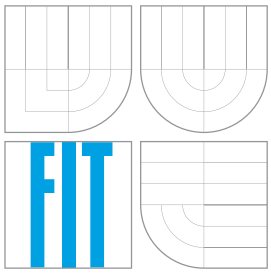
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

JEDNODUCHÝ GRAMATICKÝ KOREKTOR PRO ČEŠTINU DO OPENOFFICE

SIMPLE CZECH GRAMMAR CORRECTTOR FOR OPENOFFICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JOZEF LIČKO

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2007

Zadání bakalářské práce/5723/2006/xlicko01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Ličko Jozef**
Obor: Informační technologie
Téma: **Jednoduchý gramatický korektor pro češtinu do OpenOffice**
Kategorie: Softwarové inženýrství

Pokyny:

1. Cílem projektu je navrhnout a implementovat systém, který bude na základě jednoduchých pravidel kontrolovat některé gramatické jevy (např. psaní čárek za vedlejší větou vztažnou, shoda uvnitř jmenných skupin). Úkolem bude rovněž integrovat systém do poslední verze OpenOffice.

Literatura:

- Dokumentace k OpenOffice a korektorům pro češtinu

Při obhajobě semestrální části projektu je požadováno:

1. Funkční prototyp

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

Licenční smlouva

Licenčni smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Gramatika prirodzeného jazyka je komplexná a často ju nie je možné priamo, jednoznačne a formálne popísať. Existuje niekoľko prístupov, ako sa pokúsiť pomocou výpočtovej techniky o odhalenie a opravu gramatických chýb. Táto práca sa zaoberá vytvorením jednoduchého gramatického korektora, ktorý funguje na princípe pravidiel. Zameriava sa na neprítomnosť čiarok v súvetiach, v ktorých sa nachádzajú vedľajšie vety vzťažné. Keď takú časť vety korektor úspešne rozpozná, navrhne vhodné riešenie, ako je možné danú vetu opraviť. Aplikácia funguje ako prídavný modul do kancelárskeho balíka OpenOffice.

Klíčová slova

gramatický korektor, čeština, gramatika, vzťažné vety, čiarky, kontrola čiarok

Abstract

The grammar of natural language is generally complex. Often, it is very difficult to describe it precisely in a formal way. There are several approaches to grammar correction using computer technologies. This thesis deals with the construction of a simple rule-based Czech grammar checker. It focuses on subordinate relative constructions. The corrector is designed to identify missing comma in them. If such a sentence is found the program proposes possible suggestions how to correct it. The application operates as a plugin for OpenOffice suite.

Keywords

grammar checker, grammar corrector, grammar, czech language, relative sentences, commas, correction of commas

Citace

Jozef Ličko: Jednoduchý gramatický korektor pro češtinu do OpenOffice, bakalářská práce, Brno, FIT VUT v Brně, 2007

Jednoduchý gramatický korektor pro češtinu do OpenOffice

Prohlášení

Vyhlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením doc. RNDr. Pavla Smrže, Ph.D.

.....
Jozef Ličko
13. května 2007

Obsah

Kapitola 1

Úvod

1.1 Vymedzenie pojmov

Pojmy *kontrola gramatiky* a *kontrola pravopisu* sa v slovenčine často zamieňajú. Slovo *gramatika* primerane zodpovedá anglickému slovu *grammar*.

Pri slove *pravopis* je situácia menej jasná. Samotné slovo nie je problematické. Nejasnosť nastáva, keď je potrebné nájsť vhodný ekvivalent pre anglické pomenovanie *spellchecker* alebo *spelling*. Pri preklade týchto slov sa často používa spomenuté slovo *pravopis*. Aj keď má v slovenčine (a češtine) širší význam.

V mojej práci budem používať slovo *pravopis* a pojem *korektor pravopisu* na označenie anglického pojmu *spellchecker*, ktorý je pevne zaužívaný v oblasti informačných technológií ako označenie pre program, ktorý kontroluje správne poradie a výskyt znakov v slovách daného prirodzeného jazyka.

Výraz *korektor gramatiky* budem používať na označenie anglického *grammar checker*. Pod týmto pojmom mám na mysli nástroj, ktorý ide za hranice jedného slova, napríklad tým, že analyzuje celé vety. Kontroluje gramatickú správnosť textu, alebo naopak jej nedostatok v ňom.[10]

Prostriedkom korekcie gramatiky je pokročilejšia analýza textu. Do určitej miery sa snaží lingvisticky analyzovať vetu a rozhodnúť, či ju možno považovať za správnu alebo nie.

1.2 Postup riešenia

Pri hľadaní vhodného riešenia bolo potrebné zvoliť vhodný programovací jazyk pre implementáciu aplikácie, prípadne zistiť, či podobná aplikácia už nebola vytvorená.

Integrácia korektora do vhodného textového editora si kladie za cieľ uľahčiť jeho používanie bežným užívateľom. Podľa zadania bolo potrebné použiť kancelársky balík *OpenOffice*[1] (V čase riešenia projektu bola dostupná verzia 2.2). Pojednanie o výhodách a nevýhodách zvoleného riešenia je možné nájsť v podkapitole 3.1.

Samotné vytvorenie kvalitného rozširujúceho modulu do OpenOffice by bola časovo náročná úloha. Preto som venoval určitý čas hľadaniu vhodného a funkčného riešenia, ktoré by moju prácu uľahčilo, aby som sa mohol venovať priamo zadaniu práce. Tento postup sa nakoniec osvedčil. Gramatický korektor *LanguageTool*[2] je nástroj, ktorý podporuje kon-

[1] http://www.openoffice.org

[2] http://www.danielnaber.de/languagetool

trolu gramatiky viacerých jazykov. Dôležité pre moju prácu bolo to, že už jeden slovanský jazyk (poľštinu) podporoval. Bolo ho taktiež možné použiť ako prídavný modul do balíka OpenOffice. Jeho popisu je venovaná podkapitola 3.2.

Tu som sa mohol začať venovať rozšíreniu LanguageTool tak, aby podporoval češtinu. Môj prínos a popis samostatnej práce sa nachádza v časti II. Kapitola 4 obsahuje popis rozšírenia LanguageTool. Bolo potrebné vytvoriť potrebné morfologické dáta pre češtinu, implementovať zjednoznačňovaciu časť korektora a definovať pravidlá na kontrolu interpunkcie.

Už počas rozširovania LanguageTool vyvstala potreba vhodných testovacích dát. Potreboval som zhromaždiť dostatočne veľké množstvo gramaticky správnych viet, ktoré obsahovali vzťažné vety. Problematike získania kvalitných testovacích dát a samotnému testovaniu je venovaná kapitola 5. Je tu tiež vyhodnotená úspešnosť implementovaného riešenia.

V záverečnej 6. kapitole sú zhrnuté výsledky práce. Uvádzam tu aj získané skúsenosti a ďalší možný vývoj gramatického korektora pre český jazyk.

Část I

Teoretický základ

Kapitola 2

Podklady pre kontrolu gramatiky

Písaný text je celok, ktorý pozostáva z jednotlivých častí. Jeho jednotlivé časti sa dajú deliť na ďalšie menšie časti. Text je preto možné považovať za hierarchickú štruktúru:

- Daný text sa obyčajne skladá z odstavcov. Každý odstavec by mal sprostredkovať jednu úplnú myšlienku.
- Odstavec sa skladá z viet. Veta je obyčajne definovaná tak, že sa v nej musí nachádzať sloveso. (Existuje niekoľko prípadov, keď to nemusí platiť.)
- Vetu možno rozdeliť na ďalšie celky – slová. Tie reprezentujú jednotlivé významové jednotky.
- Slová je možné deliť ešte ďalej na slabiky a hlásky.

Je možné všeobecne tvrdiť, že prirodzený jazyk je značne komplikovaný. Samotná komplikovanosť nespôsobuje výpočtovej technike problém v spracovaní jazyka. Obtiaže sú spôsobené tým, že je problematické vytvoriť úplný formálny popis jazyka, a tým zabezpečiť úplnú a presnú kontrolu gramatiky.

2.1 Stav problematiky v súčastnosti

Pri kontrole pravopisu a gramatiky je podstatným faktorom jazyk, v ktorom je text napísaný. Rôzne jazyky majú rôzne vlastnosti a na to je potrebné myslieť pri implementácii spomenutých nástrojov.

Napríklad angličtina je charakteristická pevnou stavbou vety a charakteristickými slovnými spojeniami. Naopak v českom jazyku sa poradie vetných členov mení, čeština patrí do kategórie jazykov *flexívnych*, čo je podmnožina jazykov afigujúcich – používajúcich afixy (prípony)[12], z čoho vyplýva bohaté tvaroslovie. To naznačuje, že ku každému zo spomenutých jazykov je potrebné zaujať iný prístup.

Súčasné *korektory pravopisu* pracujú na slovnej úrovni. Majú k dispozícii vhodne spracovanú databázu korektných slov. S ňou porovnávajú kontrolovaný text. Ak sa slovo v texte nenájde v databáze, korektor ho označí za nesprávne. Ďalej je bežné, že korektor ponúkne niekoľko možností ako jednoduchou zmenou aktuálneho slova (typicky zmenou malého počtu písmen) už slovo známe bude.

Korektor gramatiky si kladie za cieľ určiť správnosť (resp. nesprávnosť) textu po gramatickej stránke. Aby to bolo možné, je nutné vziať do úvahy väčšie celky textu. Týmito

celkami môžu byť v najjednoduchšej forme slovné spojenia. Obyčajne však gramatický korektor pracuje na úrovni viet.

Pre úplnosť ešte uvediem, že existuje ešte vyššia úroveň. A to úroveň sémantická. Na tejto úrovni je dôležitý význam napísanej, prípadne inak predanej informácie. Nástroj pracujúci na sémantickej úrovni sa nazýva *štylistický korektor*.

Dva najznámejšie korektory českej gramatiky sú Lingea Grammaticon a Kontrola české gramatiky pre MS Office. *Lingea Grammaticon* pracuje na všetkých spomenutých úrovniach - snaží sa kontrolovať pravopisné, gramatické a aj štylistické chyby. Je možné ho používať ako rozšírenie MS Office, alebo ako samostatnú aplikáciu. Korektor sa dostal na trh v roku 2003 a jeho vývoj stále pokračuje.[2]

Spoločnosť Microsoft vyvinula v spolupráci s českými odborníkmi nástroj *Kontrola české gramatiky*, ktorý je možné nainštalovať ako doplnok MS Office 2003.[1] Nástroj sa taktiež snaží kontrolovať gramatiku a štýl. (Kontrola pravopisných chýb bola v MS Office integrovaná už v predošlých verziách).[1]

V súvislosti so špecifickými vlastnosťami jazyka, konkrétne češtiny, je potrebné spomenúť nasledujúcu skutočnosť. Pri snahe o korekciu gramatiky sa často stáva, že syntaktická znalosť textu je nedostatočná. Na správne rozhodnutie je potrebné poznať význam napísaného. Určitý text môže byť napísaný správne, čo sa gramatických pravidiel týka. V skutočnosti však čitateľovi nemusí dávať žiadny zmysel. Uvediem príklad z [4]

Ráno vyděl Věru.

Na prvý pohľad sa v nej vyskytuje gramatická chyba v slove „vyděl“. Je tomu tak, pokiaľ má autor vety na mysli slovo „vidět“. V skutočnosti je však možné slovo „vyděl“ interpretovať ako rozkazovací spôsob od slova „vydělit“. V tom prípade veta po gramatickej stránke chybu neobsahuje. Uvedená veta je typickým príkladom, kde je potrebná znalosť významu, aby bolo možné určiť jej správnosť. Potom je zrejmé, že „Věra“ je meno osoby, ktoré nemá zmysel „vydělit“; „vidět Věru“ však zmysel má.

V tejto časti som sa snažil stručne spomenúť oblasti, ktorými sa treba zaoberať pri tvorbe gramatického korektora. Všeobecne možno poznamenať: Aby mohol gramatický korektor fungovať čo najpresnejšie

1. Potrebuje rozdeliť kontrolovaný text na vhodné celky.
2. Musí vykonať kvalitnú morfologickú analýzu vstupného textu. Inými slovami, mal by vedieť správne rozpoznať čo najviac slovných tvarov.
3. A na koniec by mal na základe získaných dát rozhodnúť, či je analyzovaná časť textu gramaticky správna alebo nie je.

Ďalej nasleduje podrobnejší popis jednotlivých súčastí gramatického korektora.

2.2 Vhodné rozdelenie vstupného textu

Pre potreby gramatického korektora je nutné vstupný text vhodne rozdeliť. Keďže veta býva ukončená jedným z charakteristických interpunkčných znamienok, je logické riadiť sa práve podľa nich pri rozdeľovaní textu na vety. Aj tak sa spomenutý postup musí vysporiadať s nasledujúcimi problémami:

[1] Korektor je dostupný na stránkach firmy Microsoft. Najnovšia verzia kancelárskeho balíka by mala kontrolu gramatiky obsahovať štandardne.

- Najčastejšie znamienko používané na ukončenie vety „.“ (bodka) sa používa aj na iné účely. Konkrétne sa jedná o skratky, zápis dátumov. Ďalej treba vziať do úvahy znamienko pokračovania „...“.

- Vetu nie je potrebné ukončiť ani po znamienkach „!“ a „?“, keď sa jedná o vety s priamou rečou. Napríklad

  „Jeď nejrychleji jak můžeš!“, křičela.

Tieto výnimky je potrebné pri rozdeľovaní textu na jednotlivé vety brať do úvahy. Analýza okolia znakov, ktoré ukončujú vetu, by mala dostatočne poslúžiť na správne rozoznanie konca danej vety.

2.3 Morfologické značkovanie

Je proces pri ktorom sú slovám pridelené slovné druhy a ďalšie gramatické kategórie, ktoré možno pri jednotlivých slovných druhoch určiť. Zvyčajne sa tieto údaje reprezentujú morfologickými značkami.

Potrebný text je možné označkovať automaticky. Nástroj, ktorý je na to určený, sa nazýva *morfologický analyzátor*. Ten využíva určité znalosti daného jazyka a na ich základe priradí slovám v texte adekvátne značky.

Môže sa stať, že analyzátor určité slovo nerozpozná. Taká situácia nastane obyčajne vtedy, keď sa slovo vymyká pravidlám, podľa ktorých analyzátor funguje. Alebo ak sa jedná o cudzie slovo. Ak danému slovu nie je možné priradiť žiadnu značku, existujú dve možnosti ako vzniknutú situáciu vyriešiť:

1. Analyzátor slovu priradí špeciálnu „prázdnu“ značku. Tá plní jedinú funkciu: Informuje nástroje, ktoré budú označkovaný text ďalej spracovávať, že parametre daného slova nie sú známe.

2. Analyzátor sa pokúsi slovu značku priradiť. To je možné vykonať buď na základe vlastností samotného slova, alebo na základe jeho okolia. Kombinácia obidvoch metód prináša najlepšie výsledky.

Reálne morfologické analyzátory dosahujú takmer úplné pokrytie tvarov v danom jazyku. Úplné pokrytie nie je možné, pretože jazyk sa postupne mení. Napríklad prenikajú do neho slová s cudzích jazykov. Ďalej nie je možné pokryť všetky vlastné mená a slová od nich utvorené.

Často nastane práve opačná situácia, keď slovo v danom tvare možno interpretovať viacerými spôsobmi. Stáva sa to preto, lebo slová v prirodzenom jazyku nie sú významovo jednoznačné. Napríklad vo vete

Ten zmatek se děl kvůli vystřelení z děl v blízkosti galerii významných děl.

má slovo *děl* tri rôzne významy:

- Sloveso *dít se*
- Podstatné meno *dělo*
- Podstatné meno *dílo*

Existuje viac takýchto prípadov, no tie sú veľmi zriedkavé v porovnaní so situáciami, ktoré sa vyskytujú systematicky. To sa stáva preto, lebo v češtine existuje mnoho gramatických kategórií, do ktorých možno dané slovo zaradiť. Napríklad mužské neživotné podstatné mená majú v nominatíve a akuzatíve rovnaký tvar.

Ak je slovo morfologicky nejednoznačné, nie je možné na základe samotného slova vybrať správnu značku zo všetkých možných. Na rozhodovanie je potrebné využiť ďalšie informácie. Exitujú rôzne prístupy:

1. Metódy založené na pravidlách. Cieľom je zostaviť množinu pravidiel tak, aby na základe kontextu daného slova bolo možné určiť, ktoré značky sú (resp. nie sú) prijateľné. Pravidlá sú vytvárané buď ručne, alebo automaticky využitím textového jazykového korpusu[2].

   Výhodou ručného vytvárania pravidiel je možnosť vyjadrenia skutočných gramatických zákonitostí jazyka. Pretože je známe, že niektoré zoskupenia slovných druhov sú nesprávne. Napríklad sloveso nemôže nasledovať hneď po predložke.

   Naopak použitím automatizácie je človek odľahčený od vymýšľania pravidiel. Nutnou podmienkou tu je systém, ktorý dokáže pravidlá generovať. Napríklad analýzou jazykového korpusu. Tento postup je blízky nasledujúcemu typu metód.

2. Štatistické metódy. Principiálne vyberú takú značku pre dané slovo, ktorá sa v danom kontexte vyskytuje najčastejšie. Z toho vyplýva, že tieto metódy taktiež potrebujú dodatočné informácie. V najjednoduchšom prípade to môže byť percentuálne rozdelenie skutočného použitia značiek daného slova.

   V praxi sa často používajú pokročilejšie rozhodovacie techniky, napríklad *Skryté Markovove modely* (HMM z angl. *Hidden Markov models*) spolu s *Viterbiho algoritmom*. Tieto postupy vyžadujú tréning na správne označkovaných modelových dátach. Vo fáze trénovania je vytvorená databáza slov a ich vzájomných prechodov. Každé slovo a jeho prechod k ďalšiemu je ohodnotený určitou pravdepodobnosťou.

   Keď sú použité na zjednoznačnenie neoznačkovaného textu, ich princíp spočíva v predpoklade, že správna značka daného slova existuje, ale nie je známa. Spracúvajú jednotlivé slová v texte, ich kontext a porovnávajú ich so svojou databázou. Na záver zvolia riešenie s najväčšou pravdepodobnosťou.

Ešte pred použitím určitej metódy je vhodné zvážiť jej použiteľnosť na daný jazyk. Napríklad angličtina nie je jazyk s bohatou morfológiou a často sa v nej vyskytujú ustálené frázy. Nie je preto prekvapujúce, že použitím štatistických metód tu možno dosiahnuť presnosť až 97 %[3].[3]

V češtine je situácia zložitejšia. Hlavné dôvody sú: Bohatá morfológia, sloveso v češtine môže napríklad nadobúdať až 50 rôznych tvarov.[5] Ďalej väčšie množstvo gramatických kategórií (čo má za následok väčšie množstvo unikátnych morfologických značiek) a volné poradie slov vo vete. Experimenty ukazujú, že štatisticke metódy sú použiteľné, ale ich úspešnosť je nižšia ako v angličtine.[3]

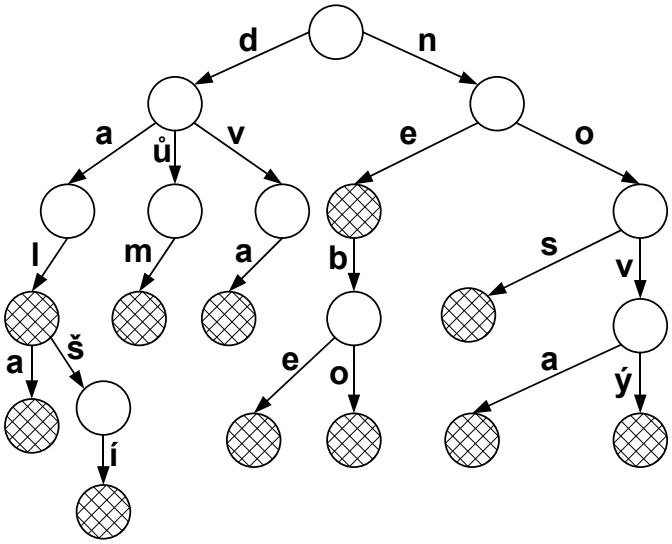
[2] Rozsiahly súbor vzorových textov, ktoré sú podľa možností čo najsprávnejšie označené metajazykovými značkami, aby poslúžili ako príklad správneho značkovania.

[3] Výsledky sú závislé na použitom pravdepodobnostnom modeli: unigram 85,5 %, bigram 93 % až 97 %, trigram 97,14 %

2.4 Vhodné dátové štruktúry

Možno konštatovať, že kvalitná morfologická analýza češtiny vyžaduje rozsiahlu a značne objemnú databázu slov. Dáta z morfologického analyzátoru *AJKA*[8], ktoré boli použité v tejto práci, obsahujú 6 miliónov slovných tvarov (80 MB v textovej podobe). Keď sa ku každému tvaru priradia základný tvar (*lemma*) a morfologické značky, veľkosť výsledných dát vzrastie na 1,2 GB (120 MB po kompresii programom *gzip*).

Je nevyhnutné tieto dáta reprezentovať v kompaktnom formáte o veľkosti jednotiek megabajtov. Ďalšia dôležitá požiadavka sa týka rýchlosti prístupu k dátam. Úlohou tejto časti je nájsť riešenie na uvedené požiadavky tým, že tu budú predstavené vhodné spôsoby a postupy reprezentácie lingvistických údajov.



Obrázek 2.1: Model *trie* uchovávajúci české slová dal, dala, další, dům, dva, ne, nebe, nebo, nos, nova, nový

Často používanou štruktúrou je *trie*[4]. V pamäti je realizovaná ako zoradený N-árny strom ako možno vidieť na obrázku 2.1. Prechod od nadradeného uzlu k podradenému je definovaný znakom. Každý znak je vlastne písmeno zo slova, ktoré je v danej štruktúre uložené. Týmto sú dosiahnuté dve hlavné výhody štruktúry *trie*:

1. Efektívne uloženie slovníkových dát. Znižuje redundanciu tým, že slová obsiahnuté v dátovej štruktúre zdieľajú spoločnú predponu (*prefix*).

2. Rýchle vyhľadávanie. Všeobecne sa udáva $O(l)$, kde l je počet znakov hľadaného slova.

2.5 Optimalizácia údajov

Štruktúra *trie* je vhodná na spomenuté účely. To ale neznamená, že je najlepšia, alebo maximálne efektívna. Existujú nasledujúce nevýhody[11]:

[4] Jej názov je odvodený z anglického *re*trie*val*. Čo sa dá preložiť ako *získavanie*, *výber*, alebo *prehľadávanie*. Výslovnosť je blízka slovu *tree*, čo naznačuje ako je dátová štruktúra reprezentovaná v pamäti.

- Efektivita môže klesnúť, pri ukladaní nevhodných dát. Toto sa týka napríklad ukladania dlhých slov, ktorých predpony sa často líšia. Alebo pri ukladaní malého počtu veľmi dlhých slov.

  Takto by mohlo dôjsť až k extrémnemu prípadu, keď nevhodne zvolené dáta nebudú obsahovať žiadnu spoločnú predponu. Vtedy by sa z *trie* v podstate stal súbor lineárnych zoznamov.

- V základnej forme štruktúra zabezpečuje, aby slová zdieľali predpony, nerieši však ako efektívne uložiť zhodné prípony.

- Algoritmy pracujúce s *trie* sú zložitejšie ako algoritmy pracujúce s *binárnym vyhľadávacím stromom.*

- Nie je jednoduché reprezentovať akékoľvek dáta ako textové reťazce.

Preto *trie* predstavuje skôr základnú myšlienku alebo smer, ktorý je ďalej rozvíjaný.

Degradáciu na lineárny zoznam rieši štruktúra s názvom *Patricia trie*[5] Tá umožňuje uloženie viac ako jedného znaku v jednom uzle. To sa použije v prípade že rodičovský uzol obsahuje jediného potomka. Vtedy sú tieto dva uzly zlúčené.

Na zdieľanie prípon sú potrebné ďalšie optimalizačné algoritmy. Tie z pôvodnej štruktúry vytvoria *smerovaný acyklický graf slov – DAWG*[6].

Na tieto štruktúry je možné nazerať ako na *konečné automaty* (KA). Potom je možné aplikovať ďalšie optimalizácie. Ich cieľom je nájsť automat s minimálnym počtom prechodov, ktorý zároveň plní rovnakú funkciu ako pôvodný automat [9].

Toto používa skupina aplikácií *FSA*[7]. Autor využíva niektoré pokročilé algoritmy. Cieľom je zmenšenie časovej a pamäťovej náročnosti spracovania konečného automatu a jeho výslednej veľkosti.

2.6 Spôsoby kontroly gramatiky

U prirodzeného jazyka existuje niekoľko prístupov ku kontrole gramatiky. Ako je uvedené v [6], je možné zvoliť nasledujúce postupy:

1. Kontrola založená na analýze syntaxe. Analyzátor sa v tomto prípade snaží k vete pristupovať z jazykovedného hľadiska. Pokúša sa vytvoriť syntaktický strom. Ak sa to úspešne podarí, označí vetu za gramaticky korektnú. V prípade, že sa strom z nejakého dôvodu nepodarilo zostaviť, vyhlási vetu za chybnú.

   Výhodou tohoto postupu je skutočnosť, že potenciálne môže zachytiť gramatické chyby, ktoré nie sú špecifické určitou postupnosťou slov vo vete. Na druhej strane vytvorenie úplného formálneho popisu určitého jazyka je veľmi komplikovaná úloha.

2. Kontrola založená na štatistike a pravdepodobnosti. Priamo súvisí z použitím štatistických metód v morfologickej analýze. Vychádza z toho, že postupnosti určitých morfologických značiek sú veľmi časté. Naopak iné sú tak zriedkavé, že takéto postupnosti slov možno označiť ako chybu.

   Jej úspešnosť je zrejmá napríklad v angličtine a iných jazykoch, kde sa možno spoľahnúť na pevnú štruktúru vety a ustálené slovné spojenia.

[5] Niekedy označovaná aj ako *Patricia tree*, *radix tree*, alebo *crit bit tree.*

[6] Z anglického *Directed acyclic word graph*

[7] `http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/fsa.html`

3. Kontrola založená na použití pravidiel. Najprv sa slovám vo vete pridelia čo najpresnejšie morfologické značky. Potom sú na vety aplikované pravidlá. Na rozdiel od predchádzajúceho spôsobu, pravidlá sú vytvárané ručne a možno v nich vyjadriť skutočné gramatické pravidlá pre daný jazyk.

Ďalej je možné korektory hodnotiť z hľadiska pozitívneho, respektíve negatívneho prístupu ku správnosti napísanej vety. Pozitívny prístup je možné stotožniť s 1. postupom. Korektor s takýmto prístupom pozná správnu gramatiku a snaží sa podľa nej danú vetu spracovať. Ak uspeje, vyhlási vetu za správnu. Ak nastane počas spracovania chyba, označí danú vetu za nesprávnu.

Negatívny prístup je zameraný na opačnú činnosť. Snaží sa vyhľadať tie vety (alebo ich časti), ktoré dokáže rozoznať ako nesprávne. Sem patria posledné dva, ale hlavne 3. zo spomenutých typov kontroly gramatiky.

Výhodou negatívneho postupu by malo byť, že nielen zistí že veta je gramaticky nesprávna, ale dokáže presnejšie navrhnúť aj riešenie danej chyby. V reálnych gramatických korektoroch sa používa kombinácia spomenutých prístupov.

Kapitola 3

Spôsob práce s aplikáciou

V zadaní bolo požadované, aby bol výsledný korektor súčasťou kancelárskeho balíka. Tu vyvstáva otázka výhod, prípadne nevýhod takéhoto riešenia. Pokúsim sa tiež zhrnúť, aké výhody by priniesla implementácia korektora ako samostatnej aplikácie.

Ak je korektor vytvorený formou prídavného modulu, potom

+ Prináša užívateľovi ďalšie možnosti a nástroje k známemu softvéru.
+ Môže použiť prostriedky kancelárskeho balíka.
+ Je väčšia pravdepodobnosť jeho použitia. Toto závisí od popularity a rozšírenosti kancelárskeho balíka, ktorého je súčasťou.
– Niektoré vlastnosti sa môžu zložito implementovať, prípadne je možné, že sa nedajú implementovať vôbec. Modul môže pracovať len takým spôsobom, akým to rozhranie kancelárskeho balíka dovoľuje.
– Ak chce užívateľ použiť gramatický korektor, potrebuje najprv nainštalovať kancelársky balík. Čo môže byť nežiaduce, napríklad pri nedostatku volného miesta na cieľovom počítači, alebo ak je balík náročná aplikácia, ktorá počítač priveľmi vyťaží.

Na druhej strane korektor ako samostatná aplikácia

+ Je obyčajne kompaktnejší a ľahšie použiteľný program.
+ Nemusí byť obmedzený rozhraním kancelárskeho balíka. Užívateľovi môže poskytnúť detailnejšie informácie ohľadom kontrolovaného textu. Napríklad môže chyby odstupňovať na niekoľko úrovní (chyby, varovania, štylistické chyby) a intuitívnym spôsobom ich rozlíšiť.[1]
+ Nie je potrebné inštalovať žiaden ďalší softvér.
– Je potrebné vytvoriť celý program od „nuly“.
– Existuje obrovské množstvo malých špecifických programov. Ak nie je možné korektor zaradiť pod meno známeho kancelárskeho balíka, mohlo by sa naň ľahko zabudnúť.

[1] Balík MS Office toto v súčastnosti podporuje. Vývojový tým OpenOffice pracuje na rozhraní, ktoré by štandardizovalo korekciu gramatiky.

Ako ideálne riešenie sa javí vytvoriť korektor, ktorý dokáže fungovať obidvoma spôsobmi. Je možné ho začleniť do kancelárskeho balíka, kde sprístupní užívateľovi kontrolu gramatiky v známom programe. V prípade potreby ho ale užívateľ môže spustiť aj ako samostatnú aplikáciu. Tu korektor už nemusí byť obmedzovaný vlastnosťami balíka. V pamäti sa kancelársky balík vôbec nemusí vyskytovať, keď nie je potrebný.

Zvolený postup napríklad používa *Lingea Gramaticon*. A taktiež *LanguageTool*, ktorý bol použitý v tejto práci.

3.1 Kancelársky balík OpenOffice

V súčastnosti väčšina ľudí pohybujúcich sa v oblasti informačných technológií OpenOffice pozná, prípadne s ním má aj priame skúsenosti. Jeho predchodcom je *StarOffice suite*, ktorý bol v roku 1999 kúpený firmou *Sun Microsystems*. Tá je aj v súčastnosti jedným zo sponzorov projektu OpenOffice a hlavným prispievateľom zdrojového kódu.[7]

Integráciu výsledného gramatického korektora do OpenOffice som si zvolil hlavne pre dostupnosť balíka, ktorý je šírený pod *LGPL* licenciou. OpenOffice je taktiež dostatočne známy a rozšírený.

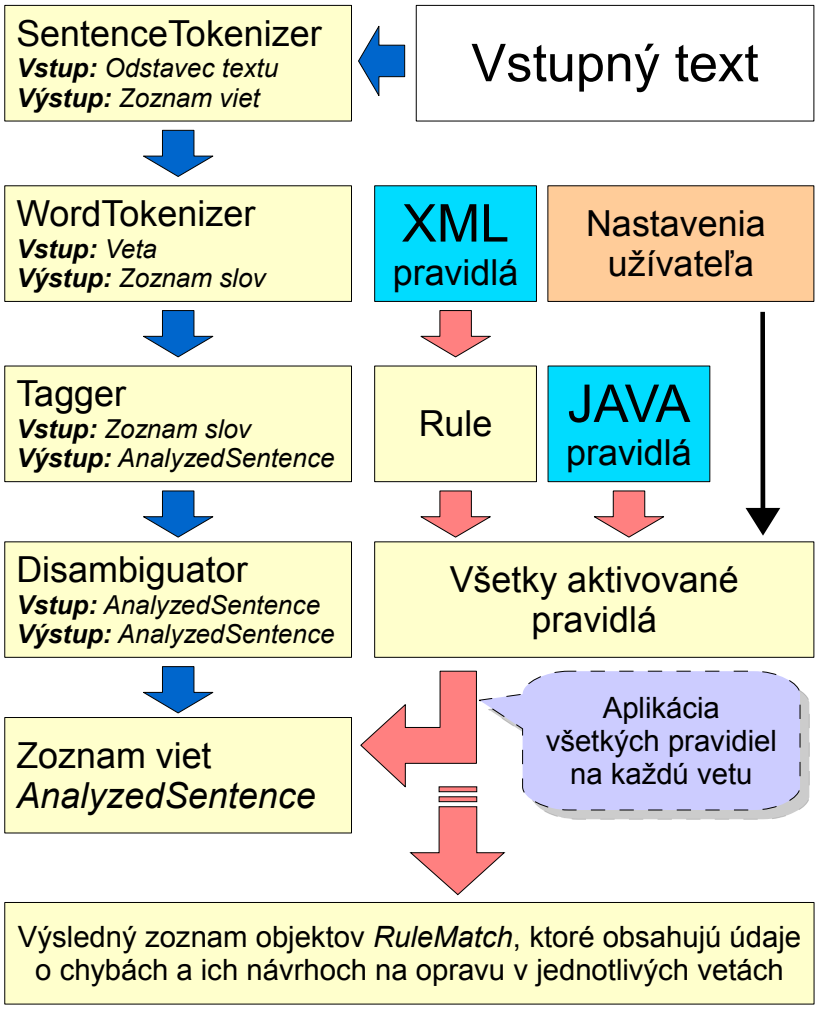
V súčastnosti podporuje pokročilý systém rozšírení, čo bolo pre môj projekt tiež dôležité. Rozšírenia je možné vytvoriť hlavne v jazykoch *C++*, *JAVA* alebo *Python*. OpenOffice rieši rozšírenia formou UNO[2] komponentov. Rozširujúci modul je možné naprogramovať napríklad v jazyku JAVA, v ktorom je podporované UNO rozhranie. A keďže aj OpenOffice podporuje UNO rozhranie, je možné, aby bolo rozšírenie vytvorené v inom jazyku (JAVA) ako samotný kancelársky balík (C++).

3.2 Charakteristika aplikácie LanguageTool

Ako už bolo spomenuté, gramatický korektor môže pracovať na princípe analýzy syntaxe, na základe štatistiky, alebo je jeho kontrola založená na použití pravidiel. Korektor LanguageTool používa posledný z vymenovaných prístupov. Je vytvorený v jazyku *JAVA*. Nie je preto problém s prenositeľnosťou aplikácie. Zároveň aplikácia poskytuje dostatočný výkon pri kontrole.
Vstupný text je postupne spracovaný jednotlivými triedami ako je uvedené na obrázku 3.1.

- Vstupný text je načítaný po jednotlivých odstavcoch
- Odstavec textu je rozdelený na jednotlivé vety.
- Veta je rozdelená na slová a neslovné súčasti ako napríklad interpunkcia.
- Každá takto rozdelená veta je spracovaná morfologickým analyzátorom, ktorý slovám priradí značky.
- Vetu následne spracuje zjednoznačňovacia časť LanguageTool a odstráni nepotrebné morfologické značky.
- Na takto pripravené vety sú aplikované pravidlá. Ak sa nejaké pravidlo podarí uspešne použiť, daná veta je zaradená do zoznamu nesprávnych viet.

[2] Universal Network Objects



Obrázek 3.1: Diagram postupnosti operácií v LanguageTool

Po spracovaní textu sú užívateľovi zobrazené odhalené chyby spolu s návrhom ich opravy. Užívateľ má možnosť určité pravidlá podľa potreby dezaktivovať v nastaveniach programu.

Pravidlá je možné definovať dvoma spôsobmi. Pokiaľ to povaha pravidla dovoľuje, je možné ho definovať v externom *XML* súbore. Jedná sa o definovanie postupnosti slov, ktorá bude označená za gramatickú chybu. Je možné odvolať sa aj na informácie z morfologického analyzátora: morfologickú značku alebo základný tvar slova. Položka môže obsahovať aj regulárny výraz namiesto konkrétneho reťazca. Súčasťou definície je nesprávny príklad – použitie danej postupnosti vo vete. A správny príklad, čo je opravená fráza. Užívateľovi je ponúknutá oprava, ktorú môže prijať, alebo zamietnuť.

Pravidlá, ktoré nie je možné definovať v externom súbore, sa dajú naprogramovať v jazyku *JAVA*, potom sa stanú súčasťou aplikácie LanguageTool. (Nie je možné ich vo výslednom balíku meniť.)

Výhodou externých pravidiel zapísaných v XML súbore je ich prehľadnosť. Ďalej LanguageTool podporuje automatickú kontrolu týchto pravidiel. Namiesto kompilácie programu

je možné spustiť testovanie, ktoré sa stará o automatické preverenie definovaných častí aplikácie. Do testov sú automaticky zahrnuté všetky XML pravidlá, ktoré majú definovanú správnu a nesprávnu modelovú vetu. Testovanie prebieha tak, že na modelové nesprávne vety je aplikované pravidlo ku ktorému patria. Kontroluje sa, či pravidlo chybu nájde a či aj správne označí jej rozsah. Na správne modelové vety je pravidlo tiež použité, v nich sa gramatická chyba nesmie nájsť.

Ako už bolo spomenuté, LanguageTool je možné použiť samostatne, alebo ako prídavný modul do balíka OpenOffice. Samostatne sa správa ako konzolová aplikácia keď je spustená príkazom

```
java -jar LanguageTool.jar (parametre)
```

LanguageTool je však možné spustiť aj ako aplikáciu s grafickým užívateľským rozhraním. Na to je potrebný príkaz

```
java -jar LanguageToolGUI.jar
```

V prípade, že je žiadúce použiť LanguageTool ako súčasť OpenOffice, je potrebné ho inštalovať[3] (menu *Nástroje* → *Správca Rozšírení...* v slovenskej lokalizácii). Do kategórie *Moje rozšírenia* je potrebné pridať nové rozšírenie. Po kliknutí na tlačítko Pridať je potrebné lokalizovať v súborovom systéme archív `LanguageTool.zip`, potvrdiť dialóg a následne reštartovať OpenOffice.

Potom by sa v hlavnom menu aplikácie už malo vyskytovať ďalšie menu *LanguageTool*. Tam je možné nastaviť korektor a aj spustiť samotnú kontrolu gramatiky.

[3]Podrobnejší postup inštalácie sa nachádza na priloženom médiu. Je tam popísaný aj postup ako sa vyhnúť jednému problému, ktorý sa vyskytuje pri inštalácii LanguageTool na slovenskej a českej lokalizácii Windows XP.

Časť II

Implementácia

Kapitola 4

Podpora češtiny v LanguageTool

Aplikácia LanguageTool je navrhnutá modulárne, čo priamo súvisí s jej implementačným jazykom. Poskytuje rozhrania jazyka JAVA zobrazené na obrázku 3.1. Tie je potrebné implementovať, alebo použiť štandardne implementované riešenie.

- *SentenceTokenizer* rozdelí odstavec vstupného textu na jednotlivé vety. Štandardná implementácia pre anglický jazyk vie spracovať aj zložitejšie súvetia a vety s priamou rečou. Ošetruje prípady keď znak, ktorý obyčajne znamená koniec vety, má v skutočnosti iný význam. Ďalej môže dochádzať k omylom, ak sa vo vete nachádzajú skratky. LanguageTool sa snaží automaticky rozoznať jednopísmenové skratky. Viacpísmenové sú závislé na jazyku. Tu bolo potrebné rozhranie reimplementovať. Vytvoril som triedu *CzechSentenceTokenizer*, ktorá používa postupy na rozoznanie viet spoločné všetkým implementáciám v LanguageTool. Naviac som do nej pridal zoznam českých skratiek, aby k spomenutým omylom pri kontrole českého textu nedochádzalo. (Respektíve aby sa počet odstavcov nesprávne rozdelených na vety čo najviac znížil.)

- *Tokenizer* má za cieľ rozdeliť vetu na jednotlivé časti (slová a neslovné súčasti vety ako čiarky a medzery medzi slovami). Tu som použil štandardný *WordTokenizer*.

- *Tagger* na vstupe získa slovné časti vety. Jeho úlohou je prideliť slovám morfologické značky. Jednému slovu je možné prideliť neobmedzený počet značiek.

  LanguageTool na značkovanie využíva externý dátový súbor spracovaný spomenutými aplikáciami FSA. Načítanie dát zo súboru zabezpečuje externý JAVA balík Stempelator/Lametyzator[1].

  Vytvoril som triedu *CzechTagger*, ktorá spomenuté rozhranie implementovala. České morfologické dáta boli optimalizované (kap 4.1), aby veľkosť výsledného súboru bola čo najmenšia. Preto má obsah dátového súboru štruktúru odlišnú od dátových súborov ostatných jazykov. Dôsledkom toho je, že aj telo triedy *CzechTagger* sa od ostatných implementácií líši.

- *Disambiguator* je rozhranie, ktoré som vytvoril a pridal do aplikácie. Ako je uvedené v nasledujúcej sekcii, pri použití českých morfologických dát dochádza často k nejednoznačnosti pridelením viacerých značiek. Preto bolo potrebné vytvoriť zjednoznačňovací krok, kde som sa pokúsil nesprávne značky odstrániť.

[1] Jedná sa o prepísanie časti FSA aplikácií do jazyka JAVA. Autorom je David Weiss. Aplikácia patrí k projektu Morfologik (`http://sourceforge.net/projects/morfologik/`)

4.1 Tvorba morfologických dát

Ako vstup som použil dáta zo spomenutého morfologického analyzátora *AJKA* [8]. V podstate to bol textový súbor, ktorý pozostával z riadkov vo formáte

```
<slovo>#<lemma>#<morfologické značky>
```

Prvá značka je slovo v určitom tvare, potom nasleduje jeho základný tvar (lemma) a posledné sú morfologické značky patriace k slovu. Databáza obsahuje 6 miliónov tvarov. No mnoho slov má v danom tvare viac významov, a preto bol tento tvar často zduplikovaný (mal priradené iné morfologické značky, prípadne aj iný základný tvar). Z tohoto dôvodu databáza obsahovala 31 564 560 záznamov a jej veľkosť bola približne 1,1 GB.

Tieto dáta som spracoval pomocou aplikácie `fsa_build`[2]. Výsledný súbor však nebol dostatočne kompaktný (jeho veľkosť bola približne 100 MB). Bolo preto potrebné databázu vopred optimalizovať a až potom ju zostaviť do podoby konečného automatu.

V balíku *FSA* sa okrem samotných aplikácií nachádzali aj rôzne skripty v jazykoch *PERL* a *AWK*. Boli určené na optimalizáciu vstupných dát. Ich princíp spočíval v tom, že konkrétny tvar slova je často podobný základnému tvaru. Jeden zo skriptov napríklad vykonal zmenu z predošlého na

```
<slovo>#X<koncovka>#<morfologické značky>
```

Kde `X` zastupuje akékoľvek veľké písmeno abecedy. Značí koľko písmen sa má zo slova `<slovo>` zmazať z konca, aby sa potom k nemu mohla pridať `<koncovka>`. Písmeno `A` značí, že sa nemá zmazať žiadne písmeno, `B` že sa má zmazať jedno písmeno, `C` treba zmazať dve atď. Napríklad

```
porážce#porážka#k1gFnSc3   →   porážce#Cka#k1gFnSc3
```

Týmto bolo možné dosiahnuť určitú kompresiu dát pri uchovaní tej istej informácie. Balík *FSA* obsahoval aj ďalšie skripty, ktoré vykonávali podobné nahradenia aj ohľadom *prefixu* a *infixu*. Toto však stále nepostačovalo. Veľkosť výsledného súboru bola stále v desiatkach MB.

Nakoniec sa ako vhodný ukázal nasledujúci postup.

1. V súbore lokalizovať záznamy, ktoré obsahujú rovnaký slovný tvar a základný tvar. Vytvoriť z nich iba jeden záznam, ktorý obsahoval morfologické značky zreťazené za sebou. Napríklad namiesto štyroch riadkov

   ```
   prstýnky#prstýnek#k1gInPc1
   prstýnky#prstýnek#k1gInPc4
   prstýnky#prstýnek#k1gInPc5
   prstýnky#prstýnek#k1gInPc7
   ```

   vznikne po tejto kompresii jeden riadok

   ```
   prstýnky#prstýnek#k1gInPc1+k1gInPc4+k1gInPc5+k1gInPc7
   ```

2. Potom sa použila spomínaná kompresia prefixov, infixov a postfixov.

3. Nakoniec boli dáta spracované aplikáciou `fsa_build`.

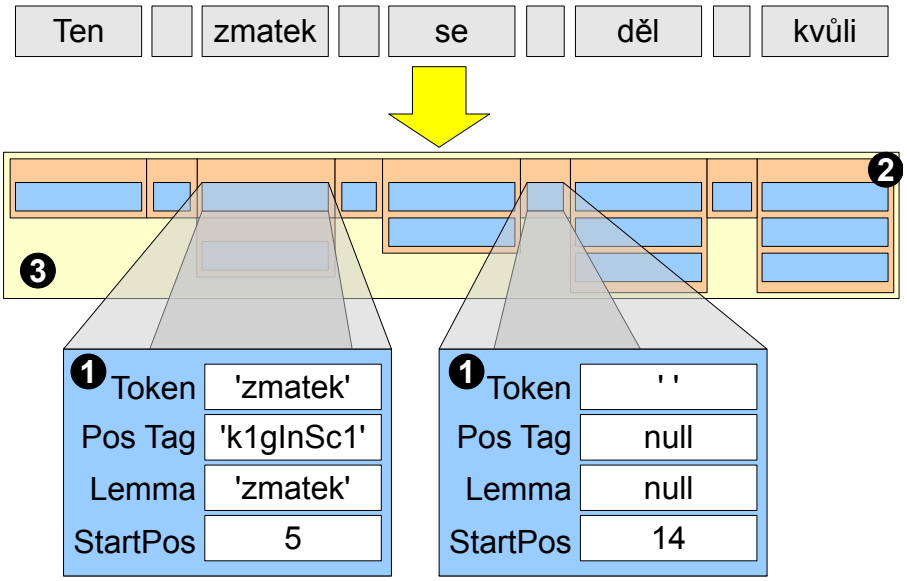
Takto získaný súbor mal veľkosť 10 MB, čo už bolo použiteľné.

[2] Jedna z aplikácií balíka FSA

4.2 Morfologický analyzátor

Každý jazyk podporovaný v LanguageTool má možnosť implementovať vlastný morfologický analyzátor. V prípade, že takýto analyzátor nie je implementovaný, je možné použiť štandardný. Ten však robí len jedinú vec – priradí každému slovu prázdnu morfologickú značku.

Pre češtinu som vytvoril triedu *CzechTagger*, v ktorej je analyzátor implementovaný. Rozhranie analyzátora je navrhnuté tak, že jeho vstupom je veta. (Presnejšie zoznam slov nachádzajúcich sa v nej – interpunkčné znamienka sú vynechané.) A výstupom je dátová štruktúra *AnalyzedSentence*. Model *AnalyzedSentence* je znázornený na obrázku 4.1.



Obrázek 4.1: Diagram dátovej štruktúry, ktorá v pamäti uchováva vetu.

① *AnalyzedToken* je dátová štruktúra, ktorá uchováva informácie o jednej morfologickej značke.

② V štruktúre *AnalyzedTokenReadings* su zoskupené jednotlivé štruktúry *AnalyzedToken*, ktoré patria k tomu istému slovu vo vete.

③ Nakoniec *AnalyzedSentence* obsahuje v sebe zoradené štruktúry *AnalyzedTokenReadings* podľa poradia výskytu slov vo vete.

Funkčnosť triedy spočíva v inicializácii spomínanej aplikácie Lametyzator s parametrami ako cesta k dátovému súboru a kódovanie znakov, ktoré sa má použiť. Následne je Lametyzator dotazovaný na požadované slová. Ak sa dané slovo v databáze nájde, vyberú sa všetky jeho základné tvary a značky. Ak sa nenájde, použije sa prázdny základný tvar a prázdna značka.

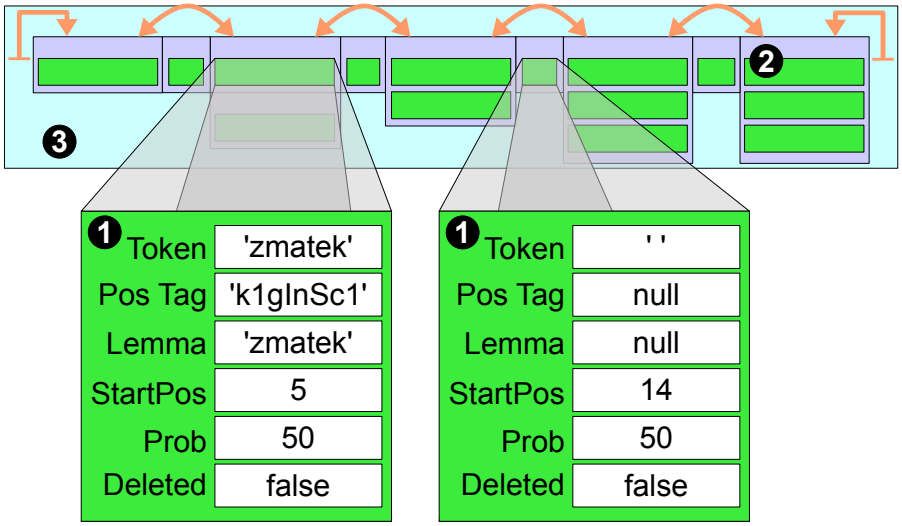
Analyzátor som ešte rozšíril o jednu vlastnosť. Tá súvisí s tým, že morfologická databáza neobsahuje slovesá končiace na `-li` (napr. *jsou-li*, *je-li*, *mám-li* atď.). Ak sa takéto slovo v texte vyskytne, analyzátor z neho odstráni spomenutú koncovku a načíta jeho morfologické značky ako keby to bolo obyčajné sloveso. Keď je ďalej potrebné vytvoriť štruktúru

AnalyzedSentence, koncovka je k slovesu znovu pripojená a ku každej morfologickej značke je pridaný atribút `qC`.

4.3 Zjednoznačnenie morfologických značiek

Výstup spomenutého analyzátora je uložený v dátovej štruktúre *AnalyzedSentence*. Ďalej je predaný na vstup zjednoznačňovacej triede (*CzechDisambiguator*). Jej úloha je zrejmá zo samotného názvu: dosiahnuť, aby mali slová čo najjednoznačnejšie pridelené morfologické značky.

Pre potreby zjednoznačňovania som rozšíril štruktúru AnalzyedSentence, ktorá je súčasťou LanguageTool tak, že som navrhol a vytvoril objekt znázornený na obrázku 4.2. Pri vytváraní objektu je potrebné ako parameter zadať konkrétnu štruktúru *AnalzyedSentence*, ktorej dáta sa na vytvorenie objektu použijú. Spomínané štruktúry *AmbiguousSentence* a *AnalzyedSentence* sú podobné, vzhľadom na hierarchiu ich vnorených objektov. *AmbiguousSentence* však disponuje množstvom pomocných metód, ktoré majú za cieľ zjednoznačňovanie uľahčiť a sprehľadniť. Niektoré metódy objektov sú spomenuté v nasledujúcom popise. Úplná dokumentácia vo formáte *Javadoc* sa nachádza na priloženom médiu.



Obrázek 4.2: Diagram dátovej štruktúry vhodnej na prácu s jednotlivými morfologickými značkami.

① *EnrichedTag* je ekvivalent štruktúry *AnalyzedToken*. Naviac si uchováva informácie ako pravdepodobnosť konkrétnej značky a jej platnosť. Obsahuje metódy na zmenu pravdepodobnosti, prípadne zmenu značky, ak to je potrebné.

② *EnrichedToken* zoskupuje jednotlivé značky (resp. dátové štruktúry *EnrichedTag*) a ponúka rozhranie na rýchlu prácu s nimi. Je možné vybrať najpravdepodobnejšiu značku z dostupných. Odstrániť značky, ktoré majú pravdepodobnosť nižšiu ako daná hranica. Ďalej ponúka možnosť zistiť predchádzajúce a nasledujúce slovo (presnejšie *token*, čiže atomickú časť vety, ktorá nie je zložená z bielych znakov).

③ *AmbiguousSentence* je na najvyššej úrovni. Zapúzdruje predošlé štruktúry a poskytuje rozhranie k práci s celou vetou. Napríklad je možné vyhľadať určité slovo a pohybovať sa po jednotlivých slovách vety. (Automatickým preskakovaním medzier medzi slovami, ktoré sú taktiež v štruktúre obsiahnuté)

Na inicializáciu *AmbiguousSentence* je potrebná spomenutá štruktúra *AnalyzedSentence*. Po vykonaní samotného zjednoznačňovania *AmbiguousSentence* obsahuje metódu, ktorá jednotlivé časti vety a ich morfologické značky, ktoré boli zachované, znovu prevedie na štruktúru *AnalyzedSentence*. S takýmto výsledkom potom zvyšok aplikácie pracuje. Vďaka tomuto postupu je zjednoznačňovacia časť transparentná pre zvyšok aplikácie.

Ako bolo už spomenuté, morfologický analyzátor disponuje rozsiahlou databázou českých slov. Preto veľkej väčšine slov v kontrolovaných vetách dokáže priradiť zodpovedajúce morfologické značky. Napriek tomu sa stane, že určitým slovám nedokáže prideliť žiadnu značku.

V zjednoznačňovacej časti som sa na takéto slová zameral. Vychádzal som z predpokladu, že väčšina z nich sú vlastné mená, názvy a rôzne skratky. Ak je dané slovo bez značiek, sú naň aplikované nasledujúce pravidlá:

- Ak je slovo zložené len z veľkých písmen abecedy, priraď mu značku `kA` (skratka).
- Ak sa dané slovo nenachádza na začiatku vety a zároveň začína veľkým písmenom abecedy, priraď mu značku `k1` (podstatné meno).

Ďalej som zjednoznačňovanie zameral na niektoré slová obsahujúce viac značiek. Išlo hlavne o slová, ktoré je možné interpretovať ako slovesá a zároveň ako iný slovný druh. Pri následnom vytváraní pravidiel na správne určenie vzťažnej vety sú totiž dôležité slovesá. Prítomnosť slova, ktoré slovesom nebolo, ale napriek tomu značku slovesa obsahovalo, mala často za následok nesprávnu aplikáciu pravidla.

Nasledujúce slová patria do spomenutej kategórie. Pri každom z nich je uvedená myšlienka, na ktorej som založil odstránenie nesprávnej značky daného slova.

se Je všeobecne jedno z najčastejšie sa vyskytujúcich slov v češtine. Medzi jeho morfologickými značkami sa síce nevyskytuje sloveso, no jeho zjednoznačnenie som implementoval hlavne pre jeho častý výskyt. Výsledné pravidlo vždy zistilo slovo vyskytujúce sa za *se*. Ak sa za ním nachádzalo podstatné meno, prídavné meno, zámeno alebo číslovka v 7. páde, zvýšila sa pravdepodobnosť značky „predložka“. Pravdepodobnosť predložky bola zvýšená aj v prípade, keď sa za *se* nachádzalo slovo začínajúce na `s` alebo `z`.

při Je slovo zaujímavejšie. Môže to byť podstatné meno (množné číslo od *pře*), sloveso (rozkazovací spôsob od *přít se;*) a predložka. Poslednej možnosti som dal prednosť v prípade, že sa za ňou nachádzalo slovo v šiestom páde. Sloveso to byť nemohlo, ak sa bezprostredne pred aktuálnym slovom už jedno sloveso nachádzalo. A podstatné meno som riešil analogicky. Ak sa pred alebo za aktuálnym slovom nachádzalo podstatné meno (bez oddelenia čiarkami), tak bola pravdepodobnosť pods. mena zmenšená.

opět Môže znamenať príslovku alebo sloveso. Ak sa už jedno sloveso v okolí nachádza, je uprednostnená príslovka.

prechodníky Napríklad slová ako *dobře*, *jistě*, *pravděpodobně*, *rychle*, *dokonale* atď. Ak slovo so značkou prechodníku obsahovalo zároveň aj značku príslovky, uprednostnil som túto značku a odstránil som informáciu o slovese v tvare prechodníku.

4.4 Pravidlá na vyhľadanie chýb

Navrhol som pravidlá, ktorých úlohou bolo nájsť vedľajšiu vetu vzťažnú, ktorá nie je ohraničená čiarkami. Definoval som ich v externom XML súbore určením charakteristických postupností slov, ktoré môžu naznačovať, že na danom mieste vo vete je vedľajšia veta vzťažná. Výsledkom bolo 11 pravidiel, ktorých úspešnosť a pokrytie sú vyhodnotené v podkapitole 5.3.

Naviac som definoval nasledujúce pravidlá, ktoré riešia ďalšie gramatické chyby. Pri každom z nich je uvedená modelová nesprávna veta. Ďalej je naznačené, ako korektor danú vetu opraví. ~~Prešikrtnuté~~ slovo znamená, že korektor navrhne jeho odstránenie. Podčiarknuté slovo (respektíve čiarka) je návrh korektora, ako danú vetu opraviť.

- Nesprávne použitie stupňovania. Ak sa je slovo *více* (resp. *méně*) pred prídavným menom v 2. stupni (komparatíve), je to chyba. Napríklad

  Toto auto je ~~více~~ krásnější než tamhleto.

- Nesprávne použitie zámena *tyto* pre stredný rod.

  Rozeznal ~~tyto~~ tato významná slova.

- Nesprávne použitie predložiek *s/se*[3] s 2. pádom.

  Strach ~~s~~ z neznámého.

- Nesprávne použitie predložiek *z/ze* so 7. pádom.

  Byl spokojen ~~z~~ s pátým místem.

- Chýbajúca čiarka pred spojovacími výrazmi začínajúcimi na *a*. Napríklad *a dokonce*, *a proto*, *a přece*, *a tak*, *a to*, *a tudíž*.

  Nežijeme v minulosti, a proto se musíme snažit.

Do aplikácie je možné pridať ďalšie pravidlá úpravou XML súboru s českými pravidlami, ktorý sa nachádza v adresári `rules/cs/`.

[3] Pravidlo rozoznávajúce predložku *se* je v súbore definované, ale je dezaktivované, pretože spôsobovalo falošné poplachy. Bolo by potrebné upraviť zjednoznačňovaciu časť aplikácie, aby toto slovo dokázala lepšie rozoznať. Súčasná verzia zjednoznačňovania je úspešná, pokiaľ sa použije na korektné vety. Pri použití na vetu v ktorej je slovo *se* použité nesprávne namiesto predložky *ze*, je často jeho morfologická značka určená nesprávne.

Kapitola 5

Testovanie a vyhodnotenie

Na overenie funkčnosti korektora bolo potrebné vytvoriť databázu vhodných viet. Ako zdroj som použil textový korpus na FIT VUT.

5.1 Príprava vhodných testovacích dát

Fakultný korpus je uchovávaný vo formáte, kde sa v texte nachádzajú meta-informácie vo forme SGML značiek. Ďalej sú tieto dáta uložené tak, že sa na jednom riadku nachádza vždy iba jedno slovo alebo značka.

Pri testovaní som LanguageTool nepoužíval ako modul OpenOffice, ale ako konzolovú aplikáciu. V tomto prípade LanguageTool akceptuje ako vstup súbor s textom, ktorého gramatiku má overiť. Vstupný text nesmie obsahovať žiadne meta-informácie a mal by byť rozdelený na odstavce. Oddelenie odstavcov štandardne značí znak nového riadku zopakovaný dvakrát (`\n\n`).

Bolo preto potrebné previesť text z fakultného korpusu do cieľového formátu a potom v ňom vyhľadať vhodné vety. Navrhol som nasledujúce kroky na získanie potrebných dát:

1. Vytvoril som skript v jazyku *Python*. Ten z vertikálneho textového korpusu vybral text, ktorý sa nachádzal medzi párovými značkami `<p>` a `</p>`. Tieto značky vymedzovali štandardný odstavec textu. Z tohoto odstavca skript odstránil ďalšie vyskytujúce sa metaznačky a spojil riadky, aby vznikol súvislý odstavec. Skript dáta načítal zo štandardného vstupu a výstup zapísal na štandardný výstup. Odstavce oddelil dvoma znakmi nového riadku.

2. Odstavce textu bolo ďalej potrebné rozdeliť na menšie celky – jednotlivé vety. Aby bolo potom možné vybrať len vhodné vety. Konkrétne tie, v ktorých sa nachádzali vedľajšie vety vzťažné.

   Bolo už spomenuté, že rozdelenie textu na jednotlivé vety nie je triviálna záležitosť. Komplikácie vznikajú tým, že sa vo vetách vyskytujú napríklad skratky a dátumy, v ktorých sa tiež nachádzajú interpunkčné znamienka.

   Tu som vhodne využil súčasť LanguageTool ktorá už danú funkčnosť má – spomínanú triedu *CzechSentenceTokenizer*. Tá už mala implementované postupy na rozoznanie konca vety pre český jazyk.

   Vytvoril som ďalšiu JAVA aplikáciu *SentenceFinder*, ktorá využíva spomenuté rutiny z LanguageTool na rozdelenie odstavcov na vety. Aplikácia je znovu riešená tak, že

odstavce načítava zo štandardného vstupu. Na výstup zapíše text rozdelený do viet. Každú vetu ukončí znakom nového riadku.

3. Posledný krok zabezpečuje ďalší skript v jazyku *Python*. Ten na štandardný vstup očakáva výstup predchádzajúcej aplikácie. Potom pomocou regulárnych výrazov vyberie vety, ktoré obsahujú čiarku a za ňou vzťažné zámeno. Výsledok zapíše na štandardný výstup.

Každá zo spomenutých aplikácií spracúva dáta po odstavcoch, prípadne po vetách. Nezhromažďuje si žiadne dielčie informácie. Toto riešenie som zvolil preto, aby boli nároky na pamäť čo najmenšie a aby bolo možné spracovať aj objemné dáta. Prepojením štandardného výstupu jednej aplikácie so vstupom ďalšej som dosiahol požadovaný výsledok – súvetia, ktoré obsahujú vety uvodené vzťažným zámenom.

5.2 Postup testovania pravidiel na získaných dátach

Prvým krokom pri testovaní bolo odstránenie čiarok vo vzorových vetách. Vytvorené pravidlá boli zamerané na doplnenie čiarok vo vetách, kde sa čiarka nachádza za vzťažným zámenom, prípadne je vo vete vložená veta uvodená vzťažným zámenom. Preto som z viet neodstraňoval všetky čiarky. Odstránil som čiarku nachádzajúcu sa pred vzťažným zámenom a čiarku bezprostredne po nej nasledujúcu (ak sa vo vete vyskytovala).

Tento postup bol implementovaný formou skriptu v jazyku *Python*. Skript zo štandardného vstupu načíta vety. Na štandardný výstup zapíše vety výsledné, ktoré majú adekvátne čiarky zmazané.

Databáza viet s odstránenými čiarkami je potom predaná na vstup aplikácie LanguageTool. Keď je aplikácia LanguageTool spustená z príkazového riadku, jej výstupom je textový zoznam. Tento zoznam obsahuje všetky úspešne aplikované pravidlá spolu s časťou vety, kde sa podarilo dané pravidlo aplikovať. Jeho účel je zrejme upozorniť užívateľa na potenciálne gramatické chyby. Napríklad položka

```
1.) Line 2, column 7, Rule ID: CS_COMMA_04
Message: oprava 'směrem kterým' na 'směrem, kterým';; oprava 'šel je' na 'šel, je'
Suggestion: směrem, kterým; šel, je
Třetím směrem kterým vývojový tým šel je vedle nových technologií...
       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

obsahuje informácie o tom, že v opravovanom texte na druhom riadku a v siedmom stĺpci bolo úspešne použité pravidlo `CS_COMMA_04`. Nasleduje správa, ktorá je definovaná pri pravidle. Nakoniec je zvýraznená časť textu, ktorej sa záznam týka.

Mojim cieľom bolo ísť ďalej, automaticky spracovať vytvorený zoznam a aplikovať všetky opravy, ktoré boli navrhnuté použitými pravidlami, na text v ktorom boli odstránené čiarky. Potom už zostával len posledný krok: Porovnať pôvodné správne vety z vetami, ktoré prešli procesom: odstránenie čiarok — analýza nástrojom LanguageTool — aplikovanie opráv.

Na obnovenie čiarok vo vetách a ich porovnanie s pôvodnými vetami som vytvoril ďalší skript. Na jeho štandardný vstup je potrebné zaslať výstup aplikácie LanguageTool. Ďalej prijíma dva povinné parametre. Prvým je názov súboru, v ktorom sa nachádzajú vety z odstránenými čiarkami – v nich budú vykonané zámeny. Druhý parameter musí byť názov súboru z pôvodnými správnymi vetami.

Skript postupne prevedie doporučené náhrady. Pri tom si uchováva počet použití jednotlivých pravidiel. Každú opravenú vetu porovná z jej správnym originálom. Ak sa vety rovnajú, pravidlu je inkrementované počítadlo správnych použití.

Zo spomenutých údajov je na koniec vypočítaná štatistika. Najprv sú to informácie o pokrytí pravidiel. Pokrytie p som definoval vzťahom

$$p = \frac{PP}{PT} \tag{5.1}$$

kde PT je počet viet v testovacej databáze a PP počet viet, v ktorých bolo aplikované aspoň jedno pravidlo.

Ďalej skript pre každé pravidlo vypočítal úspešnosť u vyjadrenú vzťahom 5.2. Tu RC značí počet všetkých použití daného pravidla a RS počet úspešných použití.

$$u = \frac{RS}{RC} \tag{5.2}$$

5.3 Dosiahnuté výsledky

Testovanie prebehlo na vzorke 2083 súvetí, každé z nich obsahovalo vzťažnú vetu. Počas morfologickej analýzy 94,94% slov získalo aspoň jednu morfologickú značku. Z tých, ktorým nebola žiadna značka pridelená, bolo 67,16% rozoznaných zjednozačňovacou časťou analyzátora buď ako podstatné meno, alebo skratka. Zvyšok zostal nedefinovaný.

V štatistike úspešnosti boli zahrnuté pravidlá týkajúce sa vzťažných zámen. Tieto pravidlá dosiahli spolu pokrytie $p = 20.5\%$. Čo na uvedenej vzorke znamenalo 427 viet. V tabuľke 5.1 sú uvedené počty všetkých viet, na ktoré boli aplikované, a zodpovedajúce úspešnosti jednotlivých pravidiel.

Pravidlo	Počet použití	u
`CS_COMMA_08`	105	43,81%
`CS_COMMA_07`	31	45,16%
`CS_COMMA_04`	84	48,81%
`CS_COMMA_09`	99	65,66%
`CS_COMMA_03`	3	66,67%
`CS_COMMA_02`	64	70,31%
`CS_COMMA_01`	21	80,95%
`CS_COMMA_10`	10	90,00%
`CS_COMMA_11`	2	100,00%
`CS_COMMA_05`	4	100,00%
`CS_COMMA_06`	4	100,00%

Tabulka 5.1: Vyhodnotenie úspešnosti pravidiel

Výsledné údaje znázorňujú nasledujúcu vlastnosť pravidiel: Čím je pravidlo špecifickejšie definované, tým je úspešnejšie a zároveň pokrýva menšiu množinu viet.

Kapitola 6

Záver

V tejto práci som zdokumentoval vytvorenie jednoduchého gramatického korektora pre český jazyk. Implementácia sa týkala rozšírenia korektora LanguageTool tak, aby podporoval češtinu. Keď je táto podpora implementovaná, je možné ďalej korektor rozvíjať aj bez znalosti programovacieho jazyka doplnením ďalších pravidiel do externého XML súboru.

Aplikáciu je možné použiť ako prídavný modul do kancelárskeho balíka OpenOffice, ale aj ako samostatnú aplikáciu.

Výsledné pokrytie testovacích dát je dosť malé. Domnievam sa, že to je spôsobené tým, že pravidlá boli optimalizované tak, aby dosahovali čo najväčšiu úspešnosť, druhou stránkou tohoto riešenia je to, že sú viac špecifické. Z toho potom vyplýva ich menšie pokrytie.

Uvedené výsledky treba chápať aj v kontexte jazyka, na ktorý bol korektor zameraný. Najväčším problémom pravdepodobne bolo volné poradie slov (resp. vetných členov) vo vete. Preto bolo pomerne náročné odhaliť miesto, kde jedna veta končí a začína druhá, len pomocou znalosti kde sa aký slovný druh nachádza.

Pri ďalšej práci na tomto gramatickom korektore by som navrhol odlišný postup pri kontrole gramatiky. Je potrebné vytvoriť kvalitnejšiu zjednoznačňovaciu časť, ktorá dokáže pracovať presnejšie a vo väčšom rozsahu. Ďalej by bolo zaujímavé ako súčasť zjednoznačňovania pridať vyhľadávač slovesných skupín a jeho výsledky vložiť do vety pomocou špeciálnych značiek. To by mohol byť kvalitnejší základ pre úspešné vyhľadanie vedľajších viet vzťažných a samozrejme aj pre ostatné pravidlá.

Literatura

[1] Behún, D.: Kontrola české gramatiky pro MS Office – konec korektorů v Čechách? *Interval.cz*, 18. 7. 2005, ISSN 1212-8651, [cit. 2007-05-09].
URL http://interval.cz

[2] Behún, D.: Lingea Grammaticon – přísný strážce jazyka českého. *Interval.cz*, 29. 8. 2005, ISSN 1212-8651, [cit. 2007-05-09].
URL http://interval.cz

[3] Hajič, J.; Hladká, B.: Probabilistic and rule-based tagger of an inflective language – a comparison. 1997.
URL http://citeseer.ist.psu.edu/hajic97probabilistic.html

[4] Lingea: *Grammaticon, uživatelská příručka*. [cit. 2007-04-30].
URL http://www.preklady.cz/download/prirucka.pdf

[5] Mráková, E.; Sedláček, R.: From Czech Morphology through Partial Parsing to Disambiguation. *Computational Linguistics and Intelligent Text Processing, Springer-Verlag*, 2003: s. 126–135.

[6] Naber, D.: *A Rule-Based Style and Grammar Checker*. Diplomová práca, Technische Fakultät, Universität Bielefeld, 2003, [cit. 2007-04-12].
URL
www.danielnaber.de/languagetool/download/style_and_grammar_checker.pdf

[7] OpenOffice: About Us. [online], [cit. 2007-04-12].
URL http://about.openoffice.org/index.html#history

[8] Sedláček, R.: *Morfologický analyzátor češtiny*. Diplomová práca, Masarykova Universita, Fakulta Informatiky, 1999.
URL http://nlp.fi.muni.cz/projekty/ajka/ajkacz.htm

[9] Skiena, S. S.: *Finite State Machine Minimization*. [cit. 2007-05-09].
URL http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK5/NODE207.HTM

[10] Wikipédia: Grammar checker. [online], [cit. 2007-04-16].
URL http://en.wikipedia.org/wiki/Grammar_checker

[11] Wikipédia: Trie. [online], [cit. 2007-04-16].
URL http://en.wikipedia.org/wiki/Trie

[12] Wikipédia: Typologická klasifikace jazyků. [online], [cit. 2007-04-16], Výraz: Typologická klasifikace jazyků.
URL http://cs.wikipedia.org/