Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
FIIT-5208-46120

Jaroslav Loebl

Automatické spracovanie textu – syntaktická analýza vety

Diplomová práca

Študijný program: Informačné systémy
Študijný odbor: 9.2.6 Informačné systémy
Miesto vypracovania: Ústav informatiky, informačných systémov a softvérového inžinierstva, FIIT STU v Bratislave
Vedúci: Ing. Marián Šimko, PhD.

máj 2016

Anotácia

Študijný program: Informačné systémy
Autor: Jaroslav Loebl
Názov: Automatické spracovanie textu – syntaktická analýza vety
Vedúci: Ing. Marián Šimko, PhD.
máj 2016

Syntaktické parsovanie je považované za jeden z najdôležitejších krokov pri snahe o hlbšie pochopenie textu strojmi. Vyhodnocovanie často prebieha na publicistických alebo beletristických textoch, ktoré na rozdiel od webového textu takmer neobsahujú žiadne chyby.

V tejto práci sme sa zamerali na simulovanie bežných chýb reálneho webového textu a skúmanie jeho vplyvu na úspešnosť parsovania, v porovnaní s textami bez chýb. Ďalej sme sa zamerali na využitie dostupných nástrojov na opravu textu a skúmali, do akej miery vieme úspešnosť parsovania navrátiť na pôvodnú úroveň.

Okrem toho sme navrhli metódu, založenú na obohatení morfologickej značky na vylepšenie úspešnosti určovania konkrétneho vzťahu vo vete.

Parsovanie a všetky overené vylepšenia sme následne implementovali ako webovú službu.

Annotation

Degree Course: Information systems
Author: Jaroslav Loebl
Thesis: Automatic text processing - syntactic analysis of a sentence
Supervisor: Ing. Marián Šimko, PhD.
May, 2016

Syntactic parsing is considered to be one of the most important steps in goal of deeper understanding of text. Parsers are usually evaluated on publicist or fiction texts, which are heavily edited, in contrast to web text.

In this paper, we aimed on simulating common of real web text and how addition of these errors affects the parsing accuracy, in comparison of parsing text without errors. Next, we aimed on utilization of available text reconstruction tools to improve parsing accuracy and return it to original baseline as close as possible.

Furthermore, we proposed a method based on enrichment of morphological feature set to improve accuracy of determining specific type of relation in sentence. Parsing, with all evaluated improvements was then implemented as a web service

Čestne prehlasujem, že som nasledovnú prácu vypracoval sám, len s použitím uvedených zdrojov.

Jaroslav Loebl

Ďakujem Ing. Mariánovi Šimkovi, PhD. za trpezlivosť, usmerňovanie a poskytovanie cenných rád na početných konzultáciách v priebehu riešenia diplomového projektu.

Obsah

1 Úvod

Jazyk, a špeciálne jeho písaná forma, je jeden z najčastejších spôsobov uchovávania informácií. Na webe, ale aj mimo neho, nájdeme enormné množstvo informácií uložených v písaných dokumentoch. Dolovanie v textoch (z angl. text mining) je dnes rýchlo rastúci odbor, s veľkým potenciálom prínosu nových poznatkov. Dokumenty z ktorých dolovanie v textoch získava poznatky ale nie sú štruktúrované dáta a je treba ich najprv spracovať do stavu vhodného pre dolovanie. Nie len tejto časti sa venuje vedný odbor známy ako spracovanie prirodzeného jazyka (z angl. natural language processing).

Spracovanie prirodzeného jazyka vystavuje celý rad výziev a problémov – každá úroveň analýzy textu má svoje špecifiká, ktoré sa navyše môžu líšiť aj medzi jednotlivými jazykmi. Medzi všetkými má špeciálne postavenie syntaktická analýza, alebo syntaktické parsovanie viet. Z nej totiž môžu ťažiť všetky ostatné úrovne analýzy a navyše stojí na pomyselnom pomedzí plytkého spracovania, kde netreba chápať význam, a hĺbkového spracovania, kde už je potrebné poznať aj význam vety (napríklad sémantická analýza) .

Syntaktické parsovanie prešlo od konca deväťdesiatych rokov búrlivým vývojom, a to hlavne pod vplyvom štatistického spracovania a dostupnosti veľkého množstva anotovaných dát a korpusov. Tie umožnili použiť rôzne techniky strojového učenia na vytváranie modelov, na základe ktorých je možné predpovedať syntaktickú štruktúru neoznačených viet. Je treba ale podotknúť, že minimálne spočiatku bola drvivá väčšina úsilia venovaná anglickému jazyku, pričom len málo sa venovalo špecifikám, ktoré v sebe majú flektívne jazyky. Tým sa výskum začal venovať trochu neskôr, ale v súčasnosti už aj parsovanie flektívnych jazykov, ako napríklad čeština, dosahuje veľmi dobré výsledky.

V tejto práci sa ale chceme zamerať na slovenčinu, nakoľko za ostatnými jazykmi ohľadom syntaktického parsovania dosť zaostáva. Je už pre ňu ale k dispozícii prvý závislostný korpus, čo otvára možnosti pre štatistické spracovanie. Cieľom je navrhnúť a implementovať metódu pre automatickú syntaktickú analýzu slovenskej vety a overiť jej úspešnosť na dostačujúcej vzorke dát. V tejto práci predstavíme základnú problematiku ohľadom slovenskej syntaxe, špecifiká ktoré pre parsovanie vystavuje a pozrieme sa na už existujúce riešenia, ktoré sú schopné parsovať aj slovenčinu.

2 Východiská pre syntaktickú analýzu

2.1 Stratifikačná sekvenčná analýza

Prirodzený jazyk je možno analyzovať na viacero úrovniach. Ich postupné spracovanie sa nazýva stratifikačná sekvenčná analýza (Páleš, 1994). Pozostáva z týchto krokov:

- Fonologická analýza – je relevantná keď je ako vstup hovorené slovo (zvukový záznam). Výstupom je text reprezentovaný ako reťazec znakov.
- Morfologická analýza – identifikácia gramatických kategórií jednotlivých slov vo vete. Výstupom je zoznam gramatických kategórií ku každému slovu.
- Syntaktická analýza – identifikácia vetných členov a syntagiem. Výstupom je syntaktická štruktúra vyjadrujúca formálne vzťahy vo vete.
- Morfematická analýza – identifikácia slovotvornej štruktúry. Výstup je analogický so syntaktickou analýzou a je jej rozšírením.
- Sémantická analýza – identifikácia významových participantov a priradenie príslušnej sémantickej roly. Výstupom je sémantická štruktúra vyjadrujúca vzťahy významovej závislosti.
- Kontextová analýza – identifikácia vzťahov jednotlivých entít medzi dvoma a viacerými vetami. Výstupom sú kontextové rovnice, ktoré vyjadrujú rôzne pomenovania tej istej entity v rámci kontextu.
- Inferencia – začlenenie novej informácie do už existujúcej poznatkovej siete.

Jednotlivé vrstvy takmer vždy závisia od vrstiev ktoré tej danej predchádzajú. Nepresnosti v jednej sa zákonite prejavujú ako chyby v nadväzujúcej. Výnimku tvorí morfologická analýza, ktorá v je v prípade spracovania textu vstupná, resp. prvá úroveň analýzy. Od predchádzajúcej úrovne, fonologickej, závisí len v prípade, že vstup ktorý spracovávame nie je text, ale zvuk.

Nástroje na spracovanie prirodzeného jazyka sa často sústreďujú osobitne iba na jednu vrstvu analýzy, pričom predošle berú iba ako vstup. Existujú ale prípady, keď jedna vrstva môže ťažiť aj z nasledujúcich. Bežne používaným príkladom môže byť morfologická a syntaktická analýza. Napríklad pri slove *mať,* nevieme jednoznačne určiť morfologickú značku (sloveso, alebo podstatné meno) bez toho, aby sme vedeli v akom kontexte je zasadené. K tomu by nám pomohla práve syntaktická analýza. Priaznivé účinky spojenia týchto dvoch vrstiev analýzy sa už podarilo overiť na viacerých vysoko flektívnych jazykoch, ako

je čeština, fínčina, nemčina a ruština (Bohnet, 2013). Ďalším príkladom nevyhnutného spojenia dvoch vrstiev je určovanie druhu príslovkového určenia (či sa jedná o príslovkové určenie miesta, času, alebo príčiny). Určovanie samotného vetného člena patri do syntaktickej analýzy, ale na to, aby sme presne vedeli určiť, o ktorý druh príslovkového určenia sa jedná, nevyhnutne potrebujeme poznať sémantiku vety (Kačala, 1998).

2.2 Syntax slovenského jazyka

Syntaktická rovina stojí medzi čisto jazykovými rovinami najvyššie, najlepšie predstavuje celosť jazyka a jeho tesnú súvislosť s myslením (Kačala, 1998). Skladba ako náuka o syntaktickej rovine opisuje, ako sa skladajú (aj ako už sú zložené) správne vety a ich časti a ďalej ako sa skladajú vety do zložitejších celkov (súvetí) (Oravec, a iní, 1986). Samotný syntaktický systém následne podľa (Kačala, 1998) pozostáva z:

1. syntaktických jednotiek,
2. syntaktických vzťahov,
3. syntaktických (konštrukčných) pravidiel a
4. syntaktických prostriedkov.

Pre potreby tejto práce sú pre nás zaujímavé hlavne prvé dve položky syntaktického systému – syntaktické jednotky a vzťahy. Poskytujú dobrý základ pre pochopenie a lepšiu orientáciu v syntaktickej analýze aj z pohľadu výpočtovej lingvistiky. Konštrukčné pravidlá a syntaktické prostriedky ďalej nerozoberáme, nakoľko sú zaujímavé skôr z pohľadu lingvistiky ako takej a pri skúmaní štatistických metód parsovania poznatky z týchto častí syntaktického systému až tak nevyužijeme.

2.2.1 Syntaktické jednotky

Medzi syntaktické jednotky zaraďujeme vetné členy, syntagmy, polopredikatívne konštrukcie, vetné konštrukcie a súvetné konštrukcie. Sústavu vetných členov tvorí:

- podmet - subjekt dvojčlennej vety,
- prísudok - predikát dvojčlennej vety,
- vetný základ - fundament jednočlennej vety,
- predmet - objekt rozvíjajúci sloveso v hocijakej vetnej pozícií,
- prívlastok - atribút rozvíja podstatné meno v hociktorej vetnej pozícií,
- prístavok - apozícia špeciálny druh zhodného substantívneho prívlastku (vedľajší vetný člen),
- doplnok - sekundárny predikát - SP (špeciálny vetný člen).

Treba podotknúť, že z hľadiska výpočtovej lingvistiky je počet vetných členov (analytických funkcií), ktoré sa vo vete určujú omnoho väčší. Je to z toho dôvodu, že pre účely počítačového spracovania potrebujeme vedieť funkciu každého tokenu – okrem rôznych neplnovýznamových slovných druhov (napríklad rôzne onomatopoických citosloviec) sú to aj rôzne grafické symboly (osobitne sa rozlišuje napríklad bodka na konci vety ako ukončovací znak vety, osobitne zas čiarky a bodky, napríklad za titulmi a skratkami v strede vety, zátvorky, úvodzovky a pod.). Pri bežnom vetnom rozbore, ako sa robí napríklad v školách, sú tieto časti zanedbávané, nakoľko majú len malý informačný prínos. Pri počítačovom spracovaní ich ale určiť potrebujeme – častokrát práve preto, aby sme vedeli, či ich môžeme v prípade potreby v ďalšom spracovaní ignorovať. Kompletný prehľad všetkých analytických funkcií tak, ako ich definuje návod pre anotátorov syntaktických korpusov[1], v prílohe C.

2.2.2 Syntaktické vzťahy

Syntaktické vzťahy na úrovni vetných členov, môžeme do istej miery chápať ako jazykovednú analógiu k stromovým reprezentáciám viet, tak ako ich produkujú rôzne závislostné parsre, napríklad do formátu Treebank korpusov. Syntaktické vzťahy rozlišujeme základné a špecifické.

Základný syntaktický vzťah, inak nazývaný aj vetotvorný je gramatický aj sémantický zároveň. Podľa spôsobu realizácie môže byť:

- predikačný - v dvojčlennej vete medzi prísudkom a podmetom (rozčlenené gramatické jadro vety), alebo samostatný predikát v jednočlennej vete.
- nepredikačný - vo vetách, kde nie sú k dispozícií predikačné kategórie času a spôsobu, hlavne teda v neslovesných vetách, napríklad: *Dohodnuté! Krátky slovník slovenského jazyka.*

Špecifický syntaktický vzťah môže byť medzi rovnorodými aj nerovnorodými vetnými členmi a medzi súvetnými konštrukciami. Patrí sem:

- subordinatívny (podmieňovací vzťah), používa sa medzi nerovnocennými syntaktickými jednotkami, jedného člena vzťahu určuje ako nadradeného, zatiaľ čo druhého ako podradeného, rozvíjacieho. Subordinatívny vzťah zisťujeme medzi:
    - substantívom a jeho zhodným alebo nezhodným prívlastkom: čierny drozd, júnový večer... ,

[1] https://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/

- prídavným menom a jeho príslovkovým určením: nekonečne dlhý, veľmi pekný,
- osobným zámenom a jeho substantívnym alebo číslovkovým prívlastkom: my rodičia, vy dvaja,
- medzi slovom a jeho predmetom: prekonať rekord, zohrať rolu, vyhýbať sa nebezpečenstvu,
- medzi slovesom a jeho príslovkovým určením: zobudiť sa zavčasu, neprísť pre chorobu, použiť na zaplatenie,
- medzi príslovkou a jej príslovkovým určením: veľmi výrazne, neprimerane dlho.

- Koordinatívny - vzťah nezávislosti. Uplatňuje sa medzi rovnorodými členmi. Vetné členy spojené koordinatívnym vzťahom dovedna tvoria spoločný (viacnásobný) vetný člen, a to:
  - podmet: Fialky a prvosienky patria medzi prvých poslov jari,
  - príslovkové určenie: Pútnici prešli vrchy, doliny aj polia,
  - prívlastok (zhodný aj nezhodný): Je to dobrý a chápavý otec. Chlapci si vyskúšali jazdu na koni aj rezanie pílou.,
  - prístavok: V Banskej Bystrici, stredoslovenskej metropole a sídle viacerých celoslovenských inštitúcií, otvorili medzinárodnú výstavu.,
  - doplnok (zhodný, nezhodný, kombinovaný): Ráno sa zobudil nevyspatý a v zlej nálade.

Syntaktický systém ale okrem jednoduchých vzťahov nadradenosti a podradenosti pozná aj komplikovanejšie konštrukcie. Postupne rozvíjajúci prívlastok, alebo viacnásobný prívlastok predstavujú zložené syntagmy, kedy sa podradené vetné členy (prívlastky) neviažu iba na nadradený člen (podstatné meno), ale aj na všetky ostatné členy ktoré sa nachádzajú ďalej pred nadradeným členom, napríklad:

naša prvá významná zahraničná cesta

Slovosled je veľmi dôležitý pri subordinatívnych syntagmách, pri slede členov, v ktorých nadradeným členom je podstatné meno. Takýto slovosled nazývame formálny. Ten sa však bežne porušuje napríklad pri zhodnom adjektívnom prívlastku, napr. v odbornom štýle: kyselina soľná, alebo bežne v umeleckom štýle.

3 Metódy syntaktickej analýzy

3.1 Základné prístupy k syntaktickej analýze

Syntaktická analýza je všeobecne uznávaná ako kľúčová pre spracovanie prirodzeného jazyka a ďalších úloh ako získavanie informácií (z angl. information retrieval), strojový preklad (z angl. machine translation) alebo získavanie znalostí (z angl. knowledge discovery). Prehľad problematiky a stav domény je veľmi dobre spracovaný v (Červenová, 2015), tu uvedieme a vysvetlíme len niektoré základné termíny, ktoré sú pre túto prácu potrebné.

Prístupy k syntaktickej analýze na poli počítačovej lingvistiky sa dajú rozdeliť na dve hlavné:

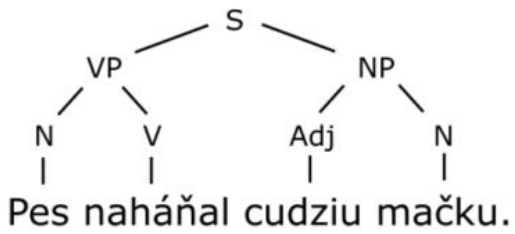
1. pravidlové systémy: jazyk parsujú na základe vopred stanovených pravidiel - gramatiky (z angl. grammar based). K prirodzenému jazyku pristupujú ako ku bezkontextovej gramatike,
2. štatistické systémy: inak nazývané aj riadené dátami (angl. data-driven). Tieto systémy používajú metódy strojového učenia a veľké trénovacie množiny dát. Na nich odvodzujú pravidlá a klasifikátory, podľa ktorých potom vedia sparsovať nové neoznačené vety.

3.1.1 Zložková a závislostná gramatika

Formát výstupu parserov sa dá rozdeliť na dve skupiny podľa gramatiky:

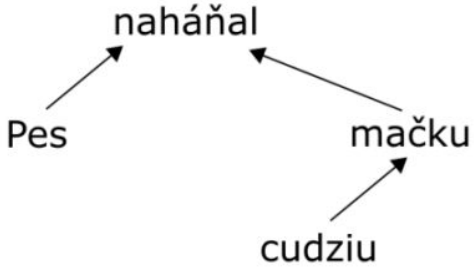
- zložkový,
- závislostný.

Zložková gramatika rekurzívne rozdeľuje vetu na dve časti, až kým neostanú samotné slová. Časti na ktoré sa veta rozdeľuje sú syntaktické kategórie, napríklad na podstatnomenná fráza (z angl. noun phrase - NP) a slovesná fráza (z angl. verb phrase - VP). Toto je hlavná informácia, ktorá sa nachádza v zložkovej gramatike, ale nie v závislostnej (Xia, a iní, 2001). V strome vygenerovanom týmto jazykom sú slová vety listy stromu (viď obrázok 1). Takáto gramatika sa hodí skôr pre angličtinu a iné jazyky, ktoré majú pevnú vetnú skladbu a poradie slov. Používa ju napríklad Standford parser.



Obrázok 1 - príklad zložkovej reprezentácie vety

Závislostná gramatika určuje vzťahy závislostí medzi slovami vo vete. V strome vygenerovaným touto gramatikou je každý vrchol slovo a orientované hrany medzi vrcholmi reprezentujú vzťah závislosti (viď obrázok 2). Na poli spracovania prirodzeného jazyka rastie záujem o parsre generujúce túto gramatiku, nakoľko veľa disciplín z tohto oboru ťaží z informácie, ktoré slovo na ktorom závisí, čo je informácia, ktorú zložkové parsre neposkytujú (De Marnaffe, 2006). Táto gramatika je vhodnejšia pre jazyky s relatívne voľným slovosledom, ako napríklad rodina slovanských jazykov (Maxwell, 2013). Parsre generujúce podľa tejto gramatiky sú napríklad MST Parser (McDonald, 2005) a Malt Parser (Nivre, a iní, 2006). V závislosti od použitého algoritmu následne ešte môžu generovať závislostné stromy projektívne a neprojektívne.

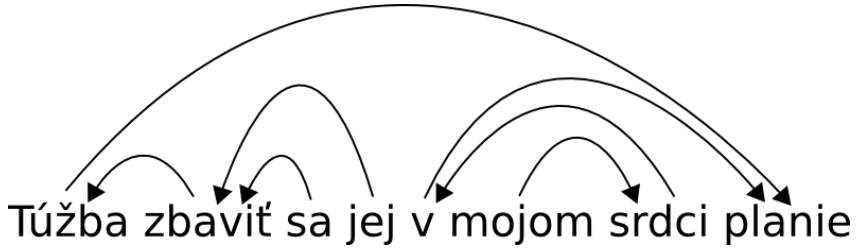


Obrázok 2 - príklad závislostnej reprezentácie vety

Každá z týchto reprezentácií vety má svoje výhody a nevýhody a existujú aj systémy, ktorá sa zameriavajú na konverziu z jedného druhu do druhého (Xia, a iní, 2001).

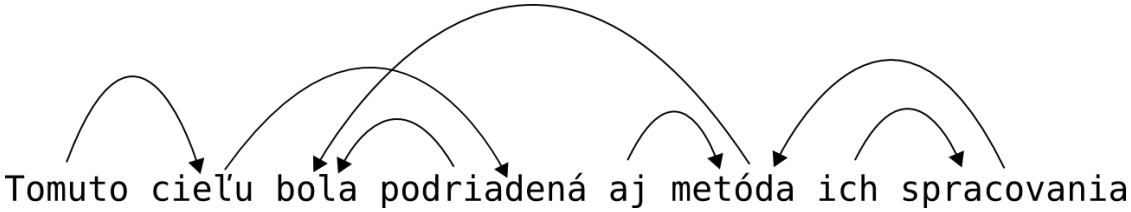
3.1.2 Projektívne a neprojektívne závislostné stromy

Zjednodušene povedané, projektívne závislostné stromu sú tie, ktorých hrany sa nejakým spôsobom nekrížia (McDonald, 2005). Príklad vidíme na obrázku 3.



Obrázok 3 - projektívny závislostný strom

Vidíme, že keď sme si slová lineárne usporiadali do jedného riadku a nakreslili všetky závislosti, žiadna z hrán sa nekrižuje s inou. Takýto typ závislostného stromu je dostačujúci pre jazyky s pevnou štruktúrou vety, ako napríklad angličtina. Penn Treebank je dokonca zostavený iba z takýchto viet, aj napriek tomu, že v angličtine sa dajú skonštruovať syntakticky správne vety, na ktorých syntaktickú reprezentáciu projektívne stromy nestačia. Takéto vety potom musíme reprezentovať neprojektívnymi závislostnými grafmi. Príklad vyextrahovaný zo Slovenského Závislostného Korpusu je na obrázku 4.



Obrázok 4 - neprojektívny závislostný strom

Takýto typ stromu je omnoho vhodnejší pre flektívne jazyky s voľným slovosledom, ako je aj slovenčina. Na základne jednoduchej analýzy Slovenského Závislostného Korpusu sme zistili, že 14,71% viet z tohto korpusu majú vzťahy ktoré sa krížia, tzn. na ich závislostnú reprezentáciu treba neprojektívny závislostný strom. Parser, ktorý by bol schopný vytvárať len projektívne závislostné stromy, by mal teda značnú nevýhodu.

3.2 Morfologická anotácia ako vstup pre syntaktickú analýzu

Syntaktické parsovanie chceme realizovať ako webovú službu. Takéto použite sa dosť líši oproti klasickému vyhodnocovaniu úspešnosti parserov na testovacej vzorke dát, tzv. zlatom štandarde. Testovacie dáta bývajú pred pripravené a správne anotované. Pri webovej službe ale nevieme kontrolovať, aký vstup nám od používateľa príde. Veta nemusí byť nutne správne napísaná, na niektorých miestach môže chýbať diakritika a môže obsahovať gramatické chyby. Vzhľadom na komplexnosť morfologickej anotácie nemôžeme od používateľov očakávať, že ju spolu s vetou poskytnú. A pritom tá je nevyhnutnou súčasťou vstupu pre morfo-

logický parser. Práve preto morfologickú anotáciu vidíme ako jeden z najpravdepodobnejších zdrojov chýb a nepresnosti parsovania.

V testovacej množine dát je morfologická anotácia poskytnutá priamo. Pri vstupe z webovej služby ju budeme musieť doplniť. Ak by sme pre angličtinu zobrali triviálny postup a každému slovu pridelili jeho najčastejšie používanú morfologickú značku, dosiahli by sme úspešnosť až okolo 90% (Charniak, 1997). Slovenčina ale patrí do kategórie vysoko flektívnych jazykov, čo znamená, že morfologická anotácia je zložitejší problém ako napríklad v prípade angličtiny. Typicky pre angličtinu sa počet morfologických značiek ktoré automatické značkovače generujú okolo 50. Napríklad pre češtinu je to ale až 3000 možných morfologických značiek (Collins, 1999). Slovenčina je veľmi príbuzný jazyk a môžeme pri nej očakávať podobný počet. V tejto sekcií predstavíme niekoľko nástrojov pre morfologickú anotáciu slovenského textu.

Dagger

Štatistický tagger, založený na skrytých Markovových modeloch. Hlavný problém pri použití týchto modelov je riedkosť dát (angl. Data sparseness). Teda ak pri klasifikovaní narazíme na inštanciu, ktorá je z hľadiska jazyka úplne správna, ale nenachádza sa v našom natrénovanom modeli, automaticky je jej pravdepodobnosť vyhodnotená ako nulová. Snaha prekonať tento problém sa nazýva zjemňovanie skrytého Markovového modelu. Najzákladnejší spôsob je pridávanie pozitívnej konštanty pre každý prípad - videný aj nevidený. Tým pádom aj nevidené udalosti dostanú nenulovú pravdepodobnosť. Nevýhoda tohto prístupu je v tom, že nevyužíva žiadne znalosti o jazyku a nenulová pravdepodobnosť bude pridelené aj inštanciám, ktoré nie sú valídne vôbec. V Daggeri implementovali dve zjemňovacie metódy, ktoré ťažili z faktu, že väčšina morfologickej informácie je v slovenčine uložená v prípone (Hládek, a iní, 2012). Tento systém v 95,91% prípadov určil správne slovný druh a v 86,16% prípadov určil správne celú morfologickú značku. Je voľne dostupný ako webová služba[2], spolu s možnosťou automatickej korekcie textu, lematizácie a rozpoznávania pomenovaných entít.

TreeTagger

Štatistický tagger vytvorený Helmutom Schmidtom, založený na Markovových modeloch (Schmid, 1994). Bol primárne vytvorený pre nemčinu, ale časom bol natrénovaný pre mnohé iné jazyky vrátane slovenčiny. Pre slovenčinu sú dokonca k dispozícií dva modely – jeden so zjednodušenou množinou morfologických značiek a jeden s kompletnou sadou. Model je

[2] http://nlp.web.tuke.sk/nlpform

voľne dostupný na stiahnutie[3]. Na dátach z nemeckého jazyka dosahoval značkovač úspešnosť až 97.53% (Schmid, 1995).

Slovak POS Tagger

Morfologický značkovač ktorý vznikol na pôde Fakulty Informatiky a Informačných technológií Slovenskej Technickej Univerzity v Bratislave (Mészáros, 2015). Je voľne dostupný ako webová služba[4] s ktorou je možné komunikovať prostredníctvom GET požiadaviek. Výstup je vo formáte XML a okrem morfologickej značky zahŕňa aj lému daného slova.

MATE tools tagger

Na univerzite v Štuttgarde v Inštitúte pre Spracovanie Prirodzeného Jazyka vznikla sada nástrojov na spracovanie prirodzeného jazyka, ktorá je voľne dostupná na internete pod názvom MATE Tools[5]. Obsahuje aj nástroje, pomocou ktorých sa dá vykonať kompletná morfologická analýza:

- POS Tagger - jednotlivým slovám určuje iba slovný druh. Neurčuje zvyšné morfologické vlastnosti,
- Morph Tagger - slovám, ktoré už majú určený slovný druh určí všetky zvyšné morfologické vlastnosti.

Keď vstupnej vete označíme najprv slovné druhy a potom morfologické vlastnosti, dostaneme rovnaký výstup, ako v prípade predchádzajúcich značkovačov (teda kompletnú morfologickú značku). Obidva značkovače sú implementované ako Java konzolová aplikácia a je možné ju využiť priamo z konzoly, prípadne ju integrovať do inej Java aplikácie.

3.2.1 Porovnanie úspešnosti nástrojov na morfologickú anotácií

Presnosť morfologickej anotácie je pre výslednú úspešnosť parsera kritická. Pokúsili sme sa s testovacími dátami nasimulovať prípad, keď je vstup, podobne ako pomyselná veta prichádzajúca do webovej služby, bez morfologickej anotácie.

Všetky nástroje okrem MATE tools značkovača nevyžadovali žiadne trénovanie. Nakoľko pre MATE tools nie je dostupný vopred natrénovaný model pre slovenčinu, museli sme použiť dáta zo Slovenského Závislostného Korpusu (Slovenský závislostný korpus, 2008) na natrénovanie značkovača. Dáta sme rozdelili na testovacie a trénovacie v pomere 1:9, pričom testovaciu vzorku sme použili aj na vyhodnotenie ostatných parserov. Testovacia množina obsahovala celkovo 87 907 slov, ktoré dohromady tvorili 5027 viet.

[3] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
[4] http://morpholyzer.fiit.stuba.sk:8080/PosTagger/
[5] https://code.google.com/p/mate-tools/

Morfologické značky sme následne porovnali s tými ktoré sú zahrnuté v závislostnom korpuse. Výsledky uvádzame v tabuľke 1.

Tabuľka 1 - úspešnosti morfologických anotátorov

	Pokrytie (Coverage)	Presnosť	Úplnosť (Recall)
Dagger	87,23%	76,46%	66,69%
Slovak PosTagger	78,46%	46,29%	36,32%
TreeTagger	100%	91.59%	91.59%
MATE Tools Tagger	100%	88,85%	88.85%

Značkovače, ktoré mali 100% pokrytie a teda by nám boli schopné označkovať všetky potrebné slová boli TreeTagger a MATE Tools Tagger. V prípade TreeTagger tak teda ostáva približne 8% slov, ktoré majú zlú morfologickú značku a teda môžu teoreticky spôsobiť väčšiu nepresnosť pri syntaktickej analýze.

Ako už bolo spomenuté, morfologická analýza nepredstavuje takú výzvu v prípade angličtiny, ale skôr v prípade flektívnych jazykov. V posledných rokoch vznikli práce, ktoré sa snažili vyťažiť zo spojenia morfologickej analýzy a syntaktického parsovania (Cohen, a iní), (Bohnet, 2013), (Bohnet, a iní, 2012). Vo všetkých prípadoch bolo pozorované zlepšenie presnosti aj morfologickej anotácie, aj syntaktického parsovania. To dokazuje, že v prípade flektívnych jazykov, je morfologická a syntaktická úroveň analýzy prepojená.

3.3 Existujúce nástroje pre syntaktickú analýzu

Spracovanie prirodzeného jazyka je oblasť, v ktorej sa výskum koná už roky, pričom drvivá väčšina tohto výskumu sa orientuje na anglický jazyk. To je vcelku prirodzené, nakoľko ide o svetovo rozšírený jazyk a je v ňom napísané obrovské množstvo dokumentov. Výskum je tu už na dosť vysokej úrovni, hlavne vďaka prudkému rozmachu štatistických metód v 90. rokoch a v oblasti syntaktickej analýzy už existujú mnohé zaujímavé práce a výsledky. V tejto časti opíšeme niekoľko významných prác zo zahraničia a zhrnieme práce o syntaktickej analýze, ktoré sa zameriavali na slovenčinu.

Syntaktický analyzátor z FEI TUKE

Na pôde Fakulty elektrotechniky a informatiky Technickej Univerzity v Košiciach vznikol pravidlový syntaktický analyzátor. Systém používa vlastný modul na morfologické značkovanie a na dizambiguáciu slov. Slovenčinu následne parsuje ako bezkontextovú gramatiku. Gramatika obsahuje 26 pravidiel, ktoré boli vyextrahované z Pravidiel Slovenského Pravo-

pisu. Analyzátor vie detegovať 5 typov vetných elementov, a to podmet (vyjadrený aj nevyjadrený), predikát, objekt, príslovky a prívlastky.

Evaluácia prebehla na 127 náhodne vybraných vetách zo Slovenského Národného Korpusu a 100 vetách získaných ako transkripcia z televízneho vysielania. Úspešnosť sa vyhodnocovala pre jednotlivé vetné elementy a pohybovala sa medzi 98,43% pre predikát a 82,60% pre prívlastky. Treba však podotknúť, že výber viet bol obmedzený na vety s počtom slov menším ako 6 a systém neurčoval závislosti medzi jednotlivými vetnými elementmi.

MaltParser

Ide o dátami riadený generátor závislostných parserov. Pracuje so štandardným závislostným treebank formátom a dovoľuje zadefinovať vlastné modely vlastností (z angl. feature models). Empiricky bol overený na švédskom, anglickom, českom, dánskom a bulharskom (Nivre, a iní, 2006) jazyku. U týchto jazykov dosahoval konzistentnú presnosť závislostí 80-90%. MaltParser pozostáva z troch častí:

1. deterministický parsovací algoritmus na budovanie závislostných grafov.
2. na histórií založené modely vlastností pre predikciu ďalšej akcie parsera (pri nedeterministických bodoch),
3. Diskriminatívne strojové učenie na mapovanie histórií na akcie parsera.

V súčasnej verzií je dostupných 9 parsovacích algoritmov, ktoré sa dajú rozdeliť do troch rodín:

1. Nivre - pracuje v lineárnom čase a je limitovaný na projektívne závislostné štruktúry,
2. Convington - pracuje v kvadratickom čase, ale je schopný parsovať neobmedzené závislostné štruktúry. Je možné ho nastaviť tak, aby parsoval iba projektívne štruktúry,
3. Stack - funguje na podobnom princípe ako Nivrého algoritmy, avšak dokáže parsovať aj projektívne aj neprojektívne štruktúry.

Mimo týchto hlavných rodín existujú ešte Planar a 2-Planar algoritmy, ktoré pracujú v lineárnom čase a sú obmedzené na štruktúry, ktoré neobsahujú žiadne krížiace sa linky.

MST Parser

Ide o parser ktorý pracuje s grafmi a hľadanie najvhodnejšej závislostnej štruktúry formalizuje ako hľadanie maximálnej kostry orientovaného grafu. Na vytvorenie modelu z trénovacích dát používa MIRA algoritmus (Margin Infused Relaxed Algorithm - (Crammer, a iní,

2003)) , čo je algoritmus pre klasifikáciu (každý závislostný strom je jedna z tried do ktorej môžeme nové dáta priradiť).

V prvej verzií bol použitý upravený Eisnerov algoritmus na konštrukciu projektívnych závislostných stromov. Systém bol experimentom otestovaný na English Penn Treebank a Czech Prague Dependency Treebank korpusoch. V prípade anglických dát bola presnosť (počet slov, ktoré mali v strome správne priradeného rodiča) 90.9 percent. V prípade českých to bolo 83.3 percent a to bez akýchkoľvek jazykovo-špecifických zmien pre trénovanie modelu (McDonald, a iní, 2005). Boli však nevyhnutné zmeny pre POS tagy v českej množine dát - tie boli zjednodušené, nakoľko bohatý morfologický systém by generoval príliš veľa unikátnych kombinácií POS tagov.

Neprojektívne stromy sa dajú pomocou MST parsera generovať tiež. Nejde dokonca ani o radikálnu úpravu, ale jednoducho o prehľadanie celého priestoru kostier grafu bez obmedzení. V práci (McDonald, 2005) bol namiesto Eisnerovho algoritmu použitý Chu-Liu-Edmonds algoritmus, vyvinutý nezávisle dvoma výskumníkmi, (Chu, a iní, 1965) a (Edmonds, 1967). Oproti pôvodnej verzií bola presnosť zlepšená na 84.4 percent, čo je zlepšenie o 1,1 percento. Testovanie prebiehalo na Prague Dependency Treebank. Zo všetkých závislostných štruktúr v tejto množine dát je menej ako 2% neprojektívnych.

Formát vstupu je buď špecifický pre MST parser, alebo CoNLL formát, ktorý používa aj Malt Parser.

Parsre z balíka MATE tools

Ako už bolo spomenuté v časti o morfologickej analýze, na Univerzite v Štuttgarde je vyvíjaná a udržiavaná sada nástrojov na spracovanie prirodzeného jazyka s názvom MATE Tools. Jej súčasťou sú dokonca až dva syntaktické parsre.

MATE Grafový parser

Tento parser vznikol s cieľom zrýchliť trénovanie a samotné parsovanie, nakoľko okrem presnosti je rýchlosť parsovania tiež jednou z dôležitých vlastností parsera (Bohnet, 2010). Základ pre tento parser tvorí rozšírený Eisnerov algoritmus (Carreras, 2007) a algoritmus pasívno-agresívneho perceptrona (Crammer, 2006) (McDonald, a iní, 2006), ktorý slúžil aj ako základ pre MST parser. Vzhľadom na to, že MATE grafový parser do veľkej miery kopíruje jeho architektúru, dá sa hovoriť o jeho vylepšení, alebo optimalizácií. Hlavná nevýhoda MST parsera spočívala v jeho spotrebe pamäti a voľného miesta na disku. Pri trénovaní parsera bolo potrených 3GB hlavnej pamäte a až 100GB voľného miesta na disku, v prípade angličtiny.

Implementáciou hešovacieho jadra sa podarilo zvýšiť rýchlosť parsovania až 3,5 násobne na jednom jadre. Zároveň hešovacie jadro umožnilo jednoduchšiu paralelizáciu, ktorá v závislosti na procesore parsovanie urýchlila 3,4 až 4,6 násobne. Celkovo bolo zaznamenané až 16 násobné zrýchlenie na anglickej testovacej množine dát, pričom nebola obetovaná úspešnosť a presnosť parsera.

MATE Prechodový parser

Väčšina parsovacích algoritmov predpokladá ako vstup vetu, ktorá už bola morfologicky analyzovaná. Morfologické analyzátory pritom len málo využívajú syntaktické vzťahy, čo sa negatívne prejavuje hlavne pri morfologicky bohatých flektívnych jazykoch. Na využitie vzájomnej interakcie morfológie a syntaxe vznikli parsre, ktoré súčasne dedukujú morfologickú a syntaktickú analýzu (Lee, a iní, 2011).

Jeden z takýchto je implementovaný aj v MATE tools sade nástrojov. Je založený na neprojektívnom prechodovom systéme (Nivre, 2009) a lúčovom vyhľadávaní (Furcy, a iní, 2005). Vzhľadom na to, že pri takomto učení je priestor, ktorý musí algoritmus prehľadávať omnoho väščí, ako pri učení iba syntaktického parsovania, aj nároky na pamäť a výkon sú vyššie. Na úspešné natrénovanie tohto parsera je potrebných až 18GB hlavnej pamäte (pre slovenčinu) a trvá netriviálne dlhú dobu (až niekoľko dní).

3.4 Možné úskalia pri vyhodnocovaní úspešnosti

Jedna z veľmi dôležitých častí syntaktickej analýzy je vyhodnocovanie úspešnosti vytvoreného parsera. Nielen pre samotné overenie úspešnosti metódy, ale aj pre porovnanie úspešnosti s inými parsermi. V rámci štatistických parserov vidíme najčastejšie vyhodnocovanie na zlatom štandarde - testovacia množina dát, často z tej istej domény, ako boli trénovacie dáta. To je aj prípad CoNLL úloh a súťaží. Týmto prístupom je možné veľmi jednoducho a rýchlo porovnať úspešnosť parserov s rovnakým typom výstupu. Problém ale nastáva v prípade, keď chceme porovnať parsery, ktoré nemajú rovnaký formát výstupu - napríklad zložkový a závislostný parser. Každý z nich má rôznu reprezentáciu výslednej syntaxe a konverzia z jedného formátu do druhého nemusí byť vždy presná (Collins, 1999), (Xia, a iní, 2001).

Otázna je aj úspešnosť parserov na surových textoch z webu. Drvivá väčšina parserov už roky býva trénovaných na tých istých korpusoch a pri konfrontácií s diverzifikovanými webovými textami, sa markantne zvyšuje chybovosť parsovania (Open Information Extraction from the Web, 2007). Ako ukazuje aj vyhodnocovanie parserov na Google Web Treebanku, doménová adaptácia na webové texty je stále problém (Petrov, a iní, 2012). Pri

parsovaní blogov, webových diskusií a iných webových zdrojov sa musíme vysporiadať s novou sadou problémov, ako napríklad väčšia frekvencia slangových výrazov, chýbajúca diakritika, preklepy a iné gramatické chyby. Úspešnosti parserov sa prepadli na úroveň 82 až 84 percent, oproti pôvodným 90 percentám.

4 Zhodnotenie súčasného stavu a ciele práce

Syntaktická analýza je všeobecne uznávaná ako jeden z najdôležitejších krokov pri automatickom spracovaní textu a je braná ako základ pre hlbšie pochopenie a analýzu textu. Informácie zo syntaktickej analýzy sú použiteľné a sú súčasťou mnohých iných, aj komerčne využívaných aplikácií. Extrakcia kolokácií, identifikácia frazeologických pomenovaní, strojový preklad, extrakcia informácií a štruktúrovaných dát zo surového textu, analýza sentimentu sú len niektoré príklady iných disciplín, ktorým syntaktická analýza poskytuje užitočné informácie.

Vzhľadom na potenciál tejto výskumnej oblasti sme svedkami aj veľkému záujmu výskumníkov o túto problematiku. Z prácne vytváraných pravidlových systémov s veľkými obmedzeniami sa výskum prepracoval až k štatistickým metódam. Vytvorenie veľkých syntakticky anotovaných korpusov a nárast výpočtovej sily obyčajných počítačov mal za následok rapídny rozmach metód založených na strojov učení a postupne vznikli parsery, ktoré dosahovali úspešnosť okolo 90%. Vytvorenie takéhoto parsera nevyhnutne vyžaduje natrénovaný model. Výskum sa teda prirodzene orientoval na jazyky, pre ktoré takéto dáta vhodné na trénovanie existovali, pričom primárne to bola angličtina (čo je vzhľadom na rozšírenosť jazyka, najmä v digitálnom priestore úplne prirodzené). Vďaka projektu Praque Dependency Treebank sa čestina postupom času stala akýmsi reprezentantom slovanských a flektívnych jazykov.

Pre slovenčinu dlho nebola k dispozícií vhodná syntakticky anotovaná množina dát, napriek existujúcemu záujmu o parsovanie. Práce ktoré tu vznikali sa teda nevyhnutne radili medzi pravidlové systémy. Vďaka projektu Slovenského Závislostného Korpusu je ale konečne možné naplno využiť roky skúmané štatistické metódy a metódy strojového učenia na experimentovanie s parsovaním slovenčiny.

Okrem samotného parsovania je zaujímavá aj otázka vyhodnocovania úspešnosti parserov. Očividný problém pri trénovaní na dátach z jedného korpusu je úspešnosť na textoch, ktoré nie sú z toho istého korpusu, resp. sú z úplne inej domény. Parsovanie naprieč rôznymi doménami je stále výzva, osobitný problém predstavuje parsovanie webových textov. Okrem otázky o rôznych doménach je zaujímavé sledovať aj to, ako sa do úspešnosti prasovania prejavia rôzne deformácie textu, ktoré sú pre textový obsah generovaný bežnými používateľmi na webe typické.

Na základe tohto stavu a otvorených problémov sme postupne sformovali ciele tejto práce. Pri ich vytváraní sme dbali sme na to, aby reflektovali súčasný stav a riešili aktuálne

otázky syntaktickej analýzy, ale aby výstup bol zároveň aj použiteľný pre ďalšie aplikácie a tak možno zväčšil záujem o výskum v tejto oblasti zameraný na slovenčinu. Ciele tejto práce sú:

- vybrať z pomedzi uvedených nástrojov na morfologickú anotáciu a syntaktickú analýzu najvhodnejšiu kombináciu na parsovanie slovenčiny,
- experimentovaním zistiť možné negatívne dopady iného vstupu ako zo zlatej množiny dát – deformovaného vstupu z needitovaných webových textov,
- preskúmať nástroje na opravu textu a experimentálne overiť ich vplyv na úspešnosť syntaktickej analýzy,
- bližšie zanalyzovať výsledky parsovania, zamerať sa na špecifický typ vzťahu a navrhnúť metódu na zlepšenie úspešnosti určovania daného vzťahu,
- realizovať parser aj s experimentálne overenými vylepšeniami ako verejne dostupnú webovú službu.

V ďalších kapitolách sa osobitne venujeme každému z uvedených cieľov.

5 Vyhodnotenie existujúcich parserov

V tejto kapitole opíšeme dáta s ktorými sme pri trénovaní a testovaní parserov pracovali, rôzne formáty vstupov pre parsre a základné metriky, ktoré sa používajú pri vyhodnocovaní parserov. Nakoniec uvedieme úspešnosť nami skúmaných parserov na dátach zo Slovenského Závislostného Korpusu a uvedieme problémy, ktoré sa môžu pri vyhodnocovaní parserov vyskytnúť.

Na prácu s dátami sme implementovali konzolovú aplikáciu Syncor. Prostredníctvom nej sme sparsovali Slovenský Závislostný Korpus do databázy. Ďalej je v nej implementovaná funkcionalita konverzie dát do rôznych formátov, rozdelovanie na trénovaciu a testovaciu množinu, odstraňovanie diakritiky z množiny dát a podrobné analýzy nad označkovanou množinou dát. Podrobne túto aplikáciu opisujeme v prílohe A.

5.1 Opis dátových rozhraní

V tejto kapitole uvádzame opis dátového formátu v ktorom je uložený Slovenský Závilostný Korpus a následne uvádzame prehľad dátových rozhraní s ktorými pracujú vyššie uvedené parsre.

Slovenský Závislostný Korpus

Na rozdiel od nástrojov na morfologickú anotáciu, ani jeden z nástrojov na syntaktickú analýzu nemal vopred natrénovaný model pre slovenčinu. Na natrénovanie potrebných modelov sme podobne ako pre morfologické značkovače použili Slovenský Závislostný Korpus. Ten je uložený v trojúrovňovom formáte Pražského závislostného korpusu (Prague Dependency Treebank) - každá úroveň predstavuje jednu vrstvu analýzy a je uložená v samostatnom súbore. Jednotlivé úrovne sú:

- úroveň slov - v týchto súboroch sa nachádzajú slová viet forme v akej sa vyskytujú vo vete,
- morfologická úroveň - v týchto súboroch sa nachádzajú slová viet, ktoré sú naviac aj morfologicky zanalyzované - obsahujú navyše lému slova a morfologickú značku,
- analytická úroveň - v týchto súboroch sa nachádzajú vety, ktoré sú syntakticky zanalyzované - pre jednotlivé slová sú určené nadradené vetné členy a funkcie vo vete.

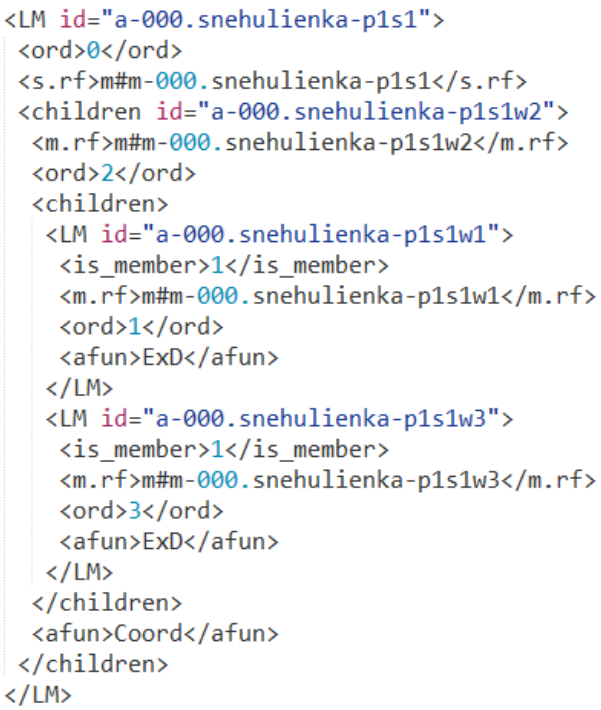
Ukážky jednotlivých úrovní vidíme na obrázku 5, obrázku 6 a obrázku 7.

```
<para>
  <othermarkup origin="csts/doc/p/@n">1</othermarkup>
  <w id="w-000.snehulienka-p1s1w1"><token>Zrkadlo</token></w>
  <w id="w-000.snehulienka-p1s1w2"><token>,</token></w>
  <w id="w-000.snehulienka-p1s1w3"><token>zrkadlo</token></w>
</para>
```

Obrázok 5 – veta zo Slovenského Závislostného Korpusu zapísaná na úrovni slov

```
<s id="m-000.snehulienka-p1s1">
  <m id="m-000.snehulienka-p1s1w1">
    <w.rf>w#w-000.snehulienka-p1s1w1</w.rf>
    <form>Zrkadlo</form>
    <tag>SSns1</tag>
    <lemma>zrkadlo</lemma>
    <src.rf>manual</src.rf>
  </m>
  <m id="m-000.snehulienka-p1s1w2">
    <w.rf>w#w-000.snehulienka-p1s1w2</w.rf>
    <form>,</form>
    <tag>Z</tag>
    <lemma>,</lemma>
    <src.rf>manual</src.rf>
  </m>
  <m id="m-000.snehulienka-p1s1w3">
    <w.rf>w#w-000.snehulienka-p1s1w3</w.rf>
    <form>zrkadlo</form>
    <tag>SSns1</tag>
    <lemma>zrkadlo</lemma>
    <src.rf>manual</src.rf>
  </m>
</s>
```

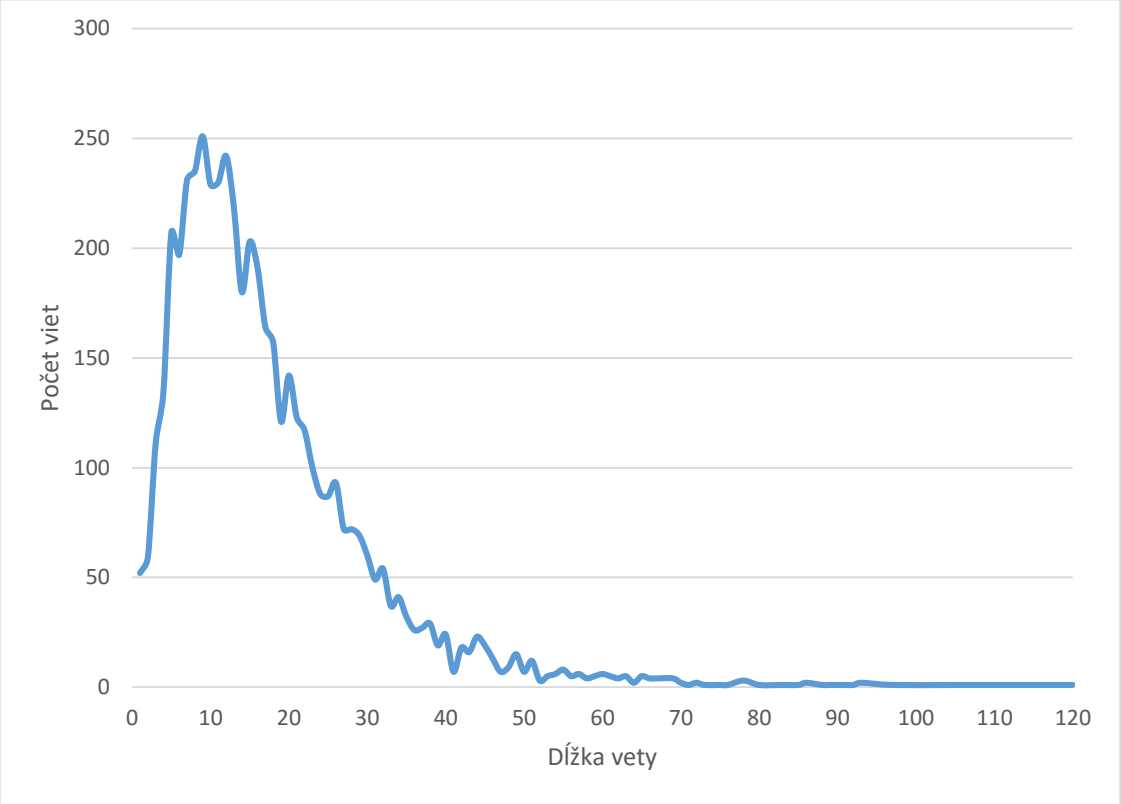
Obrázok 6 – veta zo Slovenského Závislostného Korpusu zapísaná na morfologickej úrovni



Obrázok 7 - veta zo Slovenského Závislostného Korpusu zapísaná na analytickej úrovni

V poslednej verzií Pražského Závislostného Korpusu bola ešte pridaná tektogramatická úroveň. Tá už do istej miery pracuje aj so sémantikou a dosť sa od analytickej úrovne líši. Zatiaľ čo v analytickej úrovni je každé slovo vo vete uzol závislostného stromu, v tektogramatickej analytickej úrovni to tak už nie je – uzol je iba každé autosémantické slovo (Hajič, 2005). Tektogramtická úroveň sa ale v súčasnej verzií Slovenského Závislostného Korpusu nenachádza. Celkovo sa v množine dát nachádza 846 489 tokenov, ktoré tvoria dohromady 50 276 viet. Dĺžka viet je premenlivá, najkratšie vety majú iba jeden token (celkovo ide o 52 viet) a najdlhšie majú 120. Z distribúcie dĺžok znázornenej na obrázku 8 vidíme, že najpočetnejšie sú vety s dĺžkou okolo 10 tokenov.

Obrázok 8 - počet viet jednotlivých dĺžok



CoNLLX

Ide o dátový formát (Bucholz, a iní, 2006), ktorý ako vstupný a výstupný používa Malt Parser. Každé slovo (token) vo vstupnom súbore je na osobitnom riadku. V riadku sa nachádza 10 polí, oddelených znakom tabulátora:

- ID - identifikátor a poradie tokenu vo vete, počíta sa od 1,
- FORM – forma slova alebo interpunkcie tak, ako sa píše vo vete,
- LEMMA - léma slova,
- CPOSTAG - hrubé morfologické označenie - slovný druh,
- POSTAG - jemné morfologické označenie - všetky morfologické vlastnosti,
- FEATS - množina morfologických a syntaktických vlastností pre daný token, v našom prípade ide len o rozvinutú morfologickú značku,
- HEAD – ID nadradeného vetného člena pre daný token,
- DEPREL - syntaktický vzťah medzi daným a nadradeným tokenom,
- PHEAD – projektívny HEAD pre daný token - závisostnú štruktúra vytvorená pomocou tochto pola je vždy projektívna, v našom prípade toto pole ostáva prázdne,

- PDEPREL – syntaktický vzťah medzi daným tokenom a jeho prediktívnym nadradeným vetným členom, v načom prípade toto pole ostáva prázdne.

Jednotlivé vety sú potom vo vstupnom súbore oddelené prázdnym riadkom. Ukážku jednoduchej vety zapísanej v tomto formáte vidíme na obrázku 9:

```
1 Zrkadlo zrkadlo S S;Sns1  pos=S|par=;|gen=S|num=n|case=s  2 ExD _ _
2 , , Z Z pos=Z 4 ExD _ _
3 zrkadlo zrkadlo S SSns1 pos=S|par=S|gen=n|num=s|case=1  2 ExD _ _
4 povedzže  povedzže  V VMdsb+  pos=V|vform=M|aspect=d|num=s|per=b|neg=a  0 ExD _ _
5 mi  ja  P PPhs3 pos=P|par=P|gen=h|num=s|case=3  4 Obj _ _
6 , , Z Z pos=Z 8 AuxX  _ _
7 ktože ktože P PFms1 pos=P|par=F|gen=m|num=s|case=1  8 Sb  _ _
8 je  byť V VKesc+  pos=V|vform=K|aspect=e|num=s|per=c|neg=a  4 AuxX  _ _
9 najkrajší pekný A AAms1z  pos=A|par=A|gen=m|num=s|case=1|degree=z 8 Pnom  _ _
10  v v E Eu6 pos=E|form=u|case=6 8 Atr _ _
11  našej náš P PFfs6 pos=P|par=F|gen=f|num=s|case=6  12  Atr _ _
12  zemi  zem S SSfs6 pos=S|par=S|gen=f|num=s|case=6  10  Atr _ _
13  ? ? Z Z pos=Z 0 AuxK  _ _
```

Obrázok 9 - ukážka vety zapísanej v CoNLLX formáte

CoNLL09

Ide o dátový formát[6] ktorý používajú parsre z balíka MATE Tools. Štruktúra je podobná ako pri CoNLLX formáte, je zachované jedno slovo na riadok a polia pre slovo oddelené znakom tabulátora. Líšia sa však v poradí a vo význame niektorých polí:

- ID - identifikátor a poradie tokenu vo vete, počíta sa od 1,
- FORM – forma slova alebo interpunkcie tak, ako sa píše vo vete,
- LEMMA - léma slova,
- PLEMMA - predikovaná léma slova – do tohto poľa zapisuje nástroj na automatickú lematizáciu,
- POS - slovný druh (odpovedá CPOSTAG poľu v CONLLX formáte),
- PPOS - predikovaný slovný druh – do tohto poľa zapisuje nástroj na automatické značkovanie slovných druhov,
- FEAT - množina vlastností pre daný token, podobne ako pri CONLLX formáte ide o morfologické a syntaktické vlastnosti,
- PFEAT - predikovaná množina vlastností, do tohto poľa zapisuje nástroj na automatickú morfologickú anotáciu,
- HEAD – ID nadradeného tokenu vo vete
- PHEAD - predikované ID nadradeného tokenu vo vete, do tohto poľa zapisuje syntaktický parser

[6] http://ufal.mff.cuni.cz/conll2009-st/task-description.html

- DEPREL – typ vzťahu daného tokenu s nadradeným tokenom,
- PDEPREL – predikovaný typ vzťahu daného tokenu s nadradeným tokenom, do tohto poľa taktiež zapisuje syntaktický parser.

Okrem týchto polí dátový formát ešte opisuje FILLPRED, PRED a 16 APRED polí, ktoré ale v našom prípade nepoužívame. Príklad vety v tomto formáte uvádzame na obrázku 10.

```
1 Zrkadlo zrkadlo zrkadlo S S pos=S|par=;|gen=S|num=n|case=s  pos=S|par=;|gen=S|num=n|case=s  2 2 ExD ExD
2 , , , Z Z pos=Z pos=Z 4 4 ExD ExD
3 zrkadlo zrkadlo zrkadlo S S pos=S|par=S|gen=n|num=s|case=1  pos=S|par=S|gen=n|num=s|case=1  2 2 ExD ExD
4 povedzže  povedzže  povedzže  V V pos=V|vform=M|aspect=d|num=s|per=b|neg=a  pos=V|vform=M|aspect=d|num=s|per=b|neg=a  0 0 ExD ExD
5 mi  ja  ja  P P pos=P|par=P|gen=h|num=s|case=3  pos=P|par=P|gen=h|num=s|case=3  4 4 Obj Obj
6 , , , Z Z pos=Z pos=Z 8 8 AuxX  AuxX
7 ktože ktože ktože P P pos=P|par=F|gen=m|num=s|case=1  pos=P|par=F|gen=m|num=s|case=1  8 8 Sb  Sb
8 je  byť byť V V pos=V|vform=K|aspect=e|num=s|per=c|neg=a  pos=V|vform=K|aspect=e|num=s|per=c|neg=a  4 4 AuxX  AuxX
9 najkrajší pekný pekný A A pos=A|par=A|gen=m|num=s|case=1|degree=z pos=A|par=A|gen=m|num=s|case=1|degree=z 8 8 Pnom  Pnom
10  v v v E E pos=E|form=u|case=6 pos=E|form=u|case=6 8 8 Atr Atr
11  našej náš náš P P pos=P|par=F|gen=f|num=s|case=6  pos=P|par=F|gen=f|num=s|case=6  12  12  Atr Atr
12  zemi  zem zem S S pos=S|par=S|gen=f|num=s|case=6  pos=S|par=S|gen=f|num=s|case=6  10  10  Atr Atr
13  ? ? ? Z Z pos=Z pos=Z 0 0 AuxK  AuxK
```

Obrázok 10 - veta zapísaná v CoNLL09 formáte

Formát pre MST Parser

Dátový formát pre MST parser sa od CONLL formátov výrazne líši. Každá veta je vo vstupnom súbore opísaná na 4 riadkoch:

- prvý riadok obsahuje jednotlivé slová a interpunkciu, oddelenú znakom tabulátora,
- druhý riadok obsahuje morfologickú značku, alebo iba slovný druh,
- tretí riadok obsahuje analytickú funkciu, resp. typ vzťahu s nadradeným tokenom (DEPREL v CONLL formátoch)
- štvrtý riadok obsahuje číslo určujúce poradie nadradeného tokenu vo vete (HEAD v CONLL formátoch). 0 znamená koreň stromu.

Vidíme že formát je značne jednoduchší ako príslušné CONLL formáty. Nie je tu možné určiť aj jemnú morfologickú značku (iba slovný druh) aj hrubú morfologickú značku (všetky vlastnosti) zároveň.

5.2 Základné metriky pri vyhodnocovaní úspešnosti

Istý štandard pre vyhodnocovanie úspešnosti závislostného parsovania vznikol popri zdielaných úlohách CONLL (Green, 2011), (Nilsson, a iní, 2007), (Tjong Kim Sang, a iní, 2003), (Carreras, a iní, 2005). Základné metriky sú:

- LAS - skóre označeného spojenia (angl. labeled attachment score). Určuje koľkým tokenom sa správne určil nadradený token (HEAD) a aj typ vzťahu (DEPREL),

- UAS - skóre neoznačeného spojenia (angl. unlabeled attachment score). Určuje koľkým tokenom sa správne určil nadradený token (HEAD), typ vzťahu nás v tomto prípade nezaujíma,
- total LAS - určuje počet viet, v ktorých sa podarilo každému tokenu určiť aj nadradený token aj typ vzťahu
- total UAS - určuje počet viet, v ktorých sa poradilo každému tokenu určiť nadradený token. Typ vzťahu nás v tomto prípade nezaujíma.

5.3 Úspešnosti parserov na Slovenskom Závislostnom Korpuse

Pre výber vhodného parsera sme vykonali porovnanie úspešnosti parserov na dátach zo Slovenského Závislostného Korpusu. Dáta sme konvertovali na formát potrebný pre konkrétny parser a následne rozdelili na testovaciu a trénovaciu množinu v pomere 1:9. Úspešnosť sme vyhodnocovali podľa vyššie uvedených štandardných metrík pre závislostné parsovanie. Prehľad úspešností uvádzame v tabuľke 2.

Tabuľka 2 - porovnanie úspešnosti jednotlivých parserov

	MaltParser	MST Parser hrubý	MST Parser jemný	MATE parser grafový	MATE parser prechodový
LAS	57,06%	56,40%	57,77%	**65,28%**	59,84%
UAS	75,32%	77,58%	78,24%	**82,09%**	78,88%
tLAS	6,00%	5.63%	6,79%	**8.84%**	13,15%
tUAS	14,70%	27,38%	29,05%	**36,28%**	32,27%

Malt parser ponúka viacero algoritmov, ktoré môže použiť pri trénovaní a parsovaní. Podľa Červenová 2014 je pre slovenčinu najúspešnejší Stack-Lazy algoritmus, takže sme parser natrénovali práve týmto algoritmom. Vzhľadom na to, že dátový formát MST parsera umožňuje iba jeden typ morfologickej značky (jemný alebo hrubý), natrénovali sme dvakrát - raz s hrubou množinou morfologických značiek (teda iba slovné druhy) a raz s jemnou množinou morfologických značiek (všetky morfologické vlastnosti). Asi neprekvapí, že výsledky s modelom natrénovaným na jemnej množine morfologických značiek bol mierne úspešnejší.

Najlepší sa ukázal grafový parser z balíka MATE Tools a to s dosť výrazným rozdielom. Preto budeme na ďalšie experimenty a implementáciu používať práve ten. Prechodový parser z balíka MATE Tools vykonávajúci súčasne morfologickú anotáciu a syntaktickú

analýzu prekvapivo zaostával za novším grafovým parserom, ktorý syntaktickú analýzu vykonáva separátne.

5.4 Podrobnejšie štatistiky nad výsledkami parsovania

Okrem štandardných metrík sme sa ešte pozreli aj na úspešnosť označovania vzťahov s nadradeným vetným členom. V tejto kapitole uvedieme štatistiky len pre najúspešnejší parser z vyhodnocovania, teda grafový parser z MATE Tools balíka. V tabuľke 3 uvádzame úspešnosť vyhodnocovania jednotlivých analytických funkcií.

Tabuľka 3 - prehľad analytických funkcií (DEPREL) a úspešnosti ich určovania

Názov analytickej funkcie	Celkový počet výskytov v testovacej množine	Počet správne označených výskytov v testovacej množine	Percentuálne vyjadrenie úspešnosti
AuxK	9884	9779	98,94%
AuxV	3102	2551	82.24%
AuxG	9314	6705	80,65%
AuxT	4970	3993	80,34%
AuxX	15275	12029	78,75%
Atr	44329	33465	75,49%
AuxP	7533	5627	74,70%
Sb	10145	7410	73,04%
Adv	21322	14207	66,63%
Obj	15883	10295	64,82%
Pnom	1769	1059	59,86%
Pred	6409	3785	59,06%
Coord	3741	2121	56,70%
AuxZ	4059	2281	56,19%
AuxC	2133	1055	49,46%
AuxY	4636	2226	48,02%
ExD	4526	1754	38,75%
Apos	443	138	31,15%
AtvV	167	40	23,95%
Atv	280	50	17,86%
AuxR	1186	163	13,74%
AuxO	252	24	9,52%
AtrAdv	87	1	1,15%
AtrAtr	27	0	0%
AdvAtr	11	0	0%
AtrObj	1	0	0%
???	148	0	0%
ObjAtr	4	0	0%

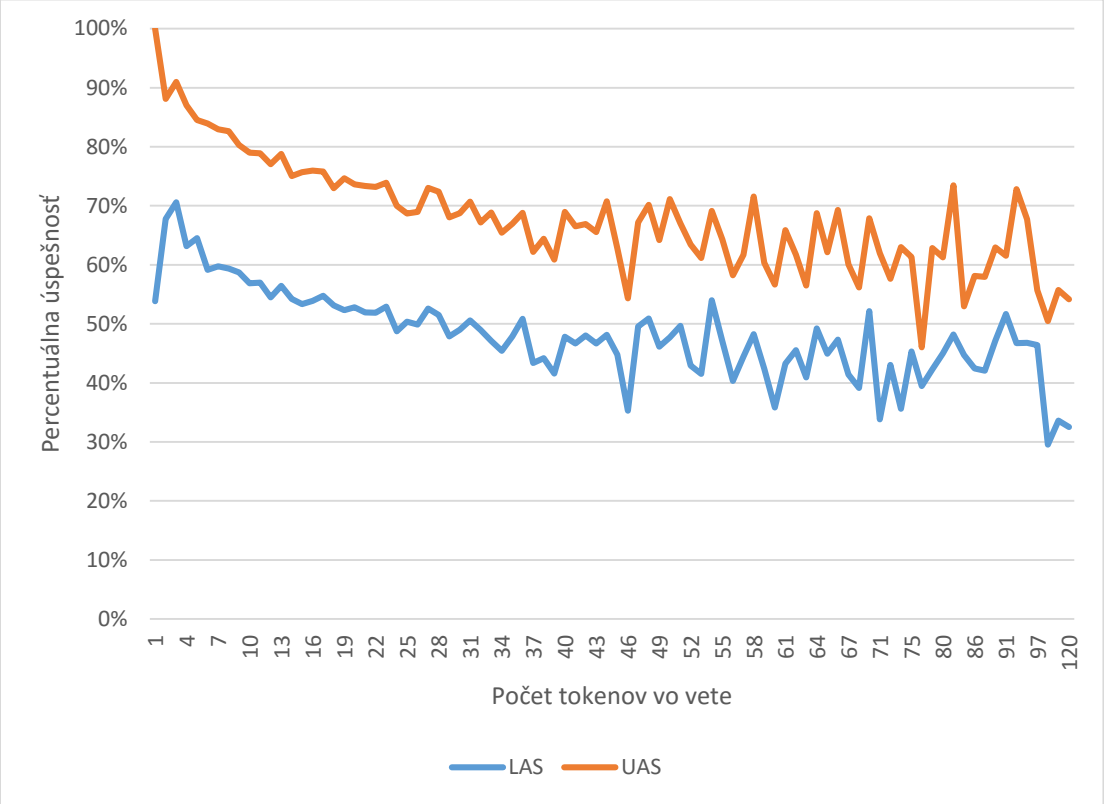
Kompletný prehľad analytických funkcií aj s ich významom uvádzame v prílohe.

Jednoznačne najúspešnejší bol parser pri určovaní koncovej interpunkcie vety (funkcia AuxK). Najčastejšie vyskytujúca sa funkcia v testovacej množine dát bol prívlastok (funkcia Atr). V jeho prípade bola úspešnosť už nižšia a to 75,49%. Parser si ju pritom najčastejšie mýlil s primárnou predložkou, alebo s časťou sekundárnej predložky (funkcia AuxP). Kompletný prehľad chýb pre jednotlivé analytické funkcie uvádzame v prílohe D. Takýto prehľad nám umožní lepšie sa zamerať na jednotlivé problémové vzťahy a navrhnúť zlepšenie ich určovania.

Ďalšou zaujímavou informáciou je úspešnosť vzhľadom na dĺžku vety. Na obrázku 11 vidíme vizualizáciu úspešností metrík LAS a UAS. S rastúcim počtom viet môžeme sledovať mierny klesajúci trend v úspešnosti. Čo je ale zaujímavé, je prudký nárast výkyvov a lokálnych extrémov, ktorý začína približne pri vetách dĺžky 40 a vyššie. Predpokladáme, že je to spôsobené tým, že od tejto dĺžky nám rapídne klesá počet viet (ako je vidieť aj na obrázku 8). Čím menej viet, tým viac sa prejaví významný ojedinelý neúspech, prípadne úspech.

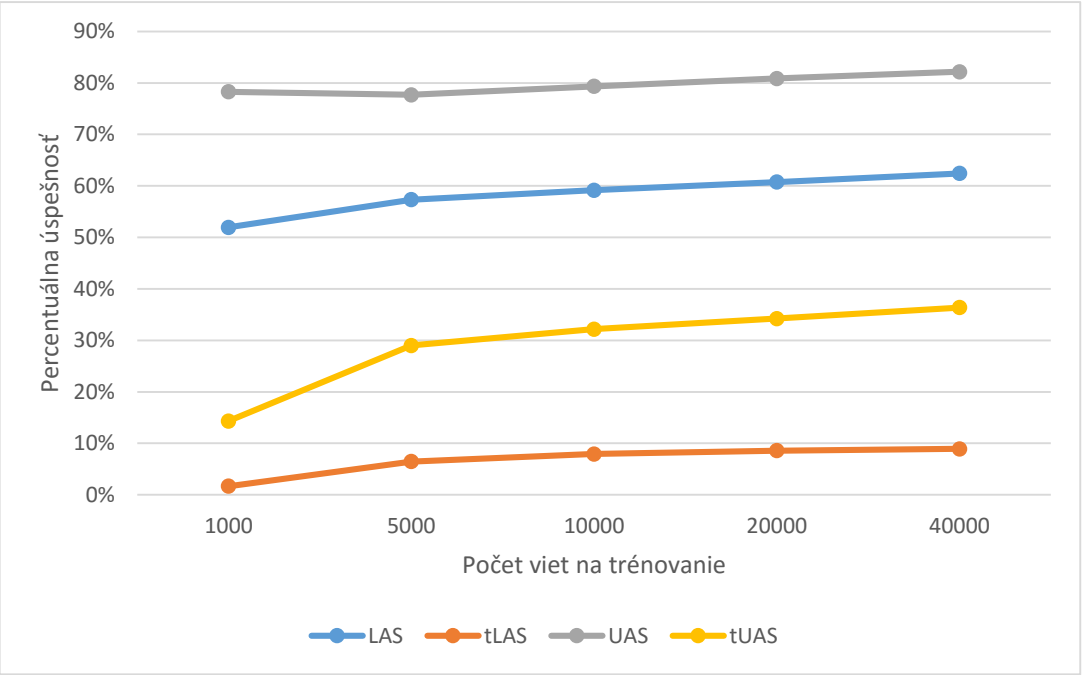
Druhá zaujímavosť vzhľadom na rozloženie viet podľa dĺžky je, že nevidíme žiadnu koreláciu medzi počtom trénovacích viet danej dĺžky a úspešnosti parsovania danej dĺžky. V rozložení viet (obrázok 8) sú najpočetnejšie vety s dĺžkou okolo 10 tokenov, z obrázku 11 ale vidíme, že v tejto oblasti sa nenachádza žiadne lokálne maximum.

Obrázok 11 - úspešnosť parsovania vzhľadom na dĺžku vety



Ďalší údaj ktorý nás pre účely neskoršieho experimentovania zaujíma je úspešnosť parsovania pre rôzne veľkosti trénovacej množiny. Trénovanie parsera je časovo dosť náročná operácia - na súčasnom priemernom notebooku (8GB RAM a procesorom Intel Core i7-4510u[7]) trvá trénovanie Mate grafového parsera na deviatich desatinách všetkých dát približne 30 hodín. Možnosť trénovať parser na menšej vzorke dát, bez markantného prepadu úspešnosti by nám výrazne uľahčil následné experimentovanie. Vykonali sme teda meranie, pri ktorom sme parser postupne natrénovali na rôzne veľkých vzorkách dát. Výsledky sú vizualizované na obrázku 12.

[7] http://ark.intel.com/products/81015/Intel-Core-i7-4510U-Processor-4M-Cache-up-to-3_10-GHz



Obrázok 12 - úspešnosti parsovania pri rôznom počte trénovacích viet

5.4.1 Iterácie trénovania a krížová validácia

Pri trénovaní parsera je možné nastaviť aj počet tréningových iterácií. Pri každej ďalšej iterácií sa model parsera upravuje a dolaďuje. Výsledky uvádzané v tabuľke 2 sú pre prednastavený počet iterácií, teda desať. Aby sme zistili akú úspešnosť parser dosiahne v prípade, že znížime počet iterácií, rozhodli sme sa ho natrénovať ešte raz, s minimálnym počtom iterácií, teda jednou. Rozdiel znázorňujeme v tabuľke 4.

Tabuľka 4 - rozdiel úspešnosti v závislosti od počtu trénovacích iterácií

Počet iterácií	LAS	tLAS	UAS	tUAS
1	61,88%	8,77%	81,32%	35,46%
10	65,28%	8,84%	82,09%	36,28%

Vidíme, že úspešnosť bola v prípade jednej iterácie o dosť horšia, čo je ale zaujímavé, tak UAS sa prepadol iba o necelé jedno percento, zatiaľ čo LAS sa prepadlo až o takmer 4 percentá. To znamená, že vyšší počet iterácií má pozitívny vplyv na určovanie analytických funkcií (*deprel* tagov), ale na určovanie samotného závislostného stromu má vplyv omnoho menší.

Aby sme zaručili korektnosť pri vyhodnocovaní výsledkov, vykonali sme ešte nad dátami desať násobnú krížovú validáciu, aby sme zistili, či sa výsledky príliš nelíšia v závislosti od

toho na akej časti dát bol model natrénovaný. V prípade veľkej odchýlky by to spôsobilo komplikácie v ďalších experimentoch a rapídne by sa zvýšila časová náročnosť pre ich vykonávanie. Pri trénovaní sme nastavili iba jednu iterácií (hlavne z časových dôvodov), nakoľko veríme, že v prípade väčšieho počtu iterácií budú výsledky analogické. Kompletné výsledky vyhodnocovania úspešnosti jednotlivých zložiek krížovej validácie uvádzame v tabuľke 5.

Tabuľka 5 - výsledky desať násobnej krížovej validácie

Beh	LAS	tLAS	UAS	tUAS
1	60,62%	7,87%	80,59%	32,58%
2	61,88%	8,77%	81,32%	35,46%
3	61,63%	8,69%	80,85%	35,30%
4	62,86%	8,08%	81,21%	33,51%
5	61,86%	8,83%	81,66%	36,44%
6	62,69%	9,38%	82,10%	37,13%
7	61,81%	9,45%	81,09%	35,95%
8	61,67%	8,24%	82,26%	34,16%
9	61,19%	8,65%	81,45%	32,91%
10	61,25%	8,02%	81,09%	35,35%

Štatistické ukazovatele, konkrétne rozptyl a smerodajná odchýlka sú nasledovné:

Rozptyl:

- LAS: 0,30%,
- tLAS: 0,27%,
- UAS: 0,24%,
- tUAS: 2,08%.

Smerodajná odchýlka:

- LAS: 0,55%,
- tLAS: 0,52%,
- UAS: 0,49%,
- tUAS: 1,44%.

Zo štatistických ukazovateľov vidíme, že rozdiely medzi jednotlivými behmi krížovej validácie nie sú až tak veľké, aby sme boli nútení vykonávať krížovú validáciu pre každý experiment, ktorý vykonávame v ďalších častiach tejto práce. To nám veľmi uľahčí vyhodnocovanie experimentov a umožní ich flexibilnejšie vykonávanie.

6 Vplyv reálnych textov na úspešnosť parsovania

Ako už bolo spomenuté, webové texty predstavujú výzvu pre syntaktické parsovanie (Petrov, a iní, 2012), ktoré sa doteraz pri vyhodnocovaní obmedzovalo hlavne na silno editované texty z domén publicistiky alebo beletrie. Okrem toho že tieto texty sú manuálne kontrolované na chyby, v korpusoch často obsahujú aj manuálnu morfologickú anotáciu. Pri parsovaní reálnych textov sa nemôžeme spoliehať na to, že nebudú obsahovať chyby a pri použití parsera ako webovej služby potrebujeme doplniť aj morfologickú anotáciu.

Pre angličtinu je dostupný syntakticky anotovaný korpus[8], ktorý overovanie parserov na webových textoch značne uľahčuje. Pre slovenčinu takýto korpus dostupný nemáme (existuje korpus webových textov, ale nie je syntakticky anotovaný) a Slovenský Závislostný Korpus sa z hľadiska štýlovo žánrovej štruktúry skladá z:

- publicistických textov,
- beletrie,
- populárno-náučného štýlu,
- odborných textov,

Čo sú tiež štýly, ktoré z pravidla podliehajú manuálnej kontrole. Aby sme teda mohli skúmať úspešnosť parsera na vstupe, ktorý reprezentuje reálne webové texty, musíme si množinu dát dodatočne upraviť pridaním chýb.

Problematike pridávania chýb sa venuje viacero prác, napríklad (Felice, a iní, 2014) a (Foster, a iní, 2009). Motivácia pre takéto upravovanie textu je napríklad generovanie textov pre účely výučby jazyka alebo overovania úspešnosti kontrol pravopisu. Podobne ako v mnohých iných oblastiach sa dajú práce rozdeliť na pravidlové, alebo štatistické so strojovým učením.

Ako prvé sme sa rozhodli vyskúšať, akú úspešnosť bude mať parser, ak namiesto manuálnej morfologickej anotácie zo Slovenského Závislostného korpusu použijeme automatický morfologickú anotáciu získanú pomocou nástroja TreeTagger. V testovacej vzorke sme nahradili všetky pôvodné morfologické značky našimi automaticky získanými a porovnali úspešnosť s pôvodným parsovaním. Výsledky uvádzame v tabuľke 6.

[8] https://catalog.ldc.upenn.edu/LDC2012T13

Tabuľka 6 – porovnanie úspešnosti pred a po použití vlastnej morfologkcej anotácie

	LAS	tLAS	UAS	tUAS
Pôvodná úspešnosť	65,28%	8,84%	82,09%	35,28%
Úspešnosť s morf. anotáciou z TreeTaggera	61,14%	8,84%	82,15%	35,92%

Pre porovnanie s pôvodnými výsledkami, LAS sa prepadol až o viac ako 4 percentá, UAS paradoxne stúpol o pár stotín percenta. Vzhľadom na to, že v ďalších experimentoch budeme nutne potrebovať vlastnú morfologickú anotáciu, tak úspešnosť parsovania s morfologickou anotáciou z TreeTaggera použijeme ako novú základnú úroveň, s ktorou budeme porovnávať úspešnosť parsovania.

6.1 Chýbajúca diakritika

Ďalší experiment ktorý sme vykonali súvisel s diakritikov. Opäť s grafový parserom z MATE Tools balíka a TreeTagger sme vykonali dve merania. Najprv sme zo vstupu odstránili všetky diakritické znamienka. Očakávame, že pri takomto vstupe sa mierne zhorší morfologická anotácia, čo sa následne premietne do úspešnosti parsera.

Keď sme zbavili vstup pre TreeTagger diakritiky, zaznamenali sme markantný prepad úspešnosti. Pokrytie značkovača zostalo 100%, ale presnosť a úplnosť sa prepadla až 64,58%. Následne sme takto označkovaný výstup sparsovali pomocou MATE Tools grafového parsera.

Tabuľka 7 - úspešnosť parsovania pri chýbajúcej diakritike

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	61,14%	8,84%	82,15%	35,92%
Úspešnosť s odstránenou diakritikou a s morf. anotáciou z TreeTaggera	49,58%	5,87%	69,02%	21,62%

Z tabuľky 7 vidíme, že zlá morfologická anotácia, ktorá vznikla v dôsledku chybnej diakritiky sa výrazne podpísala pod horšiu úspešnosť parsovania. LAS sa prepadol až o 11%, UAS o 14%.

6.2 Preklepy

Ako už bolo spomenuté, existujú práce, ktoré sa pre svoje potreby zaoberajú aj generovaním rôznych typov chýb. Rozlišujeme dve základné skupiny chýb:

- chyby pri ktorých náhodou vznikne slovo (z angl. real-word, ďalej uvádzame pod skratkou rw), ktoré je gramaticky správne (napríklad: *zástavka – zastávka)*,
- pridaním chyby vznikne slovo, ktoré nie je gramaticky správne (z angl. non-word, dalej uvádzame pod skratkou nw). Vznike „neexistujúce slovo“, (napríklad: *zástavka – zstavka*).

Vzhľadom na to, že so súčasne dostupnými nástrojmi nevieme opraviť rw chybu, zameriavame sa pri generovaní preklepov na nw chyby. Podľa (Damerau, 1964) 80% chýb spadajú do jednej z týchto štyroch kategórií:

- vloženie – pridanie jedného znaku, ktorý v slove nemal byť,
- zmazanie – vynechanie jedného znaku, ktorý v slove mal byť,
- nahradenie – nahradenie jedného znaku, ktorý mal byť v slove za druhý,
- vymenenie – vymenenie poradia dvoch susediacich znakov.

Náš algoritmus na pridávanie preklepov je založený na týchto štyroch druhoch chýb. Na to, aby simulácia preklepov v texte bola korektná, je potrebné určiť aj mieru s akou sa preklepy budú do množiny dát pridávať. Pre tieto účely sme použili a korpus webových diskusií (Hládek, a iní, 2014). Pri jednoduchej analýze sme sledovali, koľko tokenov z korpusu webových diskusií sa nachádza v morfologickej databáze a súčasne koľko ich Aspell (knižnica na kontrolu pravopisu) označí za správne. Korpus poskytuje anotáciu pre rôzne špeciálne typy slov, ako napríklad skratky, vlastné podstatné mená, stop slová, čísla, dátumy, mená firiem a podobne. Pomocou týchto značiek sme vyfiltrovali niektoré tokeny, ktoré môžu byť správne, ale v slovníkoch sa nenachádzajú (hlavne išlo o vlastné podstatné mená). Zo zostávajúcich tokenov sme potom vypočítali, že 17% z nich sa nenachádzalo ani v morfologickej databáze, ani v slovníku knižnice Aspell. Týchto 17% sme teda zobrali ako hraničnú pravdepodobnosť pridania chyby do nášho korpusu.

Algoritmus pracuje v nasledujúcich krokoch:

1. postupne sa iteruje cez všetky tokeny v množine dátn
2. pre každý token sa rozhodne, či sa doňho pridá chyba, alebo nie. Pravdepodobnosť pridania chyby je 17%,
3. náhodne sa vyberie jeden typ chyby z vyššie uvedených štyroch typoch chýb. Pravdepodobnosť určenia každej z nich je rovnaká – 25%,
4. ak je typ chyby vloženie alebo nahradenie, vyberie sa jeden zo znakov, ktorý susedí s daným znakom, pri ktorom sa má chyba vygenerovať (resp. ktorý sa má nahradiť). Znaky sa vyberajú zo štandardného QWERTZ slovenského rozloženia,

5. pôvodné slovo sa v množine dát nahradí chybným.

Samozrejme pri takomto náhodnom pridávaní preklepov sa môže stať, že sa náhodou vygeneruje existujúce slovo (teda rw chyba). Aby sme zistili koľko chybných vygenerovaných slov sú skutočne gramaticky správne slová, vykonali sme tri experimenty, pri ktorých sme spomedzi vygenerovaných chýb spočítali slová, ktoré boli gramaticky správne. Na rozhodnutie či je dané slovo správne sme použili morfologickú databázu a Aspell knižnicu na korekciu chýb. Vzhľadom na istú náhodnosť generovania chýb sme vykonali 10 takýchto meraní a pre finálne percentá sme použili priemer z nameraných hodnôt. Zhrnutie výsledkov:

- pri použití iba morfologickej databázy bolo 13% slov spomedzi vygenerovaných chybných slov detegovaných ako správne (čo tvorí 2% z celej množiny dát),
- pri použití Aspell knižnice bolo 9% slov spomedzi vygenerovaných chybných slov detegovaných ako správne (čo tvorí 1,6% z celej množiny dát),
- pri použití obidvoch nástrojov bolo 15% slov spomedzi vygenerovaných chybných slov detegovaných ako správne (čo tvorí 2,65% z celej množiny dát).

Absolútne čísla z týchto meraní uvádzame v tabuľke 8. Celkový počet slov v ktorých sa náhodne generovali chyby bol 86 282.

Tabuľka 8 - počet slov, ktoré v ktorých bol vygenerovaný preklep, ale vzniklo tak správne slovo

Morfologická databáza		Aspell		Morfologická anotácia + Aspell	
Počet chybných slov	Počet rw chýb	Počet chybných slov	Počet rw chýb	Počet chybných slov	Počet rw chýb
15 320	2013	15 112	1 420	15 078	2 272
15 065	1919	15 174	1 331	15 130	2 281
15 355	1969	15 142	1 352	15 188	2 343
15 109	1912	15 227	1 404	15 075	2 216
15 218	1979	15 166	1 424	15 332	2 361
15 126	1990	15 126	1 353	15 193	2 324
15 132	2023	15 400	1 398	15 378	2 308
15 066	2009	15 163	1 446	15 202	2 301
15 209	1979	15 170	1 384	15 247	2 288
15 146	2012	15 417	1 381	15 154	2 217

Nakoľko počet rw chýb vzhľadom na celý korpus nie je až taký veľký, rozhodli sme sa na tieto náhodne vygenerované gramaticky správne slová nezameriavať.

6.2.1 Úspešnosť parsovania po pridaní preklepov

Vyššie opísané generovanie chýb sme následne aplikovali na testovaciu vzorku dát. Po pridaní preklepov sme následne pomocou TreeTaggera pridali morfologickú anotáciu a takto upravené vstupné dáta sme sparsovali. Úspešnosť na týchto dátach uvádzame v tabuľke 9. Podobne ako v prípade merania počtu vzniknutých real-word chýb sme vykonali meraní 10 a ako výsledok použili spriemerovanú hodnotu.

Tabuľka 9 - úspešnosť parsovania po pridaní preklepov

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	61,14%	8,84%	82,15%	35,92%
Úspešnosť po pridaní preklepov	53,77%	6,66%	73,07%	23,99%

Opäť môžeme bádať prepad úspešnosti, tentoraz však o niečo menší ako v prípade odstránenej diakritiky. LAS sa prepadol o približne 7% a UAS o približne o 9%.

6.3 Gramatické chyby

V prípade preklepov ide o chyby vznikajúce nepozornosťou alebo neskúsenosťou pri písaní na klávesnici. Iný druh chýb, ktoré sa v textoch generovaných bežnými používateľmi často vyskytujú sú rôzne gramatické chyby. Vznikajú v dôsledku nevedomosti, resp. nedostatočnej vedomosti o gramatike slovenského jazyka. Identifikovali sme niekoľko takýchto základných chýb:

- zamenenie mäkkého i (jota) a tvrdého y (ypsilon),
- spodobovanie (znelostná asimilácia),
- pridanie alebo vynechanie diakritického znamienka.

Algoritmus pridávania gramatických chýb pracuje v nasledujúcich krokoch:

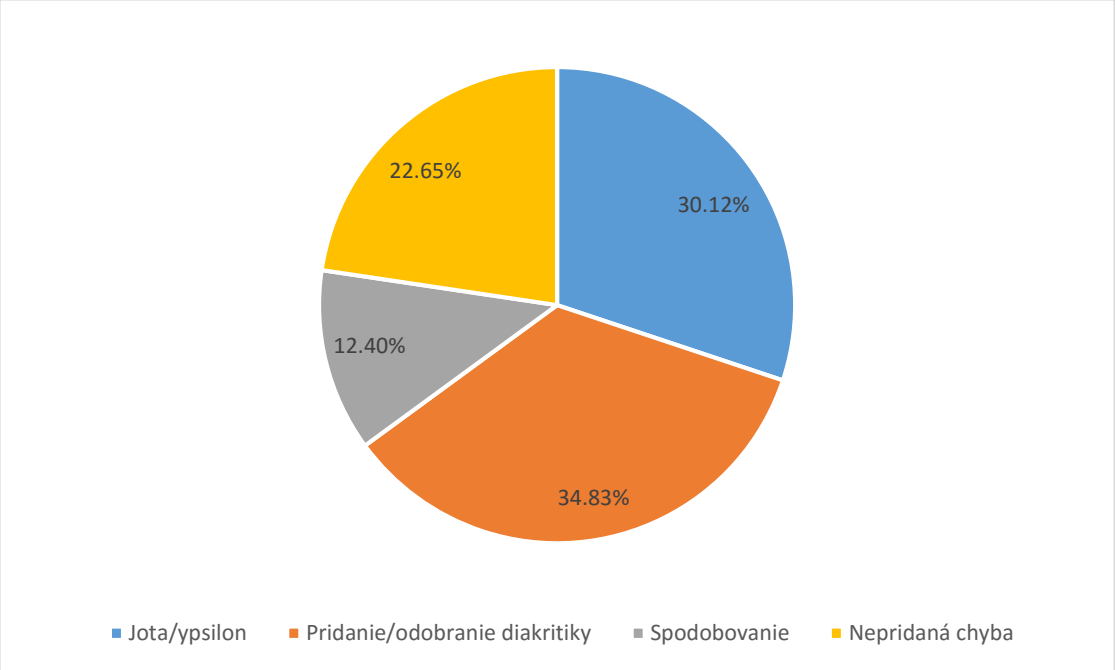
1. postupne sa iteruje cez všetky tokeny vo vstupnej množine dát,
2. s pravdepodobnosťou 17% sa určí, či sa v slove vygeneruje chyba alebo nie,
3. slovo sa zanalyzuje a zistí sa, ktoré z uvedených gramatických chýb je možné v ňom vygenerovať,
4. všetkým možným chybám sa vygeneruje rovnaká pravdepodobnosť a náhode sa jedna vyberie,
5. pridá sa chyba a výsledné slovo sa použije namiesto pôvodnej množine dát.

V prípade zámeny joty a ypsilonu a pridania alebo vynechania diakritického znamienka sa sleduje či je v slove jota alebo ypsilon, prípadne písmeno ktorému je možné pridať

alebo odobrať diakritické znamienko. Spodobovanie má v slovenčine pomerne jasne dané pravidlá, podľa ktorých sme generovanie tejto chyby implementovali:

- zmena znelej spoluhlásky na neznelú:
  - ak sa znelá spoluhláska nachádza na konci slova (napríklad *hrad - hrat, kebab - kebap*),
  - ak sa znelá spoluhláska nachádza pred neznelou spoluhláskou (napríklad *sudca - sutca*),
- zmena neznelej spoluhlásky na znelú:
  - ak sa nachádza vo vnútri slova pred znelou spoluhláskou (napríklad *kresba – krezba*).

Po pridaní chýb sme sa pozreli aj bližšie na to, v akom rozložení jednotlivé typy gramatických chýb sú, nakoľko nie pre každé slovo ktoré je náhodne vybrané na vygenerovanie chyby je možné vygenerovať všetky typy gramatických chýb. Je dokonca možné, že sa vyberie slovo v ktorom nie je možné vygenerovať žiadna z uvedených typov chýb. Prehľad uvádzame v obrázku 13.



Obrázok 13 - rozloženie kategórií pridaných gramatických chýb

Z diagramu vidíme, že v až 22,65% prípadov sa nepodarilo vygenerovať žiadnu gramatickú chybu. Vzhľadom na tento fakt sme upravili pravdepodobnosť generovania chýb tak, aby počet pridaných gramatických chýb zhruba odpovedal počtu pridaných preklepov.

6.3.1 Úspešnosť parsovania po pridaní preklepov

Pomocou implementovaného algoritmu na pridávanie chýb sme následne do testovacej množine dát pridali chyby. Do vzniknutého chybového textu sme následne pridali morfologickú anotáciu pomocou TreeTaggera. Úspešnosti parsovania takéhoto vstupu uvádzame v tabuľke 10. Vzhľadom na to, že ide opäť o do istej miery náhodné pridávanie chýb, urobili sme 10 meraní a uvádzané hodnoty v tabuľke 10 sú ich priemery.

Tabuľka 10 – úspešnosť parsovania po pridaní preklepov

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	61,14%	8,84%	82,15%	35,92%
Úspešnosť po pridaní gramatických chýb	54,33%	7,00%	73,96%	25,84%

Úspešnosť v tomto prípade klesla o niečo menej ako v prípade náhodne pridaných preklepov. LAS sa tentoraz prepadol o necelých 7% a UAS o 8%.

7 Rekonštrukcia textu pre syntaktickú analýzu

V predchádzajúcej kapitole sme sa zamerali na simuláciu chýb v texte, pričom sme sa snažili napodobniť webový text. Experimenty preukázali značne nižšiu úspešnosť parsovania pri texte, ktorý obsahoval rôzne deformácie. V tejto kapitole sa zameriavame na metódy, pomocou ktorých by bolo možné text opraviť a úspešnosť čo najviac priblížiť pôvodnej úrovni, dosiahnutej pri pôvodnom vstupe s našou vlastnou morfologickou anotáciou.

7.1 Rekonštrukcia diakritiky

Ako už bolo spomenuté chýbajúca diakritika je na webe pomerne bežná záležitosť, ktorá sa navyše dosť negatívne prejavuje na úspešnosti parsovania. S jej absenciou sa ale vieme celkom dobre vysporiadať. Je dostupných hneď niekoľko webových služieb na automatickú rekonštrukciu diakritiky v slovenskom texte:

- DIAKRITIK[9] - rekonštrukcia diakritiky vyvinutá na pôde Jazykovedného Ústavu Ľudovíta Štúra (ďalej JUĽŠ),
- KEMT NLP korekcia textu[10] - okrem rekonštrukcie diakritiky ponúka aj celkovú korekciu textu,
- Dopĺňač diakritiky[11], ktorý vznikol v rámci projektu bedrooms.sk.

Podľa (Gedera, 2015) sa ale ako najsľubnejšia služba javí tá, ktorú poskytuje JUĽŠ a dosahuje úspešnosť až 99.8%. Všeobecná korekcia textu KEMT NLP podľa (Hládek, a iní, 2013) dosahuje úspešnosť 72,6%. Treba ale podotknúť, že v čase dokončovania práce už nebola verejne dostupná. v K Dopĺňaču diakritiky sa nám nepodarilo získať žiadne štatistiky úspešnosti, nechávame ho v zozname ako referenciu na existujúci projekt súvisiaci s danou problematikou.

S nástrojom Diakritik sme následne vykonali experiment, kde sme chceli zistiť, či sa podarí deformovaný vstup zrekonštruovať natoľko, že sa úspešnosť priblíži k pôvodnej úspešnosti. Zobrali sme testovaciu množinu Slovenského Závislostného Korpusu a rovnako ako v predošlom experimente sme text zbavili diakritiky. Následne sme pomocou webovej služby zrekonštruovali diakritku a pomocou nástroja TreeTagger pridali morfologickú anotáciu. Pokrytie taggera ostalo na 100%, ale jeho presnosť a úplnosť sa dostala na úroveň 78,80%. Následne sme dáta sparsovali pomocou MATE Tools grafového parsera. Úspešnosti

[9] http://korpus.sk/diakritik.html
[10] http://nlp.web.tuke.sk/nlpform
[11] http://diakritika.brm.sk/

ktoré parser dosiahol aj s porovnaním s predchádzajúcimi úspešnosťami uvádzame v tabuľke 11.

Tabuľka 11 – úspešnosť parsera po rekonštrukcií diakritiky

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	61,14%	8,84%	82,15%	35,92%
Úspešnosť s rekonštruovanou diakritikou a s morf. anotáciou z TreeTaggera	61,55%	8,54%	81.53%	34,94%

Vidíme, že úspešnosť sa podarilo vrátiť na pôvodnú úroveň úspešnosti, čo je ale zaujímavé, tak v prípade LAS metriky sa nám ju podarilo dokonca mierne prekonať. Jedno z možných vysvetlení je, že pôvodné dáta obsahujú buď v texte alebo v morfologickej anotácií istú mieru chýb, pričom ale syntaktická anotácia ostala správna.

7.2 Kontrola pravopisu

V kapitole 6 sme predstavili metódy na pridávanie gramatických chyb a preklepov do textu. To sú chyby v texte, ktoré sa bežne v mnohých aplikáciách (a hlavne vo webových prehliadačoch a textových editoroch) používa kontrola pravopisu (z angl. spellchecking). V tejto kapitole sme sa rozhodli preskúmať, či je možné tieto nástroje na kontrolu pravopisu použiť na celkovú korekciu textu, pomocou ktorej by sa nám podarilo vylepšiť vstup a následne tak aj zlepšiť úspešnosť parsovania. Väčšina existujúcich nástrojov sú rôzne deriváty pôvodného unixového programu s názvom spell (Pirinen, a iní, 2010), (McIlroy, 1982). Pre naše experimenty sme vybrali dva:

- GNU Aspell[12] je voľne dostupný nástroj na kontrolu pravopisu. Bol vyvinutý na nahradenie nástroja Ispell, ktorý mal svojho času problémy so spracovaním textu v UTF-8 kódovaní,
- Hunspell[13] je voľne dostupný nástroj na kontrolu pravopisu, ktorý je založený na nástroji Myspell. Bol pôvodne vyvinutý pre maďarčinu, ale vďaka ľahkej rozšíriteľnosti sú dostupné slovníky pre mnohé iné jazyky, vrátane slovenčiny. Aj vďaka tomu sa Hunspell postupom času stáva jeden z najpoužívanejších nástrojov na kontrolu pravopisu a predvoleným nástrojom na kontrolu pravopisu v linuxových a unixových systémoch.

[12] http://aspell.net/index.html#dir
[13] http://hunspell.github.io/

V našich experimentoch sme sa rozhodli tieto dva nástroje na kontrolu pravopisu použiť pri snahe opraviť preklepy a gramatické chyby, ktoré sme do množiny dát pridali. Postup experimentu bol nasledovný:

1. pridanie chýb do množiny dát,
2. opravenie textu pomocou nástroja na korekciu textu – ak sa slovo nenachádzalo v slovníku, vygenerovali sa preňho návrhy. Zoznam návrhov bol utriedený podľa váhy, slovo s najvyššou váhou sme použili a nahradili ním chybné slovo v texte,
3. morfologická anotácia textu pomocou TreeTaggera,
4. parsovanie.

Experiment sme vykonali osobitne pre preklepy a gramatické chyby a osobitne pre Aspell a Hunspell. Úspešnosti parsovania uvádzame v tabuľke 12, pričom všetky riadky uvažujú použitie TreeTaggera.

Tabuľka 12 - Úspešnosti parsovania po použití nástrojov na kontrolu pravopisu

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	61,14%	8,84%	82,15%	35,92%
Úspešnosť s pridanými preklepmi a kontrolou nástroja Aspell	57,06%	7,14%	76,88%	27,77%
Úspešnosť s pridanými preklepmi a kontrolou nástroja Hunspell	57,17%	7,18%	77,05%	28,05%
Úspešnosť s pridanými gramatickými chybami a kontrolou nástroja Aspell	57,03%	7,16%	77,03%	29.08%
Úspešnosť s pridanými gramatickými chybami a kontrolou nástroja Hunspell	56,98%	7,14%	76,77%	28,63%

Vidíme, že použitie nástrojov na kontrolu pravopisu sa na úspešnosti podpísali pozitívne, aj keď nie až do takej miery ako v prípade rekonštrukcie gramatiky. Úspešnosť LAS a UAS sa podarilo dvihnúť priemerne o tri percentá v prípade opravovania preklepov a priemerne o dve percentá pri opravovaní gramatických chýb. Zaujímavé je, že aj pre preklepy, aj pre gramatické chyby sa úspešnosť po aplikovanú kontroly pravopisu dostala na približne rovnakú úroveň. Tiež vidíme, že rozdiel v jednotlivých nástrojoch na kontrolu pravopisu (Aspell, Hunspell) je len minimálny, aj keď počas experimentovania bolo vidieť, že Aspell vykonáva gramatickú kontrolu omnoho rýchlejšie.

Pri sledovaní priebehu experimentov sme si všimli, že nástroje na kontrolu pravopisu niekedy označia za zlé aj slovo, ktoré zlé nie je (teda pravdepodobne ho nemajú v slovníku). Vykonali sme ešte jedno meranie, kde sme každé slovo hľadali ešte aj v morfologickej databáze – ak sa tam slovo nachádzalo, kontrolu pravopisu sme preskočili. Cieľ tohto experimentu bol zistiť, či takto vieme znížiť počet falošných negatív (slovo je správne, ale nástroj ho označil ani nesprávne) pri kontrole pravopisu. Výsledky uvádzame v tabuľke 13.

Tabuľka 13 - úspešnosť parsovania s pridanou kontrolou pravopisu a kontrolou s v morfologickej databáze

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	61,14%	8,84%	82,15%	35,92%
Úspešnosť s pridanými preklepmi a kontrolou nástroja Aspell	56,92%	7,08%	76,73%	27,59%
Úspešnosť s pridanými preklepmi a kontrolou nástroja Hunspell	57,03%	7,18%	76,88%	27,81%
Úspešnosť s pridanými gramatickými chybami a kontrolou nástroja Aspell	56,97%	7,22%	76,86%	28.40%
Úspešnosť s pridanými gramatickými chybami a kontrolou nástroja Hunspell	56,93%	7,12%	76,78%	28,59%

Vidíme ale, že úspešnosť parsovania nestúpla, práve naopak, mierne klesla takmer vo všetkých metrikách v každom meraní. Vysvetlenie tohto javu je jednoduché. Morfologická databáza odchytila viac slov, pričom niektoré z nich ale potrebovali úpravu od nástroja na kontrolu pravopisu. Počet falošných negatív sme teda neznížili, naopak sme práve zvýšili počet falošných pozitív (slovo bolo označené ako správne, aj keď nie je správne).

Záver týchto experimentov je, že s jednoduchou pomocou existujúcich nástrojov na rekonštrukciu textu vieme viditeľne zlepšiť úspešnosť parsovania. Veríme, že v prípade parsovania reálnych textov z webu má táto technika zmysel.

8 Zlepšovanie určitých typov vzťahov

Výstup syntaktickej analýzy ponúka viac druhov informácií. Jedna z nich je závislostný strom, z ktorého vieme zistiť ktoré slovo na ktorom závisí, ako jednotlivé časti vety so sebou súvisia a ktoré slovné spojenia tvoria celky. Druhý typ informácie tvoria označenia konkrétnych vzťahov – ktoré slovo tvorí prísudok, ktoré podmet, ktoré slovo rozvíja objekt ako prívlastok a podobne. Rôzne aplikácie majú od syntaktickej analýzy rôzne požiadavky. Niektorým stačí vedieť, ktoré slovo s ktorým súvisí, pričom ich typ vzťahu, alebo rola slova vo vete nezaujíma. Typickým príkladom môže byť extrakcia kolokácií, keď potrebujeme zistiť ktoré slová danú kolokáciu tvoria (čo v prípade jazyka s voľným slovosledom nemusí byť také jednoduché). Iné aplikácie zas práve potrebujú zistiť roly slov vo vete, aby s nimi vedeli ďalej pracovať. Jeden z príkladov môže byť extrakcia entít, alebo tvorcov deja alebo samotný dej vety. Aj z tohto dôvodu sa pri vyhodnocovaní syntaktických parserov uvádzajú dve metriky – jedna ktorá typ vzťahu (analytickú funkciu) nezahŕňa a druhá ktorá ano.

Pri pohľade na podrobnejšie štatistiky parsovania (tabuľka 3, kapitola 5.4) vidíme, že najspoľahlivejšie parser určuje analytické funkcie označujúce interpunkciu označujúcu koniec vety, pomocné sloveso byť, zvratné zámeno sa a podobne. Analytické funkcie označujúce vetné členy, ktoré v slovenčine tvoria jadro vety sú v poradí o dosť nižšie, pričom napríklad funkcia *Sb*, ktorá označuje podmet je v poradí určovania úspešnosti až ôsma, s percentuálnou úspešnosťou 73,04%. Analytická funkcia *Pred* označujúca predikát (prísudok) je na tom ešte horšie: v poradí určovania úspešnosti až dvanásta s percentuálnou úspešnosťou 59,06%. To sú dosť nízke čísla na to, že tieto dve analytické funkcie tvoria jadro takmer každej vety.

Vzhľadom na nepomer medzi dôležitosťou rolí niektorých analytických funkcií a ich úspešného určovania sme sa rozhodli skúmať možné spôsoby, ako úspešnosť určovania niektorých typov analytických funkcií zlepšiť. V tejto kapitole prezentujeme návrh a overenie metódy na úpravu vstupu pre parser tak, aby lepšie určoval určitý typ vzťahu.

8.1 Opis metódy na zlepšenie úspešnosti pre určité vzťahy

Základom pre našu metódu je hypotéza, že vhodnou úpravou množiny morfologických vlastností, vieme parser natrénovať tak, aby lepšie s vyššou presnosťou určoval jeden vybraný typ vzťahu. K morfologickým vlastnostiam pridáme ďalšiu, ktorá bude indikovať, že dané slovo je pravdepodobne nositeľ daného vzťahu. Následne už necháme na trénovacom proce-

se parsera, aby rozšírenú množinu morfologických vlastností využil pri tvorbe modelu pre ďalšie parsovanie.

Hlavný problém pri tejto metóde musíme vyriešiť je vytvorenie pravidla, na základe ktorého budeme do množiny morfologických vlastností pridávať značky označujúce pravdepodobnosť, že ide o danú analytickú funkciu. Označkovať všetky inštancie danej analytickej funkcie v trénovacích dátach nestačí, vzhľadom na to, že množinu morfologických vlastností budeme musieť rozšíriť aj pred samotným parsovaním a nie iba pri trénovaní.

Rozhodli sme sa preto využiť metódu dolovania asociačných pravidiel, pomocou ktorej by sme z vyčlenenej vzorky dát vydolovali pravidlá, na základe ktorých by sme mohli označovať kandidátov na daný typ analytickej funkcie. Postup získavania pravidiel bol nasledovný:

1. vytvorenie transakcií ako vstup pre algoritmus. V tomto kroku sme zobrali dáta z trénovacej množiny a z každého tokenu ktorý reprezentoval danú analytickú funkciu sme vytvorili transakciu – tá obsahovala každú morfologickú vlastnosť a danú analytickú funkciu,
2. dolovanie pravidiel. S použitím algoritmu FP-growth (Han, a iní, 2004) sme z transakcií získali asociačné pravidlá. Ukážku pravidiel vidíme na obrázku 14, pričom prvý prvok pravidla je záver (pravá strana implikácie, naša cieľová analytická funkcia), druhý prvok je množina morfologických vlastností, ktoré sa s danou analytickou funkciou vyskytujú (ľavá strana implikácie) a posledný prvok je váha (z angl. support) daného pravidla,
3. filtrovanie pravidiel. Vzhľadom na netriviálny počet získaných pravidiel sme ich museli vyfiltrovať. Automaticky sme odmietli všetky pravidlá s váhou pod istou prahovou hodnotou. Ďalej sme vyfiltrovali pravidlá, ktoré boli príliš všeobecné (na ľavej strane implikácie bol napríklad iba slovný druh) a pravidlá ktoré boli príliš špecifické (boli aplikovateľné len na veľmi málo inštancií).

```
('Pred', (), 3241),
('Pred', ('pos=V',), 3168),
('Pred', ('neg=a',), 2799),
('Pred', ('neg=a', 'pos=V'), 2799),
('Pred', ('num=s',), 2671),
('Pred', ('num=s', 'pos=V'), 2638),
('Pred', ('num=s', 'neg=a'), 2319),
('Pred', ('num=s', 'neg=a', 'pos=V'), 2319),
('Pred', ('per=c',), 2584),
('Pred', ('per=c', 'pos=V'), 2584),
('Pred', ('per=c', 'neg=a'), 2320),
('Pred', ('per=c', 'neg=a', 'pos=V'), 2320),
('Pred', ('per=c', 'num=s'), 2187),
('Pred', ('per=c', 'num=s', 'pos=V'), 2187),
('Pred', ('per=c', 'num=s', 'neg=a'), 1959),
('Pred', ('per=c', 'num=s', 'neg=a', 'pos=V'), 1959)
```

Obrázok 14 - ukážka vydolovaných asociačných pravidiel

Počet asociačných pravidiel bol aj po vyfiltrovaní dosť veľký (rádovo stovky), bol už ale zvládnuteľný pre manuálne posúdenie. Z vyfiltrovaných pravidiel sme následne vybrali niekoľko kandidátov, s ktorými sme potom ďalej pracovali.

Ďalší krok bolo aplikovanie pravidla na obohatenie trénovacej množiny dát. Pomocou jednoduché skriptu, ktorý implementoval vybrané asociačné pravidlo sme upravili trénovaciu množinu dát a spustili trénovanie parsera. Kvôli časovým nárokom sme ale netrénovali parser na celej množine, ale iba na vzorke 5000 viet. To nám umožnilo omnoho flexibilnejšie experimentovanie a skúšanie viacerých pravidiel. Menšia trénovacia vzorka je aj dôvod, prečo vo vyhodnotení tohto experimentu budeme používať inú základnú hodnotu úspešnosti, s ktorou potom budeme porovnávať úspešnosť našej metódy.

8.2 Aplikácia metódy na *Pred* a *Pnom* analytické funkcie

Pre overenie metódy sme si vybrali analytické funkcie *Pred* a *Pnom*, označujúce prísudok a mennú časť prísudku. Obidve sú dôležitá súčasť vety a obidve majú len priemernú úroveň úspešnosti určovania. Asociačné pravidlo, ktoré sme pre analytickú funkciu *Pred* je opísané v nasledujúcom vzorci, kde M je množina morfologických vlastností.

$$\text{"vform=L"} \in M \ \wedge \text{"per=c"} \in M \ \Rightarrow M = M \ \cup \text{"pred\_candidate"}$$

S použitím tohto pravidla sme upravili trénovacie dáta a spustili trénovanie parsera. Následne sme s rovnakým pravidlom upravili aj testovacie dáta, sparsovali ich a vyhodnotili úspešnosť. Výsledky môžeme vidieť v tabuľke 14.

Tabuľka 14 - Úspešnosť parsovania po obohatení množiny morfologických značiek o *pred_candidate*

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	57,32%	6,46%	77,68%	29,02%
Úspešnosť po aplikovaní asociačného pravidla pre Pred analytickú funkciu	59,82%	8,73%	77,68%	28,70%

Z tabuľky vidíme, že pridanie asociačného pravidla sa výrazne pozitívne prejavilo na metrike LAS aj tLAS. Čo je ale zaujímavé je mierny prepad metriky tUAS pri zachovaní rovnakej hodnoty UAS. To značí, že zatiaľ čo niekde chyby pri určovaní závislosti pribudli, niekde inde zase ubudli a to takým spôsobom, že sa zmenšil počet viet kde boli absolútne správne určené všetky závislosti.

Keď sa ešte pozrieme na úspešnosť určovania samotnej *Pred* analytickej funkcie, môžeme jasne vidieť zlepšenie. Parser s pôvodným modelom určil správne 54,83% inštancií predikátu, zatiaľ čo s modelom ktorý bol natrénovaný s rozšírenou množinou morfologických vlastností bola úspešnosť 76,74% (v absolútnych číslach je to 1 777 oproti 2 487 správne určených predikátov, z celkového počtu 3241).

Pre analytickú funkciu *Pnom* sme postupovali obdobne, akurát sme použili pravidlo opísané v nasledujúcom vzorci:

$$\text{"par=A"} \in M \ \wedge \text{"case=1"} \in M \ \Rightarrow M = M \ \cup \text{"pnom\_candidate"}$$

Kde opäť M značí množinu morfologických vlastností. Upravili sme aj trénovacie aj testovacie dáta a spustili trénovanie. Úspešnosti parsovania môžeme vidieť v tabuľke 15.

Tabuľka 15 - Úspešnosť parsovania po obohatení množiny morfologickej značky o *pnom_candidate*

	LAS	tLAS	UAS	tUAS
Základá úspešnosť	57,32%	6,46%	77,68%	29,02%
Úspešnosť po aplikovaní asociačného pravidla pre Pnom analytickú funkciu	57,36%	6,43%	77,68%	28,71%

Tu už zlepšenie nie je tak jendoznačne vidieť. Dokonca pozorujeme podobný fenomén ako v predchádzajúcom prípade pri metrike UAS a tUAS. Tu sa nám stala podobná vec pre metriku LAS a tLAS, pričom LAS bola v tomto prípade dokonca lepšia ako pôvodne. Tieto drobné rozdiely sú spôsobené aj menšou kvantitou výskytu *Pnom* analytickej funkcie v dátach.

Úspešnosti určovania samotného *Pnom* tagu sú ale pozitívnejšie. Z pôvodných 51,41% sa úspešnosť dostala na 61,98% (v absolútnych číslach je to 457 oproti 511 správne určených inštancií z celkového počtu 889).

8.3 Obmedzenia metódy obohacovania morfologickej značky

Výsledky uvedené v predchádzajúcej kapitole sú pozitívne a preukázali, že pri zameraní sa na špecifický typ vzťahu je možné vylepšiť úspešnosť určovania daného vzťahu. Pri obohacovaní množiny morfologickej značky ale treba byť opatrný, nakoľko príliš veľa pridaných vlastností môže mať skôr opačný efekt.

Presvedčili sme sa o tom pri niektorých ďalších experimentoch, ktoré sme s obohacovaním vykonali. Prvý spočíval v tom, že sme pre *Pnom* analytickú funkciu použili dve pravidlá z množiny asociačných pravidiel. Úspešnosť určovania samotnej *Pnom* analytickej funkcie bola lepšia, ostatné metriky sa ale naopak zhoršili, čo indikuje, že prílišným obohatením množiny morfologických značiek sme zvýšili počet pravých pozitív ale aj falošných pozitív pri určovaní *Pnom* funkcie.

Druhý experiment spočíval v tom že sme sa snažili zlepšiť určovanie *Pred* a *Pnom* funkcie, ak sa nachádzali v rovnakej vete. Vygenerovali sme transakcie, ktoré tentoraz pozostávali zo zjednotenia množín morfologických značiek dvoch slov a dvoch analytických funkcií. Po vybratí vhodného asociačného pravidla sme upravili dáta a natrénovali parser, ale úspešnosť vo všetkých metrikách klesla (aj keď nie nejako výrazne, v priemere o jednu desatinu percenta).

Je známe, že pri robustnejšej množine morfologických značiek narážame na problém riedkosti dát. Z tohto faktu a z experimentov vyplýva, že metóda na obohatenie množiny morfologickej značky je použiteľná, ale skôr pre špecifický prípad zlepšenia úspešnosti iba jedného vzťahu. Veríme ale, že z takéhoto špecifického zlepšenia výstupu parsovania môžu mať niektoré aplikácie prospech. Navyše je pomerne triviálne rozšíriť a natrénovať model aj pre iné analytické funkcie, podľa požiadaviek konkrétnej aplikácie.

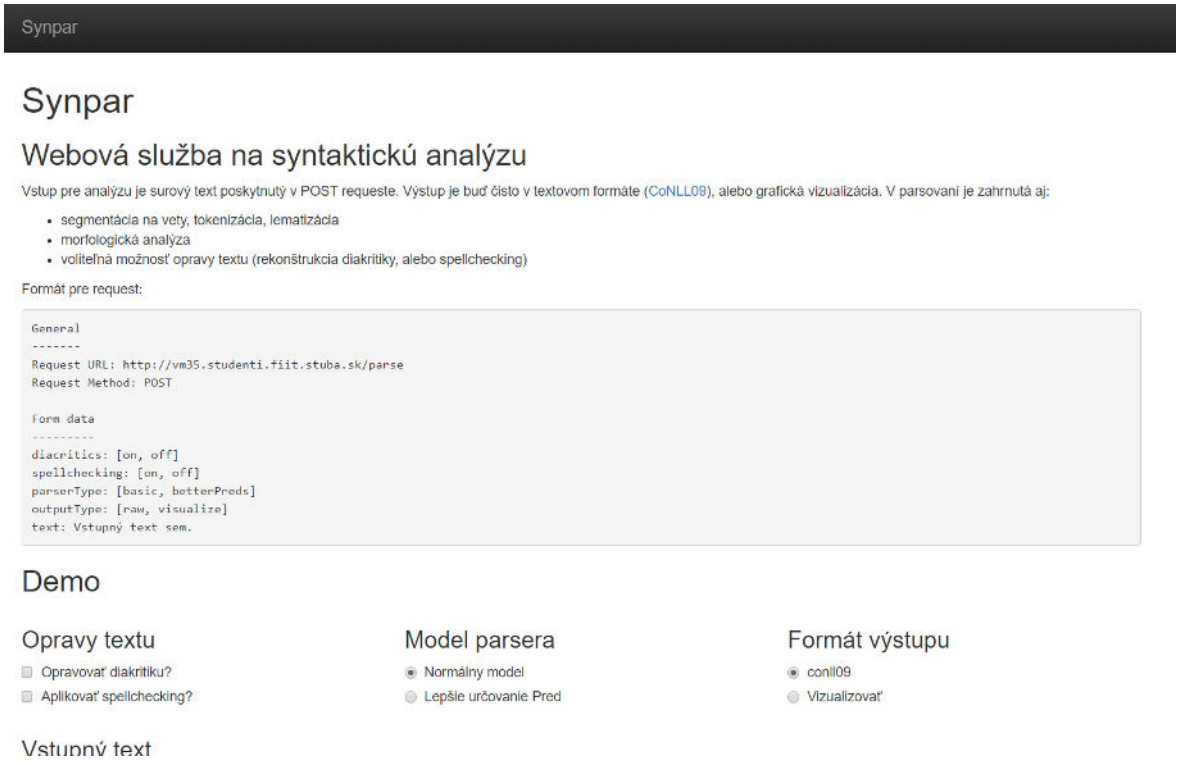
9 Realizácia parsera ako webovej služby

Webová služba je ideálna na vyskúšanie si funkcionality, overenie konceptu alebo zoznámenie sa s danou problematikou. V prípade, že je dobre naimplementovaná, môže slúžiť aj ako základ pre iné aplikácie a služby. Veľké spoločnosti dnes bežne uverejňujú aplikačné rozhrania, na ktorých potom zakladajú svoju funkcionalitu mnohé iné aplikácie. Uverejnenie parsera pre slovenčinu ako webovú službu významne uľahčí integráciu parsovania do aplikácie potenciálneho záujemca, nakoľko proces rozbehania parsera lokálne (hlavne príprava dát a trénovanie) nie je úplne triviálny. Veríme, že ľahší prístup k tejto funkcionalite zvýši na Slovensku záujem o parsovanie aj ako o nástroj aj ako o výskumnú oblasť.

Webová služba[14] (ukážka domovskej obrazovky je na obrázku 15) implementuje všetky vylepšenia, ktoré sme v rámci predchádzajúcich experimentov vytvorili. Zároveň integruje všetky potrebné kroky spracovania tak, aby ako vstup stačilo zadať surový text. Kroky v ktorých tento text webová služba spracováva sú následovné:

- segmentácia textu na vety,
- rekonštrukcia diakritiky (voliteľné) pomocou nástroja DIAKRITIK,
- kontrola pravopisu (voliteľné) pomocou nástroja Hunspell,
- tokenizácia,
- morfologická anotácia a lematizácia (pomocou nástroja TreeTagger),
- výber modelu pre parsovanie (lepšie určovanie *Pred* alebo *Pnom* tagov, alebo všeobecné parsovanie),
- parsovanie pomocou nástroja MATE Tools grafový parser.

[14] http://vm35.studenti.fiit.stuba.sk/



Obrázok 15 - domovská obrazovka webovej služby

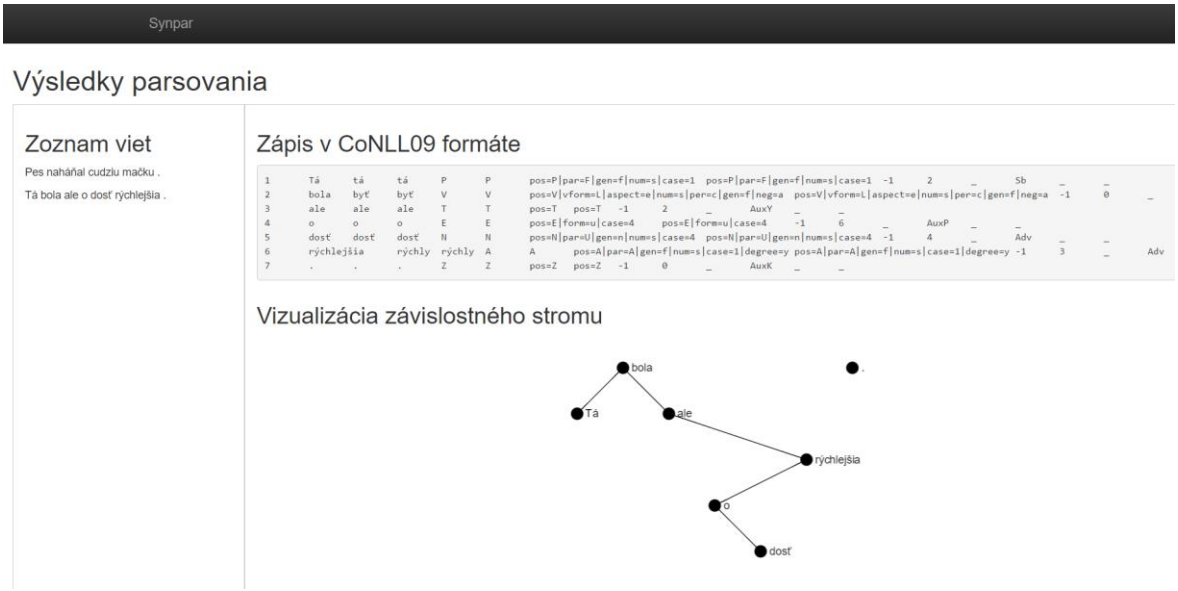
Webová služba poskytuje dva formáty výstupu. Jeden je čisto textový a ide vlastne o vety zapísané vo formáte CoNLL09 (príklad je na obrázku 16).

```
Pes     pes     pes     S       S       pos=S|par=S|gen=m|num=s|case=1  pos=S|par=S|gen=m|num=s|case=1  -1      2
naháňal naháňať naháňať V       V       pos=V|vform=L|aspect=e|num=s|per=c|gen=m|neg=a  pos=V|vform=L|aspect=e|num=s|
cudziu  cudzí   cudzí   A       A       pos=A|par=A|gen=f|num=s|case=4|degree=x pos=A|par=A|gen=f|num=s|case=4|degree
mačku   mačka   mačka   S       S       pos=S|par=S|gen=f|num=s|case=4  pos=S|par=S|gen=f|num=s|case=4  -1      2
.       .       .       Z       Z       pos=Z   pos=Z   -1      0       _       AuxK    _       _

Tá      tá      tá      P       P       pos=P|par=F|gen=f|num=s|case=1  pos=P|par=F|gen=f|num=s|case=1  -1      2
bola    byť     byť     V       V       pos=V|vform=L|aspect=e|num=s|per=c|gen=f|neg=a  pos=V|vform=L|aspect=e|num=s|
ale     ale     ale     T       T       pos=T   pos=T   -1      2       _       AuxY    _       _
o       o       o       E       E       pos=E|form=u|case=4     pos=E|form=u|case=4     -1      6       _       AuxP
dosť    dosť    dosť    N       N       pos=N|par=U|gen=n|num=s|case=4  pos=N|par=U|gen=n|num=s|case=4  -1      4
rýchlejšia      rýchly  rýchly  A       A       pos=A|par=A|gen=f|num=s|case=1|degree=y pos=A|par=A|gen=f|num=s|case=
.       .       .       Z       Z       pos=Z   pos=Z   -1      0       _       AuxK    _       _
```

Obrázok 16 - textový výstup webovej služby

CoNLL09 textový formát poskytuje všetky informácie ktoré sa v procese parsovania získali: okrem jednotlivých závislostí a analytických funkcií pre všetky tokeny sú vo výstupe aj lemy tokenov, ich slovný druh a kompletná morfologická anotácia. Jednotlivé vety sú potom oddelené prázdnym riadkom. Druhý formát výstupu je grafická vizualizácia, znázornená na obrázku 17.



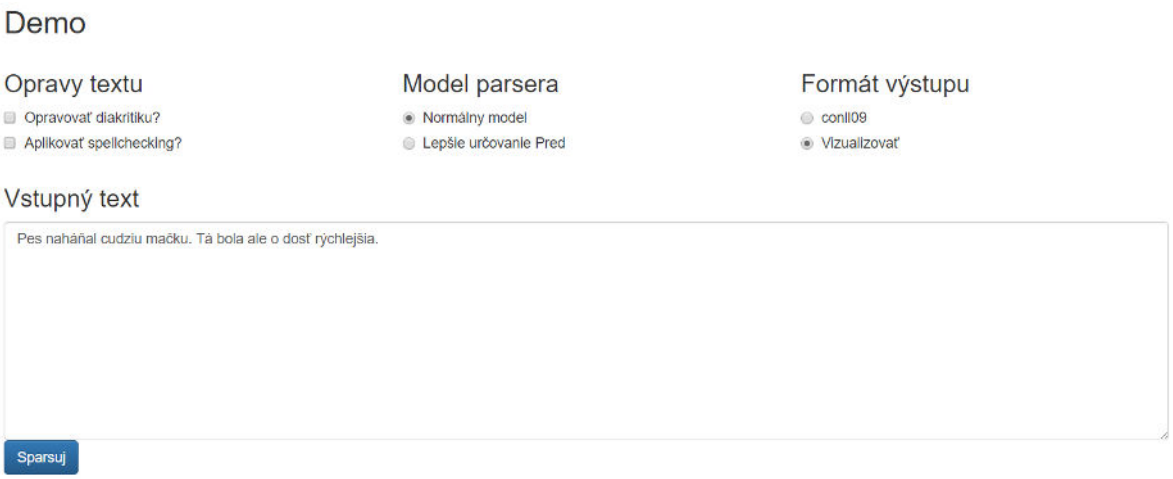
Obrázok 17 - ukážka vizualizácie vety ako jedného z formátov výstupu webovej služby

Vizualizácia parsovania je vlastne malá Javascriptová aplikácia, ktorá z CoNLL09 textového formátu vykreslí grafovú štruktúru, pričom samotný graf reprezentujúci závislostný strom je vykreslený pomocou knižnce sigma.js[15]. Na ľavej strane obrazovky je k dispozícií zoznam viet ako boli z textu vyextrahované. Po kliknutí na vetu zo zoznamu sa načíta celý pôvodný CoNLL09 do textového poľa v hlavnej časti obrazovky a pod ním sa vykreslí úž spomínaná stromová závislostná štruktúra.

Zo službou je možné komunikovať prostredníctvom POST požiadaviek na server. V tele požiadavky treba uviesť parametre podľa ktorých sa zapína a vypína funkcionalita rekonštrukcie textu a presnejšieho určovania *Pred* a *Pnom* tagov. Okrem týchto parametrov treba samozrejme uviesť aj text, ktorý sa má podľa zadaných parametrov sparsovať. Presný popis jednotlivých parametrov aj s ich možnými hodnotami je uvedený v úvodnom texte webovej služby.

Rozhranie webovej služby ponúka aj online demo ukážku, kde si používateľ môže prostredníctvom jednoduchého formulára nastaviť parametre webovej služby, text na parsovanie a formát výstupu (ukážka na obrázku 18).

[15] http://sigmajs.org/



Obrázok 18 - online demo ukážka syntaktického parsovania

Táto demo ukážka slúži hlavne na rýchle experimentovanie s parametrami webovej služby a vybratie tej najvhodnejšej.

10 Zhodnotenie

V tejto práci sme poskytli základný prehľad problematiky syntaktickej analýzy, a to aj z pohľadu jazykovedného, aj z pohľadu výpočtovej lingvistiky. Predstavili sme základné pojmy potrebné na pochopenie a orientáciu v tejto oblasti a predstavili sme základné metódy a prístupy pre syntaktickú analýzu, pričom sme sa zamerali aj na reprezentácie závislostí, neprojektívnosť viet a z toho vyplývajúce poznatky dôležité pre parsovanie slovenčiny. Okrem toho sme sa snažili poukázať aj na problémy týkajúce sa parsovania reálneho, nespracovaného textu a parsovanie webového textu. Tým sme poukázali aj na rozdiely medzi vyhodnocovaním parserov na zlatej vzorke a použitím v praxi, kde sme vystavení rôznym špecifikám, ako napríklad:

- chýbajúca morfologická anotácia, ktorá je v prípade slovenčiny viac zložitejšia a tým pádom aj viac náchylnejšia k chybám pri automatickej anotácií,
- v prípade parsovania webových textov sú to rôzne deformácie textu, ako napríklad chýbajúca diakritika, preklepy a gramatické chyby v texte,
- iná doména textov, než na akej bol parser natrénovaný.

Toto sú všetko problémy na ktorým je parser použitý v praxi vystavený a nie sú ešte úplne dobre vyriešené.

Okrem samotnej syntaktickej analýzy sme sa zamerali aj na morfologickú analýzu, nakoľko tvorí nevyhnutný vstup pre syntaktickú analýzu a úzko s ňou súvisí. Ponúkli sme prehľad existujúcich nástrojov na morfologickú analýzu, ktoré pracujú aj so slovenčinou. Každý z nich sme vyhodnotili na dátach zo Slovenského Závislostného Korpusu. Najlepšie výsledky sme dosiahli s nástrojom TreeTagger, ktorý sme potom používali aj v experimentoch.

Ďalej sme ponúkli prehľad významných svetových parserov, pričom pri ich výbere sme dbali na úspešnosť ktorú dosahujú a ich výskumnícku dôležitosť. Každý z týchto parserov sme natrénovali na Slovenskom Závislostnom Korpuse a vyhodnotili. Na základe toho sme vybrali najúspešnejší parser, ktorým sa stal grafový parser z balíka nástrojov Mate tools a podrobnejšie sme analyzovali jeho výstup. Na ňom sme následne založili ďalšiu prácu, čo bolo experimentovanie so vstupom.

V experimentoch sme skúmali ako sa rôzne deformácie textu prejavia na úspešnosti parsovania a či vieme pomocou existujúcich nástrojov prípadný prepad úspešnosti vykompenzovať. Pri úprave vstupu sme sa snažili čo najvernejšie napodobniť webový text, pričom za týmto účelom sme analyzovali korpus webových diskusií. Následne sme s použitím ná-

strojov na rekonštrukciu diakritiky a kontrolu pravopisu text opravovali a snažili sa tak dostať úspešnosť na pôvodnú úroveň. V prípade rekonštrukcie diakritiky sa to podarilo, v prípade kontroly pravopisu je ešte stále priestor na zlepšovanie, aj keď bolo dosiahnuté značné zlepšenie.

Druhá séria experimentov bola zameraná na obohatenie množiny morfologických značiek tak, aby znovunatrénovaný parser lepšie určoval niektoré vybrané typy vzťahov. Pomocou algoritmu na dolovanie asociačných pravidiel sa podarilo vylepšiť určovanie predikátu a nominálnej časti predikátu, z čoho môžu ťažiť niektoré špecifické aplikácie.

Z vykonaných experimentov a meraní sme získali nasledujúce poznatky a prínosy:

- prvé komparatívne vyhodnotenie úspešnosti viacerých svetových parserov na slovenčine,
- vyčerpávajúci zoznam rôznych nástrojov analýzu potrebných pre syntaktickú analýzu, aj s overením úspešnosti jednotlivých nástrojov,
- simulácia webového textu a skúmanie jeho chybovosti na úspešnosť parsovania, aj s návrhmi použitia existujúcich nástrojov na následné vylepšenie pokazenej úspešnosti,
- realizácia parsera ako verejne dostupnej webovej služby so všetkými experimentálne overenými vylepšeniami.

Ďalší výskum v tejto oblasti má mnoho možných smerovaní. Stav oblasti vo svete je oproti stavu oblasti u nás výrazne popredu, aj čo sa týka samotných metód parsovania a záujmu vedeckej obce, aj čo sa týka rozsahu a kvality dostupných dát, na ktorých by bolo možné štatistické parsery trénovať. Veríme že aj táto práca prispeje k všeobecnému záujmu o túto tému a bude užitočná pre ďalších výskumníkov v tejto oblasti.

Citované diela

BANKO, Michele, et al. 2007. Open information extraction for the web. In: *International Joint Conference on Artificial Intelligence*. p. 2670-2676.

BOHNET, Bernd; NIVRE, Joakim. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics*. p. 1455-1465.

BOHNET, Bernd, et al. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics.* 1: 415-428.

BOHNET, Bernd. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In: *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics. p. 89-97.

BUCHHOLZ, Sabine; MARSI, Erwin. 2006. CoNLL-X shared task on multilingual dependency parsing. In: *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics. p. 149-164.

CARRERAS, Xavier; MÀRQUEZ, Lluís. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In: *Proceedings of the Ninth Conference on Computational Natural Language Learning*. Association for Computational Linguistics. p. 152-164.

CARRERAS, Xavier. 2007. Experiments with a Higher-Order Projective Dependency Parser. In: *EMNLP-CoNLL.* Association for Computational Linguistics. p. 957-961.

ČERVENOVÁ, Dominika. 2015. Automatická syntaktická analýza textu v prirodzenom jazyku. Bratislava: FIIT STU, 2015, 68 s. Diplomová práca.

CHARNIAK, Eugene. 1997. Statistical techniques for natural language parsing. *AI magazine.* 18.4: 33.

CHU, Yoeng-Jin; LIU, Tseng-Hong. 1965. On shortest arborescence of a directed graph. *Scientia Sinica*. 14.10: 1396-1400.

COHEN, Shay B.; SMITH, Noah A. 2007. Joint morphological and syntactic disambiguation. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Lan-*

guage Processing and Computational Natural Language Learning. Association for Computational Linguistics. p. 208–217.

COLLINS, Michael, et al. 1999. A statistical parser for Czech. In: *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*. Association for Computational Linguistics. p. 505-512.

CRAMMER, Koby; SINGER, Yoram. 2003. Ultraconservative online algorithms for multiclass problems. In: *The Journal of Machine Learning Research*. MIT Press. 3: 951-991.

CRAMMER, Koby, et al. 2006. Online passive-aggressive algorithms. In: *The Journal of Machine Learning Research*. Microtome Publishing. 7: 551-585.

DAMERAU, Fred J. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*. 7.3: 171-176.

DE MARNEFFE, Marie-Catherine, et al. 2006. Generating typed dependency parses from phrase structure parses. In: *Proceedings of Language Resources and Evaluation Conference*. p. 449-454.

EDMONDS, Jack. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*. 71.4: 233-240.

FELICE, Mariano; YUAN, Zheng. 2014. Generating artificial errors for grammatical error correction. In: *European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. p. 116-126.

FOSTER, Jennifer; ANDERSEN, Øistein E. 2009. GenERRate: generating errors for use in grammatical error detection. In: *Proceedings of the fourth workshop on innovative use of nlp for building educational applications*. Association for Computational Linguistics. p. 82-90.

FURCY, David; KOENIG, Sven. 2005. Limited discrepancy beam search. In: *International Joint Conference on Artificial Intelligence*. p. 125-131.

GAJDOŠOVÁ, Mária Šimková–Katarína. 2008. Slovenský závislostný korpus. In: *Grammar & Corpora*, Academia. p. 135-141

GEDERA, Jakub. 2015. Automatic Diacritics Reconstruction in Slovak Texts. In: *IIT.SRC: 2015: Student Research Conerence in Informatics and Information Technologies*. Slovak University of Technology in Bratislava. p. 9-13

GREEN, Nathan. 2011. Dependency parsing. In: *WDS 2011 Proceedings of Contributed Papers*. p. 137-142.

HAJIČ, Jan. 2005. Complex corpus annotation: The Prague dependency treebank. *Insight into Slovak and Czech Corpus Linguistics*. Veda Bratislava. p. 54-73.

HAN, Jiawei, et al. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*. 8.1: 53-87.

HLÁDEK, Daniel; STAŠ, Ján; JUHÁR, Jozef. 2012. Dagger: The Slovak morphological classifier. In: *ELMAR, 2012 Proceedings*. IEEE. p. 195-198.

HLÁDEK, Daniel; STAŠ, Ján; JUHÁR, Jozef. 2014. Slovak Web Discussion Corpus. In: *Advances in Natural Language Processing*. Springer International Publishing. p. 463-469.

HLÁDEK, Daniel; STAS, Jan; JUHAR, Jozef. 2013. Unsupervised spelling correction for Slovak. *Advances in Electrical and Electronic Engineering*. 11.5: 392.

KAČALA, Ján.: Syntaktický systém jazyka. Formát, 1998.

LEE, John; NARADOWSKY, Jason; SMITH, David A. 2011. A discriminative model for joint morphological disambiguation and dependency parsing. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. p. 885-894.

MAXWELL, Dan. 2013. Why so many nodes?. *DepLing 2013*. Matfyzpress, p. 197-206.

MCDONALD, Ryan, et al. 2005. Non-projective dependency parsing using spanning tree algorithms. In: *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. p. 523-530.

MCDONALD, Ryan T.; PEREIRA, Fernando CN. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In: *European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistic. p. 84-88.

MCDONALD, Ryan; CRAMMER, Koby; PEREIRA, Fernando. 2005. Online large-margin training of dependency parsers. In: *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics. p. 91-98.

MCILROY, M. Douglas. 1982. Development of a spelling list. *IEEE Transactions on Communications*. 30.1: 91-99.

MÉSZÁROS, Dalibor. 2015. Determining the Parts of Speech. In: *IIT.SRC 2015: Student Research Conference in Informatics and Information Technologies*. Slovak University of Technology in Bratislava. p. 83-88

NILSSON, Jens; RIEDEL, Sebastian; YURET, Deniz. 2007. The CoNLL 2007 shared task on dependency parsing. In: *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*. p. 915-932.

NIVRE, Joakim. 2009. Non-projective dependency parsing in expected linear time. In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1*. Association for Computational Linguistics. p. 351-359.

NIVRE, Joakim; HALL, Johan; NILSSON, Jens. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In: *Proceedings of Language Resources and Evaluation Conference*. p. 2216-2219.

ORAVEC, Ján; BAJZÍKOVÁ, Eugénia. 1986. Súčasný slovenský spisovný jazyk: syntax. Slovenské pedagogické nakladateľstvo.

PÁLEŠ, Emil. 1994. SAPFO–Parafrázovač slovenčiny. Veda vydavateľstvo SAV.

PETROV, Slav; MCDONALD, Ryan. 2012. Overview of the 2012 shared task on parsing the web. In: *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*.

PIRINEN, Tommi, et al. 2010. Creating and weighting hunspell dictionaries as finite-state automata. *Investigationes Linguisticae*. 21: 1-16.

SCHMID, Helmut. 1995. Improvements in part-of-speech tagging with an application to German. In: *In Proceedings of the ACL SIGDAT-Workshop*.

SCHMID, Helmut. 1994. Probabilistic part-of-speech tagging using decision trees. In: *Proceedings of the international conference on new methods in language processing*. p. 44-49.

TJONG KIM SANG, Erik F.; DE MEULDER, Fien. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics. p. 142-147.

XIA, Fei; PALMER, Martha. 2001. Converting dependency structures to phrase structures. In: *Proceedings of the first international conference on Human language technology research*. Association for Computational Linguistics. p. 1-5.

Príloha A: Dokumentácia aplikácie Syncor

V tejto kapitole podrobne opíšeme funkcionalitu a architektúru aplikácie na spracovanie a vyhodnotenie dát, ktorú sme nazvali Syncor. Je implementovaná v jazyku Python a poskytuje jednoduché konzolové rozhranie, pomocou ktorého je možné vykonávať rôzne úpravy dát.

A.1 Prehľad funkcionality aplikácie

Aplikácia je implementovaná ako konzolová aplikácia v jazyku Python. Funkčné požiadavky aplikácie sa vyvíjali postupne s potrebami experimentov. Finálna funkcionalita ktorú ponúka je:

- sparsovanie Slovenského Závislostného Korpusu (alebo inej množiny dát v rovnakom formáte) do databázy,
- vygenerovanie množiny dát vo formáte CoNLL09, CoNLLX alebo MST z databázy a uloženie do súboru,
- rozdelenie množiny dát na trénovaciu a testovaciu množinu,
- odstránenie diakritiky z množiny dát,
- podrobnejšie štatistiky nad sparsovanou množinou dát:
  - základné LAS/UAS skóre,
  - podrobný prehľad úspešnosti pre jednotlivé DEPREL značky
  - prehľad zle určených DEPREL značiek, pre každú DEPREL značku v množine dát, aj s početnosťami,
  - prehľad všetkých DEPREL značiek pre danú morfologickú značku, aj s úspešnosťou určovania,
  - získanie konkrétnych inštancií viet pre danú DEPREL značku, ktorá bola v tej vete zle označená.
- pridanie gramatických chýb do textu v množine dát,
- kontrola pravopisu s opravou textu v množine dát,
- pridanie vlastnej morfologickej anotácie do množiny dát.

A.2 Prehľad architektúry aplikácie

Pôvodne bola aplikácia vyvíjaná ako jedna knižnica s konzolovým rozhraním. S rastúcou komplexnosťou sme nakoniec aplikáciu rozdelili na tri samostatné knižnice. Všetky obrázky

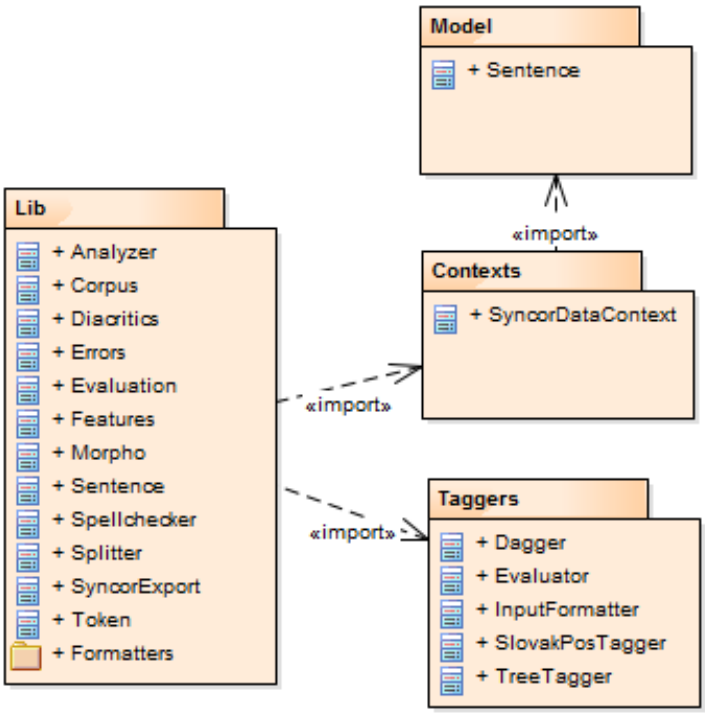
v tejto sekcií predstavujú UML diagramy. Knižnice ktoré v tejto kapitole bližšie opisujeme sú:

- *dp-syncor* – hlavná knižnica ktorá poskytujúca väčšinu funkcionality a poskytujúca konzolové rozhranie,
- *dp-common* – knižnica obsahujúca spoločné doménové entity ktoré sa používajú na mnohých miestach,
- *dp-errors* – knižnica poskytujúca funkcionalitu pridávania chýb do textu.

Okrem týchto knižníc sú implementované ešte ďalšie dve – *dp-scripts* a *dp-associations.* Obsahujú ale hlavne jednorazové skripty na vyhodnocovanie alebo spúšťanie niektorej funkcionality. Ich vnútornú štruktúru vzhľadom na ich ad hoc implementáciu a celkovú triviálnosť neuvádzame.

Nie všetky funkčné požiadavky boli implementované v samostatných knižniciach, vzhľadom na to, že niektoré sú pomerne triviálne a ich vyčlenenie by zbytočne zvýšilo komplexnosť riešenia. Generovanie chýb bolo vyčlenené vzhľadom na značnú zložitosť a množstvo logiky ktorá so zbytkom aplikácie nesúvisí.

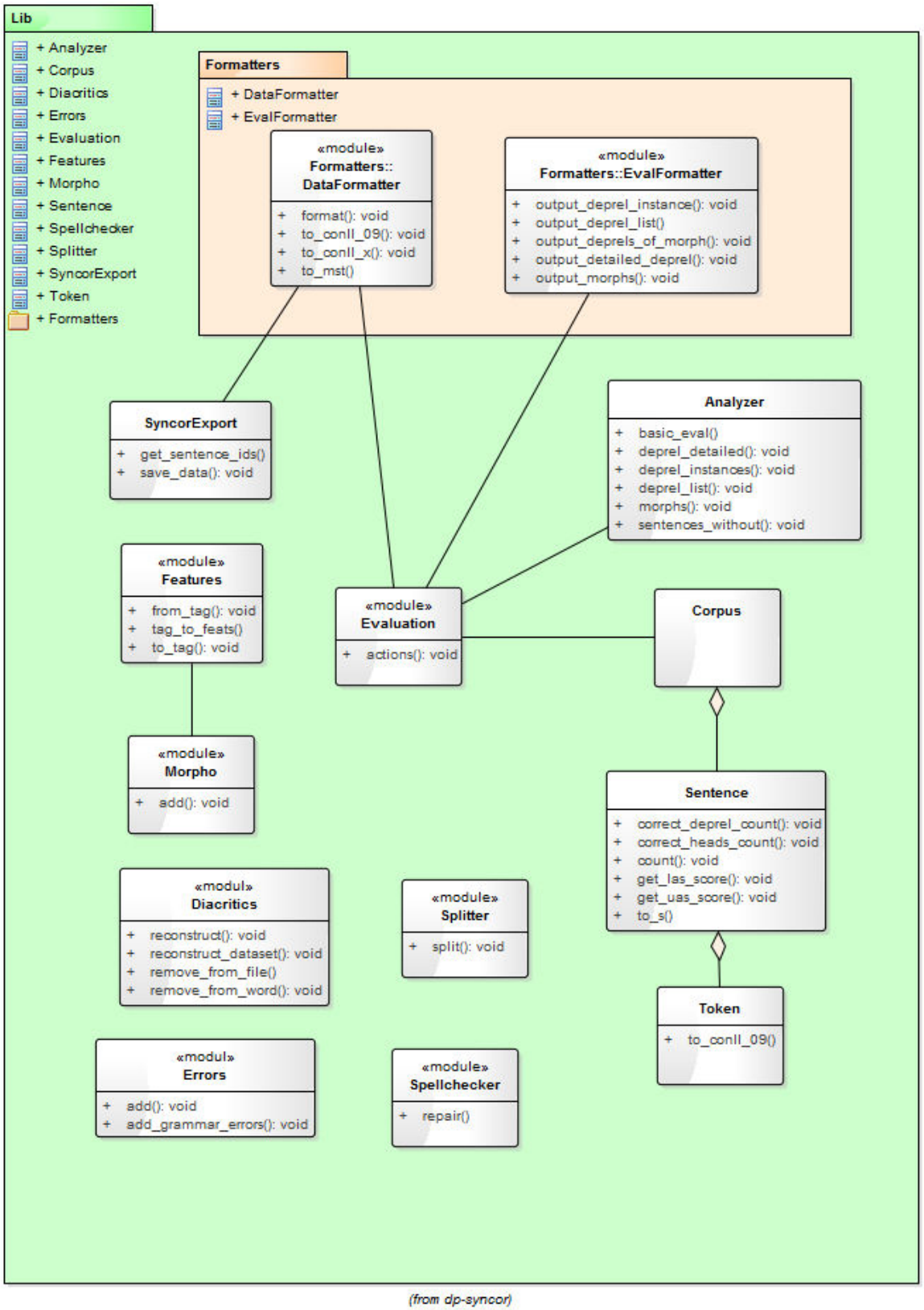
Prehľad balíkov hlavnej knižnice dp-syncor je znázornený na obrázku 19. Okrem závislostí je pri každom balíku zobrazený aj zoznam tried ktoré obsahuje.



Obrázok 19 – prehľad balíkov v knižnici dp-syncor, ich závislostí a vnútorných tried

Funkcionalitu týchto balíkov agreguje hlavný skript, ktorý poskytuje konzolové rozhranie a riadi hlavný tok riadenia (výstup do konzoly a spracovanie vstupu z konzoly). Podľa zadaných príkazov následne volá jednotlivé balíky a metódy.

Najkomplexnejší balík v tejto knižnici je balík *lib.* Jeho vnútorná štruktúra je znázornená na obrázku 20. Vidíme že naviac obsahuje vnorený balík *Formatters* – ten obsahuje dve triedy, ktoré sa starajú o formátovanie dát pri výstupu na terminál.



Obrázok 20 – vnútorná štruktúra balíka *lib* z knižnice *dp-synpar*

Balík *lib* obsahuje nasledujúce triedy:

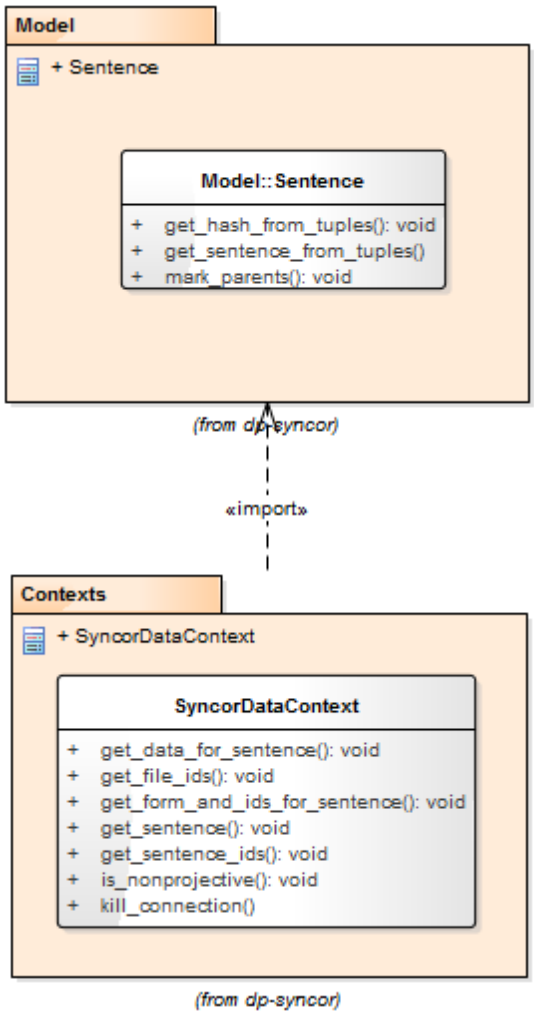
- *Analyzer* - obsahuje metódy, ktoré vykonávajú nad označkovanou a sparsovanú množinou dát podrobnejšie štatistiky a merania,
- *Corpus* – trieda ktorá reprezentuje označkovanú a sparsovanú množinu dát. Obsahuje zoznam všetkých viet v danej množine a zapuzdruje logiku čítania dát z rôznych dátových formátov. Funkcionalita tejto triedy bola postupom času nahradená knižnicou *dp-common* a jej triedou *Corpus*,
- *Diacritics* - trieda, ktorá vykonáva odstraňovanie diakritiky a rekonštrukciu pomocou webovej služby Diakritik,
- *Errors* – trieda ktorá pomocou externej knižnice *dp-errors* poskytuje funkcionalitu pridávania chýb do textu v zadanej množine dát,
- *Evaluation* – trieda, ktorá preberá riadenie toku programu v prípade, že sa používateľ rozhodol robiť nad označkovanou množinou dát podrobnejšie štatistiky. Spracováva používateľov vstup z konzoly a volá príslušné akcie triedy *Analyzer*. Výstup následne zobrazuje pomocou triedy *DataFormatter* a *EvalFormater*,
- *Features* - trieda, ktorá poskytuje funkcionalitu premieňania morfologických značiek na formát množiny vlastností tak, ako ich definujú CoNLL formáty. Rovnako poskytuje funkcionalitu pre (pole FEATS v CoNLL formátoch - potrebné pre parsre),
- *Morpho* – trieda, ktorá vykonáva morfologickú anotáciu textu, pomocou externého taggera,
- *Sentence* – trieda, ktorá reprezentuje jednu vetu v množine dát. Okrem zoznamu tokenov obsahuje ešte zopár pomocných metód, ktoré vyhodnocujú úspešnosť sparsovania,
- *Spellchecker* – trieda, ktorá vykonáva kontrolu pravopisu pomocou externého nástroja na kontrolu pravopisu,
- *Splitter* - trieda, ktorá rozdelí vstupnú množinu dát na testovaciu a trénovaciu
- *SyncorExport* – trieda, ktorá množinu dát z databázy pretransformuje na požadovaný formát a uloží ju do súboru na disk,
- *Token* – trieda reprezentujúca jedno slovo vo vete. V diagrame sú kvôli prehľadnosti zobrazené len jej dve atribúty, v skutočnosti ale obsahuje ešte naviac POS značku, morfologickú značku, množinu vlastností, DEPREL značku a HEAD

značku. Všetky navyše ešte vo variante z pôvodnej zlatej množiny dát a z množiny dát, ktorá bola označkovaná a sparsovaná.

Balík *Formatters* obsahuje dve triedy:

- *EvalFormatter* - trieda ktorá spracováva výstup z triedy Analyzer (ktorý je v podobe neupravených dátových štruktúr) a formátuje ju do podoby ktorá sa ľahšie číta, teda do rôznych tabuliek,
- *DataFormatter* - trieda, ktorá zoberie ako vstup vetu z databázy a pretransformuje ju do požadovaného formátu. Je možné volať priamo metódy na konkrétne formáty, alebo zavolať metódu format a formát výstupu nastaviť prostredníctvom parametra.

Balíky *Model* a *Context* spolu úzko súvisia a vzhľadom na ich pomerne jednoduchú štruktúru ich uvádzame na spoločnom obrázku 22.

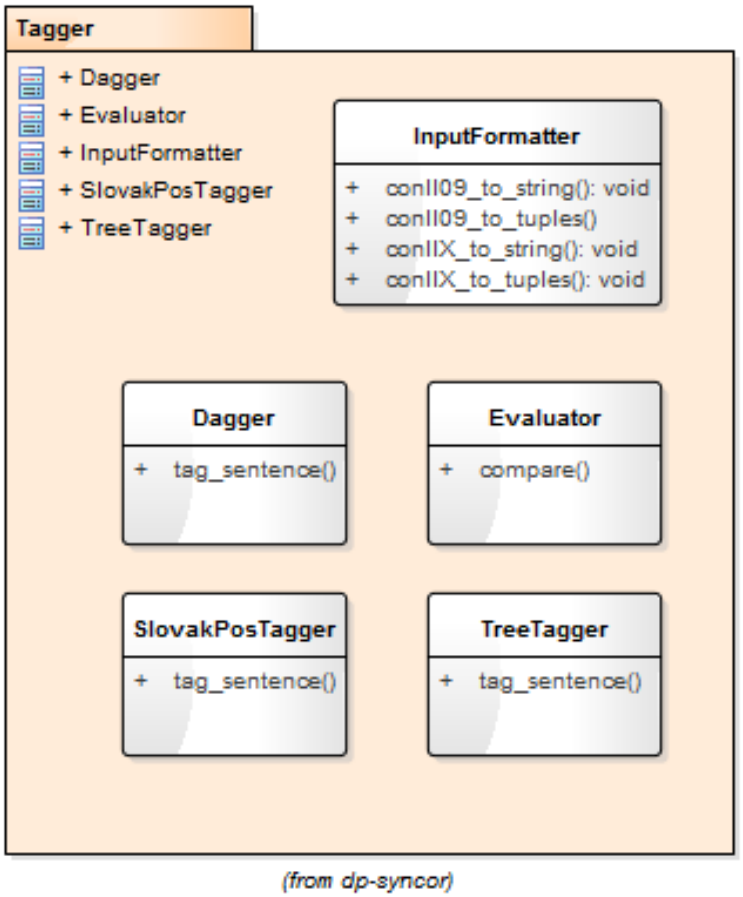


Obrázok 21 - prehľad štruktúry balíkov *model* a *contexts*

Trieda *SyncorDataContext* je zodpovedná za pripojenie k databáze a za vyťahovanie dát v príslušnej forme. Databáza v ktorej máme spracovaný Slovenský Závislostný Korpus uložená je PostgeSQL. Trieda *SyncorDataContext* využíva ako adaptér knižnicu psycopg2.

Trieda *Sentence* reprezentuje jednu vetu z databázy, z ktorej ju získa ako zoznam "dvojíc", čo je dátová štruktúra jazyka Python. V triede *Sentence* sú implementované metódy, ktoré pretransformujú dáta z týchto štruktúr do formátu vhodného na ďalšie spracovanie. Okrem toho ešte obsahuje metódu mark_parents(), ktorá opravuje chybu v pôvodnej množine dát. Konkrétne ide o označenie nadradených vetných členov. V pôvodnom Slovenskom Závislostnom Korpuse sa totiž vyskytujú vety, ktorých číslovanie nezačína od 1, ale od nejakého iného čísla. Toto chybné číslovanie spôsobovalo problémy v niektorých iných častiach aplikácie, hlavne v tých, v ktorých sa pracovalo s nadradenými vetnými členmi.

Na obrázku 23 uvádzame štruktúru balíka *Taggers*.



Obrázok 22 – vnútorná štruktúra balíka *Taggers*

Tento balík obsahuje tri triedy ktoré vykonávajú morfologickú analýzu:

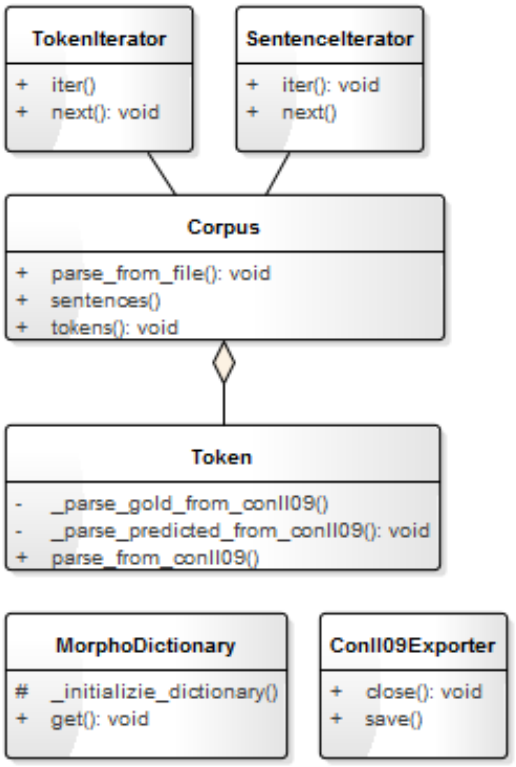
- *Dagger* – na morfologickú anotáciu používa webovú službu Dagger,

- *SlovakPosTagger* – na morfologickú anotáciu používa webovú službu SlovakPosTagger,
- *TreeTagger* - na morfologickú anotáciu používa nástroj TreeTagger, k úspešnému fungovaniu potrebuje mať nastavenú cestu k vykonateľným binárnym súborom a k modelu slovenského jazyka pre morfologickú anotáciu.

Okrem tried na vykonanie samotnej morfologickej anotácie ešte balík obsahuje ďalšie dve súvisiace triedy:

- *InputFormatter* – trieda ktorá spracováva text z formátov CoNLL09 a CoNLLX to formátu potrebného pre morfologickú anotáciu,
- *Evaluator* – trieda ktorá vykonáva jednoduché vyhodnotenie morfologickej anotácie vzhľadom na poskytnutú zlatú vzorku.

Druhou samostatnou knižnicou je *dp-common.* Jej štruktúru uvádzame na obrázku 23.



Obrázok 23 – triedy knižnice *dp-common*

Triedy ktoré poskytuje sú:

- *Corpus* – trieda zapuzdrujúca načítanie korpusu zo súboru a iteráciu cez jeho dáta. Za týmto účelom je implementovaný vzor iterátor. Trieda *Corpus* pracuje s dvoma:

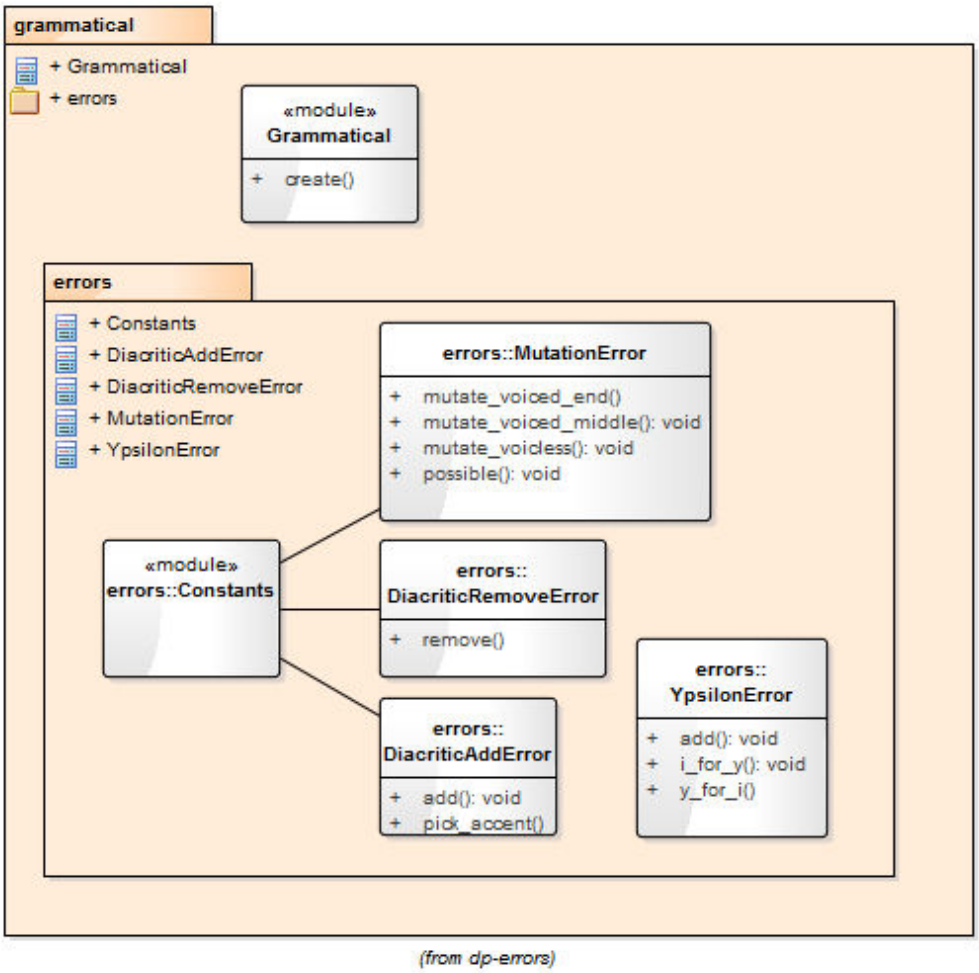
    - *TokenInterator* – prechádza korpus po jednotlivých tokenoch,
    - *SentenceIterator* – prechádza korpus po vetách – jedna iterácia vráti kolekciu tokenov reprezentujúcu jednu vetu z korpusu,
- *Token* – reprezentuje jeden token (slovo) z korpusu a je agregovaná triedou *Corpus*,
- *MorphoDictionary* – pri inicializácií načíta morfologickú databázu zo súboru a následne poskytuje funkcionalitu na získavanie údajov z tejto databázy,
- *Conll09Exporter* – trieda poskytujúca funkcionalitu na export viet do súboru.

Tieto triedy sú používané vo viacerých iných knižniciach a boli navrhnuté a vytvorené tak, aby sa dali ľahko používať aj priamo z terminálu.

Poslednou významnou samostatnou knižnicou je knižnica *dp-errors*. Skladá sa z dvoch samostatných balíkov:

- *grammatical* – poskytuje funkcionalitu pridávania gramatických chýb,
- *typos* – poskytuje funkcionalitu na pridávanie preklepov.

Vnútornú štruktúru balíka *grammatical* uvádzame na obrázku 24. Z obrázku vidíme, že obsahuje ešte vnútorný balík *errors*. Zatiaľ čo balík *errors* obsahuje logiku pridávania chýb, balík *grammatical* poskytuje verejné rozhranie pre pridávanie.



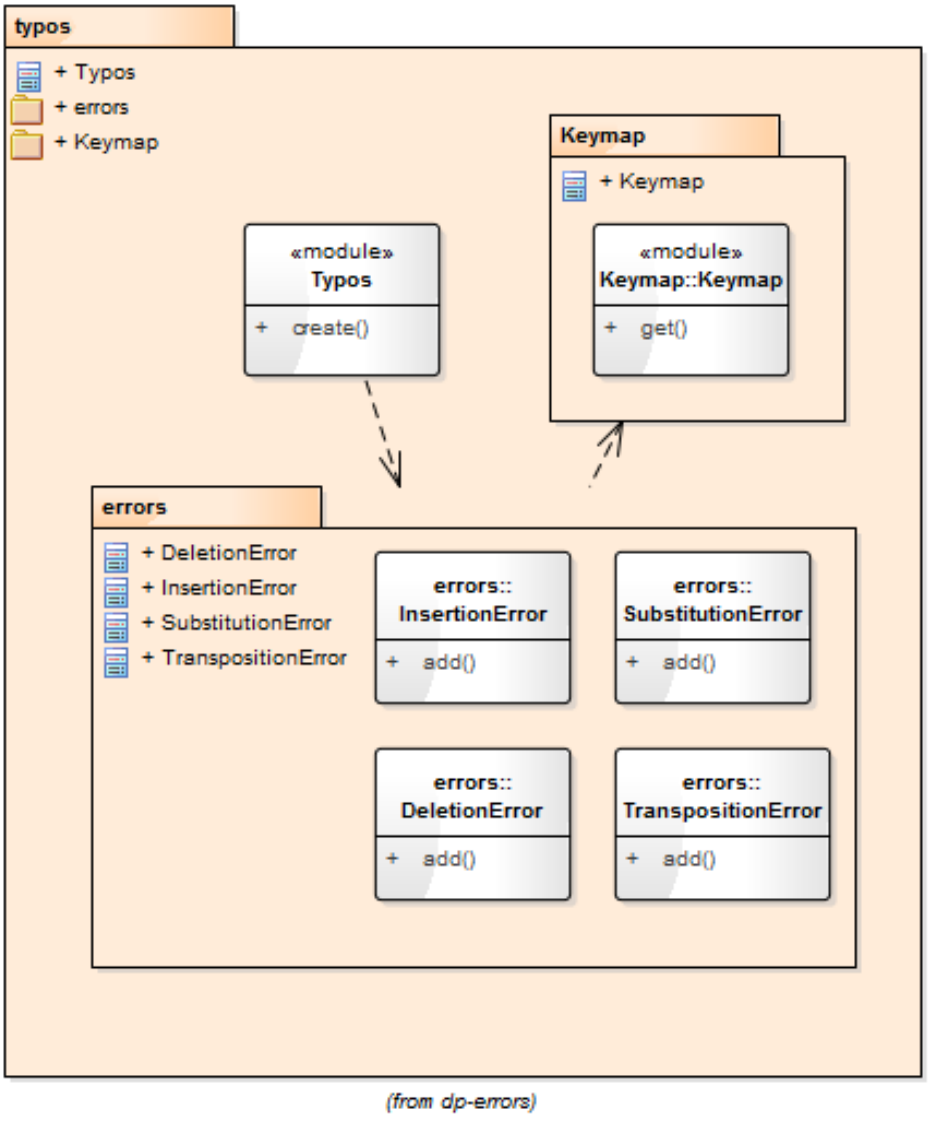
Obrázok 24 – vnútorná štruktúra balíka *grammatical*

Triedy v tomto balíku sú nasledovné:

- *Grammatical* – poskytuje rozhranie na pridanie chyby do slova,
- *Constants* – obsahuje konštanty potrebné pre pridávanie niektorých typov chýb, ako napríklad zoznamy znelých a neznelých spoluhlások, alebo zoznam znakov s diakritickými znamienkami,
- *MutationError* – trieda obsahujúca logiku pridávania chýb spodobovania. Okrem metód pre jednotlivé typy spodobovania obsahuje aj metódu, ktorá určuje, či je pre dané slovo možné vygenerovať chybu v spodobovaní,
- *DiacriticsRemoveError* – trieda obsahujúca logiku pridania chyby spočívajúcej vo vynechaní diakritického znamienka,
- *DiacriticsAddError* – trieda obsahujúca logiku pridania chyby spočívajúcej v pridaní diakritického znamienka,

- *YpsilonError* – trieda obsahujúca logiku pridania chyby spočívajúcej v zamenení znakov jota a ypsilon.

Druhým samostatným balíkom v tejto knižnici je balík *typos*. Jeho vnútorná štruktúra je znázornená na obrázku 25.



Obrázok 25 – vnútorná štruktúra balíka *typos*

Návrh tohto balíka je analogický k balíku *grammatical*. Hlavný balík *typos* poskytuje verejnú metódu na pridanie preklepu a vnorené balíky *keymap* a *errors* obsahujú logiku pridávania preklepov. Triedy v tomto balíku sú:

- *Typos* – poskytuje verejnú metódu na pridanie preklepu do slova,

- *Keymap* – trieda obsahujúca konštanty potrebné na pridanie niektorých typov preklepov. Konkrétne ide o mapu kláves, ktoré susedia s danou klávesov, podľa štandardného QWERTZ rozloženia,
- *InsertionError* – trieda obsahujúca logiku chyby typu náhodného pridania znaku do slova,
- *SubstitutionError* – trieda obsahujúca logiku chyby typu náhodného nahradenia znaku iným znakom,
- *DeletionError* – trieda obsahujúca logiku chyby typu náhodného zmazania jedného znaku zo slova,
- *TranspositionError* – trieda obsahujúca logiku chyby náhodného vymenenia poradia dvoch susediacich znakov v slove.

A.3 Inštalačná príručka

Na spustenie aplikácie na spracovanie dát treba mať nainštalovaný programovací jazyk Python minimálne vo verzií 3 (aplikácia nie je spätne kompatibilná) a databázu PostgreSQL. Do databázy potom treba naimportovať súbor database.dump prostredníctvom príkazu:

```
cat database.dump | psql meno_lokalnej_databazy
```

Kde „meno_lokalnej_databazy“ je názov inštancie Postgres databázy. Ďalším predpokladom sú nainštalované potrebné knižnice. Tie sa dajú postupne nainštalovať prostredníctvom nasledujúcich príkazov (pre operačný systém Ubuntu):

```
sudo apt-get install python3-pip
sudo pip3 install psycopg2
sudo pip3 install pyyaml
sudo pip3 install IPython
sudo pip3 install unidecode
sudo pip3 install beautifulsoup4
```

Ďalším krokom je nainštalovanie knižníc *dp-common* a *dp-errors.* Na ich nainštalovanie je potrebné prejsť do ich koreňového adresára a z terminálu spustiť príkaz:

```
sudo make
```

Ten nainštaluje knižnicu do systému, takže ju potom môže hlavná knižnica *dp-syncor* používať. Tú následne spustíme tak, že v jej koreňovom adresári spustíme príkaz:

```
python app.py
```

Je dôležité aby verzia jazyku Pyhton bola minimálne 3. Niektoré distribúcie systému Ubuntu majú v sebe jazyk Python už nainštalovaný, ale vo verzií 2. V takom prípade je potrebné spustiť aplikáciu pomocou príkazu:

```
python3 app.py
```

ktorý aplikácu spusti Pythonom vo verzií 3.

Príloha B: Dokumentácia webovej služby Synpar

V tejto kapitole opíšeme dostupnú funkcionalitu webovej služby pre parsovanie slovenčiny a jej architektúru.

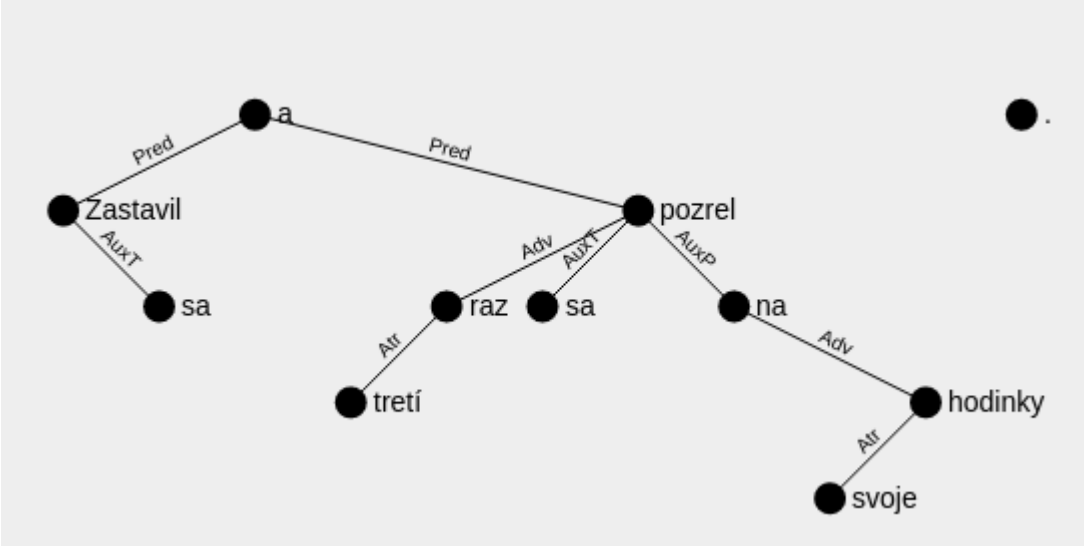
B.1 Prehľad funkcionality webovej služby

Webová aplikácia vykonáva jednu úlohu, a to parsovanie slovenčiny. Pre samotné parsovanie je možné nastaviť niekoľko parametrov ovplyvňujúcich proces parsovania a výstup.

Parsovanie textu je možné vykonávať pomocou POST žiadosti na server. Dáta ktoré sa majú v tele requestu poslať sú nasledujúce (prvé slovo je vždy kľúč pre dané pole):

- diacritics – možné hodnoty: „on“, „off“ (predvolená hodnota „off“). Nastavuje, či sa má pred parsovaním vykonávať rekonštrukcia diakritiky alebo nie,
- spellchecking - možné hodnoty: „on“, „off“ (predvolená hodnota „off“). Nastavuje, či sa má pred parsovaním vykonávať kontrola pravopisu alebo nie,
- parserType – možné hodnoty: „basic“, „betterPreds“ (predvolená hodnota „basic“). Nastavuje, ktorý model sa má pre parsovanie použiť. Pri hodnote „basic“ sa použije všeobecný model, pri hodnote „betterPreds“ sa použije model, ktorý s väčšou presnosťou určuje analytickú funkciu Pred,
- outputType – možné hodnoty: „raw“, „visualize“ (predvolená hodnota „raw“). Nastavuje formát výstupu parsera. Pri hodnote „raw“ je výstup textový vo formáte CoNLL09, pri hodnote „visualize“ sa zobrazí stránka s vizualizáciou vety v podobe grafu,
- text – text, ktorý sa ma sparsovať,

Parameter text je povinný, všetky ostatné majú preddefinované hodnoty, takže sú nepovinné. Ukážka závislostného stromu, ktorý sa vykreslí v prípade, že je parameter outputType nastavený na hodnotu „visualize“ je na obrázku 26.



Obrázok 26 - ukážka grafického znázornenia závislostného stromu

Každý uzol predstavuje token a hrana predsavuje analytickú funkciu, ktorú token spĺňa vo vzťahu k nadradenému členovi. Tokeny sú zoradené z ľava do prava podľa poradia vo vete a zhora dole podľa vzdialenosti od koreňa vety. Na funkcionalitu vykreslenia grafu sme použili voľne dostupnú Javascriptovú knižnicu Sigma.js[16].

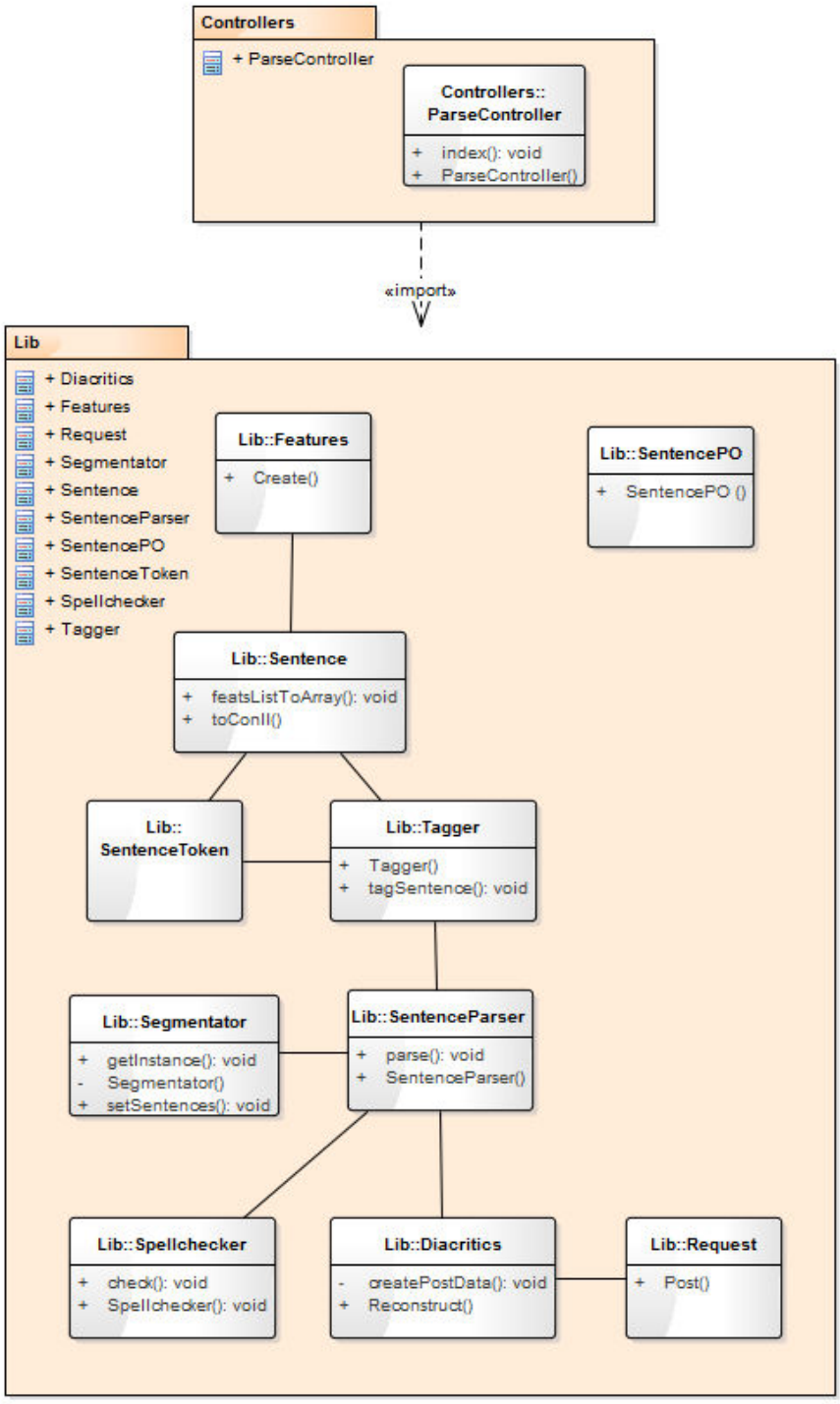
B.2 Architektúra webovej služby

Webová služba je implementovaná v jazyku Java. Toto rozhodnutie bolo vynútené tým, že Mate tools grafový parser je implementovaný a dostupný ako knižnica v jazyku Java. Pre jazyk Python, v ktorom máme implementovanú aplikáciu Syncor nebola dostupná žiadna obaľovacia knižnica, a pokiaľ vieme nie je dostupná ani pre žiadny jazyk. Zvažovali sme možnosť knižnicu Mate parsera implementovať iba ako službu bežiacu na pozadí, ktorú by volala webová služba implementovaná v Pythone. Po krátkej analýze sa nám ale v tomto riešení objavilo viacero technických problémov, po ktorých zhodnotení sme usúdili, že implementácia celej webovej služby v Jave, je menšie technické riziko a eventuálne aj jednoduchšie riešenie.

Na implementáciu webovej služby sme použili minimalistický pracovný rámec Ninja[17]. Poskytuje základnú MVC (model-view-controller) architektúru a má veľmi jednoduché API, ktoré vzhľadom na jednoduchosť svojho použitia výrazne znižuje nároky na vstupnú technickú znalosť. Použitie pracovného rámca udáva jasný smer návrhu a implementácií, v našom prípade to bolo vytvorenie základných HTML stránok (z angl. views), ktoré posky-

[16] http://sigmajs.org/
[17] http://www.ninjaframework.org/

tujú potrebné informácie pre používateľa, triedu typu controller, na spracovanie požiadavky na server a spracovanie vstupných a výstupných dát a balíček tried, ktoré implementujú experimentálne overené vylepšenia z tejto práce. Prehľad vlastných tried uvádzame v diagrame na obrázku 27.



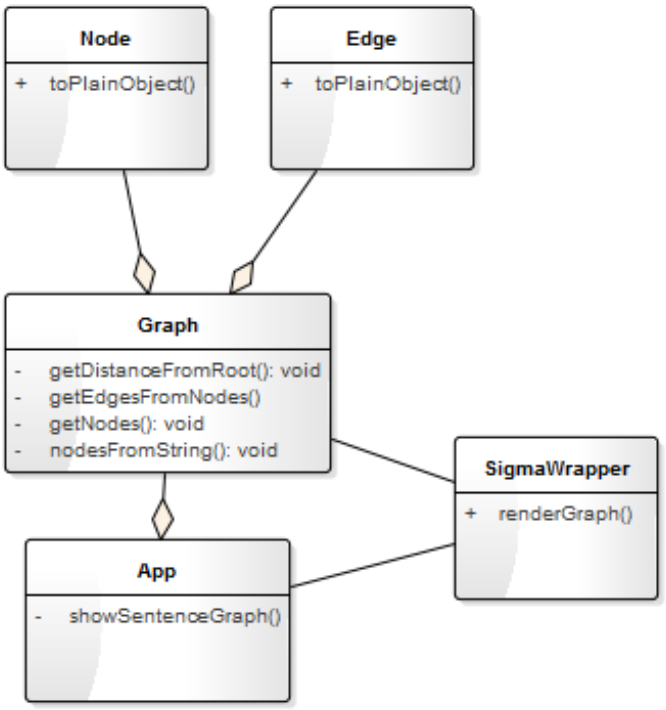
Obrázok 27 – prehľad štruktúry vlastných tried vo webovej službe Synpar

Trieda *ParseController* posúva vstup od používateľa ďalej do triedy *SentenceParser*. Okrem toho na základe vstupného parametra rozhodne, aký formát výstupu sa po ukončení parsovania pošle klientovi. *SentenceParser* tvorí jadro balíčka *Lib.* Zapuzdruje upravovanie vstupného textu na základe vstupných parametrov a vykonáva volanie knižníc na morfologickú anotáciu a parsovanie. Zvyšné triedy v balíčku *Lib* sú:

- *Diacritics* – trieda vykonávajúca rekonštrukciu diakritiky volaním webovej služby Diakritik,
- *Features* – trieda, ktorá zo štandardnej morfologickej značky vytvorí množinu morfologických vlastností vo formáte CoNLL09,
- *Request* – trieda, ktorá slúži na vykonávanie POST požiadaviek,
- *Segmentator* – trieda, ktorá delí vstupný text na jednotlivé vety. Na túto funkcionalitu sme použili knižnicu OpenNLP[18],
- *Sentence* – trieda, ktorá drží vstupné dáta, vrátane parametrov úprav vstupného textu,
- *SentenceParser* – trieda, ktorá vykonáva parsovanie, vrátane úprav vstupného textu a morfologickej anotácie,
- *SentencePO* – trieda, ktorá drží dáta určené na zobrazenie na klientskej strane (PO – prezentačný objekt, z angl. presentation object),
- *SentenceToken* – trieda, ktorá reprezentuje jeden token vo vete, používa sa hlavne pri morfologickej anotácií,
- *Spellchecker* – trieda, ktorá vykonáva kontrolu pravopisu prostredníctvom externej knižnice Hunspell,
- *Tagger* – trieda, ktorá vykonáva morfologickú anotáciu prostredníctvom externej knižnice TreeTagger.

Okrem implementácie parsovania na serveri, sme na vizualizáciu závislostného stromu sparsovanej vety implementovali aj malú knižnicu v jazyku Javascript. Jej štruktúru môžeme vidieť na obrázku 28.

[18] https://opennlp.apache.org/



Obrázok 28 – znázornenie štruktúry knižnice na zobrazovanie závislostného stromu.

Vykreslovacie jadro pre zobrazovanie grafov je knižnica sigma.js. Na to, aby mohla vykresliť graf, musíme pre ňu najprv nachystať dáta – skonvertovať ich z CoNLL09 formátu do dátovej štruktúry podľa potrieb knižnice. Na to slúži trieda *Graph*, ktorá vo svojom konštruktore spracuje textový zápis CoNLL09 formátu do objektov *Node* a *Edge*. Tie reprezentujú uzly a hrany v grafe.

Tieto triedy sa využívajú v hlavnom module aplikácie *App*. Tá po načítaní stránky prehľadá HTML DOM, pričom hľadá elementy so špecifickou triedou, ktorých textový obsah je práve CoNLL09 zápis vety. Tieto zápisy spracuje a v pamäti s vytvorí reprezentácie v dátových štruktúrach potrebných pre knižnicu sigma.js. Graf sa vykreslí po kliknutí na konkrétnu vetu zo zoznamu.

B.3 Inštalačná príručka

Aplikáciu je možné spustiť dvoma spôsobmi – buď na lokálnom serveri určenom na vývoj (Jetty), alebo nasadením na Tomcat7 server. Pre obidve je ale potrebné mať najprv správne nastavený systém, respektíve premenné prostredia a nainštalovaný TreeTagger. Ten je mož-

né pomerne ľahko nainštalovať prostredníctvom inštrukcií na domovskej stránke[19]. Následne treba nastaviť nasledujúce premenné prostredia:

- SP_MODEL_PATH – cesta k základnému modelu pre syntaktický parser
- SP_PRED_MODEL_PATH – cesta k modelu s vylepšeným určovaním pre Pred analytické funkcie
- TREETAGGER_HOME – cesta do koreňového adresára inštalácie TreeTaggera,
- TGR_MODEL – názov modelu pre TreeTagger (väčšinou to býva „slovak-utf8.par“),
- HUNSPELL_AFF – cesta k súboru s príponami pre Hunspell,
- HUNSPELL_DIC – cesta k slovníku pre Hunspell.
- SEGMENTATION_MODEL – cesta k modelu na segmentáciu.

Uvádzame ešte príklad pre tieto premenné tak, ako sú nastavené na produkčnom serveri:

```
SP_MODEL_PATH="/home/jarino/models/model-full"

SP_PRED_MODEL_PATH="/home/jarino/models/model-pred"

TREETAGGER_HOME="/home/jarino/treetagger"

TGR_MODEL="slovak-utf8.par"

HUNSPELL_AFF="/home/jarino/sk_SK.aff"

HUNSPELL_DIC="/home/jarino/sk_SK.dic"

SEGMENTATION_MODEL="/home/jarino/en-sent.bin"
```

Pre spustenie aplikácie vo vývojovom móde, treba v koreňovom adresári projektu spustiť príkaz:

```
mvn ninja:run
```

Pre nasadenie aplikácie na Tomcat7 server stačí spustiť príkaz:

```
sudo make
```

Ten automaticky skompiluje projekt, skopíruje potrebné súbory do predvoleného adresára pre Tomcat server a rovno ho aj reštartuje. Prvé spustenie aplikácie je časovo dosť náročné, Aplikácia pri ňom musí do pamäti načítať všetky potrebné modely. Na svoje úspešné fungovanie potrebuje minimálne 2GB RAM.

[19] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

Príloha C: obsah dátového nosiča

K práci je priložený dátový nosič, ktorého adresárová štruktúra je následovná:

\syncor – zdrojové kódy aplikácie na spracovanie dát

\synpar – zdrojové kódy webovej služby

\data – obsahuje všetky dáta s ktorými sme v tejto práci pracovali

\dokumenty – obsahuje elektronickú verziu tejto práce, článok prijatý na IIT.SRC a článok poslaný na HrTAL

Príloha D: prehľad analytických funkcií aj s vysvetlivkami

Pod počtom sa myslí množstvo výskytov v Slovenskom Závislostnom Korpuse.

Tabuľka 16 - prehľad analytických funkcií nachádzajúcich sa v Slovenskom Závislostnom Korpuse aj s početnosťami výskytu

Funkcia	Popis	Počet
Pred	Predikát, resp. uzol ktorý závisí na inom	33946
Sb	Subjekt (podmet)	51078
Obj	Objekt (predmet)	79145
Adv	Adverbiale (príslovkové určenie)	105440
Atv	Doplnok, iba určujúci, visí na neslovenskom člene	1528
AtvV	Doplnok, iba určujúci, visí na slovese (chýba druhý riadiaci člen)	812
Atr	Atribút (prívlastok)	212202
Pnom	Nominálny predikát, resp.menná časť prísudku so sponou byť	9805
AuxV	Pomocné sloveso byť - auxiliary verb	14303
Coord	koordinačný uzol, priraďovacie spojenie	20003
Apos	Apozícia (hlavný uzol)	2337
AuxT	Zvratné sa, neddeliteľné sa - reflexívne tantum	24605
AuxR	zvratné sa, ktoré nie je Obj ani AuxT (tvorí reflexívne pasívum)	6271
AuxP	Primárna predložka, časti sekundárnej predložky	39690
AuxC	Podraďujúca spojka	10943
AuxO	Nadbytočný element - odkazovací, emotívny	1008
AuxZ	zdôrazňovacie slovo	19943
AuxX	Čiarka, nie ale ako nositeľ koordinácie - priraďovacieho vzťahu	73959
AuxG	ine grafické symboly, ktoré neukončujú vetu	42322
AuxY	príslovky a častice, ktoré sa nedajú zaradiť inam	23110
AuxS	Koreň stromu	-
AuxK	Koncová interpunkcia vety	49705
ExD	Náhradná funkcia pre technické hrany vedúce namiesto od elifovaného člena k pseudotriediacemu slovu aleo pre hlavný člen bez predikátu	22964
AtrAtr	Riadiacim slovo atribútu môže byť vďaka štruktúrnej viacznačnosti akékoľvek z bezprostredne predchádzajúcich podstatných mien	159

AtrAdv	Štruktúrna viacznačnosť medzi závislosťou adverbálnou (príslovečnou) a adnominálnou (zavesenou na meno) bez sémantických dôsledkov	412
AdvAtr	Detto, s opačnou preferenciou	62
AtrObj	Štruktúrna viacznačnosť medzi závislosťou objektovou a adnominálnou (zavesenou na substantívum) bez sémantických dôsledkov	20
ObjAtr	Dtto, s opačnou preferenciou	6

Príloha E: prehľad úspešnosti určovania DEPREL značiek jednotlivých parserov

E.1 MATE parser grafový

Tabuľka 17 - prehľad analytických funkcií (DEPREL) aj s úspešnosťou ich určovania grafovým MATE parserom

DEPREL	Celkový počet	Počet správne určených	Percentuálne vyjadrenie
AuxK	9884	9779	98.94%
AuxV	3102	2551	82.24%
AuxG	9314	6705	80.65%
AuxT	4970	3993	80.34%
AuxX	15275	12029	78.75%
Atr	44329	33465	75.49%
AuxP	7533	5627	74.70%
Sb	10145	7410	73.04%
Adv	21322	14207	66.63%
Obj	15883	10295	64.82%
Pnom	1769	1059	59.86%
Pred	6409	3785	59.06%
Coord	3741	2121	56.70%
AuxZ	4059	2281	56.20%
AuxC	2133	1055	49.46%
AuxY	4636	2226	48.02%
ExD	4526	1754	38.75%
Apos	443	138	31.15%
AtvV	167	40	23.95%
Atv	280	50	17.86%
AuxR	1186	163	13.74%
AuxO	252	24	9.52%

AtrAdv	87	1	1.15%
AtrAtr	27	0	0
AdvAtr	11	0	0
AtrObj	1	0	0
???	148	0	0
ObjAtr	4	0	0

E.2 MST parser - hrubý

MST parser natrénovaný na morfologických značkách, ktoré obsahovali iba slovný druh

Tabuľka 18 - prehľad analytických funkcií (DEPREL) aj s úspešnosťou ich určovania MST parserom, natrénovanom na hrubej množine morfologických značiek

Deprel	Celkový počet	Počet správne určených	Percentuálne vyjadrenie
AuxK	9884	9748	98.62%
AuxT	4970	3999	80.46%
AuxX	15275	11786	77.16%
AuxP	7533	5700	75.67%
AuxV	3102	2347	75.66%
AuxG	8314	6161	74.10%
Atr	44329	32580	73.50%
Pred	6409	3972	61.98%
Adv	21322	13175	61.79%
Coord	3741	2272	60.73%
Sb	10145	5889	58.05%
Obj	15883	8715	54.87%
Pnom	1769	925	52.29%
AuxZ	4059	2080	51.24%
AuxC	2133	959	44.96%
AuxY	4636	1996	43.05%
ExD	4526	1424	31.46%
Apos	443	120	27.09%
AtvV	167	21	12.57%
AuxR	1186	104	8.77%

Atv	280	18	6.43%
AuxO	252	12	4.76%
AdvAtr	11	0	0.00%
AtrAdv	87	0	0.00%
AtrObj	1	0	0.00%
ObjAtr	4	0	0.00%
???	148	0	0.00%
AtrAtr	27	0	0.00%

E.3 MST parser - jemný

MST parser natrénovaný na morfologických značkach, ktoré obsahovali všetky morfologické vlastnosti.

Tabuľka 19- prehľad analytických funkcií (DEPREL) aj s úspešnosťou ich určovania MST parserom, natrénovanom na jemnej množine morfologických značiek

Deprel	Celkový počet	Počet správne určených	Percentuálne vyjadrenie
AuxK	9884	9750	0.986443
AuxV	3102	2465	0.794649
AuxX	15275	11874	0.777349
AuxT	4970	3860	0.77666
AuxG	8314	6184	0.743806
Atr	44329	32351	0.729793
AuxP	7533	5440	0.722156
Sb	10145	7143	0.704091
Adv	21322	13616	0.638589
Obj	15883	9793	0.616571
Pred	6409	3886	0.606335
Pnom	1769	1006	0.568683
Coord	3741	2027	0.541834
AuxZ	4059	2007	0.494457
AuxC	2133	1019	0.477731
AuxY	4636	2075	0.447584
ExD	4526	1400	0.309324

Apos	443	109	0.24605
AtvV	167	27	0.161677
AuxR	1186	151	0.127319
AuxO	252	22	0.087302
Atv	280	17	0.060714
AtrAdv	87	1	0.011494
???	148	0	0
AtrObj	1	0	0
AtrAtr	27	0	0
AdvAtr	11	0	0
ObjAtr	4	0	0

E.4 Malt parser

Malt parser natrénovaný pomocou algoritmu Stack-Lazy.

Tabuľka 20 - prehľad analytických funkcií (DEPREL) aj s úspešnosťou ich určovania Malt parserom

Deprel	Celkový počet	Počet správne určených	Percentuálne vyjadrenie
AuxK	9666	9884	0.977944
AuxV	2445	3102	0.788201
AuxT	3869	4970	0.778471
AuxX	11876	15275	0.77748
AuxG	6377	8314	0.767019
Atr	33310	44329	0.751427
Sb	6837	10145	0.673928
AuxP	4975	7533	0.660427
Adv	13262	21322	0.621987
AuxZ	2361	4059	0.58167
Obj	9177	15883	0.577788
Pred	3338	6409	0.52083
Coord	1911	3741	0.510826
Pnom	897	1769	0.507066
AuxC	1001	2133	0.469292

AuxY	1802	4636	0.388697
ExD	1485	4526	0.328104
AtvV	26	167	0.155689
Apos	62	443	0.139955
AuxR	141	1186	0.118887
Atv	18	280	0.064286
AuxO	16	252	0.063492
???	6	148	0.040541
AdvAtr	0	11	0
ObjAtr	0	4	0
AtrAtr	0	27	0
AtrObj	0	1	0
AtrAdv	0	87	0

Príloha F: článok prijatý na IIT.SRC

Automatic text processing – syntactic analysis of a sentence

Jaroslav LOEBL*
Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
jaroslavloebl@gmail.com

Abstract. Syntactic analysis in terms of natural language processing (NLP) is an important step in deeper understanding of text. Extracted dependencies between words can be profitable for many other tasks in NLP, i.e. machine translation, semantic analysis, or information extraction. Throughout last two decades we witness advances in quality of syntactic parsing, mainly due to statistical and machine learning methods, which gained popularity with growing processing power of computers. It is no surprise that very little of this research was done specifically for Slovak language, which was caused by the absence of proper syntactically annotated dataset. Thankfully, this is slowly changing with existence of Slovak Dependency Corpus (SDC). In this paper we focus on comprehensive evaluation of existing parsers on this dataset, while aiming at specific issues related to Slovak language and real-world data.

1. Introduction

Syntactic analysis of a sentence can be simply understood as extracting dependency structure from input sentence. Thanks to this structure we know which word depends on another and type of this relation. These are quite useful information for many other tasks from the field of NLP 0. At first, syntactic parsing was performed via context-free grammars 0. This approach was significantly outperformed by data-driven approaches that emerged in the late nineties. Two primary reasons were rapid growth of processing power and existence of large syntactically annotated corpora - treebanks. During the years of research two main paradigms emerged - graph-based parsing 3)4) and transition-based parsing 5)6). Typically, graph-based parsers perform parsing by searching for highest scoring graph0, while transition parsers take highest-scoring transition from every state until the whole sentence is processed 0. Besides this, two ways of representing parsed sentence were used - phrase and dependency structure. Phrase structure recursively split sentence into noun phrase and verb phrase, until words are reached, which are then leafs of the tree structure (figure 1 left). In dependency structure words are nodes of the tree and edges mark relationship between two words (figure 1 right). Many tasks in NLP benefits from having direct relationship between words 7), rather than having sentence split into constituents.

---

* Master degree study programme in field: Information Systems
Supervisor: Dr. Marián Šimko, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

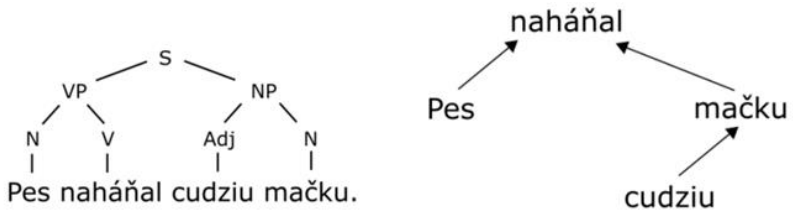


Figure 1. Example of phrase structure tree (left) and dependency structure tree (right)

As mentioned above, syntactically annotated corpus is necessary for training a data-driven parser. For many years there are available corpora for English20 and many other languages, including Czech21. Creation of SDC in recent years made data-driven parsing research finally available also for Slovak language 8). Like its foreign counterparts, it consists of corrected texts. Current state-of-the art parsers have been evaluated mainly on those edited texts, achieving accuracies well above 90%. However, it has been shown that parsing accuracies are barely over 80% when used in practice 0. When parsing texts acquired from the wild Web (blogs, forums, discussions), we have to cope with the fact that most of these texts are unedited and tend to contain spelling or grammar mistakes or no punctuation. This is an open problem in syntactic analysis very little attention has been put in by research communities. The main aim of our work is to examine syntactic analysis of real-world data and recognize its major obstacles causing reduction of parsing accuracy. We evaluate existing top-performing parsers on data in Slovak and we experiment with different setups. The other important issue is that very little is known about specificities of parsing Slovak language. We identify the most significant problems with text quality of text input and will try to overcome these problems with use of available tools for text reconstruction and simulating errors of unedited text in our edited corpora.

2. Related work

2.1. Parsers

Development and research of data-driven parsers for Slovak language was held back by absence of syntactically annotated dataset. Thus research focused mainly on rule based systems, such as parsing context free grammar 10), bottom-up knowledge based parsing using regular expressions and xml based rules 11), or adaptation of Czech knowledge based parsers on Slovak language 12). Recently, with emergence of SDC, first work focused on data-driven parsing emerged, which adapted MaltParser for Slovak language 8). MaltParser is an implementation of inductive dependency parsing implementing several algorithms for deterministic parsing. Besides that, many other settings are available, which needed to be optimized for Slovak language. MaltParser was comprehensively evaluated on 10 languages, including Czech and English and consistently giving a dependency accuracy of 80–90% 6). Also, it is the only one parser, which has also been evaluated in Slovak language, achieving 71.72% labeled attachment score and 77.81% unlabeled attachment score 8) (explanation of these metrics is provided in table 2). As we focus on adaptation of existing parsers on Slovak language, we further examined another three parsers: Maximum Spanning Tree parser (MST parser) 3), transition based parser from Mate tools 5) and graph based parser from Mate tools 4).

MST parser formalize parsing as search for maximum spanning tree. First version used modified Eisner algorithm, which was later substituted by Chu-Liu-Edmonds algorithm. Though it was not at first designed to parse non-projective structures, it is capable of such parsing. It achieved 90.20% accuracy (number of words which correctly identified their parent in sentence) for English and 84.40% accuracy for Czech.

Graph based Mate tools parser is based on MST parsers, modified to speed up training and parsing. It reached 90,33% labeled attachment score for English and 80,96% for Czech 4). Transition based Mate tools parser is a deterministic shift-reduce parser with beam search and optimizing models to predict sequence of decisions to parse the sentence. This parser also performs joint morphological

[20] https://www.cis.upenn.edu/~treebank/

[21] https://ufal.mff.cuni.cz/pdt3.0

and syntactic analysis. It reached 87.79% labeled attachment score for English and 83.11% for Czech 5). None of these three parsers was evaluated on Slovak language.

2.2. Morphological analyzers

Morphological analysis is inevitable step in preprocessing, as it is needed input for every syntactic parsers (except transition based parser from Mate tools). There are few available taggers which can handle Slovak language:

Slovak POS tagger - knowledge based morphological tagger developed at STU FIIT[22],

Dagger - statistical morphological tagger based on Hidden Markov Models 13) developed at KEMT TUKE[23],

TreeTagger - statistical morphological tagger based on decision trees 14), formerly trained for German language but gradually models for other languages were provided, including Slovak[24],

Mate tools tagger - has two separate taggers - one for determining only part-of-speech and one which then determine rest of morphological features[25].

We evaluated mentioned tools on SDC, the best accuracy (91.59% and 100% coverage) was provided by TreeTagger, which we will use in following experiments.

2.3. Tools for reconstruction of text

During analysis we evaluated two tools for reconstruction of diacritics:

Diakritik - web service for diacritics reconstruction developed at JUĽŠ[26],

KEMT NLP text correction - web service for text reconstruction[27], besides diacritics the service attempted to correct general errors in text 15).

As pointed out in 16), the best accuracy is achieved by Diakritik. KEMT NLP text correction seemed interesting by the time of analysis, because of general error correction, however, this web service became unavailable in time of writing this paper. We instead utilized GNU Aspell[28] library for general spellchecking.

3. Evaluating top-performing parsers

Since there is - to the best of our knowledge - no comprehensive evaluation of performance of existing parsers on Slovak data, we aimed to measure accuracy of parsers based on comparison with gold standard. Slovak Dependency Corpora contains 846 489 tokens which forms 50 275 sentences in data format of Prague Dependency Treebank. None of the parsers work with this format, so we needed to transform data to proper format for each parser (usually one of the CoNLL shared tasks formats). We then split the data to test and train parts and performed training and parsing. We evaluated MaltParser, MST Parser, MATE graph based parser and MATE transition based parser. We chose these parsers because of their public availability, research significance and top accuracy their provide for English and Czech language. Accuracy of parsers measured in standard metrics for evaluation of dependency parsers on Slovak Dependency Corpora are shown in table 1.

Table 1. Scores of parsers evaluated on SDC

	MaltParser	MST Parser	MATE graph based parser	MATE transition based parser
UAS	75,35%	78,24%	82,09%	78,88%

[22] http://morpholyzer.fiit.stuba.sk:8080/PosTagger/
[23] http://nlp.web.tuke.sk/nlpform
[24] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
[25] https://code.google.com/archive/p/mate-tools/
[26] http://diakritik.korpus.sk/
[27] http://nlp.web.tuke.sk/nlpform
[28] http://aspell.net

LAS	57,06%	57,77%	65,28%	59,84%
tUAS	14,70%	29,05%	36,28%	32,27%
tLAS	6,00%	6,79%	8,84%	16,15%

Table 2. Meanings of metrics used for evaluation of syntactic parsers

UAS	unlabeled attachment score - percentage of tokens for which the parent word in sentence was correctly set (HEAD tag of the token)
LAS	labeled attachment score - percentage of tokens for which the parent word in sentence and type of dependency were correctly set (HEAD ag and DEPREL tag of the token)
tUAS	total unlabeled attachment score - percentage of sentences, where every token has correctly set HEAD tag
tLAS	total labeled attachment score - percentage of sentences, where every token has correctly set HEAD and DEPREL tags

We see that the best performing parser was graph based parser from Mate tools. Therefore we use this parser for our next experiments.

In the next experiment, we wanted to see how quality of morphological analysis affects the accuracy of syntactic parsing. We stripped the test data of morphological tags, replaced them with tags acquired by TreeTagger and then we performed syntactic analysis. We witnessed performance drop, when UAS dropped by 1% and LAS by 4%.

Next we were interested in performance influenced by missing diacritics as an example of text defect usually present on the Web. The assumption was that missing diacritics will have significant impact in morphological analysis, which will then affect the syntactic parsing. We removed all accents from test data, performed morphological analysis and then syntactic parsing. Performance drop was even more significant in this case (UAS by 12% and LAS by 16%).

We then attempted to overcome performance difference by employing automatic reconstruction on diacritics. Test data without accents were reconstructed by Diakritik web service, then morphologically analyzed by TreeTagger and then parsed. Accuracy of parsing was restored, as LAS and UAS returned to scores achieved in experiment with our own morphological analysis.

In last experiment we tried to simulate common typographical errors. By analysis of Slovak Web Discussion Corpus [17] we discovered, that approximately 22% of words contains some sort of error. We therefore modified input text for parser by adding random typographical errors at 22% rate. Parsing of this corpus was significantly worse (see table 3). We then tried to recover the accuracy by editing input text with spellchecking provided by Aspell library. The LAS and UAS score increased by 5-6%, however they did not achieve baseline accuracy.

Table 3. Scores of MATE graph based parser achieved in experiments with input

	LAS	UAS	tLAS	tUAS
MATE graph based parser, gold standard (baseline)	65,28%	82,09%	8,84%	36,28%
Morphological analysis by TreeTagger	61,14%	82,15%	8,84%	35,92%
No diacritics, morphological analysis by TreeTagger	49,58%	69,02%	5,87%	21,62%
Reconstructed diacritics, morphological analysis by TreeTagger	61,55%	81,53%	8,54%	34,94%
Added typographical errors, parsed	42,75%	63,69%	3,78%	17,59%
Added typographical errors, spellchecked, parsed	47,39%	68,00%	4,02%	19,28%

4. Improving parser accuracy

We analyzed output of parsers from evaluation and looked at accuracy of determining the correct DEPREL tag. We also analyzed, which DEPREL tag was mistaken for the correct tag, trying to discover common errors. One of the errors with potential to be corrected by a simple rule was AuxV tag incorrectly determined as Atr tag. AuxV belongs specifically to auxiliary verb "*byt*" and it cannot be marked as Atr. We therefore formed a rule:

If lemma of word is "byt" and DEPREL tag is Atr -> wrong DEPREL, correct to AuxV.

By similar method we induced rule for AuxP tag, only it was based on comparing with morphological tag:

If DEPREL is AuxP and morphological tag is not Eu -> wrong DEPREL, correct to AuxP.

We performed evaluation after output of parser was processed by these rules. However performance was not affected positively, it only dropped by few tenth of percent. We therefore think that it would require much more sophisticated rules to overcome the statistical power of machine learning, to the point that focus on this way of improving is questionable.

Second experiment we performed was based on enrichment of input, rather than correcting the output. Simple rule in Slovak grammar is, that sentence usually contains one verb (unless its compound sentence) and that usually is predicate (Pred tag). Every verb in test and training data, which is the only verb in sentence was marked with *pred_candidate* feature. We then trained the parser on small portion of data of 10 000 samples and evaluated on test data. The LAS attachment score was slightly worse (by 0,03%), however UAS score was better (by 0,12%). This leads us to believe, that this kind of enrichment of input is worth investigating. We are now currently experimenting with creating of useful features by means of association rules learning.

5. Conclusion and future work

In this paper we focused on syntactic analysis of Slovak texts, with special attention to problems with real-world input. We presented overview of existing tools for syntactic parsing, morphological analysis and diacritics reconstruction, which all are usable for Slovak language. So far, the main contributions of our work are:

evaluation of top-performing parsers on large texts in Slovak language,

the initial study on parsing accuracy on real-world data, including morphologically-ill inputs,

initial findings resulting from the employment of simple heuristic rules on parsing process.

Our next work will focus on further exploration of influence of heuristic rules applied to enrich the input for the parser on the accuracy of parsers. We believe that with this method, we can improve mainly the labeled attachment score.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG 1/0646/15.

References

1) Nivre, J., McDonald, R.: Integrating Graph-Based and Transition-Based Dependency Parsers. In: *Proc. of ACL-08: HLT*, ACL, (2008), pp. 950–958.

2) Charniak, E.: Statistical techniques for natural language parsing. *AI magazine*, AAAI, (1997), vol. 18, no. 4, pp 33–43.

3) McDonald, R. Pereira, F., Ribarov, K., Hajič, J.: Non-projective dependency parsing using spanning tree algorithms. In: *Proc. of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, (2005), pp. 523–530.

4) Bohnet, B.: Very high accuracy and fast dependency parsing is not a contradiction. In: *Proc. of the 23rd International Conference on Computational Linguistics,* Association for Computational Linguistics, (2010), pp. 89–97.

5) Bohnet, B., Nivre, J.: A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In: *Proc. of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Association for Computational Linguistics, (2012), pp. 1455–1465.

6) Nivre, J., Hall, J., Nilsson, J. Chanev, A., Eryigit, G., Kubler, S., Marinov, S., Marsi, E.: MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, Cambridge University Press, (2007), vol. 13, no. 2, pp. 95–135.

7) De Marneffe, M., MacCartney, B., Manning, C.: Generating typed dependency parses from phrase structure parses. In: *Proc. of LREC*, Stanford, (2006), pp. 449–454.

8) Červeňová, D.: Automated Syntactic Analysis of Natural Language. Diploma thesis at FIIT STU, Bratislava, (2015).

9) Petrov, S., McDonald, R.: Overview of the 2012 shared task on parsing the web. In: *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, Google, (2012), vol. 59.

10) Ondáš, S., Juhár, J., Čižmár, A.: Extracting sentence elements for the natural language understanding based on slovak national corpus. In: *Analysis of Verbal and Nonverbal Communication and Enactment. The Processing Issues*. Springer Berlin Heidelberg, (2011), pp. 171–177.

11) Trabalka, M., Bieliková, M.: Using XML and Regular Expressions in the Syntactic Analysis of Inflectional Language. In: *ADBIS-DASFAA Symposium*, (2000), pp. 185–194.

12) Medved M., Jakubícek M., Kovár, V., Nemcík, V.: Adaptation of Czech Parsers for Slovak. In: *Proc.of Recent Advances in Slavonic Natural Language Processing*, RASLAN, (2012), pp. 23–30.

13) Hládek, D., Staš, J., Juhár, J.: Dagger: The Slovak morphological classifier. In: *ELMAR, 2012 Proceedings*. IEEE, (2012), pp. 195–198.

14) Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: *Proc. of the international conference on new methods in language processing*, Universität Stuttgart, (1994), pp. 44–49.

15) Hladek, D., Staš, J., Juhár, J.: Unsupervised spelling correction for Slovak. *Information and Communication Technologies and Services,* AEEE, (2013), vol. 11, no. 5, pp. 392-297.

16) Gedera, J.: Automatic Diacritics Reconstruction in Slovak Texts. *IIT.SRC: 2015: Student Research Conerence in Informatics and Information Technologies*, STU Bratislava, (2015).

17) Hládek, J., Staš, J., Juhár, J.: Slovak Web Discussion Corpus, *Advances in Natural Language Processing*, 9th International Conference on NLP, PoITAL 2014, Warsaw, Poland, (2014).

Príloha G: článok poslaný na HrTAL konferenciu

Syntactic Analysis in Real-World Corpora

Anonymous

Author information purposely omitted due to submission requirements of the conference.

Abstract. Syntactic analysis in terms of natural language processing (NLP) is an important step in deeper understanding of text. Extracted dependencies between words can be profitable for many other tasks in NLP, i.e. machine translation, semantic analysis, or information extraction. Throughout last two decades we witness advances in quality of syntactic parsing, mainly due to statistical and machine learning methods, which gained popularity with growing processing power of computers. It is no surprise that very little of this research was done specifically for Slovak language, which was caused by the absence of proper syntactically annotated dataset. Thankfully, this is slowly changing with existence of Slovak Dependency Treebank (SDT). In this paper we focus on comprehensive evaluation of existing parsers on this dataset, while aiming at specific issues related to Slovak language and real-world data.

Keywords: natural language processing • syntactic parsing • Slovak language

Introduction

Syntactic analysis of a sentence can be simply understood as extracting dependency structure from input sentence. Thanks to this structure we know which word depends on another and type of this relation. These are quite useful information for many other tasks from the field of NLP [1], which are an integral part of real-world applications.

Data-driven parsing became main approach, thanks to available processing power and available data resources. Corpora for English[29] and many other languages, including Czech[30] (considered being the closest language to Slovak), have been available for many years. Creation of Slovak Dependency Treebank (SDT) in recent years made data-driven parsing research finally available also for Slovak language [2]. Like its foreign counterparts, it consists of a collection of corrected and rather errorless texts. Current state-of-the art parsers have been evaluated mainly on those edited texts, achieving accuracies well above 90%. However, it has been shown that parsing accuracies are barely over 80% when used in practice [3].

When parsing texts acquired from the wild Web (blogs, forums, discussions or other user-created content), we have to cope with the fact that most of these texts are unedited and tend to contain spelling or grammar mistakes or no punctuation. This is an open problem in the field of syntactic analysis and research communities have put in only a little attention.

The long term goal of our research is improvement of syntactic analysis for Slovak language, particularly when considering real-world text parsing. The main aim of this paper is to examine syntactic analysis of real-world data and recognize its major obstacles causing reduction of parsing accuracy. We evaluate existing top-performing parsers on Slovak texts and report on experiments with different setups for real-world text simulation of errors. Very little is known about specificities of parsing Slovak language. We identify the most significant problems with quality of text input and try to overcome these problems with use of available tools for text reconstruction.

[29] https://www.cis.upenn.edu/~treebank/

[30] https://ufal.mff.cuni.cz/pdt3.0

Related work

Development and research of data-driven parsers for Slovak language was held back by absence of syntactically annotated dataset. Thus research focused mainly on rule based systems, such as parsing context free grammar [4], bottom-up knowledge based parsing using regular expressions and xml based rules [5], or adaptation of Czech knowledge based parsers on Slovak language [6]. With recent development of SDT, the first aims of employing MaltParser emerged [7]. MaltParser is an implementation of inductive dependency parsing, it was comprehensively evaluated on 10 languages, including Czech and English, giving a dependency accuracy of 80–90% [8]. Also, it is the only parser, which has also been evaluated in Slovak language, achieving 71.72% labeled attachment score (LAS) and 77.81% unlabeled attachment score (UAS) [7]. As we focus on application of existing parsers to Slovak language, we further examine another three parsers: Maximum Spanning Tree parser (MST parser) [9], transition based parser from Mate tools [10] and graph based parser from Mate tools [11].

MST parser formalizes parsing as a search for maximum spanning tree. First version used modified Eisner algorithm, which was later substituted by Chu-Liu-Edmonds algorithm [9]. Though it was not at first designed to parse non-projective structures, it is capable of such parsing. It achieved 90.20% accuracy (number of words which correctly identified their parent in sentence) for English and 84.40% accuracy for Czech.

Graph based Mate tools parser is based on MST parsers, modified to speed up training and parsing. It reached 90.33% labeled attachment score for English and 80.96% for Czech [11]. Transition based Mate tools parser is a deterministic shift-reduce parser with beam search and optimizing models to predict sequence of decisions to parse the sentence. This parser also performs joint morphological and syntactic analysis. It reached 87.79% labeled attachment score for English and 83.11% for Czech [10]. None of these three parsers was evaluated on Slovak language.

Comparison of top-performing parsers

Since there is – to the best of our knowledge – no comprehensive evaluation of performance of existing parsers on Slovak data, we aimed to measure accuracy of parsers based on comparison with the gold standard. SDT contains 846 489 tokens which forms 50 275 sentences in data format of Prague Dependency Treebank [2]. First we transformed data to proper format for each parser (usually one of the CoNLL shared tasks formats). We then split the data to test and train parts and performed training and parsing. We evaluated MaltParser, MST Parser, MATE graph based parser and MATE transition based parser. We chose these parsers because of their public availability, research significance and top accuracy they provide for English and Czech language. Accuracy of parsers measured in standard metrics for evaluation of dependency parsers on Slovak Dependency Treebank are shown in Table 1.

Table 1. Scores of parsers evaluated on SDT.

	MaltParser	MST Parser	MATE graph based	MATE transition based
UAS	75.35%	78.24%	82.09%	78.88%
LAS	57.06%	57.77%	65.28%	59.84%
tUAS	14.70%	29.05%	36.28%	32.27%
tLAS	6.00%	6.79%	8.84%	16.15%

We see that the best performing parser was graph based parser from Mate tools. We use this parser for our next experiments aiming at simulating parsing real-world text, mainly user-generated text originating on the web. Such texts incline to have various errors and defects, not usually occurring in edited texts available in treebanks.

Simulating real-world text parsing

We designed number of experiment setups to simulate various text imperfections, commonly occurring in real-world environment, e.g. in the web text data. They include automatic creation of morphological annotation as a part of input for parsing. Here we provide a brief description of performed experiments and introduce particular setups.

Custom morphological annotation. First, we wanted to see how quality of morphological analysis affects the accuracy of syntactic parsing. There are few available taggers which can handle Slovak language:

Slovak POS tagger – knowledge based morphological tagger developed at STU FIIT[31], which was based on data provided by Slovak National Corpus [12],

Dagger – statistical morphological tagger based on Hidden Markov Models [13] developed at KEMT TUKE[32],

TreeTagger – statistical morphological tagger based on decision trees [14], formerly trained for German language but gradually models for other languages were provided, including Slovak[33],

Mate tools tagger – has two separate taggers: one for determining only part-of-speech and one that determines the rest of morphological features[34].

We evaluated the mentioned tools on SDT, the best accuracy (91.59% and 100% coverage) was provided by TreeTagger, which we will use in following experiments. We replaced the gold morphological annotation with the one provided by TreeTagger and then performed parsing (Setup 1, see Table 2).

Missing diacritics. Next, we were interested in evaluating performance influenced by missing diacritics as an example of text defect usually present on the Web. The assumption was that missing diacritics will have significant impact in morphological analysis, which will then affect the syntactic parsing. We removed all accents from test data, performed morphological analysis and then syntactic parsing (Setup 2, see Table 2). We then attempted to overcome performance difference by employing automatic reconstruction on diacritics (Setup 3, see Table 2). Test data without accents were reconstructed by Diakritik web service[35], then morphologically analyzed by TreeTagger and then parsed.

Typographical errors. By analysing Slovak Web Discussion Corpus [15] we discovered, that approximately 22% of words contain some sort of error. We therefore modified input text for parser by adding random typographical errors at the observed rate. Parsing this corpus was significantly worse (Setup 4, see Table 2). We then tried to recover the accuracy by editing input text with spellchecking provided by Aspell library – the state-of-the-art spell-checker available for Slovak language (Setup 5, see Table 2).

Grammar mistakes. In our last setup, we also aimed to simulate common grammar mistakes that are less random than typographical errors. We implemented generation of three kinds of grammar errors, most often occurring in (and specific to) Slovak language (1) interchange of *i* and *y*, (2) addition or omission of accent on letter, and (3) interchange of letters due to phonological assimilation. We generated these errors in input text with same rate as typographical errors (Setup 6, see Table 2). We then again used Aspell library to recover the accuracy (Setup 7, see Table 2).

Overview of results of all experiments is shown in Table 2. To better outline the settings of every experiment, we structured the table as follows: D stands for diacritics, TY stands for typographical errors and G stands for grammar mistakes. MA stands for custom morphological analysis by TreeTagger.

[31] http://morpholyzer.fiit.stuba.sk:8080/PosTagger/
[32] http://nlp.web.tuke.sk/nlpform
[33] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
[34] https://code.google.com/archive/p/mate-tools/
[35] http://diakritik.korpus.sk/

Each of these columns contains number indicating what kind of action for the particular section was taken:

0 – no action was taken,

1 – deformation was applied (removed diacritics for D and added errors for TY and G) or custom morphology annotation for MA was provided

2 – deformation was applied and then a tool to repair input was used (reconstruction of diacritics for D and Aspell correction for TY and G).

Table 2. Scores of MATE graph based parser achieved in experiments with input. tLAS is stands for total labeled attachment score and tUAS for total unlabeled attachment score.

	D	TY	G	MA	LAS	UAS	tLAS	tUAS
Baseline (ideal input)	0	0	0	0	65,28%	82,09%	8,84%	36,28%
Setup 1	0	0	0	1	61,14%	82,15%	8,84%	35,92%
Setup 2	1	0	0	1	49,81%	69,05%	6,01%	21,27%
Setup 3	2	0	0	1	62,64%	82,21%	8,89%	36,70%
Setup 4	0	1	0	1	53,77%	73,07%	6,66%	23,99%
Setup 5	0	2	0	1	56,92%	76,73%	7,08%	27,59%
Setup 6	0	0	1	1	54,33%	73,96%	7,00%	25,84%
Setup 7	0	0	2	1	56,97%	76,84%	7,22%	28,41%

We see that tools used for text correction were able to improve the quality of text to the point, it had positive impact on parsing accuracy. The most significant improvement was observed on diacritic reconstruction, which slightly overcome the performance of original text with TreeTagger morphological annotation. Despite substantial improvement in parsing accuracy after application of existing tools, the results suggest that we need to seek additional ways how to further improve performance of tools employed for real-world text processing in order to approach the baseline parsing accuracy.

Conclusion and future work

In this paper we focused on syntactic analysis of Slovak texts, with special attention to problems with real-world input. We presented overview of existing tools for syntactic parsing, morphological analysis and diacritics reconstruction, which all are usable for Slovak language. So far, the main contributions of our work are:

evaluation of top-performing parsers on large texts in Slovak language,

the initial study on parsing accuracy on real-world data, including morphologically-ill inputs.

The experiment showed that substantial attention should be drawn to quality of input, since when unedited imperfect real-world input is provided, parsing accuracy decreases. Methods and techniques for text reconstruction in Slovak language should be further improved to better support process of real-world data parsing. Besides, novel ways to parsing accuracy even for non-morphologically-ill inputs should be devised by addressing the specifics of Slovak language. One such direction we see is focusing on improvement of specific, more important deprel tags computation.

Research of syntactic parsing of Slovak language is only at the beginning. There is plenty of other possibilities to catch up the state-of-the-art parsing techniques, and many possible ways of research, including deep learning, or extending the SDT and creating new data resources for purpose of syntactic analysis.

References

Nivre, J., McDonald, R.: Integrating Graph-Based and Transition-Based Dependency Parsers. In: Proc. of ACL-08, pp. 950–958, ACL (2008)

Gajdošová, K.: Syntactic analysis of selected text in Slovak national corpus (In Slovak). In: VARIA XVI, Young linquists colloquia, pp. 140-148, Slovak linguistic society, (2009)

Petrov, S., McDonald, R.: Overview of the 2012 shared task on parsing the web. In: Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL), vol. 59 (2012)
Ondáš, S., Juhár, J., Čižmár, A.: Extracting sentence elements for the natural language understanding based on slovak national corpus. In: Analysis of Verbal and Nonverbal Communication and Enactment. The Processing Issues, pp. 171–177, Springer (2011)
Trabalka, M., Bieliková, M.: Using XML and Regular Expressions in the Syntactic Analysis of Inflectional Language. In: ADBIS-DASFAA Symposium, pp. 185–194 (2000)
Medved, M., Jakubícek, M., Kovár, V., Nemcík, V.: Adaptation of Czech Parsers for Slovak. In: Proc. of Recent Advances in Slavonic Natural Language Processing, pp. 23–30, RASLAN (2012)
Červeňová, D.: Automated Syntactic Analysis of Natural Language. Diploma thesis at FIIT STU, Bratislava (2015)
Nivre, J., Hall, J., Nilsson, J. Chanev, A., Eryigit, G., Kubler, S., Marinov, S., Marsi, E.: MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, Cambridge University Press, 13.02, 95–135 (2007)
McDonald, R. Pereira, F., Ribarov, K., Hajič, J.: Non-projective dependency parsing using spanning tree algorithms. In: Proc. of the conf. on Human Language Technology and Empirical Methods in Natural Language Processing, pp. 523–530, ACL (2005)
Bohnet, B., Nivre, J.: A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In: Proc. of the 2012 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 1455–1465, ACL (2012)
Bohnet, B.: Very high accuracy and fast dependency parsing is not a contradiction. In: Proc. of the 23rd Int. Conf. on Computational Linguistics, pp. 89–97, ACL (2010)
Horák, A., Gianitsová, L., Šimková, M., Šmotlák, M., Garabík, R.: Slovak national corpus. In: Text, Speech and Dialogue, pp. 89-93, Springer Berlin Heidelberg (2004)
Hládek, D., Staš, J., Juhár, J.: Dagger: The Slovak morphological classifier. In: ELMAR, 2012 Proceedings, pp. 195–198, IEEE (2012)
Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: Proc. of the int. conf. on new methods in language processing, pp. 44–49, Universität Stuttgart, (1994)
Hládek, J., Staš, J., Juhár, J.: Slovak Web Discussion Corpus, Advances in Natural Language Processing, 9th Int. Conf. on NLP, PoITAL 2014, Warsaw, Poland, (2014)