 1{,}0 \cdot 0{,}7 \cdot 0{,}1 = 0{,}00024696$$
$$P(s) = P(t_1) + P(t_2) = 0{,}0008232 + 0{,}00024696 = 0{,}00107016$$

Příklad 5.3: Výpočet pravděpodobnosti zadaného řetězce a derivačních stromů [3].

Na obrázku 5.1 můžeme také pozorovat rozdílné napojení vyznačených podstromů, ve kterých se oba derivační stromy liší. Čárkovaně vyznačené podstromy derivačního stromu t_1 jsou přímo

závislé na slovese v tzv. slovesné vazbě, v druhém případě jsou podstromy syntaktického stromu t_2 vázány na podstatné jméno v tzv. jmenné vazbě. Tyto vazbu také určují různé vnímání jednotlivých frází ve větě.

5.3 Převod gramatiky do Chomského normální formy

V této podkapitole prakticky aplikujeme algoritmy pro transformace bezkontextových gramatik definované v podkapitole 2.6 tak, abychom na výstupu posloupnosti potřebných kroků obdrželi gramatiku v Chomského normální formě.

Mějme tedy frázovou gramatiku definovanou v tabulce 5.1. Na vstupu algoritmu pro převod bezkontextové gramatiky do Chomského normální formy je požadována vlastní gramatika bez jednoduchých pravidel. Tato gramatika však obsahuje jak ε-pravidlo, tak také jednoduché pravidlo, která chceme z gramatiky odstranit. Nejdříve z gramatiky eliminujeme ε-pravidlo $NP \to \varepsilon$, k čemuž využijeme algoritmus 2.2. Slovně lze také tuto transformaci gramatiky popsat tak, že najdeme všechna pravidla, která mají na pravé straně nonterminál, který je použit na levé straně ε-pravidla (v našem případě tedy NP). Pro každé takto nalezené pravidlo přidáme poté do množiny pravidel nové, které z pravé strany daného pravidla vynechá nonterminál použitý v ε-pravidle. Simulujeme tak přepsání tohoto nonterminálu prázdným řetězcem. Gramatika po odstranění ε-pravidel je uvedena v tabulce 5.3.

$S \to NP\ \ VP$	$NP \to NP\ \ NP$	$N \to people$
$S \to VP$	$NP \to NP$	$N \to fish$
$VP \to V\ \ NP$	$NP \to NP\ \ PP$	$N \to tanks$
$VP \to V$	$NP \to PP$	$N \to rods$
$VP \to V\ \ NP\ \ PP$	$NP \to N$	$V \to people$
$VP \to V\ \ PP$	$PP \to P\ \ NP$	$V \to fish$
	$PP \to P$	$V \to tanks$
		$P \to with$

Tabulka 5.3: Frázová gramatika po odstranění ε-pravidel. Podle [3].

Transformací gramatiky jsme do ní zavedli nová jednoduchá pravidla, pro jejichž odstranění použijeme algoritmus 2.3. Podobně jako výše lze postup popsat zjednodušeně, postupujeme rekurzivně od kořenového nonterminálu až k pravidlům s terminály na pravé straně. Pro nonterminál na pravé straně unárního pravidla najdeme všechna pravidla s tímto nonterminálem na levé straně. Do množiny pravidel následně přidáme nová pravidla vytvořená z levé strany unárního pravidla a z pravé strany nalezených pravidel, čímž opět simulujeme provedení daného unárního přepisovacího pravidla. Samotné unární pravidlo z množiny odstraníme, provádíme do eliminace všech jednoduchých pravidel z gramatiky. Tabulka 5.4 obsahuje gramatiku po odstranění unárních pravidel.

$S \rightarrow NP\ \ VP$	$S \rightarrow people$	$NP \rightarrow people$
$S \rightarrow V\ \ NP$	$S \rightarrow fish$	$NP \rightarrow fish$
$S \rightarrow V\ \ NP\ \ PP$	$S \rightarrow\ tanks$	$NP \rightarrow\ tanks$
$S \rightarrow V\ \ PP$	$VP \rightarrow people$	$NP \rightarrow rods$
$VP \rightarrow V\ \ NP$	$VP \rightarrow fish$	$NP \rightarrow with$
$VP \rightarrow V\ \ NP\ \ PP$	$VP \rightarrow\ tanks$	$PP \rightarrow with$
$VP \rightarrow V\ \ PP$	$V \rightarrow people$	$P \rightarrow with$
$NP \rightarrow NP\ \ NP$	$V \rightarrow fish$	
$NP \rightarrow NP\ \ PP$	$V \rightarrow\ tanks$	
$NP \rightarrow P\ \ NP$		
$PP \rightarrow P\ \ NP$		

Tabulka 5.4: Frázová gramatika po odstranění jednoduchých pravidel. Podle [3].

Výsledkem této transformace je námi požadovaná vlastní gramatika bez jednoduchých pravidel a nyní můžeme pomocí algoritmu 2.1 gramatiku transformovat do Chomského normální formy. V gramatice vidíme dvě pravidla, která mají tři nonterminální symboly na pravé straně. Binarizaci těchto pravidel docílíme vytvořením pomocných nonterminálů (označujeme např. znakem ‚@'), kterými postupně od konce nahrazujeme dvojice nonterminálů pravé strany a zapisujeme pro ně nová binární pravidla. Provedením této transformace dostáváme bezkontextovou gramatiku v Chomského normální formě (tabulka 5.5), která je ekvivalentní s původní gramatikou z tabulky 5.1.

$S \rightarrow NP\ \ VP$	$NP \rightarrow NP\ \ NP$	$V \rightarrow people$
$S \rightarrow V\ \ NP$	$NP \rightarrow NP\ \ PP$	$V \rightarrow fish$
$S \rightarrow V\ @S\_V$	$NP \rightarrow P\ \ NP$	$V \rightarrow\ tanks$
$@S\_V \rightarrow NP\ \ PP$	$PP \rightarrow P\ \ NP$	$NP \rightarrow people$
$S \rightarrow V\ \ PP$	$S \rightarrow people$	$NP \rightarrow fish$
$VP \rightarrow V\ \ NP$	$S \rightarrow fish$	$NP \rightarrow\ tanks$
$VP \rightarrow V\ @VP\_V$	$S \rightarrow\ tanks$	$NP \rightarrow rods$
$@VP\_V \rightarrow NP\ \ PP$	$VP \rightarrow people$	$NP \rightarrow with$
$VP \rightarrow V\ \ PP$	$VP \rightarrow fish$	$PP \rightarrow with$
	$VP \rightarrow\ tanks$	$P \rightarrow with$

Tabulka 5.5: Frázová gramatika v Chomského normální formě. Podle [3].

5.4 CKY syntaktická analýza

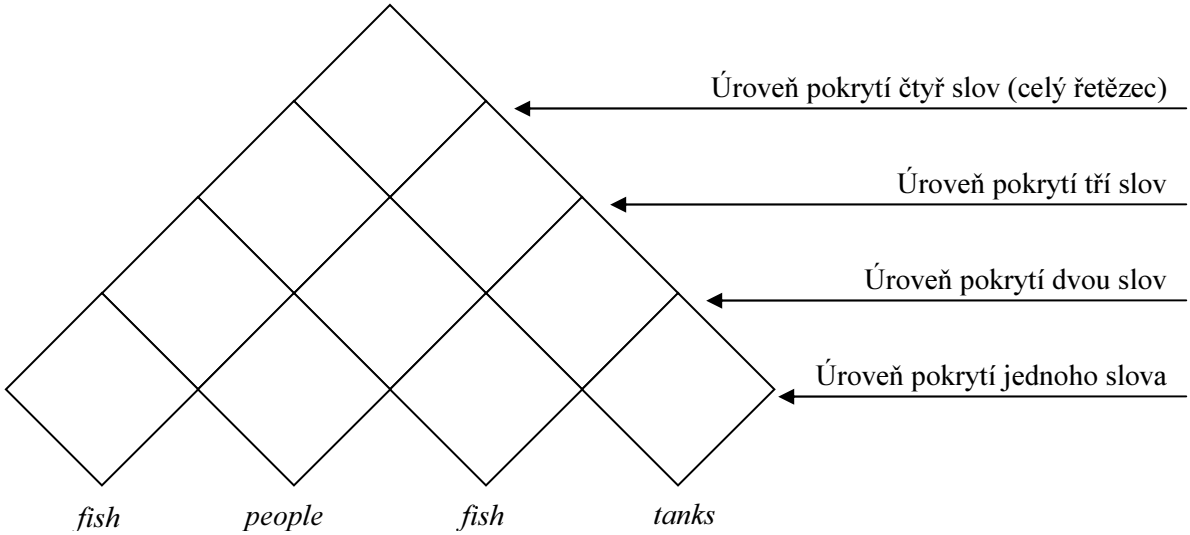
V této podkapitole budeme demonstrovat práci rozšířeného CKY algoritmu zpracovávající pravděpodobnostní bezkontextovou gramatiku uvedeného v podkapitole 4.7. Vzhledem k tomu, že uvedená varianta algoritmu nevyžaduje na vstupu striktně gramatiku v Chomského normální formě, ale zpracovává i jednoduchá (unární) pravidla, vyjdeme z bezkontextové gramatiky definované v tabulce 5.3. Binarizace je ovšem stále klíčovým prvkem algoritmu pro dodržení kubické složitosti, proto pravidla s více než dvěma nonterminály na pravé straně transformujeme pomocí algoritmu 2.1.

Po doplnění pravděpodobností k jednotlivým pravidlům (hodnoty jsou odhadnuty pro účely příkladu) dostáváme pravděpodobnostní bezkontextovou gramatiku v tabulce 5.6.

$S \rightarrow NP\ \ VP$	0,9	$N \rightarrow people$	0,5
$S \rightarrow VP$	0,1	$N \rightarrow fish$	0,2
$VP \rightarrow V\ \ NP$	0,5	$N \rightarrow\ tanks$	0,2
$VP \rightarrow V$	0,1	$N \rightarrow rods$	0,1
$VP \rightarrow V\ @VP\_V$	0,3	$V \rightarrow people$	0,1
$VP \rightarrow V\ \ PP$	0,1	$V \rightarrow fish$	0,6
$@VP\_V \rightarrow NP\ \ PP$	1,0	$V \rightarrow\ tanks$	0,3
$NP \rightarrow NP\ \ NP$	0,1	$P \rightarrow with$	1,0
$NP \rightarrow NP\ \ PP$	0,2		
$NP \rightarrow N$	0,7		
$PP \rightarrow P\ \ NP$	1,0		

Tabulka 5.6: (Binární) pravděpodobnostní bezkontextová gramatika. Podle [3].

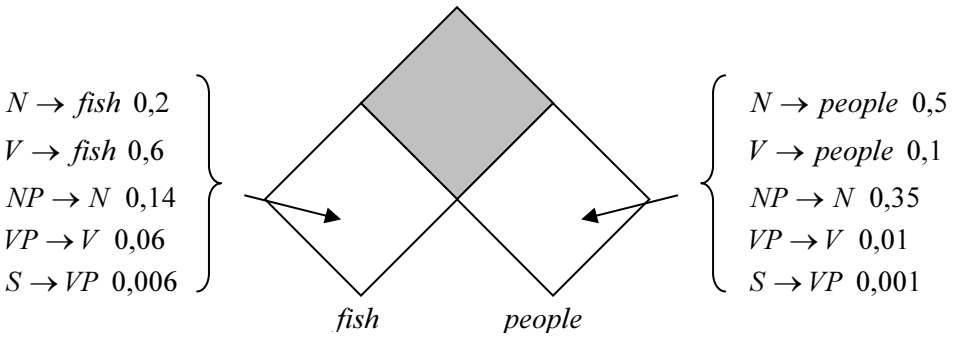
Uvažujme tedy vzorový řetězec $s = fish\ people\ fish\ tanks$. Cílem syntaktické analýzy tohoto vstupního řetězce je konstrukce derivačního stromu podle pravděpodobnostní bezkontextové gramatiky uvedené v tabulce 5.6. Zjišťujeme, zda-li lze takový derivační strom zkonstruovat, a pokud ano a je vytvořeno stromů více, požadujeme na výstupu ten s největší pravděpodobností. Při průběhu algoritmu s výhodou využíváme tabulku (chart), která je vhodná i pro jeho grafickou demonstraci. Rozšířený CKY algoritmus definovaný v podkapitole 4.7 pracuje s indexy v tabulce jiným způsobem oproti základnímu CKY algoritmu uvedenému v podkapitole 4.2. Tato úprava nemá na algoritmus prakticky žádný dopad, liší se pouze způsob ukládání do tabulky, a vidíme tedy, že implementace se mohou v různých variantách mírně odlišovat. Tabulka pokrytí pro vzorový řetězec je uvedena na obrázku 5.2. Čtverce na nejnižší úrovni popisují terminální symboly a budou v nich uvedena pravidla, která nad nimi můžeme přepsat. Čtverce ve vyšších úrovních následně pokrývají čtverce pod nimi, dohromady obrazně vytvářejí trojúhelník.



Obrázek 5.2: Tabulka pokrytí vstupního řetězce. Podle [3].

5.4.1 Základní krok rozšířeného CKY algoritmu

Na jedné úrovni si nyní ukážeme výpočet pravděpodobností (aplikace Viterbiho algoritmu z podkapitoly 4.7.1) pro dvou-slovní frázi vstupního řetězce *fish people*. Tento výpočet analogicky prováděný pro vyplnění celé tabulky je jádrem rozšířeného CKY algoritmu. Po inicializačních krocích včetně aplikace unárních pravidel (bude popsáno podrobněji níže) vypadá tabulka pro tuto dvojici následovně:



Obrázek 5.3: Stav tabulky po inicializačních krocích pro dvojici slov *fish people*. Podle [3].

Do čtverce označeného šedou barvou nyní dopočteme pravděpodobnosti pravidel, která je možné pro dvojici o úroveň níže aplikovat. Hlavní smyčka algoritmu nejprve hledá všechna binární pravidla tak, že levý „synovský“ čtverec odpovídá prvnímu nonterminálu pravé strany a pravý „synovský“ čtverec odpovídá druhému nonterminálu pravé strany přepisovacího pravidla. Pro vzorovou situaci výše tedy můžeme z gramatiky v tabulce 5.6 použít níže uvedená pravidla a jejich pravděpodobnosti vypočítáme jako součin pravděpodobností nonterminálů pravé strany pravidla (uvedené v tabulce – označené jako P_L a P_R pro levý a pravý nonterminál) a pravděpodobnosti daného pravidla.

$NP \rightarrow NP\ NP$ $\quad P = P_L(NP \rightarrow N) \cdot P_R(NP \rightarrow N) \cdot P(NP \rightarrow NP\ NP) = 0{,}14 \cdot 0{,}35 \cdot 0{,}1 = 0{,}0049$

$VP \rightarrow V\ NP$ $\quad P = P_L(V \rightarrow fish) \cdot P_R(NP \rightarrow N) \cdot P(VP \rightarrow V\ NP) = 0{,}6 \cdot 0{,}35 \cdot 0{,}5 = 0{,}105$

$S \rightarrow NP\ VP$ $\quad P = P_L(NP \rightarrow N) \cdot P_R(VP \rightarrow V) \cdot P(S \rightarrow NP\ VP) = 0{,}14 \cdot 0{,}01 \cdot 0{,}9 = 0{,}00126$

Ve druhém kroku hlavní smyčky jsou nad těmito pravidly aplikována jednoduchá pravidla (je-li to možné). V našem případě můžeme použít pravidlo $S \rightarrow VP$, pro které vypočteme pravděpodobnost jako součin pravděpodobností použitého pravidla a pravidla ze šedého čtverce přepisujícího nonterminál na pravé straně daného unárního pravidla.

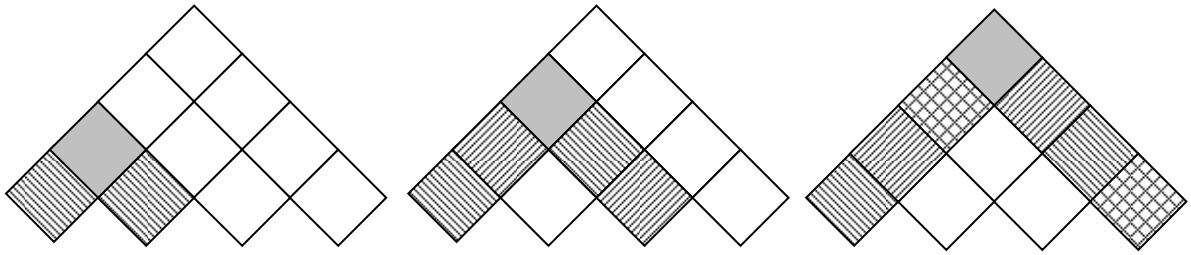
$$S \rightarrow VP \qquad P = P(VP \rightarrow V\ NP) \cdot P(S \rightarrow VP) = 0{,}105 \cdot 0{,}1 = 0{,}0105$$

Nyní máme ale ve stejné buňce tabulky dvě pravidla přepisující stejný nonterminál. Protože chceme na výstupu derivační strom s nejvyšší pravděpodobností, uložíme na dané pozici pouze pravidlo s vyšší pravděpodobností, čímž tuto kolizi odstraníme.

Vrátíme-li se zpět k inicializační fázi algoritmu, vyplňujeme v ní buňky tabulky na nejnižší úrovni. Nejprve vepíšeme pravidla s pravděpodobnostmi, která mají na pravé straně terminální symbol pokrytý daným místem. Následně analogicky jako výše aplikujeme na danou buňku použitelná unární pravidla. Pro podřetězec *fish* na začátku zpracovávaného řetězce budou tedy v tabulce zapsána následující pravidla:

$N \rightarrow fish$	$P = 0{,}2$
$V \rightarrow fish$	$P = 0{,}6$
$NP \rightarrow N$	$P = P(N \rightarrow fish) \cdot P(NP \rightarrow N) = 0{,}2 \cdot 0{,}7 = 0{,}14$
$VP \rightarrow V$	$P = P(V \rightarrow fish) \cdot P(VP \rightarrow V) = 0{,}6 \cdot 0{,}1 = 0{,}06$
$S \rightarrow VP$	$P = P(VP \rightarrow V) \cdot P(S \rightarrow VP) = 0{,}06 \cdot 0{,}1 = 0{,}006$

Po doplnění pravidel a pravděpodobností na nejnižší úrovni postupujeme postupně vždy o úroveň výše a provádíme základní krok algoritmu popsaný v podkapitole 5.4.1. Zde je důležité opět zdůraznit význam binarizace. Protože hlavní smyčka algoritmu uvažuje binární pravidla gramatiky (prakticky by měl algoritmus pracovat pouze s gramatikou v Chomského normální formě, unární pravidla jsou pouze rozšířením), potřebujeme při zapisování pravidel do buňky a výpočtu pravděpodobností zpracovávat pravděpodobnosti právě ze dvou zdrojových buněk. Pro pokrytí třech a více slov tedy musíme uvažovat vhodné rozdělení, abychom obsáhli celý zpracovávaný podřetězec. Na obrázku 5.4 jsou pro různé úrovně pokrytí graficky znázorněny zdrojové buňky používané při výpočtu – pro zapsání pravidel a pravděpodobností do šedých (plných) buněk jsou uvažovány vždy dvojice stejně označených buněk. Například tedy pro výpočet pravidel na nejvyšší úrovni pokrývají dvojice buněk vždy celý vstupní řetězec (uvažujme vzorový) – mřížkovaná buňka vlevo část *fish people fish* a její druhá část do dvojice podřetězec *tanks*.



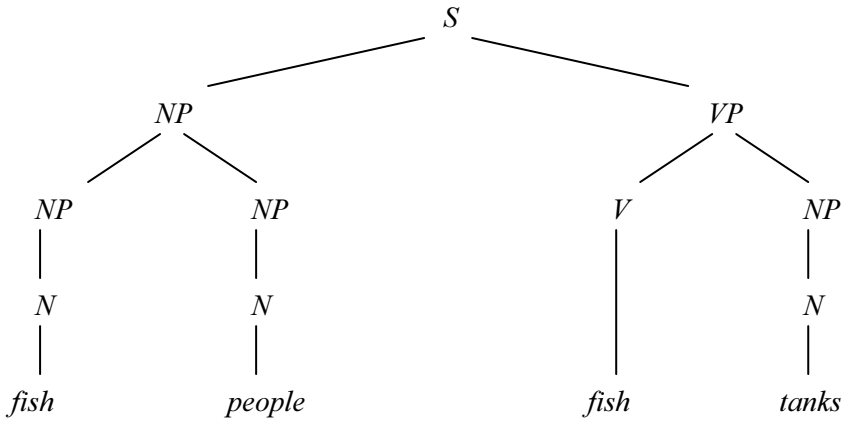
Obrázek 5.4: Zdrojové buňky pro výpočet pravděpodobností.

Vyplněná tabulka po ukončení běhu rozšířeného CKY algoritmu je uvedená v tabulce 5.7 (kvůli přehlednému vepisování je pootočena o 45° doprava).

$\boldsymbol{N \rightarrow fish}$ 0,2 $V \rightarrow fish$ 0,6 $\boldsymbol{NP \rightarrow N}$ 0,14 $VP \rightarrow V$ 0,06 $S \rightarrow VP$ 0,006	$NP \rightarrow NP\ NP$ 0,0049 $VP \rightarrow V\ NP$ 0,105 $S \rightarrow VP$ 0,0105	$\boldsymbol{NP \rightarrow NP\ NP}$ $0{,}686 \cdot 10^{-4}$ $VP \rightarrow V\ NP$ 0,00147 $S \rightarrow NP\ VP$ 0,000882	$NP \rightarrow NP\ NP$ $0{,}9604 \cdot 10^{-6}$ $VP \rightarrow V\ NP$ 0,00002058 $\boldsymbol{S \rightarrow NP\ VP}$ 0,00018522
	$\boldsymbol{N \rightarrow people}$ 0,5 $V \rightarrow people$ 0,1 $\boldsymbol{NP \rightarrow N}$ 0,35 $VP \rightarrow V$ 0,01 $S \rightarrow VP$ 0,001	$NP \rightarrow NP\ NP$ 0,0049 $VP \rightarrow V\ NP$ 0,007 $S \rightarrow NP\ VP$ 0,0189	$NP \rightarrow NP\ NP$ $0{,}686 \cdot 10^{-4}$ $\boldsymbol{VP \rightarrow V\ NP}$ 0,000098 $S \rightarrow NP\ VP$ 0,01323
		$N \rightarrow fish$ 0,2 $\boldsymbol{V \rightarrow fish}$ 0,6 $NP \rightarrow N$ 0,14 $VP \rightarrow V$ 0,06 $S \rightarrow VP$ 0,006	$NP \rightarrow NP\ NP$ 0,00196 $VP \rightarrow V\ NP$ 0,042 $S \rightarrow VP$ 0,0042
			$N \rightarrow \boldsymbol{tanks}$ 0,2 $V \rightarrow tanks$ 0,1 $\boldsymbol{NP \rightarrow N}$ 0,14 $VP \rightarrow V$ 0,03 $S \rightarrow VP$ 0,003

Tabulka 5.7: Tabulka po dokončení běhu CKY algoritmu. Podle [3].

Posledním krokem celé syntaktické analýzy je rekonstrukce nejpravděpodobnějšího derivačního stromu z vypočtené tabulky. Společně s přepisovacími pravidly a jejich pravděpodobnostmi jsou v tabulce pro každé pravidlo uloženy i zdrojové buňky, které se uvažovaly při jeho výpočtu. Každý nonterminál na pravé straně pravidla tedy odkazuje na pravidlo, které se má dále použít při rekonstrukci derivačního stromu pro vstupní řetězec. Tímto zpětným průchodem získáme nejpravděpodobnější derivační strom, který je pro náš vstupní řetězec *fish people fish tanks* uveden na obrázku 5.5. Pravidla použitá pro vytvoření tohoto derivačního stromu jsou v tabulce 5.7 zvýrazněna tučně.

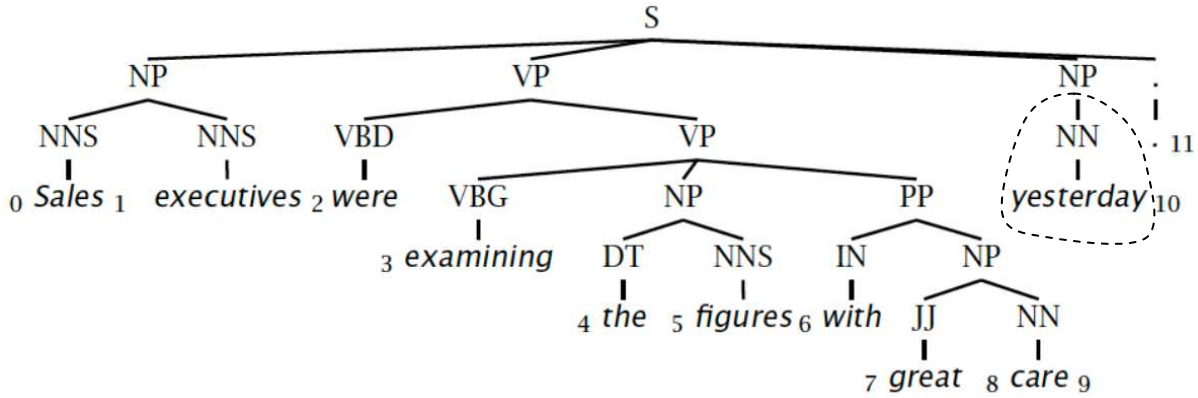


Obrázek 5.5: Nejpravděpodobnější derivační strom získaný syntaktickou analýzou pomocí rozšířeného CKY algoritmu.

Pravděpodobnost tohoto derivačního stromu je v tabulce uložena u počátečního pravidla v buňce pokrývající celý vstupní řetězec. Výpočet můžeme také ověřit postupem popsaným v podkapitole 5.2, kdy vynásobíme pravděpodobnosti jednotlivých pravidel použitých pro konstrukci derivačního stromu, což prakticky provádíme i při průběhu samotného CKY algoritmu.

5.5 Vyhodnocení syntaktické analýzy

Úspěšnost syntaktické analýzy pro frázový přístup ke stavbě věty udáváme pomocí parametrů *precision*, *recall* a *F measure*, které jsme definovali a popsali v podkapitole 4.8. Proces vyhodnocení úspěšnosti je založen na porovnání dvojic derivačních stromů – tzv. *gold standard tree*, který reprezentuje očekávaný výsledek, a (nejpravděpodobnějšího) derivačního stromu, který jsme získali jako výstup syntaktické analýzy (např. výše demonstrovaného rozšířeného CKY algoritmu). Při výpočtu s výhodou využíváme *závorkové notace*, kterou lze derivační strom také popisovat.



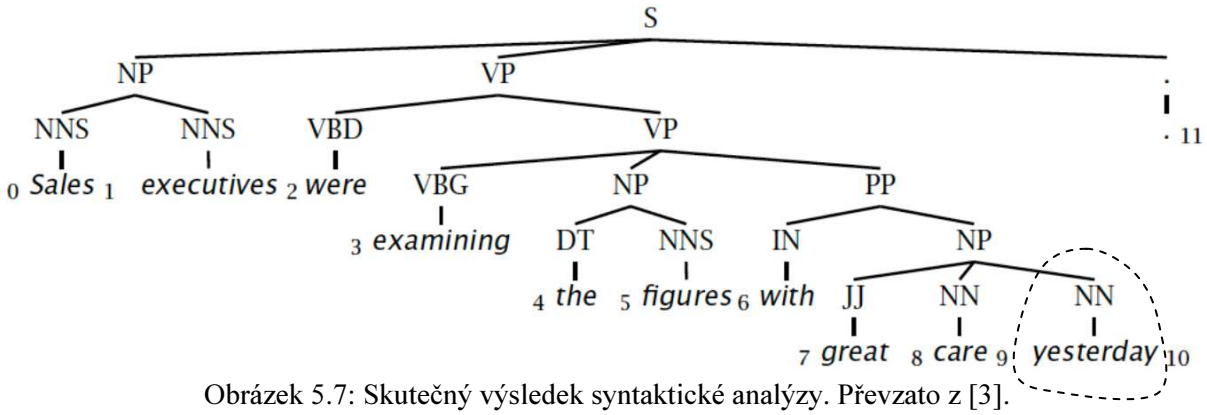
Obrázek 5.6: Očekávaný výsledek syntaktické analýzy (gold standard tree). Převzato z [3].

Mějme tedy demonstrační řetězec (větu) *„Sales executives were examining the figures with great care yesterday.“* a její očekávaný derivační strom, který je znázorněn na obrázku 5.6. Při tvorbě zápisu derivačního stromu v závorkové notaci používáme tzv. *zarážky* (fencepost). Jedná se o číselné označení oddělovačů jednotlivých terminálů věty, abychom jednoznačně dokázali popsat, které terminální symboly daný nonterminál pokrývá (na obrázku 5.6 je vidíme jako čísla před a za listy derivačního stromu – jednotlivými slovy). Při tvorbě závorek postupujeme stromem například shora dolů a zleva doprava (podobně jako při prohledávání do šířky) a zapisujeme k jednotlivým nonterminálům jejich pokrytí vstupního řetězce. Do zápisu již neuvádíme pravidla přepisující nonterminální symboly (někdy také označované jako předterminální symboly) na terminály. Derivační strom na obrázku 5.6 potom zapíšeme v závorkové notaci následovně:

***S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6:9), NP-(7:9), NP-(9:10)*

Na obrázku 5.7 je zobrazen derivační strom, který jsme (například) získali provedením syntaktické analýzy nad vzorovou větou. Jeho závorkový zápis vypadá potom takto:

***S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6:10), NP-(7:10)*



Obrázek 5.7: Skutečný výsledek syntaktické analýzy. Převzato z [3].

Nyní můžeme tyto dva derivační stromy porovnat a vypočítat úspěšnost syntaktické analýzy, kterou určují výše uvedené parametry. Parametr *precision P* udává, kolik závorek z celkového počtu výstupu syntaktické analýzy je napárováno na závorky očekávaného derivačního stromu (v závorkových zápisech výše označeno tučně).

$$P = \frac{3}{7} = 0{,}429 = 42{,}9\%$$

Parametr *recall R* poté vypočteme jako počet závorek z očekávaného derivačního stromu, které jsou obsaženy ve výstupu syntaktické analýzy (opět označeno výše tučně).

$$R = \frac{3}{8} = 0{,}375 = 37{,}5\%$$

Souhrnný ukazatel úspěšnosti syntaktické analýzy pro frázový přístup *F measure* následně vyčíslíme podle vzorce uvedeného v podkapitole 4.8.

$$F = \frac{2PR}{(R+P)} = \frac{2 \cdot 0{,}429 \cdot 0{,}375}{(0{,}429 + 0{,}375)} = 0{,}4 = 40\%$$

Výpočet úspěšnosti syntaktické analýzy, který jsme právě demonstrovali, bývá také někdy nazýván jako *labeled* (označovaný). Pojmenování vychází z toho, že při porovnávání závorek bereme v úvahu i nonterminály, které označují pokrytí dané části vstupního řetězce. Existuje i varianta výpočtu, která pracuje pouze s číselným pokrytím vstupu a nezajímá se o nonterminál, který danou část vstupu reprezentuje.

Uvedený příklad také ukazuje potencionální nevýhodu tohoto vyhodnocování, kdy je výsledná úspěšnost syntaktické analýzy relativně nízká, i když jsme špatně určili pouze jediný prvek celého derivačního stromu (ve stromech vyznačen čárkovaně). Toto je způsobeno tím, že podstrom pod nonterminálem *VP* poté pokrývá špatnou část řetězce a tato chyba se kaskádovitě propisuje pro všechny jeho podstromy, kterými procházíme až k chybně určenému nonterminálu. Tuto vlastnost vyhodnocování úspěšnosti ale při použití v kontextu zpracování přirozeného jazyka akceptujeme.

Pro část derivačního stromu, kterou jsme při výpočtu výše neuvažovali, můžeme vypočítat přesnost přiřazení kategorií (např. slovních druhů) jednotlivým slovům vstupního řetězce, v angličtině pojmenovanou jako *tagging accuracy.* Přesnost tedy vyhodnocujeme na úrovni přepisu předterminálnách symbolů na terminály. V uvedeném příkladu mají všechny terminální symboly na výstupu syntaktického analyzátoru přiřazeny kategorie podle očekávaného výsledku, přesnost označování je tedy 100%.

6 Implementace syntaktického analyzátoru

Cílem této diplomové práce je navrhnout a implementovat syntaktický analyzátor pro jazyk český. Teoretický rozbor jednotlivých přístupů a možností řešení jsme provedli v předchozích kapitolách tohoto textu, v této kapitole popíšeme právě implementaci syntaktického analyzátoru, jehož jádrem je rozšířený CKY algoritmus definovaný v podkapitole 4.7 a demonstrovaný v podkapitole 5.4. Pro implementaci jsme CKY algoritmus zvolili nejen z důvodu jeho vhodnosti pro práci s frázovými pravděpodobnostními gramatikami, ale také kvůli jeho rozšířené a dostupné dokumentaci v různých variantách zpracování.

V první části se budeme věnovat požadavkům na aplikaci, zvolíme přístup k vytvoření gramatiky a popíšeme vytvoření anotovaného vzorového korpusu českého jazyka. Následně na základě definovaných požadavků navrhneme aplikaci, detailněji rozebereme vlastní implementaci analyzátoru a poté popíšeme způsob testování programu a zhodnotíme jeho výsledky.

6.1 Požadavky na aplikaci

Obecný syntaktický analyzátor by měl na základě gramatiky rozhodnout, zda-li vstupní řetězec náleží, či nenáleží do jazyka generovaného danou gramatikou a v kladném případě potom vytvoří pro zpracovávaný vstup derivační strom, který vrátí jako výstup programu. V našem případě pracujeme s pravděpodobnostní bezkontextovou gramatikou, výsledkem analýzy tedy bude nejpravděpodobnější derivační strom. Způsob sestavení gramatiky potom dále ovlivňuje návrh aplikace.

6.1.1 Volba přístupu k tvorbě gramatiky

Základním určujícím faktorem pro formulaci požadavků na implementaci syntaktického analyzátoru je rozhodnutí o přístupu k vytvoření používané pravděpodobnostní bezkontextové gramatiky. Jak už jsme nastínili v podkapitole 4.6, v oboru zpracování přirozeného jazyka lze gramatiku pro syntaktický analyzátor definovat dvěma způsoby:

- Gramatika (včetně jejího slovníku) je vytvořena *manuálně před spuštěním syntaktické analýzy*, do které je následně předána jako jeden z jejích vstupů. Pro sestavení takové gramatiky pokrývající značnou část jazyka je v ideálním případě potřeba lingvista nebo alespoň expert na syntaxi jazyka. Pravděpodobnosti jednotlivých gramatických pravidel jsou na počátku nastaveny nejčastěji na základě zkušeností tvůrců gramatiky a následně iteračně upravovány do požadované podoby s využitím trénovaní množiny dat. Takto vytvořená gramatika bude nejspíše dosahovat přesnějších výsledků zejména pro úzce vymezené oblasti jazyka, její sestavení bude ovšem náročné a pracné.
- Pravděpodobnostní gramatika vzniká *automaticky až při běhu syntaktické analýzy*. Nutným vstupem analyzátoru je v tomto případě anotovaný korpus, který je programově zpracován – z označkovaných dat trénovaní množiny jsou získána gramatická pravidla, na základě jejichž četností jsou následně dopočítány a uloženy pravděpodobnosti jednotlivých pravidel. Tímto způsobem získáme spíše generickou gramatiku daného jazyka, jejíž tvorba je z programového hlediska mnohem přístupnější a lépe zpracovatelná.

Po zhodnocení obou výše uvedených přístupů jsme pro implementaci zvolili způsob automatické tvorby gramatiky v průběhu syntaktické analýzy (prakticky se gramatika sestavuje ještě před samotným během CKY algoritmu, může však být součástí stejného programu), který pro své strojové učení vyžaduje anotovaný korpus.

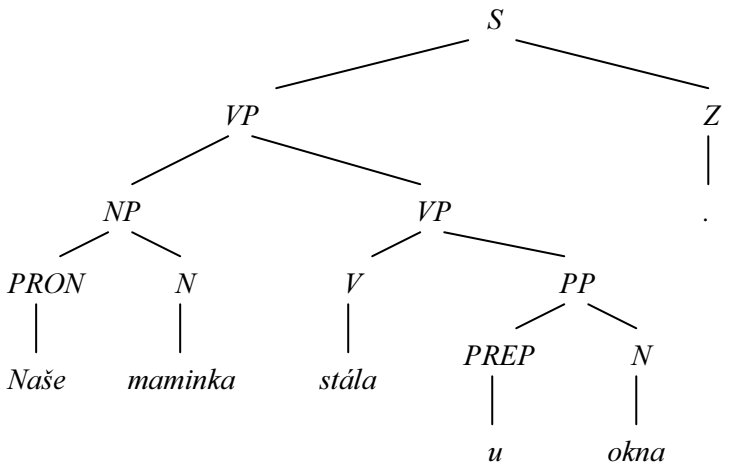
6.1.2 Anotovaný frázový korpus

Korpus českého jazyka anotovaný frázovým přístupem ke stavbě věty (například Brněnský frázový korpus uvedený v podkapitole 3.3.2) není ovšem veřejně k dispozici. Pro češtinu všeobecně existuje velmi malé množství frázově anotovaných korpusů. Tento jev je způsoben odlišným přístupem ke stavbě věty v českém jazyce (např. v porovnání s angličtinou), kdy syntaxe vět v češtině je spíše tvořena (a také vyučována) závislostmi mezi jednotlivými větnými členy věty či souvětí. Dokazuje to také i relativně velmi malý rozsah uvedeného Brněnského frázového korpusu, jehož desítky tisíc slov jsou v porovnání s jednotkami milionů slov závislostně anotovaných korpusů pouze malým zlomkem.

Pro účely této diplomové práce byl tedy manuálně vytvořen jednoduchý frázově anotovaný korpus českého jazyka, který poskytl dostatečný vzorek pro testování a ověření funkčnosti implementovaného syntaktického analyzátoru. Jako zdroj dat byla vybrána známá dětská knížka Honzíkova cesta [14], která patří mezi doporučené četby pro základní školy a obsahuje jednoduchou základní češtinu. Sestavený korpus se skládá ze 100 jednoduchých vět, které byly postupně ručně anotovány a uloženy do jednotlivých souborů ve formátu inspirovaném projektem The Penn Treebank[9]. Pro tak malý vzorek dat je také důležitá úzká slovní zásoba, abychom mohli analyzátor vhodným způsobem testovat. Výsledná závorková notace těchto vět reprezentuje binární stromy vytvořené nad množinou označkovaných dat. Příklad věty, výstup její anotace a grafická reprezentace jsou uvedeny v příkladu 6.1.

Naše maminka stála u okna.

```
(S
  (VP
    (NP
      (PRON:Naše)
      (N:maminka)
    )
    (VP
      (V:stála)
      (PP
        (PREP:u)
        (N:okna)
      )
    )
  )
  (C:.)
)
```



Příklad 6.1: Věta, výstup její anotace a grafické znázornění stromu.

[9] http://www.cis.upenn.edu/~treebank/home.html

Při anotaci vět vytváříme výsledný strom (prakticky jde o derivační strom) postupem zdola nahoru. Nejdříve označkujeme jednotlivá slova vstupní věty – přiřadíme jim jejich slovní druhy a následně je sdružujeme do frází (nejčastěji jmenných, slovesných či předložkových). Toto shlukování provádíme až do vytvoření jediné fráze (kořene stromu), která pokrývá celou zpracovávanou větu. V uloženém závorkovém formátu potom každý obsah závorky popisuje danou frázi, případně označkovaná slova věty, kdy po určeném slovním druhu následuje znak dvojtečky a dané slovo. Ve vytvořené budoucí gramatice potom závorka prakticky reprezentuje přepisovací pravidlo a pojmenování frází jednotlivé nonterminály. Slovník gramatiky (definice 3.1) je sestaven ze závorek obsahující slova věty. Značky použité při anotaci vzorových dat jsou uvedeny v tabulce 6.1.

	Značka	**Význam**
Předterminální symboly	N	Podstatné jméno
	ADJ	Přídavné jméno
	PRON	Zájmeno
	NUM	Číslovka
	V	Sloveso
	ADV	Příslovce
	PREP	Předložka
	CON	Spojka
	PART	Částice
	INT	Citoslovce
	Z	Interpunkce (ukončení věty)
	X	Univerzální nonterminál
Nonterminální symboly	NP	Jmenná fráze
	VP	Slovesná fráze
	PP	Předložková fráze
	S	Věta (kořen stromu)

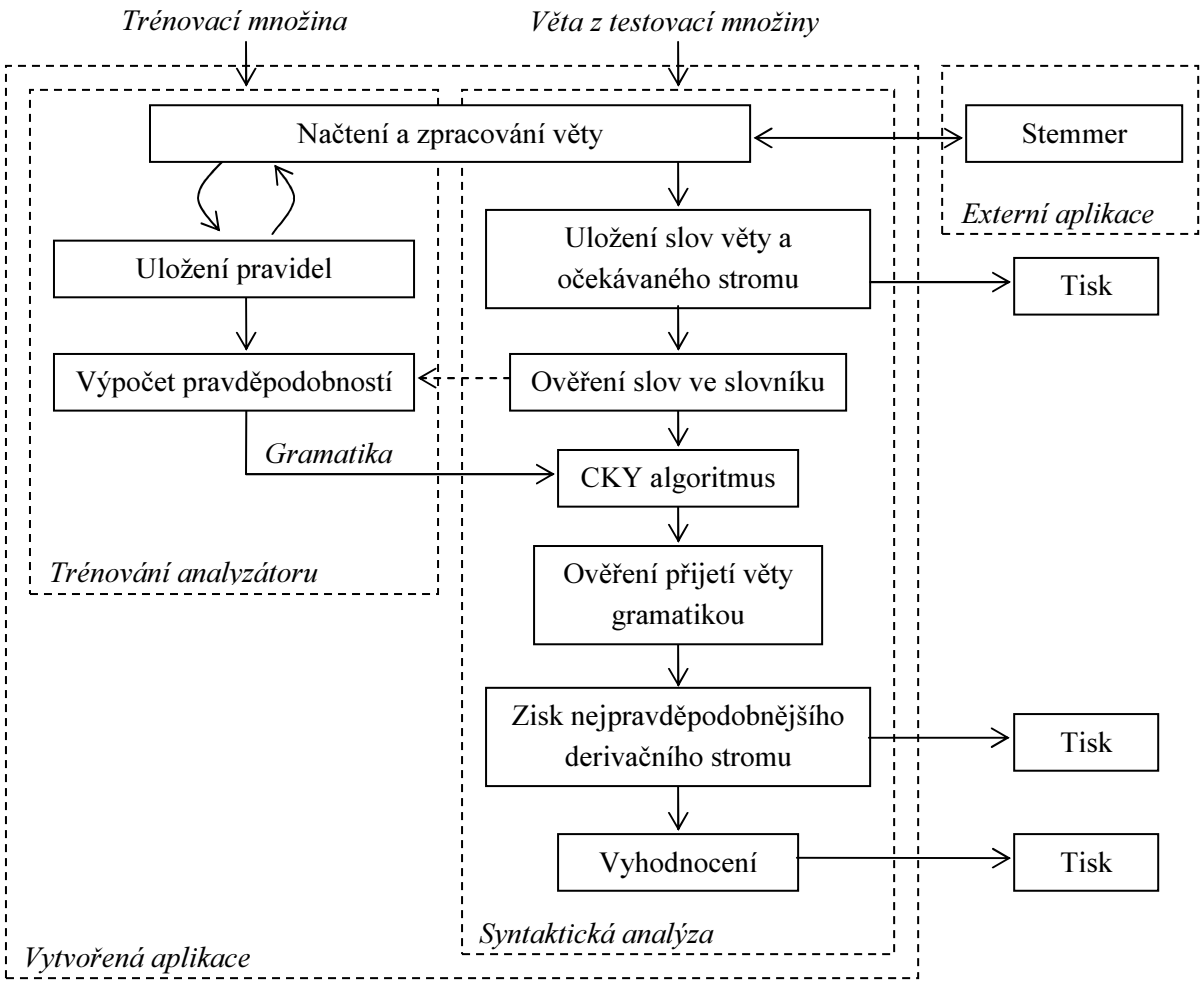
Tabulka 6.1: Značky použité při anotaci korpusu.

Shrneme-li krátce funkčnost požadované aplikace, analyzátor sestaví na základě anotované trénovací množiny dat pravděpodobnostní bezkontextovou gramatiku a poté pro testovací množinu dat vrátí nejpravděpodobnější derivační stromy. Výstupy syntaktické analýzy následně porovná s očekávanými výsledky a vyhodnotí úspěšnost analyzátoru.

6.2 Návrh syntaktického analyzátoru

V této podkapitole navrhneme jednotlivé části implementovaného programu a také popíšeme jejich vzájemnou součinnost.

Jak již napovídá podkapitola 6.1.1, aplikace bude zpracovávat vstupy ve dvou základních fázích – *trénování analyzátoru* a vlastní *syntaktická analýza*, které si blíže rozebereme níže. Celkové schéma struktury aplikace, které graficky znázorňuje logické celky programu, je uvedeno na obrázku 6.1.



Obrázek 6.1: Navržená struktura aplikace.

6.2.1 Načtení a zpracování vstupních dat

Společným prvkem obou částí je načtení a zpracování vstupních dat. Protože je každá anotovaná věta vytvořeného korpusu uložena v samostatném souboru, pracujeme s nimi postupně. Pro získávání požadovaných informací je využíván jednoduchý lexikální analyzátor, který v souboru identifikuje gramatické prvky (nonterminály a terminály seskupené do pravidel) a který úzce komunikuje s následujícími částmi v běhu jednotlivých fází programu.

Tento blok programové struktury také spolupracuje s externí aplikací, tzv. *stemmerem* [15], který byl vytvořen v rámci jiné diplomové práce na FIT VUT v Brně. Cílem stemmingu je určení základního tvaru daného slova (stemu), který se může a nemusí shodovat s kořenem slova, jenž bývá v českém jazyce často používán. Stem získáváme algoritmickým ořezáváním slova o přípony vyskytující se ve zpracovávaném jazyce.

V našem případě stemming s výhodou využíváme pro sdružování slov se stejným základem (často jsou to slova příbuzná), čímž zvyšujeme pokrytí jazyka syntaktickým analyzátorem. V češtině (např. v porovnání s angličtinou) je stemming potom mnohem důležitější, protože slova v českém jazyce skloňujeme a časujeme a při syntaktické analýze nejsou různé tvary a pády např. podstatných jmen žádoucí. Slova *„kanec“*, *„kanci“* a *„kancem“* mají společný stem *„kan“*, při syntaktické analýze tedy pracujeme pouze s jejich společným základním tvarem a nikoliv se třemi různými.

Potenciální nevýhodou stemmingu může být sdružení slov s jinými slovními druhy (vznikají vzájemným odvozováním) do stejné skupiny. Podstatné jméno *„práce“*, přídavné jméno *„pracující“* a sloveso *„pracovat“* tedy mohou být všechna označena stejným základem *„prac“*. Tento jev ale částečně potlačujeme právě použitím pravděpodobnostní bezkontextové gramatiky.

6.2.2 Trénování analyzátoru

Hlavním cílem fáze trénování analyzátoru je vytvoření pravděpodobnostní bezkontextové gramatiky na základě trénovací množiny datového korpusu. Pro uložení pravidel se využívá jednoduchý syntaktický analyzátor, který spolupracuje s lexikálním analyzátorem zpracovávajícím vstupní soubor. Tato dvojice analyzátorů postupně prochází jednotlivé věty (soubory) trénovací množiny. Ze zápisu těchto anotovaných vět, který reprezentuje binární derivační strom daného řetězce, se následně sestavují binární pravidla, která po průchodu všemi soubory tvoří (redundantní) množinu pravidel gramatiky.

V dalším kroku strojového učení analyzátoru, kterému může předcházet doplnění univerzálních pravidel popsané dále v podkapitole 6.2.3, se na základě četností výskytu pravidel vyčíslí jejich jednotlivé pravděpodobnosti. Výpočet probíhá vždy pro skupiny pravidel se stejným nonterminálem na levé straně, kdy pravděpodobnost každého pravidla je určena jako podíl počtu výskytů daného pravidla vůči celkovému počtu všech pravidel se stejnou levou stranou. Současně s doplněním pravděpodobností do množiny přepisovacích pravidel gramatiky eliminujeme také duplicitní pravidla z této množiny.

6.2.3 Syntaktická analýza

Jak již název této části napovídá, úkolem této fáze je pro zadanou větu najít nejpravděpodobnější derivační strom, který se následně porovná s očekávaným výsledkem a vyhodnotí se úspěšnost syntaktické analýzy.

Na vstupu tedy očekáváme soubor s anotovanou větou z testovací množiny, při jehož zpracování využíváme podobný syntaktický analyzátor jako při trénování analyzátoru a opět spolupracujeme se stejným lexikálním analyzátorem. Ten ovšem nevytváří množinu gramatických pravidel, ale ze souboru extrahuje pouze slova věty bez jakýchkoliv značek. Dále také ukládá očekávaný výsledek syntaktické analýzy (gold standard tree) v závorkové notaci, kterou jsme popisovali v kapitole 5.5.

Součástí této fáze jsou také dvě důležitá ověření. První z nich probíhá před provedením CKY algoritmu a kontroluje, zda-li jsou slova testované věty obsažena ve slovníku gramatiky. Je-li slovo, které není uloženo ve slovníku, v dané větě nalezeno, do množiny přepisovacích pravidel je přidáno tzv. univerzální pravidlo ve tvaru $X \rightarrow slovo$. Přidáme-li takové pravidlo do gramatiky, doplníme také do množiny přepisovacích pravidel pro každý nonterminál jednoduché pravidlo tvaru $nonterminál \rightarrow X$, abychom zajistili přijetí testované věty syntaktickým analyzátorem, ovšem s již sníženým hodnocením úspěšnosti analýzy. Druhá kontrola probíhá po algoritmu syntaktické analýzy, která verifikuje úspěšnost přijetí vstupního řetězce gramatikou tím, že v buňce pokrývající celou větu (nejvyšší úroveň) nalezne startovací symbol gramatiky S. Je-li výsledek tohoto ověření negativní, testovaná věta není přijímána bezkontextovou gramatikou generovanou z trénovací množiny dat.

Vlastní rozšířený CKY algoritmus, který jsme již detailně popisovali v předchozích částech textu, využívá při svém běhu dvě základní datové struktury – tabulku (chart), do které jsou ukládána nejpravděpodobnější pravidla pro dané pokrytí s průběžnými pravděpodobnostmi větných frází, a obraz této tabulky, který po provedení algoritmu slouží pro zpětnou rekonstrukci

nejpravděpodobnějšího derivačního stromu. Pro získání takového stromu procházíme obraz tabulky od nejvyšší úrovně (začínáme startovacím symbolem gramatiky) po nejnižší, postupně aplikujeme odkazovaná přepisovací pravidla a ukládáme výsledný derivační strom v závorkové notaci.

Posledním krokem celého programu je vyhodnocení úspěšnosti analyzátoru, kde porovnáváme derivační stromy v závorkové notaci pro očekávaný a skutečný výsledek syntaktické analýzy a vypočteme hodnoty parametrů precision, recall a F measure.

6.3 Vlastní implementace

6.3.1 Použití aplikace

Vytvořený program je implementovaný jako konzolová aplikace v programovacím jazyce C++, která načítá vstupní data (trénovací množinu a větu z testovací množiny) z textových souborů, které jsou číselně pojmenované, například tedy `1.txt`. Relativní cesta k jejich umístění, stejně jako číselný rozsah souborů trénovací množiny a daného testovacího souboru jsou zadávány jako parametry příkazového řádku (popsáno níže). V případě, že jsou parametry zadané neúplně nebo vůbec, program uživatele upozorní a je spuštěn s implicitními hodnotami proměnných pro vzorový příklad (datový korpus s pěti soubory).

Na standardní výstup tiskne aplikace přepisovací pravidla pro konstrukci očekávaného derivačního stromu (se slovy) a skutečného výstupu analyzátoru (se stemy pro porovnání), jejich závorkové notace pro účely vyhodnocení a procentuální hodnoty vypočítaných parametrů úspěšnosti syntaktické analýzy. Chybová hlášení (neúspěch při otevření souboru apod.) jsou vypisována na standardní chybový výstup. Ukázka vzorového výstupu programu je uvedena v příloze A této práce.

Použití programu s parametry příkazového řádku je následující:

```
./synt_analyzer path start end test
```

- `path` – relativní cesta k umístění souborů s anotovanými daty,
- `start, end` – čísla souborů, určující počátek a konec rozsahu trénovací množiny,
- `test` – číslo testovaného souboru z testovací množiny anotovaných dat.

6.3.2 Popis funkčnosti programu

Hlavní funkce `main()` řídí celý průběh programu a volá jednotlivé funkce, provádějící výše navržené kroky syntaktického analyzátoru. Nejprve jsou uloženy parametry příkazového řádku, na základě kterých jsou identifikovány trénovací a testovací množiny vstupních dat.

První z nich je zpracována ve funkci `train_grammar()`, která postupně po znacích načítá jednotlivé věty ze souborů, které dále předává syntaktickému analyzátoru `get_rules()`. Tato rekurzivní funkce na základě vstupů od lexikálního analyzátoru `get_next_element()`, který prochází získaný řetězec, sestavuje a ukládá pravidla vytvářené bezkontextové gramatiky. Kromě pravidel také zapisuje i nonterminální a terminální symboly gramatiky. Po vytvoření množiny přepisovacích pravidel je řízení programu vráceno zpět do hlavní funkce `main()`.

Zadaná testovací věta je opět po znacích načtena a dále zpracována ve funkci `read_test_tree()`, která volá funkce `get_words()` a `get_gold_brackets()`. První z nich získá slova dané věty v jejich přesném pořadí a druhá uloží očekávaný výsledek syntaktické

analýzy v závorkové notaci spolu s použitými přepisovacími pravidly. Obě tyto funkce opět spolupracují s lexikálním analyzátorem `get_next_element()`.

Některé z výše uvedených funkcí také používají volání `get_word_stem()` externě vytvořeného stemmeru [14]. Vzorová aplikace uvedené diplomové práce, která pracovala se standardním vstupem a výstupem, byla pro účely našeho programu mírně upravena do podoby volání funkce vracející základní tvar slova (stem) pro zadané slovo. Tento stemmer byl implementován v jazyce Snowball, který vytvořené skripty překládá do zdrojových kódů v jazyce C, se kterými je možné dále pracovat.

Po zpracování testovací věty se ověří, že jsou všechna její slova součástí slovníku programem vytvořené gramatiky a případně se do gramatiky doplní univerzální pravidla. Následně jsou ve funkci `count_probabilities()` vypočteny pravděpodobnosti jednotlivých pravidel, čímž je dokončena pravděpodobnostní bezkontextová gramatika, předávaná do vlastního algoritmu syntaktické analýzy `CKY_algorithm()`. Po jeho dokončení probíhá kontrola přijetí vstupního testovacího řetězce vygenerovanou gramatikou tím, že buňka výstupní tabulky CKY algoritmu, pokrývající celý vstupní řetězec, obsahuje počáteční symbol gramatiky. V případě úspěchu se volá funkce `get_best_tree()` pro získání závorkové notace a přepisovacích pravidel nejpravděpodobnějšího derivačního stromu podle výpočtu algoritmu syntaktické analýzy.

V posledním kroku programu se ve funkci `evaluate()` vyhodnocuje úspěšnost syntaktické analýzy porovnáním očekávaného a skutečného výstupu analyzátoru.

6.3.3 Základní datové struktury

Většinu datových struktur programu jsme implementovali pomocí kontejnerů standardní knihovny STL (Standard Template Library), která je v dnešní době dodávána s téměř každým překladačem jazyka C++.

Pro uložení množiny přepisovacích pravidel generované pravděpodobnostní bezkontextové gramatiky jsme použili asociativní pole (pracuje s dvojicí klíč a hodnota), konkrétně `multimap`, které může mít pro jeden klíč (na rozdíl od podobného kontejneru `map`) více asociovaných hodnot. Deklarace proměnné potom vypadá následovně:

```
multimap<string, t_rule> rules;
```

První položka (klíč) datového typu `string` obsahuje nonterminál levé strany daného pravidla, ve druhém prvku kontejneru (hodnota) je poté uložena struktura `t_rule` reprezentující celé pravidlo včetně jeho pravděpodobnosti. Ta je deklarována:

```
typedef struct {
  string parent;
  string left_leaf;
  string right_leaf;
  double probability;
  int count;
} t_rule;
```

Řetězec `parent` ukládá nonterminál levé strany pravidla, řetězce `left_leaf` a `right_leaf` poté reprezentují pravou stranu přepisovacího pravidla. Je-li položka `right_leaf` prázdná, jedná se buď o jednoduché pravidlo nebo se nonterminální symbol levé strany přepisuje na

terminál pravé strany. Zbývající dva prvky struktury obsahují číselné hodnoty `probability`, ukládající pravděpodobnost daného pravidla, a `count`, udávající jeho četnost.

Datové struktury rozšířeného CKY algoritmu, tabulka (chart) a její obraz, jsou definovány jako dvojdimenzionální vektory asociativních polí `map` s velikostí dimenze podle počtu slov testované věty `words_size+1`. Tabulka `score` (podle algoritmu definovaného v podkapitole 4.7) je poté deklarována níže, kde řetězec ukládá získaný nonterminál a číselná hodnota průběžnou pravděpodobnost při výpočtu CKY algoritmu.

```
vector<map<string,double> > score_rows(words_size+1);
vector<vector<map<string,double> > > score(words_size+1,score_rows);
```

6.4 Testování

Implementovaný syntaktický analyzátor byl vyvíjen a také testován na virtualizovaném operačním systému Linux. Pro kontrolu korektní správy paměti za běhu programu byl využíván nástroj `valgrind`.

Pro vývoj a odladění praktické části této diplomové práce byl používán jednoduchý malý korpus s pěti větami (uložený v adresáři `src/test_corp/`). Při samotném testování výsledného programu jsme pracovali s anotovaným frázovým korpusem českého jazyka, vytvořeným právě pro tyto účely (adresář `src/corp/`). Tento datový vzorek 100 vět jsme vždy rozdělili na trénovací a testovací množinu, které byly předávány jako vstupy do aplikace.

Vzhledem k relativně malému počtu připravených anotovaných dat probíhalo testování metodou *křížové validace* (cross-validace), díky které minimalizujeme potencionální odchylky způsobené omezeným rozsahem použitých dat. Datová sada se rozdělí na vhodný počet podmnožin, kdy jedna z nich je označena jako testovací a zbylé jsou určeny pro trénování analyzátoru. Testování se poté opakovaně provádí pokaždé s jinou testovanou podmnožinou a získané výsledky se zprůměrují.

Výstupy testů (uložené v adresáři `test_results/`) jsou reprezentovány parametry pro vyhodnocení úspěšnosti syntaktické analýzy (precision, recall a F measure), které jsme definovali v předchozích částech tohoto textu. K těmto hodnotám jsme také přidali ukazatel pokrytí testované věty slovníkem gramatiky vygenerované z trénovací množiny dat, který je definován poměrem vět, pro které nebylo do gramatiky přidáno univerzální pravidlo s nonterminálem X na levé straně, a tudíž také není součástí derivačního stromu na výstupu analyzátoru.

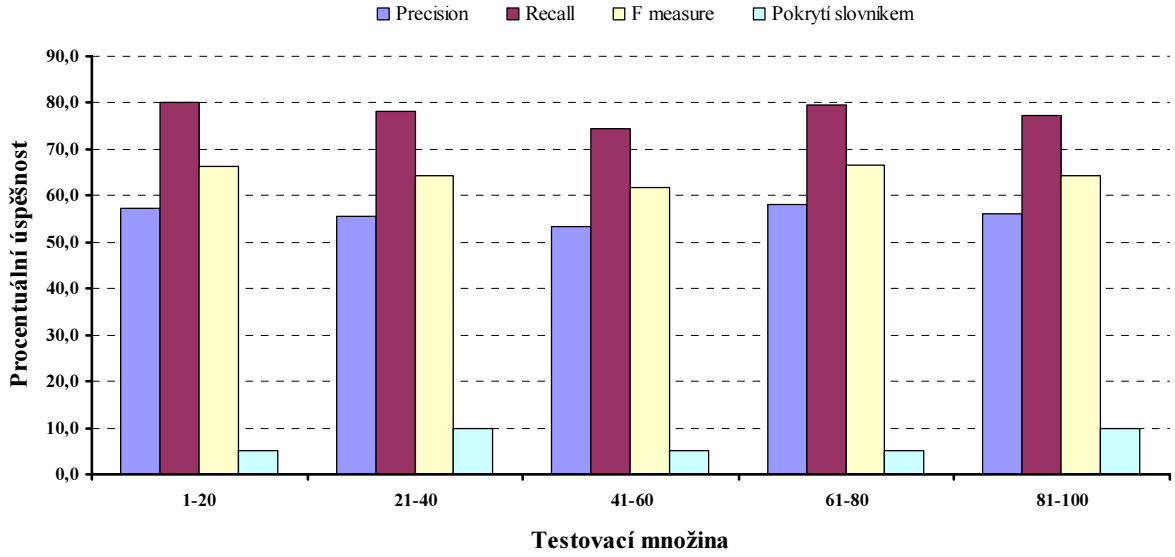
Testy byly provedeny ve třech variantách, které jsou popsány dále.

6.4.1 Varianta 1 – stem jako terminál, rozdělení 80:20

Jako základní testovací scénář byla vybrána varianta, kdy jsou terminální symboly gramatiky i slova testovaných vět transformovány a uloženy ve tvaru stemu. Trénovací a testovací množiny byly rozděleny v poměru 80:20, postup byl tedy pětkrát opakován pro různé množiny testovacích dat.

Výsledky testů jsou znázorněny v grafu 6.1, kde vodorovná osa reprezentuje jednotlivé cykly provedení a svislá osa popisuje procentuální úspěšnost. Z uvedeného grafu můžeme vidět, že hodnota parametru recall je pro všechny případy přibližně o čtvrtinu vyšší než vypočtené výsledky parametru presicion. To znamená, že derivační stromy na výstupu syntaktického analyzátoru jsou oproti očekávaným stromům konstruovány pomocí většího počtu pravidel, což je pravděpodobně způsobeno použitím univerzálních pravidel, které nastává v majoritní většině případů. Celková úspěšnost

syntaktického analyzátoru je průměrně o 10% nižší než hodnota 73% udávaná pro základní anglický analyzátor Penn Wall Street Journal [3]. Námi naměřená nižší hodnota může být částečně ovlivněna i nižší úspěšností použitého stemmeru, která je podle [15] přibližně 60%.

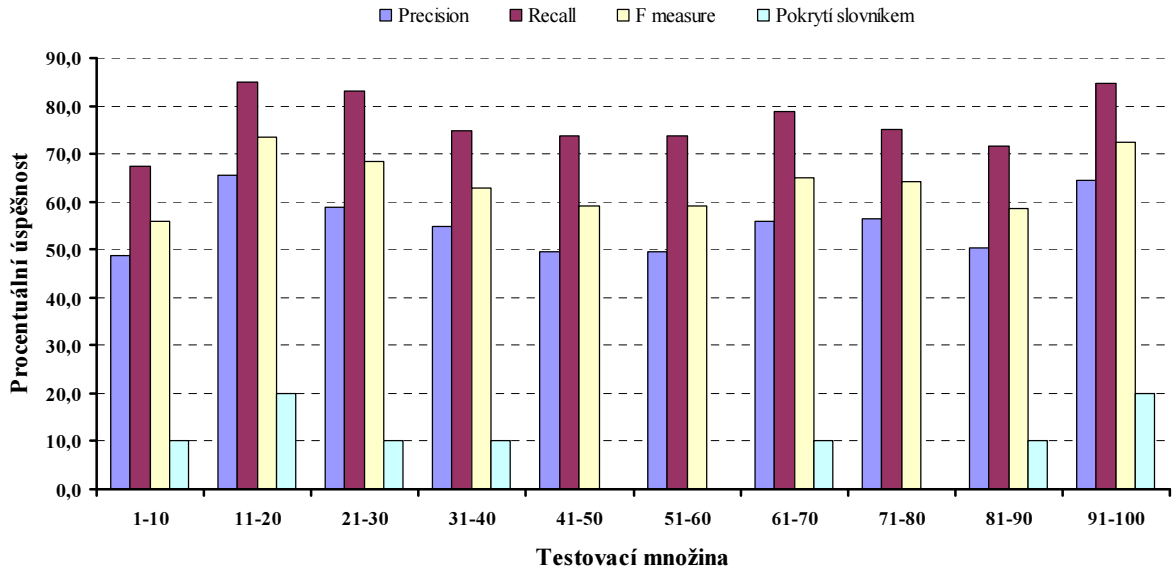


Graf 6.1: Výsledky testování varianty 1 – stem jako terminál, rozdělení 80:20.

6.4.2 Varianta 2 – stem jako terminál, rozdělení 90:10

Cílem tohoto testovacího scénáře bylo porovnání výsledné úspěšnosti analyzátoru pro jiné rozdělení množiny zpracovávaných dat.

V grafu 6.2 můžeme pozorovat, že průměrné hodnoty celého testu odpovídají výsledkům varianty 1, nicméně zde již mezi jednotlivými cykly této varianty nastávají znatelnější rozdíly.

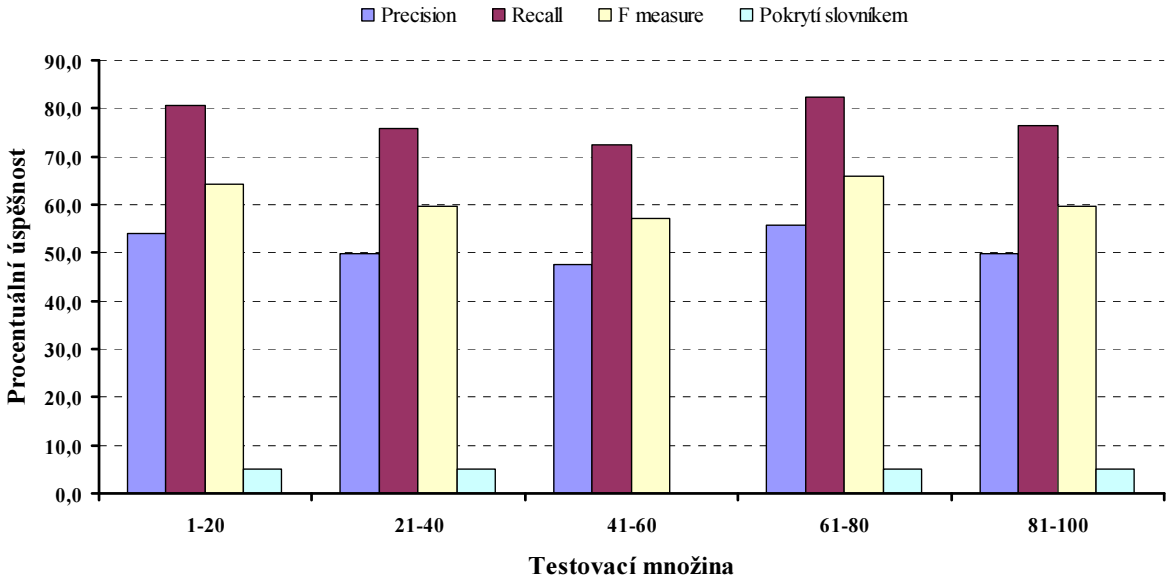


Graf 6.2: Výsledky testování varianty 2 – stem jako terminál, rozdělení 90:10.

6.4.3 Varianta 3 – slovo jako terminál, rozdělení 80:20

Základní testovací varianta byla v tomto případě upravena pro odlišnou reprezentaci terminálních symbolů a slov testované věty, které byly ukládány přímo ve formě slov načtených ze vstupních souborů, tedy například v různých pádech či jiných tvarech.

Ve výsledku byla kvůli předpokládanému častějšímu využití univerzálních pravidel očekávána snížená úspěšnost syntaktické analýzy, která se ale na tomto malém vzorku dat projevila nejvýše o jednotky procent (viz graf 6.3). Zanedbatelná změna je také způsobena malým slovníkovým pokrytím testovaných vět již ve výsledcích varianty 1.



Graf 6.3: Výsledky testování varianty 3 – slovo jako terminál, rozdělení 80:20.

Shrneme-li výsledky testování celkově, implementovaný syntaktický analyzátor dosahuje vzhledem k velmi malé množině anotovaných dat přiměřených výsledků. Výstupní nejpravděpodobnější derivační stromy jsou s vyšší úspěšností konstruovány pomocí očekávaných pravidel, která jsou ale ve většině případů doplněna o univerzální pravidla snižující úspěšnost analýzy.

7 Závěr

Na základě teoretických znalostí popsaných v úvodních kapitolách této práce a rozboru algoritmů syntaktické analýzy, používaných pro zpracování přirozeného jazyka, jsem navrhl a následně implementoval syntaktický analyzátor pro jazyk český založený na frázovém přístupu ke stavbě věty. Jádrem analyzátoru je rozšířený CKY algoritmus pracující s pravděpodobnostní bezkontextovou gramatikou, která je generována za běhu programu na základě vstupní trénovací množiny dat. Pro účely testování aplikace jsem také vytvořil anotovaný frázový korpus češtiny. Hodnoty úspěšnosti syntaktické analýzy získané testováním odpovídají předpokládaným výsledkům, zejména s přihlédnutím k relativně malému množství testovacích dat.

7.1 Další vývoj projektu

Jak již bylo zmíněno výše, jedním z hlavních limitujících prvků pro dosažení lepších výsledků syntaktické analýzy, je relativně malý rozsah vytvořeného korpusu českého jazyka, což se také odvíjí od menšího rozvoje a zpracování frázových korpusů češtiny vůbec. Vhodnou návazností na tento projekt by tedy mohlo být nejen jeho rozšíření co do počtu vět a slov, ale také i další zkvalitňování jeho obsahu např. zavedením souvětí či umožněním anotace obecných stromů.

Dalšího zvýšení úspěšnosti syntaktické analýzy by bylo dosaženo integrací s morfologickým analyzátorem, která by nahradila vytváření univerzálního přepisovacího pravidla. Do generované gramatiky by se tak ukládala přímo pravidla se získanou kategorií slovního druhu, což by přineslo výraznější zvýšení pravděpodobnosti konstrukce správného derivačního stromu.

Lexikalizace pravděpodobnostní bezkontextové gramatiky je také jedna z možností pro rozšíření tohoto projektu. Při lexikalizaci jsou s jednotlivými pravidly ukládány dodatečné informace týkající se zejména kontextu a konkrétního významu, které mohou různými způsoby vyřešit některé víceznačnosti gramatik.

Literatura

[1] ČEŠKA, M., T. VOJNAR a A. SMRČKA. *Teoretická informatika TIN*: *Studijní opora* [online]. 1. září 2011 [cit. 2014-03-28]. Dostupné z: https://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf

[2] MANNING, Christopher D a Hinrich SCHÜTZE. *Foundations of statistical natural language processing*. Cambridge: MIT Press, c1999, xxxvii, 680 s. ISBN 0262133601.

[3] JURAFSKY, Dan a Chris MANNING. Natural Language Processing. *Coursea* [online]. 2014 [cit. 2014-03-28]. Dostupné z: https://class.coursera.org/nlp/lecture/165

[4] Úvod do korpusové lingvistiky. *Centrum NLP* [online]. 2012-03-16 [cit. 2014-03-29]. Dostupné z: http://nlp.fi.muni.cz/cs/Uvod_do_korpusove_lingvistiky

[5] ÚFAL & CKL. *Pražský závislostní korpus 2.0* [online]. c 2006 [cit. 2014-03-29]. Dostupné z: http://ufal.mff.cuni.cz/pdt2.0/index-cz.html

[6] Státnice I3: Závislostní syntax. In: *ωικι.matfyz.cz* [online]. 22 Aug 2010, 5 Feb 2012 [cit. 2014-03-29]. Dostupné z: http://wiki.matfyz.cz/index.php?title=Státnice_I3:_Závislostní_syntax

[7] HORÁK, Aleš. Computer processing of Czech syntax and semantics [online]. 1st ed. Brno: [Tribun EU], 2008, 229 s. [cit. 2014-03-29]. Librix.eu. ISBN 978-80-7399-375-7. Dostupné z: http://www.fi.muni.cz/~hales/proc/proc.pdf

[8] Treebank. *Wikipedia: The Free Encyclopedia* [online]. 2006, 26 March 2014 [cit. 2014-04-05]. Dostupné z: http://en.wikipedia.org/wiki/Treebank

[9] MITKOV, Ruslan. *The Oxford handbook of computational linguistics*. Oxford: Oxford University Press, 2003, XX, 784 s. ISBN 9780199276349.

[10] HABIBALLA, Hashim. *Regulární a bezkontextové jazyky II* [online]. první. Ostrava, 2005 [cit. 2014-04-05]. Dostupné z: http://www1.osu.cz/home/habibal/publ/rabj2.pdf

[11] HALL, Johan, Jens NILSSON a Joakim NIVRE. Introduction. *MaltParser* [online]. [2008], April 5 2014 [cit. 2014-04-06]. Dostupné z: http://www.maltparser.org/intro.html

[12] MEDUNA, Alexander. *Automata and languages: theory and applications*. London: Springer, 2000, xv, 916 s. ISBN 18-523-3074-0.

[13] Viterbiho algoritmus. *Wikipedie: Otevřená encyklopedie* [online]. 2013, 8.2.2014 [cit. 2014-04-10]. Dostupné z: http://cs.wikipedia.org/wiki/Viterbiho_algoritmus

[14] ŘÍHA, Bohumil. *Honzíkova cesta*. Upr. vyd. Ilustrace Helena Zmatlíková. Praha: Axióma, 2006, 73 s. ISBN 80-729-2114-2.

[15] HELLEBRAND, David. *Nalezení slovních kořenů v češtině*. Brno, 2010. Diplomová práce. FIT VUT v Brně. Vedoucí práce Ing. Petr Chmelař.

Seznam příloh

Příloha A Vzorový výstup programu

```
Gold standard tree rules with words:
------------------------------------
S -> VP Z
VP -> NP VP
NP -> PRON N
PRON -> Naše
N -> maminka
VP -> V PP
V -> stála
PP -> PREP N
PREP -> v
N -> kuchyni
Z -> .

Most probable tree rules with stems:
------------------------------------
S -> VP Z
VP -> NP VP
NP -> PRON N
PRON -> nas
N -> mam
VP -> V PP
V -> stat
PP -> PREP N
PREP -> v
N -> kuch
Z -> .

Gold standard brackets:
-----------------------
S-(0:6) VP-(0:5) NP-(0:2) VP-(2:5) PP-(3:5)

Most probable parse brackets:
-----------------------------
S-(0:6) VP-(0:5) NP-(0:2) VP-(2:5) PP-(3:5)

Parser evaluation:
------------------
P: 100%
R: 100%
F: 100%
```

Příloha B Obsah přiloženého CD

Adresářová struktura přiloženého CD je následující:

- `src`
  - `corp` – vytvořený frázový korpus českého jazyka
  - `stemmer` – zdrojové soubory externí aplikace
  - `test_corp` – testovací korpus pro vývojové účely
  - soubor `corp_list.txt` obsahuje seznam vět vytvořeného korpusu
  - soubor `makefile` slouží pro překlad vytvořené aplikace
  - soubor `synt_analyzer.cc` obsahuje výkonný kód programu
  - soubor `synt_analyzer.h` je hlavičkový soubor programu
  - soubor `README` obsahuje pokyny ke kompilaci a spuštění
- `test_results` – soubory s výsledky testovacích scénářů
- soubor `xbenes04_dip_tech_zprava.doc` je zdrojový soubor této práce
- soubor `xbenes04_dip_tech_zprava.pdf` je tato práce ve formátu pdf