Obrazac PD-6/2002

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Robert Lujo

Lociranje sličnih logičkih cjelina u tekstualnim dokumentima na hrvatskome jeziku

MAGISTARSKI RAD

Zagreb, 2010.

Magistarski rad je izrađen u

Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave
Fakulteta elektrotehnike i računarstva

Mentor: Prof.dr.sc. Nikola Bogunović

Magistarski rad ima: 137 stranica

Magistarski rad br.:

Sadržaj

1 Uvod

U današnje je vrijeme pretraživanje informacija jedna od vrlo čestih potreba modernog čovjeka.

Zadatak ovog rada je bio osmisliti i implementirati sustav koji bi omogućio pretraživanje tekstova na hrvatskom jeziku po kriteriju sličnosti tekstova (engl. *text similarity*). Zadatak bi se sveo na sljedeće:

1. opojavničenje (engl. *tokenization*)
2. normalizacija teksta
3. izrada indeksa teksta
4. primjena algoritama za *sličnost* prilikom pretraživanja teksta

Normalizacija teksta [1] je dosta širok pojam koji govori o tome da se tekst pretvori u neki drugi oblik koji je pogodan za neku vrstu računalne obrade. U ovom slučaju oblik treba biti pogodan za potrebe pretraživanja. Proces normalizacije za potrebe pretraživanja bi imao sljedeće zadaće:

1. pretvaranje teksta u istu kodnu stranicu [2] - npr. *unicode* [3]
2. izbacivanje riječi bez veće sadržajne vrijednosti izvan pretražnog prostora - zaustavne riječi
3. svođenje različitih oblika iste riječi (flektivni oblik) na zajednički oblik

Tekst koji je ovako pripremljen optimalan je za potrebe pretraživanja. Zadatak 1) nije problematičan, no zadaci 2) i posebno 3), kad je riječ o hrvatskom jeziku za potrebe ovog projekta izgledali su posebno zahtjevni [4].

U vrijeme kad se osmišljavao sustav (veljača 2009.) nije bilo moguće naći dovoljno kvalitetan skup *zaustavnih rije*či za hrvatski jezik. Za ideju o izradi vlastitog skupa zaustavnih riječi napravljena je analiza, testiranje i prototip. To je pokazalo da je taj zadatak ostvariv u razumnom roku, pa je izabrano to rješenje.

1 *http://en.wikipedia.org/wiki/Text_normalization*

2 *http://en.wikipedia.org/wiki/Code_page*

3 *http://en.wikipedia.org/wiki/Unicode*

4 Hrvatski jezik je morfološki, odnosno infleksijski bogat jezik, tj. za istu riječ možemo naći velik broj različitih oblika.

Postoje dva glavna pristupa rješavanju problema svođenja riječi na osnovni oblik:

- postupak korjenovanja riječi *(engl. stemming process)*
- proces lematizacije.

Za potrebe pretraživanja, rezultat oba procesa je isti: za više različitih oblika iste riječi dobiti oznaku koja označava da ti različiti oblici riječi pripadaju nekoj istoj riječi/entitetu.

Korjenovanje *(engl. stemming process)*[5] način svodi riječi na nekakav zajednički oblik, odnosno ne teži morfološkoj ispravnosti i svođenju na pravi osnovni oblik, dok proces **lematizacije** [6] teži morfološkoj ispravnosti dobivenog osnovnog oblika, koji se još naziva i korijenski oblik ili **lema** [7].

Primjer: riječi *ruka*, *ruci*, *rukama* oblici su imenice *ruka* koje je potrebno svesti na zajednički oblik, tako da sva tri oblika budu označena istim oblikom. U slučaju lematizacije bio bi to pravi morfološki osnovni oblik - *ruka*, a u postupku korjenovanja to bi bila, recimo, *riječ* "ruk".

Od gotovih rješenja i resursa vezanih uz rješavanje problematike koji su nađeni (veljača 2009.) izdvajaju se sljedeći:

- Hrvatski morfološki leksikon (*http://hml.ffzg.hr/hml/*) - koji omogućuje lematizaciju jedne riječi ili veće količine teksta i obratno, iz leme dobiti oblike. Sustav nije za slobodnu upotrebu i ne odgovara u potpunosti potrebama ovog projekta. Radovi objavljeni u vezi s razvojem ovog projekta bili su korisni.
- Cadial projekt (*http://www.cadial.org/*) - projekt koji između ostalih stvari ima modul za morfološku normalizaciju za hrvatski jezik. Premda je programski dio projekta zatvorenog tipa (nije za slobodnu upotrebu), radovi objavljeni u vezi s razvojem ovog projekta bili su korisni.
- rad "Retrieving Information in Croatian: building a simple and efficient rule-based stemmer" [1] daje opis pristupa izrade programskog rješenja za ovu problematiku, što je nađeno dosta zanimljivim i korisnim. Opisani prototip programskog rješenja nije bio raspoloživ.
- "NN: pretraživač za hrvatski" (*http://www.rot13.org/~dpavlin/nn.html*) -

5 *http://en.wikipedia.org/wiki/Stemming*

6 *http://en.wikipedia.org/wiki/Lemmatization*

7 *http://en.wikipedia.org/wiki/Lemma_%28linguistics%29*

dio ovog mini projekta je *stemmer* za hrvatski jezik. Pristup je zanimljiv, kôd je bio dostupan, no sve zajedno bilo je neodgovarajuće za potrebe ovog projekta.

- rad "Automatic Acquisition of Inflectional Lexica for Morphological Normalisation" [2] koji opisuje izgradnju morfološkog leksikona na osnovi rječnika dobivenog iz korpusa za potrebe normalizacije teksta.
- rad "Postupci morfološke normalizacije u pretraživanju informacija i klasifikaciji teksta" [3] vezan je uz problematiku morfološke normalizacije.

Na internetu postoji veliki broj objavljenih članaka koji su izravno ili neizravno vezani uz ovu temu, no nije nađeno gotovo programsko rješenje za slobodnu upotrebu koje bi bilo u mogućnosti obaviti kvalitetnu normalizaciju (u smislu svođenja riječi na osnovni oblik) teksta na hrvatskom jeziku. U poglavlju *Popis poveznica vezanih uz normalizaciju tekstova na hrvatskom jeziku* mogu se naći poveznice na ostale resurse.

Prihvaćen je izazov i odlučeno je da se napravi vlastito programsko rješenje koje će imati zadaću svoditi oblike riječi na osnovni oblik procesom lematizacije. Pritom će se izraditi vlastiti morfološki leksikon [8], koji će biti osnova za proces lematizacije. Za taj će se zadatak formirati vlastiti korpus iz kojeg će se kreirati katalog riječi zajedno s informacijama o njihovim učestalostima (*frequency*).

Za potrebe pretraživanja po sličnosti odabran je način da se u procesu opojavničenja (engl. *tokenization*) ne izdvajaju samo pojavnice *riječi* (*engl. token*), već da proces opojavničenja omogući prelamanje teksta na odjeljke i rečenice, što u ovom slučaju omogućuje jednostavnije i preciznije pretraživanje po mjeri sličnosti tekstova.

1.1 Pregled rada

Ovaj je rad podijeljen po poglavljima koja otprilike odgovaraju svim potrebnim zadaćama koje je potrebno riješiti za ostvarenje cilja:

- da bi se mogao napraviti morfološki leksikon potrebno je skupiti fond riječi što predstavlja prvi zadatak - poglavlje *Korpus;*
- tekst je potrebno opojavničiti na odjeljke, rečenice i riječi. To će se spremiti u bazu riječi - budući *morfološki leksikon* i bazu knjiga (indeksirane knjige). O tome je riječ u poglavlju *Opojavničenje na odjeljke, rečenice i riječi*

8 *http://en.wikipedia.org/wiki/Morphology_%28linguistics%29#Lexical_morphology*

- pomoćne riječi *biti* i *htjeti*, zatim *modalne* riječi morati, trebati, željeti itd., nepromjenjive riječi kao što su prilozi, prijedlozi, usklici itd. proglašene su *zaustavnim riječima (engl. stopwords)*. A kako su dobivene govori navedeno poglavlje. Time je napravljen prvi korak u smanjenju pretražnog prostora;
- drugi i dosta zahtjevniji dio smanjenja pretražnog prostora je *Svođenje riječi na osnovni oblik* o čemu je riječ u navedenom poglavlju. Izrađen je cijeli podsustav koji je u mogućnosti sa željenom opsežnošću, a time i preciznošću, iz zadanog skupa riječi *izvući* osnovni oblik riječi, vrstu te riječi, te neke druge parametre. Preciznost i točnost ovisi o samom fondu riječi i koliko duboko sustav treba pretražiti moguće kombinacije;
- *Pretraživanje po sličnosti* je zadnje poglavlje koje primjenjuje tehnike smanjenja prostora traženja, kao jedinice koristi rečenice, te koristi algoritme za pretraživanje po sličnosti za dobivanje optimalnih rezultata.

Izrađeno rješenje nije konačno rješenje, već je jedna vrsta pilot projekta. Ovo je prva iteracija u kojoj je bio glavni cilj doseći željene funkcije, a što se planira za sljedeće faze napisano je, uz kratki pregled rada, u poglavlju *Zaključak*.

U poglavljima *Dodatci* i *Prilozi* mogu se naći neke dodatne informacije vezane uz rad.

2 Model podataka

U ovom će se poglavlju opisati tehnički dio implementacije, točnije rečeno njegov kostur - model podataka. Namjera je da se na ovaj način odmah prezentiraju neke od implementacijskih odluka koje će se u sljedećim poglavljima spominjati i koristiti.

Potrebno je osmisliti podatkovni dio sustava čija će zadaća biti držati podatke o sljedećem:

- morfološki leksikon - osnovne riječi i svi njihovi oblici
- indeksirani korpus - odnosno *knjige* gdje je sadržaj u obliku skupa riječi koje su samo poveznice prema rječniku unutar morfološkog leksikona

Morfološki leksikon sadrži sljedeće podatkovne klase objekata (u daljnjem se tekstu spominje još kao i podatkovni model ili skraćeno samo **model**):

klasa	***opis***
Word	Riječ, tj. pojam koji se pojavio u nekoj od indeksiranih knjiga. Ta se riječ u kasnijoj fazi preko WordForm povezuje na osnovnu riječ (WordObj klasu) i ukoliko se pojavljuje kao izvedeni oblik više različitih osnovnih riječi onda se samo jedna od njih bira kao *glavna*. Više će se pojedinosti dati u poglavlju *Svođenje riječi na osnovni oblik*.
WordObj	Osnovna riječ, tj. objekt riječ. Ova klasa ima svoju vrstu, svoj osnovni oblik, popis oblika riječi, razne druge parametre i statističke podatke koji se koriste/pune kroz sustav
WordForm	Tablica služi kao pomoćna tablica za ostvarenje više-više relacije između WordObj i Word klasa (*Many-to-many*). Jedna osnovna riječ može i obično ima više oblika riječi a jedna riječ može biti u više oblika različitih osnovnih riječi

Indeksirani korpus sadrži sljedeće podatkovne modele:

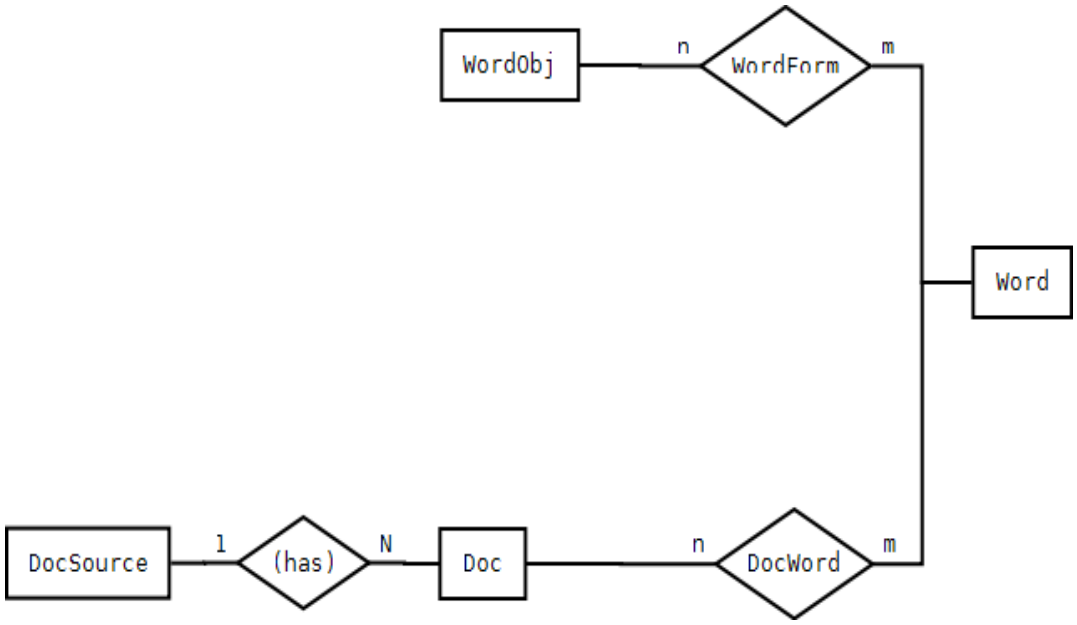
klasa	***opis***
DocSource	To je model koji predstavlja izvor podataka sa svim svojim karakteristikama (izvor, format, parametri procesiranja). To su izvori podataka iscrpnije opisani u poglavlju *Postojeći izvori i izbor*. Jedan DocSource ima jednu ili više Doc objekata - knjiga / dokumenata.
Doc	Predstavlja jedan dokument / knjigu koja je indeksirana. Ona ima svoj izvor (DocSource) i sadržaj kao skup relacija prema Word modelu preko DocWord više-više relacije
DocWord	To je sam sadržaj jednog dokumenta, i pomoćna je tablica za vezu Doc <-> Word više-više relaciju (*Many-to-many*). Osim same relacije posjeduje neke druge podatke o samoj riječi koja se pojavila unutar dokumenta. Zbog očekivanog problema u performansama sustava ova je tablica napravljena u distribuiranom obliku - tako da svaka knjiga ima svoju vlastitu *bazu* kao posebnu datoteku na disku. Pojedinosti o tome mogu se naći u *Opojavničenje na odjeljke, rečenice i riječi*
DocWordOccured	To je sumarna tablica [9] nastala nad tablicom DocWord i daje podatak koja riječ se pojavljuje u kojem dokumentu i s kojom učestalošću (*frequency*)

U poglavlju *Iscrpniji pregled modela podataka* mogu se naći sve potrebne dodatne pojedinosti vezane uz model podataka.

9 *summary table*, negdje *materialized table* ili slično

2.1 ER dijagram

Na sljedećoj slici može se vidjeti međuodnos modela, tj. entiteta u ER terminologiji:



2.2 Primjer

Ovim će se primjerom malo približiti način na koji je osmišljen proces opojavničenja (engl. *tokenization*), odnosno proces spremanja informacija u bazu.

Imamo izvor podataka *Narodne novine*. Iz tog je izvora povučeno 2187 dokumenata. Svaki je dokument html datoteka, koje su smještene na disku u direktoriju D:\KORPUS\NN. Dokumenti su po poddirektorijima po godinama (primjer D:\KORPUS\NN\sluzbeni\2009\00001_2009_01_1_1.html).

U ovom slučaju u model DocSource spremamo jedan zapis

DocSource polje	*vrijednost*
id	5
name	Narodne novine
root	D:\KORPUS\NN
is_recursive	da
filter	*.html
preprocessor	html
doc_count	2187

U modelu Doc spremit ćemo 2187 zapisa, a primjer dokumenta D:\KORPUS\NN\sluzbeni\2009\00001_2009_01_1_1.html bio bi ovakav:

Doc polje	*vrijednost*
id	226
source	5
status	NU1
url	D:\KORPUS\NN\sluzbeni\2009\00001_2009_01_1_1.html
preprocessor	html
codepage	cp1250
-	**Statistička polja**
url_size	58918
url_modified	2009-03-27 01:27:02
last_update	2009-07-01 00:10:02
duration	0:02
cnt_tok_all	8161
cnt_tok_wrd	6597

U modelu Word trebale bi se nalaziti sve riječi koje su se pojavile u ovom dokumentu, pa bi se na primjer riječ *zakonske* ubacila ovako:

Word polje	***vrijednost***
id	190336
value	zakonske
valtype	WRD
status	U
freq	173
wordobj_main_id	128823 [10]

A sam sadržaj dokumenta bio bi ostvaren pomoću tablice DocWord. Na primjer, pojavljivanje riječi *zakonske* u dokumentu iz primjera na lokaciji 237 zapisalo bi se ovako:

DocWord polje	***vrijednost***
doc	226
word	190336
location	237
struct_tag	-

Sumarna tablica DocWordOccured sadržavala bi sumarne tj. statističke podatke za modele Doc i Word. Stoga bi za ovaj primjer riječ *zakonske* koja se pojavljuje 10 puta u ovom dokumentu, prvi put na lokaciji 237, a zadnji put na lokaciji 2035 zapis izgledao ovako:

DocWordOccured polje	***vrijednost***
doc	226
word	190336
first_location	237
last_location	2035
freq	10

Nakon procesa prepoznavanja utvrđeno je da je riječ *zakonske* jedan od mogućih oblika osnovne riječi *zakonski*. Osnovna riječ *zakonski* pridjev je koji ima 11

10 Ovaj se podatak puni nakon procesa prepoznavanja osnovnih riječi i njihovih oblika. U ovom je primjeru prepoznato da je ova riječ samo oblik od osnovne riječi zakonski, što će biti objašnjeno iscrpnije u nastavku.

različitih oblika. Za svih 11 oblika, kao i za oblik *zakonske* osnovna riječ *zakonski* proglašena je glavnom. Broj dokumenata u kojima se nalazi bilo koji od oblika ove osnovne riječi je 224.

WordObj polje	***vrijednost***
id	128823
value_base	zakonski
word_type	ADJ
subtype	-A/-N/-C/ijiS
freq_type	-
status	U
detect_spec	SRI(u'skoga', u'ADJ-od2_iji_-_SKI-skoga-0')
key	ADJ|zakonski|-A/-N/-C/ijiS
-	**Statistička polja**
sum_forms_freq	768
cnt_forms_freq	11
sum_main_freq	768
cnt_doc	224

I na kraju sam podatak da je riječ *zakonske* samo jedan od oblika osnovne riječi *zakonski* pohranjen je u tablici WordForm ovako:

WordForm polje	***vrijednost***
wordobj	128823
word	190336

3 Programski alati

U ovom se poglavlju opisuju najvažniji od izabranih programskih alata pomoću kojih je implementirano programsko rješenje. Iscrpniji pregled ovih i ostalih programskih i razvojnih alata daje se u poglavlju dodatka *Programsko okruženje - pojedinosti*.

3.1 Python

Na osnovu prethodnog pozitivnog iskustva, za programski jezik *poslovne logike* izabran je Python [11]. To se kasnije pokazalo kao dobar izbor, primjerice nije se trebalo baviti programskim jezikom, problemima i nedostacima istog, nije bilo potrebno razmišljati o bibliotekama i sintaksi, već se moglo najveći dio vremena usredotočiti na sam zadatak i programsku logiku. Razvoj je bio brz, rezultati kod promjena kôda mogli su se vidjeti u vrlo kratkom vremenu. Sve u svemu, sve preporuke tom programskom jeziku.

Budući da je sintaksa ovog programskog jezika srževita i nalikuje pseudo-kodu, onda će primjeri i izvadci koda u radu biti *pročišćeni* izvorni programski kod u Pythonu.

U dodatku *Python programski jezik* mogu se naći dodatne informacije o ovom programskom jeziku.

3.2 Sqlite

Sqlite [12] je izabran kao sql baza podataka za ovaj produkt prvenstveno zbog sljedećih razloga:

- nije potrebna nikakva instalacija - jer dolazi kao dio standardnog skupa biblioteka s Pythonom 2.5;
- izravno je podržana od *Django*-a;
- baza je unutar jedne datoteke - jednostavna izrada sigurnosne kopije i prebacivanje;
- baza je transakcijska - vrlo važno u slučaju oporavka od greške;
- podržava veliki dio skupa standardnih SQL naredbi.

11 Programski jezik Python *http://www.python.org http://en.wikipedia.org/wiki/Python_%28programming_language%29*

12 *http://www.sqlite.org*

Prilikom implementacije su se pokazale sljedeće mogućnosti kao korisne:

- omogućuje kreiranje baze u memoriji - što omogućuje vrlo brzu obradu velikih količina podataka;
- moguće je povezati više baza te obavljati SQL naredbe nad takvim povezanim skupom baza - bilo da su baze na disku ili u memoriji;
- nije ograničena na veličinu - premda se najčešće upotrebljava za male baze, nema razloga da se ne upotrebljava u slučajevima kad baze prelaze stotine megabajta - kao što je ovdje bio slučaj;
- omogućena je jednostavna izrada sigurnosne kopije, jednostavno premještanje baze s jednog mjesta na drugo;
- baza je brza - prema iskustvu dobivenim u izradi rada kad god je baza bila spora, to je bilo zbog nedostatka indeksa. Nakon kreiranja indeksa promjena je bila ogromna.

Dodatni podatci o ovoj bazi podataka se mogu naći u dodatku *Sqlite baza podataka*.

3.3 Django

Zbog svih prednosti što ih ima web aplikacija, cilj je bio napraviti GUI dio rješenja kao web aplikaciju. Zbog pozitivnog iskustva u prethodnom profesionalnom radu, za serversku stranu odabran je Django. Django [13] je razvojno okruženje za web aplikacije (*web framework*) za Python programski jezik. Tijekom razvoja Django nije *stajao na putu* već se pokazao kao dobar izbor.

Više pojedinosti naći ćete u dodatku *Django razvojno okruženje za web aplikacije*.

13 *http://www.djangoproject.org*

4 Korpus

Cilj je izgraditi morfološki leksikon za hrvatski jezik. Osnova za takvo nešto je baza riječi, pa je potrebno skupiti fond riječi da bi se na osnovu skupljenih riječi pomoću posebnog algoritma/sustava moglo što točnije naći što više korijena riječi (osnove). Kad se nađu korijeni riječi (*osnovne riječi*) onda se u pretraživanju umjesto korištenja samog oblika koristi osnovna riječ, a time se značajno smanjuje pretražni prostor i na prirodan način normalizira - osnovna riječ nosi za pretraživanje istu sadržajnu vrijednost kao i njen oblik.

Poželjno je da korpus bude što veći i raznolik - da bi se skupio što veći broj različitih riječi sa što većom učestalosti svake od njih.

Tekstovi koji izvorno nisu u txt formatu u prvom će se sljedećem koraku pretvoriti u txt format te prebaciti u bazu podataka. Na taj će se način dobiti registar svih riječi s podatcima o tome gdje se pojavljuju i u kojem broju.

4.1 Postojeći izvori i izbor

Prilikom izbora izvora bilo je potrebno i odabrati koji su ulazni formati poželjni. Odabrani su xml, html i txt formati zbog jednostavnosti izdvajanja samog sadržaja. S druge strane, za formate kao što su npr. pdf i Microsoft Word formati (doc), proces izdvajanja samog teksta dosta je složeniji i zahtijeva dodatne vanjske alate, što usložnjava čitav proces. Sljedeći je razlog taj što se najviše tekstova može naći baš u ta tri formata.

Nakon analize onoga što je javno dostupno na internetu i onog što je prikupljeno iz ostalih izvora izabrano je sljedeće:

naziv	***izvor***
Narodne novine	*www.nn.hr*
Biblija	*www.hbk.hr* + vlastiti prijelom [14]
Bulaja klasici	izdanje na CD-u [15]
ElektronickeKnjige.com	*elektronickeknjige.com*
blog.hr	*blog.hr*
Vjesnik	*www.vjesnik.hr*
groups.google.com /hr/*	*groups.google.com*

4.2 Implementacija i opis problema

Cilj je povući tekstove na hrvatskom jeziku na lokalni sustav u txt ili html formatu.

Što se tiče izvora s interneta, dva su načina na koji su povlačeni sadržaji izvora:

- http protokolom povlačenje i snimanje HTML sadržaja
- http protokolom povlačenje i snimanje *RSS feed*-ova [16].

Svaki od izabranih izvora na internetu ima sljedeće karakteristike:

- ima zajednički početni URL [17]
- prati određenu strukturu i organizaciju dokumenata.

Za svaki je izvor organizaciju i strukturu dokumenata potrebno analizirati i otkriti, te onda na određen način prenijeti u programski kod.

U sljedeća dva podpoglavlja ukratko će se objasniti svaki od navedenih načina.

14 Izvorni format je html no prebacivanje u tekstovni oblik nije napravljen na način kako je to napravljeno za ostale izvore. Više pojedinosti u *Biblija*.

15 CD izdanje od Bulaja naklada d.o.o. Više pojedinosti u *Bulaja klasici.*

16 *http://hr.wikipedia.org/wiki/RSS*

17 *http://hr.wikipedia.org/wiki/URL*

4.2.1 Prebacivanje HTML sadržaja s interneta

Za html iskorištena je logika [18]:

```
import urllib2
for link in link_list:
    http_connection = urllib2.urlopen(link)
    html_content = http_connection.read()
    output_file_path = get_output_file_path(link)
    file(output_file_path, "w").write(html_content)
```

Kao što se može primijetiti, Python ima ugrađenu biblioteku za komuniciranje http protokolom (urllib2) koja omogućuje povlačenje sadržaja metodom *read()*.

Ovaj je problem bio dvojak:

1. poveznice na dokumente spremljene su na jednoj ili više stranica pregled sadržaja (*contents*)
2. može se doznati logika imenovanja i smještaja dokumenata.

Za problem 1) rješenje je snimiti stranice pregleda sadržaja. Pomoću makroa tekstovnog editora ili u nekim slučajevima i manjeg programskog koda izvučene su samo poveznice na stranice, koje su potom snimljene u tekstovnu datoteku. Tada se pokreće glavna logika u modu *čitaj poveznice iz tekstovne datoteke*, te program za svaku od njih povlači sadržaj i snima na lokalni sustav kako je već gore opisano.

Za problem 2) rješenje je bilo da se glavna logika proširi programskom logikom koja dohvaća poveznice na način kako su organizirane na internetskom izvorniku. Program iterira jednu po jednu i zove glavnu logiku. U tom slučaju način na koji su dokumenti izvorno smješteni na internetskom izvoru koristi se i kod smještanja sadržaja na lokalnom sustavu. Primjerice *Narodne novine* imaju strukturu po godinama i mjesecima na *http://www.nn.hr*, a tako se isto snima na disku.

4.2.2 Povlačenje RSS sadržaja s interneta

RSS feed nešto je noviji način posluživanja sadržaja na internetu. RSS je XML jezik kojem je namjena da poslužuje samo kratki pregled odabranih sadržaja s nekog internetskog izvora. U samom *dokumentu* koji se poslužuje (koji je u RSS XML formatu) postoji popis *najsvježijih* [19] izvadaka HTML dokumenata koje to internetsko

18 Kao što je objašnjeno u poglavlju *Python* svi primjeri u ovom radu će biti u Python programskom jeziku

19 RSS kriterij *najsvježijih* izvadaka je određen po vlastitoj logici. To su nekad najnoviji dokumenti,

mjesto poslužuje. Izvadak sadrži podatak o poveznici cjelovitog sadržaja (HTML), naslov, te obično ima i kratki izvadak - obično početak, može imati ključne riječi ili oznake (*keywords*, *tags*), te autora, datum i slične metapodatke.

Za potrebe ovog rada korištena je biblioteka feedparser [20] koja omogućuje čitanje *RSS feed*-ova, te pretvara elemente RSS XML-a u programske objekte.

Glavni kod za čitanje *RSS-feed*-ova je sljedeći:

```
import feedparser, codecs

def get_content_iter(url):
    rss_document = feedparser.parse(url)

    # Loop over all the entries
    for e in rss_document.entries:
        summary = None
        if 'content' in e and len(e.content) and "value" in
e.content[0]:
            summary = e.content[0].value
        if summary is None and 'summary' in e:
            summary = e.summary
        if summary is None:
            summary = e.description
        id = e.get("id", None)
        ...
        link = None
        if "href" in e.links[0]:
            link = e.links[0].href
        # NOTE: html content is title and summary
        html = e.title+'\n\n'+summary
        # Remove all the HTML tags
        txt = clean_tags(html)
        yield id, link, tags, updated_parsed, d.feed.title,txt

def download_rss_content(fname_links, fname_out):
    file = codecs.open(fname_out, "w", "utf-8")
    for feedurl in file(fname_links):
        for id, link, cnt_feeds, tags, updated, title, txt in
get_content_iter(feedurl):
            file.write(u"url: %s\ntitle: %s\nsummary: %s\n\n" %
(feedurl, title, txt))
    file.close()
```

Ovaj primjer ima za ulaz datoteku s popisom poveznica, a za svaku se poveznicu zove *get_content_iter()* funkcija koja povlači RSS sadržaj, te svaki RSS izvadak

nekad najzanimljiviji po vlastitom izboru, nekad najtraženiji ili kombinacija ili slično.

20 *http://www.feedparser.org/*

dokumenta šalje natrag u pozivatelja (*for* petlja), koji to sve zajedno snima u izlaznu datoteku (fname_out).

Budući da je RSS XML format, koji ponekad u sebi sprema izvatke u jednostavnom HTML formatu, onda je napravljeno *čišćenje* sadržaja, tako da se cijeli sadržaj odmah sprema u tekstovnom formatu. Izlaz procesiranja je jedna datoteka koja ima sve izvatke u sebi na način da za svaki izvadak sadrži naslov i ispod čisti tekst.

S obzirom da su skoro svi RSS izvori bili u kodnoj stranici utf-8 [21], onda je i sadržaj potrebno snimiti u tom istom formatu. Za tu se potrebu koristi ugrađena Python biblioteka *codecs*.

Problematika vezana uz dohvat i organizaciju početnih poveznica analizirana u prethodnom poglavlju *Prebacivanje HTML sadržaja s interneta* pojavila se i slučaju RSS-a, te je primijenjen isti način rješavanja.

Potrebno je dodatno objašnjenje za naredbu *yield*. To je ključna riječ u Pythonu koja omogućuje izradu funkcija iteratora. Kreiranje funkcija iteratora jedna je od mogućih implementacija **coroutine** [22] vrsta programskih komponenti.

U ovom konkretnom primjeru funkcija *get_content_iter()* pozvana je kao iterator funkcija u desnom dijelu *for* petlje, a lijevi dio *for* petlje dobiva rezultat kad god iterator funkcija pozove *yield* funkciju. Kontekst iterator funkcije spremljen je i čeka na novi poziv iz for petlje.

U sljedećim će poglavljima biti više riječi o svakom od pojedinih izabranih izvora.

4.2.3 Narodne novine

Izvorna adresa za sadržaj Narodnih novina je *http://narodne-novine.nn.hr/default.aspx*. Sadržaj dokumenata je u html formatu. Sadržaj je podijeljen na dva dijela:

1. međunarodni - *http://narodne-novine.nn.hr/clanci/medunarodni/*
2. službeni - *http://narodne-novine.nn.hr/clanci/sluzbeni/*

Jedan HTML dokument je poglavlje u jednom izdanju.

21 O kodnim će stranicama više biti riječi u poglavlju vezanom za predprocesiranje u *Opojavničenje na odjeljke, rečenice i riječi.*

22 *http://en.wikipedia.org/wiki/Coroutine*

Oba dijela imaju strukturu dokumenata koja u sebi ima podatke:

- godina i mjesec izdanja
- redni broj izdanju u godini
- redni broj odjeljka u svim izdanjima u toj godini

Primjer jednog poglavlja izdanja je:

- "7. Rješenje o imenovanju savjetnice potpredsjednika Vlade Republike Hrvatske"
- izdanom u Narodnim novinama br.: 22
- dan izdanja 17.02.2010.

Poveznica je *http://narodne-novine.nn.hr/clanci/sluzbeni/2010_02_22_538.html*. U poveznici **2010_02_22_538** je identifikator koji nosi sljedeće informacije:

2010	godina izdanja
02	mjesec izdanja
22	je redni broj izdanja
538	je redni broj poglavlja u svim izdanjima u godini 2010

Implementacija povlačenja sadržaja napravljena je na način da se radi uvijek na osnovi godine, tako da se počinje od prvog izdanja i prvog poglavlja i onda metodom pokušaja i promašaja traži prvi valjani slijedni broj poglavlja ili izdanja. Skraćena logika je:

```
for year in range(from_year, to_year+1):
    article = 0
    volume = 0
    volume_good = article_good = None
    while month<13:
        month += 1
        while True:
            if article_good:
                article = article_good
                volume = volume_good + 1
            else:
                article = 0
                volume+=1
            while True:
                article+=1
                pagename = "%04d_%02d_%d_%d.html" % (year,
                                    month, volume, article)
                try:
```

```
            c=urllib2.urlopen(page)
            content = c.read()
        except Exception, e:
            continue
        volume_good  = volume
        article_good = article
```

Potrebno je napomenuti da je u kod ugrađena logika da su dopuštena najviše 3 uzastopna *loša* pokušaja, nakon čega program prekida rad. Isto tako, ugrađene su pojedine iznimke nađene za pojedine godine i izdanja. Za potrebe analize snimljene su za službeni dio 1990., 2000., 2007., 2008. i 2009., a za međunarodni 2001., 2002., 2008. i 2009. To ukupno čini 2187 dokumenata.

4.2.4 Biblija

Prvobitni izvor ovog teksta bile su stranice Hrvatske biskupske konferencije, izdanje Biblije od Kršćanske sadašnjosti. Sadržaj je struktuiran na način:

- Stari zavjet - popis knjiga i poveznica:
  http://www.hbk.hr/biblija/sz/index.html
- knjige Starog zavjeta, *http://www.hbk.hr/biblija/sz/*<kod-knjige>.html, npr. za Knjigu postanka je *http://www.hbk.hr/biblija/sz/post.htm*
- Novi zavjet - popis knjiga i poveznica:
  http://www.hbk.hr/biblija/nz/index.html
- knjige Novog zavjeta, *http://www.hbk.hr/biblija/nz/*<kod-knjige>.html. Primjer, za Evanđelje po Mateju je *http://www.hbk.hr/biblija/nz/mt.htm*

U sklopu jednog drugog projekta, taj tekst je analiziran, prelomljen i prefomatiran u tekstovni format sa stupcima jednake širine. Taj je prelomljeni tekst korišten kao ulaz za ovaj rad, pa nije bilo potrebno nikakvo povlačenje s interneta i dodatno procesiranje. Izvadak teksta je:

```
00000200_H1_sz_post  _  0_  0.000_Knjiga Postanka
00000500_H2_sz_post  _  0_  0.000_I. POČECI SVIJETA I ČOVJEČANSTVA
00000800_H1_sz_post  _  0_  0.000_1. STVARANJE SVIJETA
00001000_I3_sz_post  _  0_  0.000_Prvi izvještaj o stvaranju
00001300_S _sz_post  _  1_  1.000_U početku stvori Bog nebo i zemlju.
00001400_S _sz_post  _  1_  2.000_Zemlja bijaše pusta i prazna; tama se
prostirala nad bezdanom i Duh Božji lebdio je nad vodama.
00001500_S _sz_post  _  1_  3.000_I reče Bog: "Neka bude svjetlost!" I
bi svjetlost.
```

Budući da je sadržaj već na lokalnom sustavu onda nije bio potreban program za

povlačenje i snimanje, no budući da je tekst u posebnom formatu, prilikom opojavničenja na rečenice bilo je potrebno uzeti u obzir specifičnost ovog formata (poseban predprocesor).

4.2.5 Bulaja klasici

Poduzeće Bulaja naklada d.o.o. [23] izdalo je CD "Klasici hrvatske književnosti, epika, romani, novele" [24]. Taj CD je iskorišten kao jedan od izvora tekstova na hrvatskom jeziku. Razlog uzimanja tog izvora je dostupnost, jednostavnost korištenja, količina tekstova i vrsta - klasična hrvatska književnost.

Tekstovi su dani u više formata od kojih je jedan tekstovni (txt), koji je izabran kao izvor. Za ovaj izvor nije bio potreban program za povlačenje s interneta, niti predprocesor za pretvaranje u txt format. Ovaj izvor ima 63 knjige od 30 autora, sva su djela slobodna za korištenje - s obzirom da su autorska prava istekla (50 godina od objavljivanja). Posebnost ovog izvora je što je većina tekstova dosta stara, no neki su za ovu analizu *prestari* s obzirom da koriste vrlo arhaičan ili čak izumrli dijalekt hrvatskog jezika (primjer Marko Marulić, Judita). To je uzrokovalo dodatni *šum* u analizi i prepoznavanju osnovnih riječi, što je znalo nekad zbuniti sustav. Ipak, sve u svemu to nije bilo toliko značajno da je pravilo velike probleme, ali u nekoj od sljedećih faza bit će potrebno izbaciti takve tekstove u primarnoj analizi i prepoznavanju.

4.2.6 elektronickeknjige.com

Internetske stranice *http://www.elektronickeknjige.com/* projekt su nevladine udruge *Društvo za promicanje književnosti na novim medijima*. Cilj projekta je:

> "... promicanje, prvenstveno, suvremene hrvatske književnosti i povećanje dostupnosti književnih djela suvremenih hrvatskih autora. Posebno je važno istaknuti sljedeća tri cilja. Kao prvo, trenutna dostupnost: knjige su trenutno dostupne svim zainteresiranim korisnicima interneta. Kao drugo, besplatni pristup: knjige su dostupne bez naknade, ... Kao treće, povećanje raznovrsnosti: moguće je objaviti i naslove koji teže nalaze put do komercijalnih izdavača, ... Često je riječ o neafirmiranim mladim autorima s čijim se naslovima izdavači rijetko usuđuju ulaziti u poslovni rizik, ..."

Razlog uzimanja ovog izvora je dostupnost, jednostavnost dostupa, količina tekstova

23 *http://www.bulaja.com* Bulaja naklada d.o.o.

24 U istoj seriji izdani su i CD-i "Klasici hrvatske književnosti, drama i kazalište" i "Klasici hrvatske književnosti, pjesništvo", no nakon analize odbacio su odbačeni zbog nedovoljne prikladnosti problematici koja se rješava.

i vrsta - moderna hrvatska književnost.

Sam izvor u trenutku povlačenja imao je 95 knjiga / izdanja, od 71 autora. Dokumenti su u HTML formatu. Dohvat dokumenata išao je na način da su se sa stranice sadržaja *http://www.elektronickeknjige.com/elektronicke_knjige/kazalo_naslova.php* izvukle sve poveznice na sadržaje svake knjige i snimile u tekstovnu datoteku (npr. *http://www.elektronickeknjige.com/bencic_rimay_tea/sipine_kosti/index_page_000.htm*). Svaka stranica sadržaja ima poveznice na poglavlja knjige. Sva poglavlja prate strukturu brojača, tako da je za zadnji primjer knjige sav sadržaj na stranicama:

- od /bencic_rimay_tea/sipine_kosti/pages/002.php
- itd.
- do /bencic_rimay_tea/sipine_kosti/pages/037.php

Za povlačenje svih poglavlja napravljena je posebna programska logika koja iterira i povlači sva poglavlja koja postoje.

4.2.7 groups.google.hr

Internetski servis groups.google.com agregator je velikog broja *Usenet News groups* [25], od kojih su za rad bile zanimljive sve hr.* grupe. groups.google.com omogućuje povlačenje zadnjih poruka preko *RSS*-a (pojedinosti u *Povlačenje RSS sadržaja s interneta*). Maksimalan broj poruka je 100, a ono što dodatno usložnjava proces je međustruktura *thread*, odnosno *news* teme. Ovaj izvor je izabran zbog toga što sadrži *govorni* hrvatski jezik, no dodatni problem predstavlja što takav jezik pri komunikaciji na internetu ima svoje posebnosti, bolje rečeno nepravilnosti, npr.:

- početak rečenice malim slovom,
- samo mala slova, samo velika slova,
- nepostojanje interpunkcije,
- česta upotreba skraćenica,
- nepravilne riječi.

Od svih 527 hr.* grupa ručno je napravljen filter i povučeni su podatci s 415 njih, u oko 14.500 direktorija (svaki direktorij jedan *thread*), te sadržaja oko 625 MB.

25 *http://en.wikipedia.org/wiki/Usenet_newsgroup*

4.2.8 blog.hr

Internetske stranice na *http://www.blog.hr/* nakon prijave nude mogućnost pisanja vlastitih *blog* [26] članaka. Sadržaj je povučen preko *RSS*-a, na način da su povučeni linkovi na stranice najčitanijih autora (195), te za njih pomoću RSS-a zadnje objavljeni tekstovi.

4.2.9 Vjesnik

Internetske stranice Vjesnik (www.vjesnik.hr) bile su jedini izvor dokumenata dnevnih novina koji je imao arhivu na struktuiran način i u html obliku. Napravljena je analiza i napravljen dohvat tekstova naslovnica za svaki dan u 2008. godini. Struktura poveznice je:

```
http://www.vjesnik.hr/html/<godina>/<mjesec>/<dan>/<godina>_<mjesec>_<da
n>.html
```

Za dohvat je korištena logika iz *Prebacivanje HTML sadržaja s interneta*, s time da je napravljen dio programske logike koji iterira po danima i kreira poveznice prema opisanoj strukturi.

26 *http://hr.wikipedia.org/wiki/Blog*

5 Opojavničenje na odjeljke, rečenice i riječi

Cilj ovog dijela zadatka je prikupljene tekstove prebaciti u txt format, ukoliko već nisu, te ih opojavničiti na odjeljke, rečenice i riječi [27]. Nakon toga je opojavničeni tekst (engl. *tokenization process*[28]) potrebno snimiti u bazu – pojavnicu po pojavnicu (engl. *token*), na način kako je to predviđeno u modelu (pogledati *Model podataka*).

5.1 Predprocesiranje

Za sve one dokumente koji nisu pohranjeni u čistom txt formatu potrebno je obaviti predprocesiranje koje ima sljedeće zadaće:

- da se sadržaj pretvori u txt format (html, *Biblija* poseban format);
- po potrebi grupiranje pojedinih dokumenata u logičke cjeline. Primjer, svaka knjiga iz izvora *elektronickeknjige.com* razbijena je po poglavljima u posebne html datoteke, pa će se one u postupku predprocesiranja gledati kao jedan cjeloviti tekst i na taj način snimiti u bazu;
- svi se tekstovi pretvaraju u unicode - što znači da moramo znati ili na neki način detektirati u kojoj se kodnoj stranici nalaze izvorni tekstovi.

U opisu modela DocSource u tabličnom je obliku dan popis svih predprocesora.

Od svih dijelova predprocesiranja, jedini koji zaslužuje veću pozornost je Html predprocesor. Html predprocesor pretvara html sadržaj u tekst na sljedeći način:

```
import BeautifulSoup as bp

html_content = prune_tags(html_content)
html_content = convert_to_unicode(html_content)
soup=bp.BeautifulSoup(html_content)
txt_content = convert_html2txt(soup)
```

27 Prvenstveno zbog ograničenosti vlastitih računalnih resursa nisu opojavničeni svi dokumenti iz korpusa već je napravljena dodatna selekciju. Izabrani su sljedeći izvori: *Narodne novine*, *Biblija*, *Bulaja klasici* i *elektronickeknjige.com*.

28 *http://en.wikipedia.org/wiki/Lexical_analysis* (engl.): A token is a string of characters, categorized according to the rules as a symbol (e.g. IDENTIFIER, NUMBER, COMMA, etc.). The process of forming tokens from an input stream of characters is called (tokenization) and the lexer categorizes them according to a symbol type.

Funkcija *prune_tags()* radi sljedeće:

- čisti CDATA dijelove
- uklanja sve atribute iz početnih tag-ova, završnih ili početno-završnih (npr.
)
- uklanja sve blokove komentara, npr. <!-- komentar -->
- uklanja tag-ove i sav njihov sadržaj koje ne nose tekstovni sadržaj iz sljedećeg popisa: *script, link, style, object, meta*
- uklanja tag-ove ali ostavlja sadržaj za *span* tag
- uklanja posebne tag-ove kao npr. <![if !something]> <![endif]>
- uklanja tagove, i ostavlja sadržaj, one koji imaju *namespace* informaciju kao npr. <o:p ... >

Uloga funkcije *convert_to_unicode()* je da se svaki sadržaj, bez obzira na kodnu stranicu u kojoj izvorno jest, pretvori u unicode [29]. Na taj se način svi tekstovi zapisuju na isti način (unicode), te se nakon pretvorbe opojavniče (engl. *tokenization*) i spremaju u bazu. Na kraju je sav sadržaj u bazi u unicode-u. To ne vrijedi samo za tekstove u html formatu, već za sve tekstove. Konkretno za detekciju izvorne kodne stranice html dokumenta, ona se vrši u sljedećem redoslijedu:

- ako je sam sadržaj u unicode formatu, nema potrebe za pretvorbom
- ukoliko ulazni parametar u funkciju ima podatak o kodnoj stranici on se uzima (u modelu taj podatak stoji u Doc.codepage)
- pokušava se iz strukture HTML stranice doznati kodna stranica (atribut *charset*)
- koristi se biblioteka *chardet* [31] za detekciju - zbog brzine za detekciju uzima se prvih 5000 i zadnjih 5000 znakova dokumenta

Nakon što se html sadržaj *pročisti* i pretvori u *unicode*, potrebno ga je prebaciti u txt format. Za to se koristi python biblioteka *BeautifulSoup* [32]. Algoritam koji se koristi u biblioteci dovoljno je robustan da može podnijeti i *loš* html, a rezultat su hijerahijski vezani programski objekti (slično kao DOM). Logika rekurzivnog koda koji je

29 Sve operacije nad tekstovnim podatcima obavljaju se u *unicode*-u, dok se u bazu podataka spremaju u *utf-8* kodnoj stranici. Zbog toga je potrebno sve tekstovne podatke prije obrade pretvoriti u *unicode.*

31 *http://chardet.feedparser.org/*

32 *http://www.crummy.com/software/BeautifulSoup/*

korišten je sljedeći:

```
def convert_html2txt(soup):
    v=soup.string
    if v==None:
        c=soup.contents
        resulttext=''
        for t in c:
            # RECURSION
            subtext=convert_html2txt(t)
            if subtext:
                resulttext+=subtext+'\n'
        return resulttext
    else:
        v = v.strip()
        if isinstance(soup, (bp.Comment, bp.ProcessingInstruction,
bp.Declaration)):
            return u""
        v = unescape(v)
        if isinstance(soup, (bp.Tag,)):
            if soup.name.lower() in ("h1","h2","h3","h4","p", "title"):
                return "\n\n"+v+"\n\n"
            elif soup.name.lower() in ("script",):
                return u""
    return v
```

Za modul koji radi pretvorbu sadržaja iz html-a u txt format napravljen je doc-testing [30] koji daje dodatnu razinu kontrole kvalitete modula. Primjer doc-test za modul:

```
>>> prune_tags(r'''<span lang=EN-US><![if !supportEmptyParas]> 
                   <![endif]><o:p></o:p></span>''')
' '
```

5.2 Zadaće sustava za opojavničenje

U zadaći prijeloma na odjeljke, rečenice i riječi (opojavničenje, *engl*. *tokenization*), najveći je problem prijelom po rečenicama. Pisani tekst nema stroga pravila kao što ih ima npr. računalni program u npr. Python-u. Uzmimo za primjer prethodnu rečenicu :

> Pisani tekst nema stroga pravila kao što ih ima npr. računalni program u npr. Python-u.

Kako natjerati računalni program da prepozna da je npr. samo skraćenica, da točka iza npr. nije kraj rečenice već dio skraćenice, a da riječ Python nije početak nove rečenice nego ime.

Osim prepoznavanja skraćenica i imena, kraja i početka rečenice, tu postoji još

30 *http://en.wikipedia.org/wiki/Doctest*

problema i zadaća s kojima se modul mora suočiti:

- upravni govor - npr.: "on reče: 'To je bilo nedopustivo.'"
- prepoznavanje brojeva - novčanih, arapskih i rimskih: 100.000,00 ili 3939.00 ili MCXXII
- popisi, tablice i slično
- izoliranje interpunkcijskih znakova: "iza toga, slijedi uskličnik!"
- prepoznavanje kraja odjeljaka i prepoznavanje naslova
- itd.

U sljedećem se poglavlju daje sažeti pregled algoritma koji je upotrijebljen za rješavanje ove problematike.

5.3 Implementacija sustava za opojavničenje

Za potrebe opojavničenja (engl. *tokenization*) napravljen je sljedeći skup klasa:

Tokenizer	Glavna klasa - ulaz je tekst, popis postojećih potvrđenih imena i skraćenica, te dodatni parametri. Kreira listu Token objekata kao izlaz.
Token	Predstavlja jednu pojavnicu (engl. *token*). Svaka pojavnica ima svoju vrijednost, vrstu, podatke o formatu, sadržaju, strukturnu ulogu u tekstu (odlomak, rečenica, itd.).
Name	Objekt je koji predstavlja ime, odnosno riječ koja treba počinjati s velikim slovom. Može biti predefiniran ili detektiran.
Abbr	Objekt je koji predstavlja skraćenicu, odnosno riječ koja treba završavati s točkom. Može biti predefiniran ili detektiran.
ItemList	Objekt je koji sadrži niz imena ili skraćenica, ili omogućuje pristup do njih (npr. ako su u bazi i ima ih puno). Koristi se kod provjere postoji li neki kandidat za ime ili skraćenicu već postoji, te za dohvat takvog objekta.

Za primijetiti je da su 3 klase vezane za problematiku velikog slova i točke.

Logika glavnog programa je:

```
class Tokenizer(object):
    ...
    def tokenize(self, text, cp="utf-8", in_abbr_list=None,
                 in_name_list=None):

        abbr_list = self.prepare_item_list(Abbr, abbr_list)
        name_list = self.prepare_item_list(Name, name_list)

        text = to_unicode(text, cp)
        text = self.preprocess_text(text)

        # ------ SPLIT BY WHITESPACE
        word_list = self.split_text(text)

        # ------ FIRST PASS - preprocess
        word_list = self.preprocess_word_list(word_list)

        # ------ SECOND PASS - tokenize and yield
        current_sentence = []
        ind=-1
        while True:
            ind += 1

            # 'split' word
            word_inner_list = self.split_word(word_list[ind],
                                              abbr_list)
            word_list[ind:ind+1] = word_inner_list

            # last word - yield last sentence
            if ind==len(word_list):
                for i, token in enumerate(current_sentence):
                    yield token
                break # EXIT LOOP

            # take next word
            word = word_list[ind]

            # based on word and context (words around)
            # collect information
            word_ends_with_sentence_end_punctation = word[-1] in (".!?")
            word_next_is_titled = self.get_next_is_punct_title(...)
            word_looks_like_abbr = self.looks_like_abbr(word)
            word_looks_like_roman_nr = self.looks_like_roman_number(
                                                                word)
            word_looks_like_nr = self.looks_like_roman_number(word)
            ...
            # based on collected information detection of
            # start/end of sentence / chapter
```

```
if is_new_sentence:
    for i, token in enumerate(current_sentence):
        yield token

# based on collected information new token is created
# with all needed information
self._add_token(current_sentence, word)
```

Ovo je prilagođeni prikaz algoritma, a modul koji sadrži svu logiku dugačak je oko 1400 redova programskog koda i ima još dodatnih 400 redova doc-testinga.

Funkcija *prepare_item_list()* ulazne liste imena ili skraćenica, ili pristupne objekte (ako je u pitanju baza s puno njih) priprema i pretvara u listu objekata *Name*/*Abbr* i sprema u objekt *ItemList*.

Funkcija *preprocess_text()* radi prvi prolaz kroz tekst, a koristi regularne izraze [33] da bi se tekst pročistio i pripremio na način:

- da se ubace razmaci između nekih riječi i interpunkcijskih znakova: "(naziv)," -> '(', 'naziv', ')', ','
- da se izbace neki znakovi i spoje dijelovi riječi: "po-li-ra-nje" -> 'poliranje'
- da se veći broj oznaka novih redova (*newlines*) označe na poseban način i time označe početak nove rečenice ili novog poglavlja.

Podjela teksta na riječi obavlja se pozivom funkcije *split_text()*, a rezultat se sprema u *word_list* objekt kao lista riječi. Podjela riječi obavlja se pozivom posebne metode Python objekta regularnog izraza *split* po svim nepraznim znakovima (\s).

Sljedeći je korak dodatno preprocesiranje dobivene liste riječi pozivom funkcije *preprocess_word_list()*:

- prebacivanjem niza riječi koje su sve pisane velikim slovima u mala (barem 4 u nizu). Npr. "THIS IS UPPER CASE" -> ['This', 'is', 'upper', 'case']
- "upravni govor" se u pojedinim slučajevima pretvara u neupravni, zbog jednostavnosti implementacije i izbjegavanja primijećenih problema. Npr. 'Andjeo Jahvin je zapita : " Hagaro , sluskinjo Sarajina , odakle dolazis i kamo ides ? -> ['Andjeo', 'Jahvin', 'je', 'zapita', '!', 'Hagaro', ',', 'sluskinjo', 'Sarajina', ',', 'odakle', 'dolazis', 'i', 'kamo', 'ides', '?',]
- izbacivanjem praznih riječi - što ih zna ostaviti *split_text()* funkcija zbog logike algoritma

33 *http://en.wikipedia.org/wiki/Regular_expression*

Nakog toga pročišćeni skup riječi ulazi u glavnu petlju (drugi prolaz) i formiraju se rečenice. Objekt koji drži trenutnu rečenicu je *current_sentence*, a indeks trenutne riječi je u varijabli *ind*. Za trenutnu se riječ poziva funkcija *split_word()* koja unutar riječi pokušava razdvojiti znakove interpunkcije tamo gdje je to potrebno i moguće - potrebno je imati bazu potvrđenih skraćenica. Npr. za slučaj da su *prof.* i *dr.* potvrđene skraćenice onda:

"prof.dr.J.Malkovich" -> ['prof.', 'dr.', 'J.', 'Malkovich']

Nakon toga je sadržaj riječi konačan, tj. ne mijenja se. Sljedeće je odrediti kakve je vrste pojavnica (engl. *token*) temeljena na toj riječi. Pri tome se gleda okruženje riječi (sljedeće riječi), gleda se je li riječ samo slovčana, ili je u formatu broja, rimskog broja, rednog broja, je li riječ o nečem što sliči na skraćenicu, što sliči na ime itd. Na osnovu svih tih informacija kreira se nova pojavnica sa svim potrebnim atributima koje nosi. Ukoliko pojavnica ima oznaku kraja rečenice, onda se cijela rečenica šalje u pozivatelja preko naredbe *yield* - što čini ovu funkciju iterator funkcijom (više o *yield* naredbi pogledati u *Povlačenje RSS sadržaja s interneta*). Nakon toga se pojavnica dodaje u trenutnu rečenicu.

Za prepoznavanje brojeva (rednih, arapskih, rimskih) koriste se regularni izrazi.

Logika određivanja novih imena i skraćenica dosta je stroga, odnosno kao potvrđena imena i skraćenice uzimaju se samo one koje su došle iz pozivatelja, dok se sva novodobivena imena i skraćenice u procesu prelamanja ne upotrebljavaju u daljnjem prelamanju (nemaju status *potvrđeno*).

Objekti klase *Token* sadrži informacije pojavnice o vrijednosti, vrstu, podatke o formatu, sadržaju i strukturnoj ulozi u tekstu (odlomak, rečenica, unutar rečenična itd.):

atribut	***opis***
value	vrijednost riječi
is_sent_start	je li pojavnica u tekstu prva u rečenici
is_sent_end	je li pojavnica u tekstu zadnji u rečenici
known_obj	ako je pojavnica prepoznata kao postojeće ime ili skraćenica, onda ovaj atribut sadrži taj objekt
is_number	vrijednost je prepoznata kao brojčana. (npr. 10, +123.00, -20.000,03, 2007.)
is_romnr	vrijednost je prepoznata kao rimski broj (npr. XIII., MCMVIII)
is_abbr	je li vrijednost prepoznata kao skraćenica
is_name	je li vrijednost prepoznata kao ime
is_inner_sep	unutar rečenična struktura ,:;-&/
is_sent_sub1	unutar rečenična struktura "'`»«”’
is_sent_sub2_s	unutar reč. struktura - početak ([{
is_sent_sub2_e	unutar reč. struktura - kraj)]}
tag	za pojedine potrebe preprocesor može sam *prisiljavati* određene parametre pojavnice - npr. *tag* - početak poglavlja
is_par_start	je li to prva pojavnica u odjeljku (engl. *paragraph*)
is_fuzzy_abbr	vrijednost je prepoznata kao *neobična* skraćenica (npr. d.o.o.)
is_fuzzy_type	vrijednost je *neobična*, odnosno nije niti čista brojčana niti slovčana (npr. abc123)

Sljedeći primjer pokazuje prepoznavanje riječi, rečenica, upotrebu postojećih potvrđenih skraćenica i imena, te prepoznavanje novih kandidata:

```
>>> abbr_list = ItemList(Abbr, [Abbr("Mr.", a_bef_name=True),
Abbr("ing.", a_bef_name=True), Abbr("prof.", a_bef_name=True),
Abbr("dr.", a_bef_name=True)])
>>> name_list = ItemList(Name, [Name("Pero"),
Name("Mato",can_start_sent=True)])
>>> print list(t.tokenize("""This is Mr. Miro , Mato , ing.Pero. Mato je
```

```
jeo obrok. Miro nije jeo danas""",
...                         in_name_list=name_list,
...                         in_abbr_list=abbr_list))
    [T('this'/sent_start), T('is'), T('Mr.'/abbr+known),
     T('Miro'/name), T(','/inner_sep), T('Mato'/name+known),
     T(','/inner_sep), T('ing.'/abbr+known), T('Pero'/name+known),
     T('.'/sent_end), T('Mato'/name+sent_start+known), T('je'),
     T('jeo'), T('obrok'), T('.'/sent_end), T('miro'/sent_start),
     T('nije'), T('jeo'), T('danas'/sent_end)]

>>> list(name_list.iter_only_new())
[N('Miro'/2/new/conf/0), N('This'/1/new)]

>>> list(abbr_list.iter_only_new())
[A('Pero.'/1/new), A('danas.'/1/new)]
```

Ovaj je algoritam prošao niz poboljšanja i prilagodbi, tako da se na kraju pokazao dosta zadovoljavajući. S obzirom na to da se proces izrade zadovoljavajućeg rješenja radio u koracima, odmah na početku uz razvoj programskog koda pisali su se testovi (doc-tests), što se pokazalo kao ključni faktor održanju i poboljšanju kvalitete rješenja.

6 Normalizacija tekstova na hrvatskom jeziku

Ciljevi koji su postavljen ovoj temi su:

- izraditi popis zaustavnih riječi (engl. *stopwords***)**
- svesti riječi na osnovne oblike - lematizacija

Pretpostavka je da imenice, pridjevi i glagoli nose sadržaj teksta, pa se sve ostale vrste riječi uzimaju s puno manjom važnošću – zaustavne riječi (*engl. stopwords*), koje je većim dijelom potrebno ručno evidentirati, a manjim dijelom prema učestalosti u korpusu. A i među odabranim vrstama riječi određene će se riječi odmah u početku uzimati s manjom važnošću (npr. glagol "biti" i svi njegovi oblici).

Ulaz za ovaj zadatak je popis riječi i za svaku od njih mjera učestalosti (frekvencija).

6.1 Kratki pregled gramatike hrvatskog jezika

Osnovna referenca korištena u analizi gramatike hrvatskog jezika je knjiga "Gramatika hrvatskog jezika, Težak, Babić 2005" [4].

Sljedeće su vrste riječi: imenice, pridjevi, zamjenice, brojevi, glagoli, prilozi, prijedlozi, veznici, usklici i čestice. Riječi mogu biti promjenjive ili nepromjenjive.

Ovo je pregled vrsta riječi u hrvatskom jeziku s obzirom na promjenjivost - stupac *kod* označava kod koji je korišten u programskom rješenju:

vrsta riječi	***kod***	***promjenjivost***	***engleski naziv***
imenice	N	DA	nouns
pridjevi	ADJ	DA	adjectives
glagoli	V	DA	verbs
brojevi [36]	NUM	DA	numbers
zamjenice	PRON	DA	pronouns
prilozi	ADV	ne	adverbs
prijedlozi	PREP	ne	prepositions
veznici	CONJ	ne	conjuctions
usklici	EXCL	ne	exclamations
čestice	PART	ne	particles

Svaka vrsta riječi ima imati svoj skup atributa. Atributi mogu biti jedan od sljedećih:

atribut	***moguće vrijednosti***	***kod***	***vrijedn.***
rod	muški, ženski ili neutralni	GENDER	M,F,N
broj	jednina ili množina	NUMBER	S,P
padež	nominativ, genitiv, ..., instrumental (7)	DECLENSION	N,G,...,I
osoba	ja, ti, on, itd.	PERSON	1,2,3
vrijeme	sadašnje, prošlo, buduće, itd.	TIME	PRE,AOR,itd.
komparacija	pozitiv, komparativ, superlativ	COMPARATION	POS,COM,SUP

Stupac *kod* ima nazive koji su korišteni u programskom rješenju, a time i u primjerima koji će se kasnije navoditi u radu. Stupac *vrijednost* popis je prvih vrijednosti atributa koji su korišteni u programskom rješenju, ali i u primjerima ovog rada. Dodatna napomena vezana uz atribut *osoba*, a to je da je u nekim slučajevima potrebno treće lice razlučeno po rodu (posvojne zamjenice). U tom slučaju imamo atribut *PERSON_MFN* s vrijednostima 1,2,3M,3F,3N.

36 Zbog nedostatka vremena, brojevi kao vrsta riječi nisu obrađeni. Analiza je napravljena, slični su drugim promjenjivim riječima, no imaju svoje specifičnosti. Zbog pretpostavke da nose manju sadržajnu vrijednost zanimljivu za pretraživanje, njihova detekcija i analiza odgođeni su za sljedeću fazu.

Za napomenuti je da se gramatički rod ne podudara uvijek sa spolom riječi. Primjer, riječ *gazda* sklanja se (deklinira) po sklonidbi ženskog roda, a u stvarnosti predstavlja osobu muškog spola. Prirodne pojave i stvari u stvarnosti nemaju spola, a nerijetko nisu u neutralnom rodu - npr. čekić je muškog roda.

Za određenu vrstu riječi neki su atributi nepromjenjivi, a neki promjenjivi.

Primjer

Riječ "lopta" je imenica, njen nepromjenjivi atribut je rod, ovo je imenica ženskog roda, a promjenjivi atributi su broj i padež. Jedan oblik riječi "lopta" je "loptama" što je oblik s vrijednostima promjenjivih atributa - broj je množina, padež je instrumental.

Ako vrsta riječi ima barem jedan promjenjivi atribut, onda je to promjenjiva vrsta riječi.

Na koji se način dobivaju oblici promjenjivih riječi određeno je *morfološkim* pravilima, a znanost koja se bavi oblicima riječi se zove *oblikoslovlje* ili *morfologija* [37]. Radi boljeg razumijevanja terminologije, *morfem* je dio riječi koji ima neko značenje. **Infleksija** [38] je vrsta oblikovanja riječi s kojim se ne dobiva novo značenje već **novi oblik iste riječi** (više o tome u *Flektivna pravila*) [39].

U ovom je radu korišten pojam **flektivno pravilo** kao skup pravila za jednu riječ s kojom se dobije skup njenih oblika. Primjer: flektivno pravilo za glagole za dobivanje prezenta glagola.

S druge strane, *tvorba riječi* dio je gramatike koja se bavi proučavanjem načina dobivanja **po značenju novih riječi** na osnovu postojećih, koji mogu biti po značenju vrlo slične, srodne ili drugačije. Više o tome bit će govora u poglavlju *Tvorba riječi u hrvatskom jeziku*.

Za neke se riječi pojedina pravila ne primjenjuju, što ih čini izuzecima u morfološkom smislu. Prema iskustvu dobivenom u izradi ovog rada, količina tih izuzetaka nije u značajnoj količini da bi napravila značajniji negativni utjecaj pri rješavanju zadanih zadataka.

Sljedeća stvar koja usložnjava problematiku su **glasovne promjene**. Glasovne se promjene događaju na *granici* osnove i nastavka, gdje se uslijed kombinacije glasova

37 *http://en.wikipedia.org/wiki/Morphology_%28linguistics%29*

38 *http://en.wikipedia.org/wiki/Inflection*

39 Infleksija za imenice, pridjeve i zamjenice još se naziva i deklinacija (engl. *declension*), a za glagole konjugacija (engl. *conjugation*)

koji se susreću, zbog određenih pravila i *prirodnosti* izgovora pojedini glasovi gube, dodaju ili pretvaraju u druge glasove. Za neke je glasovne promjene jednostavnije napraviti pravila, za neke teže, dok za neke nema pravila, već je bitno što *zvuči* prirodno. Glasovne promjene su sljedeće - kod je oznaka koja je korištena u programskom rješenju za tu glasovnu promjenu:

naziv	***kod***	***primjer***
nepostojano a	A	momak -> momka
nepostojano e	-[34]	Mirkovec -> Mirkovca
navezak		njim -> njime
vokalizacija	L	mislilac -> mislioca
palatalizacija	P	junak -> junače
sibilarizacija	S	junak -> junaci
jotacija	J	izraditi -> izrađen
jednačenje po zvučnosti	Z	gladak -> glatka
jednačenje po mjestu tvorbe	M	misao -> mišlju
gubljenje suglasnika	D	bez+zvučan -> bezvučan
smjenjivanje ie-je-e-i	-[35]	prijelaz -> prelaziti

Cilj ovog dijela rada je napraviti sustav koji implementira sva potrebna morfološka pravila za ispunjenje zadanih ciljeva: skup zaustavnih riječi, dobivanje osnovnih oblika riječi, te posljedično morfološki leksikon i morfološki *engine*.

6.2 Zaustavne riječi (engl. stopwords)

Zaustavne riječi (*engl. stopwords* [41]) su riječi koje nose manju sadržajnu vrijednost za potrebe pretraživanja. To su riječi s prevelikom učestalosti u većini dokumenata (recimo riječi "je"), ili su to riječi koje ne nose značenje, već igraju ulogu u strukturi rečenice (npr. veznici, usklici, prijedlozi, itd.).

Za potrebe pretraživanja te se riječi vade iz promatranog teksta, što se može promatrati kao smanjenje *šuma* u pretražnom prostoru.

I u pretraživanju i za neke druge potrebe računalne obrade pisane riječi (NLP *natural*

34 Trenutno ova glasovna promjena nije obrađena u programskom rješenju.

35 Ova glasovna promjena nije u potpunosti rješena u programskom rješenju.

41 *http://en.wikipedia.org/wiki/Stop_words*

language processing [42]) zaustavne riječi ne smiju se zanemariti, a u pojedinim slučajevima nose značajnu ulogu. Primjeri:

- strogo pretraživanje po izrazu - obično se stavlja u navodnike npr. "i reče",
- *strojno razumijevanje* (engl. *machine reasoning*)[43] pisane riječi ,
- analiza strukture rečenice,
- u *Part-of-speach tagging* procesu [44] mogu biti dosta korisne,
- itd.

Primjer obrade koja ilustrira važnost zaustavnih riječi može se naći u poglavlju *Prijedlog nastavka rada - strojno razumijevanje*, odnosno u poglavlju *Zaustavne riječi su važne*.

Kako je već objašnjeno u poglavlju *Uvod*, za hrvatski jezik za slobodnu upotrebu za potrebe rada nije nađen popis koji je bio zadovoljavajući, to jest da je dovoljno velik i kvalitetan, te je odlučeno da se izradi vlastito rješenje. To rješenje se kasnije pokazalo dovoljno obuhvatno i kvalitetno.

6.2.1 Osnovna zamisao

Osnovna pretpostavka je da tri vrste riječi nose većinu težine značenja izrečene misli (semantička vrijednost). To su :

- **imenice** - vrsta riječi koja se koristi za imenovanje bića, stvari i pojava. Imenice obično nose glavne uloge u strukturi rečenice, odnosno subjekti i objekti [4]. Primjer: ormar.
- **glagoli** - vrsta riječi kojom se izriče radnju, stanje ili zbivanje. U strukturi rečenice najčešće nose vrlo važnu ulogu predikata, unose dinamiku i opisuju mogućnosti i odnose između imenica [4]. Primjer: hodati.
- **pridjevi** - vrsta riječi kojom se pobliže opisuju imenice, a odgovaraju na pitanja koji, čiji i kakav. U strukturi rečenice nose važnu ulogu ili atributa ili dijela imenskog predikata - time izričući različita obilježja bića, stvari i pojava (tj. imenica) [4]. Primjer: crven.

42 *http://en.wikipedia.org/wiki/Natural_language_processing*

43 Strojno razumijevanje (*http://en.wikipedia.org/wiki/Machine_reasoning*) pripada u područje umjetne inteligencije, a bavi se razvojem kognitivnih računalnih sustava.

44 *http://en.wikipedia.org/wiki/Part-of-speech_tagging*

Mogle bi se povući istosmjernice (*paralele*) pojmovima iz računalnog svijeta:

- imenice s objektima,
- glagoli s metodama objekata,
- pridjevi s atributima objekata.

Primjer primjene takve analogije može se naći u poglavlju *Prijedlog nastavka rada - strojno razumijevanje*.

Što se tiče ostalih vrsta riječi:

- **brojevi** - također imaju svoju značenjsku težinu, no nisu obrađeni ovim radom, kao što je već objašnjeno u poglavlju *Kratki pregled gramatike hrvatskog jezika* str. 38. One nisu svrstane u zaustavne riječi, no nisu obrađene ni u programskom dijelu *Svođenje riječi na osnovni oblik*, pa ostaju u sustavu aktivne, no ne lematizirane. Primjer: dvadesetpet.
- **zamjenice** - zamjenjuju druge riječi, najčešće imenice i pridjeve. Za potrebe pretraživanja bilo bi potrebno saznati što konkretna zamjenica zamjenjuje, za što je potreban kontekst - ili u rečenici ili u prethodnom tekstu, što je vrlo složen problem. Budući da su klasični načini pretraživanja "bezkontekstni", onda obrada zamjenica na ovaj način nije moguća. S druge strane, budući da je učestalost zamjenica vrlo visoka, to dvoje ih kvalificira da ih svrstam u skup zaustavnih riječi. Primjer: svoj.
- **prilozi** - nepromjenjiva vrste riječi, najčešće stavljena uz glagole, da opisuje mjesto, vrijeme, način, uzrok i sl. S obzirom da je njihov broj relativno malen, učestalost dosta visoka i sadržajno su manje značajne, to ih kvalificira zaustavnim riječima. Potrebno je napomenuti da od svih odabranih vrsta riječi koje su svrštene u skup zaustavnih riječi, prilozi imaju najveću sadržajnu vrijednost, neki od njih manju (npr. riječ *ovako*), a neki od njih višu (npr. riječ *danas*), no radi jednostavnosti izvedbe svi su svrstani u skup zaustavnih riječi.
- **prijedlozi** - nepromjenjiva *pomoćna* vrsta riječi, koja izriče međusobne odnose među radnjama i stvarima u rečenici. Budući da im je broj malen, učestalost dosta visoka, a sadržajno nisu značajne, to ih kvalificira da ih stavljam u skup zaustavnih riječi. Primjer: od.
- **veznici** - nepromjenjiva *pomoćna* vrsta riječi, koja služi za povezivanje riječi ili podrečenice u rečenici. Budući da ih ima malo, učestalost im je

dosta visoka, a sadržajno nisu značajne, to ih kvalificira da ih stavljam u skup zaustavnih riječi. Primjer: ili.

- **usklici** - nepromjenjiva pomoćna vrsta riječi, koja služi za izricanje osjećaja, dozive, poticaje ili oponašanje zvukova iz prirode. Ima ih relativno malo, sadržajno nisu značajne. To ih kvalificira zaustavnim riječima. Primjer: oh.
- **čestice** - nepromjenjiva pomoćna vrsta riječi, koja služi za oblikovanje ili preoblikovanje rečeničnog ustrojstva i slično. Budući da ih ima malo i sadržajno nisu značajne, to ih kvalificira da ih stavim u skup zaustavnih riječi. Primjer: zar.

Osim riječi koje su svrštene u skup *zaustavnih riječi* samo po kategoriji *vrsta riječi*, neki glagoli su pridruženi u skup zaustavih riječi jer su vrlo česti, a sadržajno nisu značajni. To su sljedeći glagoli:

- **biti** - pomoćni glagol u raznim glagolskim vremenima, vrlo učestao, ima vlastiti način sklanjanja
- **htjeti** - pomoćni glagol u nekim glagolskim vremenima, vrlo učestao, ima vlastiti način sklanjanja
- **morati**, **trebati**, **željeti**, **moći**, **smjeti** - tzv. *modalni* glagoli, dosta učestali i nose manju sadržajnu vrijednost, prate normalni način sklanjanja, uz neke posebnosti

U sljedećem će se poglavlju opisati način na koji je dobiven skup zaustavnih riječi.

6.2.2 Zaustavne riječi - prilozi, prijedlozi, veznici, usklici i čestice

Iz nabrojane skupine koje su uvrštene u skup zaustavnih riječi u ovom se poglavlju opisuje dobivanje popisa nepromjenjivih vrsta riječi, odnosno **priloga**, **prijedloga**, **veznika**, **usklika** i **čestica**. Početni popis tih vrsta riječi je uzet iz knjige "Gramatika hrvatskog jezika, Težak, Babić 2005" [4].

Primjer

U navedenoj se knjizi pod točkom 326. (str. 161) nalazi poglavlje o prijedlozima. Oni su popisani i podijeljeni po kategorijama - za prijedloge je to kategorija uz koji

padež stoji. To je popisano u program na sljedeći način:

```
class FixedWordType(WordTypeBase):
    def __init__(self, code, name, attrs_fix=(),
                 allow_duplicate_words=False):
        ...
...

ATTR_PREP_TYPE   = WordAttr("PREP_T", True , False,
                      "Vrste prijedloga", ["UZ_G", "UZ_D",
                      "UZ_A", ...])
...

PREPOSITIONS  = base.FixedWordType("PREP",  "Prijedlozi",
                   attrs_fix=[base.ATTR_PREP_TYPE],
                   allow_duplicate_words=True)
...
# ---------- PRIJEDLOZI -----------------
# Uz genitiv
PREPOSITIONS.std_words.add_set("UZ_G", """
    bez blizu čelo do duž ispod ispred iz iza između
    iznad izvan kod kraj mjesto mimo nakon nakraj niže
    od oko osim pokraj poput pored poslije preko prije
    protiv put radi s sa sred u umjesto uzduž van više
    vrh za zbog
    """)
# Uz dativ
PREPOSITIONS.std_words.add_set("UZ_D", """
    k ka suprot nasuprot unatoč usprkos
    """)
# Uz akuzativ
PREPOSITIONS.std_words.add_set("UZ_A", """
    kroz među mimo na nad nada niz niza o po pod poda
    pred preda u uz uza za
    """)
...
```

Neki prijedlozi mogu stajati uz imenicu u više padeža, recimo prijedlog *za* koji može stajati uz imenicu u genitivu i akuzativu. Zbog toga kod definicije vrste riječi *PREPOSITIONS* parametar za konstruktor *allow_duplicate_words* ima vrijednost *True*. Kao što je rečeno, ovo je osnovni popis, no vrlo je vjerojatno da kroz proces pročišćavanja skupa skupljenih riječi kroz proces prepoznavanja osnovnih oblika riječi (opisanom u sljedećem poglavlju *Svođenje riječi na osnovni oblik*), neke riječi neće biti prepoznate kao promjenjive, i vjerojatno će biti neka od nepromjenjivih, te će se popis nepromjenjivih riječi i skup zaustavnih riječi proširiti. Na primjer, u djelima Ivane Brlić Mažuranić nađen je učestao usklik *nu* koji nije bio u predefiniranom skupu.

Po istoj su logici napravljeni popisi i za ostale vrste riječi, pa je ovo popis broja unesenih riječi po vrstama:

vrsta riječi	*broj riječi*	*broj kategorija*
prijedlozi	65	5
prilozi	224	21
veznici	38	3
usklici	51	4
čestice	12	1

6.2.3 Flektivna pravila

Prije nego što se započne razlaganje kako su dobiveni oblici promjenjivih riječi koje su odabrane za skup zaustavnih riječi (glagoli biti, htjeti, morati, zamjenice itd.), potrebno je napraviti uvod i pregled u programsko rješenje infleksijskog sustava za hrvatski jezik.

Kao što je već navedeno u poglavlju *Kratki pregled gramatike hrvatskog jezika,* promjenjive riječi imaju osnovni oblik riječi i njene oblike. Svaka promjenjiva vrsta riječi ima skup promjenjivih atributa po kojoj se vrši **infleksija**, odnosno izrada oblika riječi.

Svaki infleksijski atribut ima skup vrijednosti, npr. atribut *rod* ima skup vrijednosti *muški*, *ženski* i *srednji*. U poglavlju *Kratki pregled gramatike hrvatskog jezika* dan je pregled atributa i njihovih vrijednosti. Za neku vrstu riječi broj oblika neke riječi te vrste obično je umnožak vrijednosti svih atributa koji su za tu vrstu.

Primjer

Za vrstu riječi imenica promjenjivi atributi su broj i padež (deklinacija, engl. "declension"). Atribut broj ima dvije vrijednosti: jednina i množina, dok atribut padež ima sedam vrijednosti: nominativ, genitiv, dativ, akuzativ, vokativ, lokativ i instrumental. Iz toga slijedi da za svaku imenicu imamo 14 oblika, počevši od jednine/nominativ, do množine/instrumental.

Potrebno je uvesti jednu malu korekciju u navedeno, a to je da za jednu kombinaciju vrijednosti promjenjivih atributa, pojedina riječ može imati više mogućih valjanih oblika, dok u nekim slučajevima neke kombinacije nisu dozvoljene, pa nemamo niti jedan valjani oblik.

Primjer

Imenica noga za dativ jednine ima dva valjana oblika, "nozi" gdje je izvršena glasovna promjena sibilarizacije (nog + i), te "nogi", gdje nije izvršena sibilarizacija (proces desibilarizacije u hrvatskom jeziku).

Primjer

Povratna zamjenica "sebe" ima jedan promjenjivi atribut, a to je padež. No ne postoji oblik za nominativ i vokativ.

Kod formiranja oblika promjenjive riječi imamo dio riječi koji se ne mijenja i nazivamo ga **leksem** (engl. *lexem*), dok se dio koji je promjenjiv zove *afiks* (engl. *affix*). Ukoliko se promjenjivi dio dodaje sa stražnje strane naziva se **sufiks** ili nastavak (engl. *suffix*), a ako se dodaje s prednje strane zove se **prefiks** (engl. *prefix*).

Kada u hrvatskom jeziku govorimo o infleksiji skoro uvijek se radi u sufiksalnoj tvorbi oblika riječi. Točnije, samo kod pridjeva imamo sufiksalno-prefiksalnu tvorbu riječi, odnosno u superlativu se dodaju dijelovi riječi s prednje i zadnje strane.

Jedna riječ ne mora nužno imati jedan leksem za sve oblike, već ih može biti i više. Primjer su glagoli koji imaju infinitivnu osnovu i prezentsku osnovu. Uobičajeno je, ali ne i nužno, da se leksem može na neki način dobiti iz osnovnog oblika riječi, obično oduzimanjem zadnjih *slova* od osnove. Isto je tako uobičajeno, da je osnovni oblik po redoslijedu prvi nabrojani oblik u popisu svih oblika te riječi (npr. za imenice nominativ u jednini je jednak osnovi riječi).

Primjer

Riječ "slika" ima osnovni oblik "slika", leksem je dobiven oduzimanjem zadnjeg slova "a", te se dobije "slik". Genitiv se tvori pomoću sufiksa "e", te se dobiva spajanjem leksema i sufiksa "slik" + "e" -> "slike".

Primjer

Pridjev "crven" ima osnovni oblik "crven", leksem je jednak osnovnom obliku, a superlativ jednine muškog roda u nominativu se dobiva pomoću prefiksa "naj" i sufiksa "iji", te se dobije "naj" + "crven" + "iji" -> "najcrveniji".

Primjer

Glagol "mljeti" ima osnovni oblik "mljeti", leksem za prezent je "melj", pa se može primijetiti da nema izravne veze kako iz "mljeti" dobiti leksem "melj".

Za potrebe programskog rješenja razvijen je jednostavan *jezik* kojim se pišu flektivna pravila. Jedno pravilo se napiše u jednoj velikoj tekstovnoj varijabli gdje je svaki sufiks odvojen praznim znakovima od sljedećeg, a redoslijed je diktiran iteracijom vrijednosti svih promjenjivih atributa. To znači da konačan broj sufiksa mora odgovarati umnošku svih brojeva vrijednosti promjenjivih atributa - npr. za imenicu je 14 7 padeža x (jednina, množina). Ukoliko za pojedinu kombinaciju vrijednosti atributa oblika, tj. sufiksa, može biti više, oni su spojeni s "|" znakom. Ostatak sintakse predstavljen je u sljedećoj tablici:

znak	***značenje***
|	više sufiksa za istu kombinaciju promj.atributa se dijeli s ovim znakom
0	bez nastavka
$word_base	osnovni oblik riječi
%A	primijeniti glasovnu promjenu 'nepostojano A'
%S	primijeniti glasovnu promjenu 'sibilarizacija'
%P	primijeniti glasovnu promjenu 'palatalizacija'
-	nema oblika za tu kombinaciju
!	prisilni oblik (*forced*)
#	riječ se ignorira
##	ostatak reda se ignorira

Primjer

Sljedeći je primjer napravljen za atribute rod ("GENDER") i padež ("DECLENSION"):

```
##   M                 F       N
##   ----------------  ------- --------------------
#N   0                 a       e
#G   eg|ega            e       eg|ega
#D   em|emu            oj      em|emu
#A   eg|ega|0          u       e
#V   0                 a       e
#L   em|emu            oj      em|emu
#I   im                om      im
```

Interpretacija:

- za lokativ srednjeg roda postoje dva valjana sufiksa "em" i "emu"
- za vokativ muškog roda upotrebljava se samo leksem - nema nastavka

- redovi koji počinju s ## ignoriraju se, a isto tako i riječi koje počinju s #

6.2.4 Glasovne promjene

Dodatni problem koji se javlja pri formiranju oblika neke riječi su glasovne promjene. U uvodu *Kratki pregled gramatike hrvatskog jezika* dan je kratki pregled i popis glasovnih promjena. Kao što je tamo navedeno, u programskom rješenju su implementirane sve glasovne promjene za kojima se, prilikom izrade i testiranja, pojavila potreba. Sljedeća tablica daje pregled implementiranih glasovnih promjena s nazivom klasnih metoda u klasi *WordSuffixes* u kodu - jer će se te metode kasnije referencirati u primjerima:

glasovna promjena	***metoda***
gubljenje suglasnika	apply_vc_D
jotacija	apply_vc_J
vokalizacija	apply_vc_L
jednačenje po mjestu tvorbe	apply_vc_M
jednačenje po zvučnosti	apply_vc_Z
palatalizacija	apply_vc_P
sibilarizacija	apply_vc_S
nepostojano A	apply_vc_A
izmjenjivanje i|e|je|ije	get_form

Za razliku od većine glasovnih promjena, za glasovnu promjenu *nepostojano A* teško je programski detektirati kad se primjenjuje, a kad ne - tako da je u programskom smislu njen naziv potpuno opravdan: *nepostojano*. Kada i na koji način primijeniti glasovnu promjenu *izmjenjivanje i|e|je|ije* još je veći implementacijski problem, no srećom ta se glasovna promjena uglavnom primjenjuje pri tvorbi riječi, a ne pri infleksiji. Ono što se odnosi na infleksiju napravljeno je kroz *iznimke* (*if* naredbe) unutar glavne funkcije za određivanje oblika riječi **get_form**.

Primjenjivanje glasovnih promjena ponekad je potrebno naglasiti unutar flektivnih pravila kako je opisano u prethodnom poglavlju (npr. %A, %S, %P), no najčešće nije ništa posebno potrebno definirati već se one primjenjuju automatski u pozivu glavne funkcije **get_form**. U tom slučaju, nekad se metode koje obavljaju glasovne promjene pozivaju izravno iz funkcije *get_form*, a ponekad pojedina glasovna promjena unutar svoje programske logike nakon primjene svoje glasovne promjene, pozove metodu za drugu glasovnu promjenu (ulančavanje).

Primjer

Ovo je primjer iz kojeg se može vidjeti u programskoj logici da jedna glasovna promjena može inicirati drugu glasovnu promjenu

```
def apply_vc_J(cls, base, suffix, trace_vc_list=None):
    ...
    # primijeni jotaciju
    ...
    # pozovi glasovnu promjenu 'gubljenje suglasnika'
    bb, bs = cls.apply_vc_D(bb, bs, trace_vc_list)
    # pozovi glasovnu promjenu 'jednačenje po zvučnosti'
    bb, bs = cls.apply_vc_M(bb, bs, trace_vc_list)
    ...
    return bb, bs ...
```

U prethodnom se primjeru može primijetiti neobvezni parametar trace_vc_list koji u slaganju oblika riječi može evidentirati popis svih primjenjenih glasovnih promjena, što je vrlo korisno pri analizi, otkrivanju grešaka (*debugging*), učenja kako sustav radi i slično.

Primjer

U sljedećem se primjeru primjenjuje prvo glasovna promjena jotacija (t + j -> ć), a nakon toga glasovna promjena jednačenja po zvučnosti (s + ć -> š), te se dobiva "radošću":

```
>>> apply_vc_J("radost", "ju", vc_list)
('radošć', '')
>>> vc_list
['+J0', '+M']
```

Primjer

Demonstracija istog primjera prilikom poziva glavne funkcije "get_form()", jotacija se prepoznaje automatski i primjenjuje, te je rezultat isti kao i u prethodnom primjeru:

```
>>> get_form("radost", "radost", "ju")
'radošću'
```

Primjer

Implementirani dio glasovne promjene izmjenjivanje i|e|je|ije moguće je pozvati samo preko glavne funkcije "get_form()":

```
>>> get_form("radio", "radi", "ima", vc_list)
```

```
'radijima'
>>> vc_list
['+I1']
```

Važno pravilo kod glasovnih promjena je da se **propagacija glasovne promjene** kreće od prvog znaka sufiksa (u prethodnom primjeru slovo *j*) nalijevo. Prema tom pravilu, u prvom primjeru (*radost+ju*) drugo slovo u sufiksu "*u"* neće nikad biti promijenjeno zbog glasovne promjene. Drugim riječima, glasovne promjene uvijek nastaju samo na **spoju**, a novonastalo stanje se pripaja lijevoj strani, te nastaje novi spoj, koji se opet propagira nalijevo. To se pravilo pokazalo kao vrlo važno za dobivanje kvalitetnijih rezultata u problematici postavljenoj u poglavlju *Svođenje riječi na osnovni oblik*.

Statistički gledano, najčešće se događa da se promjena ne propagira, odnosno prvi spoj je jedini koji trpi promjene (zadnje slovo lijeve strane - leksema, i prvo slovo desne strane - sufiks). Međutim, nije rijetko da se promjena propagira još jednom, tj. predzadnje slovo leksema.

Sve glasovne promjene, glavna funkcija *get_form*, ali i cijeli infleksijski dio programskog rješenja razvijani su paralelno s testnim primjerima, koji se uvijek iznova pokreću i provjeravaju (*doc-testing*). To je vrlo osjetljivi dio koda, te je bilo vrlo važno ostvariti stabilnost i kvalitetu programskog rješenja.

Kao demonstraciju rada sustava za glasovne promjene u sljedećoj je tablici naveden manji izvadak iz primjera iz *doc-testing* dijela. Neki od primjera bili su rađeni s dodatnim opcijskim parametrom vc_list, no većina primjera nije pozvano s tim parametrom, pa je za njih stupac *vc_list* prazan:

poziv	***izlaz***	***vc_list***
apply_vc_A("pripovjetk", "a")	('pripovjedak', 'a')	
apply_vc_A("borc", "a")	('borac', 'a')	
apply_vc_A("ovakv", "")	('ovakav', '')	
apply_vc_A("ovakv", "", vc_list)	('ovakav', '')	['+A']
remove_vc_A("ovakav", "a")	('ovakv', 'a')	
remove_vc_A("nikakav", "a")	('nikakv', 'a')	
remove_vc_A("vrabac", "a")	('vrapc', 'a')	
remove_vc_A(*apply_vc_L("kabao", "a"))	('kabl', 'a')	
remove_vc_A("misao", "")	('misl', '')	

poziv	*izlaz*	*vc_list*
remove_vc_A("toranj", "evi", vc_list)	('tornj', 'evi')	['-A']
apply_vc_D("toran", "j")	('toranj', '')	
apply_vc_J("tret", "ji")	('treć', 'i')	
apply_vc_J("brz", "ji")	('brž', 'i')	
apply_vc_J("glod", "jem")	('gloď', 'em')	
apply_vc_J("radost", "ju", vc_list)	('radošć', '')	['+J0', '+M']
sljedeći primjer 'epentetsko L'		
apply_vc_J("grub", "ji", vc_list)	('grublj', 'i')	['+J1']
apply_vc_L("kabao", "a", vc_list)	('kabal', 'a')	
remove_vc_A("kabao", "a", vc_list)	('kabl', 'a')	['+L0', '-A']
apply_vc_M("rados", "ću")	('radoš', 'ć')	
apply_vc_M("bijes", "nji")	('biješ', 'nji')	
apply_vc_M("mis", "lju")	('miš', 'lj')	
apply_vc_M("paz", "nja")	('paž', 'nja')	
apply_vc_M("obran", "ben")	('obram', 'ben')	
apply_vc_M("crn", "purast", vc_list)	('crm', 'purast')	['+M']
remove_vc_M("radoš", "ću")	(['rados', 'radoh'],'ć')	
remove_vc_M("biješ", "nji")	('bijes', 'nji')	
remove_vc_M("oraš", "čić")	(['oras', 'orah'], 'čić')	
remove_vc_M("miš", "lju")	('mis', 'lj')	
remove_vc_M("paž", "nja")	('paz', 'nja')	
remove_vc_M("obram", "ben")	('obran', 'ben')	
remove_vc_M("crm", "purast", vc_list)	('crn', 'purast')	['-M']
remove_vc_Z("vrap", "c")	('vrab', 'c')	
remove_vc_Z("glat", "k")	('glad', 'k')	
remove_vc_Z("tob", "dž")	('top', 'dž')	
remove_vc_Z("svag", "danji")	('svak', 'danji')	
remove_vc_Z("iš", "čistiti", vc_list)	('iz', 'čistiti')	['-M?', '-Z2']
remove_vc_Z("pripovjet", "ka")	('pripovjed', 'ka')	
apply_vc_Z("vrab", "c")	('vrap', 'c')	

poziv	*izlaz*	*vc_list*
apply_vc_Z("žež", "ka")	('žeš', 'ka')	
"".join(apply_vc_Z("s", "Bogom"))	'zbogom'	['+Z2']
apply_vc_S("majk", "i")	('majc', 'i')	
apply_vc_S("jarug", "i", vc_list)	('jaruz', 'i')	['+S']
remove_vc_S("pec", "i", vc_list)	('pek', 'i')	['-S']
apply_vc_P("junak", "e", vc_list)	('junač', 'e')	['+P']
remove_vc_P("junač", "e", vc_list)	('junak', 'e')	['-P']
get_form("radio", "radi", "ima")	'radijima'	['+I1']
get_form("poni", "poni", "ima")	'ponijima'	['+I3']
get_form("list", "list", "je")	'lišće' ['+J0', '+M']	

6.2.5 Zaustavne riječi - glagoli

S obzirom na to da je u prethodna dva poglavlja dan pregled implementacije infleksijskog sustava za hrvatski jezik, sad je moguće opisati na koji su način dobiveni svi oblici pomoćnih glagola biti, htjeti, te modalnih glagola morati, trebati, željeti, moći i smjeti, a za potrebu izrade skupa zaustavnih riječi.

Pomoćni glagoli **biti** i **htjeti** su s obzirom na infleksiju posebne vrste glagola. Ti glagoli imaju u nekim glagolskim vremenima naglašeni i nenaglašeni oblik (npr. jesam / sam, hoću / ću), imaju niječne oblike (npr. nisam, neću), više od dva leksema i slično. Zbog toga je za njih bilo potrebno napraviti poseban skup flektivnih pravila, a njihovom primjenom dobiti sve oblike tih riječi.

Primjer

Ovaj primjer demonstrira jedno flektivno pravilo i njegovu primjenu na tri oblika pomoćnog glagola "htjeti" za glagolsko vrijeme sadašnje (prezent)

```
self._suffixes_pre_htjeti = self.add_suffixes("PRE", "SINGLE", "htjeti",
                      "", (base.ATTR_NUMBER, base.ATTR_PERSON),
                        u"""##   SINGULAR
                        ##   M
                        ##   ----------
                        #1   u
                        #2   eš
                        #3   e
                        ##   PLURAL
                        ##   -------------
```

```
                        #1   emo
                        #2   ete
                        #3   e
                    """)

# prvi je naglašeni - hoću
word_htjeti.add_forms(u"hoć", "PRE", self._suffixes_pre_htjeti,
                      attr_extra="NAG")
# drugi je nenaglašeni - ću
word_htjeti.add_forms(u"ć", "PRE", self._suffixes_pre_htjeti,
                      attr_extra="NEN")
# treći je nijekani - neću
word_htjeti.add_forms(u"neć", "PRE", self._suffixes_pre_htjeti,
                      attr_extra="NIJ")

print "\n".join(["%s %s" % (k,v) for k,v in word_htjeti.get_forms(
                "V#PRE#NIJ").forms_ordered])
S/1 [u'neću']
S/2 [u'nećeš']
S/3 [u'neće']
P/1 [u'nećemo']
P/2 [u'nećete']
P/3 [u'neće']
```

Rezultat ovog je da je za riječ htjeti za sva glagolska vremena i glagolske oblike dobiveno 38 oblika, od kojih su 33 različita. Za glagol biti je ukupno 50 oblika, od kojih je 45 različito. Glagol biti ima više oblika jer ima imperativ, dvovidni oblik prezenta itd.

Modalni glagoli **morati**, **trebati**, **željeti**, **moći** i **smjeti** napravljeni su na istoj osnovi s time da se njihova konjugacija (infleksija) ne razlikuje bitno od konjugacije običnih glagola. Za svaki je glagol dobiveno 20 oblika.

Ukupno je dobiveno 153 različite riječi vrste glagoli.

6.2.6 Zaustavne riječi - zamjenice

Premda su zamjenice jedna vrsta riječi, zamjenica ima više podvrsta, a svaka od njih ima svoja nepromjenjiva i promjenjiva svojstva, a time i različitu infleksiju [46]:

Vrsta zamjenica	*Kod*	*Nepromj.svojs.*	*Promjenjiva svojstva*
Osobne	OSO	NUMBER, PERSON_MFN	DECLENSION
Povratne	POV	-	DECLENSION
Posvojne	POS	NUMBER, PERSON_MFN	NUMBER, DECLENSION, GENDER
Povratno-posvojne	PPO	-	NUMBER, DECLENSION, GENDER
Pokazne	POK	-	NUMBER, DECLENSION, GENDER
Upitne odnosne imenicne	UOD.IME	-	NUMBER, DECLENSION
Upitne odnosne pridjevne	UOD.PRD	-	NUMBER, DECLENSION, GENDER
Neodređene promjenjive	NEO.CH1	-	NUMBER, DECLENSION
Neodređene promjenjive - spol	NEO.CH2	-	NUMBER, DECLENSION, GENDER
Neodređene nepromjenjive	NEO.FIX	-	-

Stupac *kod* sadrži vrijednost korištenu u programu. Za neke od zamjenica utvrdilo se da su nepromjenjive (npr. kakavgod, čijigod, kojigod, itd.).

Kao što je to bio slučaj s glagolima iz prethodnog poglavlja, tako su i za zamjenice napravljena flektivna pravila pomoću kojih su se dobili različiti oblici zamjenica.

46 Odabrana je klasifikacija zamjenica po flektivnim pravilima, no postoji klasifikacija i po ulozi u rečenici - neodređene, niječne, opće.

Budući da i samih vrsta zamjenica već ima dosta, onda je i flektivnih pravila dosta.

Primjer

Ovaj primjer daje primjer flektivnih pravila za posvojne zamjenice, za sva lica množine "naš", "vaš" i "njihov":

```
_suffixes_pron_pos_p1 = PRONOUNS_POS.add_suffixes("P/1", "SOME", u"naš",
"",
                        (base.ATTR_NUMBER, base.ATTR_DECLENSION,
                         base.ATTR_GENDER),
    """##   SINGULAR
       ##   M                 F       N
       ##   ----------------  ------- --------
       #N   0                 a       e
       #G   eg|ega            e       eg|ega
       #D   em|emu            oj      em|emu
       #A   eg|ega|0          u       e
       #V   0                 a       e
       #L   em|emu            oj      em|emu
       #I   im                om      im

       ##   PLURAL
       #N   i                 e       a
       #G   ih                ih      ih
       #D   im|ima            im|ima  im|ima
       #A   e                 e       a
       #V   i                 e       a
       #L   im|ima            im|ima  im|ima
       #I   im|ima            im|ima  im|ima
    """)
PRONOUNS_POS.add_suffixes("P/2",  "SOME", "vaš",    "",
                          suffixes_force = _suffixes_pron_pos_p1)
PRONOUNS_POS.add_suffixes("P/3M",  "SOME", "njihov", "",
                          suffixes_force = _suffixes_adj_pos_neo)
```

Primjer

Ovaj primjer demonstrira primjenu infleksijkog pravila za posvojne zamjenice, no ovaj puta za jedninu ("moj", "tvoj", itd.):

```
>>> from pronouns import *
>>> s = PRONOUNS_POS.get_suffixes("S/1")
>>> forms, words = s.get_forms("moj")
>>> print s.pp_forms(forms)
S/N/M     = [u'moj']
S/N/F     = [u'moja']
S/N/N     = [u'moje']
S/G/M     = [u'mojeg', u'mojega', u'mog', u'moga']
```

```
S/G/F     = [u'moje']
S/G/N     = [u'mojeg', u'mojega', u'mog', u'moga']
...
S/I/M     = [u'mojim']
S/I/F     = [u'mojom']
S/I/N     = [u'mojim']
P/N/M     = [u'moji']
P/N/F     = [u'moje']
P/N/N     = [u'moja']
...
P/I/M     = [u'mojim', u'mojima']
P/I/F     = [u'mojim', u'mojima']
P/I/N     = [u'mojim', u'mojima']
```

Rezultat je u sljedećoj tablici:

vrsta zamjenica	*ukupan broj oblika*	*zamjenice*
NEO_CH1	168	ništa, kojetko, netko, itd.
NEO_CH2	588	ničiji, nikakav, itd.
OSO	70	on, oni, ono, vi, ona, itd.
POK	336	ovakav, takav, ovolik, itd.
POS	294	on, oni, vi, ona, mi, itd.
POV	7	sebe
PPO	42	svoj
UOD_IME	28	tko, što
UOD_PRD	168	kakav, koji, kolik, čiji

Ukupno je 1701 oblik svih promjenjivih zamjenica[45].

45 Iz obrade su izostavljene neke od složenih neodređenih nepromjenjivih zamjenica.

6.3 Svođenje riječi na osnovni oblik

Kao što je već spomenuto, najveći je problem obaviti normalizaciju tekstova na hrvatskom jeziku na način da se oblici riječi svedu na zajednički oblik. U slučaju da se to radi procesom lematizacije, onda je taj zajednički oblik ujedno i osnovni ili korijenski oblik (lema). Rezultat tog procesa je da se dobivaju znatne prednosti za računalnu obradu tekstova, a u ovom konkretnom slučaju značajno se smanjuje prostor pretraživanja (samo imenice, glagoli, pridjevi, samo osnovne riječi - korijenske riječi).

Drugi razlog je to da se i **traženi tekst i tekst po kojem se traži lematizacijom normalizira na isti oblik**, što omogućuje da traženje po jednom obliku neke riječi pronađe u traženom tekstu drugi oblik te iste riječi.

Primjer

Tražimo riječ "lopatama", to se lematizira u osnovnu riječ "lopata", a sustav to prepozna u riječi "lopatom" jer se ona lematizira u istu osnovnu riječ "lopata".

U prethodnom se poglavlju za *dobivanje svih oblika neke riječi* išlo u smjeru (infleksija)

```
osnovni oblik riječ
    -> preko flektivnih pravila i leksema
        -> svi oblici riječi
```

U ovom se odjeljku postavlja puno teži problem *kako iz nepotpunog skupa oblika neke riječi dobiti njihovu zajedničku osnovnu riječ*. U tom je slučaju potrebno ići suprotnim smjerom (**obrnuta infleksija**):

```
neki oblici riječi
    -> preko skupa svih flektivnih pravila i mogućih leksema
        -> skup potencijalnih lema
            -> izabrati jednu ili više najboljih kandidata
```

U ovom će se poglavlju opisati obrnuti infleksijski sustav koji omogućuje rješavanje zadane zadaće.

6.3.1 Tvorba riječi u hrvatskom jeziku

U hrvatskom jeziku postoji određeni skup načina koji mogu na osnovi postojećih riječi, koje imaju svoje značenje kreirati nove riječi s novim značenjem. U tom slučaju značenje nove riječi može biti vrlo slično, srodno ili drugačije od značenja

one iz koje je ta nova riječ nastala.

Taj se proces zove **tvorba riječi**, koju treba razlikovati od *infleksije* - gdje se radi uvijek o skupu oblika jedne riječi, odnosno, oblici zadržavaju isto značenje (više o infleksiji u poglavlju *Kratki pregled gramatike hrvatskog jezika*). U knjizi "Gramatika hrvatskog jezika, Težak, Babić 2005" [4] je obrađena ova tema, a taj je dio bio osnovni izvor informacija o gramatici hrvatskog jezika iskorištenih u ovom dijelu rada.

Postoji više načina tvorbe novih riječi:

- *izvođenje* - na osnovu riječi dodaje se nastavak ili *sufiks* koji sam po sebi ne nosi značenje. Dobivena riječ zove se *izvedenica*, te je po značenju obično ili vrlo slična ili srodna. Primjer: pridjev *star*, nastavak *-ac* -> nastaje riječ *starac* - imenica srodna po značenju imenici star.
- *slaganje* - je dobivanje novih riječi spajanjem osnova dviju ili više riječi, gdje svaka od njih ima svoje značenje. Nova riječ koja nastaje zove se *složenica*. Primjer: spajanjem imenice *čudo* i glagola *tvoriti* -> nastaje nova riječ *čudotvorac* koja je po značenju različita i od prve i od druge riječi od kojih je nastala. U ovom načinu nastajanja novih riječi potrebno je naglasiti podvrstu *prefiksalna tvorba*, gdje se s prednje strane postojeće riječi dodaje prijedlog ili niječnica *ne* te nastaje riječ koja je obično srodna toj riječi. Primjeri: od+jedriti, pod+liti, ne+sputan, itd.
- *preobrazba* - gdje bez dodavanja tvorbenih nastavaka nastaje nova vrsta i/ili značenje riječi. Primjer: pridjev *mlad* -> njen oblik za ženski rod je *mlada*, a ta riječ može imati značenje mladenke, nevjeste.
- *stvaranje skraćenica* - nastaje skraćivanjem postojeće riječi i zadržavanjem istog značenja. Primjer: automobil -> auto, Industrija NAfte -> INA.

6.3.2 Izvođenje novih imenica, glagola i pridjeva

Za ovaj su rad posebno zanimljivi načini nastajanje novih imenica, glagola i pridjeva **izvođenjem**. Razlog tome je što veoma veliki broj riječi u hrvatskom jeziku nastaje baš *izvođenjem* imenica, glagola i pridjeva. Pri tome možemo razlikovati:

- izvođenje novih riječi u istu vrstu riječi - primjer: imenica -> imenica, stol -> stolić (umanjenica)
- izvođenje novih riječi u drugu vrstu riječi - primjer: pridjev -> imenica,

sirot -> sirotan

Za svaku od tih vrsta riječi postoji popis *sufiksa* koji dodavanjem na osnovu riječi daju novu riječ. U ovom će se radu za te sufikse koristiti naziv **tvorbeni sufiks**. Svaki tvorbeni sufiks ima obilježje *plodnosti* koje daje mjeru učestalosti tvorbe novih riječi upotrebom takvog sufiksa. Općenito govoreći, razlikujemo slabo plodne sufikse - koji su rjeđe u upotrebi, i plodne sufikse - koji su češće u upotrebi.

Iz knjige "Gramatika hrvatskog jezika, Težak, Babić 2005" [4] svi sufiksi koji se odnose na tvorbu imenica, pridjeva i glagola, su registrirani unutar objekata vrste riječi u programskom rješenju. Ti su sufiksi kasnije poslužili za znatno povećanje kvalitete sustava za prepoznavanje osnovnog oblika riječi (korijenskog oblika).

Ovaj složeni primjer izvadak je iz izvornog koda pomoću kojeg će se demonstrirati prijava tvorbenih sufiksa za tvorbu novih imenica izvođenjem:

```
class Noun(ChangeableWordBase):
    def __init__(self, word_base, gender=None,
                 apply_vc_a=True, ... ):
        ...

class WordTypeBase(object):
    ...
    def _add_wts(self, attrs_fix, suff_value, freq_type, group,
                 desc="", examples="", vc_list=None,
                 suffixes=None, base_ends_aeiou=None,
                 \*\*kwargs):
        ...
        wts = WordTypeSuffix(word_type=self, attrs_fix=attrs_fix,
                             suff_value=suff_value,
                             freq_type=freq_type, group=group,
                             desc=desc, examples=examples,
                             vc_list=vc_list,
                             base_ends_aeiou=base_ends_aeiou,
                             \*\*kwargs)

class NounType(base.ChangeableWordType):
    ...
    def __init_wts(self):
        # VR", "vršitelj radnje
        self._add_wts({"GEN":"M"}, "ac" ,   1, "vr",
                      "", "pisac", apply_vc_a=True)
        self._add_wts({"GEN":"M"}, "ač" ,   3, "vr",
                      "ie cijepati->", "cjepač")
        self._add_wts({"GEN":"M"}, "ar" ,  -1, "vr",
                      "ie mlijeko->", "mljekar")
        self._add_wts({"GEN":"M"}, "jar",  -1, "vr",
                      "ie mlijeko->", "mljekar", vc_list="J")
```

```
self._add_wts({"GEN":"M"}, "ič" ,   0,
              "vr", "LIST:", "branič,gonič,vodič,ribič")
self._add_wts({"GEN":"M"}, "lac",   0, "vr",
              "ie/je, from infinitive, telj", "mislilac")
# PR", "imenica s osobinom izrečenu pridjevom",
#      "pridjevna osnova"
self._add_wts({"GEN":"M"}, "ac" ,   1, "PR",
              "", "krivac, aristotelovac",
              apply_vc_a=True)
# OS", "ostalo", "različite osnove"
self._add_wts({"GEN":"M"}, "jak",   2, "os",
              "", "čudak, imenjak, zemljak",
              apply_vc_a=False)
self._add_wts({"GEN":"M"}, "an" ,   1, "os",
              "osjećajno", "blesan, tupan, sirotan")
self._add_wts({"GEN":"M"},"ac"  ,   1, "bl",
              "", "krastavac, kakaovac", apply_vc_a=True)
...
self._add_wts({"GEN":"F"},"ina" ,  -1, "oz",
              "", "trećina")
self._add_wts({"GEN":"F"},"ština", -1, "oz",
              "", "štokavština")
```

Svaka promjenjiva vrsta riječi nasljeđuje klasu *ChangeableWordType*. Interna metoda *_add_wts()* koristi se za registriranje tvorbenog sufiksa, a unutar klase se na osnovu proslijeđenih parametara kreira objekt klase *WordTypeSuffix* koji predstavlja tvorbeni sufiks. Klasa *Noun* je navedena da se vidi na koji se način naknadno mogu kreirati objekti riječi vrste imenica. Primijetimo da je sufiks "ac" naveden dva puta, pod grupu "vr" i grupu "pr". **WTS** je skraćenica od naziva klase *WordTypeSuffix*, a ta klasa predstavlja jedan tvorbeni sufiks za neku vrstu riječi. Od parametara koji se šalju u metodu *_add_wts()* i konstruktor potrebno je pojasniti sljedeće:

- attrs_fix - skup svih nepromjenjivih atributa koje dobivaju objekti vrste riječi koji nastaju ovim tvorbenim sufiksom. U konkretnom primjeru to su imenice, a nepromj. atribut je jedan i to je gramatički rod imenice ("G" - gender, primjer vrijednosti je "M" - muški rod)
- suff_value - sama vrijednost tvorbenog sufiksa - primjer "ac", čijim se dodavanjem na osnovu dobivaju nove imenice, primjer pisati -> pis + ac -> pisac
- freq_type - mjera plodnosti sufiksa - mjere su od 0 do 3 - gdje je 0 izrazito plodan sufiks, a 3 vrlo plodan. Mjera -1 koristi se za neodređenu mjeru
- group - kod grupe u koji pripada tvorbeni sufiks prema klasifikaciji značenja koji nosi nova riječ [4] primjer grupa "vr" - dobivanje novih riječi

koje nose značenje Vršitelja Radnje.

- desc - dodatni opis ili napomena - neobvezno polje
- examples - primjeri riječi odvojenih zarezom koji su nastali na osnovi ovog tvorbenog sufiksa. Ukoliko popis počinje s rječju "LIST:" tada je popis riječi koji su nastali na osnovu ovog sufiksa konačan, što je slučaj za vrlo slabo plodnim sufiksima. Takvi sufiksi nisu zanimljivi za sustav *obrnute infleksije*.
- kwargs - svi ostali argumenti koji se mogu proslijediti konstruktoru klase za objekte riječi koji nastaju na osnovi ovog sufiksa. U ovom primjeru taj parametar je iskorišten za proslijeđivanje parametra *apply_vc_a* konstruktoru *Noun()* koji služi za kreiranje objekata riječi tipa imenice, a parametar *apply_vc_a* govori treba li u sklonidbi primjenjivati glasovnu promjenu nepostojano A.

Sljedeća tablica prikazuje broj tvorbenih sufiksa prijavljenih u sustav za vrste riječi imenice, glagole i pridjeve:

vrsta riječi	***broj tvorb. sufiksa***
len(NOUNS.wts_list.keys())	116
len(VERBS.wts_list.keys())	114
len(ADJECTIVES.wts_list.keys())	122

6.3.3 Obrnuta infleksija - osnovna zamisao

Prije izlaganja osnovne zamisli napravit će se kratka analiza problematike.

Svaka od promatranih vrsta riječi ima svoja flektivna pravila, čiji je broj konačan. Sva su ta flektivna pravila sufiksalna, a jedina iznimka je prefiksalno-sufiksalna tvorba superlativa za pridjev. S obzirom da prefiksalni dio te tvorbe nije problematičan, jer je stalan i ne podliježe glasovnim promjenama, možemo se koncentrirati samo na sufiksalnu tvorbu.

Za promjenjive riječi (u ovom slučaju imenice, glagole i pridjeve), u tekstovima se puno češće pojavljuju oblici riječi, a ne njihovi korijeni. Oblici riječi su riječi nastale spajanjem leksema - koji je obično dobiven iz osnove riječi i sufiksa iz flektivnog pravila.

Postavlja se sljedeće pitanje:

Je li moguće samo na osnovi podudarnosti sufiksa neke riječi zaključiti s određenom

sigurnošću da se točno radi o obliku riječi X, a za tu riječ X samo na osnovu tog sufiksa doznati vrstu riječi i ostale parametre koji su potrebni za preciznije određivanje flektivnog pravila koja se primjenjuje na tu riječ.

Osnovna misao vodilja prije izrade rješenja je bila: Da. Moguće je.

Za ostvarenje te ideje neće se koristiti samo flektivni nastavci, već tvorbeni nastavci zajedno s flektivnim - u nastavku rada koristi se naziv **tvorbeno-flektivni sufiksi**.

Sljedeća dva jednostavna primjera ilustriraju način razmišljanja koji bi vodio rješenju.

Primjer

Ako imamo riječ koja počinje s "naj" vrlo je vjerojatno da se radi o pridjevu u superlativnom obliku. Taj je zaključak izveden na osnovu poznavanja svih flektivnih pravila za promjenjive riječi u hrvatskom jeziku -> nijedna druga vrsta riječi nema prefiks "naj" u flektivnim pravilima.

Primjer

Za riječ koja završava na "ićima", vrlo je vjerojatno da je imenica muškog roda, oblik je množina, oblik je za padež dativ, lokativ ili instrumental, a sama imenica je umanjenica (deminutiv) neke druge imenice muškoga roda (npr. grad -> gradić). Svi ti zaključci su izvedeni na istoj podlozi kao i za prethodni primjer, odnosno od svih riječi, a osobito onih koji se tvore izvođenjem (sufiksalna tvorba), samo tvorba novih imenica muškog roda, koji su umanjenice imenica od kojih potječu, završavaju na "ić" a kombinacija "ićima" još ih preciznije otkriva - odnosno samo takve riječi od svih mogućih tvorbi riječi, imaju takav oblik (množina D/L/I).

Izrada rješenja za otkrivanje osnova i vrsta riječi na temelju ove zamisli treba sljedeće pripreme:

- napraviti sva flektivna pravila za imenice, pridjeve i glagole;
- popisati sve dodatne parametre za imenice, pridjeve i glagole, koji jednoznačno određuju način na koji se vrši infleksija konkretne riječi. To se može implementirati pomoću različitih klasa za svaku vrstu riječi;
- popisati sve tvorbene sufikse za imenice, pridjeve i glagole;
- napraviti infleksiju za svaki tvorbeni sufiks, uključujući i onaj nulti (bez sufiksa) za sve promatrane vrste riječi (imenice, pridjevi i glagoli), rezultat toga su *tvorbeno-flektivni* sufiksi;

- nakon tog procesa napraviti registar svih tvorbeno-flektivnih sufiksa koji se pojavljuju, te ih registrirati u objektu rječnika (dictionary) na način da jedan član rječnika kao ključ ima sufiks - a vrijednost je popis svih tvorbeno-flektivnih sufiksa koji su nastali u nekom flektivnom obliku tvorbenog sufiksa;
- prilikom prepoznavanja osnova pokušati prepoznavati oblike riječi preko tih registriranih sufiksa, te odvajanjem prepoznatog sufiksa od riječi dobiti potencijalnu osnovu koju treba pustiti da kreira objekt te vrste riječi. Na taj se način dobivaju svi ostali (flektivni) oblici te riječi;
- napraviti filter nad svim prepoznatim osnovama i odlučiti se za najkvalitetnije - na osnovi statistike i nekih drugih pravila.

Prethodna lista ovog složenog procesa može djelovati na prvi pogled donekle nerazumljivo, no dana je prvenstveno radi pregleda, a u nastavku će se iscrpnije objasniti potrebne pojedinosti.

6.3.4 Registracija tvorbeno-flektivnih sufiksa

Ovim primjerom dajem pregled svih važnijih klasa i metoda koje se koriste u procesu registracije **tvorbeno-flektivnih sufiksa** (skraćenica je **TI** sufiks), te sam proces registracije. Za primjer je uzeta vrsta riječi imenica:

```
class Noun(ChangeableWordBase):
    def __init__(self, word_base, gender=None,
                 apply_vc_a=True, spec=None, ext=None,
                 is_suffix=False, sri=None, decl=None
        ...
        self.__init_forms()

class NounType(base.ChangeableWordType):
    ...
    def __init_wts(self):
        ...
        self._add_wts({"GEN":"M"}, "ić"    ,   0, "os",
                      "(umanjenice) i podrijetlo/srodstvo", "bratić,
banović, Kovačević")
        ...
        # NOTE: add 0-type WTS objects
        for key, attrs_fix, params_init in (
            self.iter_param_combinations()):
            self._add_wts(attrs_fix, "$BASE$"
                    ,-1, key, "", ""  , **params_init)

    def init_wts_forms(self):
```

```
        for code, wts in self.wts_list.iteritems():
            gender, apply_vc_a, spec, ext, decl = ...
            noun = Noun(wts.suff_value, gender=gender,
                        apply_vc_a=apply_vc_a, spec=spec,
                        ext=ext, decl=decl)
            for key, form_list in noun.word_forms:
                for ordnr, form in enumerate(form_list):
                    wts_forms.append((noun, key, ordnr+1, form))
        # NOTE: this call adds list of wts entries into
        #       suff.registry
        wts.add_forms(wts_forms)

class WordTypeSuffix(object):
    def __init__(self, word_type, attrs_fix, suff_value,
                 freq_type, group, desc="", examples="",
                 vc_list=None, register=True,
                 base_ends_aeiou=None, \*\*kwargs):
        ...
    def add_forms(self):
        for word_obj, suff_key, ordnr, suff_value in (
            wts_suff_list):
            SUFF_REGISTRY.add_suffix(self, word_obj,
                        suff_key, ordnr, suff_value)

class SuffixRegistryItem(object):
    ...
    def __init__(self, wts, word_obj, suff_key,
                 ordnr, suff_value):


class SuffixRegistry(object):
    ...
    def add_suffix(self, wts, word_obj, suff_key,
                   ordnr, suff_value):
        item = SuffixRegistryItem(wts, word_obj,
                    suff_key, ordnr, suff_value)
        if suff_value not in self.suffix_dict:
            self.suffix_dict[suff_value]=[]
        self.suffix_dict[suff_value].append(item)

# NOTE: global object
SUFF_REGISTRY = SuffixRegistry()
```

Objekti klase *Noun* predstavljaju imenice, odnosno objekte koji za određenu riječ sadrže sve potrebne podatke o njoj, npr. osnovni oblik riječi, vrstu riječi, sve oblike te riječi i slično. Inače svaka vrsta riječi koju ćemo obraditi nasljeđuje klasu *ChangeableWordBase* gdje su stavljene sve zajedničke metode i atributi svih vrsta

riječi. Klasa za objekte imenica bit će podrobnije objašnjena u poglavlju *Imenice*, klasa za pridjeve u poglavlju *Pridjevi*, a klasa za glagole u poglavlju *Glagoli*.

Klasa *NounType* je *singleton* klasa (klasa sa samo jednim instanciranim objektom) za vrstu riječi imenica, odnosno objekt ove klase ima u sebi sve potrebne informacije za vrstu riječi imenica, npr. koji su atributi promjenjivi, koji su nepromjenjivi, koja su flektivna pravila, koji su tvorbeni sufiksi, koji su tvorbeno-flektivni sufiksi i slično. U konstruktoru objekta se poziva metoda *__init_wts()* koja prijavljuje sve tvorbene sufikse na način da za svaki od njih poziva metodu *_add_wts()*. Taj dio procesa je iscrpnije objašnjen u poglavlju *Izvođenje novih imenica, glagola i pridjeva*. **WTS** je skraćenica za *tvorbeni sufiks* (WordTypeSuffix).

Ono što je potrebno primijetiti je da se dodaju **0-type** WTS objekti, odnosno "tvorbeni" sufiksi koji imaju vrijednost prazan string (""). To je potrebno za detekciju onih oblika riječi koji nisu nastali tvorbom izvođenja, već su *netvorbeni*. Primjer: imenica zid nije izvedena iz neke druge riječi, no sustav će iz popisa oblika te riječi, npr. zidom, zidu, zidovi, moći prepoznati osnovni oblik - zid.

Metoda init_wts_forms() središnji je dio registracije tvorbeno-flektivnih sufiksa. Za svaki tvorbeni sufiks (WTS) kreira se objekt imenice instanciranjem objekta iz klase Noun, a taj objekt interno kreira sve flektivne oblike riječi pozivanjem metode __init_forms(). Tada se iterira po svim flektivnim oblicima riječi, pa se ti oblici skupljaju u listu, koja se na kraju registira u središnji registar tvorbeno-flektivnih sufiksa uz pomoć metode WTS objekta add_forms().

Klasa *SuffixRegistry* je *singleton* klasa, te njezin jedini instancirani objekt sadrži popis svih tvorbeno-flektivnih sufiksa (WTS). Interni registar je objekt rječnika (*dictionary*) koji za ključ ima vrijednost WTS sufiksa, a vrijednost je popis svih objekata klase *SuffixRegistryItem*. Skraćenica koja se koristi za *SuffixRegistryItem* je **SRI**. SRI objekt ima sve potrebne podatke za određeni tvorbeno-flektivni sufiks, odnosno podatke o WTS objektu, vrijednost tvorbeno-flektivnog sufiksa, za koju kombinaciju promjenjivih atributa je taj sufiks dobiven, i slično.

Statistika registra tvorbeno-flektivnih sufiksa je sljedeća:

- u registru je prijavljeno za imenice, pridjeve i glagole ukupno 6218 tvorbeno-flektivnih sufiksa
- u rječnicima svih tvorbeno-flektivnih sufiksa ukupno 8776 SRI objekata
- za glagole je 4591 SRI objekata

- za pridjeve je ukupno 2776 SRI objekata
- za imenice je ukupno 1409 SRI objekata

Sljedeća tablica daje pregled SRI sufiksa (TI sufiksa) po duljini samog sufiksa. U zadnja dva stupca primjer je jednog člana - predzadnji stupac ima vrijednost TI sufiksa, a zadnji stupac ima vrijednost njegovog tvorbenog sufiksa od kojeg je nastao (<kod vrste riječi>.<vrijednost tv. sufiksa>).

duljina sufiksa	***broj SRI obj.***	***primjer TI sufiksa***	***za tv.sufiksa***
1	15	k	V.ći
2	120	r	V.rijeti
3	510	uče	V.ući
4	1172	aten	V.asti
5	1452	njemo	V.njeti
6	1336	pasene	V.pasti
7	888	uštinom	N.ština
8	459	jarevima	N.jar
9	188	ojujijaše	V.ojevati
10	66	čaninovima	N.čanin
11	12	sjednijaste	V.sjesti

Postavlja se pitanje: u koliko slučaja sufiks jedinstveno određuje vrstu riječi, osnovni oblik i infleksiju? To je moguće vidjeti iz sljedeće tablice, koja daje pregled SRI sufiksa (TI sufiksa) po broju kandidata po svakom TI sufiksu.

broj kandidata	*broj SRI obj.*	*primjer TI sufiksa*	*za tv.sufiks*
1	4692	njemo	V.njeti
2	1224	skoga	ADJ.ski
3	104	jemo	V.jeti
4	86	ačkomu	ADJ.ačak
5	12	ak	N.ak
6	33	anomu	ADJ.an
7	31	smo	V.ti
8	7	anom	N.an
9	9	jahu	V.ti
10	3	je	V.jeti
11	2	iju	N.
12	1	em	V.ti
16	6	ome	ADJ.
18	1	im	V.jeti
20	1	o	V.ti
25	1	ima	N.
26	1	om	N.
27	1	a	V.ati
28	1	i	V.ti
31	1	e	V.ti
33	1	u	V.ti

Kad je broj kandidata 1, to znači da je za taj sufiks samo jedan SRI objekt prijavljen. Za primjer uzimo da riječ završava na *njemo*, to nužno znači da je to glagol koji je nastao od tvorbenog sufiksa *njeti*. Nijedan drugi tvorbeni sufiks ne može kreirati takav TI sufiks. Od ukupno 8776 sufiksa više od polovine, tj. 4692 je takvih.

Potrebno je primijetiti opće pravilo da što je kraći TI sufiks, to je veći broj SRI kandidata za taj sufiks (primjer: sufiks "u" ima 33 SRI objekta).

Uz to se može primijetiti da je u stupcima primjera pokazano par primjera *0-type* sufiksa - primjerice za red s brojem kandidata 11, 25 i 26. Recimo za red 11 to znači da imenice u jednom od flektivnih pravila proizvodi sufiks *iju*, a za taj sufiks ima ukupno 11 ostalih SRI objekata koji ga mogu proizvesti.

Prilikom upotrebe ovog registra primijećeno je da se između vrsta riječi najviše konflikata događa između imenica i pridjeva, što je i logično s obzirom na to da imaju slična flektivna pravila.

U nastavku slijedi pregled klasa *Noun*, *Verb* i *Adjective* koje unutar sebe sadrže svu potrebnu logiku i informacije za te vrste riječi.

6.3.5 Imenice

Postoji 5 glavnih flektivnih pravila koja postoje za imenice, a za svako od njih napravljeni su predlošci:

- A-M0 - sklonidba A - ništični nastavak - muški rod
- A-Moe - sklonidba A - nastavak o ili e - muški rod
- A-N - sklonidba A - srednji rod
- E-F - sklonidba E - ženski rod
- I-F - sklonidba I - ženski rod

Slijedi izvadak iz koda koji ilustrira kako to izgleda:

```
class NounType(base.ChangeableWordType):
    def __init__(...)
        self._SUFFIX_TEMPL["A-M0"] = """  ##   SINGULAR
                                  ##   M
                                  ##   ----------
                                  #N   $word_base0
                                  #G   a
                                  #D   u
                                  #A   a|$word_base0
                                  #V   %%Pe|u
                                  #L   u
                                  #I   om|em

                                  ##   PLURAL
                                  #N   %%S%(ext)si
                                  #G   %(pg)s
                                  #D   %%S%(ext)sima
                                  #A   %(ext)se
                                  #V   %%S%(ext)si
```

```
                                #L   %%S%(ext)sima
                                #I   %%S%(ext)sima
                                """
        self.PARAMS_ADD["A-M0"] = {"ext" : ["",
                                            "ov",  "ev"],
                                "pg"  : ["$WORD_BASE%(ext)sa", "i",
                                         "i|$WORD_BASEa",
                                         "iju|$WORD_BASEa"],
                               }
        ...
        self._add_suffixes("A-M0")
        ...
        self._add_suffixes("A-Moe")
        ...
        self._add_suffixes("A-N")
        ...
        self._add_suffixes("E-F")
        ...
        self._add_suffixes("I-F")
```

Što se tiče vrste riječi imenica, klasa je *Noun* s konstruktorom koji izgleda ovako:

```
class Noun(ChangeableWordBase):
    def __init__(self, word_base, gender=None,
                 apply_vc_a=True, ext=None,
                 decl=None, ...):
        ...
```

Parametri su sljedeći:

- word_base - osnovni oblik riječi
- gender - gramatički rod imenice - vrijednost jedinog nepromjenjivog atributa za imenice
- apply_vc_a - treba li se glasovna promjena nepostojano A primjenjivati ili ne (primjer: momak -> momka ili momaka)
- ext - koji je nastavak u množini - bez nastavka, ov, ev, itd.
- decl - u slučaju da sustav ne može odgonetnuti koje flektivno pravilo primijeniti, to se mora eksplicitno reći ovim parametrom.

Primjeri instanciranja objekata klase imenica:

```
>>> auto = Noun("auto", "M", ext="", spec="0")
>>> auto.params_key, auto.params_key
('A-MoeD/+A/MG/#E/0S', 'A-MoeD/+A/MG/#E/0S')
>>> auto.forms["S/N"]
[u'auto']
>>> auto.forms["S/G"]
```

```
[u'auta']
>>> auto.forms["P/I"]
[u'autima']
>>> auto[10]
('P/A', [u'aute'])
>>> auto.forms_flat[0], auto.forms_flat[-1]
(('S/N', [u'auto']), ('P/I', [u'autima']))
```

Većina ovih parametara nije obvezna, pa ukoliko neki od njih nisu dani, sustav ih pokušava sam odgonetnuti ili uzima najšire rješenje. U sljedećem primjeru nismo dali parametar ext koji govori o nastavku u množini. Tada sustav primjenjuje sve nastavke, jer jedan od njih mora biti točan:

```
>>> ugao = Noun("ugao", "M")
>>> ugao.forms_flat
[('S/N', [u'ugao']), ('S/G', [u'ugla']), ('S/D', [u'uglu']),
 ('S/A', [u'ugla', u'ugao']), ('S/V', [u'ugao']),
 ('S/L', [u'uglu']), ('S/I', [u'ugaom', u'uglom', u'uglem']),
 ('P/N', [u'ugli', u'uglovi', u'uglevi']),
 ('P/G', [u'ugla', u'uglova', u'ugleva']),
 ('P/D', [u'uglima', u'uglovima', u'uglevima']),
 ('P/A', [u'ugle', u'uglove', u'ugleve']),
 ('P/V', [u'ugli', u'uglovi', u'uglevi']),
 ('P/L', [u'uglima', u'uglovima', u'uglevima']),
 ('P/I', [u'uglima', u'uglovima', u'uglevima'])]
```

6.3.6 Glagoli

Flektivna pravila za ovu vrstu riječi podijeljena su po glagolskim vremenima, što je prikazano sljedećim izvatkom iz programskog koda. Oznake su PRE - prezent, AOR - aorist, IMP - imperfekt, IMV - imperativ, VA_A - glagolski pridjev radni, VA_P - glagolski pridjev trpni - pasiv, itd.

```
class VerbType(base.ChangeableWordType):
    def __init__(...):
        self._add_suffixes("PRE")
        self._add_suffixes("AOR")
        self._add_suffixes("IMP")
        self._add_suffixes("IMV")
        self._add_suffixes("VA_A")
        self._add_suffixes("VA_P")
        ...
```

Ova vrsta podatka nema nepromjenjivih atributa, niti ima veliki broj različitih ulaznih parametara koji preciznije određuju infleksiju:

```
class Verb(ChangeableWordBase):
    def __init__(self, word_base, ext=None):
        ...
```

Parametar *ext* dodatno određuje na koji način primijeniti infleksijko pravilo (npr. koji prezentski nastavak: am, im, em).

Ono što je bilo vrlo teško odrediti kod glagola je **prezentska osnova**. Glagoli imaju dvije osnove, prezentsku i infinitivnu. Infinitivna se dobije jednostavnim putem iz osnovnog oblika (infinitiv) odbacivanjem nastavaka ti i ći. No za otkrivanje prezentske osnove bilo je potrebno proučiti sva pravila iz knjige "Gramatika hrvatskog jezika, Težak, Babić 2005" [4], te ih korigirati i nadopuniti s vlastito otkrivenim pravilima (pravila iz knjige nisu potpuna i nisu uvijek točna). Pravila koja su ugrađena u programsko rješenje nastala su kombinacijom pravila iz knjige i pravila dobivenim heurističko-statističkim pristupom nad postojećim katalogom riječi koji je prikupljen (pogledati *Korpus*).

U sljedećem odlomku koda popisana su neka od pravila kako dobiti prezentsku osnovu iz infinitiva. Prvi stupac - vrijednost *True* - znači da se to pravilo primjenjuje točno za taj infinitiv - slučajevi pojedinačnih iznimaka, dok vrijednost *False* znači riječ treba završavati na sufiks. Drugi stupac je vrijednost ili sufiks infinitiva, a treći je stupac pravilo za dobivanje prezentske osnove:

```
WORD_BASE_TO_LEXEM_PRE_RULES = [
 (True , unicode("žeti", "utf-8")  , (unicode("žanj", "utf-8"),
                                      "e")),
 (True , "kleti"                   , ("kun"  , "e") ),
 (True , "mljeti"                  , ("melj" , "e") ),
 (True , "stati"                   , ("stan" , "e") ),
 (False, "piti"                    ,[(len("iti")   , "" , "i"),
                                     (len("ti")    , "" , "je")]),
 (False, "lodati"                  , (len("ati")   , ""  , "je")),
 (False, "asti"                    , (len("sti"),   "t" , "e")),
 (False, "isti"                    , (len("sti"),   "z" , "e")),
 (False, "sti"                     , (len("sti"),   ""  , "e")),
 (False, "mjeti"                   , (len("mjeti"), "mij","e")),
 (False, "zvati"                   , (len("zvati"), "zov","e")),
 (False, "ojevati"                 , (len("evati"), "uj", "e")),
 (False, "uti"                     , (len("uti")   , ""  , "e")),
 (False, "eti"                     , (len("eti")   , ""  , "e")),
 (False, "iti"                     , (len("iti")   , ""  , "i")),
 (False, unicode("ući" , "utf-8"), (len(u"ci"),
                                    unicode("č", "utf-8"), "e")),
```

```
  (False, unicode("ći", "utf-8"), [(len("ci"),
                                    unicode("k", "utf-8"), "je"),
                                   (len("ci"),
                                    unicode("g", "utf-8"), "je"),
                                  ]),
```

Primijetimo da neke infinitivne osnove imaju po dvije moguće prezentske osnove (npr. "ći"), od kojih je nekad samo jedna valjana, ali postoje slučajevi kad su obje valjane.

Primjer objekta glagola *hodati* i *krenuti* koji demonstrira i flektivna pravila i automatsko otkrivanje prezentske osnove:

```
>>> krenuti = Verb("krenuti")
>>> print krenuti
W(u'krenuti'=V)

>>> krenuti.params_key, krenuti.params_key
('$E', '$E')
>>> krenuti.word_base, krenuti.word_lexem_pre, krenuti.word_lexem_inf
(u'krenuti', u'kren', u'krenu')

>>> hodati = Verb("hodati", ext="a")
>>> print hodati
W(u'hodati'=V)
>>> hodati.pp_forms("PRE")
PRE/S/1 [u'hodam'], PRE/S/2 [u'hoda\u0161'], PRE/S/3 [u'hoda'],
PRE/P/1 [u'hodamo'], PRE/P/2 [u'hodate'], PRE/P/3 [u'hodaju'],

>>> hodati.pp_forms("AOR")
AOR/S/1 [u'hodah'], AOR/S/2 [u'hoda'], AOR/S/3 [u'hoda'],
AOR/P/1 [u'hodasmo'], AOR/P/2 [u'hodaste'], AOR/P/3 [u'hoda\u0161e'],
...
```

6.3.7 Pridjevi

Flektivna pravila za pridjeve samo su dva - za određene i neodređene pridjeve. Neki pridjevi mogu imati i određeni i neodređeni oblik - pa se za njih primjenjuje unija ta dva flektivna pravila (infl. pravilo P_N+O):

```
class AdjectiveType(base.ChangeableWordType):
    WORD_TYPE.add_suffixes("P_N", "MANY", "", "P_N#-",
    WORD_TYPE.add_suffixes("P_O", "MANY", "", "P_O#-",
    WORD_TYPE.add_suffixes("P_N+O", "MANY", "", "P_NO#-",
```

Za konkretni objekt pridjeva konstruktor je ovakav:

```
class Adjective(ChangeableWordBase):
    def __init__(self, word_base, apply_vc_a=True,
```

```
                has_neo=True, has_com=True,
                com_wp1_suff=None):
```

Značenje parametara je sljedeće:

- apply_vc_a - primijeniti glasovnu promjenu nepostojano a ili ne
- has_neo - ima li neodređeni oblik ili ne
- has_com - ima li komparativ i superlativ ili ne
- com_wp1_suff - ukoliko ima komparativ i superlativ, koji je nastavak u komparativu - npr. "ije" ili "je"

Kao i kod ostalih opisanih vrsta riječi, ako neki od atributa nije unesen, sustav ga pokušava sam odgonetnuti.

Primjeri objekata pridjeva:

```
>>> bijel2 = Adjective("bijel")
>>> print bijel2
W(u'bijel'=ADJ)
>>> bijel2.forms["P_N/S/N/M"]
[u'bijel']
>>> bijel2.forms["COM/S/N/M"]
[u'bjelji']

>>> otporan = Adjective("otporan")
>>> otporan[10]
('P_N/S/A/F', [u'otpornu'])
>>> otporan[100]
('COM/S/L/F', [u'otpornijoj'])

>>> bodljikav = Adjective("bodljikav", apply_vc_a=False)
>>> bodljikav[10]
('P_N/S/A/F', [u'bodljikavu'])
>>> bodljikav[100]
('COM/S/L/F', [u'bodljikavijoj'])
>>> len(bodljikav.forms_stats)
43
```

Promjenjivi atributi pridjeva su rod, broj, padež i komparacija. Zbog toga ima dosta kombinacija promjenjivih atributa za konkretni pridjev, a u najbrojnijem ih je slučaju 168 = (3+1) * 2 * 7 * 3. Faktor 3+1 je za slučaj da pridjev u pozitivu ima i određeni i neodređeni oblik. Sljedeći primjeri demonstriraju broj oblika riječi:

```
>>> crven = Adjective("crven")
>>> print crven.com_base, crven.com_base_ws, crven.com_lexem
crveniji (u'crven', u'iji') crvenij
>>> print crven.sup_base, crven.sup_base_ws, crven.sup_lexem
```

```
najcrveniji (u'najcrven', u'iji') najcrvenij

All possible forms count values
>>> len(crven.ATTR_VALUES_ALL), len(crven.ATTR_VALUES_NO_NEO),
    len(crven.ATTR_VALUES_NO_COM), len(crven.ATTR_VALUES_NO_NEO_COM)
(168, 126, 84, 42)

1st case - has all
---------------------------
>>> len(crven.attr_values)
168
>>> crven.forms_flat[0],crven.forms_flat[-1]
(('P_N/S/N/M', [u'crven']), ('SUP/P/I/N', [u'najcrvenijim',
u'najcrvenijima']))

2rd case - has no neodr.form but has comp./supl.:
-----------------------------------------------
>>> hrvatski = Adjective("hrvatski", has_neo=False)
>>> len(hrvatski.attr_values)
126
>>> print hrvatski.forms_flat[0]
('P_O/S/N/M', [u'hrvatski'])
>>> print hrvatski["COM/S/I/N"]
[u'hrvatskijim']

3rd case - has neodr.form but no comp./supl.:
---------------------------------------
>>> perov = Adjective("perov", has_com=False)
>>> len(perov.attr_values)
84
>>> print perov["P_O/S/N/M"]
[u'perovi']

4rd case - no neodr.form and no comp./supl.:
-----------------------------------------
>>> brodski = Adjective("brodski", has_neo=False, has_com=False)
>>> len(brodski.attr_values)
42
>>> print brodski.forms_flat[0]
('P_O/S/N/M', [u'brodski'])
```

6.3.8 Algoritam za prepoznavanje osnovnih oblika riječi

Ovaj dio programskog rješenja sadrži središnji i najvažniji dio za proces prepoznavanja osnovne riječi. Ulaz za proces je popis oblika riječi koje se pojavljuju u korpusu i registar TI sufiksa. Popis oblika riječi dobiven procesom opojavničenje (engl. *tokenization*) skupljenog korpusa (poglavlje *Korpus*). Prema tome, sve što je dosad opisano u ovom poglavlju i prethodnim poglavljima je preduvjet za rad ovog

dijela sustava.

Ovaj dio programskog rješenja mogao bi se nazvati i **sustav za obrnutu infleksiju**.

Programska logika treba imati pristup:

- katalogu riječi s njihovim učestalostima (*frequency*) [47],
- registru svih tvorbeno-flektivnih sufiksa,
- Noun, Verb, Adjective klasama za instanciranje objekata riječi,
- bazi u koju će spremiti skupljene rezultate.

Skraćeni algoritam izvadak iz izvornog koda:

```
class IWordTypeRecognizer(object):

    def detect(self, level=2, ...):

        for suff, sri_list in suff_registry:
            if len(suff)<level: # shorter suffix longer search/more
                                # candidates - better results
                continue
            word_list = list(self.filter_endswith(suff))
            for i_word, item in enumerate(word_list):
                for i_sri, sri in enumerate(sri_list_reduced):
                    word_base = word_prepared[:-len(suff)]
                    word_base = word_base+sri.word_obj.word_base
                    params_key=sri.word_obj.params_key
                    params_init=sri.word_obj.params_init
                    word_obj_found = self.word_base_exists(
                                        word_type=word_type,
                                        word_base=word_base,
                                        params_key=params_key)
                    if word_obj_found:
                        continue
                    word_class = sri.word_obj.__class__
                    word_obj = word_class(word_base=word_base,
                                          sri=sri,
                                          \*\*params_init)
                    if not_valid(word_obj):
                        self.reject(word_obj, None)
                        continue
                    for wf in word_obj.forms_stats.keys():
                        result = self.get_word_freq(wf)
                        word_obj.set_form_freq(wf, word_freq,
                                               word_id)
```

47 U nekoj od sljedećih faza koristit će se internetski izvori za informaciju o postojanju i učestalosti riječi - jednostavan primjer bio bi postaviti upit pretraživačkom servisu *http://www.google.com.*

```
                    self.save_new(word_type, word_base,
                                  word_obj.params_key,
                                  word_obj, sri)

        self._detect_reduction()

    # --------------------------

    def _detect_reduction(self):
        for word_obj in self.word_objs_list:
            this_exists_set = word_obj.get_forms_exists_set()
            for wf in this_exists_set:
                for other_obj in self.word_forms[wf]:
                    if already_compared:
                        continue
                    other_exists_set = other_obj.get_forms_exists_set()

                    has_this  = this_exists_set - other_exists_set
                    has_other = other_exists_set - this_exists_set
                    has_both  = this_exists_set & other_exists_set

                    if len(has_forms_both)>=5 or len(has_both)>=3:
                        word_obj.similar_words[other_obj.key]=(
                                                other_obj,
                                                len(has_forms_both),
                                                len(has_both))
                        other_obj.similar_words[word_obj.key]=(
                                                 word_obj,
                                                 len(has_forms_both),
                                                 len(has_both))
                    if has_this and has_other:
                        # will leave both - both are good
                        pass
                    elif has_this:
                        # this is better, reject other
                        self.reject(other_obj, word_obj)
                    elif has_other:
                        # other is better, reject this
                        self.reject(word_obj, other_obj)
                    else:
                        if word_obj.is_better_than(other_obj):
                            # this is better, reject other
                            self.reject(other_obj, word_obj)
                        else:
                            # other is better, reject this
                            self.reject(word_obj, other_obj):

class ChangeableWordBase(object):
    def is_better_than(self, other):
        has_base_this  = self.forms_stats[self.word_base][0]>0
```

```
        has_base_other = other.forms_stats[other.word_base][0]>0
        suff_len_diff = len(self.sri.wts.suff_value)-len(
                            other.sri.wts.suff_value)
        form_len_diff = len(self.forms_flat)-len(other.forms_flat)
```

Osnovni algoritam radi po sljedećem principu (metoda **detect()**)

```
za svaki TI sufiks
   za svaki SRI objekt za trenutni TI sufiks
       nađi sve riječi koje završavaju s TI sufiksom
       za svaku tu riječ
           odvoji sufiks od potencijalne osnove
           od potencijalne osnove napravi osnovni oblik riječi
           napravi objekt riječi iz te osnovne riječi
           skupi statistiku za sve oblike tog objekta riječi
           ako je sve dosad prošlo bez problema
               spremi objekt u listu kandidata
```

Kad sve to završi dolazi se do dodatnog dijela koda koji za sve skupljene objekte detektira "dvostruke" - razlike skupova oblika riječi je prazan skup. Za svaki skup dvostrukih objekata bira onaj koji je najbolji, a ostale se briše. Glavna logika implementirana je u metodi **detect_reduction()**, a biranje najbolje riječi implementirano je u **ChangeableWordBase.is_better_than()**.

Na kraju, algoritam sprema podatke natrag u bazu, s time da se ne bilježi samo osnovni objekt riječi, već i svi novonastali oblici koji nisu postojali u katalogu riječi. Ovaj dio koda nije prikazan, no radi se o sql naredbama koje ubacuju zapise u *WordObj*, *WordForm* i *Word* tablice (tablice su opisane u poglavljima WordObj i WordForm i Word).

Prva stvar koju treba podrobnije objasniti je **kako dobiti iz oblika i sufiksa korijenski oblik riječi**. To je napravljeno prema ovom principu:

```
word_base = word_prepared[:-len(suff)]
word_base = word_base+sri.word_obj.word_base
```

Za konkretni primjer recimo da je TI sufiks "osti" koji se odnosi na imenice s tvorbenim sufiksom *ost*. Prepoznat je oblik riječi radost - *radosti*. U tom je slučaju flektivni dio nastavka u *osti* vrijednost "i". Vrijednost *word_base* dobije se odvajanjem "osti" i dodavanjem "ost", tj.:

radosti - osti -> rad + ost -> radost

Međutim, što ako je primjer TI sufiks "ošću" koji ima u sebi primjenjene dvije glasovne promjene, a odnosi se na iste imenice (tv. sufiks "ost")? Prepoznat je oblik

riječi radost - *radošću*. Pravi flektivni sufiks u ovom slučaju je "ju", no zbog dvostruke glasovne promjene "st + j" -> "sć" -> "šć", on se "gubi". No način na koji se računa u našem slučaju je drugačiji:

radošću - ošću -> rad + ost -> radost

I sve dok glasovne promjene ne mijenjaju broj slova u riječi, ovaj način radi dobro. U slučaju da glasovna promjena "proguta" slovo, onda će osnova biti pogrešna. No obično je tako da ako riječi doživljavaju glasovne promjene broj slova prilično se rijetko mijenja. A čak i u tom slučaju vrlo je vjerojatno da će neki drugi oblik te riječi, koji nije doživio problematičnu promjenu, biti prepoznat, pa će se zbog tog drugog oblika i ovaj primarno nedetektirani primjer ipak otkriti (npr. da sustav nije otkrio *radošću* prema TI sufiksu, a da je otkrio *radosti*, posredno bi onda otkrio i *radošću* preko *radosti*).

Sljedeće što treba dodatno pojasniti je **eliminacija kandidata**, odnosno usporedba oblika riječi i biranje najbolje. Prva se eliminacija događa u *detect()* funkciji gdje se ne spremaju riječi koje su već spremljene (potpuni duplikat). U funkciji **detect_reduction()** koristi se rječnik *self.word_forms* koji ima za svaki oblik riječi popis svih tvorbeno-flektivnih objekata riječi (instance *ChangeableWordBase* klase) koji tvore taj oblik riječi.

Radi se analiza usporedbe skupova oblika riječi koje se pojavljuju u katalogu. Kad se uspoređuju dvije riječi:

- ako obje riječi imaju neke oblike koje druga riječ nema, obje se zadržavaju
- ako jedna riječ ima sve oblike druge riječi, te ima još neke koje druga riječ nema (nadskup), druga se riječ odbacuje
- ako obje riječi imaju isti skup zajedničkih oblika, zove se funkcija *ChangeableWordBase.is_better_than()* koja na osnovi sljedećih podataka bira bolju, a lošiju odbacuje:
  - has_base_* - je li osnovni oblik u katalogu riječi
  - suff_len_diff - duljine TI sufiksa
  - form_len_diff - ukupnog broja svih oblika - prijašnja se analiza odnosila samo na one koje su se pojavile.

Optimalni se rezultati dobivaju kad riječ ima puno oblika koji se pojavljuju u ulaznom katalogu riječi (rječniku). U slučaju da riječ ima samo par oblika koji se

pojavljuju, onda je puno vjerojatnija pogreška u prepoznavanju tipa riječi ili osnovnog oblika ili flektivnih pravila koje treba primijeniti na tu riječ. Najčešći problemi koji su uočeni su:

- u nekim se slučajevima znaju miješati imenice i pridjevi zbog sličnih flektivnih pravila,
- treba li primjenjivati glasovnu promjenu nepostojano a ili ne.

Ono što je još potrebno upotrijebiti za bolje prepoznavanje u nekoj od sljedećih faza, je uvođenje jednog poslijeprocesnog (*postprocessing*) koraka koji koristi podatke koji se mogu dobiti iz SRI, odnosno WTS objekata, a to je tvorbeni sufiks.

Primjer

Imenica "zakon" protumačena je kao pridjev i kao imenica na osnovu "0-type" TI sufiksa. Odabran je pridjev. Da postoji dodatni korak koji provjerava postoji li pridjev koji nastaje tvorbenim sufiksom "ski" na imenicu - tj. "zakon-ski", onda bi sustav znao da zakon ne može biti pridjev, nego mora biti imenica.

Dodatne informacije koje nisu u najboljoj mjeri korištene prilikom biranja najboljih kandidata je plodnost tvorbenog sufiksa, a nešto širi pristup trebao bi iskoristiti statistiku uspješnosti nekih sufiksa.

Ono što se još može iskoristiti je informacija o obliku riječi u strukturi rečenice. pridjevi idu uz pridjeve, no češće uz imenice, prijedlozi diktiraju padež imenica i pridjeva koji slijede, prilozi stoje blizu glagola i slično. Te su informacije dodatne statističke informacije koje zasigurno mogu poboljšati točnost prepoznavanja. No ovaj je proces računalno intenzivan, jer je potrebno ući u strukturu svih rečenica gdje se pojedini oblik pojavio.

Dodatno objašnjenje potrebno je za ulazni parametar **level**. Algoritam je linearan, no i "spor". U početku je bio definiran tako da pretražuje cijeli pretražni prostor, no kasnije zbog problema i testiranja, ali i problema ograničenih računalnih resursa, napravljene su iscrpne analize i poboljšanja u smjeru ubrzanja. Napravljen je niz poboljšanja u smislu da se nijedan dio logike ne vrti bespotrebno, no krajnji rezultat opet nije bio dovoljno brz. Algoritam ima potrebu pretražiti sve, jer ako to ne napravi, postoji mogućnost da će izmaći neko dobro/optimalno rješenje. Zbog praktičnih razloga morao se uvesti dodatni filter - *level*. Princip rada parametra *level*

je vrlo jednostavan: napraviti filter po duljini TI sufiksa. Primjer: ako je level=2 onda se od svih TI sufiksa testiraju samo oni koji imaju duljinu 2 i više. Optimum između brzine i održanja broja prepoznatih riječi je davao level 2 ili 3. To je značajno ubrzalo rad, a broj riječi koji je prepoznat nije znatno pao. Nedostatak je što je jedan dio riječi koji bi bio prepoznat po tvorbenim sufiksima *0-type* za level 0 ili 1.

U nastavku se daje pregled testiranja na 1989 riječi za različite level-e gdje se mogu primijetiti razlike u brzini izvođenja i broju otkrivenih riječi:

```
> py detect all 4
Duration 0:00:22.219000
from 1989 matched 1048 forms within 282 word objects
(bad/dupl/ignored 320)
    with exists freq 1280 (avg ~4.5), and occ.fr eq 66404
    (avg ~235.5). output in testing_w_br-lev-4.out.

> py detect all 3
Duration 0:00:57.765000
from 1989 matched 1549 forms within 549 word objects (bad/dupl/ignored
    2353) with exists freq 2076 (avg ~3.8), and occ.freq 100935
    (avg ~183.9). output in testing_w_br-lev-3.out.

> py detect all 2
Duration 0:02:05.532000
from 1989 matched 1768 forms within 769 word objects (bad/dupl/ignored
    6083) with exists freq 2495 (avg ~3.2), and occ.freq 110514
    (avg ~143.7). output in testing_w_br.out.

> py detect all 1
Duration 0:08:52.718000
from 1989 matched 1823 forms within 790 word objects (bad/dupl/ignored
    20434) with exists freq 2600 (avg ~3.3), and occ.freq 117979
    (avg ~149.3). output in testing_w_br-lev-1.out.

> py detect all 0
više od 20 minuta, broj riječi nije značajno veći kao za level 1
```

Obrada nad rječnikom iz korpusa rađena je s level-om 3, a bila je podijeljena po početnim slovima riječi (sve riječi koje počinju s "a", pa onda sa "b", itd.), te je trajala oko 15 minuta po jednom slovu.

Sljedeći primjeri izvađeni su iz *doc-test* sustava koji je korišten za kontrolu rada i poboljšanje kvalitete sustava, a ilustriraju rezultate prepoznavanja objekata riječi iz popisa oblika riječi - primijetiti da su korišteni različiti *level*-i:

```
>>> word_list = [
...   "Broj    84"
... , "broji   34"
... , "Brojila  28"
... , "broje   23"
... , "brojeći 22"
... , "brojim   7"
... , "brojimo  5"
... , "brojiš   4"
... , "brojahu  2"
... , "brojaše  1"
... , "brojite  1"
... , "-brijestovu 1"
... , "brijestovi 1"
... , "-brijestove 1"
... , "-brijestova 1"
... ]

Lowest quality, but fastest
>>> wdh = test_it(word_list, level=4)
" 10/  183 -> brojati              (u'V-XX_-_JATI-je\\u0107i-0')
    84/broj,34/broji,23/broje,22/broje\xe6i,7/brojim,
    5/brojimo,4/broji\x9a,2/brojahu,1/brojite,1/broja\x9ae"

>>> wdh = test_it(word_list, level=3)
" 10/  183 -> brojati              (u'V-XX_-_JATI-je\\u0107i-0')
    84/broj,34/broji,23/broje,22/broje\xe6i,7/brojim,5/brojimo,
    4/broji\x9a,2/brojahu,1/brojite,1/broja\x9ae"
" 10/  127 -> brojiti              (u'V-XX_-_ITI-ila-0')
    34/broji,28/brojila,23/broje,22/broje\xe6i,7/brojim,5/brojimo,
    4/broji\x9a,2/brojahu,1/brojite,1/broja\x9ae"
"  4/    4 -> brijestov            (u'ADJ-od1_iji_-_OV-ovi-0')
    1/brijestovu,1/brijestovi,1/brijestove,1/brijestova"
'  1/   28 -> brojilo              (confirmed) 28/brojila'

Slowest, best quality - brojil and brojilo are confirmed
because Brojilo starts with big letter -> Noun
>>> wdh = test_it(word_list, level=0)
" 10/  183 -> brojati              (u'V-XX_-_JATI-je\\u0107i-0')
    84/broj,34/broji,23/broje,22/broje\xe6i,7/brojim,5/brojimo,4/
    broji\x9a,2/brojahu,1/brojite,1/broja\x9ae"
" 10/  127 -> brojiti              (u'V-XX_-_ITI-ila-0')
    34/broji,28/brojila,23/broje,22/broje\xe6i,7/brojim,
    5/brojimo,4/broji\x9a,2/brojahu,1/brojite,1/broja\x9ae"
"  4/    4 -> brijestov            (u'ADJ-od1_iji_-_OV-ovi-0')
```

```
    1/brijestovu,1/brijestovi,1/brijestove,1/brijestova"
'  3/  141 -> broj                 (confirmed)
84/broj,34/broji,23/broje'
'  1/   28 -> brojilo              (confirmed) 28/brojila'
'  1/   28 -> brojil               (confirmed) 28/brojila'


VERB with ADJ overlap
>>> word_list = [
...   "broj    84"
... , "broji   34"
... , "brojan   23"
... , "brojni   23"
... , "brojani   23"
... , "brojnoga 23"
... , "najbrojniji 23"
... ]

Both are accepted since there is diff in set they use - verb has no
comparative and no neo and no apply-vc-a
>>> wdh = test_it(word_list) # doctest: +ELLIPSIS
"  4/  164 -> brojati              (u'V-XX_-_JATI-ji-0')
    84/broj,34/broji,23/brojani,23/brojan"
"  4/   92 -> brojan               (u'ADJ-oo_iji_-_AN-niji-0')
    23/najbrojniji,23/brojnoga,23/brojni,23/brojan"
```

6.4 Odabir glavne riječi

Nakon što su se iz oblika riječi prepoznale osnovne riječi (objekti riječi), rezultati su ubačeni u Word, WordObj i WordForm tablice (pogledati podrobnije opise tablica u poglavljima Word, WordObj i WordForm). Ali za svaki Word zapis potrebno je odrediti njen osnovni oblik. No problem su **hiponimi**. Hiponim je riječ koja postoji kao oblik za više riječi objekata.

Primjer

Riječ biti - infinitiv pomoćne riječi "biti" - postojati, i "biti" - infinitiv riječi "biti" - tući. U tom i svim sličnim slučajevima moramo se odlučiti za jednu od njih.

Za rješavanje tog problema nije korištena posebno napredna logika (u redoslijedu):

- uzmi onu riječ koja je potvrđena
- uzmi onu riječ koja ima veću učestalost osnovnog oblika veća.

Ovaj jednostavan način pokazao je dobre rezultate, no u nekoj od sljedećih faza vjerojatno će se revidirati i pokušati naći još bolji.

6.5 Rezultat normalizacije

Pri kreiranju objekata riječi u sustavu prepoznavanja riječi, kreiraju se svi mogući oblici riječi po flektivnim pravilima. Ukupan broj oblika u tom slučaju nije isti kao i ukupan broj različitih oblika riječi, jer se neki oblici po vrijednosti ponavljaju. U bazu se ne spremaju oblici po infleksiji, već samo različiti oblici. To je napravljeno zbog toga što za potrebe pretraživanja nije potrebno imati višestruko zapisane iste oblike iste riječi (lematizacija je funkcija *oblik -> korijen*).

Primjer

Za imenice akuzativ je u pravilu isti kao i nominativ ili genitiv, a jednako tako je lokativ isti kao i dativ. Imenica "čovjek" u jednini ima genitiv "čovjeka" koji je isti kao i akuzativ "čovjeka", te dativ "čovjeku" koji je isti kao i lokativ "čovjeku". U bazu će ući po jednom oblici "čovjeka" i "čovjeku".

Prvo će se napraviti analiza smanjenja pretražnog prostora zbog postignute normalizacije pomoću *skupa zaustavnih riječi*. Zbog prije spomenutog razloga brojevi oblika koji su u bazi i bit će dani u sljedećim tablicama, razlikovat će se od brojeva u poglavljima *Zaustavne riječi - prilozi, prijedlozi, veznici, usklici i čestice*, *Zaustavne riječi - glagoli* i *Zaustavne riječi - zamjenice*.

Ukupno je u sustavu prepoznato oko 430.000 različitih pojavnica (engl. *token*) u tekstu. Ukupno ima oko 2300 dokumenata u skupljenom korpusu, a u svima njima ima oko 12.5 milijuna pojavnica, odnosno oko 5400 pojavnica po dokumentu.

Prilikom procesa opojavničenja (engl. *tokenization*) u bazu se sprema cjelokupni sadržaj teksta, što znači da se osim samih riječi, označavaju, klasificiraju i spremaju i pojavnice za oznake novog odlomka, poglavlja, oznake početka i kraja rečenice, ostali interpunkcijski znakovi, skraćenice, neklasificirane pojavnice itd. Više pojedinosti o tome može se naći u poglavlju *Opojavničenje na odjeljke, rečenice i riječi.*

Sljedeća tablica prikazuje udjele riječi po vrsti pojavnica:

vrsta jed.	*broj jed.*	*udio*	*učestalost*	*udio*
Riječi	396.835	93.0 %	10.291.316	82.5 %
Brojevi	22.275	5.2 %	372.817	3.0 %
Ostali	7.610	1.8 %	107.719	0.9 %
Interpunkcije	158	0.0 %	1.198.544	9.6 %
Prazno	17	0.0 %	500.307	4.0 %
UKUPNO	426.895	-	12.470.703	-

U ovoj će se analizi promatrati samo one pojavnice vrste *riječi*, jer oni su primarna meta pretraživanja, te se nad njima vrši normalizacija. Govoreći o brojevima, analiza će se provesti nad oko 400.000 pojavnica.

Pretražni prostor je broj pojavnica vrste *riječ* u svim dokumentima, što znači oko 10 milijuna pojavnica, što u prosjeku na 2300 dokumenata iznosi oko 4300 pojavnica.

U poglavlju *Opojavničenje na odjeljke, rečenice i riječi* objašnjen je način dobivanja *skupa zaustavnih riječi*. Ti objekti riječi i njihovi oblici ubačeni su u bazu riječi.

U poglavlju *Svođenje riječi na osnovni oblik* objašnjen je proces prepoznavanja korijenskog oblika riječi iz skupa oblika riječi dobivenih iz skupljenog korpusa. I ti objekti riječi i njihovi oblici ubačeni su u bazu riječi.

U poglavlju *Odabir glavne riječi* objašnjen je postupak kako su postojeći oblici riječi povezani s objektima riječi (engl. *stopwords* i prepoznate) preko *Word.wordobj_main* relacije. Ta veza omogućuje normalizaciju, a time smanjenje pretražnog prostora i svođenje traženog i teksta po kojem se traži na iste oblike.

Sljedeća tablica daje statistiku:

vrsta jed.	*broj jed.*	*udio*	*učestalost*	*udio*	*prosj.učest.*
prepoznate	283.963	71.6 %	4.991.554	48.5 %	1.757
zaustavne	3.174	0.8 %	3.920.489	38.1 %	123.518
neprepoznate	109.698	27.6 %	1.379.273	13.4 %	1.257
UKUPNO	396.835	-	10.291.316	-	2.593

Iz tablice se može vidjeti da skup *zaustavnih* riječi zaista ima vrlo visoki udio učestalosti (38%) u odnosu na udio u broju pojavnica (1%), odnosno prosječnu

učestalost 80-tak puta veću u odnosu na prepoznate i neprepoznate riječi. U pretražnom se prostoru *zaustavnih* riječi ne uzimaju, pa je samo **zbog utjecaja skupa zaustavnih riječi pretražni prostor smanjen za gotovo 40%**.

Zanimljivo je promatrati i postotak neprepoznatih riječi, gdje udio u broju pojavnica nije zanemariv (gotovo 30%), no pravi se utjecaj vidi gledajući ujedno i vrijednost udjela u ukupnom pretražnom prostoru - što je 13%. Iscrpnija će se analiza napraviti u nastavku, no prvo slijedi analiza skupa zaustavnih riječi.

Gledajući samo *zaustavne rije*či, sljedeća tablica daje osnovni pregled s obzirom jesu li se riječi pojavile ili ne:

pojavile su se?	***broj obj.riječi***	***udio***	***broj obl.riječi***	***udio***
ne	55	11.5 %	69	5.5 %
DA	425	88.5 %	1194	94.5 %
UKUPNO	480	-	1263	_

Dva su razloga zašto se neki oblici nisu pojavili:

1. postoji isti oblik riječi na drugom objektu riječi
2. oblik riječi nije se pojavio u korpusu jer je zastario ili vrlo rijetko u upotrebi.

Primjeri za 1) dani su u sljedećoj tablici:

id	***riječ***	***vrsta riječi***	***broj oblika***
61	eto	EXCL	1
62	eto	PART	-
16	blizu	ADV	1
17	blizu	PREP	-
299	oko	EXCL	1
300	oko	PREP	-
itd.			

Stupac *broj oblika* daje informaciju o tome koliko oblika riječi ima ovaj objekt riječi odabran kao *glavni* (word_main). Iz tog se stupca vidi da recimo za riječ *eto* postoje dva objekta riječi, 61 i 62, a objekt 61 odabran je kao *glavni* predstavnik oblika riječi *eto*.

Primjeri za razlog 2) su u sljedećoj tablici:

id	*riječ*	*vrsta riječi*	*broj oblika*
71	gdjekakav	PRON.NEO.CH2	-
75	gic	EXCL	-
itd.			

Sljedeća tablica prikazuje analitiku *zaustavnih riječi* po vrsti riječi:

vrsta riječi	*broj riječi*	*udio*	*broj oblika*	*udio*	*učestalost*	*udio*
ADV	216	50.8 %	216	20.7 %	253.555	6.5 %
CONJ	30	7.1 %	30	2.9 %	864.553	22.2 %
EXCL	47	11.1 %	47	4.5 %	477.035	12.3 %
PART	8	1.9 %	8	0.8 %	307.694	7.9 %
PREP	55	12.9 %	55	5.3 %	491.551	12.6 %
PRON	61	14.4 %	534	51.2 %	861.484	22.2 %
V	8	1.9 %	153	14.7 %	631.187	16.2 %
0	425		1043		3.920.489	

Posebno je zanimljiva vrsta riječi glagoli:

glagol	*broj oblika*	*udio*	*učestalost*	*udio*
biti	45	24.5 %	500.375	79.2 %
bivati	9	4.9 %	351	0.1 %
htjeti	33	17.9 %	68.562	10.8 %
morati	18	9.8 %	15.684	2.5 %
moći	19	10.3 %	30.469	4.8 %
smjeti	18	9.8 %	4.340	0.7 %
trebati	18	9.8 %	8.520	1.3 %
željeti	24	13.0 %	3.819	0.6 %
UKUPNO	154	-	631.187	-

Zanimljivo je vidjeti udio od oko 80% koju ima riječ *biti* od svih *stopwords* glagola, a s učestalošću od 0.5 milijuna čini oko 5% cjelokupnog korpusa.

U ovom će se odjeljku napraviti analiza **neprepoznatih oblika riječi**. Već je ranije

navedeno da je udio neprepoznatih riječi u broju pojavnica gotovo 30%, a u ukupnom pretražnom prostoru 13%. Razloga zašto neke riječi nisu prepoznate ima dosta, a navest će se važniji:

1. U korpusu su riječi iz drugog jezika - najčešće engleskog, vrlo česti su izvor *Narodne novine* (npr. riječ 'off'). Takvih riječi je dosta i imaju dosta veliki udio u učestalim neprepoznatim riječima.
2. Zbog problema što je algoritam za prepoznavanje osnovnog oblika riječi vrlo zahtjevan za resurse, a platforma na kojoj su se obavljale obrade je za takve vrste obrada neprilagođena (pogledati *Sklopovsko okruženje* za pojedinosti). Zbog toga se obrada prepoznavanja morala vršiti s dubinom prepoznavanja stupnja 3 (level = 3, pogledati **level** u *Algoritam za prepoznavanje osnovnih oblika riječi*). Posljedica toga je da stupanj prepoznavanja nije bio na najvišoj razini, pa stoga nisu prepoznate neke riječi koje su možda mogle biti prepoznate (npr. na osnovu *0-type* TI sufiksa).
3. Neke se promjenjive riječi pojavljuju u malom broju oblika - pa je teško prepoznati vrstu riječi i ostale oblike (što može biti i posljedica prethodne točke). Primjer ime *Mehujael* pojavljuje se samo jednom i ne može se prepoznati po tvorbeno-flektivnom nastavku koja je vrsta riječi.
4. Neke vrste riječi imaju pravopisnu pogrešku, npr. sedmerostr.
5. Neke riječi pripadaju u skup nepromjenjivih no nisu popisane (npr. posred) - pogledati Zaustavne riječi - prilozi, prijedlozi, veznici, usklici i čestice za pojedinosti.
6. Zbog pogreške ili neobrađenih stanja u algoritmima za opojavničenje i prepoznavanje osnovnih oblika riječi.

Smanjenje broja neprepoznatih oblika riječi jedan je od prvih zadataka u nastavku ovog projekta. Za većinu ovih točaka problem nije moguće na jednostavan način riješiti unaprjeđenjem sustava, pa je optimalan put ručno označavanje riječi, po obrnutom poretku mjere učestalosti u korpusu - prvo najčešće riječi, a onda rjeđe i

rjeđe.

Što se tiče točke 6), potrebno je analizirati zašto neke riječi nisu prepoznate, te unaprijediti algoritme u smislu otklanjanja nedostataka i unaprjeđenjem sustava. I u tom slučaju redoslijed treba biti od najčešćih riječi prema rjeđim. U nekim će se slučajevima otkriti da problem nije u algoritmu, već je uzrok nedovoljno opsežno prepoznavanje, što je objašnjeno u točki 2).

Potrebno je naglasiti da je u smislu pretraživanja "neuspjela" normalizacija samo nad promjenjivim riječima hrvatskog jezika, dok nad svim ostalima normalizacija u ovom smislu ni ne postoji (lematizacija).

Cilj lematizacije za potrebe pretraživanja je:

1. da se smanji pretražni prostor - lematizacija teksta koji se pretražuje
2. da se lematiziraju riječi teksta koji se traži i po kojem se traži.

U tom smislu ako promatramo "neuspjelo" lematizirane riječi (promjenjive), koje su navedene pod brojem 3) (nedovoljan broj oblika), neuspjelost je samo za potrebe cilja 2), jer za 1) to nije moguće - odnosno normalizacija jednog oblika riječi u jedan osnovni oblik nema učinka.

U ovom će se odjeljku napraviti analiza **prepoznatih oblika riječi**.

vrsta riječi	***broj riječi***	***udio***	***broj oblika***	***udio***	***obl/rij.***	***učestalost***	***udio***
ADJ	22.433	25.2 %	80.602	28.4 %	3.6	1.215.369	24.3 %
N	37.430	42.0 %	97.042	34.2 %	2.6	2.273.035	45.5 %
V	29.208	32.8 %	106.319	37.4 %	3.6	1.503.150	30.1 %
UKUPNO	89.071		283.963		3.2	4.991.554	

Stupac broj riječi govori o osnovnim oblicima riječ koji idu u tablicu WordObj (još se nazivaju i objekti riječi ili korijenske riječi). Stupac broj oblika govori o broju oblika koji se pojavljuju barem jednom u korpusu, a stupac učestalost je zbroj svih učestalosti u korpusu. Po broju riječi imenice su dominantne (42%), a zanimljivo je da su glagoli po broju oblika vodeći (37%). Međutim ako gledamo po učestalosti, imenice su opet dominantne (45%).

Algoritam za traženje korijenskog oblika riječi nije u mogućnosti uvijek odrediti točno vrstu riječi ili neke parametre riječi (npr. primjenjuje li se nepostojano a ili ne primjenjuje).

No za potrebe pretraživanja to nije ni toliko bitno, jer na način kako je lematiziran tekst koji se traži, na isti način se lematizira i tekst po kojem se traži, te je učinak isti kao da je algoritam pronašao točnu vrstu riječi.

Stupac *obl/rij.* daje prosječan broj oblika po jednoj riječi, i za primijetiti je da imenice, glagoli i pridjevi imaju sličan prosjek - od 2.6-3.6 oblika za sve tri vrste riječi. Taj stupac predstavlja i **smanjenje pretražnog prostora s obzirom na broj riječi/oblika** i u prosjeku daje učinak oko **3.2 puta**. Umjesto da se pretražuje po katalogu riječi od oko 290.000 riječi (oblici riječi - tablica Word), nakon lematizacije pretraživat će se po katalogu riječi od 90.000 riječi, što je u prosjeku 3.2 puta manje.

Ako u tu računicu uvedemo i učestalost, dobivamo tablicu:

vrsta riječi	***učestalost***	***broj oblika***	***prosj.učest. oblika***	***broj riječi***	***prosj.učest. riječi***	***omjer***
ADJ	1.215.369	80.602	15.1 %	22.433	54.2	3.6 %
N	2.273.035	97.042	23.4 %	37.430	60.7	2.6 %
V	1.503.150	106.319	14.1 %	29.208	51.5	3.6 %
UKUPNO	4.991.554	283.963	17.6 %	89.071	56.0	3.2 %

Ova tablica pokazuje u stupcu *prosj.učest.oblika* koliku učestalost u prosjeku je jedna riječ (oblik) imala prije lematizacije (oko 17). Nakon lematizacije broj riječi je smanjen (stupac *prosj.učest.riječi*), a iz stupca *prosj.učest.riječi* se vidi da je smanjeni broj riječi povećao prosječnu učestalost na 56. Kao krajnji rezultat ponovno dolazimo do istog kvocijenta 3.2 - što je i logično s obzirom na to da su u oba omjera količnici isti.

Zadnja tablica u ovom poglavlju daje pregled prepoznatih riječi po broju oblika koji se barem jednom pojavljuju u korpusu - stupac *broj obl.s učest.*

broj obl.s učest.	***broj riječi***	***udio***	***učestalost***	***udio***
=1	38.844	43.6 %	147.857	3.0 %
=2	15.414	17.3 %	196.529	3.9 %
=3	8.861	9.9 %	208.495	4.2 %
=4	6.413	7.2 %	299.741	6.0 %
=5	4.673	5.2 %	439.867	8.8 %
<=10	10.844	12.2 %	2.066.532	41.4 %
<=15	2.830	3.2 %	810.283	16.2 %
<=20	790	0.9 %	352.945	7.1 %
<=25	278	0.3 %	233.613	4.7 %
>25	124	0.1 %	235.692	4.7 %
UKUPNO	89.071		4.991.554	

Za riječi koji imaju u prvom stupcu *vrijednost =1* lematizacija nema velikog učinka, odnosno prema ciljevima iz prethodnog odlomka (*cilj normalizacije za potrebe pretraživanja*) za cilj 1) nema učinka, no za cilj 2) učinak je postignut. Takve su riječi prepoznate po tvorbeno-infklesijskom nastavku, no nije nađen niti jedan drugi oblik te riječi u korpusu. Kod takvih je riječi obično veći udio *pogrešnog* prepoznavanja (pojedinosti u sljedećem odlomku o općim pravilima). Te riječi imaju vrlo visok udio u broju objekata riječi (43%), ali učestalost im je vrlo niska (3%). Možemo zaključiti da su to riječi koje se vrlo rijetko pojavljuju, te stoga nemaju veliki broj oblika po kojima bi mogle biti prepoznate.

Riječi s dva oblika vjerojatno pripadaju istoj grupu riječi. Kako idemo dalje po tablici moramo primijetiti da raste udio učestalosti, a smanjuje se broj riječi. To se može protumačiti na način da više prepoznatih oblika jedne riječi kumulira svoju učestalost u svoj osnovni oblik - što je i traženi **učinak normalizacije**.

U ovome dijelu bi se navela neka opća pravila do kojih se došlo prilikom razvoja ovog programskog rješenja, testiranja i upotrebe nad skupljenim korpusom.

Prvo opće pravilo:

Što je više oblika jedne riječi u katalogu riječi, to je manja vjerojatnost pogreške

prepoznavanja tipa i ostalih karakteristika osnovne riječi.

Za određeni niz oblika riječi obično se dobije više mogućih kandidata zbog podudarnosti presjeka njihovih flektivnih pravila za te oblike - premda su flektivna pravila međusobno različita. Što se povećava broj oblika, to se smanjuje broj flektivnih pravila koji imaju presjeke po tim oblicima. Iz toga slijedi da što je više oblika, to će proces biti točniji.

Za prethodnu bi tablicu to značilo da je veća točnost prepoznavanja kako se ide prema dnu tablice.

Drugo opće pravilo:

> Što promjenjiva riječ ima veću učestalost, tako i raste broj njezinih oblika u tom istom korpusu.

Laički rečeno, ako se neka promjenjiva riječ upotrebljava u tekstu, vrlo je vjerojatno da će se upotrijebiti u većem broju svojih oblika.

Iz ovoga se može izvesti treće opće pravilo:

> Ako riječ ima veliku učestalost i ako u procesu a nije prepoznata ili je prepoznata s malim brojem oblika koji se pojavljuju, onda je vjerojatno nepromjenjiva i moguće je da pripada u *skup zaustavnih rije*či.

Proces normalizacije uklanjanjem riječi iz *skupa zaustavnih riječi* smanjio je pretražni prostor za oko 40%, a svođenje riječi na osnovni oblik na preostali skup djelovalo je smanjenjem od oko 3 puta na taj dio skupa. Zajednički učinak smanjenja je *40% + 50% * 66% ~ 60%*, odnosno oko 40% prostora je ostalo za pretraživanje. U formuli 40% je količina pojavnica smanjena zbog *skupa zaustavnih riječi*, 50% je prepoznatih riječi - a zbog lematizacije je izbačeno 2/3 takvih pojavnica, tj. 66%. S obzirom da je ovo prva faza projekta, to je zadovoljavajući rezultat.

7 Pretraživanje po sličnosti

U ovome se poglavlju daje pregled dijela programskog rješenja kojemu je cilj prema traženom tekstu pronaći najsličniji u skupljenom korpusu.

7.1 Pregled postojećih sustava i rješenja za pretraživanje teksta

Pretraživanja tekstova danas je vrlo raširena potreba, te stoga postoji veliki broj gotovih programskih rješenja koja to kvalitetno obavljaju. Mnoga od tih rješenja su za slobodnu upotrebu - bilo s licencom za besplatno korištenje, ili često s licencom na izvorni kod (*Open source* [48]). Primjeri OSS rješenja:

- Lucene - *http://lucene.apache.org/*
- Xapian - *http://xapian.org/*
- Sphinx - *http://sphinxsearch.com/*
- Solr - *http://lucene.apache.org/solr/* - internetska aplikacija koja se temelji se na Lucene rješenju
- Haystack - *http://haystacksearch.org/* - *Django* internetska aplikacija - sučelje prema Solr, Xapian ili Whoosh pretraživačkim sustavima
- itd.

Veliki broj poznatih i raširenih sustava za upravljanje bazama podataka (*DBMS*), imaju podršku za pretraživanjem teksta posebnom vrstom indeksiranja teksta (**FTS** - *full-text-search* [49]. Takva vrsta indeksiranja omogućuje da se tekstovi spremaju po retcima tablice, u polja posebnog tipa ili običnog dugog tekstovnog tipa (npr. VARCHAR, CLOBitd.), i nad tim se poljima kreiraju indeksi posebne vrste koji omogućuju vrlo brzo pretraživanje po vrijednostima tih polja. FTS proširenja dodatno omogućuju posebne operacije pretraživanja po tekstovima, obično proširenjem standardnog skupa SQL naredbi. Navest će se poveznice za FTS podršku za neke od poznatih komercijalnih i OSS DBMS sustava:

- Oracle - FTS podrška *http://www.oracle.com/technology/products/text/index.html*, uključeno u

48 *Open source software* (OSS) (*http://www.opensource.org/*) - ili u slobodnom prijevodu *Programi otvorenog koda* (*http://en.wikipedia.org/wiki/Open_source*), programi su čije je korištenje *slobodno*, a programski kôd je također dostupan.

49 *http://en.wikipedia.org/wiki/Full_text_search*

besplatnu inačicu (Oracle XE)

- IBM DB2 - FTS podrška *https://publib.boulder.ibm.com/infocenter/db2luw/* v9r5/topic/com.ibm.db2.luw.admin.ts.doc/doc/c_overview.html, uključeno u besplatnu inačicu (IBM DB2 Express-C)
- Microsoft SQL Server - FTS podrška *http://msdn.microsoft.com/en-us/library/ms142571.aspx*, može biti uključeno u besplatnu inačicu
- Postgresql - OSS DBMS - FTS podrška *http://www.postgresql.org/docs/8.3/static/textsearch.html*
- MySql - OSS DBMS - FTS podrška na *http://dev.mysql.com/doc/refman/5.1/en/fulltext-search.html*
- Sqlite - OSS DBMS - FTS podrška u razvoju *http://www.sqlite.org/cvstrac/wiki?p=FtsOne*
- Firebird - OSS DBSM - FTS nije podržan izravno u DBMS-u, no postoje *vanjski* paketi koji to omogućuju - *http://www.firebirdfaq.org/faq328/*
- itd.

Zanimljivo je istaknuti podršku za FTS indeksiranje nad XML vrstom podataka, jer otvara nove načine razmišljanja i mogućnosti kod pretrage teksta - pogotovo ako se uzme u obzir da se danas najveća količina podataka poslužuje upravo u HTML-u ili XHTML formatu. Kad je tekst obavijen XML-om onda možemo u *tagove* XML-a stavljati metapodatke s informacijama po potrebi, npr. oznaka poglavlja, naslova, odjeljka, struktura unutar rečenice, odabrana riječ u slučaju hiponima, težine, i sl. Za potrebe ovog projekta to bi vjerojatno bilo optimalno rješenje s obzirom na to da je sustav osmišljen tako da se prilikom opojavničenja (engl. *tokenization*) dobivaju metapodatci koji se snimaju u bazu, a naknadno će biti korisni za potrebe daljnjih analiza.

Primjeri DBMS sustava koji imaju mogućnost FTS indeksiranja XML vrste podataka su IBM DB2 [50] i MS Sql Serveru [51].

50 *http://www.ibm.com/developerworks/data/library/techarticle/dm-0606seubert/index.html*

51 *http://msdn.microsoft.com/en-us/library/ms142571.aspx*

7.2 Implementacija funkcije pretraživanja teksta

Budući da postoje gotovi FTS sustavi koji su stabilni, raširenі, kvalitetni i skoro u pravilu slobodni za upotrebu, cilj ovog programskog rješenja nije bio razviti novi, vlastiti sustav za FTS. Za potrebe provjera, analize i ispitivanja razvijenog sustava za normalizaciju teksta na hrvatskom jeziku, razvijen je mali sustav za pretraživanje koji ispunjava te ciljeve. Drugi razlog zašto je implementiran takav sustav je izrada prototipa sustava za pretraživanje po sličnosti teksta (engl. *text similarity*), umjesto uobičajenijeg *web search query* načina (*http://en.wikipedia.org/wiki/Web_search_query*).

Ovaj projekt primarno cilja na sljedeće:

- dobivanje *skupa zaustavnih riječi* za hrvatski jezik,
- lematizacija teksta na hrvatskom jeziku, i
- opojavničenje na odjeljke, rečenice i riječi.

Za sve te ciljeve nije nađeno zadovoljavajuće rješenje u postojećim FTS ili sličnim sustavima.

U nekoj od sljedećih faza razvoja ovoga projekta ovaj će se sustav unaprijediti na način da se odabere jedno od slobodnih FTS/DBMS-FTS sustava, te izvrši integracija između ovog rješenja i FTS sustava.

Cilj ovog prototipa bio je da se za zadani tekst u određenom tekstu poredaju rečenice od najsličnije do najmanje slične.

Zadaci su sljedeći:

1. potrebno je postojeći opojavničeni tekst pretvoriti u objekte rečenice
2. potrebno je primijeniti filter *skupa zaustavnih riječi* i primjeniti lematizaciju
3. potrebno je primijeniti algoritme za sličnost teksta na sve rečenice
4. potrebno je napraviti grafičko sučelje u kojem je to moguće koristiti.

Ovaj će pregled početi od strane korisnika, tj. korisnik unosi traženi tekst u polje za pretraživanje i onda se poziva funkcija (skraćeni kod):

```
def doc_content_search(request):
    # ... prepare data
    # ... fetch text from database
    docword_list = DocWord.objects.first_filter(doc).filter(
                    location__gte=page.location_from,
                    location__lte=page.location_to
                    ).select_related("word", "word__wordobj_main")

    # ... convert tokens to list of sentences
    sentence_tuple_list = docword_list.get_sentence_list()
    sentence_list = SentenceList(sentence_tuple_list)

    # ... fetch searched text and convert it to one Sentence object
    text = form.cleaned_data["text"]
    for location, word in enumerate(splitter.split(text)):
        word_item = list(Word.objects.filter(value=word))[0]
        wtoken = WordToken(dw=None, word=word_item, word_value=word)
        token_list.append(wtoken)
    sentence_search = Sentence(sentence_tuple=None,
                               token_list_to_search=token_list)

    # do search
    sentence_list.search(sentence_search)

    # ... display result by rendering it to HTML page
    return render_to_response("tran/doc_content_search.html",
                              {"doc" : doc, "page" : page,
                               "form" : form,
                               "sentence_search" : sentence_search,
                               "text" : text,
                               "sentence_list" : sentence_list })
```

Ovdje je potrebno primijetiti DocWord klasu preko koje se dohvaća tekst po kojem se pretražuje, a vlastitom iterator funkcijom *get_sentence_list()* taj se tekst dijeli u rečenice korištenjem posebnih pojavnica (engl. *token*) početka i kraja dobivenih u procesu opojavničenja. Rečenice i riječi unutar tog teksta pretvaraju se u *WordToken*, *Sentence* i *SentenceList* tim redoslijedom, gdje *SentenceList* predstavlja pripremljeni tekst. Na sličan način kao što se pripremio tekst po kojem se traži, tako se priprema i tekst koji se traži, a rezultira u objektu *sentence_search* tipa *Sentence*. Tada se poziva metoda *sentence_list.search(sentence_search)* koja obavlja traženje i omogućuje željeni poredak i sve podatke nastale pretraživanjem.

Klase *SentenceList*, *Sentence* i *WordToken* opisane su u kodu koji slijedi:

```
class WordToken(object):
    def __init__(self, dw, word=None, word_value=None):
        self.dw = dw
        self.word = self.dw.word
        self.value = self.word.value

class Sentence(object):
    def __init__(self, sentence_tuple=None,
                 token_list_to_search=None):
        # ... prepare data
        ...
        self.word_dict = {}
        self.word_main_dict = {}
        for wt in self.token_list:
            # ... skip stopwords
            if wt.word.wordobj_main_id and
               wt.word.wordobj_main.is_stopword():
                continue

            # ... calculate tf-idf factor for wordobj_main
            #     (lemmatized) and word
            tf_idf_word = wt.dw.get_idf(main=False)
            tf_idf_main = wt.dw.get_idf(main=True)
            self.word_dict[wt.word.value.lower()]=tf_idf_word

            # ... if no wordobj_main then use word
            if wt.word.wordobj_main_id:
                self.word_main_dict[wt.word_main_value]=tf_idf_main
            else:
                self.word_main_dict[wt.word_main_value]=tf_idf_word
        self.word_main_set = set(self.word_main_dict.keys())
        self.word_set = set(self.word_dict.keys())

    def match(self, sentence_search):
        ... later ...

class SentenceList(object):
    def __init__(self, sentence_tuple_list):
        self.sentence_list = []
        for sentence_tuple in sentence_tuple_list:
            sentence_obj = Sentence(sentence_tuple)
            self.sentence_list.append(sentence_obj)

    def search(self, sentence_search):
        for sentence in sentence_list:
            sentence.match(sentence_search)
```

Objekti klase *WordToken* predstavljaju riječi, objekti klase *Sentence* rečenice, a

objekti klase *SentenceList* tekst po kojemu se traži. Već se u objektu klase *Sentence* primjenjuje lematizacija korištenjem *wordobj_main* vezom objekata riječi *Word*. Tu se odmah i izračunava TF-IDF faktor [52], koji je uobičajeni težinski faktor koji se koristi za potrebe pretraživanja tekstova. U ovom radu tf-idf je izračunat na sljedeći način

```
tf = word_freq / all_words_freq
idf = math.log(float(1 + doc_count) / (1 + word_doc_count))
tf_idf = tf * idf
```

gdje su veličine:

- *word_freq* - učestalost "riječi" u dokumentu,
- *all_words_freq* - suma učestalosti svih riječi u dokumentu,
- *doc_count* - broj svih dokumenata,
- *word_doc_count* - broj dokumenata gdje se pojavljuje "riječ".

Veličina tf-idf daje težinski faktor koji nosi informaciju važnosti riječi za neki dokument [53].

Sljedeći je zadatak napraviti izračun faktora koji pokazuje koliko je određeni tekst sličan nekom drugom. Za te je potrebe korištena formula za *mjeru sličnosti po kosinusu* (*cosine similarity*, slika preuzeta sa [54]):

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$$

koja daje sličnost dva vektora A i B u n-dimenzionalnom prostoru na način da se izračuna kosinus kuta koji ta dva vektora međusobno čine. U našem slučaju dimenzionalnost vektora broj je svih riječi iz kataloga, a vrijednosti vektora A i B su težinski faktori riječi. Brojnik je skalarni produkt dva vektora, a nazivnik umožak duljina svakoga od vektora.

Sljedeće pitanje koje se postavlja je pitanje vrijednosti koordinata vektora. Uobičajeno je da se uzimaju težinski faktor *tf-idf*, no moguće je uzimati i binarne faktore - vrijednost 0 - riječ se ne pojavljuje, vrijednost 1 - riječ se pojavljuje u

52 *http://en.wikipedia.org/wiki/Tf%E2%80%93idf*

53 Kod pretraživanja po rečenicama, dokument je podijeljen po rečenicama, no svejedno je mjera *word_freq* računata s obzirom na cijeli dokument, a ne samo na rečenicu.

54 *http://en.wikipedia.org/wiki/Cosine_similarity*

tekstu. U tom se slučaju uzima druga formula, odnosno formula za sličnost po kosinusu prilagođena s *Jaccardovim koeficijentom*, te se dobiva *Tanimotova mjera sličnosti* (slika preuzeta sa [55]):

$$T(A,B) = \frac{A \cdot B}{\|A\|^2 + \|B\|^2 - A \cdot B}.$$

Obje su mjere implementirane radi usporedbe i isprobavanja učinkovitosti. Prednost Tanimotove mjere je što ne treba izračun tf-idf, pa je jednostavnija za implementaciju i izvršavanje, a time i brža. No kosinusna mjera sličnosti, premda nešto složenija i sporija, ipak daje nešto kvalitetnije rezultate.

Slijedi pregled sažetog izvornog koda koji izračunava međusobnu sličnost dviju rečenica (klasa *Sentence*):

```
def match(self, sentence_search):
    int_main = self.word_main_set.intersection(
                    sentence_search.word_main_set)
    int_word = self.word_set.intersection(
                    sentence_search.word_set)
    div1 = (len(self.word_main_set)**2
            + len(sentence_search.word_main_set)**2
            - len(int_main))
    self.match_main_tanimoto = (float(len(int_main)) /
                                div1)
    div2 =(len(self.word_set)**2
           + len(sentence_search.word_set)**2
           - len(int_word))
    self.match_word_tanimoto = (float(len(int_word)) /
                                div2)

    # TF IDF for sentence_search is based on document where we search
    div3 = (len(self.word_main_set)**2
            + len(sentence_search.word_main_set)**2)
    fact3 = sum([float(self.word_main_dict[word]**2)
                 for word in int_main])
    self.match_main_cosine = ( fact3 / div3 )

    div4 =(len(self.word_set)**2
           + len(sentence_search.word_set)**2)
    fact4 = sum([float(self.word_dict[word]**2)
                 for word in int_word])
    self.match_word_cosine = ( fact4 / div4 )
```

55 *Manning & Schütze, Foundations of Statistical Natural Language Processing, MIT Press. 1999., http://en.wikipedia.org/wiki/Jaccard_index*

```
int_main = self.word_main_set.intersection(
                sentence_search.word_main_set)
int_word = self.word_set.intersection(
                sentence_search.word_set)
```

Da bi se uočio učinak traženja osnove riječi izračunate su 4 mjere:

- match_word_tanimoto - tanimoto mjera sličnosti - samo oblici -> bez lematizacije
- match_word_cosine - kosinusna mjera sličnosti - samo oblici -> bez lematizacije
- match_main_tanimoto - tanimoto mjera sličnosti - osnovne riječi -> s lematizacijom
- match_main_cosine - kosinusna mjera sličnosti - osnovne riječi -> s lematizacijom

Za kosinusnu mjeru sličnosti korišteni su *tf-idf* težinski faktori koji su već ranije spremljeni u varijable *word_dict* za oblike i *word_main_dict* za osnovne riječi (lema).

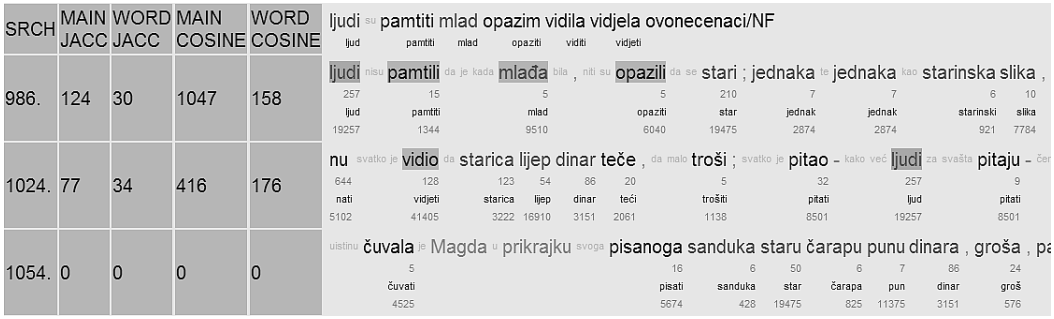
Za demonstracijske potrebe uzet je primjer iz teksta književnog djela "Zlatarevo zlato", Augusta Šenoe, od otprilike 1000-te do 2000-te pojavnice:

986.	**ljudi** nisu **pamtili** da je kada **mlađa** bila **,** niti su **opazili** da se **stari ; jednaka** te **jednaka** kao **starinska slika ,** koja **, viseći godine** i **godine** u **zabitnoj kuli ,** svoga **lika** ne **mijenja .**
1024.	**nu** svatko je **vidio** da **starica lijep dinar teče ,** da malo **troši ;** svatko je **pitao -** kako već **ljudi** za svašta **pitaju -** čemu **Magdi novaca ?**
1054.	uistinu **čuvala** je **Magda** u **prikrajku** svoga **pisanoga sanduka staru čarapu punu dinara , groša , pače** i **starih cekina** od **kralja Matijaša .**
1078.	**čarapa** se **debljala , Magda** se **sušila .**
1087.	ali čemu **Magdi novaca ?**

Svaki redak predstavlja jednu rečenicu. U prvom se retku nalazi oznaka položaja prve pojavnice u rečenici, odnosno njegova apsolutna pozicija u tekstu (986. pojavnica od početka teksta).

Traženi tekst je *"ljudi su pamtiti mlad opazim vidila vidjela ovonecenaci"*, što je dosta slično prvoj rečenici, no ima riječi koje se neće naći u prvoj rečenici.

Rezultat je prikazan na sljedećoj slici:



SRCH	MAIN JACC	WORD JACC	MAIN COSINE	WORD COSINE	
					ljudi su pamtiti mlad opazim vidila vidjela ovonecenaci/NF ljud pamtiti mlad opaziti viditi vidjeti
986.	124	30	1047	158	ljudi nisu pamtili da je kada mlađa bila , niti su opazili da se stari ; jednaka te jednaka kao starinska slika , 257 15 5 5 210 7 7 6 10 ljud pamtiti mlad opaziti star jednak jednak starinski slika 19257 1344 9510 6040 19475 2874 2874 921 7784
1024.	77	34	416	176	nu svatko je vidio da starica lijep dinar teče , da malo troši ; svatko je pitao – kako već ljudi za svašta pitaju – čer 644 128 123 54 86 20 5 32 257 9 nati vidjeti starica lijep dinar teći trošiti pitati ljud pitati 5102 41405 3222 16910 3151 2061 1138 8501 19257 8501
1054.	0	0	0	0	uistinu čuvala je Magda u prikrajku svoga pisanoga sanduka staru čarapu punu dinara , groša , pa 5 16 6 50 6 7 86 24 čuvati pisati sanduka star čarapa pun dinar groš 4525 5674 428 19475 825 11375 3151 576

U prvom se retku nalazi pripremljeni tekst koji se pretražuje. Sljedeći retci sadrže tekst koji je pretražen, podijeljen po rečenicama, s time da je redoslijed od najsličnijeg prema manje sličnima.

Za jedan redak malim su slovima prikazane riječi iz *skupa zaustavnih riječi*. Za ostale se riječi ispod njih nalaze redom:

- tf-idf mjera za oblik riječi
- osnovni oblik riječi
- tf-idf mjera za osnovni oblik riječi.

Prva četiri stupca sadrže sljedeće vrijednosti:

- pozicija rečenice - položaj prve pojavnice rečenice u cijelom tekstu dokumenta
- MAIN JACC - tanimoto mjera sličnosti za osnovne riječi (*match_main_tanimoto* atribut)
- WORD JACC - tanimoto mjera sličnosti za oblike riječi (*match_word_tanimoto* atribut)
- MAIN COSINE - kosinusna mjera sličnosti za osnovne riječi (*match_main_cosine* atribut)
- WORD COSINE - kosinusna mjera sličnosti za oblike riječi (*match_word_cosine* atribut).

S obzirom na to da tf-idf mjere nisu normalizirane mjere već se trebaju koristiti samo u međusobnoj usporedbi u tekstu, onda ih se ne smije promatrati okomito, već prvo normalizirati vodoravno, pa onda okomito.

Primjer

Za riječ "pamtili" mjera je 15, a od njene osnovne riječi "pamtiti" mjera je 1344. Te se dvije mjere ne smiju uspoređivati izravno, već ih se treba normalizirati u retku, npr. zbroj svih tf-idf vrijednosti retka i podjela svih s tim zbrojem, pa je tek onda moguća usporedba.

U istom svjetlu trebalo bi promatrati i uspoređivati kosinusnu mjeru sličnosti i tanimoto mjeru sličnosti (stupci *Jacc*), s time da normalizacija treba ići okomito, a kasnija usporedba vodoravno.

Iz ovog bismo primjera mogli izvući zaključke:

- izbacivanje riječi iz *skupa zaustavnih riječi* vrlo je koristan filter - u ovom je primjeru takvih riječi dosta, a ne donose zamjetnu sadržajnu vrijednost;
- samo prva i druga rečenica imaju zamjetnu sličnost s traženim tekstom;
- bez lematizacije prva i druga rečenica bile bi proglašene "slabo sličnima" traženom tekstu s vrlo malim brojem riječi koje su pronađene (ljudi) - stupci *WORD JACC* i *WORD COSINE;*
- primjenom lematizacije faktori prilično rastu jer se broj riječi koje su pronađene dosta povećava (ljudi [56], pamtili, mlad, opaziti, vidjeti). Za primijetiti je da recimo tražena riječ *vidjela* prepoznaje u obliku *vidio* njenog osnovne riječi *vidjeti*, što je posljedica lematizacije traženog i teksta koji se traži. Konačan rezultat je da je prva rečenica znatno sličnija od druge - stupci *MAIN JACC* i *MAIN COSINE;*
- iz usporedbe tanimoto i kosinusne mjere sličnosti proizlazi da kosinusna mjera sličnosti daje u ovom slučaju bolji rezultat - odnos je 2.5 (1024 / 416) prema 1.7 (124 / 74);
- za primijetiti su dvije riječi u traženom tekstu koje nisu pronađene te ne pridonose učinku - prva je *vidila* - što je oblik pravopisno nevaljane riječi *viditi*, te riječi *ovonecenaci* koja je namjerno stavljena da se vidi da sustav takve riječi ignorira.

Iz svega se toga može zaključiti da je proces normalizacije na ovom prototipu dao zamjetne rezultate, te je pretpostavka da će se u daljnjem razvoju ovog projekta to iznova dokazivati.

56 Riječ ljudi je množina riječi čovjek i iznimka je u hrvatskom jeziku. Ta iznimka trenutno nije registrirana u programskom rješenju.

8 Prijedlog nastavka rada - strojno razumijevanje

Uobičajeni scenarij u procesu pretraživanja je:

- Korisnik sustava (*end user*) postavlja sustavu upit za pretraživanje po određenom pojmu, skupu pojmova
- Sustav analizira upit, traži optimalan put do rezultata, dohvaća rezultate, rangira rezultate i prikazuje ih

Napravit će se kratka analiza:

Pitanje	***Odgovor***
Tko postavlja upit?	Korisnik sustava, odnosno čovjek
Koji je oblik je upita?	Najčešće u obliku najvažnijih pojmova, redoslijed najčešće nije važan.
Zašto je takav oblik upita?	Jer sustav za takve oblike upita daje najbolje rezultate.

Postavlja se sljedeće pitanje: Što bi, s korisničke strane gledajući, bilo optimalno rješenje?

8.1 Pretraživanje - optimalno rješenje

Jedna od osnovnih kvaliteta software-a koja se želi postići je da se sustav približi korisniku (*user-friendly*). Kad ga koristi, korisnik treba sve manje razmišljati o samom sustavu, a sustav mora biti dovoljno inteligentan i prilagodljiv kako bi se mogao sam pobrinuti da pravilno interpretira sve korisničke zahtjeve. U krajnjem slučaju, sustav je samo alat i treba biti *podložan* korisniku.

Način na koji se obično postavljaju upiti nad sustavom za pretraživanje tekstova, je vrlo napredan u odnosu na to kakav je bio u prošlosti. S druge strane, postoji još puno prostora za unaprjeđenje. Činjenica da korisnik mora razmišljati koje riječi koristiti kad postavlja upit za pretraživanjem indikator je da sustav nije dovoljno *podložan*. Jedan od ciljeva trebao bi biti da kad propitkujemo sustav da ga pitamo na isti način kao što bi sebi ili drugom čovjeku postavili pitanje - bilo pisanom rječju, bilo govorom ili mišlju.

S gledišta interpretacije upita, sustav treba biti u mogućnosti *razumjeti* čovjekov upit: što korisnik stvarno misli, odnosno što želi doznati upitom?

S gledišta znanja, računalo bi trebalo neprestano *učiti*, odnosno pratiti informacije, *razumijevati* ih i puniti svoju bazu znanja.

Oba problema zadiru u posebno područje računalne znanosti koje se naziva *strojno razumijevanje* (*machine reasoning*). Budući da su čovjek, njegov način razmišljanja, njegov način razumijevanja, njegov način komunikacije (govor, jezik itd.) same po sebi vrlo složene stvari, teško je očekivati da će stroj odnosno računalni program koji obavlja slične stvari biti jednostavan. U nastavku ovog poglavlja napravljeno je kratko idejno rješenje jednoga takvog sustava.

U nastavku će se kroz nekoliko primjera probati u grubim crtama opisati idejno rješenje sustava koji bi se mogao koristiti za potrebe "strojnog razumijevanja".

8.2 Primjer 1

Imamo rečenicu:

"Veliki dječak hoda."

Prije nego krenem u dio samoga razumijevanja, potrebno je napraviti jednostavnu programsku okolinu u kojoj bi se ova rečenica mogla interpretirati:

```
class MoveState(State):
    VALUES = ("stajati", "hodati", "ležati")
    def __init__(self, value):
        assert value in VALUES
        self.value = value

class AttributeVisina(Attribute):
    VALUES = ("veliko", "srednje", "malo")
    def __init__(self, value):
        assert value in VALUES
        self.value = value

class MoveStateHistory(StateHistory):
    def __init__(self):
        self.history = []
        self.current_state = None

    def add(value, time):
       self.current_state = MoveState(value)
       self.history.append([time, self.current_state])
       return self.current_state

class Djecak(Person):
    def __init__(self, visina):
        self.visina = AttributeVisina(visina)
```

```
        self.move_state = MoveState("stajati")
        self.move_state_history = MoveStateHistory()

    def move_state_change(self, value, time=None):
        self.move_state = self.move_state_history.add(value, time)
```

Klasa *Djecak* ima za primjer atribut visina, stanje kretanja koje je inicijalno postavljeno na vrijednost *stajati*. Metodom *move_state_change()* stanje kretanja se može mijenjati na druge vrijednosti. Sve promjene stanja zabilježene su u dnevniku *move_state_history*.

U ovakvoj programskoj okolini, rečenicu "Veliki dječak hoda" računalni bi program mogao "razumjeti", tj. prevesti u sljedeće

```
>>> djecak = Djecak(visina="velik")
>>> djecak.move_state_change("hodati")
>>> print djecak
Djecak(visina='veliko')
>>> print djecak.move_state
MoveState('hodati')
```

Ako dalje piše:

"On je stao."

U tom slučaju računalni algoritam mora znati da se "on" - što je zamjenica, odnosi na dječaka iz prošle rečenice. S druge strane, u dijelu "je stao" imamo pomoćni glagol biti i radni pridjev (aktiv) glagola stajati. To bi se trebalo ovako "razumjeti":

```
>>> djecak.move_state_change('stajati')
>>> djecak.move_state
MoveState('stajati')
>>> print djecak.move_state_history
MoveStateHistory(
    (time=Moment(1), MoveState('hodati' )),
    (time=Moment(2), MoveState('stajati')),
    )
```

U ovom je primjeru programska okolina predefinirana, no postoji način da se programska okolina gradi automatski u procesu računalnog učenja.

Kao što malo dijete dok odrasta promatra, istražuje, uči od starijih i na vlastitom primjeru, tako i računalni program može imati takve algoritme. Računalo treba imati i bazu svega prikupljenog znanja zajedno sa "sjećanjima" odnosno svim uzorcima na osnovi kojih je znanje prikupljeno - bilo "rečeno" od autoriteta - čovjeka, bilo potvrđeno od autoriteta ili vlastitim zaključivanjem.

8.3 Primjer 2

Recimo da imamo sustav zamišljen na sličan način kao iz prethodnog primjera, gdje imamo znanje o većem skupu vrsta objekata, nemamo znanje o objektima klase *djecak*, ali imamo znanje o klasama *ljudi* i *životinje*. U tom slučaju sustav recimo da zna da su samo vrste objekata ljudi i neke od životinja u mogućnosti "hodati". U interpretaciji rečenice:

"Dječak hoda."

sustav prepoznaje "hodati", a budući, da svih klasa, svi ljudi i neke životinje mogu "hodati", sustav bi mogao donijeti sljedeće zaključke:

- "dječak" je ili čovjek ili životinja koja može hodati:
  - "dječak" može biti sinonim (ili *alias*) za klasu *čovjek* ili *životinja*
  - može biti specifičan oblik "čovjeka" ili vrste životinje
- može biti neka potpuno nova vrsta objekta - nevezana na postojeće vrste objekata koja isto kao i *čovjek* ili *životinja* može "hodati"
- itd.

Vođen statistikom, može se odrediti težina svih tih zaključaka, te zaključiti da se recimo od svih objekata za koje je spomenuto da "hodaju", najčešće govori o "čovjeku". Tada je najvjerojatnija pretpostavka da je "dječak" ili podvrsta "čovjeka" ili sinonim za "čovjeka".

Prikupljanjem ostalih podataka o vrsti objekta "dječak", svi doneseni novi zaključci koji nisu sigurni mogu se staviti u sustav logičkog zaključivanja koji metodom kombiniranja i eliminacije izbacuje sve nemoguće scenarije i donosi određeni skup mogućih zaključaka - zajedno s težinama tj. vjerojatnostima.

Dok je sustav vrlo "mlad" nužna je pomoć korisnika sustava koji potvrđuje dobre ili odbacuje krive zaključke, te time omogućuje sustavu da bira pravilne zaključke.

Ovakav sustav podsjeća na sustav koji rješava križaljku: potrebno je naći riječi koje imaju određena slova na određenom mjestu, te moraju biti određene duljine i odgovarati na postavljeno pitanje. Najteže je početi rješavati križaljku, ali što je više kvadratića popunjeno, to je sustavu sve lakše prepoznavati i popunjavati riječi koje nedostaju.

8.4 Primjer 3

Ovaj se primjer odnosi na sličan scenarij, s razlikom da znamo sve o objektu "dječak" osim što nam je nepoznato da "dječak" može hodati. Iz rečenice "dječak hoda" sustav zaključuje da postoji novo stanje kretanja koje je valjano za objekt "dječak", a to je "hodati". To se registrira u sustavu kao novi podatak za vrstu objekta dječak, čime je sustav naučio nešto novo o toj vrsti objekta.

8.5 Srž ideje

Srž ovakve ideje bila bi naučiti računalo da pokušava *naučiti* svijet kakav jest, odnosno da skuplja znanje o određenim stvarnim pojavama - skupljajući informacije i donoseći zaključke. Osnovna ideja je da se nauči računalo *čitati* tekstove, te da ih onda pokuša *razumjeti*, a u kombinaciji sa sustavom za zaključivanje i korisničkom pomoći kreira novo znanje ili potvrđuje postojeće.

Zašto učiti računalo da *čita* i da mu pročitani tekstovi budu osnova za razumijevanje i skupljanje znanja?

Pretpostavka je da je to jedan od jednostavnijih puteva. Napravit će se kratka analiza. Čovjek i ostatak živog svijeta uči na način:

1. ima potrebu učiti - osnova opstanka i urođena znatiželja;
2. uči promatranjem - svim osjetilima, vanjskim (npr. vid, sluh, opip), i unutarnjim (npr. osjećaji);
3. sve osjete i doživljaje skuplja, promatra i uz pomoć ugrađenog kognitivnog sklopa donosi zaključke ili formira dodatna pitanja;
4. sve što nauči se ili potvrđuje sličnim iskustvima, ili nadograđuje novim *preciznijim* osjetima i doživljajima;
5. uz pomoć učitelja, potvrđuje ili popravlja svoje znanje.

U računalni sustav možemo jednostavno ugraditi točku 1) - potrebu učenja (beskonačna petlja *do while true*). Što se tiče točke 2) tu nastaje problem. Što računalni sustav doživljava? Ono što mu damo na ulaz. Gdje može računalo skupiti najviše znanja o svijetu pripremljeno za jednostavnu računalnu obradu? U današnje doba, internet je, između ostalog, postao jedna velika knjižnica, baza znanja - gdje je znanje spremljeno u elektronskom tekstovnom obliku, u obliku članaka, knjiga i

slično. Kao primjer može poslužiti projekt Wikipedia [58] koji vjerojatno sadrži količine znanja veće nego najveće enciklopedije u tiskanom ili elektroničkom obliku.

Napraviti *pametni* računalni sustav bi se mogao na način da se računalo nauči da čita, razumjeva i zaključuje iz takvih elektroničkih knjižnica - slično kao što to čovjek radi.

Implementacijski gledano to bi se moglo razlučiti na ove zadatke:

- indeksaciju tekstova - tj. opojavničenje - riječ, rečenica, odlomak, poglavlje
- prepoznavanje vrste riječi (*Part-of-speach tagging* [41])
- odabir tekstova za učenje - nakon prepoznavanja teme (*automatic text summarization* [59]), ovisno o zadanim pravilima i temama koje računalo ima za naučiti, biraju se tekstovi i redoslijed kojim ih treba učiti. Početne tekstove vjerojatno bi trebao birati ekspert, odnosno korisnik sustava, a nakon toga bi i računalni sustav mogao sam birati i određivati redoslijed.
- kognitivni dio - *čitanje*, slaganje premisa, izvođenjem svih potrebnih logičkih operacija nad premisama i zaključaka iz novoskupljenog znanja. Potrebno je uvesti i statističku obradu, što je vjerojatnije, što manje vjerojatno, te što nije vjerojatno.
- nakon što se kreira novo stanje baze znanja ekspertu je potrebno omogućiti da pomogne računalu pri napretku - korisnik je neka vrsta učitelja. To bi mogli usporediti s dječjim dobrim i *pogrešnim* zaključivanjem koje uvijek ima nekakvu logiku iza sebe, no roditelj ili učitelj ga navodi na pravi put i otkriva istinu.

Kao rezultat imali bismo računalu razumljivu bazu znanja, a onda bi računalo bilo sposobno posluživati takvo znanje.

U poglavlju *Pretraživanje - optimalno rješenje* predstavljena je problematika pretraživanja. Ovakav bi sustav bio put k optimalnom rješenju tog problema. Ovakav bi sustav mogao odgovarati na pitanja kao što su:

Gdje je Zagreb? [60]

58 *http://www.wikipedia.org*

59 *http://en.wikipedia.org/wiki/Automatic_summarization*

60 Danas postoje javni sustavi koji imaju karakteristike slične opisanom sustavu, primjer *http://www.wolframalpha.com/* ili nešto jednostavnija vrsta su "natural language" sustavi za traženje opisani (*http://en.wikipedia.org/wiki/Natural_language_search_engine*), npr. *http://www.ask.com/* .

Sustav bi obavio analizu upitne rečenice, mogao bi prepoznati da je to pitanje vjerojatno o geografskoj poziciji, te da se odnosi na grad Zagreb - što je prepoznato na način što je *Zagreb* vlastita imenica i nema sinonima.

Ovo je samo jedna od mogućih primjena ovakvih *pametnih* sustava.

8.6 Zaustavne riječi su važne

Pomoćne riječi (biti, htjeti), veznici, prijedlozi, itd. su za potrebe pretraživanja (poglavlje *Zaustavne riječi (engl. stopwords)*) odabrane u *skup zaustavnih riječi* i time izbačene iz analize (klasificirane kao neka vrsta "šuma"). U ovoj vrsti analize, one nose ključnu ulogu - jer su vezivni faktor u strukturi rečenice. Zamjenice su poseban problem, jer laički rečeno "glume" imenice i ostale riječi iz nekog prethodno izrečenog konteksta, što ih čini dosta važnim. U ovakvim vrstama obrade teksta, puno je teže izolirati nevažne riječi, jer je obično svaka riječ zbog nekog razloga izrečena i zbog nekog razloga stoji baš na određenom mjestu.

8.7 Složenost problematike računalnog razumijevanja ljudske misli

Prikazani sustav vrlo je primitivna ilustracija računalnog razumijevanja i učenja iz pisanog teksta. Problematika koja stoji iza ovoga znatno je složenija.

U usporedbi s današnjim najnaprednijim računalnim kognitivnim sustavom, ljudi imaju kognitivni sustav koji je puno složeniji i napredniji. No i samim je ljudima u nekim slučajevima teško prodrijeti u misao izrečenu govorom ili pisanom riječi (dolazi do ne-sporazum-a). Čovjek se pri tome služi svim mogućim pomoćnim sredstvima i alatima - protupitanja, promatranje izraza lica, pokreta tijela, prepoznavanje uzorka načina govora, prepoznavanje uzorka ponašanja, naglasak, ton, itd. Problematika razumijevanja ne završava tu, već imamo dodatne teme za rješavanje: dvosmislenost, preneseno značenje, frazemi, pjesništvo, igra riječi, izreke, terminologija, hiponimi, sinonimi, itd. Za čovjeka je to *prirodno*, te sam ne vidi veliki problem u tome, no kad se radi o pretvaranju logike razumijevanja u računalni program/sustav, tek se onda vidi koliko je to složena problematika.

U usporedbi s ovim *ekspertom* za razumijevanje ljudskog govora i riječi, današnji računalni sustavi još su uvijek vrlo primitivni, što ne znači da se s vremenom neće poboljšati i u neko dogledno vrijeme doći na razinu usporedivu s razinom koju ljudi posjeduju.

Premda je računalo, u ovom smislu razumijevanja, još uvijek primitivno, ono ima neke prednosti nad čovjekom. Primjerice:

- brzina obrade informacija
- količina informacija koja se može spremiti
- međukomunikacijske sposobnosti - savršeno međurazumijevanje, svedostupnost i brzina razmjena informacija među računalnim sustavima
- stroga i precizna organizacija informacija
- vrlo velika preciznost izvođenja operacija i vrlo mala tolerantnost na greške
- stabilnost - npr. ne radi greške zbog umora, može raditi 24h dnevno
- itd.

Gledajući ovako, jednoga dana kad računalo bude imalo kognitivne sposobnosti usporedive s čovjekom, bazu znanja koja će očigledno biti puno veća od baze znanja najučenijih i najpametnijih ljudi, te uz prethodno navedene prednosti računalnog sustava nad čovjekom (a koje će u to vrijeme biti još veće), postavlja se pitanje:

> Hoće li čovjek izgubiti bitku na tom području, a računalni sustavi pobijediti? Koja će onda biti uloga čovjeka na tom području?

Pretpostavka je da će se računala koristiti za neke kognitivne potrebe za koje se danas *služimo* čovjekom, no s druge strane računalo nikad neće moći simulirati i zamijeniti ljudsku kreativnost koja je izvor i srž napretka ljudskog društva.

9 Zaključak

U početku je ovaj projekt bio zamišljen kao jedan jednostavan sustav za pretraživanje. No već nakon početnih analiza i proučavanja postojećih sustava osnovni je problem predstavljao nedostatak sustava za normalizaciju teksta na hrvatskom jeziku.

Hrvatski je jezik morfološki bogat jezik što je veliki problem za računalnu obradu tekstova. Napraviti sustav za lematizaciju i dobiti *skup zaustavnih rije*či je jedna vrsta izazova, koji je nakon testiranja i promišljanja, ipak prihvaćen. Način na koji je osmišljena lematizacija i sastavljanje *skupa zaustavnih riječi* predstavlja nešto dublji način promišljanja i traženja rješenja za ovu problematiku.

U ovaj je sustav uložen veliki trud, no taj trud se isplatio i ciljevi su ostvareni. Ovo je samo prva faza u razvoju ovog projekta, a projekt će se dijelom objaviti kao projekt za slobodno korištenje sa slobodnim pristupom do izvornog koda (*Open source*), zajedno s dijelom rezultata do kojih se došlo (npr. *skup zaustavnih rije*či).

Pretpostavka je da koliko god "morfološko bogatstvo" hrvatskog jezika otežava računalnu obradu, s druge pak strane informacije koje se dobiju iz samog teksta procesom lematizacije/*Part-of-speach tagging*, mogu postati vrlo korisne za neke druge potrebe računalne obrade prirodnog jezika (*natural language processing* - NLP).

Primjer

Sam oblik riječi svojim sufiksom daje dodatne meta podatke (rod, spol, padež, itd.) koji opisuje odnose, pripadnost, objekte i predikate, vršitelje radnje, vrijeme radnje itd. Takve se informacije ne bi mogle dobiti na takav način u nekom "jednostavnijem" jeziku.

Ciljevi i teme za sljedeće faze razvoja ovog projekta su:

- smanjenje broja neprepoznatih oblika riječi
- optimiranje izvornog koda koji se odnosi na vrste riječi
- implementacija vrste riječi *brojevi*
- dokumentacija i upute
- unaprjeđenje sustava za prepoznavanje osnovnih riječi, uključiti ostale faktore - statistiku, plodnost sufiksa, uloga i položaj u rečenici, korištenje

informacija iz tvorbenog sufiksa (zakon -> zakonski)

- definicija novog dijela modela - *korijenska* riječ koja može imati mogućnost unošenja novih semantičkih odnosa među riječima - za početak korištenjem tvorbenih sufiksa, npr. zakon i zakonski, pisati i pisac, itd.
- korištenje drugih izvora *kataloga riječi* - npr. *google* upit
- integracija drugih postojećih NLP alata s ovim programskim rješenjem - npr. NLTK [57]
- integracija s postojećim FTS rješenjima - indeksiranje i pretraživanje teksta - npr. SOLR, Haystack, DB2 XML Text search
- druge tehnike dubinske analize teksta - npr. klasifikacija teksta
- analiza strukture rečenice
- itd.

Ako se projekt bude odvijao prema planu, u jednoj daljnjoj fazi predviđa se i zalazak u područje strojnog razumijevanja. Pretpostavka je da će to područje postajati sve popularnije i sve dostupnije krajnjem korisniku.

57 *http://www.nltk.org/*

10 Dodatci

10.1 Statistika razvoja projekta

Razvoj ovog programskog rješenja započeo je u veljači 2009. godine, a prvo kodiranje bilo je 10.3.2009. Sljedeća tablica daje pregled važnijih događaja po datumu kao jedna vrsta skraćenog dnevnika:

datum	***kratki opis događaja***
10.3.2009	završene sve početne analize, traženje gotovih rješenja, dogovori i sve je spremno za kodiranje
11.3.2009	prve linije programskog koda, pretvorba html -> txt
27.3.2009	prvi program za povlačenje sadržaja s interneta: www.nn.hr
30.3.2009	povučen sav planirani sadržaj s interneta
31.3.2009	izrada modela baze podataka
13.4.2009	počela izrada sustava za izradu morfološkog leksikona za hrvatski jezik - registracija glasova
18.4.2009	unesene sve nepromjenjive vrste riječi i njihov popis za skup zaustavnih riječi. Započet sustav za infleksiju - promjenjive vrste riječi
12.5.2009	implementirane sve vrste zamjenica, svi *stopwords* glagoli i ubačeni u bazu
13.5.2009	sustav za opojavničenje- traženje postojećeg rješenja, isprobavanje i sl.
15.5.2009	započeta izrada vlastitog sustava za opojavničenje
22.5.2009	prva verzija sustava za opojavničenje gotova, pripreme za ubacivanje korpusa u bazu
08.6.2009	prvi tekstovi u bazu, *refactoring* sustava za opojavničenje po potrebi
13.7.2009	gui sučelje za tekstove i riječi - web aplikacija
06.8.2009	korpus je opojavničen i u bazi, gui sučelje gotovo
07.8.2009	započet rad nad sustavom za prepoznavanje osnovnih oblika - obrnuta infleksija
01.9.2009	pauza u razvoju od 3 tjedna
12.10.2009	svi WTS objekti za imenice, pridjeve i glagole unešeni, zajednički kod odrađen. Pridjevi - objekti riječi
23.10.2009	pridjevi gotovi, *refactoring* sustava za opojavničenje, kodne stranice,

datum	*kratki opis događaja*
	bolje detekcije i sl.
27.10.2009	ubrzanja – sustav za opojavničenje, dohvat iz baze za grafičko sučelje
04.11.2009	započeta izrada sustava za prepoznavanje osnovnih oblika riječi prema tvorbeno-flektivnim sufiksima - radi samo sa pridjevima
26.11.2009	završena prva verzija, započet rad na objektima riječi vrste imenice
13.12.2009	završene imenice, slijedi prilagodba sustava za prepoznavanje osn. oblika za imenicama
16.12.2009	završene prilagodbe za imenice, započinje se rad na objektima riječi vrste glagoli
31.12.2009	završena klasa za glagole, prilagodba sustava za prepozn. osn. oblika za glagole
03.1.2010	završene prilagodbe, započeta prilagodba za ubacivanje prepoznatih objekata u bazu podataka + *refactoring* zbog brzine i ostalih potrebnih prilagodbi
24.1.2010	sustav pripremljen za punjenje baze s prepoznatim objektima riječi (osnovni oblici)
26.1.2010	prepoznavanje pokretano inkrementalno i svi podatci prebačeni u bazu, započete analize i rad nad prototipom sustava za pretraživanje
04.2.2010	prototip sustava napravljen i spreman za rad. Implementacijski dio rješenja završen.
11.2.2010	započet rad nad tekstom ovog rada

Veličina izvornog koda je oko 820 KB (bez doc-testova), s tim da je podjela ovakva:

- 290 KB - grafičko sučelje,
- 80 KB – sustav za opojavničenje,
- 400 KB - infleksijski sustav za hrvatski jezik ,
- 50 KB - ostalo.

U ukupno 228 dana kodiranja izvršeno je oko 700 *commit-ova* u repozitorij izvornog koda, te je, prema gruboj procjeni, utrošeno oko 380 radnih sati.

10.2 Iscrpniji pregled modela podataka

U ovome se poglavlju daju iscrpne informacije o modelu podataka koji je opisan u poglavlju *Model podataka*. Dana je lista svih modela s popisom polja koje sadrže, kratkim opisima i dodatnim specifikacijama vrijednosti koje mogu poprimiti. U modelima nije naveden primarni ključ, no primarni ključ postoji, zove se id, te se od same baze dodjeljuje automatski (*autonumber*).

10.2.1 Word

Ovaj model ima sljedeća polja:

Naziv	*Null?*	*Tip*	*Opis*
value	ne	Char, unique	Sama riječ - npr. pišem
valtype	ne	Char(3)	Vrsta riječi po sadržaju, iscrpnije opisano u nastavku
wordobj_main	DA	ForeignKey(WordObj)	U slučaju da se ista riječ pojavljuje u više oblika različitih osnovnih riječi onda ovo polje određuje koja je osnovna riječ nositelj ovog oblika (važno pri procesu normalizacije)
status	ne	Char(1)	Status riječi
-			**Statistička polja**
freq	ne	Integer	Učestalost riječi (**frequency**)
doc_inserted	DA	FK(Doc)	Koji je dokument/knjiga prvi put ubacio ovu riječ
forms_count	ne	Integer	Broj različitih osnovnih riječi (WordObj) koji imaju ovu riječ kao jednu od svojih oblika

Premda se ovdje govori o riječi, točnije bi bilo govoriti o pojavnicama, odnosno o najmanjim gradivnim pojavnicama teksta kojeg promatramo. To su u najvećem broju riječi, ali mogu biti i znaci interpunkcije, prazni redovi, rimski brojevi, arapski brojevi itd.

Polje *valtype* govori o vrsti sadržaja riječi, a više o tome kako se ove vrijednosti pune objašnjeno je u poglavlju *Opojavničenje na odjeljke, rečenice i riječi*:

valtype	***Opis***
WRD	Obična riječ
WRDN	Riječ koja je ujedno i ime
WRDNS	Riječ-ime koja može započeti rečenicu
WAB	Skraćenica (*abbreviation*)
EMP	Prazno - recimo tabulator ili slično - no ne i razmaci
ENL	Prazno - samo oznake novih redova
-	**Znakovi interpunkcije**
SEN	Oznaka kraja rečenice
SIS	Oznaka rečeničnog prekida, npr. ,
S1-	Oznaka podrečeničnog dijela, npr. "
S2S	Oznaka podrečeničnog dijela - početak , npr. (
S2E	Oznaka podrečeničnog dijela - kraj, npr. }
-	**Brojevni simboli**
NUM	Arapski broj, npr. 123.309,99
RNR	Rimski broj, npr. MDXXXIX
-	**Neprecizno prepoznavanje**
?AB	Neobična skraćenica
?FZ	Neobičan sadržaj
?UN	Nepoznato

Razmaci nisu registrirani, već se ignoriraju.

Predefinirana vrijednost za status je U, a status može biti jedan od sljedećih:

Sts	*Opis*
U	Neprovjerena i neobrađena riječ
C	Potvrđena riječ
F	Obrađena, sustav sugerira da je dobra
T	Obrađena, sustav nema sugestija
R	Odbačena
D	Dvostruka

10.2.2 WordObj

Ovaj model ima sljedeća polja:

Naziv	*Null?*	*Tip*	*Opis*
value_base	ne	Char	Osnovni oblik riječi (npr. infinitiv za glagole)
word_type	ne	Char	Vrsta riječi - pogledati sljedeću tablicu
subtype	ne	Char	Za prepoznate riječi drži vrijednosti koje treba proslijediti konstruktoru objekta da bi se dobio *pravi* objekt. Više detalja u *Svođenje riječi na osnovni oblik.*
freq_type	ne	Char(1)	Označava *zaustavnu riječ*, rijetke ili obične riječi. Popis vrijednosti u jednoj od tablica koje slijedi.
status	ne	Char(1)	Status osnovne riječi. Vrijednost statusa je ista kao i za Word.
detect_spec	ne	Char(100)	Kod prepoznate osn.riječi ovo polje drži informacije na osnovu kojeg *detection* objekta je ova riječ prepoznata. Više detalja u *Svođenje riječi na osnovni oblik.*
key	ne	Char(30)	Kod prepoznatih osnovnih riječi ovo polje sadrži ključ koji ju čini jedinstvenom. (tj. vrsta riječi/osnovni oblik/parametri konstruktora). Više detalja u *Svođenje riječi na osnovni oblik.*
word_forms	ne	M2Many(Word)	Preko tablice WordForm
-			**Statistička polja**

Naziv	*Null?*	*Tip*	*Opis*
sum_forms _freq	ne	Integer	Suma učestalosti svih oblika riječi za ovu osnovnu riječ.
cnt_forms_f req	ne	Integer	Od svih svojih oblika - broj onih koji imaju ovu osnovnu riječ izabranu za glavnu (wordobj_main - pogledati u Word). Više detalja u *Svođenje riječi na osnovni oblik.*
sum_main_ freq	ne	Integer	Isto kao prethodno polje, samo se ne gleda broj već suma svih učestalosti..
cnt_doc	ne	Integer	Broj dokumenata gdje se pojavljuje bilo koji od oblika ove osnovne riječi

Polje *word_type* može imati jednu od sljedećih vrijednosti:

word_type	*Promjenjive*	*Opis*
ABBR	ne	skraćenice
ADV	ne	prilozi
CONJ	ne	veznici
EXCL	ne	uzvici
PART	ne	čestice
PREP	ne	prijedlozi
PRON.NEO.FIX	ne	Zamjenice - podvrsta neodredjene i nepromjenjive
ADJ	DA	pridjevi
N	DA	imenice
V	DA	glagoli
NUM	DA	brojevi 38
PRON.OSO	DA	Zamjenice osobne
PRON.POV	DA	Zamjenice.povratne
PRON.POS	DA	Zamjenice.posvojne
PRON.PPO	DA	Zamjenice.povratno-posvojne
PRON.POK	DA	Zamjenice.pokazne
PRON.UOD.IME	DA	Zamjenice.upitne_odnosne.imenicne

word_type	*Promjenjive*	*Opis*
PRON.UOD.PRD	DA	Zamjenice.upitne_odnosne.pridjevne
PRON.NEO.CH1	DA	Zamjenice.neodredjene.promjenjive
PRON.NEO.CH2	DA	Zamjenice.neodredjene.promjenjive+spol

Polje *freq_type* može imati sljedeće vrijednosti:

freq_type	*Opis*
-	Obične riječi (*normal*)
S	*Stopword* - više pojedinosti u *Zaustavne riječi (engl. stopwords)*
R	Vrlo rijetke riječi - trenutno nije u upotrebi

Kako se koristi status i kako se mijenja bit će iscrpnije objašnjeno u poglavlju *Svođenje riječi na osnovni oblik*. Predefinirana vrijednost za status je U, a sadržaj polja status je isti kao i za model Word, pa je potrebno pogledati tamo za popis vrijedosti s kratkim opisom.

10.2.3 WordForm

Riječ je o jednostavnoj tablici koja služi za ostvarenje više-više relacije između WordObj i Word modela. Smisao je da se kroz ovu tablicu unesu podatci o oblicima same osnovne riječi (npr. za osnovnu riječ *pisati* - veza prema *pisati*, *pišem*, *pišeš*, ... *pisali*, *pisan* ...).

Ovaj model ima sljedeća polja:

Naziv	*Null?*	*Tip*	*Opis*
wordobj	ne	ForeignKey(WordObj)	strani ključ
word	ne	ForeignKey(Word)	strani ključ

10.2.4 DocSource

Ovaj model ima sljedeća polja:

Naziv	*Null?*	*Tip*	*Opis*
name	ne	Char	Naziv izvora dokumenata (npr. Narodne novine)
root	ne	Char	Putanja do direktorija
is_recursive	ne	Bool	Kod pretrage direktorija za dokumentima je li treba ići rekurzivno prema dolje u hijerahiju poddirektorija.
filter	ne	Char	Filter za datoteke koje predstavljaju dokumente (npr. *.html, *.txt itd.)
preprocessor	ne	Char	Prilikom opojavničenja koja se vrsta predprocesiranja treba primijeniti na dokument da se pretvori iz izvornog formata u tekstovni format. Popis implementiranih predprocesora je u sljedećoj tablici. Više pojedinosti u *Opojavničenje na odjeljke, rečenice i riječi.*
-			**Statistička polja**
last_update	ne	DateTime	Datum i vrijeme zadnjeg ažuriranja popisa dokumenata.
doc_count	ne	Integer	Koliko je dokumenata prijavljeno pod ovim izvorom dokumenata.

Polje *preprocessor* može imati sljedeće vrijednosti:

preprocessor	*Opis*
none	Bez predprocesiranja - tekstovna datoteka
bible	Poseban predprocesor za Bibliju. Više pojedinosti u *Biblija*
html	Predprocesor za html datoteke
doc	Predprocesor za Microsoft Word datoteke

10.2.5 Doc

Ovaj model ima sljedeća polja:

Naziv	*Null?*	*Tip*	*Opis*
source	ne	FK(DocSource)	Strani ključ
status	ne	Char	Status dokumenta. Popis vrijednosti u sljedećoj tablici.
errmsg	ne	Text	U slučaju greške kod opojavničenja u ovom polju su pojedinosti pogreške.
url	ne	Char	Lokacija datoteke dokumenta u datotečnom sustavu.
preprocessor	ne	Char	Predprocesor za ovaj dokument. predefinirana je vrijednost ona koja je definirana za DocSource kojem pripada.
codepage	ne	Char	Kodna stranica teksta ovog dokumenta. Više pojedinosti u *Opojavničenje na odjeljke, rečenice i riječi.*
words_occured	ne	ManyToMany(Word)	Preko tablice DocWordOccured
-			**Statistička polja**
url_size	ne	Integer	Veličina datoteke dokumenta u datotečnom sustavu.
url_modified	ne	DateTime	Zadnje vrijeme promjene datoteke dokumenta u datotečnom sustavu.
last_update	ne	DateTime	Datum i vrijeme zadnjeg opojavničenja.
duration	ne	Integer	Trajanje zadnjeg opojavničenja.
cnt_tok_all	ne	Integer	Broj svih pojavnica ovog dokumenta.
cnt_tok_wrd	ne	Integer	Broj pojavnica vrste *riječ* ovog dokumenta

Polje *status* može imati sljedeće vrijednosti:

status	***Opis***
NU1	Dokument upravo registriran, opojavničenje nije još obavljeno
NU2	Verzija dokumenta u datotečnom sustavu se promijenila – gledaju se atributi veličine i datuma/vremena zadnje promjene (polja *url_size* i *url_modified).* Potrebno novo opojavničenje.
NU3	Ručno označen za opojavničenje.
NUE	Zbog greške u prethodnom opojavničenju, označen za novo opojavničenje.
UP	Nije potrebno novo opojavničenje, dokument je ažuran.
MS	Registrirani dokument više ne postoji u datotečnom sustavu pod tim imenom i mjestom.
DL	Dokument je bio registriran i označen da je pobrisan u datotečnom sustavu.

10.2.6 DocWord

Ovaj je model pomoćna tablica za ostvarenje više-više relacije (*many to many*) za modele Doc i Word. Budućida ova tablica ima ogroman broj zapisa, ona je razbijena tako da svaki Doc objekt ima ovu tablicu u posebnoj *bazi*. Više o tome u poglavlju *Opojavničenje na odjeljke, rečenice i riječi*.

Primjećuje se da je jedan red u ovoj tablici ono što nazivamo **pojavnica** (engl. ***token*)**, odnosno najmanja gradivna jedinica koju promatramo. Osim podatka o samoj pojavnici, u kombinaciji s podatcima iz Word modela pohranjene su i neke strukturne meta informacije:

- pojavnica je *riječ* - Word -> *valtype* = WRD,
- pojavnica nosi informaciju o strukturi unutar rečenice - interpunkcijski znakovi kao zarez, točka, itd. Pogledati tablicu u Word modelu za *valtype*,
- u određenom dokumentu za određenu pojavnicu – je li nosi informaciju o strukturi unutar dokumenta - kao što je početak i kraj rečenice, početak poglavlja. To je spremljeno u ovom modelu DocWord u polju struct_tag (pogledati i tablicu s mogućim vrijednostima).

Ovaj model ima sljedeća polja:

Naziv	*Null?*	*Tip*	*Opis*
doc	ne	ForeignKey(Doc)	Strani ključ
word	ne	ForeignKey(Word)	Strani ključ
location	ne	Integer	Položaj pojavnice u dokumentu. Budući da u relacijskim bazama redoslijed nije *zagarantiran,* onda se redoslijed mora sačuvati u polju.
struct_tag	ne	Char	Informacija o strukturi unutar dokumenta. U sljedećoj je tablici popis dopuštenih vrijednosti.

Polje *struct_tag* može imati sljedeće vrijednosti:

struct_tag	*Opis*
-	Bez uloge
PST	Početak poglavlja
SST	Početak rečenice
SEN	Kraj rečenice

10.2.7 DocWordOccured

Ovaj je model sumarna tablica nastala nad tablicom DocWord i daje podatak koja se riječ pojavljuje u kojem dokumentu i s kojom učestalošću (*frequency*).

Ovaj model ima sljedeća polja:

Naziv	***Null?***	***Tip***	***Opis***
doc	ne	ForeignKey(Doc)	Strani ključ
word	ne	ForeignKey(Word)	Strani ključ
first_location	ne	Integer	Lokacija prve pojave riječi/pojavnice u dokumentu
last_location	ne	Integer	Lokacija zadnje pojave riječi/pojavnice u dokumentu
freq	ne	Integer	Broj pojavljivanja riječi/pojavnice u dokumentu (*frequency*)

11 Popis literature

Ovo je popis korištene literature koja je referencirana u radu:

- [1] "Retrieving Information in Croatian: building a simple and efficient rule-based stemmer" *http://infoz.ffzg.hr/ljubesic/nldbok_inf07.pdf*, 2007, Nikola Ljubešić, Damir Boras, Ozren Kubelka
- [2] "Automatic Acquisition of Inflectional Lexica for Morphological Normalisation Information Processing & Management", Jan Šnajder, Bojana Dalbelo Bašić, Marko Tadić; Information Processing & Management, vol. 44, no. 5, str. 1720-1731, 2008.
- [3] "Postupci morfološke normalizacije u pretraživanju informacija i klasifikaciji teksta", Jan Šnajder; *http://www.fer.hr/_download/repository/Snajder_KDI.pdf*
- [4] "Gramatika hrvatskog jezika, priručnik za osnovno jezično obrazovanje", Stjepko Težak, Stjepan Babić, Školska knjiga, 15. izdanje, 2005

Ovo je popis ostale korištene literature:

- "Data Mining, Concepts and techniques", Jiawei Han, Micheline Kamber; 2006, Morgan Kaufman
- "The definitive guide to Django, Web development done right", Adrian Holovaty, Jacob Kaplan Moss; 2008, Apress
- "Practical Django projects", James Benett; 2008, Apress
- "Programming collective intelligence", Toby Segaran; 2007, O'Reilly
- "Ekstremno programiranje", Steward Baird; 2002, Kompjuter biblioteka

12 Popis internetskih poveznica

12.1 Popis poveznica vezanih uz normalizaciju tekstova na hrvatskom jeziku

Najznačajniji resursi popisani su u poglavlju *Uvod*, a ovdje slijedi popis ostalih resursa (2009-03):

- *http://www.hnk.ffzg.hr/* - Hrvatski nacionalni korpus
- *http://hjp.srce.hr/* - Hrvatski jezični portal, vrlo zanimljiv projekt
- *http://www.ihjj.hr/#etrjecnik* - Etimološki rječnik hrvatskoga jezika
- *http://crodip.ffzg.hr/* - Hrvatska rječnička baština i prikaz rječničkoga znanja
- *http://ling.unizd.hr/~dcavar/news/files/* ba40b4e8869d8e948831b68595948d43-20.html CroMo - Morfološki parser, obilježavač i lematizator za hrvatski jezik
- *http://jthj.ffzg.hr/rjecnici.htm* - Jezične tehnologije za hrvatski jezik, poveznice na hrvatske digitalne rječnike
- *http://hacheck.tel.fer.hr/* - Hrvatski akademski spelling checker
- *http://riznica.ihjj.hr/* - Hrvatska jezična riznica
- *http://www.ihjj.hr/Projekti.aspx* - Institut za hrvatski jezik i jezikoslovlje, Veliki rječnik hrvatskoga jezika
- *http://rmjt.ffzg.hr/p5_en.html*, *http://flambard.zemris.fer.hr/* - Knowledge discovery in textual data
- *http://www.irb.hr/hr/home/mmalenica/radovi/* - Primjena jezgrenih metoda u kategorizaciji teksta
- *http://june.irb.hr/~sthagon/irbweb/kategorizacija_teksta.pdf* - tekst nedostupan
- *http://bib.irb.hr/prikazi-rad?&rad=314312* - znanstveni rad "N-Grams and Morphological Normalization in Text Classification: A Comparison on a Croatian-English Parallel Corpus Book Serie", tekst nedostupan
- *http://www.hidra.hr/* - Hrvatska informacijsko-dokumentacijska referalna

agencija, Vlade Republike Hrvatske

- *http://hobs.ffzg.hr/poveznice.html* - Hrvatska ovisnosna banka stabala - popis poveznica
- *http://www.hidra.hr/eurovoc/eurovoc1.HTM*, *http://europa.eu/eurovoc/*, *http://www.hidra.hr/cro/pojmovnik_eurovoc* - Eurovoc thesaurus, postoji i za hrvatski jezik

13 Prilozi

13.1 Programsko okruženje - pojedinosti

U poglavlju *Programski alati* dane su za ovaj rad najvažnije karaktistike nekih od programskih alata koji su korišteni, dok će se u ovom poglavlju dati malo više pojedinosti o svakome od njih i pregled ostalih programskih alata koji su korišteni.

13.1.1 Sklopovsko okruženje

Sustav je razvijan na:

- prijenosnom računalu Hp nx9420
- 1 GB memorije
- CPU - Intel Core Duo 1.8 GHz
- OS - Windows XP Pro

Računalo se pokazalo da se može nositi s većinom zahtjevnijih zadataka. Za neke teže zadatke zahtjevi su ipak bili preveliki, pa unatoč svom trudu koji je uložen u optimiranje procesa, obrade su se ipak trebale vrtiti satima. Takav radni ritam nije prirodan za prijenosno računalo, te je zaključak da se u sljedećoj iteraciji napravi sljedeće:

- obaviti iscrpnu reviziju svih zahtjevnijih obrada, te obaviti *refactoring* gdje je to moguće i isplativo [61]
- pokušati takve obrade paralelizirati
- puštati takve obrade na snažnijem računalu/računalima

13.1.2 Python programski jezik

Python je *Open source* interpreterski jezik s visokom razinom abstrakcije, specifičnom i vrlo čistom sintaksom. Baš zbog toga je brzina učenja vrlo brza, te ga preporučuju i za edukacijske svrhe.

Po TIOBI indeksu, sedmi je jezik po upotrebi [62].

61 *http://www.wikipedia.org/Code_refactoring*

62 TIOBI indeks o popularnosti programskih jezika *http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html*

Python je primarno objektni jezik, no omogućuje i pisanje strukturnih programa [63]. Ima elemente funkcijskog programiranja [64] i aspektno usmjerenog programiranja [65].

Python je ujedno i dinamički jezik [66], što znači da se vrsta objekata ne provjerava, već se primjenjuje *duck typing* princip [67]. Dinamički su jezici sve popularniji i prema TIOBI istraživanju sve su zastupljeniji, a trenutno je stanje 40% u odnosu na 60% koje zauzimaju statički jezici .

Radi ilustracije, njemu slični programski jezici su PHP, JavaScript, Ruby.

Sve je češća upotreba python-a kao skriptnog jezika [68], a posebno u ulogama za pisanje ugradbenih komponenti u postojeće sustave (*plugin*, *embedded language*), skripti za sistemsku i db adminstraciju, obradu teksta i pisanje instalacijskih i aplikacija za upravljanjem sustavom. Sve je popularniji i u području pisanja web aplikacija.

Primjeri upotreba:

- Ubuntu, Red Hat Linux, Gentoo Linux imaju instalacijske i/ili administracijsko/upravljačke GUI alate napisane u pythonu.
- Groups.google.com je web aplikacija napisana u pythonu.
- BitTorrent klijent napisan je u pythonu.
- Mercurial [77] je sve popularniji distribuirani sustav za vođenje verzija programa (*distributed versioning control system*) i napisan je u pythonu.
- Python je standardno uključen u Linux distribucije i u Mac OS X.
- Koristi se u kao skriptni jezik u programima za 2D i 3D animaciju i modeliranje kao što su Maya, modo, Blender, GIMP, Inkscape, Scribus, Paint Shop Pro itd.
- Python je preferirani skripni jezik za ESRI ArcGis sustave.
- Python je prvi jezik implementiran za Google Application Engine (*https://appengine.google.com/*).

63 *http://en.wikipedia.org/wiki/Structured_programming*
64 *http://en.wikipedia.org/wiki/Functional_programming*
65 *http://en.wikipedia.org/wiki/Aspect-oriented_programming*
66 *http://en.wikipedia.org/wiki/Type_system#Dynamic_typing*
67 *http://en.wikipedia.org/wiki/Type_system#Duck_typing*
68 *http://en.wikipedia.org/wiki/Scripting_language*
77 *http://mercurial.selenic.com/*

- Često se koristi kao skriptni jezik za računalne igre, a posebno za mrežne.
- od velikih kompanija koriste ga Google, Yahoo, CERN, NASA itd.

Python je multiplatformski jezik i isti se program može vrtjeti na Microsoft Windows, Linux, Unix, BSD, Mac OS X, na malim uređajima (Symbian, Android), unutar Jave (www.jython.org), unutar .Net *framework* (www.ironpython.net) itd.

Python je stabilan i brz programski jezik, jednostavan za instalaciju i pokretanje. Jezik standardno podržava složene tipove podataka kao string, polja (*list*), rječnike (*dictionary*), kompleksni brojevi itd.

Dolazi sa ugrađenim skupom biblioteka za datotečni sustav, procese, mrežne protokole, *unit-testing* [69], *doc-testing*, XML/HTML, sažimanje podataka, serializaciju podataka, sqlite bazu podataka, berkleydb bazu podataka [70], *logging*, direktan pristup funkcijama i strukturama iz biblioteka operacijskog sustava (*.dll, *.so), GUI, *debugger*, *profiler*, internim strukturama samog programa (AST), i18n, paralelno programiranje (*multi-threading*, *multi-processing*) itd.

Za sve ono što nije pokriveno ugrađenim bibliotekama vrlo je vjerojatno da već postoji dodatna biblioteka koja se može slobodno koristiti (*Open source*). Ilustracije radi, na *http://pypi.python.org/pypi* stranicama, koje služe za prijavu takvih biblioteka, ima prijavljeno više od 9000 biblioteka.

12.1.3 Sqlite baza podataka

Od ostalih karakteristika možda je važno napomenuti da je ova baza *Open source* projekt, te je vrlo raširena i po nekim istraživanjima baza s najviše primjeraka na svijetu (500 milijuna). Upotrebljava se u svakom Firefox internetskom pregledniku, u Symbian operacijskom sustavu za mobitele, standardna baza u Adobe Air sustavima, koristi se u Dropbox [80] klijentu itd. Od *nedostataka*, najvažniji je to što nije mrežna baza podataka, odnosno ukoliko ne postoji mogućnost pristupa samoj datoteci u kojoj su smješteni podatci, ne postoji mogućnost pristupa samoj bazi.

69 *http://en.wikipedia.org/wiki/Unit_testing*

70 *http://www.oracle.com/technology/products/berkeley-db/index.html*

80 *https://www.dropbox.com/*

13.1.4 Django razvojno okruženje za web aplikacije

Django je razvojno okruženje za web aplikacije (*web framework*) za *Python* programski jezik. Od prednosti Django razvojnog okruženja izdvaja se sljedeće:

- diktira Model-template-view (MTV [71]) arhitekturu programa koja, premda dosta slična, ipak bolja od *popularnije* Model-view-controller [72] arhitekture. Prateći ovu arhitekturu i dizajn programa koji Django diktira, konačno rješenje vrlo je uredno i organizirano.
- Django je sam pisan i promovira *Don't repeat yourself* (DRY) [73] pristup razvoja programa, što pridonosi kvaliteti programa.
- Kao i za DRY - Django je dobro istestiran (Unit/doc-testing), te sam promovira pisanje testnih programa (unit/doc-testing).
- Omogućuje brz razvoj web aplikacija (*Rapid web development*). U roku od nekoliko minuta možete napraviti kostur prve aplikacije i pokrenuti je u djangovom razvojnom serveru (*development server*).
- Pristup bazi podataka obavlja se preko objekata na unificiran način pa stoga nema potrebe za pisanjem SQL-ova.
- Instalacija je *monolitna* - odnosno ne ovisi ni o jednom drugom python *vanjskom* paketu.
- Dokumentacija je odlična, podrška razvojnog tima i ostatka *community*-a je na vrlo visokom nivou, broj *Open source* projekata napravljenih na Djangu veoma je velik, velik je broj blog članaka s temom vezanom uz Django, veliki broj primjera [74] itd.

Najsličnije web razvojno okruženje Djangu, a s kojim ga se često uspoređuje, je Ruby on rails [75]. Prema google trends upitu nazire se da popularnost Djanga neprestano raste, dok RoR pada [76].

71 *http://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names*

72 *http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller*

73 *http://en.wikipedia.org/wiki/Don%27t_repeat_yourself*

74 *http://www.djangosnippets.org/*

75 *http://rubyonrails.org/*

13.1.5 Ostalo

Od ostalih alata i internetskih servisa koji su korišteni izdvoja se sljedeće:

Nltk - Natural Language Toolkit - vrlo opsežna OSS python biblioteka koja omogućuje veliki broj NLP [41] funkcija. Vrlo dobra dokumentacija, veliki broj primjera, stabilan i u stalnom razvoju. Za potrebe ovog rada ovaj je paket korišten primarno radi potreba učenja. U budućnosti će se izvršiti integracija i korištenje dijelova ovog paketa.

Mercurial - distribuirani sustav za vođenje verzija (DVCS *distributed versioning control system* [79]). Podržava sve glavne naredbe već svjetski raširenog Subversion-a (svn), većinu njegovih mogućnosti, te dodaje vrlo vrijedan skup naredbi koje uvelike olakšavaju distribuirani razvoj. Jedna od značajnijih razlika u odnosu na standardne VCS sustave je da je svaka radna kopija (*working copy*), ujedno i repozitorij koji je jednako vrijedan kao i svaki drugi. Kloniranje, razmjena promjena u *push*/*pull* i slično, stvar je pozivanja samo jedne naredbe (npr. hg clone, hg push, hg pull, itd.). Brz, stabilan, jednostavan za instaliranje i upotrebu, a za korisnike Subversion-a omogućuje jednostavan prelazak. To je ujedno i *Open source* projekt i pisan je u Pythonu. Slični sustavi su Git i Bazaar. Sav programski kod, ovaj rad i popratna dokumentacija sprema se u lokalni Mercurial repozitorij i dalje distribuira na BitBucket i još jedan sustav (pojedinosti u sljedećem odjeljku).

BitBucket [78] - internetski servis za držanje Mercurial repozitorija. U svojoj besplatnoj inačici daje jedan privatni repozitorij do veličine od 1 GB i neograničeni broj javnih repozitorija. Repozitorij ovog projekta sinkronizira se na 3 mjesta, od kojih je jedan BitBucket koji ima ulogu *online* sigurnosne kopije i distribucijskog čvora.

Dropbox - internetski servis za spremanje i distribuciju datoteka. Koncept je sličan sustavima za vođenje verzija (VCS - *versioning control system*) - koji je više namijenjen razvoju programskog koda u timovima, a Dropbox je osobni servis za držanje i distribuciju bilo kakvih datoteka - obično osobnih. Takvi i slični servisi nazivaju se i *online backup*. Za potrebe ovog rada je korišten za držanje sigurnosne kopije baze podataka, sinkronizaciju alata i dokumentacije i korespodencije. Dva su klijenta instalirana, jedan na privatnom prijenosniku na kojem je rađen sav razvoj, a drugi na svom radnom mjestu. Besplatni servis koji pruža Dropbox je ograničenje na

76 *http://www.google.com/trends?q=django%2C+ruby+on+rails+*

79 *http://en.wikipedia.org/wiki/Distributed_Version_Control_System*

78 *http://www.bitbucket.org*

2 GB prostora, koji se može proširiti i do 5 GB.

reStructuredText [81] - jednostavni tekstovni jezik koji se s posebnim alatom može prebaciti u neki od sljedećih formata: html, laTex, pdf, odt (open office document), xml, docbook itd. Sintaksa je dosta jednostavna, te omogućuje najvažniji skup mogućnosti potrebnih kod kreiranja tekstovnih dokumenata (naslovi, paragrafi, poglavlja, sadržaj, reference, fusnote, slike, tablice, poveznice, nabrojane stavke, bojanje programskog koda, napredne mogućnosti stiliziranja uz pomoć CSS-a i slično. Ono što se vidi kao najveća prednost je da su tekstovi u txt formatu i mogu se spremiti u VCS (*versioning control system*), te se mogu lako pratiti promjene nad tekstom. Uređivanje takvih dokumenata može se napraviti u bilo kojem tekstovnom editoru. Rst format zamišljen je tako da bude jednako dobro čitljiv i u izvornom obliku (rst tekstovna datoteka).

Gvim [82] - GUI verzija poboljšane verzije (*Graphical VI iMproved*) standardnog i legenderanog Vi *nix editora. Editor ima dugu povijest, ali još je uvijek živ i dosta popularan. Premda ima strmu krivulju učenja i za današnje doba vrlo specifičan način uređivanja teksta (npr. Modovi), trud uložen u učenje se isplati i vrlo brzo nakon toga čovjek postane učinkovitiji pri pisanju i uređivanju tekstova. Svi tekstovi koji su se obrađivali prilikom implementacije ovog programskog rješenja i pisanja ovog rada rađeni su u gvim-u.

81 *http://docutils.sourceforge.net/rst.html*

82 *http://www.gvim.org*

Sažetak

Naslov: Lociranje sličnih logičkih cjelina u tekstualnim dokumentima na hrvatskome jeziku

Rad opisuje proces izrade programskog rješenja za pretraživanje dokumenata na hrvatskom jeziku na osnovu sličnosti tekstova (engl. *text similarity*).

Za potrebe pretraživanja potrebno je tekstove na hrvatskom jeziku normalizirati, što znači izdvojiti *zaustavne rije*či iz tekstova, te onda svesti oblike riječi na korijenske oblike procesom lematizacije. Napravljeno je vlastito rješenje koje ostvaruje tražene ciljeve. U tu svrhu izrađen je morfološki leksikon koji je dobiven na osnovu rječnika dobivenog iz skupljenog korpusa i sustava za obrnutu infleksiju.

Za potrebe pretraživanja po sličnosti odabran je način da se u procesu opojavničenja (engl. *tokenization*) u kojem se ne označavaju samo *normalne* riječi, već proces opojavničenja omogućava i prijelom teksta na odjeljke i rečenice, što u ovom slučaju omogućuje jednostavnije i preciznije pretraživanje.

Abstract

Title: Locating similar logical units in textual documents in Croatian language

The thesis describes the process of developing a software solution for search of documents in the Croatian language on the basis of text similarity.

For searching purposes, a Croatian text should first be normalised, that is to say, stopwords should be extracted from the text, after which word forms should be reduced to their canonical forms in the process of lemmatization. The author has designed his own solution capable of attaining the goals set. To this end, a morphological lexicon has been created from the vocabulary of the collected corpus, using a reverse inflexion system.

For the purposes of similarity-based search, a method has been selected whereby it is not only normal words which are parsed during the tokenization process. Rather, the tokenization process is used in order to enable splitting of the text into paragraphs and sentences, which makes search simpler and more precise.

Ključne riječi

Ključne riječi: hrvatski jezik, lematizacija, zaustavne riječi, morfološki leksikon, opojavničenje, sličnost tekstova, part-of-speach tagging

Keywords: Croatian language, lemmatization, stopwords, morphological lexicon, tokenization, text similarity, part-of-speach tagging

Životopis

Robert Lujo rođen je 1974. u Dubrovniku. Osnovnu i srednju školu završio je u Dubrovniku, a godine 1993. upisuje Fakultet elektrotehnike i računarstva. Na početku četvrte godine fakulteta u slobodno vrijeme počinje se aktivno baviti digitalnom elektronikom na Zavodu za računalne sisteme i procese, kod prof. doc. Maria Kovača. Godine 1997. diplomirao s temom "Programska podrška za digitalizator video signala".

Iste godine zapošljava se u "Končar informatika i elektronika (INEM) d.o.o.", gdje kao pripravnik radi u Odjelu za organizaciju i obradu podataka. Godine 1998. zapošljava se u osiguravateljnoj tvrtci "Sunce osiguranje d.d." na mjesto projektanta-programera. Radi na projektu uvođenja integralnog informacijskog sustava, te nakon nekog vremena postaje voditelj projekta.

Godine 2000. zapošljava se u tvrtci "Combis d.o.o." na radnom mjestu projektanta-programera. Godine 2002. zapošljava se u tvrtci "Intes computer, predstavništvo u RH", na radno mjesto voditelja uvođenja integralnog informacijskog sustava u slovenskoj osiguravateljnoj tvrtci "Triglav zdravstvena zavarovalnica d.d.". Tvrtka "Intes computer d.o.o." se bavi razvojem programskih rješenja i konzaltingom primarno vezanih za poslovanje osiguravateljnih kuća. Godine 2009. osniva se *sestrinska* firma "Eu-informatika d.o.o." u kojoj se zapošljava iste godine i radi na istim poslovima.

Primarna profesionalna orijentiranost mu je razvoj kvalitetnih programskih rješenja. Zagovornik je *Open source* incijative, privatno i profesionalno koristi Open source programska rješenja, te koliko mu prilike dopuštaju aktivno sudjeluje u razvoju različitih OSS projekata.

Curriculum vitae

Robert Lujo was born in 1974 in Dubrovnik, where he completed primary and secondary education. In 1993 he enrolled in the Faculty of Electrical Engineering and Computing of Zagreb University. In the beginning of the fourth year at the Faculty, he took an active interest in digital electronics in spare time, and was in this connection involved in some activities of the Department for Computer Systems and Processes, under the guidance of Prof Mario Kovač. He graduated in 1997 with the thesis on "Application for Video Digitalizer".

In 1997 he was hired by the company Končar informatika i elektronika (INEM) d.o.o. as a trainee in their Data Organization and Processing Department. His second job was with the insurance company Sunce osiguranje d.d. (1998), where he held the position of Software Engineer. He worked on introducing an integral IT system, and after a while he became the Project Manager.

In 2000 he found a job as a Software Engineer with the company Combis d.o.o. In 2002 he became employed with Intes Computer – Croatia Office at the position of the manager responsible for introducing an integral IT system in the Slovene insurance company Triglav zdravstvena zavarovalnica d.d. The company Intes Computer is engaged in developing applications and providing consulting services primarily for the insurance industry. In 2009 its daughter company EU informatika d.o.o. was established, and in the same year he took employment there, performing the same job.

His primary professional interest has been focused on developing high-quality software applications. He is a strong supporter of the Open Source Initiative, and uses Open Source software both privately and professionally. To the extent practicable under his present circumstances, he is actively involved in contributing to Open Source projects.