UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Robert Jakomin

Algoritem za analizo kratke povedi v slovenščini

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2012

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Robert Jakomin

Algoritem za analizo kratke povedi v slovenščini

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Matija Marolt
Somentor: dr. Domen Marinčič

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01885/2012
Datum: 03.12.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROBERT JAKOMIN**

Naslov: **ALGORITEM ZA ANALIZO KRATKE POVEDI V SLOVENŠČINI**

ALGORITHM FOR ANALYSIS OF SHORT SENTENCES IN SLOVENIAN LANGUAGE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V okviru diplomskega dela razvijte učinkovit algoritem za analizo kratke povedi v slovenščini s pomočjo lastnega oblikoslovnega označevalnika. Ocenite njegovo časovno zahtevnost in ga vključite v program, ki bo preko grafičnega vmesnika omogočal iskanje po bazi označenih povedi.

Mentor:

doc. dr. Matija Marolt

Dekan:

prof. dr. Nikolaj Zimic

Somentor:

strok. raz. sod. dr. Domen Marinčič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Robert Jakomin,

z vpisno številko 63060523,

sem avtor/-ica diplomskega dela z naslovom:

Algoritem za analizo kratke povedi v slovenščini

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. **Matije Marolta**

  in somentorstvom (naziv, ime in priimek)

  dr. **Domna Marinčiča**

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Najprej se želim zahvaliti svojemu mentorju doc. dr. Matiji Maroltu za pomoč pri izdelavi diplomskega dela ter še posebej za mentorstvo in uvajanje v profesionalno delovno okolje v Laboratoriju za grafiko in multimedije. Zahvaljujem se tudi somentorju, dr. Domnu Marinčiču, za odlične ideje, ki so vodile do še uspešnejšega algoritma. Pomembna zahvala gre doc. dr. Primožu Jakopinu za samo idejo diplomskega dela ter napotke in pomoč pri izdelavi predhodnega diplomskega dela na Filozofski fakulteti. Zahvaljujem se tudi raziskovalcema as. mag. Cirilu Bohaku ter as. Jerneju Južni – oba sta mi pomagala z učinkovitimi nasveti za programsko realizacijo diplomskega dela. Zahvaljujem se tudi doc. dr. Andreji Žele za nekaj konstruktivnih pogovorov in elektronsko različico njenega vezljivostnega slovarja.

Posebno zahvalo izrekam tudi mnogim vnetim računalničarjem, ki na spletu objavljajo brezplačne praktične vaje in zglede, programe, svoje izkušnje in odgovore na najpogostejše težave programerjev – brez njih tega diplomskega dela zagotovo ne bi bilo v tako dodelani obliki.

Zadnja, a nikakor ne najmanj pomembna zahvala velja moji ženi Martini in mojim staršem za vso podporo in razumevanje.

Kazalo

Izvleček

Računalništvo se v zadnjih letih uspešno povezuje z jezikoslovjem. Še posebej učinkovita je postala ta povezava na področju skladnje, kjer zbirke računalniško obdelanih besedil, korpusi, omogočajo hitra iskanja in preverbe jezikoslovnih hipotez. Problem korpusov pa so pogosto za računalništva neveščega uporabnika neprijazni tekstovni iskalniki in morfološko-skladenjska neoznačenost besedila. Oba problema sta za sintetične jezike, kot je tudi slovenščina, še posebno pereč problem. Cilj diplomskega dela je bilo razviti učinkovit algoritem za analizo kratke povedi v slovenščini (do treh besed) s pomočjo lastnega oblikoslovnega označevalnika in algoritem implementirati v nekem programskem jeziku skupaj z grafičnim vmesnikom za iskanje po bazi označenih povedi. Časovna zahtevnost razvitega oblikoslovnega označevalnika je reda O(n), pri čemer je n število besednih oblik vhodnega jezika. Časovna zahtevnost razvitega algoritma je reda k*O(n), pri čemer je *n* število besed v povedi, *k* pa konstantno število iteracij pri analizi besed določene povedi. Morfološki označevalnik je primeren tudi za druge inflektivne jezike, stavčni analizator pa tudi za druge slovanske jezike, z morebitnimi manjšimi prilagoditvami. Algoritem za stavčno analizo ima določene omejitve: je povsem odvisen od pravilnosti oblikoslovnega označevalnika, zaradi osredotočenosti na kratko poved ne prepozna naštevanj in prirednih besednih zvez, zaradi neupoštevanja konteksta povedi ne more povsem natančno ločiti med osebkom in povedkovim določilom (predpostavi se, da je osebek prvi stavčni člen), nadalje algoritem za vsako poved določi le eno možnost analize oz. interpretacije, in sicer prvo, zaradi svoje formalne narave algoritem tudi ne more uspešno ločevati med predmeti in prislovnimi določili. Analiza daljših povedi je sicer možna že na trenutni stopnji razvoja, ni pa še bila preizkušena.

Abstract

In recent years computing has been successfully applied to problem solving in linguistics. This statement is especially valid in the area of syntax, where computer-processed collections of texts, corpora, allow instant searching and verification of linguistic assumptions. For the users unskilled in computing, complicated user interfaces of text search engines and morpho-syntactically unmarked texts are often a problem. This is particularly acute for synthetic languages such as Slovenian. The aim of the thesis was to develop an efficient algorithm for the analysis of short sentences in Slovenian language (up to three words) using own morphological tagging and to implement the algorithm in a programming language with a graphical user interface for searching the database of syntactically and morphologically marked sentences. The time complexity of the developed morphological tagger is of the order O(n), where n is the number of given word forms of a language. The time complexity of the developed algorithm is of the order k*O(n), where n is the number of words in a sentence and k is the constant number of iterations in the sentence analysis. The morphological tagger is also suitable for other inflective languages, while sentence analyzer only for other Slavic languages, possibly with minor adjustments. The sentence analysis algorithm has certain limitations: it is entirely dependent on the effectiveness of the morphological word tagger, as it is focused on a short sentence it cannot recognize enumerations and coordinate phrases, as it does not delve into the context of the sentence the algorithm cannot exactly distinguish between subject and predicate phrase (it is assumed that the first found noun article in nominative is the subject and the second the verbal phrase), furthermore the algorithm finds only one possibility of the often many that could result from the analysis of each sentence. Moreover, the algorithm cannot, due to its formal nature, effectively distinguish between objects and adverbial phrases. The analysis of longer sentences than three words is already possible at the current stage of development, but has not been tested yet.

Ključne besede

Algoritem, oblikoslovno označevanje, računalniška stavčna analiza, korpus

Key words

Algorithm, morphological tagging, computer sentence analysis, text corpus

1. Uvod

Računalništvo v zadnjih desetletjih uspešno prodira v vse segmente družbe in znanosti, tudi jezikoslovje že dolgo ni več izjema. Na prvi pogled se stroki morda ne zdita tako povezani, vendar je računalništvo kompleksna stroka, na katero lahko gledamo in jo uspešno razlagamo z različnih zornih kotov z zelo raznolikimi modeli, od konkretnega fizikalno-kemijskega in elektrotehniškega prek abstrahiziranega računalniškoarhitekturnega pa vse do formalnih matematičnega in jezikoslovnega. Vsi ti modeli niso povsem alternativni drug drugemu: koncept računalniškega programa, tj. algoritma, je navsezadnje neodvisen od elektronske realizacije. V svojem bistvu gre za program, to je za z navodili podan postopek, ki bi ga lahko izvajal kakršenkoli avtomat, elektronski, elektromehanski ali celo povsem mehanski – lahko bi bil navsezadnje tudi stisnjen zrak. Z višanjem stopnje abstrakcije programskega jezika se dosega večjo neodvisnost od konkretne tehnologije implementacije strojne pa tudi nižjeležeče programske opreme, kot je operacijski sistem. Na nivoju višjih programskih jezikov, tj. na nivoju t. i. tretje[1] in četrte[2] generacije, programski jeziki na nek način zelo počasi napredujejo k bistveno kompleksnejšim in formalno nezajemljivim naravnim jezikom.

Programski del računalništva tako kaže jasne vzporednice tudi z jezikoslovjem, ne le z diskretno oziroma formalno matematiko; bilo bi nenavadno, če se ne bi stroki uspešno povezovali. Računalništvo in računalniško podprto jezikoslovje se je doslej že uspešno uveljavilo na mnogih področjih jezikoslovja, še posebej na področju splošnega in sinhronega. Na področju zadnjega se poleg fonetičnih analiz uspešno uporablja tudi na področju skladnje – pri preučevanju množice računalniško obdelanih besedil, t. i. korpusov. Korpusi ponujajo hitro iskanje besednih zvez po ogromnih zbirkah besedil v rangu nekaj 100 milijonov besed ali celo milijard[3] – obseg, ki je bil še pred nekaj desetletji za jezikoslovca povsem nepredstavljiv in neobvladljiv. Korpusi so zelo učinkovito sredstvo zlasti za preučevanje morfologije, skladnje, frazeologije in besediloslovja, so nepogrešljiv pripomoček za leksikologijo, saj poleg pojavnosti ponujajo tudi informacijo o številu pojavitev posamezne fraze.

Problem korpusov pa so pogosto uporabniku neprijazni uporabniški vmesniki (še posebej za uporabnika, ki ni vešč programiranja oz. vsaj sestavljanja regularnih izrazov) in morfološko-skladenjska neoznačenost besedila. Oba problema sta za sintetične jezike z veliko morfologije, kot so slovanski jeziki, torej tudi slovenščina, še posebno pereč problem – pri analitičnih jezikih, kot je npr. angleščina, to ni taka težava, saj je temelj informacije o skladenjski organiziranosti vsebovan v razmeroma fiksnem besednem redu, ne pa v morfoloških elementih, kot so končnice in pripone. Za angleščino je bilo avtomatsko označevanje na ravni morfologije praktično povsem rešeno že pred dobrim desetletjem. Primer: Britanski nacionalni korpus uporablja sistem CLAWS4, temelječ na skritem modelu Markova, z več kot 97% natančnostjo označevanja, problemi so le pri tujkah [2]. Nekoliko slabši rezultat dajejo označevalniki pri drugih germanskih

[1] Sem štejemo pretežno imperativne programske jezike, ki so nastajali od 50. let 20. stoletja dalje, npr.: Fortran, C/C++, Basic, Pascal, Java, C# itd.

[2] Sem štejemo jezike, ki so nastajali od 70. let 20. stoletja dalje, to so npr.transformacijski jezik za poizvedbe v bazah podatkov SQL (z množico dialektov), statistični R, matematični MATLAB itd.

[3] Največji korpus angleškega jezika, Google N-Grams Corpus, danes vsebuje že prek bilijon besed (1000 milijard) [4].

in romanskih jezikih z več morfologije oz. fleksije, npr. pri nemščini pod 93 % [5], z velikim odstopanjem glede na zvrst besedila; pri slovanskih jezikih so rezultati še slabši [6].

Za slovenščino sicer že obstaja nekaj morfoloških označevalnikov (npr. eden prvih morfoloških označevalnikov za slovenščino Tomaža Erjavca [16], označevalnik Primoža Jakopina, ki je na voljo na spletu [1]; označevalnik podjetja Amebis, ki pa ni javno dostopen [10]), toda njihova težava je, da za večino pojavitev besed v besedilu obstaja več možnosti – dokončen odgovor na to, katera je prava, pa bi dali šele skladenjska, pomenska in sobesedilna analiza. Ker je skladenjska analiza za slovenščino šele v razvoju (doktorska disertacija Domna Marinčiča [9]; Amebis – projekt ni javno dostopen) in me omenjena tematika zaradi interdisciplinarne povezave računalništva (programiranja) in jezikoslovja še posebej zanima, sem se z veseljem lotil algoritma za skladenjsko analizo kratke povedi v slovenščini (samo do treh besed), ki bi se ga kasneje eventualno dalo razširiti tudi na daljše povedi, poleg algoritma pa bi se že v okviru diplomskega dela izdelala tudi konkretna implementacija v programskem jeziku z naprednejšim, a čim bolj enostavnim iskalnikom, namenjenim predvsem za jezikoslovce. Glavni namen je bil torej razvoj stavčnega analizatorja, ki bi povedi razdelil na stavke, te na besedne zveze, slednje pa bi označil s stavčnim členom, v katerega vlogi nastopajo. Omejitev do treh besed se morda zdi skromna, vendar je, upoštevaje predviden obseg in zahtevnost diplomskega dela na splošno, edina realna. Kakršnokoli daljšanje števila besed povedi za analizo bi zahtevalo tudi analizo naštevanj v slovenščini, analiza naštevanj pa je že tema, ki je po zahtevnosti in obsegu primerna za doktorat (prim. [9]).

Po razvoju algoritma za stavčno analizo v okviru moje diplome na Filozofski fakulteti za študij slovenščine ([15]), kjer sem uporabil slovar vnaprej označenih besed (z označevalnikom Primoža Jakopina [1]), sem se odločil diplomsko delo razširiti še z razvojem lastnega oblikoslovnega označevalnika. Izkazalo se je namreč, da pravilnost analize oblikoslovnega označevalnika močno vpliva na pravilnost delovanja algoritma za stavčno analizo, kar je bilo še posebej očitno pri publicističnih besedilih, ki so bila uporabljena za testiranje, saj vsebujejo veliko lastnih imen, ki jih Jakopinov morfološki označevalnik ne pokriva.

2. Analiza algoritma za morfološko označevanje

2.1. Opis zasnove

Algoritem za morfološko označevanje[4] je nastal zaradi potrebe po prepoznavi besed, ki jih morfološki označevalniki, temelječi pretežno na slovarju besednih oblik in najpogostejših n-teric (npr. [1]), ne prepoznajo: predvsem gre za lastna imena. V publicističnih besedilih, ki so služila za preizkušanje algoritma za stavčno analizo, pa je lastnih imen precej. Nepravilno morfološko označene besede precej poslabšajo ali povsem pokvarijo stavčno analizo, zato je bil v okviru diplomskega dela razvit nov algoritem za morfološko označevanje.

Razviti algoritem temelji na strojnem učenju in je v osnovi pravilski klasifikator; poleg morfološkega označevanja je obenem tudi lematizator[5]. Pravila so enonivojska, označevalni algoritem pa deluje povsem formalno in brez upoštevanja konteksta besede, se pravi zgolj na podlagi upoštevanja oblike besede brez upoštevanja njenega pomena in sobesedila. Izpiše vse možne označitve in leme posamezne oblike v uteženem vrstnem redu – najbolj verjetna morfološka označitev je navedena na prvem mestu. Algoritem je uporaben predvsem tedaj, ko enoznačna označitev besedne oblike ni potrebna, ali pa za delovanje v kombinaciji s stavčnim analizatorjem, ki med ponujenimi možnimi označitvami izbere pravo. Algoritem je v precejšnji meri uporaben tudi za narečna in pogovorna besedila z veliko vokalne redukcije, če le imajo morfologijo pretežno enaka kot v jeziku/zvrsti, ki je bila uporabljena za učenje. Razviti algoritem je primeren tudi za druge sintetične oz. inflektivne jezike.

2.2. Psevdokoda[6] algoritma za morfološko označevanje

Spodaj je v javanski psevdokodi zapisan učni del algoritma za morfološko označevanje. Razlaga je podana v nadaljevanju.

```
var Hash<string, List<Rule> transformatioRules;

function LearnMorphologicalTagging{

        foreach(wordForm : wordForms){
                var transformation = DefineTransformation(wordForm.Form, wordForm.Lemma);
                var[] wordEndings = FindEndingsOfLengthFrom1To5(wordForm.Form)
```

[4] Morfološko označevanje pomeni določanje slovničnih kategorij posamezne oblike besede. Npr. besedni obliki 'hišo' lahko določimo sledeče slovnične kategorije: samostalnik, ženski spol, ednina, tožilnik (4. sklon) ali mestnik (6. sklon) – besedna oblika je zaradi dveh možnosti torej dvoumna. Razviti algoritem (kot tudi označevalnik Primoža Jakopina [1]) bi obliko označil sledeče: Sže4, Sže6.

[5] Lematizacija je določanje nevtralne (osnovne) oblike za posamezno obliko besede. Npr. oblika 'hišo' bi bila lematzirana kot 'hiša'. Lematizacija za slovenščino kljub obsežni morfologiji ni zelo hud problem, obstaja več dobrih lematizatorjev za slovenščino, npr. lematizator, razvit na IJS [11], s pribl. 90% natančnostjo.

[6] Zapis v psevdokodi je eden izmed najbolj učinkovitih načinov za predstavitev algoritma. Psevdokoda je način, kako predstaviti algoritem, ki sicer upošteva semantiko, ni pa nujno sintaktično pravilna in je namenjena izključno razumevanju delovanja algoritma. Ni omejena na določen programski jezik, napisana mora biti tako, da je na njeni podlagi mogoča implementacija v kateremkoli programskem jeziku.

```
                if(transformation.changingEnding.Length == wordForm.Form.Length || wordForm.Form.Length
                    <= 2 || wordForm.MorphTag.IsNonFlexibleWordType()){
                    SaveTransformationToExceptions(wordForm.Form, wordForm.MorphTag, transformation);
                    return;
                }

                foreach(wordEnding in wordEndings){
                    SaveTransformationToHash(wordEnding, wordForm.MorphTag, transformation);
                }
        }
}

function SaveTransformationToHash(wordEnding, morphTag, transformation){
        var newRule = new Rule(morphTag, transformation)
        if(transformationRules.Contains(newRule)
                IncreaseWeight(newRule)
        else
                transformationRules.Add(wordEnding, newRule);
}
```

Učni algoritem označevalnika dobi na vhod seznam vseh oblik besed nekega jezika z določeno nevtralno obliko (lemo) in določenimi slovničnimi kategorijami. Za seznam vseh besed slovenščine je bil uporabljen Amebisov besedni zaklad slovenščine (tezaver), ki je del Amebisove zbirke slovarjev. Vsebuje 267.483 gesel (tj. nevtralnih oblik oz. lem), vsako geslo pa vsebuje vse možne odvisne oblike. Vseh oblik v slovarju je prek 1,200.000. Za vhod sicer ni potreben tako popoln seznam besednih oblik, označevalnik lahko razmeroma dobro deluje tudi z manjšo podmnožico, ki pa mora biti čim bolj reprezentativna, to je zajemati mora čim bolj raznolike besedne oblike, ki pokrijejo vse sklanjatve in spregatve ter izjeme.

Učni algoritem za vsako besedno obliko, ki jo dobi na vhod, preveri dolžino končaja besede, ki se spremeni pri transformaciji v lemo (npr. pri obliki 'matere' je transformacija 'ere' → 'i', torej je dolžina končaja za transformacijo 3) – če je dolžina transformacijskega končaja enaka celi besedi (npr. pri zaimku 'njo' → 'ona') ali če je beseda krajša ali enaka od dveh znakov ali pa če gre za nepregibno besedno vrsto (to so predlogi, vezniki, medmeti, členki in nedoločni števniki), algoritem transformacijo shrani v zgoščeno tabelo izjem. Ta del algoritma je torej enak slovarju in ne temelji na strojnem učenju. Pri vseh ostalih primerih pa se razviti klasifikator »uči« tako, da vsako obliko razčleni na končaje: od dolžine 1 do dolžine 5 (oz. do dolžine oblike besede, če je oblika krajša od 5 znakov). Za vsak končaj[7] shrani transformacijo do nevtralne oblike (npr. 'hišo' – transformacija bi bila 'o' → 'a', tj. 'hiša') in morfološko označitev oblike besede (za 'hišo' 'Sže4;Sže6'[8]). Nato v seznamu shranjenih transformacij (transformacijskih pravil) preveri, če že obstaja kakšno tako pravilo (npr. za 'hišo': 'o' → 'a' bi jih bilo kar nekaj, saj gre za precej pogosto tožilniško ali mestniško končnico 1. ženske sklanjatve samostalnika), in če obstaja, se poveča utež za to pravilo. Na ta način se izboljša izvajalni algoritem, saj so tako možnosti morfološke označitve, ki jih vrne, izpisane v verjetnostnem vrstnem redu.

[7] Končaj je v slovenistični terminologiji končni del besede ne glede na morfemskost (zadnjih n-znakov), medtem ko je končnica morfem za sklon (ima vedno svoj pomen, končaj pa ne, če ni obenem tudi morfem). [12]
[8] Gl. podpoglavje 2.5.

Izvajalni algoritem je v osnovi zgoščena tabela iz dveh delov: tabela izjem in tabela transformacijskih pravil z utežmi (verjetnostjo za vsako pravilo). Najprej se vstavljena oblika za označitev preveri, če gre morda za niz števk (cifer), arabskih ali rimskih, v tem primeru se vrne označitev za števnik, sicer pa se vrne označitev in transformacijo v lemo iz ali tabele izjem, če vsebuje to obliko, ali pa, sicer, iz tabele transformacijskih pravil v uteženem vrstnem redu (najprej najbolj verjetna oblika).

2.3. Časovna zahtevnost algoritma

Časovna zahtevnost učnega algoritma je reda O(n * k * m), pri čemer je n število besednih oblik nekega jezika, k število končajev, m pa število nepregibnih tipov besed. Ker sta k in m konstanti, je časovna zahtevnost dejansko linearna, torej reda O(n). Časovna zahtevnost izvajalnega algoritma je enaka časovni zahtevnosti izvajanja zgoščene tabele, to je konstantna – reda O(1).

2.4. Omejitve algoritma

Algoritem zelo dobro odkrije vse teoretične možnosti za morfološko določitev besede. Zaradi neupoštevanja konteksta in pa pomena ter ostalih izvenjezikovnih dejavnikov pri sklepanju govorca jezika pa je možnosti navadno več, kot bi jih predvidel govorec. Zato je razviti označevalnik za uporabo s stavčnim analizatorjem primeren le v kombinaciji s slovarjem, saj zaradi preveč možnosti za označitev besede nastane več veljavnih možnosti za analizo stavka, tako da stavčni analizator težko izbere pravo. V obstoječi rešitvi se oblika besede najprej preveri v slovarju (uporabljen je Amebisov tezaver), in če ta ne vsebuje označene oblike besede, se uporabi tu opisani morfološki označevalnik. Dodatna možnost izboljšanja označevalnika v prihodnosti bi bila dodelava sistema za uteževanje in razvrščanje označitev po verjetnosti, tako da bi recimo označevalnik zavrgel premalo verjetne možnosti.

2.5. Oznake

Za morfološke oznake besednih oblik so bile uporabljene natanko enake oznake, kot jih uporablja označevalnik Primoža Jakopina [1]. Natančneje so razložene na spletnem naslovu [4], kratke opis pa je podan v nadaljevanju.

Oznake za besedne vrste:

- S – samostalnik (primer: 'dan': Sme1):
  - spol: {m, ž, s}
  - število: {e, d, p}
  - sklon: {1, 2, 3, 4, 5, 6}
- G – glagol (primer: 'plava': Gce):
  - oseba: {a, b, c}
  - število: {e, d, p}
- P – pridevnik (primer: 'pomladni': Pme1i):
  - spol, število, sklon – gl. samostalnik
  - stopnjevanje: {/, j, jj}
  - določnost: {/, i}

- Z – zaimki
    - za označitev slovničnih kategorij gl. samostalnik
- Š – števniki
    - za označitev slovničnih kategorij gl. samostalnik

- A – prislov
- Č – členek
- E – predlog (primer: 'iz': E2):
    - sklon, v katerem je podrejena samostalniška beseda: {2, 3, 4, 5, 6}
- V – veznik (primer: 'in': Vpr):
    - prirednost/podrednost: {pr, po}
- M – medmet

3. Analiza algoritma za stavčno analizo

3.1. Opis zasnove

Algoritem je osredotočen na formalno stavčno analizo glagolske povedi in ne upošteva konteksta povedi, deluje zgolj na podlagi slovničnih lastnosti (kategorij) posameznih besed. Rezultat stavčne analize algoritma so tudi slovnične oznake besed (razdvoumi se oznake morfološkega označevalnika). Za uspešnost njegovega delovanja je ključen čim boljši oblikoslovni označevalnik. Najprej je bil uporabljen oblikoslovni označevalnik Primoža Jakopina [1], ki se je sicer dobro obnesel, slabši je bil le pri prepoznavi lastnih imen. Teh pa je v publicističnih besedilih, ki so bila uporabljena za testiranje algoritma, precej, zato je bil na koncu uporabljen lastni morfološki označevalnik, opisan v poglavju 0, v kombinaciji s slovarjem – za slovar je bil uporabljen Amebisov tezaver, saj je popolnejši kot pa označevalnik Primoža Jakopina. Temeljna besedna vrsta za delovanje algoritma je glagol oz. glagolska besedna zveza, na podlagi katere algoritem identificira stavčne meje in ostale stavčne člene, temeljna skladenjska lastnost za delovanje pa je ujemanje med osebkom in povedkom ter med jedrom in prilastki. Algoritem je bil zasnovan z namenom nadaljnjega razvoja in uporabe na daljših povedi. Analiza daljših povedi je sicer možna že na trenutni stopnji razvoja, ni pa še bila preizkušena. Predvideno področje uporabe algoritma je iskanje po tipih stavkov oz. povedi v besedilnih korpusih slovenskega jezika. Razviti algoritem je, z eventualnimi manjšimi prilagoditvami, primeren tudi za druge slovanske jezike s podobnimi razmerji med stavčnimi členi, kot jih ima slovenščina.

3.2. Psevdokoda algoritma za analizo povedi

V nadaljevanju je v javanski psevdokodi zapisan glavni del algoritma, za lažje razumevanje je psevdokoda v slovenščini (imena v programu so v angleščini), tudi struktura objektov je nekoliko poenostavljena. Algoritem najprej razvrsti povedi na tipe glede na to, ali vsebujejo glagolsko besedo. Neglagolskih povedi se v trenutni različici algoritma ne analizira v globino, na podlagi vsebine besed se določi le, ali gre za samostalniške neglagolske ali pa nesamostalniške neglagolske povedi (te ne vsebujejo nobenega samostalnika). Glagolske povedi se analizira tako, da se najprej določi stavčne meje na podlagi formalnih (pravopisnih) označitev v stavku, kot so ločila in velike začetnice, nato pa se v več iteracijah določa stavčne člene. Najprej se določi povedek – algoritem poišče prvo besedno obliko, katere ene izmed morfoloških označitev je glagol, nato pa poišče prvo besedno obliko, katere ena izmed morfoloških označitev je samostalnik, ki se v slovničnih kategorijah ujema z domnevnim povedkom. V nadaljnjih iteracijah se določi še povedkova določila (tu se prav tako preverja slovnično ujemanje s povedkom), predmete in prislovna določila.

```
var[] povedi = razcleniDatotekoKorpusaNaPovedi();

foreach(poved : povedi){

        if(poved.vsebujeGlagol()){
                poved.tip = Tip.glagolska;
                poved.poisciStavcneMeje();
```

```
                poved.analizirajStavke();

        }
        else{
                if(poved.vsebujeSamostalnik())
                        poved.tip = Tip.samostalniskaNeglagolska;
                else
                        poved.tip = Tip.nesamostalniskaNeglagolska;

                poved.analizirajNeglagPoved();
        }
}


class Poved{

var tip;
var[] besede;
var[] stavki;

        poisci_stavcne_meje(){
                var stavek = this.novStavek();

                foreach(beseda : besedePovedi){
                        if(beseda.jeNikalniClen()){
                                this.tip = Tip.glagolskaZanikana;
                                beseda.stavcniClen = Clen.povedek;
                        }
                        else if(beseda.jeGlagol()){
                                if(!stavek.preveriCeLahkoDelPovedka(beseda)){
                                        poisciStavcnoMejoOdDo(mejaNazadnjeDolocenegaStavka, beseda);

                                        if(stavcnaMejaNajdenaDoNovegaGlagola())
                                                stavek = this.novStavek();
                                        else //algoritem ne zna najti konca stavka pred novim glagolom
                                                stavek.odstraniPrejsnjiGlagol(); //verjetno gre za drugo b. vrsto
                                }
                                else{
                                        stavek.dodajGlagolVPovedek(beseda);
                                }
                        }
                }
                poisciStavcnoMejoOdDo(mejaPrejsnjegaStavka, zadnjaBesedaPovedi());
        }

        analizirajStavke(){

                foreach(stavek : stavkiZlozenePovedi){
                        stavek.poisiciSamostInPridBesedneZveze();
                        stavek.poisciPrislDol();
                }
        }

        poisiciSamostInPridBesedneZveze(){

                stavek.posiciOsebek();
```

```
                stavek.poisciPovDolocilo();

                if(!stavek.jeOsebekNajden()){
                        if(stavek.!jePovDolociloNajdeno() && stavek.vsebujeLeGlagolBitiObstajanja())
                                stavek.poisciNegiranOsebek();
                        else
                                stavek.poisciPosamostaljenOsebek();
                }

                if(!stavek.ceVStavkuLeGlagolBitiObstajanja()){
                        stavek.poisciPredmete();
                        stavek.poisciPosamostaljenjePredmete();
                }
        }
}


class Stavek{
        poisciOsebek(){
                for(i = 0; i < besedeStavka.length; i++){
                        beseda = besedeStavka.dobiBesedo(i);

                        if(beseda.jeZeAnalizirana() || beseda.moznostiOznacitve.length == 0)
                                continue;
                        else{
                                foreach(moznaOblikOznaka : beseda.dobiMozneOblikOznake()){
                                        if(moznaOblikOznaka.jeLahkoTipa(Tip.predlog)){
                                                i = poisciKonecPredlozneZveze();
                                                nadaljujZunanjoZanko();
                                        }
                                        else if(moznaOblikOznaka.jeLahkoTipa(Tip.osebek)){
                                                if(beseda.seUjemaZ(povedek, moznaOblikOznaka)){
                                                        beseda.tip = Tip.osebek;
                                                        najdiPrilastke(i, moznaOblikOznaka);
                                                        return;
                                                }
                                        }
                                }

                        }
                }
        }
        […]
}
```

Na podoben način kot metoda *poisciOsebek()* so realizirane tudi ostale metode za iskanje stavčnih členov (pri predmetih in prislovnih določilih se seveda ob prvem najdenem primeru metoda ne konča, kot se pri osebku in pri povedkovem določilu); natančna implementacija je v poglavju Izvorna koda. Metoda *analizirajNeglagPoved()* samo za vsako besedo predpostavi, da je prava oblika kar prva od možnih, stavkov ali naštevanj ne išče.

3.3. Časovna zahtevnost algoritma

Časovna zahtevnost algoritma je reda O(n) oz. natančneje f(x) = k * n = Θ(n), pri čemer je *n* število besed v povedi, *k* pa konstantno število iteracij pri analizi besed določene povedi[9]. Z naraščanjem števila besed in kompleksnosti povedi se število iteracij ne poveča.

3.4. Omejitve algoritma

Algoritem je zelo odvisen od oblikoslovnega označevalnika: če oblikoslovni označevalnik neke besede ne prepozna, npr. lastnega imena, je tudi algoritem ne more uspešno vključiti v analizo. Problem predstavljajo tudi vse možnosti za neko besedno obliko: pri nekaterih uporabljeni oblikoslovni označevalnik ne prepozna vseh možnosti, zaradi česar je analiza napačna (npr. poved »Absolutne garancije ni«). Poleg tega oblikoslovni označevalniki navadno prepoznajo le pravopisno in slovnično pravilno zapisane besede knjižnega jezika, algoritem pa je poleg tega razmeroma odvisen tudi od pravilno postavljenih ločil v besedilu – za raziskave korpusov neknjižnih besedil tako ni primeren. Algoritem slabo prepoznava tudi vrinjene neglagolske stavke, ne prepozna pa tudi naštevanj, saj jih v trobesednih glagolskih povedih skoraj ni. Pri nadaljnjem razvoju in uporabi algoritma za daljše povedi bi bilo tako nujno v algoritem vključiti dognanja Domna Marinčiča pri njegovih raziskavah strojnega razčlenjevanja besedila [9]. Algoritem nadalje zaradi neupoštevanja konteksta povedi ne more povsem natančno ločiti med osebkom in povedkovim določilom – predpostavi se, da je osebek prvi stavčni člen, povedkovo določilo pa mu sledi (kar pa ni vedno res zaradi prostega besednega reda v slovenščini, členitve po aktualnosti).

Algoritem za vsako poved določi le eno možnost analize oz. interpretacije, in sicer prvo, do katere pride (požrešno preiskovanje); to sicer ni večji problem za nadaljnji razvoj, saj bi se dalo formalno enako poved večkrat vnesti v bazo podatkov, vsakič z drugačno analizo oz. interpretacijo; več analiz pa bi lahko dobili z več iteracijami po različnih možnostih za označitev posameznih besed. Bi pa to občutno povečalo časovno zahtevnost algoritma. Na trenutni stopnji razvoja algoritma to še ni smiselno, saj je takih povedi, ki bi jih algoritem lahko uspešno analiziral na več možnih načinov, malo. Zaradi svoje formalne narave algoritem ne more biti uspešen tudi pri ločevanju med predmeti in prislovnimi določili – na trenutni stopnji razvoja se kot predmeti označijo vse nepredložne samostalniške (ali posamostaljene) besedne zveze v odvisnih sklonih, kot prislovna določila pa vse predložne. Za uspešnejše ločevanje med predmeti in prislovnimi določili bi bilo v nadaljnjem razvoju moč uporabiti podatke o vezljivosti glagolov iz elektronskega vezljivostnega slovarja Andreje Žele [8]. Manjši problem bi sicer predstavljali izpeljani glagoli, ki jih slovar ne navaja, vendar bi se tudi to dalo rešiti s sestavitvijo podalgoritma za identifikacijo izpeljanih glagolov.

[9] Notacija t. i. *velikega O* se na področju informatike in matematike uporablja za označevanje asimptotske tendence (hitrosti naraščanja) poljubne funkcije; na področju informatike gre navadno za funkcijo časovne ali prostorske zahtevnosti algoritma (porabe procesorskega časa ali pomnilnika). $g(x) = O(f)$ pomeni, da funkcija g asimptotsko gledano ne raste hitreje od f, $g(x) = \Theta(f)$ pa, da g in f asimptotsko gledano rasteta enako.

4. Razvoj in implementacija

4.1. Izbira jezikoslovnega modela za stavčno analizo

Prvo vprašanje, ki se je pojavilo pri izdelavi algoritma za stavčno analizo, je bila izbira jezikoslovnega modela oz. slovnice[10], na podlagi kateri bi algoritem analiziral poved; se pravi, kakšno vrsto stavčne analize izbrati. Prva možnost, ki je še najbližje matematičnemu pristopu oz. pristopu računalniške znanosti, je tvorbeno-pretvorbena slovnica, ki jo je uvedel Noam Chomsky[11]. Kljub privlačnosti in interdisciplinarnosti omenjenega pristopa sem se raje odločil za danes splošno uveljavljen strukturalistični pristop[12] oz. slovnico Jožeta Toporišiča [13], ki se poučuje tudi na osnovnih in srednjih šolah. Glavni razlog za izbiro je bil predvideni uporabnik algoritma za stavčno analizo, jezikoslovec, ki navadno preferira strukturalistično stavčno analizo. Tvorbeno-pretvorbena slovnica namreč v sodobnem slovenskem jezikoslovju, kljub določeni modnosti in navdušenju v polpreteklem obdobju, ni splošno sprejeta niti uporabljana in poučevana.

Toporišičeva strukturalistična skladnja, ki v glavnem zajema tudi vse bolj popularno odvisnostno skladnjo[13] Luciena Tesnièrja, temelji na stavčnih členih, ki so pomenske eno- ali večbesedne enote s tipično obliko in posebnim mestom v stavčni zgradbi [12]. Glavni stavčni členi so: osebek, povedek (med njima je prisojevalna relacija), predmet (nase ga veže povedek) in prislovno določilo (povedek ga ob sebi dopušča). Stavčni členi so besednozvezni (npr. *Govorec je lažnivec*) ali pa stavčni (*Kdor to govori, je lažnivec*). V algoritmu sem se osredotočil predvsem na prve, je pa predvidena kasnejša razširitev tudi na stavčne stavčne člene, to je na analizo večstavčne povedi. Tipe povedi sem prav tako povzel po Toporišiču, in sicer so to: : glagolska (vsebuje glagol), neglagolska samostalniška (ne vsebuje glagola, vsebuje pa vsaj en samostalnik) in neglagolska nesamostalniška poved (ne vsebuje ne glagola ne samostalnika).

4.2. Izbira programskega jezika in tehnologije

Na začetku sem imel na voljo v tekstni datoteki korpus z nekaj več kot 57.000 povedmi, dolžine do treh besed, iz dnevnika DELO za leto 2007 (vseh povedi je bilo 1,040.000). Enobesednih povedi je bilo 13, dvobesednih povedi 872, povedi dolžine treh besed pa 56275. Korpus povedi je pridobljen iz besedilnega korpusa Nova beseda. V drugi tekstni datoteki pa so z oblikoslovnim

[10] Slovnica je v širšem smislu množica pravil, ki ureja tvorbo besed, besednih zvez in stavkov. V ožjem smislu pa gre za jezikovni priročnik, ki opisuje omenjena pravila.

[11] Gre za tip tvorbene (generativne) slovnice, ki sestoji iz dveh delov: fraznega in pretvorbenega. Prvi tvori jedrne stavke, drugi pa jih preoblikuje v konkretne stavke danega jezika. Tvorbena slovnica je sicer jezikoslovna teorija, ki jezikovno sposobnost utemeljuje na prirojenih univerzalnih strukturah. Tvorbena slovnica sestoji iz določenega števila simbolov (imenovanega slovar) in določenega števila slovničnih pravil (navadno razumljenih kot zamenjavnih ali prepisnih/substitucijskih). [12]

[12] Strukturalno jezikoslovje obravnava jezik kot statični sistem med seboj povezanih enot, definiranih z nasprotji oz. razločevanji med njimi. Deloma je bilo jezikoslovje vedno strukturalno, vendar so do polnega razvoja strukturalistične jezikoslovne analize prispevali šele jezikoslovci 19. in 20. stoletja, kot sta De Saussure in de Courtenay. [12] V središče jezikoslovnega raziskovanja se postavlja jezik v istem času (sinhronija), se pravi v nekem odbobju, za razliko od diahronega pristopa, kot je primerjalno jezikoslovje, ki primerja različne razvojne stopnje jezika.

[13] Odvisnostna slovnica je slovnica, ki raziskuje predvsem odvisnostna razmerja v stavku [12]. Stavek se da hierarhično analizirati oz. prikazati v obliki drevesa ali grafa.

označevalnikom Primoža Jakopina označene pojavitve oblik besed iz tega korpusa [1]. Namen diplomskega dela je bilo stavčno analizirati povedi iz korpusa in rezultat shraniti, zatorej je bila uspešnost algoritma pomembnejša od hitrosti izvajanja. Pozneje bi bila zaželena tudi možnost analize povedi, vnesenih s strani uporabnika. Stavčni analizator naj bi bil javno dostopen, najbolje prek spleta, uporabnik bi do analiziranih besed dostopal prek filtrov, realiziranih z grafičnim vmesnikom. Iskanje po korpusu z neposrednim vnosom zmogljivih posebnih (regularnih) izrazov je za ciljnega uporabnika, ki ni računalničar kaj šele programer, precej težko, vsak iskalnik ima pogosto svojo sintakso, tudi sintaksa razmeroma univerzalnih regularnih izrazov iz okolij UNIX ni vedno lahka, še posebej za pričakovane kompleksnejše poizvedbe po analiziranem korpusu. Enostavnejše poizvedbe, za katere lahko uporabimo nadomestne znake (npr. znaka '*' ali '?'), ki jih uporabniki večinoma dobro poznajo, pa niso dovolj zmogljive za želene kombinacije iskalnih pogojev.

Na podlagi vseh zahtev sem se odločil za programsko tehnologijo Java, saj je odlično dokumentirana, prosto dostopna, vključno z dvema dobrima razvojnima okoljema[14], omogoča prenosljivost med operacijskimi sistemi[15], je objektno orientirana[16], vključuje tudi dobri programski knjižnici za delo z grafičnim vmesnikom (AWT in Swing), knjižnice za povezavo z bazami podatkov in zmogljivo implementacijo regularnih izrazov sistemov UNIX; je razmeroma hitra (dinamično prevajanje omogoča kombinirano interpretativno-prevedeno izvajanje), poleg tega pa je enostavna za uporabo na spletu v obliki posebnih spletnih programov, apletov, ki tečejo na strani odjemalca, tako da pisanje posebne spletne strani ni potrebno, poenostavi pa tudi morebitno kasnejše dodajanje vmesnika za sprotno analizo od uporabnika vnesenih povedi – analiza lahko tako poteka na odjemalčevem računalniku brez obremenjevanja in čakanja na strežnik. Java je sicer počasnejša od programskega jezika C/C++, nekoliko tudi od konkurenčnih C#/VB.NET, ki sicer prav tako tečeta na podobnem navideznem stroju kot Java (CLR), Java zasede tudi več pomnilniškega prostora, kar pa je pa je danes vse manjši problem. Menim, da vse omenjene slabosti odtehta javanska prenosljivost, odprtokodnost in razširjenost njene uporabe.

Glede hrambe analiziranih podatkov sem se odločal med tremi možnostmi: shranjevanje v binarni ali tekstovni datoteki, shranjevanje v XML-datoteki in shranjevanje v bazi podatkov. Prva možnost je razmeroma počasna za iskanje, pohitritev oz. preoblikovanje v bazi podatkov podobno obliko bi zahtevalo preveč dela, enako iskanje. Druga možnost, XML, je danim podatkom (tj. analiziranim povedim) najprimernejša, saj omogoča preprost vnos hierarhije po vozliščih XML-drevesa in ne potrebuje dodatnega programa oziroma sistema za upravljanje baze, je pa razmeroma počasna pri iskanju – podatki se hranijo v tekstovni, in ne binarni obliki – ter brez dodatne kompresije zasede precej več prostora na disku. Tako sem se raje odločil za podatkovno bazo SQL, in sicer za sistem za upravljanje podatkovnih baz MySQL, ki je prosto dostopen, razmeroma hiter in dobro dokumentiran. Hierarhija je realizirana prek tujih ključev s t. i. relacijami »eden-do-mnogih« (angl. one-to-many relation).

[14] Netbeans (http://www.netbeans.org/) in Eclipse (http://www.eclipse.org/).
[15] Javanski programi ne tečejo neposredno na operacijskem sistemu uporabnika, temveč na posebnem navideznem (programsko realiziranem) stroju.
[16] Objektno orientiran pristop je danes najpogostejši pristop pri programiranju – nekoliko večjo porabo pomnilnika odtehta lažja berljivost izvorne kode, posledično lažje vzdrževanje programov ter večja podobnost programov oz. programskih objektov z objekti realnega sveta, ki ga program modelira.

Odločil sem za uporabo angleških imen spremenljivk, delov programa in komentarjev, saj je to običajna praksa profesionalnega programiranja – tako kodo je možno enostavneje izmenjati s programerji v tujini, enostavneje je poslati del izvorne kode programa za morebitno tehnično pomoč pri programiranju: programer se namreč z nerazumljivimi imeni spremenljivk precej težje poglobi v tujo kodo; za uporabnika pa to ne predstavlja nobene težave: obvestila programa in grafični vmesnik so v slovenščini.

4.3. Zgradba programa

Program lahko razdelimo na štiri glavne dele:

1. del za analizo povedi:
    - **Category.java** – razred, namenjen za tvorbo objekta, ki vsebuje podatke o slovničnih kategorijah posamezne oblike besede (obliko besede predstavlja razred Lemma, gl. nadaljevanje); razred se zaradi poenostavitve pri programiranju ne deduje, vsi objekti besednih vrst imajo skupne kategorije, neobstoječe za posamezno besedno vrsto imajo vrednost »null« (v obliki za bazo MySQL); vsebuje tudi konstante tipov povedi, stavkov in stavčnih členov ter vzorce regularnih izrazov za iskanje po besednih vrstah in kategorijah;
    - **CmpSentence.java** – razred, namenjen za analizo zloženih povedi: vsaka poved vsebuje več objektov Element (stavkov) in seznam vseh besed povedi (objekti Word);
    - **Corpus.java** – razred, namenjen za razčlenitev korpusa iz tekstovne datoteke: vsebuje objekt Dictionary;
    - **Dictionary.java** – razred, namenjen za razčlenitev slovarja iz tekstovne datoteke: vsebuje razpršeno tabelo, v kateri je shranjen slovar vseh oblik besed (ključi so oblike besed, vrednosti pa seznami objektov Lemma, tj. vse možnosti analize posamezne oblike);
    - **Element.java** – razred, namenjen za tvorbo objektov podenot znotraj povedi (stavki), znotraj stavkov (stavčni členi) in znotraj stavčnih členov (deli stavčnih členov); vsebuje niza za označitev tipa in podtipa ter razpršeno tabelo podenot;
    - **Lemma.java** – razred, namenjen za tvorbo objektov, ki vsebujejo podatke o kategorijah in osnovni obliki[17] posamezne oblike besede; vsaka instanca (objekt) predstavlja eno možnost analize posamezne oblike;
    - **Word.java** – razred, namenjen za označitev posamezne pojavitve besede: vsebuje niza označitev tipa besede in tipa stavčnega člena, ki mu beseda pripada;
2. grafični uporabniški vmesnik:
    - **GUIApplet.java** – razred, ki vsebuje grafični vmesnik (panel) za spletno aplikacijo – aplet;

[17] Osnovna ali nevtralana besedna oblika je za samostalniško besedo imenovalnik ednine, za pridevniško imenovalnik moškega spola ednine (osnovna stopnja), za glagole nedoločnik, za prislove pa osnovna stopnja pri stopnjevanju.

- **GUIApplication.java** – razred, ki vsebuje grafični vmesnik (panel) za namizno aplikacijo;
- **SAApplet.java** – razred za zagon apleta;
- **SAApplication.java** – razred za zagon namizne aplikacije;

3. vmesnik za shranjevanje podatkov v bazo in poizvedbe:
   - **DBConnection.java** – razred, ki vsebuje in sestavlja SQL-stavke za poizvedbe in shranjevanje podatkov v bazo;
   - **DBRequest.java** – razred za prenos podatkov, ki jih posreduje aplet servletu, da ta opravi poizvedbo v bazi;
   - **DBServletConnection.java** – razred za komunikacijo med apletom in servletom;
   - **SAServlet.java** – razred za strežniško aplikacijo – servlet (potreben za aplet), ki komunicira z bazo (prek DBConnection.java).
4. tabele v bazi podatkov MySQL:
   - **cmpsentences** – tabela zloženih povedi;
   - **dictionary** – tabela možnih kombinacij kategorij;
   - **sentences** – tabela vseh stavkov – podenot zložene povedi;
   - **words** – tabela vseh besed: vsaka beseda predstavlja svojo vrstico; z ostalimi tabelami je povezana s tujimi ključi: cmpSentID (iz tabele *cmpsentences*), sentID (iz tabele *sentence*) in dictID (iz tabele *dictionary*).

4.4. Opis poteka programa

Ob zagonu samostojne aplikacije se najprej v pomnilniku ustvari slovar besednih oblik, implementiran je z razpršeno tabelo (Hashtable) za učinkovito iskanje. Program ob zagonu prikaže potrditveni dialog, ki uporabnika vpraša, ali želi uporabiti lastni morfološki označevalnik ali ne. Če pritrdi, se iz diska (serializirani objekti) naloži izvajalni algoritem morfološkega označevalnika (opisan v poglavju 0) in Amebisov slovar oz. tezaver, ki se uporablja z njim v kombinaciji. Če zavrne, pa se naloži prvotni slovar oblikoslovno označenih besed, narejen z označevalnikom Primoža Jakopina oz. Inštituta Frana Ramovša ZRC SAZU (gl. [1], [14]). Ključ v tabeli slovarja je niz – besedna oblika, vrednost pa seznam objektov Lemma – vse možnosti analize posamezne oblike. Po deserializaciji slovarja se začne gradnja korpusa v pomnilniku. Zaradi precejšne obsežnosti podatkov se v pomnilniku ne hrani podatkov za vse povedi – hrani se le podatke za posamezno poved, ki se v nekem trenutku analizira oz. shranjuje v bazo podatkov. Povedi se razčlenjuje iz datoteke »korpus.txt«, za vsako besedo posamezne povedi se konstruira objekt Word; ti se hranijo v seznamu tipa ArrayList. Za vsako besedo se v slovarju oblik poišče ustrezne možnosti, na tej stopnji pa program še ne ve, katere bi lahko bile prave.

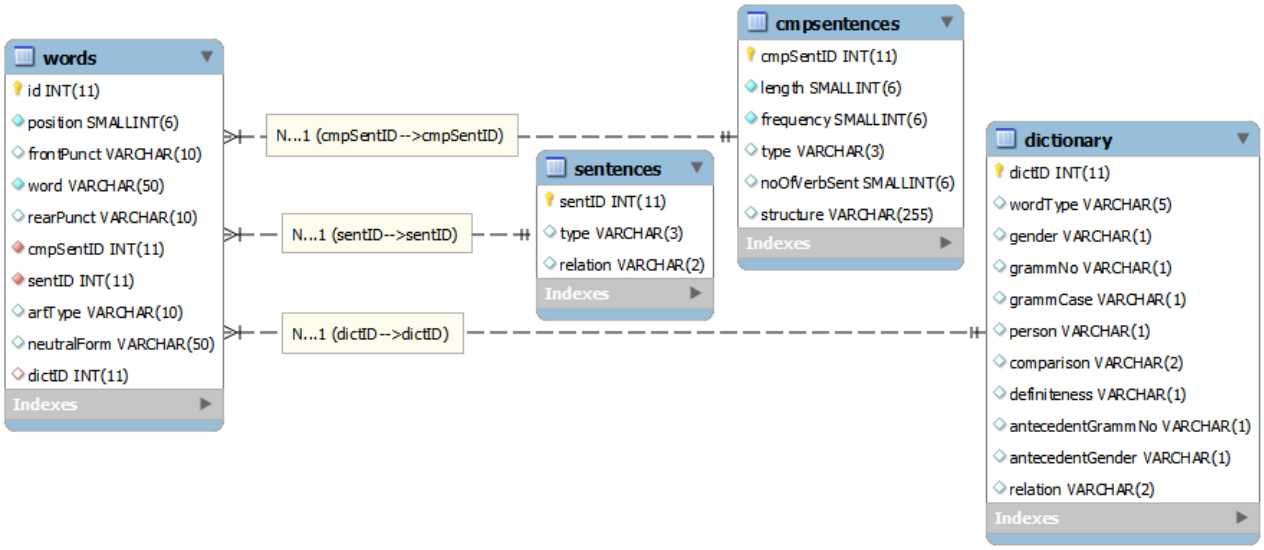
Vsako poved se analizira sproti (podroben opis algoritma za stavčno analizo je podan v naslednjem poglavju). Po analizi posamezne povedi in enoznačni določitvi besednih vrst in stavčnih členov besed se poved zapiše v bazo. Zapis v bazo je realiziran prek zapisa v tekstovno datoteko, ki po koncu analize vseh povedi zapiše oz. prevede v bazo podatkov MySQL (MySQL-stavek »LOAD DATA INFILE«). Ta način je precej hitrejši od sprotnega shranjevanja vsake povedi posebej [7] – razlog za zahtevo po večji hitrosti shranjevanja je v številnih testiranjih in preverjanjih algoritma. Pred zapisom analiziranih povedi v bazo se stare podatke iz baze izbriše.

Po koncu analize se odpre grafični vmesnik, v katerem lahko uporabnik izbere želene kriterije za iskanje in izpis analiziranih povedi (podrobnejši opis je v 4. poglavju). Pri zagonu apleta se na začetku povedi ne analizirajo ponovno, naloži se le slovar (zgolj v informacijo uporabniku, katere besedne oblike program sploh prepozna), grafični vmesnik pa je enak. Ko uporabnik klikne na gumb »išči«, se pošlje poizvedba v bazo, ki nato vrne ustrezne vrstice iz tabele *words* in ostalih s stiki povezanih tabel (v obliki povezanega seznama ResultSet) – pri samostojni aplikaciji se to izvrši neposredno, pri apletu pa se najprej serializira tabela kriterijev, natančneje podatkovni seznam (tipa Vector) v modelu tabele, ki se nato pošlje strežniški aplikaciji – servletu, ta pa pošlje poizvedbo bazi. Vrnjen povezani seznam zadetkov (ResultSet) servlet prepiše v model za grafično kontrolo Jlist in ga serializiranega pošlje apletu. Razlog za posredniško aplikacijo – servlet – je v varnosti, saj je zaradi številnih vdorov v baze podatkov praksa sistemskih administratorjev, da vrat za dostop do baze podatkov ne odpirajo navzven, tako da neposreden dostop apleta do baze prek spleta ni možen.

Vrstice v seznamu povedi so sestavljene iz posameznih vrstic tabele *words* – ob kliku na posamezno vrstico se sproži iskanje v bazi glede na identifikacijsko številko povedi: program po kliku prikaže podrobnosti o posamezni povedi. Stavčni členi so v seznamu povedi označeni z več vrstami oklepajev: najprej sem nameraval uporabiti HTML-jevsko označevanje z barvami, vendar je razčlenjevanje oznak povsem prepočasno za normalno delo s korpusom povedi takega obsega. Da se izpisi še pospešijo, sem uporabil tudi konkatenacijo oklepajev in nizov besed v bazi podatkov (v stavkih *SELECT*), javanska je bila kljub uporabi StringBufferjev počasnejša (problem je verjetno veliko število samostojnih nizov).

4.5. Hramba analiziranih povedi v bazi podatkov

Podatki analiziranih povedi so shranjeni v štirih tabelah: *cmpsentences*, *sentences*, *words* in *dictionary* (gl. Slika 1: ER-shema podatkovne baze). Vsaka beseda predstavlja svojo vrstico v tabeli *words*, besede določene povedi so povezane s tujim ključem iz tabele *cmpsentences*. V tabeli ***cmpsentences*** so naslednji atributi (stolpci): **cmpSentID** – identifikacijska številka posamezne povedi; **length** – dolžina posamezne besede (v veliki večini primerov je to 3, obstaja nekaj izjem dolžine 2 in 1 zaradi napak pri izbiranju povedi iz korpusa oziroma napačne členitve nizov pri ločilih); **frequency** – frekvenca posamezne povedi, podatek je pridobljen iz korpusa Nova beseda; **type** – tip povedi, možni so trije: glagolska (G), neglagolska samostalniška (nG) in



Slika 1: ER-shema podatkovne baze

neglagolska nesamostalniška (nnG); **noOfVerbSent** – število glagolskih stavkov v povedi; **structure** – struktura povedi (atribut na trenutni stopnji razvoja algoritma še ni v uporabi).

V tabeli ***sentences*** so atributi: **sentID** – identifikacijska številka posameznega stavka; **type** – vrsta stavka, možnosti so: glagolski (G), samostalniški pastavek (PS), drugi pastavki (PO); če je stavek zanikan, ima na koncu znak '-', npr. 'PS-'; **relation** – razmerje do glavnega stavka: priredno, podredno ali soredno (atribut na trenutni stopnji razvoja algoritma še ni v uporabi).

V tabeli ***words*** so atributi (gl. Slika 2: Zapis povedi v bazi podatkov MySQL (tabela *words*)): **id –** identifikacijska številka besede, **position** – vrstni red besede v stavku; **frontPunct** – ločila pred besedo (npr. narekovaj), **word** – beseda; **rearPunct** – ločila za besedo (npr. vejica, pika); **cmpSentID** – tuji ključ povedi (iz tabele *cmpsentences*), v katero spada beseda; ***sentID*** – tuji ključ stavka (iz tabele *sentences*), v katerega spada beseda; a**rtType** – vrsta stavčnega člena; **neutralForm** – osnovna besedna oblika; **dictID** – tuji ključ kategorij besede oz. besedne oblike (iz tabele *dictionary*).

	id	position	frontPunct	word	rearPunct	cmpSentID	sentID	artType	neutralForm	dictID
	31	0		Abotna		11	11	S	aboten	35
	32	1		zmaga		11	11	P	zmagati	79
	33	2		denarja	.	11	11	O	denar	14
	34	0		Abramovič		12	12	*NULL*	*NULL*	*NULL*
	35	1		kupuje		12	12	P	kupovati	79
	36	2		Beckhama	?	12	12	*NULL*	*NULL*	*NULL*
	37	0		Absolutiziranje		13	13	S	absolutiziranje	41
	38	1		rojeva		13	13	P	rojevati	79
	39	2		malike	.	13	13	O	malik	71
	40	0		Absolutne		14	14	S	absoluten	19
	41	1		garancije		14	14	*NULL*	garancija	7
	42	2		ni	.	14	14	P	ne biti o	966
	43	0		Absolutne		15	15	S	absoluten	19
	44	1		varnosti		15	15	S	varnost	5
	45	2		ni	.	15	15	P	ne biti o	966

Slika 2: Zapis povedi v bazi podatkov MySQL (tabela *words*)

V tabeli ***dictionary*** (gl. Slika 3**: Zapis možnih morfoloških oznak v bazi podatkov MySQL (tabela *dictionary*)**) pa so atributi: **dictID** – identifikacijska številka posamezne kombinacije kategorij; **wordType** – besedna vrsta; **gender –** spol; **grammNo** – število; **grammCase** – sklon; **person** – oseba; **comparison** –primerjalna stopnja; **definiteness –** določnost; **antecedentGrammNo** – število reference zaimka; **antecedentGender** – spol reference zaimka; **relation** – razmerje predloga.

	dictID	wordType	gender	grammNo	grammCase	person	comparison	definiteness	antecedentGrammNo	antecedentGender	relation
☐ ✎ ✕	1208	ZO	m	e	1	c	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1209	ZK	ž	e	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1210	ZO	ž	e	1	c	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1211	ZO	ž	e	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1219	PL	m	e	6	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1220	ZK	m	e	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1221	ZK	m	e	4	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1224	ZO	m	e	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1248	PČ	s	e	6	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1282	PČ	m	e	6	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1435	ZSVP	m	e	6	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1482	ZO	*NULL*	e	1	b	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1490	ŠM	m	e	4	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
☐ ✎ ✕	1491	ZSV	m	e	1	b	*NULL*	*NULL*	e	*NULL*	*NULL*
☐ ✎ ✕	1492	ZSV	m	e	4	b	*NULL*	*NULL*	e	*NULL*	*NULL*

Slika 3: Zapis možnih morfoloških oznak v bazi podatkov MySQL (tabela *dictionary*)

5. Primeri uporabe algoritma za stavčno analizo

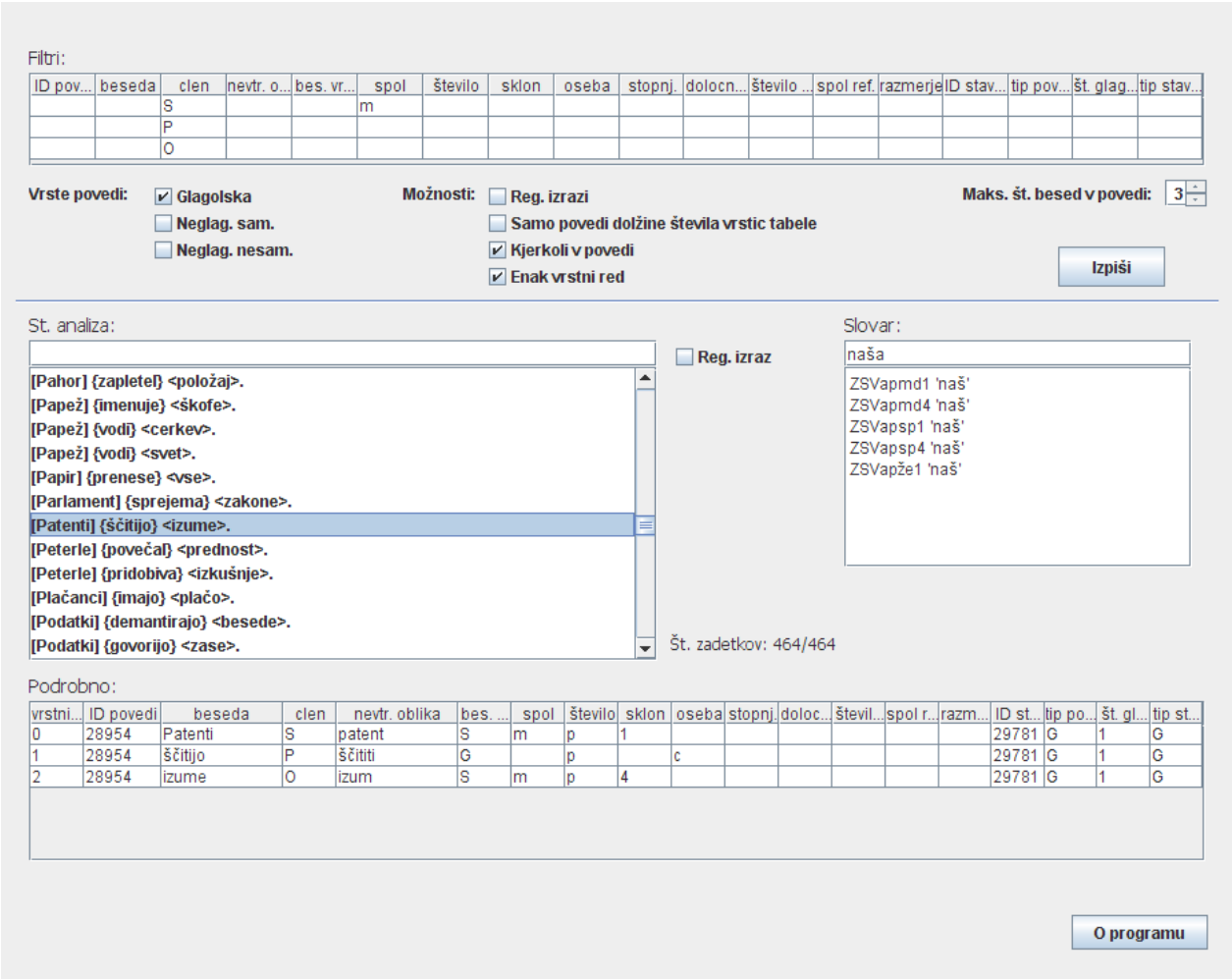
5.1. Opis programa za pregled analiziranih povedi

Uporabnik ob zagonu apleta ali samostojne aplikacije zagleda grafični vmesnik s Slike 3. V zgornjem delu vmesnika vnese želene filtre za iskanje po bazi analiziranih povedi. Program omogoča dva načina iskanja: med njima preklapljamo z označitvijo ali neoznačitvijo določila »Kjerkoli v povedi«. Pri prvem načinu, tj. z neoznačeno možnostjo »Kjerkoli v povedi«, vsaka vrstica filtrirne tabele predstavlja eno besedo (od treh) v povedi; prva vrstica predstavlja prvo besedo povedi, druga drugo itd. Če je celica oz. vrstica prazna, se to obravnava kot karkoli – kot nadomestni znak. Če ni izbrana možnost »Samo povedi dolžine števila vrstic tabele« in če je poved krajša od vnesenih filtrov v vrstice, program preverja ujemanje le s toliko začetnimi vrsticami, kot ima poved besed – izpiše torej tudi povedi krajše od števila vrstic tabele. Pri drugem načinu iskanju, tj. z neoznačeno možnostjo »Kjerkoli v povedi«, vsaka vrstica predstavlja kriterij za eno besedo povedi – njeno mesto v povedi ni določeno, lahko pa določimo relativni vrstni red kriterijev v filtru z označitvijo možnosti »Enak vrstni red« – program v tem primeru izbere le povedi, v katerih se pojavljajo besede v enakem vrstnem redu, kot so v vneseni filtrirni tabeli – med besedami v filtrirni tabeli pa se lahko v izpisanih povedih vrinejo tudi druge, ohranja se vrstni red vrstic. Za iskanje ni potrebno izpolniti vseh vrstic filtrirne tabele, prazne vrstice pa so lahko le za izpolnjenimi vrsticami, ne med njimi, kajti program upošteva le izpolnjene vrstice do prve prazne. Število vrstic filtrirne tabele še vedno določa število besed v povedih, po katerih iščemo. Z možnostjo »Samo povedi dolžine števila vrstic tabele« določimo, ali je to število največje število besed povedi – program potem izpisuje krajše ali enako dolge povedi – ali pa točno število besed povedi – program potem izpisuje le natanko tako dolge povedi.

Možna sta dva podnačina iskanja oz. vnosa filtrov: navadno iskanje (brez regularnih izrazov v celicah) in z regularnimi izrazi – uporabnik možnost izbere z ustrezno označitvjo okenca »Reg. izrazi«. Regularni izrazi so formalni izrazi za učinkovito identifikacijo nizov; poenostavljeno: gre za kompleksnejši sistem nadomestnih znakov. Uporabljeni so regularni izrazi sistema MySQL – v primerjavi z regularnimi izrazi sistema UNIX in Jave so nekoliko okrnjeni (dokumentacija: http://dev.mysql.com/doc/refman/5.1/en/regexp.html; http://www.regular-expressions.info/). Primer: iskanje določenega zaimka v vseh šestih sklonih in v vseh treh osebah dobimo z vpisom regularnega izraza '[1-6]' ali '[123456]' v stolpec *sklon* in izraza '[a-c]' ali '[abc]' v stolpec *oseba*. Iskanje dveh različnih besed hkrati, npr. besed *hiša* in *stanovanje*, pa dobimo z vpisom 'hiša|stanovanje' v stolpec *nevtr. oblika*. Nadomestna znaka sta:

- '*' – nadomešča poljubno število znakov, tudi prazen niz (pri navadnem iskanju je to znak '%');
- '.' – nadomešča natanko en znak (pri navadnem iskanju je to znak '_').

Pod tabelo filtrov uporabnik izbere želene tipe povedi – glagolska poved je vsaka poved, ki vsebuje glagol, neglagolska samostalniška pa vsaka, ki vsebuje vsaj en samostalnik in nobenega glagola. Uporabnik sproži iskanje s klikom na gumb **Izpiši**.



Slika 4: Program/aplet za iskanje po bazi analiziranih povedi

V osrednjem delu okna je seznam z analiziranimi povedmi – tekstovno polje nad seznamom omogoča vnos filtrov za sprotno filtriranje seznama zadetkov v pomnilniku. Iskalnik ne ločuje med malimi in velikimi črkami, možen pa je tudi vnos javanskih regularnih izrazov, pri čemer je potrebno paziti na znake '[]{}()', ki so za regularne izraze rezervirani znaki – pri iskanju z regularnimi izrazi jih je potrebno predznačiti z ubežnim znakom '\'. Iskanje z regularnimi izrazi se izbere z označitvijo okenca »Reg. izraz« poleg vnosnega polja. Iskanje uporabnik sproži s pritiskom na tipko **Enter**. V seznamu povedi so naštete povedi, ki ustrezajo iskalnim kriterijem v razdelku filtri in iskalnim kriterijem za iskanje v pomnilniku. Stavčni členi so označeni z ustreznimi oklepaji: '{}' predstavlja povedek', '[]' osebek, '<>' predmet, '//' prisl. določilo, '{{}}' pov. določilo, '|' pa ločuje glagolske stavke. S klikom na posamezno poved se na dnu ekrana v tabeli izpišuje atributi vseh besed. Možno je tudi izbiranje več povedi s pritiskom na tipko Ctrl in kliki. Na desni strani osrednjega dela je tudi za analizo povedi služivši[18] slovar – namenjen je zgolj za preverjanje uspešnosti algoritma in vsebuje le besedne oblike, ki so v korpusu.

[18] Raba deležij na -ši odraža željo po ohranitvi izredno učinkovite izrazne možnosti jezika, ki žal izginja. Omenjena deležja so, neupoštevajoč kvalifikatorje SSKJ oz. pravopisa, rabljena kot slogovno nezaznamovana.

5.2. Oblikoslovne in skladenjske oznake

Uporabljene oblikoslovne oznake so enake kot v oblikoslovnem označevalniku Primoža Jakopina [1] – opisane so v podpoglavju 2.5, natančneje pa na podstrani Razlaga oblikoslovnih oznak vira [4]. Na novo uvedene skladenjske oznake za stavčne člene pa so:

- *P* – povedek;
- *S* – osebek;
- *O* – predmet;
- *A* – prislovno določilo;
- *PD* – povedkovo določilo.

Uvedene skladenjske oznake za tipe povedi so:

- *G* – glagolska poved, tj. poved, ki vsebuje vsaj en glagolski stavek;
- *nG* – neglagolska samostalniška poved, tj. poved, ki ne vsebuje nobenega glagolskega stavka, vsebuje pa vsaj en samostalnik;
- *nnG* – neglagolska nesamostalniška poved, tj. poved, ki ne vsebuje nobenega glagolskega stavka niti nobenega samostalnika.

Uvedene skladenjske oznake za tipe stavkov so:

- *G* – glagolski stavek;
- *PS* – samostalniški pastavek;
- *PO* – nesamostalniški pastavek;

Če je stavek zanikan, ima na koncu dodan znak '-' (zaenkrat algoritem zanikane stavke prepoznava le pri glagolskih povedih).

5.3. Primeri iskanja

Navajam nekaj možnih primerov iskanja. Korpus analiziranih povedi obsega, kot že omenjeno, povedi do vključno dolžine treh besed. Primeri:

- Vse zanikane povedi tipa *osebek + vez + povedkovo določilo* izpišemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vstico filtrirne tabele v stolpec *clen* vpišemo *S*, v drugo vrstico v isti stolpec *P*, v tretjo vrstico v isti stolpec pa *PD*. Stolpec *tip stavka* pa izpolnimo z *G-*. V filtru Možnosti označimo »Kjerkoli v povedi« in poženemo iskanje s pritiskom na gumb **Izpiši**:
  - Število zadetkov: 371;
  - Prvih deset najdenih povedi:
    - [Argument] {ni} {{slab}}.
    - [Ateist] {ni} {{nevernik}}?
    - [Bilanca] {ni} {{spodbudna}}.
    - [Bojazen] {ni} {{neutemeljena}}.
    - [Center] {ni} {{mrtev}}.
    - [Cerkev] {ni} {{nadškof}}.
    - [Čakanje] {ni} {{modro}}.
    - {{Čisto}} [tako] {ni}.
    - [Človek] {ni} {{Bog}}.
    - [Človek] {ni} {{most}}.

- Vse povedi, ki vsebujejo osebek, povedek in predmet, izpišemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vstico filtrirne tabele v stolpec *člen* vpišemo *S*, v drugo vrstico v isti stolpec *P*, v tretjo vrstico v isti stolpec pa *O*. V filtru Možnosti označimo »Kjerkoli v povedi« in nadaljujemo iskanje z gumbom **Izpiši**;
    - Število zadetkov: 3243;
    - Prvih deset najdenih povedi:
        - [Abotna] {zmaga}.
        - [Absolutiziranje] {rojeva}.
        - [Aeroplani] {so}.
        - [Ajda] {pobira}.
        - <Akcijo> {jemlje} [osebno].
        - [Alah] {nagrajuje}.
        - [Albanci] {slavijo}.
        - [Albin] {išče}.
        - [Amater] {ovadil}.
        - [Američani] {sledijo}.
- Vse povedi z zanikanim osebkom v rodilniku poiščemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vrstico filtrirne tabele v stolpec *clen* vpišemo *S*, v stolpec *sklon* pa vpišemo *2* ter v filtru Možnosti označimo »Kjerkoli v povedi« (možnost »Enak vrstni red« naj ne bo označena) in pritisnemo na gumb Izpiši;
    - Število zadetkov: 1390;
    - Prvih deset najdenih povedi:
        - [Absolutne] garancije {ni}.
        - [Absolutne] [varnosti] {ni}.
        - [Absolutnih] [pogojev] {ni}.
        - [Absolutnih] [resnic] {ni}.
        - Ali {ni} [skeptičnosti]?
        - Ampak {nimamo} [časa]!
        - Ampak {ni} [problema].
        - Ampak {ni} [sile]!
        - Ampak [saj] {niso}!
        - [Avta] {niso} {ukradli} ...
- Vse zanikane povedi s prehodom tožilniškega predmeta v rodilnik izpišemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vrstico filtrirne tabele v stolpec *člen* vpišemo *O*, v stolpec *sklon* vnesemo *2*, v stolpec *tip stavka* pa *G-*. V filtru Možnosti označimo »Kjerkoli v povedi« in nadaljujemo z **Izpiši**;
    - Število zadetkov: 396;
    - Prvih deset najdenih povedi:
        - <Agresije> {ni} {čutiti}.
        - <Alkohola> {ne} {strežemo}.
        - <Ambicij> {ne} {skrivam}.
        - <Avtorizacije> {ne} {zahteva}.
        - <Barabij> {ne} {zagovarjam}.
        - <Brezbrižnosti> {ne} {manjka}.
        - <Brezdelja> {ne} {prenaša}.
        - <Česar> {ne} {skriva}.

        - <Česa> {ne} {moremo}?
        - <Česa> {ne} {smem}?

- Program lahko uporabimo tudi za preučevanje vezljivosti glagolov. Vse vezave glagola *delati* z dajalnikom in tožilnikom izpišemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vrstico filtrirne tabele v stolpec *nevtr. oblika* vpišemo *delati*, v naslednjo vrstico vpišemo v stolpec *sklon* številko *3*, v zadnjo vrstico pa vpišemo *4.* V filtru Možnosti označimo »Kjerkoli v povedi« (možnost »Enak vrstni red« naj ne bo označena) in gremo naprej z gumbom **Izpiši**;
    - Število zadetkov: 4;
    - Najdene povedi:
        - {Delam} <si> <zapiske>.
        - <Kaj> {delajo} <njegovi>?
        - <Komu> {delajo} <uslugo>?
        - <Priznanji> <časnikarjema> {Dela}.

- Program je zelo prikladen tudi za iskanje besednih zvez oziroma frazemov. Vse trdilne in nikalne pojavitve besedne zveze *imeti rad* poiščemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vrstico filtrirne tabele v stolpec *nevtr. oblika* vpišemo *%imeti,* v naslednjo vrstico v isti stolpec pa *rad,* v filtru Možnosti pa označimo »Kjerkoli v povedi« (možnost »Enak vrstni red« naj ne bo označena) in - **Izpiši**;
    - Število zadetkov: 59;
    - Prvih deset najdenih povedi:
        - <Ameriko> {imam} rad.
        - <Časopise> {ima} rad.
        - {Imam} rad <pse>.
        - {Imeti} {se} rad.
        - Irak {imam} rad.
        - <Jih> {imaš} rad?
        - {Nimam} rad [primerjav].
        - <Oba> {imam} rad.
        - <Oboje> {imam} rad.

- <Polemike> {imam} rad. Vse pojavitve frazema *dobiti jih* izpišemo tako, da v filtru Vrste povedi označimo možnost »Glagolska«, v prvo vrstico filtrirne tabele v stolpec *nevtr. oblika* vpišemo *dobiti,* v naslednjo vrstico v isti stolpec pa *ona*, v stolpec število vpišemo *p,* v stolpec *sklon* pa vnesemo *2.* V filtru Možnosti označimo »Kjerkoli v povedi« (možnost »Enak vrstni red« naj ne bo označena) in poženemo iskanje s pritiskom na gumb **Izpiši**.
    - Število zadetkov: 4;
    - Najdene povedi:
        - {Dobili} {so} .
        - {Dobite} /takoj/!
        - Kje {dobiti}?
        - {Ni} [jih] {dobil}.

6. Sklepne ugotovitve

Cilj diplomskega dela, razviti učinkovit algoritem za analizo kratke povedi, je bil z razvojem algoritmov za morfološko označevanje in stavčno analizo dosežen. Algoritma sicer ne dosegata natančnosti, ki je možna pri analitičnih jezikih, vendar sta kljub temu soliden dosežek za sintetičen jezik z veliko morfologije in prostim besednim redom, kot je slovenščina. Pomembna je tudi eventualna uporabnost za druge jezike: morfološki označevalnik načeloma za večino sintetičnih oz. inflektivnih jezikov, stavčni analizator pa predvsem za slovanske jezike, z morebitnimi manjšimi prilagoditvami. Algoritem za morfološko označevanje je uporaben predvsem, ko enoznačna označitev besedne oblike ni potrebna, stavčni analizator pa predvsem za potrebe označevanje korpusa z namenom izboljšati iskanje po tipih povedi glede na vsebovane stavčne člene, lahko pa se uporabi tudi za nedvoumno morfološko označitev korpusa. Še posebno pomembni sta ugodni časovni zahtevnosti (linearni), saj je sicer skladenjska analiza jezika v splošnem problem eksponentne časovne zahtevnosti. Z vidika predvidenega končnega uporabnika, jezikoslovca, je še posebej pomemben uspešen razvoj razmeroma enostavnega grafičnega uporabniškega vmesnika (namizna aplikacija in aplet), ki omogoča kompleksna iskanja po bazi analiziranih povedi.

Algoritem za stavčno analizo se zdi obetavna novost na področju računalniškega jezikoslovja, saj poved analizira s stališča humanistične slovenistične stroke, na podlagi strukturalistične in odvisnostne slovnice, ne pa na podlagi na jezikoslovnem področju manj uveljavljenih formalnejših slovnic, kot je tvorbeno-pretvorbena N. Chomskega. Uporabljena slovnica J. Toporišiča za analizo je splošno znana in uveljavljena, poučuje se že v osnovnih šolah v Sloveniji, kar naredi rezultate analize algoritma na omenjenem področju splošno razumljive.

Razvoj algoritma za stavčno analizo kljub začetnim težavam in pomanjkljivostim kaže v pravo smer, vsekakor pa bi bilo potrebno pri nadaljnjem delu algoritem razširiti tudi za analizo povedi, daljših od treh besed. V ta namen se zdi posebno obetavna vključitev vezljivostnega slovarja, ki bi omogočila učinkovito ločevanje med predmeti in prislovnimi določili, hkrati pa bi v algoritem dodala tudi pomensko občutljivost, ki je sedaj ni. Pri nadaljnjem razvoju in uporabi algoritma za daljše povedi bi bilo tudi nujno vključiti dognanja in izkušnje Domna Marinčiča pri strojnem razčlenjevanju besedila [9]. Ozko grlo algoritma oz. tudi oblikoslovnega označevanja pa ostaja zahteva po pravopisno in slovnično pravilnem zapisu besedila, kar zožuje njegovo uporabnost na predvsem na področje zbornega knjižnega jezika. To pomanjkljivost sicer v precejšnji meri odpravlja razviti oblikoslovni označevalnik, vendar pa brez slovarskega dela zaenkrat ni dovolj specifičen oz. je razviti stavčni analizator premalo robusten.

7. Viri

[1] (2011) Primož Jakopin, *Oblikoslovno označevanje*. Dostopno na: http://bos.zrc-sazu.si/oblikoslovno_oznacevanje.html

[2] (2012) POS-tagging Error Rates. Dostopno na: http://ucrel.lancs.ac.uk/bnc2/bnc2error.htm

[3] (2012) Primož Jakopin, A. Bizjak. *Razlaga oblikoslovnih oznak*. Dostopno na: http://bos.zrc-sazu.si/bibliografija/o_oznake.html

[4] (2012) All Our N-gram are Belong to You. Dostopno na: http://googleresearch.blogspot.ch/2006/08/all-our-n-gram-are-belong-to-you.html

[5] (2012) Is Part-of-Speech Tagging a Solved Task? An Evaluation of POS Taggers for the German Web as Corpus. Dostopno na: http://cogsci.uni-osnabrueck.de/~severt/PUB/GiesbrechtEvert2009_Tagging.pdf

[6] (2012) Morphological tagging: data vs. dictionaries. Dostopno na: http://delivery.acm.org/10.1145/980000/974318/p94-hajic.pdf?ip=93.103.158.193&acc=OPEN&CFID=221726034&CFTOKEN=47325525&__acm__=1354829035_8015427d761bf6868a4d773db1d8de30

[7] (2012) MySQL Documentation. Dostopno na: http://dev.mysql.com/doc/

[8] Andreja Žele. *Vezljivostni slovar slovenskih glagolov*. Ljubljana: Založba ZRC SAZU, 2008.

[9] Domen Marinčič. *Strojno razčlenjevanje besedila z iskanjem stavkov in naštevanj: doktorska disertacija*. Ljubljana: [D. Marinčič], 2008.

[10] Jan Rupnik, Miha Grčar, Tomaž Erjavec. "Improving Morphosyntactic Tagging of Slovene Language through Meta-tagging," *Informatica* (Ljublj.), vol. 34, no. 2, str. 169–175, 2010.

[11] Joël Plisson, Nada Lavrač in Dunja Mladenic, *"A rule based approach to word lemmatization"*, In SiKDD 2004.

[12] Jože Toporišič. *Enciklopedija slovenskega jezika*. Ljubljana, Cankarjeva založba, 1992. str. 85; 316–317; 336.

[13] Jože Toporišič. *Slovenska slovnica*. Maribor: Obzorja, 2000. str. 469–693.

[14] Primož Jakopin. *Zgornja meja entropije pri leposlovnih besedilih v slovenskem jeziku: doktorska disertacija*. Ljubljana: [P. Jakopin], 1999. 35–45.

[15] Robert Jakomin. *Algoritem za analizo kratke povedi v slovenščini: diplomsko delo*. Ljubljana: [R. Jakomin], 2009.

[16] Tomaž Erjavec, Peter Tancig. "A system for morphological analysis of the Slovene language," *Informatica* (Ljublj.), vol. 14, str. 1–4, 1990.