Domen Marinčič

STROJNO RAZČLENJEVANJE BESEDILA Z ISKANJEM STAVKOV IN NAŠTEVANJ

Doktorska disertacija

AUTOMATIC TEXT PARSING AIDED BY CLAUSE SPLITTING AND INTRACLAUSAL COORDINATION DETECTION

Doctoral Dissertation

Mentor: prof. dr. Matjaž Gams

Somentor: dr. Tomaž Šef

December 2008

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL
Ljubljana, Slovenia

Kazalo

Tabele

Slike

Algoritmi

Povzetek

Skladenjsko razčlenjevanje na področju jezikovnih tehnologij predstavlja enega od vmesnih korakov analize besedila v aplikacijah, kot so strojno prevajanje, luščenje informacij, odgovarjanje na vprašanja itd. Za opis strukture povedi se pogosto uporablja skladenjska drevesa. Posebna vrsta skladenjskega razčenjevanja je odvisnostno razčlenjevanje.

Razčlenjevalniki iz besedila zgradijo drevesa. Pri podatkovno orientiranem razčlenjevanju je vir učnih in testnih podatkov drevesnica, to je besedilni korpus, ki je ročno označen z odvisnostnimi drevesi. Točnost razčlenjevanja se oceni tako, da se avtomatsko zgrajena odvisnostna drevesa iz besedila v testnem delu drevesnice primerja z ročno zgrajenimi drevesi.

V doktorski disertaciji je predstavljen novo razviti *a*lgoritem za *r*azčlenjevanje z *i*skanjem *s*tavkov *i*n *n*aštevanj – ARISiN, ki vključuje strojno učenje in hevristična pravila za predstavitev predznanja o jeziku. Algoritem se uporablja kot nadgradnja poljubnega obstoječega razčlenjevalnika. Slovenska odvisnostna drevesnica, SDT (angl. Slovene Dependency Treebank) je služila kot učna in testna množica za algoritem ARISiN. Algoritem je sestavljen iz dveh faz:

1. Iskanje in redukcija stavkov in naštevanj. Algoritem najprej s pomočjo hevurističnih pravil identificira kandidate za redukcijo. Nato uporabi strojne klasifikatorje in neprimerne kandidate zavrže. Preostale kandidate reducira v meta pojavnice. Ta postopek se ponavlja do takrat, ko algoritem ne uspe najti nobenega stavka ali naštevanja več.

2. Gradnja odvisnostnih dreves. Zaporedja besed, ki jih je algoritem reduciral v prvi fazi, razčlenijo trije različni osnovni razčlenjevalni modeli oziroma novo razviti razčlenjevalnik s pravili. Nova odvisnostna drevesa nato algoritem ARISiN združi v končno odvisnostno drevo povedi.

Poskusi so pokazali, da uporaba algoritma ARISiN v primerjavi z osnovnim razčlenjevalnikom MSTP poveča točnost razčlenjevanja za 1,27 odstotne točke (6,4% relativno zmanjšanje števila napak) ter za 1,91 odstotne točke (9,2% relativno zmanjšanje števila napak) v primerjavi z osnovnim razčlenjevalnikom Malt. Časovna zahtevnost algoritma za iskanje in redukcijo stavkov in naštevanj je $O(n)$, pri čemer je n število pojavnic v povedi. Glede na časovno zahtevnost razčlenjevalnikov MSTP $O(n^2)$ in Malt $O(n)$, ki sta bila uporabljena za izdelavo osnovnih razčlenjevalnih modelov, je dodatna poraba časa sprejemljiva.

Splošna ugotovitev, ki sledi iz opravljenega dela je naslednja: (i) dekompozicija kompleksnih razčlenjevalnih problemov v manjše podprobleme ter (ii) uporaba dodatnih

informacij, ki so na voljo v visoko pregibnih jezih, pozitivno vplivata na točnost razčlenjevanja.

Abstract

In language technologies, syntactic parsing represents one of the possible intermediate steps of text analysis in the applications such as machine translation, information extraction, question answering, etc. Syntactic trees are often used to demonstrate the structure of text. In the last decades, the dependency framework became a popular syntactic representation, describing the relations among the constituents of the sentence with dependency trees.

Parsing algorithms are used to infer dependency trees from text. In the data driven approach to parsing, a corpus of text, manually annotated with the dependency trees, serves for training and testing the algorithms. To estimate the accuracy of parsing, the test sentences are parsed and the output is then compared to the manually created trees.

We propose a new algorithm for *PA*rsing with *C*lause and *I*ntraclausal coordination *D*etection – PACID, which includes machine learning and the use of heuristic rules to incorporate language specifics. The algorithm is used as an upgrade of the existing parsing algorithms. For training and evaluating the algorithm, Slovene Dependency Treebank (SDT) was used, a corpus of Slovene text annotated with dependency trees. The algorithm constists of two phases:

1. Detection and reduction of clauses and intraclausal coordinations. First, the candidates for reduction are detected by heuristic rules. Then, the algorithm filters out false positives by machine learning classifiers. At the end, the remaining candidates are reduced to meta tokens. This procedure iterates until no more clauses and intraclausal coordinations can be retrieved.

2. Dependency tree construction. The sequences of text reduced in the first phase are parsed by three different base parsing models and a newly developed rule-based parser. At the end, the resulting trees are merged to form the final dependency tree of the sentence.

The experiments have shown that the use of the algorithm PACID raises the parsing accuracy by 1.27 percentage points (6.4% relative error reduction) compared to the plain MSTP parser and by 1.91 percentage points compared to the plain Malt parser (9.2% relative error reduction). The time complexity of the reduction algorithm equals $O(n)$, n being the number of tokens in the sentence, compared to the complexity $O(n^2)$ of the MSTP parser and $O(n)$ of the Malt parser, which were used to build the base parsing models. Additional time consumption thus seems acceptable.

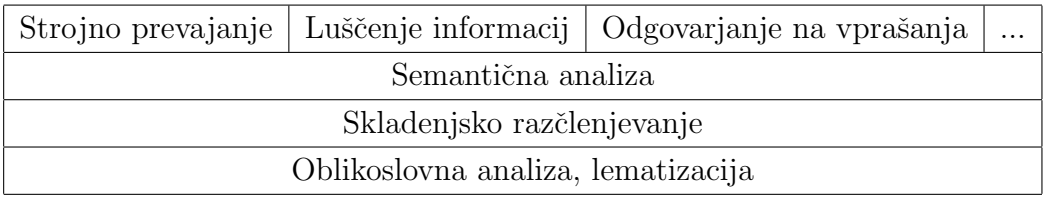
In summary, we have shown that decomposing large complex parsing problems to smaller ones is beneficial in terms of improving overall parsing accuracy and that additional information provided by the richly inflected languages can improve parsing results.

Poglavje 1

Uvod

Jezikovne tehnologije so interdisciplinarno področje računalništva in jezikoslovja, ki se ukvarja z avtomatsko obdelavo naravnega jezika. Opravila, kot so prevajanje med naravnimi jeziki, zapis govora, iskanje informacij v besedilih, izdelava slovarjev ipd. so bila donedavna izključno človekova domena. Po zaslugi razvoja jezikovnih tehnologij v zadnjih desetletjih marsikaj od naštetega uspešno naredijo računalniki. Večinoma sicer še niso sposobni popolnoma nadomestiti človeka, vendar v veliki meri olajšajo delo ljudem.

Skladenjsko razčlenjevanje predstavlja enega od postopkov analize besedila, točneje posameznih povedi, v okviru aplikacij, kot so strojno prevajanje, luščenje informacij, odgovarjanje na vprašanja ipd. (slika 1.1). Odvisnostno razčlenjevanje je posebna vrsta skladenjskega razčlenjevanja. Prvi korak obdelave besedila praviloma predstavlja oblikoslovna analiza in morebitna lematizacija. Temu lahko sledita skladenjska razčlenitev in semantična analiza, kar omogoči vpogled v strukturo in delno tudi pomen besedila. Možen je tudi pristop, v katerem se določene korake naredi hkrati.



Slika 1.1: Jezikovne tehnologije.

Osnovna enota besedila, ki jo obravnava skladenjsko razčlenjevanju, je poved. Opisana je z drevesom, ki ponazarja medsebojne odnose sestavnih delov povedi. Naloga avtomatskih razčlenjevalnikov je zgraditi čim bolj točno drevo iz vhodnega besedila. Pri tem se točnost presoja glede na *drevesnice* (angl. treebank) [1], t.j. korpuse besedil, kjer so posamezne povedi ročno označene z drevesi. Kompleksne povedi je smiselno pred razčlenjevanjem razgraditi na manjše podenote, kot so *stavki* in *naštevanja*.

Problem razčlenjevanja povedi tako poenostavimo, saj vsaka podenota predstavlja manjši podproblem, ki ga je možno reševati ločeno.

1.1 Namen, cilji

Kakovostno skladenjsko razčlenjevanje v splošnem omogoči boljše delovanje aplikacij s področja jezikovnih tehnologij. Namen raziskav v okviru doktorske disertacije je razvoj novih algoritmov za izboljšanje točnosti skladenjskega razčlenjevanja. Konkretni cilji so naslednji:

- Razvoj novih metod za iskanje stavkov in naštevanj z upoštevanjem posebnosti slovenskega jezika. Metode temeljijo na strojnem učenju in hevurističnih pravilih, in so poleg slovenščine uporabne tudi za sorodne visoko pregibne jezike.
- Vključitev iskanja stavkov in naštevanj v proces razčlenjevanja.
- Ovrednotenje novih algoritmov s pomočjo ročno zgrajene drevesnice.
- S pomočjo novih algoritmov za iskanje stavkov in naštevanj zvišati točnost skladenjskega razčlenjevanja za slovenščino.

1.2 Pregled vsebine

Drugo poglavje podaja pregled področja, ki ga obravnava doktorska disertacija, in predstavlja sorodna dela o skladenjskem razčlenjevanju ter iskanju stavkov in naštevanj. V tretjem poglavju so opisane tehnologije, uporabljene pri razvoju novih algoritmov. Četrto poglavje je namenjeno formalnim definicijam jezikovnih struktur, ki smo jih raziskovali v okviru doktorske disertacije. V petem poglavju je predstavljen ustroj novo razvitega *a*lgoritma za *r*azčlenjevanje z *i*skanjem *s*tavkov *i*n *n*aštevanj – ARISiN. Šesto poglavje ponuja izčrpen opis poskusov za ovrednotenje novih algoritmov. Zadnje poglavje vsebuje povzetek opravljenega dela in rezultatov, opisuje znanstveno relevantnost pričujočega dela in nakazuje možne poti nadaljnjega raziskovanja.

Poglavje 2

Pregled področja in sorodna dela

Poglavje vsebuje opis področij skladenjskega razčlenjevanja, iskanja stavkov in naštevanj ter podaja pregled sorodnega dela.

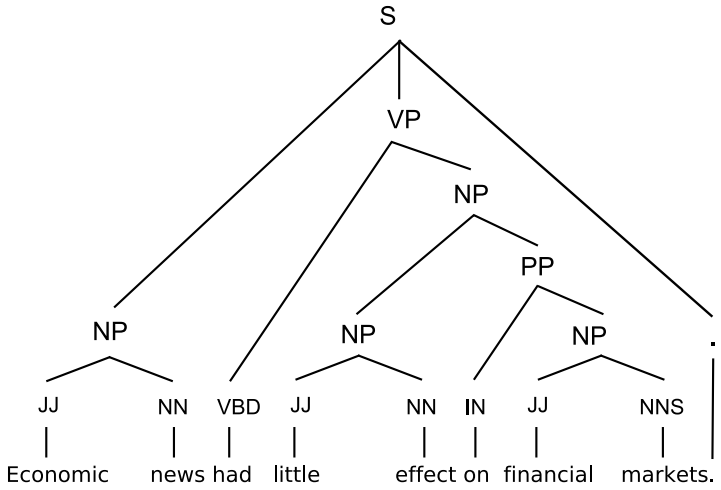
2.1 Skladenjsko razčlenjevanje

Obstajajo različne vrste skladenjskega razčlenjevanja besedila. Njihova skupna točka je, da za opis povedi v besedilu uporabljajo skladenjska drevesa. Razdelek najprej opiše dve pogosto uporabljani predstavitvi za opis skladenjske sestave povedi, predstavi dve odvisnostni drevesnici in nato poda še pregled metod avtomatskega razčlenjevanja.

2.1.1 Predstavitve za opis skladenjske sestave besedila

V zadnjih petdesetih letih je bila za ponazoritev strukture besedila najpogosteje v uporabi *dekompozicijska predstavitev* (angl. constituency representation). Dekompozicijska drevesa poved rekurzivno razstavijo na manjše dele. Glede na njihovo notranjo sestavo tem delom določijo tipe, kot so glagolska fraza (angl. verb phrase, VP), samostalniška fraza (angl. noun phrase, NP), predložna fraza (angl. prepositional phrase, PP) itd. Slika 2.1 prikazuje dekompozicijsko drevo za angleško poved "Economic news had little effect on financial markets." Slovenski prevod: "Gospodarske novice so imele majhen vpliv na finančne trge." (Povzeto po [62], drevo izvira iz drevesnice Penn Treebank [49].) Temelje dekompozicijske analize je leta 1933 postavil Bloomfield [3], formalno pa jo je z razvojem kontekstno neodvisne slovnice (CFG, Context-free grammar) leta 1956 opisal Chomsky [8].

Veliko teorij o skladnji temelji na dekompozicijski predstavitvi. Poleg del Chomskega [9, 10, 11, 12] so vidnejši predstavniki še LFG (Lexical functional grammar) [4, 44], GPSG (Generalized phrase strucure grammar) [31], TAG (Tree adjoining grammar) [43, 42] in HPSG (Head-driven phrase structure grammar) [69, 70].



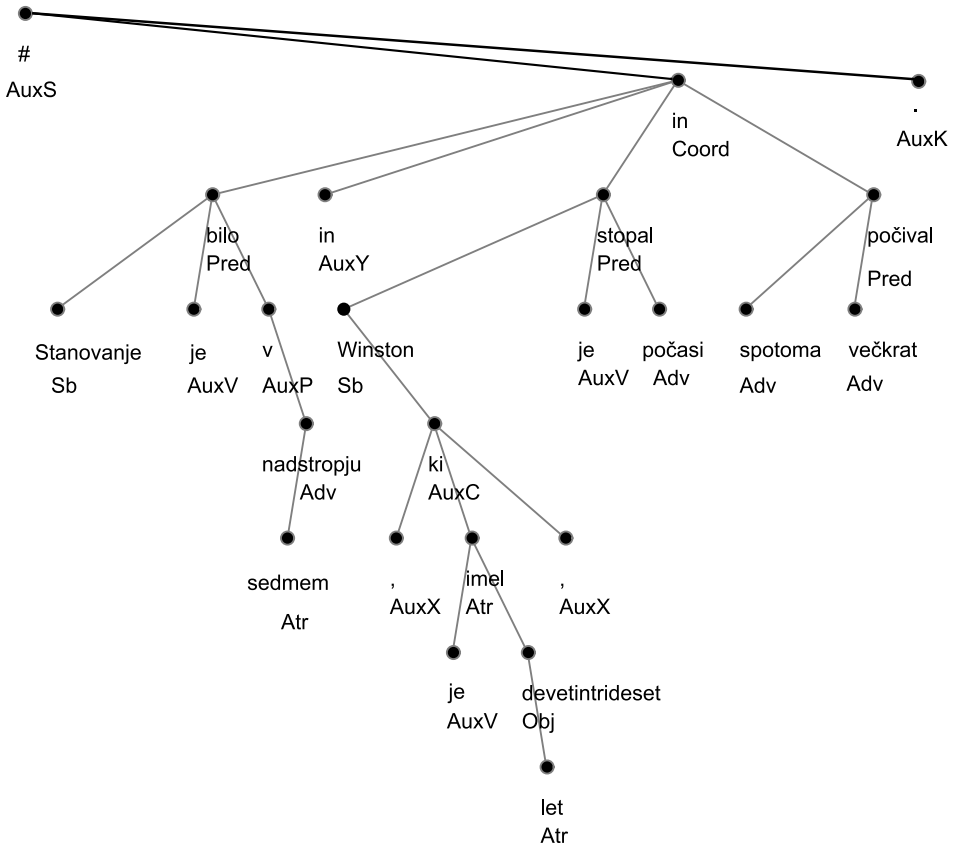
Slika 2.1: Primer dekompozicijskega drevesa.

Odvisnostna predstavitev (angl. dependency representation) je drug način za opis strukture besedila. V odvisnostnih drevesih so pojavnice (besede in ločila) med seboj povezane neposredno, brez vmesnih elementov. Povezave so označene glede na funkcionalno vlogo relacije med povezanima vozliščema. V nadalje sta izmenjujoče v uporabi izraz oče in nadrejeno vozlišče ter izraz otrok in podrejeno vozlišče v povezavi. Poleg vozlišč, ki predstavljajo pojavnice, je običajno dodano še tehnično vozlišče '#', s katerim zagotovimo, da je graf enodelen. Slika 2.2 prikazuje primer odvisnostnega drevesa za poved "Stanovanje je bilo v sedmem nadstropju in Winston, ki je imel devetintrideset let, je stopal počasi in se vmes večkrat ustavljal." Pod pojavnicami se nahajajo oznake povezav do nadrejenega vozlišča ('Pred': povedek, 'Sb': osebek, 'Obj': predmet itd.). Primer izvira iz Slovenske odvisnostne drevesnice SDT [20].

Tesnière velja za utemeljitelja odvisnostne analize [76]. Nekatere druge jezikoslovne teorije, ki temeljijo na odvisnostni predstavitvi, so še WG (Word grammar) [39, 40], FGD (Functional generative description) [73], Lexicase [74] in MTT (Meaning-text theory) [57].

Pojem odvisnosti temelji na tem, da je možno skladenjsko strukturo povedi predstaviti z binarnimi asimetričnimi relacijami med pojavnicami v povedi, ki so predstavljene z vozlišči v drevesu. Kriteriji za določanje odvisnosti med nadrejenimi in podrejenimi vozlišči so pri tem očitno najpomembnejši. Naslednji seznam podaja zbirko kriterijev, s pomočjo katerih je možno določiti odvisnost med nadrejenim (N) in podrejenim (P) vozliščem, ki skupaj sestavljata skladenjsko enoto S.

- N določa skladenjsko vlogo S in lahko mnogokrat nadomesti S.
- N določa semantično vlogo S, P podaja semantično opredelitev.
- N je obvezen, P je lahko neobvezen.



Slika 2.2: Primer odvisnostnega drevesa.

- N izbere P in določi, ali je P obvezen ali neobvezen.
- P se mora v slovničnih atributih ujemati z N.
- N določa položaj P v povedi.

Ti kriteriji predstavljajo neformalne smernice in jih lahko vzamemo kot okvirni skupni imenovalec za različne odvisnostne prestavitve; vsaka predstavitev zase točno določa pravila za določanje odvisnostnih relacij.

Predstavitve se med seboj razlikujejo glede na določene lastnosti. Nekateri uporabljajo več nivojev predstavitve, kjer poleg skladenjskega nivoja opisujejo še semantični nivo [73, 57], medtem ko Tesnièrovo prvo delo predlaga le skladenjski nivo.

Druga lastnost, po kateri se predstavitve razlikujejo med seboj, je nabor oznak povezav. Te so lahko omejene le na kategorije, kot so osebek, povedek, predmet itd., nekateri pa uporabljajo označevanje, ki opisuje tudi semantične lastnosti [41, 28, 21].

Odvisnostne predstavitve se ločijo tudi po tem, ali dopuščajo neprojektivne povezave ali ne. Povezava je projektivna, če ne seka nobene navidezne navpičnice, ki poteka navzdol iz višje ležečih vozlišč v drevesu, in neprojektivna sicer (definicija je vezana na predstavitev

drevesa takega tipa, kot je prikazana na sliki 2.2). V jezikih s prostim besednim vrstnim redom (primer: slovanski jeziki) pogosteje nastopajo skladenjske enote, katerih sestavni deli v povedi ne nastopajo v neprekinjenem zaporedju, poimenujmo jih nezvezne enote. Če so neprojektivne povezave dovoljene, je nezvezne enote laže opisati. To je tudi ena od prednosti odvisnostnih predstavitev v primerjavi z dekompozicijskimi. Pri zadnjih pojem neprojektivnosti sploh ne obstaja in je tako bistveno teže ustrezno opisovati nezvezne skladenjske enote.

2.1.2 Slovenska odvisnostna drevesnica (SDT) in Praška odvisnostna drevesnica (PDT)

Razdelek opisuje drevesnici SDT in PDT [33] v katerih je za zapis dreves uporabljena odvisnostna predstavitev. Drevesnica SDT je služila kot učno-testna množica podatkov pri raziskavah opisanih v pričujočem delu. Narejena je bila po vzoru drevesnice PDT [33] v češkem jeziku (Prague Dependency Treebank). Pravila za ročno označevanje drevesnice SDT je v svojem diplomskem delu po priročniku za označevanje drevesnice PDT priredila Ledinek [47].

Obe drevesnici temeljita na odvisnostni predstavitvi FGD (Functional generative description) [73], pri čemer drevesnica SDT vsebuje 55.208 pojavnic, ki so označene z odvisnostnimi drevesi na sintaktičnem nivoju. V drevesnici PDT je 1,5 miljonov pojavnic označenih na sintaktičnem nivoju, 0,8 miljona pa še na semantičnem nivoju. Obe drevesnici vsebujeta neprojektivne povezave. Vsako vozlišče predstavlja eno pojavnico razen tehničnega vozlišča – korena drevesa. Besedni vrstni red v povedi je enak vrstnemu redu vozlišč v drevesu, gledano z leve proti desni.

2.1.3 Avtomatsko razčlenjevanje

Eden od možnih pristopov k razčlenjevanju je slovnično orientiran pristop, kjer se uporablja formalno definirane slovnice, ki poskušajo opisati naravni jezik L. Ta pristop je problematičen zaradi sestave naravnih jezikov, ki se ne podrejajo formalnim omejitvam.

Pri drugem, podatkovno orientiranem pristopu se ne predpostavlja formalne strukture naravnega jezika L. Razčlenjevanje je definirano kot problem, kjer je za vsako poved treba najti pravilno razčlenitev v obliki drevesa. Ker v tem primeru ni na voljo formalne slovnice, je treba uporabiti drug kriterij za presojanje pravilnosti – zlati standard. To je referenčni korpus besedila z razčlenitvami, drevesnica, ki so jo ročno izdelali eksperti. Omeniti velja še, da v praksi razčlenjevalni algoritmi pogosto predstavljajo srednjo pot z združevanjem pozitivnih lastnosti obeh pristopov.

Pričujoče delo podrobneje obravnava podatkovno orientiran pristop k razčlenjevanju. V splošnem pri tem pristopu obstajajo tri komponente:

- Formalni model M, ki definira dovoljene razčlenitve povedi v jeziku L.
- Vzorec besedila $T_v = (x_1, ..., x_n)$, ki je sestavljen iz povedi x_i iz L in morebitnih pravilnih razčlenitev $A_v = (y_1, ..., y_n)$.
- Induktivni mehanizem I, ki določa dejanske razčlenitve povedi kakršnegakoli besedila $T = (x_1, ..., x_m)$ iz L glede na M, T_v in A_v, če je le-ta na voljo.

Model M je v praksi lahko kar formalna slovnica, kot je to primer pri predstavitvi PCFG (Probabilistic context-free grammar). Podatkovno orientirano razčlenjevanje je torej problem induktivnega sklepanja z možnimi omejitvami v obliki formalnih slovnic. Vzorec besedila T_v s pripadajočimi razčlenitvami A_v so *učni podatki*, ki jih dobimo iz drevesnice za jezik L. Pri tem razčlenitev y_i pripada povedi x_i. Induktivni mehanizem I običajno temelji na algoritmih nadzorovanega strojnega učenja [59]. Če razčlenitve A_v ne obstajajo, bo to primer nenadzorovanega učenja; tak pristop je redko v uporabi, saj tipično daje bistveno slabše rezultate.

Pri razčlenjevalnih sistemih je induktivni mehanizem običajno razdeljen na tri zaključene sklope:

- Parametrični model M_Θ, ki vsaki dopustni razčlenitvi y za poved x dodeli oceno $S(x, y)$ glede na množico parametrov Θ.
- Učna metoda, ki iz vzorca besedila T_v z induktivnim mehanizmom določi vrednosti parametrov iz množice Θ in tako definira parametrični model s konkretnimi vrednostmi.
- Razčlenjevalna metoda, katere izhod je najboljša razčlenitev y za poved x glede na $S(x, y)$ in določeno vrednost parametrov iz množice Θ.

Kakor je običajno pri metodah strojnega učenja, določanje parametrov z učenjem poteka samo enkrat, postopek razčlenjevanja pa poljubno mnogokrat. Za to, da je sistem uporaben v praksi, mora biti predvsem razčlenjevanje učinkovito s stališča porabe časa, medtem ko je za učenje možno uporabiti časovno zahtevnejše metode.

2.1.4 Sorodna dela

Ker je tema doktorske disertacije odvisnostno razčlenjevanje, se ta razdelek osredotoča le na to vrsto razčlenjevanja. Prvi sistemi za podatkovno orientirano odvisnostno razčlenjevanje so temeljili na formalnih modelih v obliki slovnic. Tak primer je delo Carrola idr. [7], kjer je opisan sistem za izdelavo slovnice z nenadzorovanim učenjem nad umetno izdelanim korpusom besedila. Rezultati so bili slabši v primerjavi z rezultati sistemov za dekompozicijsko razčlenjevanje v tistem času.

Eisner [22, 23] opisuje več razčlenjevalnih modelov. Ovrednotil jih je s pomočjo dela drevesnice Penn Treebank, ki izvira iz časopisa Wall Street Journal, s tem da je dekompozicijska drevesa najprej pretvoril v odvisnostna. Eisner je bil prvi, ki je uspel doseči podobno točnost pri odvisnostnem razčlenjevanju, kot so jo dosegali algoritmi za dekompozicijsko razčlenjevanje, ob upoštevanju dejstva, da kriteriji za ovrednotenje obeh vrst razčlenjevanja niso striktno ekvivalentni. Poleg tega je pokazal, da je za odvisnostno razčlenjevanje možno razviti algoritme s polinomsko časovno zahtevnostjo, ki pregledajo celoten prostor možnih razčlenitev, kar pri dekompozicijskem razčlenjevanju do danes ni možno. Njegov pristop je zasnovan tako, da predpostavlja projektivnost odvisnostnih dreves. S kombinacijo uporabe diskriminativnih ocenjevalnih metod in algoritma Chu-Liu-Edmonds za iskanje maksimalnih vpetih dreves v usmerjenih grafih je omenjeni pristop mogoče razširiti tako, da so v drevesih dovoljene tudi neprojektivne povezave [56].

Collins idr. [14] so za odvisnostno razčlenjevanje uporabili generativne verjetnostne modele na podatkih odvisnostne drevesnice PDT [32]. Postopek razčlenjevanja je potekal tako, da so odvisnostno drevesnico najprej pretvorili v dekompozicijski format. Podatki v taki obliki so služili kot učni primeri. Po razčlenjevanju so dekompozicijska drevesa pretvorili nazaj v odvisnostna.

Vsi do sedaj omenjeni pristopi so namenjeni le določanju povezav v drevesu. Wang in Harper [82] sta razvila razčlenjevalnik, temelječ na slovnici CDG (Constraint dependency grammar) [53, 36, 58], kjer je v postopek razčlenjevanja vključeno tudi označevanje povezav. Razčlenjevalnik v prvem koraku za pojavnice določi omejitve, t.j. možne kandidate za njihova nadrejena vozlišča. V drugem koraku določi končne povezave. V splošnem je algoritem omejen na projektivna drevesa, za določene vrste vprašalnih stavkov pa dovoljuje tudi neprojektivne povezave. Sistem so ovrednotili na drevesnici Penn Treebank, na besedilu iz časopisa Wall Street Journal.

Do zdaj omenjeni sistemi so temeljili na generativnih verjetnostnih modelih. Deterministični diskriminativni pristop sta prvič predlagala Kudo in Matsumoto [46] in nato še Yamada in Matsumoto [84]. Njihov razčlenjevalni algoritem deluje po načelu premakni – reduciraj. Z metodo podpornih vektorjev [81] napoveduje naslednjo akcijo. Vsako poved pregleda tolikokrat, dokler vsem pojavnicam ne določi nadrejenega vozlišča. Podoben sistem je razvil tudi Nivre [62], le da njegov sistem celotno drevo zgradi v enem prehodu čez poved, hkrati določa tudi oznake povezav in za izbiro razčlenjevalnih akcij uporablja leno učenje (angl. lazy learning).

Najobsežnejši pregled odvisnostnega razčlenjevanja ponujata konferenci CoNLL 2006 (Conference on Computational Natural Language Learning) [5] in CoNLL 2007 [63], kjer sta bili organizirani tekmovanji 19 (2006) oziroma 23 (2007) odvisnostnih razčlenjevalnikov, ki so bili preizkušeni na 12 (2006) oziroma 10 (2007) drevesnicah. Na

konferenci CoNLL 2006 je najboljše rezultate dosegal razčlenjevalnik MSTP (Maximum spanning-tree parser) [55], na konferenci CoNLL 2007 pa razčlenjevalnik Malt [34].

2.2 Iskanje stavkov

V znanstveni literaturi obstaja več pogledov na to, kaj sestavlja stavek [48, 79]. Za slovenščino Toporišič [80] predlaga naslednjo definicijo: "Stavek je enota besedila, katere jedro je sestavljena glagolska oblika. Vsebuje lahko dodatne elemente, kot so osebek, predmet, prilastek ipd., ki glagolsko obliko točneje opisujejo." Vendar se takoj postavijo vprašanja. Kaj storiti s primeri, kjer je glagol opuščen (elipsa glagola)? Kako postopati z deli povedi, ki ne vsebujejo glagola in so skladenjsko in pomensko ločeni od drugih stavkov? Ker ne obstaja splošno sprejeta enotna definicija stavka, je v navadi, da glede na podatke, ki jih obdelujemo, izdelamo točno formalno definicijo.

V drevesnici Penn Treebank so v besedilu razmejeni tudi stavki. Začetki stavkov so označeni s predklepaji, konci stavkov pa z zaklepaji. Tik za odprtim oklepajem je za vsak stavek določen še tip, kot prikazuje primer na sliki 2.3.

```
(S Coach them in
  (S-NOM handling complaints)
  (SBAR-PRP so that
    (S they can resolve problems immediately)
   )
  .
)
```

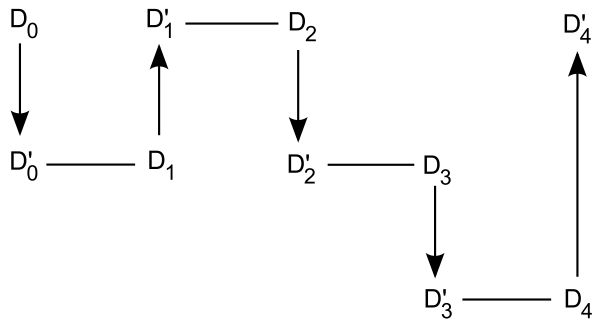
Slika 2.3: Primer označitve stavkov.

Kuboň [45] za češčino predlaga sistem za razmejitev povedi na segmente s pomočjo diagramov, ki opisujejo relacijo podrejenosti/nadrejenosti. Segmente razmejujejo ločila in vezniki. Diagram predstavlja osnovo, kako razbrati stavke znotraj povedi. Češka poved "Zatímco neúspěch bývá sirotkem, úspěch mívá mnoho tatínků, horlivě se hlásících, že zrovna oni byli u jeho početí." je razdeljena na segmente na naslednji način (pojavnice označene z D_i predstavljajo mejnike med segmenti, poleg češkega besedila je v oklepajih še prevod v slovenščino):

- D_0: Zatímco (Medtem ko)
- neúspěch bývá sirotkem (je neuspeh sirota)
- D_1: ,

- úspěch mívá mnoho tatínků (ima uspeh mnogo očetov)
- D_2: ,
- horlivě se hlásících (goreče se oglašajočih)
- D_3:, že (, da)
- zrovna oni byli u jeho početí (so ga ravno oni spočeli)
- D_4: .

Na sliki 2.4 je prikazan diagram segmentov za to poved. Vodoravne črte prikazujejo segmente, ki so na različnih nivojih podrejenosti/nadrejenosti. Iz diagrama lahko razberemo, da drugi segment predstavlja glavni stavek v povedi, medtem ko so ostali segmenti podrejeni glavnemu stavku.



Slika 2.4: Primer diagrama segmentov v povedi.

2.2.1 Sorodna dela

Med pionirje na področju iskanja stavkov sodita Abney [2] za angleščino in Ejerhed [24] za švedščino. Abney iskanje stavkov uporablja kot filter v svojem razčlenjevalniku CASS (Cascaded analysis of syntactic structure). Iskalnik stavkov s pravili poišče začetke in konce enostavnih stavkov, ki vsebujejo osebek in povedek, in ni namenjen iskanju vrinjenih stavčnih struktur. Sistem Ejerhedove je namenjen iskanju stavkov za izboljšanje sinteze govora. Če pri sintezi prepozna stavčne meje, je možno sintetizirati bolj naraven govor. Njen sistem ravno tako ni zmožen prepoznavati vrinjenih stavkov.

Papageorgiu [67] je razvil sistem za določanje stavčnih mej za splošno angleško besedilo. Avtor navaja 93% točnost pri ovrednotenju na 562 stavkih. Sistem je bil v uporabi kot predobdelava za dvojezično poravnavo besedila. Leffa [48] je razvil sistem, ki predpostavlja, da je možno stavke reducirati na posamezne besede, samostalnike,

pridevnike ali prislove, ne glede na njihovo dolžino ali število vrinjenih stavkov, ki jih vsebujejo. V članku je naveden priklic preko 95 % stavkov, kar je bilo ocenjeno s testi na 500 povedih. Iskanje stavkov je del strojnega prevajalnika med angleščino in portugalščino.

Do sedaj opisani sistemi, temelječi na ekspertnih pravilih, so dosegali dobre rezultate s točnostjo oziroma priklicem nad 90 % (primerjava rezultatov različnih avtorjev služi lahko le za orientacijo, saj mere točnosti niso vedno enake, prav tako tudi ne definicija in kompleksnost stavkov). V izogib ročnemu delu pri izdelavi pravil je možno uporabiti strojno učenje nad korpusom, označenim s stavčnimi mejami. Orasăn [65] je za iskanje stavkov v angleškem besedilu uporabil leno učenje in manjše število pravil za naknadno obdelavo. Sistem je dosegel F-mero (angl. F-measure) 85 %. Njegovo delo je razširila Puscasu [71], ki je algoritme ovrednotila tudi na romunščini (F-mera 88,76 %).

Na konferenci CoNLL 2001 je bilo prirejeno tekmovanje sistemov za iskanje stavkov. Tekmovanje je bilo razdeljeno na tri dele: določanje začetkov in koncev stavkov ter iskanje celih stavkov. Tekmovalo je šest sistemov. Ob njihovem opisu je navedena dosežena F-mera pri iskanju celotnih stavkov v besedilu iz korpusa Penn Treebank. Patrick in Goyal [68] sta uporabila algoritem AdaBoost nad odločitvenimi grafi (F-mera 66 %). Hammertonov sistem [35] temelji na nerekurentnih nevronskih mrežah (angl. feed-forward neural networks) (F-mera 50 %). Déjean [18] je uporabil lasten sistem ALLiS (Architecture for learning linguistic structure) (F-mera 62 %). Tjong Kim Sang [77] je sistem za iskanje stavkov zgradil na podlagi lenega učenja (F-mera 67 %). Molina in Pla [60] sta uporabila posebej prirejen prikriti markovski model (F-mera 68 %). Carreras in Marquez [6] sta dosegla najboljši rezultat s sistemom binarnih odločitev, ki sta jih modelirala z odločitvenimi drevesi in algoritmom AdaBoost (F-mera 78 %). Nadaljnji razvoj nekaterih omenjenih sistemov je opisan v [78, 61, 19].

2.3 Razčlenjevanje z iskanjem stavkov

Osnovni princip delovanja razčlenjevalnikov je, da poved obravnavajo kot nedeljivo celoto. Pri tem izrecno ne upoštevajo, da so pojavnice znotraj povedi strukturirane v manjše zaključene podenote, kot so npr. stavki, ki jih je možno obravnavati ločeno [65], [25]. Razen [2] tej ideji po našem vedenju niso posvečali veliko pozornosti. Holán in Žabokrtský [38] opisujeta razčlenjevalnik s pravili za češčino. Prepoznavanje stavkov, ki poteka s pomočjo ekspertnih pravil, je vpleteno v proces razčlenjevanja. Ohno in drugi [64] so razvili sistem za razčlenjevanje govorjenega monologa v japonščini. Sistem obravnava besedilo sproti, v živo, preden je celoten govor dejansko na voljo. Osnova algoritma je razmejitev na stavke, kjer stavek predstavlja temeljno enoto razčlenjevanja. Za razčlenjevanje in iskanje stavkov uporabljajo lasten sistem CBAP (Clause boundary analysis) [54].

2.4 Iskanje naštevanj

Naštevanje (angl. intraclausal coordination) ravno tako predstavlja manjšo zaključeno enoto znotraj povedi oziroma stavka. Hogan [37] predstavlja algoritem za iskanje naštevanj samostalnikov v angleščini, ki izkorišča semantične lastnosti besed. Avtorica kot možnost navaja, da bi lahko algoritem uporabili kot predhodno obdelavo pred razčlenjevanjem, vendar je algoritem preizkusila samostojno. Marinčič idr. [50] smo predstavili algoritem, ki vključuje iskanje naštevanj v odvisnostno razčlenjevanje, kar je podrobno opisano v nadaljevanju doktorske disertacije.

Poglavje 3

Pregled uporabljenih tehnologij

Poglavje predstavlja obstoječe tehnologije in metode, ki so v uporabi v našem algoritmu za razčlenjevanje z iskanjem stavkov in naštevanj (ARISiN). V vlogi osnovnega odvisnostnega razčlenjevalnika, vključenega v algoritem ARISiN, sta pri poskusih nastopala algoritma MSTP [56] in Malt [62]. Tehnologija delitve povedi na segmente, prirejeno po [45], predstavlja osnovo za iskanje stavkov. Za klasifikacijo stavkov in naštevanj so v uporabi metode strojnega učenja.

Najprej formalno definirajmo osnovne pojme, s katerimi operirajo razčlenjevalniki. Naj bo *besedilo* $B = (x_1, ..., x_n)$ zaporedje *povedi* $x_i = (t_{i,1}, ..., t_{i,n_i})$, poved x_i pa zaporedje *pojavnic* $t_{i,j}$. Naloga odvisnostnega razčlenjevalnika je za vsako poved x_i zgraditi *odvisnostno drevo* y_i, ki je definirano kot usmerjen graf $G = (V, E, L)$, kjer je $V = \{\nu_1, ..., \nu_{n_V}\}$ množica vozlišč, ki predstavljajo pojavnice v povedi, $E = \{(i, j)\}, i = 1...n_V, j = 1...n_V$ množica povezav ter $L = \{l_1, ..., l_{n_V}\}$ množica oznak povezav. Graf mora biti povezan in acikličen. Definirajmo, da povezava (i, j) poteka od nadrejenega vozlišča ν_i do podrejenega vozlišča ν_j. Pri grafičnem prikazu drevesa smer povezave običajno ni označena. Zaradi zahteve, da ima drevo en sam koren, je lahko samo tehnično vozlišče (glej 2.1.1) brez vhodnih povezav, vsa ostala vozlišča pa imajo natanko eno vhodno povezavo.

3.1 Razčlenjevalnik MSTP

Razčlenjevalnik MSTP odvisnostno razčlenjevanje modelira kot iskanje maksimalnega vpetega drevesa v usmerjenih grafih. Nad pojavnicami v povedi, predstavljenimi z vozlišči, algoritem najprej zgradi graf. Povezavam med vozlišči priredi ocene glede na lastnosti povezave in vrednosti utežnega vektorja, ki jih pridobi z učenjem nad ročno označeno drevesnico. Nato algoritem poišče maksimalno vpeto drevo grafa, ki predstavlja odvisnostno drevo povedi. Določanje oznak povezav je ločeno od gradnje odvisnostnega drevesa. Za določanje oznak povezav so avtorji uporabili markovski model prvega reda,

ki upošteva soodvisnost oznak povezav od vozlišča do vseh njegovih otrok.

3.1.1 Določanje vrednosti utežnih vektorjev

Učni algoritem deluje po vzoru algoritma MIRA (Margin infused relaxed algorithm) [16, 15]. V več iteracijah preko vseh učnih podatkov izračuna vrednosti utežnega vektorja (algoritem 1). V vsakem koraku algoritem poskuša vrednost uteži obdržati čim bliže prejšnji vrednosti. Pri tem upošteva omejitev, da mora biti razlika ocen pravega drevesa in kateregakoli nepravega drevesa večja od števila napačnih povezav v nepravem drevesu.

Algoritem 1 Algoritem za učenje uteži (povzeto po [16]).

Vhod:

- $\mathcal{T} = \{(x_t, y_t)\}, t = 1..N_t$: učni podatki. x_t je poved in y_t pripadajoče ročno zgrajeno odvisnostno drevo.
- $s(\nu_i, \nu_j)$: ocena povezave, definirana kot skalarni produkt $\mathbf{u} \cdot \mathbf{l_{(i,j)}}$, pri čemer je $\mathbf{u}$ vektor uteži, $\mathbf{l_{(i,j)}}$ pa vektor lastnosti z dvojiškimi vrednostmi, ki predstavljajo povezavo med vozliščema ν_i in ν_j.
- $S(x, y)$: ocena drevesa $y = (V, E)$, definirana kot $\sum_{(i,j)\in E} s(\nu_i, \nu_j)$.
- $dt(x)$: množica vseh možnih dreves povedi x.
- $L(x, y^{'})$: število nepravilnih povezav v poljubnem drevesu $y^{'}$ nad povedjo x glede na pravilno drevo y.

Izhod:

- $\mathbf{u^n}$: vektor uteži v n-ti iteraciji učenja.

1: $\mathbf{u^0} := \mathbf{0}$
2: $\mathbf{v} := \mathbf{0}$
3: **for all** $n := 1..N$ **do**
4: **for all** $t := 1..N_t$ **do**
5: $\mathbf{u^n}$:= popravi $\mathbf{u^{n-1}}$, tako da bo norma razlike $\|\mathbf{u^n} - \mathbf{u^{n-1}}\|$ minimalna, pri omejitvi $\forall y_t^{'} \in dt(x_t) : S(x_t, y_t) - S(x_t, y_t^{'}) \geq L(x_t, y_t^{'})$
6: $\mathbf{v} := \mathbf{v} + \mathbf{u^n}$
7: **end for**
8: **end for**
9: $\mathbf{u} = \mathbf{v}/(N * N_t)$

3.1.2 Iskanje maksimalnega vpetega drevesa

Algoritem MSTP najprej sestavi poln graf, v katerem vsako vozlišče predstavlja eno od pojavnic v povedi. Nato doda še tehnično vozlišče in povezave od tehničnega vozlišča do

vsakega od ostalih vozlišč. Za iskanje maksimalnega vpetega drevesa uporablja algoritem Chu-Liu-Edmonds [13] (glej algoritma 2, 3), ki preišče celoten prostor možnih rešitev. Za vsako vozlišče izbere vhodno povezavo z največjo oceno in tako dobi podgraf. Če takoj zgradi drevo, je to maksimalno vpeto drevo in s tem končna rešitev, sicer poišče cikle v podgrafu. Skupine vozlišč v ciklih reducira v eno samo vozlišče v originalnem grafu ter na novo preračuna ocene vhodnih in izhodnih povezav novega vozlišča. Nato sledi rekurziven klic, dokler algoritem ne najde drevesa. Izvedba algoritma, ki jo predlaga Tarjan [75], ima časovno zahtevnost $O(n^2)$.

Algoritem 2 Chu-Liu-Edmonds (povzeto po [16]).

Vhod:

- $P = (V, E)$: graf nad povedjo, V: množica vozlišč, E: množica povezav.
- $s(\nu_i, \nu_j)$: Glej algoritem 1.

Izhod:

- P_M: maksimalno vpeto drevo.

1: $M := \{(\nu^*, \nu) : \nu, \nu^*, \nu^{'} \in V; \nu^* = argmax_{\nu'} s(\nu^{'}, \nu)\}$
2: $P_M :=$ (V,M)
3: **if** P_M ne vsebuje ciklov **then**
4: **return** P_M
5: **else**
6: najdi cikel C v P_M
7: $P_C :=$ reduciraj(P, C, s)
8: $y :=$ Chu-Liu-Edmonds
9: Najdi vozlišče $\nu \in C$, tako da $(\nu^{'}, \nu) \in y \wedge (\nu'', \nu) \in C$
10: **return** $y \cup C - \{(\nu'', \nu)\}$
11: **end if**

3.2 Malt

Razčlenjevalnik Malt postopek razčlenjevanja opiše kot zaporedje stanj, t.i. konfiguracij, ki vodijo od začetnega (brez povezav med vozlišči) do končnega stanja (vsa vozlišča so povezana v odvisnostno drevo). Z vsakim prehodom med stanji razčlenjevalnik obravnava eno pojavnico v povedi. Ker razčlenjevalnik vsako pojavnico obdela samo enkrat, je njegova časovna zahtevnost $O(n)$, pri čemer je n število pojavnic v povedi. Algoritem določa oznake povezav sproti, hkrati z gradnjo drevesa.

Algoritem 3 Funkcija reduciraj(P, C, s).

Vhod:

- $P = (V, E)$: graf nad povedjo, V: množica vozlišč, E: množica povezav.
- C : cikel v grafu P.
- s : ocenjevalna funkcija (glej algoritem 1).

Izhod:

- $P_C = (V_C, E_C)$: podgraf grafa P brez vozlišč cikla C.

1: V množico V_C dodaj novo vozlišče c, ki predstavlja cikel C
2: **for all** $\nu \in V - C : \exists_{\nu' \in C}(\nu', \nu) \in E$ **do**
3: $P_C := P_C \cup (c, \nu)$, tako da $s(c, \nu) = max_{\nu' \in C} s(\nu', \nu)$
4: **end for**
5: **for all** $\nu \in V - C : \exists_{\nu' \in C}(\nu, \nu') \in E$ **do**
6: $P_C := P_C \cup (\nu, c)$, tako da $s(c, \nu) = max_{\nu' \in C}[s(\nu, \nu') - s(a(\nu'), \nu') + s(C)]$ pri čemer je $a(v)$ predhodnik v v C in $s(C) = \sum_{v \in C} s(a(v), v)$
7: **end for**
8: **return** P_C

3.2.1 Razčlenjevanje kot zaporedje prehodov med konfiguracijami

Vozlišča, ki predstavljajo pojavnice iz povedi x dolžine n, poimenujmo z indeksi i, $1 \leq i \leq n$.

Definicija 1. Konfiguracija je četvorka $c = (\Sigma, T, h, d)$, kjer je:

- Σ sklad vozlišč k $(1 \leq k \leq j$ pri čemer je $j \leq n)$,
- T urejen seznam vozlišč l $(j < l \leq n)$,
- $h(i)$: funkcija, ki določa očeta vozlišča i, z vrednostjo 0, če oče ni definiran,
- $d(i)$: funkcija, ki določa oznako povezave med vozliščem i in njegovim očetom, z vrednostjo 0, če oznaka ni definirana.

Sklad Σ predstavlja vozlišča pojavnic, ki jih je razčlenjevalnik že obravnaval, medtem ko so v seznamu T preostala vozlišča. Funkciji h in d predstavljata delno zgrajeno odvisnostno drevo. Postopek razčlenjevanja opisuje algoritem 4. Razčlenjevanje se začne z *začetno konfiguracijo* $((), (1, ..., n), h_0, d_0)$ kjer velja $\forall i : h_0(i) = 0 \wedge d_0(i) = 0$ in zaključi s *končno konfiguracijo* $((1, ..., n), (), h, d)$. V začetni konfiguraciji je seznam neobravnavanih vozlišč poln in se s prehodi proti končni konfiguraciji prazni. S prehodom med konfiguracijami se spreminjata funkciji h in d, ki v končni konfiguraciji definirata

odvisnostno drevo povedi. Razčlenjevalnik pozna naslednje prehode med konfiguracijami, poleg njih so navedeni pogoji, kdaj so prehodi izvedljivi:

1. *LEFT-ARC*: $(\Sigma|i, j|T, h, d) \rightarrow (\Sigma, j|T, h \cup [i \rightarrow j], d \cup [i \rightarrow r])$, če $h(i) = 0$,
2. *RIGHT-ARC*: $(\Sigma|i, j|T, h, d) \rightarrow (\Sigma|i|j, T, h \cup [j \rightarrow i], d \cup [j \rightarrow r])$, če $h(j) = 0$,
3. *REDUCE*: $(\Sigma|i, T, h, d) \rightarrow (\Sigma, T, h, d)$, če $h(i) \neq 0$,
4. *SHIFT*: $(\Sigma, i|T, h, d) \rightarrow (\Sigma|i, T, h, d)$.

Algoritem 4 Algoritem Malt, postopek razčlenjevanja.

Vhod:

- $g(c)$: klasifikator, ki glede na konfiguracijo c določi naslednji prehod p.
- $\leftarrow p(c)$: operacija, ki vrne novo konfiguracijo glede na prehod p iz konfiguracije c.

Izhod:

- h, d: funkciji, ki definirata odvisnostno drevo.

```
1: c := ((), (1, ..., n), h_0, d_0)
2: while c = (Σ, T, h, d) ni končna konfiguracija do
3:     if Σ = () then
4:         c ← SHIFT(c)
5:     else
6:         c ← [g(c)](c)
7:     end if
8: end while
```

3.2.2 Priprava učne množice za klasifikator prehodov

Množico učnih primerov razčlenjevalnik pridobi na osnovi zlatega standarda, to je ročno izdelane odvisnostne drevesnice. Določanje učnih primerov prikazuje algoritem 5. Ravno tako kot pri razčlenjevanju je učni algoritem zaporedje prehodov med začetno in končno konfiguracijo, le da se namesto s klasifikatorjem nov prehod določi skladno z drevesom v zlatem standardu (glej algoritem 6, funkcija *oracle*). Ob vsakem prehodu algoritem doda v učno množico en primer. Ker je število prehodov enako številu pojavnic v povedi, se iz celotne drevesnice dobi število učnih primerov, ki je enako dolžini besedila v drevesnici.

Pri poskusih, opisanih v [62], so avtorji za strojni klasifikator uporabili programski paket TiMBL (Tilburg Memory-Based Learner) [17], ki temelji na algoritmih lenega učenja.

Algoritem 5 Algoritem Malt, priprava učne množice.

Vhod:

- $\mathcal{D} = \{(x, y)\}$: drevesnica. x je poved in $y = (V, E, L)$ pripadajoče ročno zgrajeno odvisnostno drevo.
- $\leftarrow p(c)$: operacija, ki vrne novo konfiguracijo glede na prehod p iz konfiguracije c.
- $h(i) = j \Leftrightarrow (j, i) \in E$.
- $d(i) = r \Leftrightarrow \exists j \in V_t : ((j, i), r) \in L$.

Izhod:

- $U = \{(c, p)\}$: množica učnih primerov, kjer so c konfiguracije in p prehodi iz konfiguracij.

for all $(x, y) \in \mathcal{D}$ **do**
    $c := ((), (1, ..., n), h_0, d_0)$
    while $c = (\Sigma, T, h, d)$ ni končna konfiguracija **do**
        if $\Sigma = ()$ **then**
            $c \leftarrow SHIFT(c)$
        else
            $p \leftarrow oracle(c, h, d)$
            $U = U \cup (c, p)$
            $c \leftarrow p(c)$
        end if
    end while
end for

Algoritem 6 Funkcija $oracle(c = (\Sigma|i, j|T, h_c, d_c), h, d)$.

Vhod: glej algoritem 5

Izhod:

- Ustrezen prehod, glede na strukturo drevesa.

if $h(i) = j$ **then**
    return $LEFT - ARC(d(i))$
else if $h(j) = i$ **then**
    return $RIGHT - ARC(d(j))$
else if $\exists k \in \Sigma : h(j) = k$ **or** $h(k) = j$ **then**
    return $REDUCE$
else
    return $SHIFT$
end if

3.2.3 Atributni opis konfiguracij

Ker bi opis konfiguracij s celotnim seznamom T, skladom Σ in funkcijama h ter d povzročil problem redkosti podatkov, so v razčlenjevalniku Malt z atributnim opisom določeni ekvivalenčni razredi konfiguracij.

V atributnem opisu nastopajo lastnosti naslednjih vozlišč:

- σ_0: zgornje vozlišče na skladu Σ,
- σ_n: n-to vozlišče na skladu Σ,
- τ_0: naslednje obravnavano vozlišče na seznamu T,
- τ_n: n-to naslednje vozlišče na seznamu T,
- $h(\sigma_0)$: oče zgornjega vozlišča,
- $l(\sigma_0)$: skrajno levi otrok zgornjega vozlišča (sklad Σ),
- $r(\sigma_0)$: skrajno desni otrok zgornjega vozlišča (sklad Σ),
- $l(\tau_0)$: skrajno levi otrok naslednjega vozlišča (seznam T).

Od naštetih vozlišč so v atributni opis vključene naslednje tri lastnosti: MSD oznake (angl. Morphosyntactic description tags) in besedne oblike pripadajočih pojavnic ter oznake povezav. Označene so s funkcijami m, w in d. Na primer, $m(\sigma_0)$ označuje atribut z vrednostjo MSD oznake zgornjega vozlišča, $w(\tau_0)$ je atribut z vrednostjo besedne oblike naslednjega vozlišča, $d(l(\tau_0))$ pa označuje atribut, kjer je zapisana oznaka povezave med naslednjim vozliščem in njegovim skrajno levim otrokom.

Razčlenjevalnik Malt tako loči tri različne modele, ki so združeni v atributni opis:

- MSD model, označen kot Φ^m_{ij}, $i, j \geq 0$. V ta model so vključeni atributi $m(\sigma_0), m(\sigma_1), ..., m(\sigma_i)$, in $m(\tau_0), m(\tau_1), ..., m(\tau_j)$.
- Slovarski model, označen kot Φ^w_{ij}, $i, j \geq 0$. Atributi tega modela so $w(\sigma_0), w(\sigma_1), ..., w(\sigma_i)$, in $w(\tau_0), w(\tau_1), ..., w(\tau_j)$.
- Odvisnostni model, označen kot Φ^d_{ijk}. Model vedno vsebuje atribut $d(\sigma_0)$. Parametri i, j in k (dvojiške vrednosti) določajo prisotnost atributov $d(l(\sigma_0))$, $d(r(\sigma_0))$ in $d(l(\tau_0))$.

Pri uporabi razčlenjevalnika Malt je možno s parametri določiti, kateri in kakšni modeli so vključeni v atributni opis.

3.3 Segmentacija povedi

Segmentacija povedi je razdelitev pojavnic na krajša zaporedja, pri čemer meje predstavljajo ločila in vezniki. Naj bo poved naslednje zaporedje pojavnic:

$$(s_{1,1}, ..., s_{1,k_1}, d_{1,1}, ..., d_{1,l_1}, ..., s_{i,j}, ..., d_{i,j}, ..., s_{n,1}, ..., s_{n,k_n}, d_{n,1}, ..., d_{n,l_n}).$$

Pri tem so pojavnice $d_{i,j}$ ločila ali vezniki, $s_{i,j}$ pa ostale besede. Zaporedja $(s_{i,1}, ..., s_{i,k_i})$ so segmenti, medtem ko so zaporedja $(d_{i,1}, ..., d_{i,l_i})$ mejniki. Glagolski segmenti vsebujejo vsaj en določni glagol, medtem ko neglagolski segmenti od glagolov lahko vsebujejo zgolj nedoločnike.

Definicija 2. Segmentacija povedi je zaporedje mejnikov in segmentov $(\mathcal{S}_1, \mathcal{D}_1, ..., \mathcal{S}_n, \mathcal{D}_n)$, pri čemer je $\mathcal{S}_i = (s_{i,1}, ..., s_{i,k_i})$ in $\mathcal{D}_i = (d_{i,1}, ..., d_{i,l_i})$.

3.4 Strojno učenje

Razdelek podaja kratek opis strojnega učenja in delovanja strojnih klasifikatorjev, s poudarkom na tistih algoritmih, ki so sestavni del algoritma ARISiN. Metode strojnega učenja so namenjene odkrivanju zakonitosti v strukturiranih ali nestrukturiranih podatkih. Motivacija za uporabo teh metod je, da človek laže poda konkretne primere, kot pa da bi izluščil pravila, ki primere jedrnato opisujejo. Iskanje pravil se torej prepusti računalniku. Običajno metode strojnega učenja razdelimo v tri skupine:

- *Nadzorovano učenje.* Učni algoritmi odkrivajo pravila v strukturiranih podatkih. Večinoma gre tu za klasifikacijo podatkov, kjer se algoritmi naučijo, kateremu razredu pripada nek primer. Učno množico predstavljajo primeri, ki imajo vnaprej določen razred.

- *Nenadzorovano učenje.* Učni algoritmi odkrivajo zakonitosti v podatkih, ne da bi bili ti podatki kakorkoli vnaprej strukturirani. Končni rezultat so gruče primerov, v katerih primeri izkazujejo določene skupne lastnosti.

- *Spodbujevano učenje.* V tem primeru algoritem izbira odločitev izmed več, ki jih ima na voljo. Ko izbere odločitev, kot povratno informacijo dobi le to, ali je bila odločitev pravilna (nagrada) ali ne (kazen). Nagrade in kazni vodijo postopek učenja in tako vplivajo na prihodnje odločitve.

Predstavnik prve skupine algoritmov je J48 [83] (priredba algoritma C4.5 [72]), ki gradi odločitvena drevesa. Meta klasifikatorji, npr. algoritem AdaBoost [29], kombinirajo več osnovnih učnih algoritmov [30]. Ti iz ene učne množice naredijo več osnovnih klasifikatorjev, katerih napovedi je treba utežiti, da dobimo razred za konkretni primer.

Poglavje 4

Jezikoslovne definicije

Poglavje podaja formalne definicije nekaterih jezikoslovnih pojmov (stavek, naštevanje), kot so uporabljene v doktorski disertaciji. Definicije temeljijo na strukturi Slovenske odvisnostne drevesnice (Slovene Dependency Treebank) SDT [20], ki je predstavljala učno-testne podatke za naše algoritme in hkrati tudi na definicijah iz Toporišičeve Slovenske slovnice [80].

Struktura odvisnostnih dreves vsebuje večino potrebnih informacij, ki so potrebne za definicije. Poleg tega je v definicije vključen še morfološki nivo opisa besedila, t.i. MSD oznake, ki za vsako pojavnico hranijo podatke o njenih lastnostih, kot so število, oseba, sklon, spol ipd. Format MSD oznake je definiran s standardom MultextEast [26]. Za vsako pojavnico je v drevesnici na voljo tudi lema.

Sledi definicija pojavnic in vozlišč v drevesnici SDT.

Definicija 3. Pojavnica je trojka $t = (f, m, e)$, kjer je f njena oblika, m predstavlja MSD oznako in e lemo.

Definicija 4. Vozlišče je trojka $\nu = (t, \pi, l)$, kjer je t pripadajoča pojavnica, π oče in l oznaka povezave iz očeta do vozlišča ν.

Polni opis besede ‘ki’ v drevesu na sliki 2.2 je tako (‘ki’, ‘Css’, ‘ki’). Pripadajoče vozlišče opišemo kot ((‘ki’, ‘Css’, ‘ki’), ‘AuxC’, π), kjer π predstavlja očeta. V doktorski disertaciji so pojavnice označene z malimi latinskimi, vozlišča pa z malimi grškimi črkami. Kjer ni dvoumno, je vozlišče poimenovano s pojavnico, ki jo predstavlja.

Potrebna je še definicija nekaterih posebnih tipov pojavnic. Naj bo $pos(t, n)$ funkcija, ki vrne vrednost na n-tem mestu v MSD oznaki pojavnice t. Tipi pojavnic t, ki so predstavljeni z vozliščem $\nu = (t, \pi, l)$, so prikazani v tabeli 4.1. Za točnejši opis MSD oznak glej Dodatek.

Tabela 4.1: Tipi pojavnic.

Tip pojavnice	Pogoj
Priredni veznik	$pos(t,1) =$ 'C' in $pos(t,2) =$ 'c'
Podredni veznik	$pos(t,1) =$ 'C' in $pos(t,2) =$'s'
Določni glagol	$pos(t,1) =$ 'V' in $pos(t,3) \neq$ 'n'
Oziralni zaimek	$pos(t,1) =$ 'R'
Pomožni glagol	$l =$ 'AuxV'

4.1 Definicije stavkov

V drevesnici SDT smo identificirali tri vrste stavkov, ki so predstavljeni kot poddrevesa v odvisnostnem drevesu povedi:

- podredni stavki, začenši s podrednim veznikom - tip 1,
- podredni stavki, ki se ne začnejo s podrednim veznikom - tip 2,
- priredni stavki.

Definicija korena stavčnega priredja:

Definicija 5. Vozlišče ν z oznako l je *koren stavčnega priredja*, če je $l =$ 'Coord' in je vsaj eden od otrok nepomožni določni glagol ali drug koren stavčnega priredja.

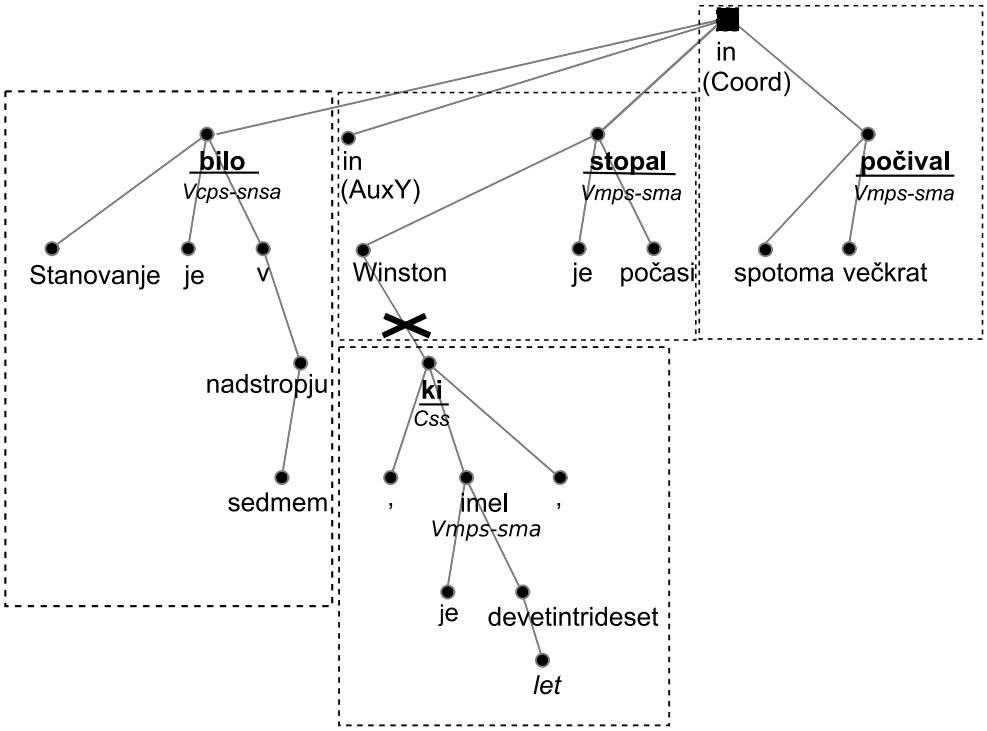
Zgornja definicija je rekurzivna. Na njej temeljijo še definicije korenov poddreves, ki predstavljajo vsakega od treh tipov stavkov.

Definicija 6. Vozlišče ν, ki predstavlja pojavnico t, je *koren podrednega stavka tipa 1*, če je pojavnica t podredni veznik in je eden od otrok nepomožni določni glagol ali koren stavčnega priredja.

Definicija 7. Vozlišče ν, ki predstavlja pojavnico t, je *koren podrednega stavka tipa 2*, če je pojavnica t nepomožni določni glagol, katerega oče ni niti podredni veznik niti koren stavčnega priredja niti beseda z lemo 'biti'.

Definicija 8. Vozlišče ν, ki predstavlja pojavnico t, je *koren prirednega stavka*, če je pojavnica t nepomožni določni glagol in je njen oče koren stavčnega priredja.

Na sliki 4.1 so besede stavčnih korenov podčrtane. Vozlišča besed 'bilo', 'stopal' in 'počival' so koreni prirednih stavkov. Vozlišče besede 'ki' je koren podrednega stavka tipa 1.



Slika 4.1: Stavčna struktura drevesa s slike 2.2. Stavki so označeni s črtkanimi pravokotniki. Zaradi jasnosti predstavitve so prikazane samo tiste MSD oznake in oznake povezav, ki so povezane s pravili za definicijo stavkov.

Poddrevesa, ki so določena s koreni stavkov, nato obrežemo. Naj bo T poddrevo, ki ga opazujemo (poddrevo s korenom 'stopal' na sliki 4.1). Naj bo K množica vseh korenov prirednih stavkov in stavčnih priredij, ki so potomci korena poddrevesa T. Poddrevo T je treba obrezati pri vseh vozliščih $\nu \in K$ (označeno s križcem na sliki 4.1). Z obrezovanjem so podredni stavki definirani – pojavnice (obrezanega) poddrevesa predstavljajo podredni stavek.

Za dokončno definicijo prirednih stavkov je treba vključiti dodatna vozlišča k (obrezanemu) poddrevesu, ki ga določa stavčni koren. Naj bo ρ koren stavčnega priredja (s kvadratom označeno vozlišče 'in' na sliki 4.1). Naj bo $(\nu_{1,1}, \nu_{1,2}, ..., \beta_1, \nu_{2,1}, ..., \beta_2, ..., \nu_{n,1}, ..., \beta_n, \nu_{n+1,1}, ..., \nu_{n+1,k})$ zaporedje otrok vozlišča ρ, pri čemer vozlišča β_i (vozlišča 'bilo', 'stopal' in 'počival' na sliki 4.1) predstavljajo korene prirednih stavkov in vozlišča $\nu_{i,j}$ predstavljajo ostale otroke (vozlišče 'in' z oznako 'AuxY' na sliki 4.1). Skrajno desni stavek sestavljajo naslednja vozlišča:

- poddrevo vozlišča β_n (na sliki 4.1, poddrevo vozlišča 'počival'),

- poddrevesa vozlišč $\nu_{n,j}$ in $\nu_{n+1,j}$ (na sliki 4.1 niso prisotna) in
- vozlišče ρ.

Ostali stavki priredja so sestavljeni iz naslednjih poddreves:

- poddrevo vozlišča β_i (na primer, poddrevo vozlišča 'stopal' na sliki 4.1) in
- poddrevesa vozlišč $\nu_{i,j}$ (na sliki 4.1 poddrevo – v tem primeru le eno vozlišče – vozlišča 'in' z oznako povezave 'AuxY'.

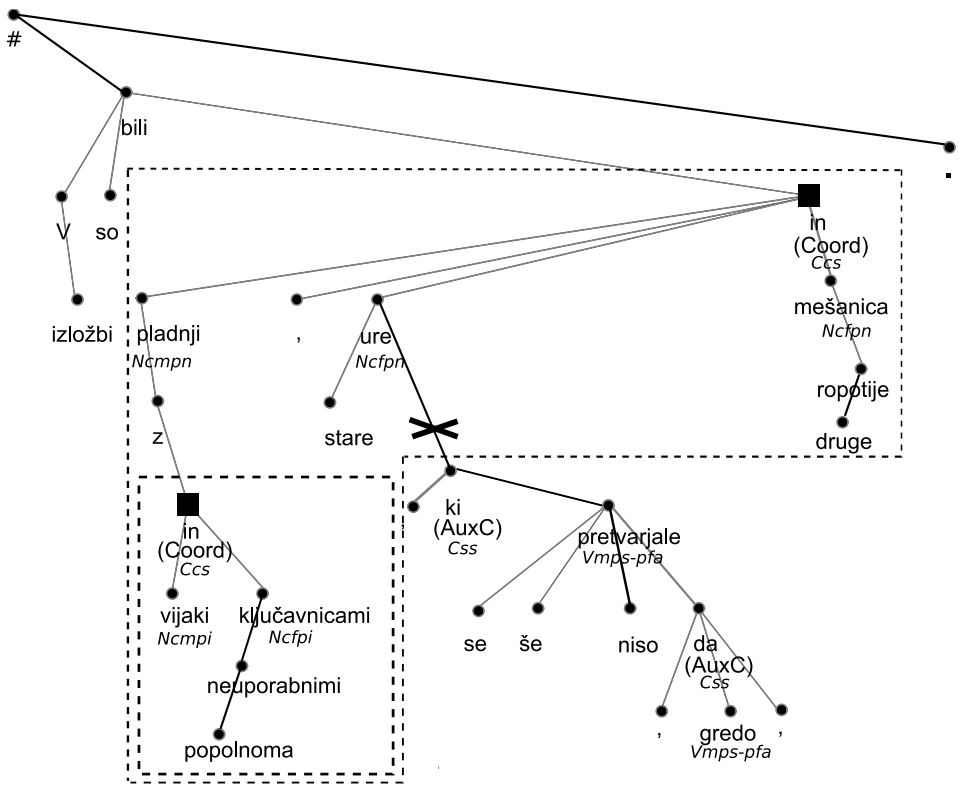
4.2 Definicija naštevanja

Naštevanja (znotrajstavčna koordinacija, angl. intraclausal coordination) so v odvisnostnem drevesu predstavljena kot poddrevesa, podobno kot stavki. Naštevanje je definirano s pomočjo njegovega korena.

Definicija 9. Vozlišče ν z oznako l je *koren naštevanja*, če je l = 'Coord' in noben od otrok ni določni glagol.

Obrezovanje drevesa poteka podobno kot pri stavkih: poddrevo se obreže pri vseh korenih stavčnih priredij in korenih podrednih stavkov. Pojavnice vozlišč, ki ostanejo v poddrevesu, po obrezovanju predstavljajo naštevanje.

Odvisnostno drevo na sliki 4.2 vsebuje dve poddrevesi naštevanj. Njuna korena sta označena s kvadratnimi vozlišči. Malo poddrevo se nahaja znotraj večjega. Točka obrezovanja večjega drevesa je označena s križcem. Vozlišča naštevanj so omejena s črtkanimi črtami. Drevo je razčlenitev povedi "V izložbi so bili pladnji z vijaki in popolnoma neuporabnimi ključavnicami, stare ure, ki se še pretvarjale niso, da gredo, in mešanica druge ropotije."



V	izložbi	so	bili	pladnji	z	vijaki	in	popolnoma	neuporabnimi	ključavnicami	,	stare		
Spsl	*Ncfsl*	*Vcip3p--n*	*Vmps-pma*	*Ncmpn*	*Spsi*	*Ncmpi*	*Ccs*	*Rgp*	*Afpfpi*	*Ncfpi*	*Z*	*Afpnpn*		

ure	,	ki	se	še	pretvarjale	niso	,	da	gredo	,	in	mešanica	druge	ropotije	.
Ncnpn	*Z*	*Css*	*Px------y*	*Q*	*Vmps-pfa*	*Vcip3p--y*	*Z*	*Css*	*Vmip3p--n*	*Z*	*Ccs*	*Ncfsn*	*Pi-fsg----a*	*Ncfsg*	*Z*

Slika 4.2: Dve naštevanji, omejeni s črtkanimi črtami. Manjše se nahaja znotraj večjega. Prikazane so samo oznake povezav (v oklepajih) in MSD oznake (poševno), ki so potrebne za primer definicije.

Poglavje 5

ARISiN, algoritem za razčlenjevanje z iskanjem stavkov in naštevanj

Poglavje podaja opis novega algoritma za razčlenjevanje ARISiN, ki vsebuje nove metode za iskanje stavkov in naštevanj temelječe na hevrističnih pravilih in strojnem učenju, podobno kot [52, 51]. Algoritem ARISiN vključuje več različnih osnovnih razčlenjevalnih modelov, ki jih izdelamo tako, da poljubni jezikovno neodvisni odvisnostni razčlenjevalnik naučimo na drevesnici ciljnega jezika. Poleg tega je bil v okviru algoritma ARISiN razvit še nov razčlenjevalnik s pravili. Algoritem ARISiN je sestavljen iz dveh faz:

1. Prva faza: Iskanje in redukcija stavkov ter naštevanj,

2. Druga faza: Gradnja odvisnostnega drevesa povedi.

V poglavju sta najprej opisani obe fazi algoritma, nato je opisan postopek priprave učnih množic za strojne klasifikatorje, uporabljene v prvi fazi in razčlenjevalne modele uporabljene v drugi fazi. Na koncu je podan še opis razčlenjevalnika s pravili.

5.1 Prva faza: iskanje in redukcija stavkov ter naštevanj

V tej fazi algoritem ARISiN poved razstavi na stavke in naštevanja (glej algoritem 7). Najprej poved razdeli na segmente, kot je obrazloženo v razdelku 3.3. Nato poišče naštevanja in jih reducira. Ker se segmentna struktura zaradi redukcije naštevanj spremeni, ponovno razdeli delno reducirano poved na segmente ter na koncu poišče in reducira še stavke. Opisan postopek se ponavlja, dokler algoritem uspe najti stavke in naštevanja oziroma dokler v povedi obstaja več kot en glagolski segment.

Algoritem 7 Prva faza: Iskanje in redukcija stavkov in naštevanj.

1: **repeat**
2: $segmentiraj()$
3: $boolNaselNastevanje := najdiNastevanja()$
4: $segmentiraj()$
5: $boolNaselStavek := najdiStavke()$
6: **until not** ($boolNaselNastevanje$ **or** $boolNaselStavek$)

5.1.1 Algoritem za iskanje naštevanj

Algoritem deluje z naštevanji, kjer so glavne besede (angl. head words) predlogi, samostalniki ali pridevniki. V prvem koraku poišče kandidate za glavne besede naštevanj. V drugem koraku neprimerne kandidate izloči. Na koncu iz glavnih besed naredi naštevanja in jih reducira v meta pojavnico (glej algoritem 8).

Algoritem 8 Funkcija najdiNastevanja().

1: **for all** $vrsta \in \{predlog, samostalnik, pridevnik\}$ **do**
2: $najdiKandidate(vrsta)$
3: $filtrirajKandidate(vrsta)$
4: $reduciraj(vrsta)$
5: **end for**

Naj bo $(c_1, t_{1,1}, ..., t_{1,j}, ..., c_i, t_{i,1}, ..., t_{i,j}, ..., c_n)$ poljubno neprekinjeno zaporedje pojavnic v povedi. Naj bo $(\mathcal{S}_1, \mathcal{D}_1, ..., \mathcal{S}_n, \mathcal{D}_n)$ segmentacija celotne povedi. Naj bo $cat(t)$ funkcija, ki vrne vrsto pojavnice t. Naj bo $case(t)$ funkcija, ki vrne sklon pojavnice t, če je pojavnica samostalnik ali pridevnik. Če je pojavnica t predlog, $case(t)$ vrne sklon samostalnika, ki se veže na predlog. V prvem koraku, pri iskanju kandidatov (algoritem 8, $najdiKandidate()$), algoritem uporabi naslednje hevristično pravilo:

Definicija 10. Hevristično pravilo A: če $\forall i, j : cat(c_i) = cat(c_j) \wedge case(c_i) = case(c_j)$, potem pojavnice $(c_1, ..., c_n)$ predstavljajo kandidata za skupino glavnih besed.

V drugem koraku algoritem množico kandidatov filtrira (algoritem 8, $filtrirajKandidate()$). Najprej jih pretvori v pare sosednjih glavnih besed (c_i, c_{i+1}), $0 < i < n$. Vsak par preveri z novim hevurističnim pravilom, ki je vsebuje tri pogoje. Naj funkcije $cc(t)$, $sc(t)$, $rp(t)$ in $fv(t)$ vrnejo pravilno vrednost (true), če je pojavnica t priredni veznik, podredni veznik, oziralni zaimek ali določni glagol in nepravilno vrednost (false) sicer.

Definicija 11. Pogoj I drži, če $\forall t_{i,j} = (f_{i,j}, m_{i,j}, e_{i,j})$: $f_{i,j} \notin \{\text{-}, :, ;, (,)\} \wedge \neg sc(t_{i,j}) \wedge \neg fv(t_{i,j}) \wedge \neg rp(t_{i,j})$.

Pogoj I zahteva, da med glavnimi besedami ne sme biti dvopičij, podpičij, pomišljajev, vezajev, oklepajev, določnih glagolov, oziralnih zaimkov in podrednih veznikov.

Definicija 12. Pogoj II drži, če $\forall i, i < n \; \exists! t_{i,j} = (f_{i,j}, m_{i,j}, e_{i,j}) : f_{i,j} = \text{‘,’} \vee cc(t_{i,j})$.

Pogoj II zahteva, da je izmed vseh pojavnic med dvema sosednjima glavnima beseda natanko en mejnik med segmenti $\mathcal{D}_k = (t_{i,j})$, ki vsebuje natanko eno pojavnico. Ta pojavnica mora biti vejica ali priredni veznik. Naj bo $\mathcal{D}_k$ mejnik med segmentoma $\mathcal{S}_k$ in $\mathcal{S}_{k+1}$.

Definicija 13. Pogoj III drži, če $\forall s, s \in \mathcal{S}_k : \neg fv(s) \vee \forall s, s \in \mathcal{S}_{k+1} : \neg fv(s)$.

Pogoj III zahteva, da je vsaj eden od sosednjih segmentov vsakega mejnika neglagolski segment.

Definicija 14. Hevristično pravilo B: če katerikoli od pogojev I, II ali III za katerikoli par (c_i, c_{i+1}) ni izpolnjen, potem skupino besed $(c_1, ..., c_n)$ algoritem izloči iz množice kandidatov.

V povedi na sliki 5.1 je en kandidat sestavljen iz dveh glavnih besed (‘vijaki’, ‘ključavnicami’). Obstaja sicer še skupina treh besed iste vrste in v istem sklonu (‘pladnji’, ‘ure’, ‘mešanica’), ki se skladajo s hevrističnim pravilom A, vendar jih hevristično pravilo B izloči iz množice kandidatov; med njimi je več nedovoljenih veznikov, vejic, določnih glagolov.

Kandidate nato algoritem strojno klasificira in sicer vsak par (c_i, c_{i+1}) posebej. Če je vsaj en par klasificiran negativno, algoritem zavrže celotno skupino glavnih besed, kateri pripada par. Z uporabo programskega paketa WEKA [83] so bili razviti trije različni klasifikatorji, za vsako vrsto besed, ki jo algoritem ARISiN obvlada, svojega. Klasifikatorji so tipa AdaBoost [29] z J48 drevesi kot osnovnim klasifikatorjem. Vir učnih primerov je bila drevesnica SDT. Za opis primerov z atributi algoritem uporabi pojavnice med glavnima besedama v paru, ki so razdeljene na dve skupini: pojavnice med levo glavno besedo in vejico/prirednim veznikom (skupina A) ter pojavnice med vejico/prirednim veznikom in desno glavno besedo (skupina B). Naslednji atributi so v

V	izložbi	so	bili	pladnji	z	**vijaki**	[in]	popolnoma	neuporabnimi	**ključavnicami**	[,]
				Ncmpn		*Ncmpi*	*Ccs*	*Rgp*		*Afpfpi*	*Ncfpi*

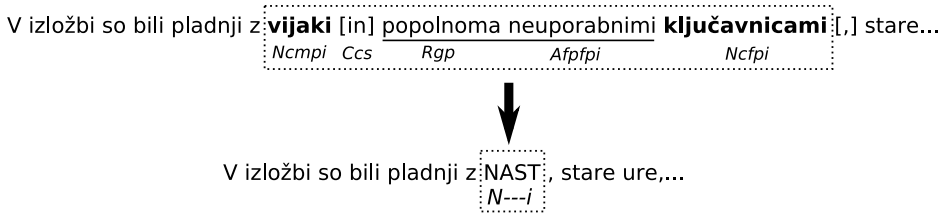
stare	ure	[, ki]	se	še	pretvarjale	niso	[, da]	gredo	[, in]	mešanica	druge	ropotije.
Ncfpn	*Css*				*Vmps-pfa*	*Vcip3p--y*	*Css*	*Vmps-pfa*	*Ccs*	*Ncfpn*		

Slika 5.1: V stavku je en kandidat za skupino sestavljeno iz dveh glavnih besed (krepki tisk). Tri podčrtane besede se skladajo hevrističnim pravilom A, vendar za njih ne drži hevristično pravilo B. MSD oznake (poševno) so prikazane pod pojavnicami.

uporabi za opis skupine (v celoti se torej vsak atribut v opisu pojavlja dvakrat, kot je zapisano v oklepajih):

- prisotnost predloga/pridevnika v skupini (štirje atributi, dvojiške vrednosti),
- prisotnost z glavno besedo ujemajočega/neujemajočega samostalnika/pridevnika v sklonu, spolu in številu (osem atributov, dvojiške vrednosti),
- število besed v skupini (dva atributa, vrednosti: 0, 1, 2, >2) in
- razred (en atribut, dvojiške vrednosti).

Če so vsi pari skupine klasificirani pozitivno, algoritem nadaljuje s tretjim korakom, z redukcijo (algoritem 8, *reduciraj*()). Zaporedja pojavnic, ki se začenjano s prvo glavno besedo in končujejo z zadnjo glavno besedo zamenja z meta pojavnico z imenom 'NAST' (naštevanje). Priredi ji MSD oznako z enako besedno vrsto in sklonom kot pri glavnih besedah. Na sliki 5.2 je prikazana redukcija naštevanja. V tabeli 5.1 se nahaja atributni opis para glavnih besed s slike 5.2. V zgornjem levem delu so predstavljeni atributi, ki pripadajo pojavnicam iz skupine A, na desni pa atributi, izvirajoči iz skupine pojavnic B. Spodaj je izpisan še atribut – razred.



Slika 5.2: Redukcija naštevanja. Glavne besede so izpisane krepko, skupina B je podčrtana. Skupina A je v tem primeru prazna. MSD oznake so izpisane pod besedami.

5.1.2 Algoritem za iskanje stavkov

Stavek je sestavljen iz enega ali več segmentov in mejnikov. Na sliki 5.3 prvi, drugi, tretji in zadnji segment s predhodnimi mejniki sestavljajo en stavek. Četrti in peti segment vsak s svojim predhodnim mejnikom predstavljata dva enosegmentna stavka – segmente tega tipa išče algoritem za iskanje stavkov. Iskanje stavkov poteka v povedih, ki vsebujejo več kot en glagolski segment (glej algoritem 9).

Naj funkcija $vs(\mathcal{S})$ vrne pravilno vrednost, če je $\mathcal{S}$ glagolski segment. Naj bo $(\mathcal{S}_1, \mathcal{D}_1, ..., \mathcal{S}_n, \mathcal{D}_n)$ segmentacija povedi. Algoritem najprej za vsak glagolski segment $\mathcal{S}_i$ preveri naslednji pogoj (algoritem 9, *sosednjiGlagSegmenti*()):

Tabela 5.1: Atributni opis para glavnih besed, ki sestavljata naštevanje na sliki 5.2.

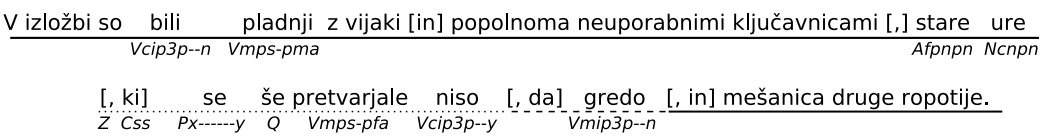
Skupina A: [prazno]

Atribut	Vrednost
Pridevnik, skupina A	0
Predlog, skupina A	0
Ujemajoči sam., skupina A	0
Neujemajoči sam., skupina A	0
Ujemajoči prid., skupina A	0
Neujemajoči prid., skupina A	0
Velikost skupine, skupina A	0

Skupina B: 'popolnoma neuporabnimi'

Atribut	Vrednost
Pridevnik, skupina B	1
Predlog, skupina B	0
Ujemajoči sam., skupina A	0
Neujemajoči sam., skupina A	0
Ujemajoči prid., skupina A	1
Neujemajoči prid., skupina A	0
Velikost skupine, skupina B	2

Atribut	Vrednost
Razred	1



Slika 5.3: V povedi so trije stavki. Dva stavka, podčrtana s črtkanimi in pikčastimi črtami, sta vrinjena v tretjem, ki je podčrtan z neprekinjeno črto.

Algoritem 9 Funkcija najdiStavke() (glej algoritem 7).

```
1: for all S, S je glagolski segment do
2:     if sosednjiGlagSegmenti(S) ∨ klasificiraj(S) then
3:         reduciraj(S)
4:     end if
5: end for
```

Definicija 15. Pogoj IV drži, če $(vs(\mathcal{S}_{i-2}) \wedge i \geq 3 \vee i < 3) \quad \wedge \quad (vs(\mathcal{S}_{i-1}) \wedge i \geq 2 \vee i < 2) \quad \wedge \quad (vs(\mathcal{S}_{i+1}) \wedge i \leq n-1 \vee i = n) \quad \wedge \quad (vs(\mathcal{S}_{i+2}) \wedge i \leq n-2 \vee i \geq n-1)$.

Če sta dva predhodna in dva naslednja segmenta glagolska, potem pogoj IV drži. Če algoritem preverja prvi, drugi, predzadnji ali zadnji segment v povedi, manjkajoče sosednje segmente obravnava kot glagolske segmente. Če pogoj IV ne drži, algoritem glagolski segment strojno klasificira (algoritem 9, $klasificiraj()$). Razvita sta bila dva klasifikatorja, poimenovana Alfa in Beta. Oba temeljita na algoritmu AdaBoost z drevesi

J48 kot osnovnim klasifikatorjem. Klasifikator Alfa je namenjen segmentom, katerih oba sosednja segmenta sta ravno tako glagolska. Klasifikator Beta je v uporabi za ostale primere. V atributni opis je poleg obravnavanega segmenta $\mathcal{S}_i$ vključen še predhodni mejnik $\mathcal{D}_{i-1}$ ter pari mejnik/segment $\mathcal{D}_{i-2}/\mathcal{S}_{i-1}$, $\mathcal{D}_{i-3}/\mathcal{S}_{i-2}$, $\mathcal{D}_i/\mathcal{S}_{i+1}$ in $\mathcal{D}_{i+1}/\mathcal{S}_{i+2}$. Če kateri izmed opisanih elementov ne obstaja, atributi dobijo vrednost 'nedefinirano'. Vsak par mejnik/segment $\mathcal{D}_{k-1}/\mathcal{S}_k$ je opisan z naslednjimi atributi:

- Prisotnost prirednega veznika (dvojiške vrednosti).
- Prisotnost podrednega veznika (dvojiške vrednosti).
- Prisotnost ločila (vrednosti: 'brez', 'vejica', 'dvopicje_ali_podpicje', 'ostalo').
- Prisotnost oziralnega zaimka (dvojiške vrednosti).
- Pomožni glagol se pojavlja pred deležnikom (vrednosti: 'da', 'ne', 'ni_definirano').
- Tip segmenta (vrednosti: 'glagolski', 'neglagolski').
- Možen obstoj križajočega naštevanja. Naštevanje je križajoče v primeru, da nekatere glavne besede naštevanja ležijo v opisovanem segmentu, nekatere pa v sosednjih segmentih. Za iskanje takih skupin glavnih besed se uporabi hevristični pravili A in B, pri čemer nekatere pogoje opustimo. S pravilom A se preveri tudi prislove, nedoločne glagole in številnike, a se pri tem ne upošteva sklona, ker pri teh besednih vrstah ne obstaja. Pri pravilu B lahko mejnik med glavnima besedama predstavlja tudi podpičje.
- Razred (dvojiške vrednosti).

V vektorju atributnih vrednosti se torej vsak od zgoraj opisanih atributov pojavlja petkrat razen atribut 'razred'. V tabeli 5.2 je prikazan atributni opis segmenta 'se še pretvarjale niso' s slike 5.3. Polni atributni opis tega segmenta zajema informacije iz naslednjih parov mejnik/segment:

- [in] / 'popolnoma neuporabnimi ključavnicami',
- [,] / 'stare ure',
- [, ki] / 'se še pretvarjale niso',
- [, da] / 'gredo',
- [, in] / 'mešanica druge ropotije'.

Tabela 5.2: Atributni opis segmenta ‘se še pretvarjale niso’ s slike 5.3. Zaradi jasnosti predstavitve so prikazane samo vrednosti atributov, ki pripadajo paroma mejnik/segment [, ki] / ‘se še pretvarjale niso’ in sosednjemu paru [,] / ‘stare ure’.

[, ki] / ‘se še pretvarjale niso’

Atribut	Vrednost
Priredni veznik	0
Podredni veznik	1
Ločilo	‘vejica’
Oziralni zaimek	0
Pomož. gl. pred delež.	‘ne’
Tip segmenta	‘glagolski’
Križajoče naštevanje	0

[,] / ‘stare ure’

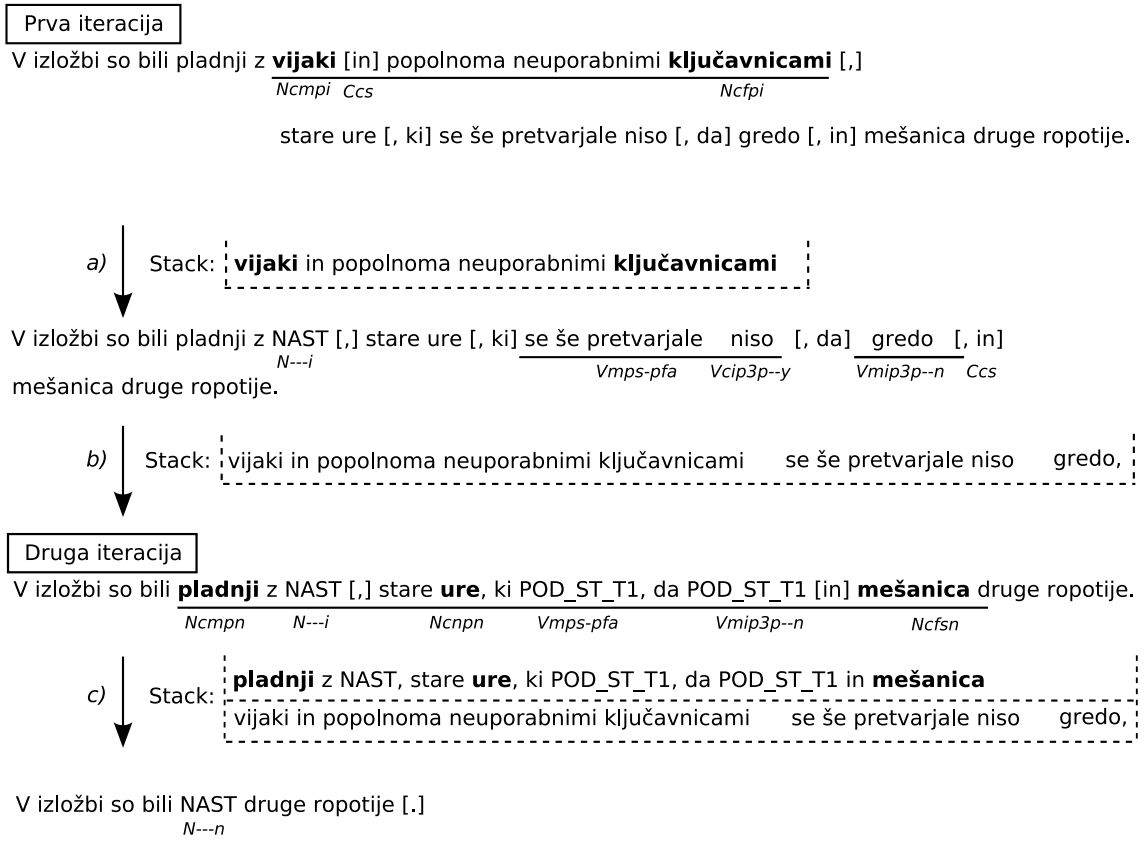
Atribut	Vrednost
Priredni veznik	0
Podredni veznik	0
Ločilo	‘vejica’
Oziralni zaimek	0
Pomož. gl. pred delež.	‘ni_definirano’
Tip segmenta	‘neglagolski’
Križajoče naštevanje	0

Atribut	Vrednost
Razred	1

Če je segment klasificiran pozitivno ali pogoj IV drži, pomeni, da je algoritem našel enosegmentni stavek, ki ga lahko reducira (algoritem 9, $reduciraj()$). Čeprav celoten stavek sestavljata segment in mejnik pred njim, algoritem mejnika ne reducira; mejniki namreč igrajo pomembno vlogo v drugi fazi, pri gradnji odvisnostnega drevesa. Posebno pozornost je treba posvetiti mejniku desno od reduciranega segmenta. Če se ta mejnik začne z vejico, kateri sledi veznik vezalnega, stopnjevalnega ali ločnega priredja, vejica pomeni konec stavka. V tem primeru algoritem vejico reducira skupaj z obravnavanim segmentom.

Na sliki 5.4 je prikazano, kako prva faza algoritma ARISiN obdela poved s slike 4.2. Izvedeta se dve iteraciji. V koraku a) algoritem reducira naštevanje s slike 5.2. Reducirana zaporedja algoritem odloži na sklad za kasnejšo obdelavo v drugi fazi. Zaporedja, ki so reducirana v isti iteraciji, se nahajajo na istem nivoju sklada. V koraku b) algoritem najde dva enosegmentna stavka. Segmente v povedi nadomesti z meta pojavnicama. Skupaj z drugim segmentom reducira tudi vejico iz naslednjega mejnika, kajti vejici sledi veznik vezalnega priredja. Meta pojavnici algoritem priredi MSD oznako glavnega glagola v segmentu. Ime meta pojavnice določijo hevristična pravila. Naj bo $\mathcal{S}_i = (s_1, ..., s_n)$ segment, ki ga nadomesti meta pojavnica. Pri tem sta $\mathcal{D}_{i-1} = (a_1, ..., a_k)$ in $\mathcal{D}_i = (b_1, ..., b_l)$ mejnika levo in desno od segmenta.

Definicija 16. Hevristično pravilo C: če $\exists a_i : sc(a_i) \wedge \neg cc(b_1)$, potem je ime meta



Slika 5.4: Prva faza algoritma ARISiN. Reducirana zaporedja so podčrtana, glavne besede naštevanj so izpisane krepko, mejniki med segmenti so omejeni z oglatimi oklepaji, MSD oznake so izpisane poševno. Meta pojavnice naštevanj so poimenovane 'NAST', meta pojavnice stavkov pa 'POD_ST_T1', 'POD_ST_T2' ali 'PRIR_ST'.

pojavnice 'POD_ST_T1'.

Definicija 17. Hevristično pravilo Č: če $\forall a_i : \neg sc(a_i) \wedge \exists s_i : rp(s_i)$, potem je ime meta pojavnice 'POD_ST_T2'.

Če mejnik $\mathcal{D}_{i-1}$ vsebuje podredni veznik in se mejnik $\mathcal{D}_i$ ne začne s prirednim veznikom, algoritem meta pojavnico poimenuje 'POD_ST_T1' (podredni stavek tipa 1). Če mejnik $\mathcal{D}_{i-1}$ ne vsebuje podrednega veznika in segment vsebuje oziralni zaimek, meta pojavnica dobi ime 'POD_ST_T2' (podredni stavek tipa 2). Če noben od pogojev hevrističnih pravil C ali Č ne drži, algoritem meta pojavnici priredi ime 'PRIR_ST' (priredni stavek).

V naslednjih iteracijah med segmentacijo stavka in iskanjem naštevanj algoritem začasno odstrani pojavnice mejnikov tik pred meta pojavnicami. Pojavnice nimajo vloge mejnika niti ne vplivajo na pogoje hevrističnega pravila B. Na sliki 5.4 v koraku b) sta dva taka mejnika: [, ki] in [, da]. V koraku c), kjer zaporedje pojavnic ', stare ure, ki POD_ST_T1, da POD_ST_T1' predstavlja en sam segment, algoritem najde še eno

naštevanje. V povedi ostane samo še en glagolski segment in ni več možno najti naštevanj, zato se faza iskanja in redukcije stavkov ter naštevanj zaključi.

5.2 Druga faza: gradnja odvisnostnih dreves

V tej fazi algoritem ARISiN zaporedja, reducirana v prvi fazi, razčleni z enim od treh osnovnih razčlenjevalnih modelov: začetni razčlenjevalni model, stavčni razčlenjevalni model ali razčlenjevalni model za naštevanja. Tako dobljena odvisnostna drevesa algoritem združi v končno odvisnostno drevo povedi. Na sliki 5.5 je orisano delovanje celotne faze, ki predstavlja nadaljevanje obdelave besedila, prikazane na sliki 5.4.

Na začetku faze algoritem ARISiN zaporedja pojavnic, ki v prvi fazi ostanejo nereducirane, razčleni z začetnim razčlenjevalnim modelom. Omenimo, da pred razčlenjevanjem algoritem doda na začetek povedi tehnično vozlišče '#'. S tem algoritem zgradi začetno verzijo odvisnostnega drevesa povedi (slika 5.5a). V drevesu je možno odkriti določene napake. V takem primeru algoritem drevo zavrže, uporabi razčlenjevalnik s pravili in na novo zgradi začetno drevo.

Faza se nadaljuje z iteracijo, ki obdeluje zaporedja s sklada (glej algoritem 10). V prvem koraku iteracije algoritem vzame z zgornjega nivoja sklada vsa zaporedja (slika 5.5b). Nato zaporedja združi s potomci vozlišč pripadajočih meta pojavnic v drevesu povedi (slika 5.5c). Razširjena zaporedja algoritem razčleni s stavčnim razčlenjevalnim modelom oziroma z razčlenjevalnim modelom za naštevanja (slika 5.5č). V drevesu povedi poddrevesa meta pojavnic algoritem zamenja z novo izdelanimi poddrevesi (slika 5.5d).

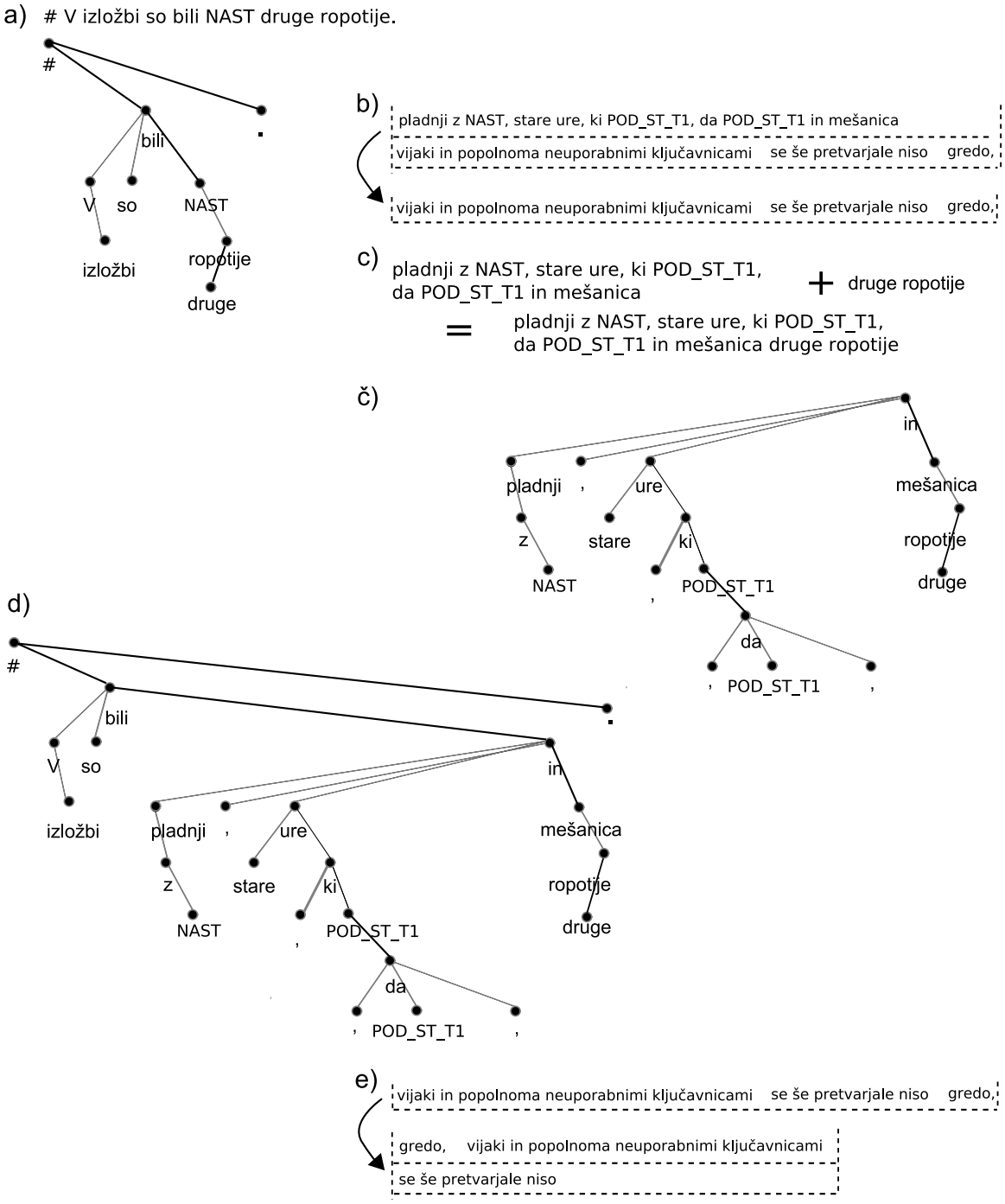
Nato algoritem preišče zaporedja, ki so trenutno na skladu. Vozlišča meta pojavnic nekaterih zaporedij, ki se sicer nahajajo na enakem nivoju sklada, so v drevesu povedi lahko v nadrejeno-podrejenem odnosu. Tak primer sta obe vozlišči 'POD_ST_T1' na sliki 5.5d. V tem primeru algoritem sklad preuredi, tako da zaporedja z meta pojavnicami bliže listom razčleni najprej (slika 5.5e).

Algoritem se zaključi, ko je sklad prazen. Na sliki 5.5 je prikazana samo prva iteracija. Algoritem za dokončno izgradnjo odvisnostnega drevesa rabi še dve iteraciji.

5.3 Priprava učnih množic

Luščenje učnih primerov iz drevesnice SDT poteka v treh ločenih sklopih. Algoritem najprej poišče učne primere za klasifikatorja pri iskanju stavkov; poimenujmo to učno množico U_{k_s}. V drugem sklopu pripravi učne množice za vse razčlenjevalne modele. Poimenujmo te množice U_{r_z} (začetni razčlenjevalni model), U_{r_s} (stavčni razčlenjevalni model) in U_{r_n} (model za razčlenjevanje naštevanj). V tem sklopu najde tudi pozitivne primere za klasifikatorje skupin glavnih besed naštevanj (množico poimenujmo $U^+_{k_n}$),

V izložbi so bili pladnji z vijaki in popolnoma neuporabnimi ključavnicami, stare ure, ki se še pretvarjale niso, da gredo, in mešanica druge ropotije.



Slika 5.5: Prva iteracija gradnje odvisnostnega drevesa povedi. Na levi strani slike (a, d) je prikazana rast drevesa povedi. Desna stran (b, c, č) prikazuje, kako algoritem prazni sklad in razčlenjuje zaporedja pojavnic, reducirana v prvi fazi.

Algoritem 10 Druga faza: gradnja odvisnostnih dreves.

Vhod:

- $STACK = [(S_{1,1}, ..., S_{1,n_1}), ..., (S_{m,1}, ..., S_{m,n_m})]$: sklad z m nivoji, kjer je $S_{i,j}$ zaporedje reducirano v meta pojavnico vozlišča $\nu_{i,j}$.
- $T(\tau)$: poddrevo s korenom τ.

Izhod:

- T: odvisnostno drevo povedi.

repeat
  $(S_{i,1}, ..., S_{i,n_i}) = pop(STACK)$
  for all $(S_{i,1}, ..., S_{i,n_i})$ **do**
    $S_{i,j} := S_{i,j} \cup T(\nu_{i,j})$
    $N := parse(S_{i,j})$
    $T := replace(T, T(\nu_{i,j}), N)$
  end for
  $rearrange(STACK)$
until $STACK$ *is empty*

negativne primere za ta klasifikator pa poišče v tretjem sklopu (to množico poimenujmo $U^-_{k_n}$).

V nadaljnjih razdelkih so opisani algoritmi za pridobivanje primerov iz enega odvisnostnega drevesa. Primeri iz vseh dreves skupaj sestavljajo celotne učne množice, kar pomeni, da se nekateri primeri znotraj posameznih množic ponavljajo. Pri tem je bilo upoštevano še pravilo, da v učnih množicah klasifikatorjev en primer ne more biti hkrati pozitiven in negativen, tako da se v takih slučajih negativne primere izloči iz učnih množic.

5.3.1 Učni primeri za klasifikator stavkov

Algoritem 11 v odvisnostnem drevesu najprej poišče stavke glede na definicije, opisane v 4. Nato izdela segmentacijo povedi in za vsak stavek določi, koliko segmentov ga sestavlja. Enosegmentni stavki predstavljajo pozitivne učne primere (podčrtano na slikah 5.6 in 5.7). Negativni primeri (krepko na slikah 5.6 in 5.7) so tisti glagolski segmenti, ki predstavljajo samo del stavka, ostali deli stavka pa so segmenti drugje v povedi (ležeče na slikah 5.6 in 5.7).

Algoritem 11 Algoritem za iskanje učnih primerov klasifikatorja pri iskanju stavkov.

Vhod:

- (x, y): par poved x / odvisnostno drevo y, iz katerega algoritem poišče učne primere.
- $findClauses(y)$: funkcija, ki vrne seznam stavkov v drevesu y.
- $findSegments(x)$: funkcija, ki vrne seznam segmentov v povedi x.
- $isOneClause(s, C)$: funkcija, ki vrne pravilno, če segment s s predhodnim mejnikom sestavlja celoten stavek in napačno sicer.

Izhod:

- U^+, U^-: učni množici pozitivnih in negativnih primerov iz obravnavane povedi.

1: $C := findClauses(T)$
2: $S := findSegments(x)$
3: **for all** $s \in S$ **do**
4: **if** $isOneClause(s, C)$ **then**
5: dodaj s v U^+
6: **else**
7: dodaj s v U^-
8: **end if**
9: **end for**
10: **for all** $p^- \in U^-$ **do**
11: **if not** $\exists p^+ \in U_{k_s} : p^- = p^+$ **then**
12: dodaj p^- v U_{k_s}
13: **end if**
14: **end for**
15: dodaj U^+ v U_{k_s}

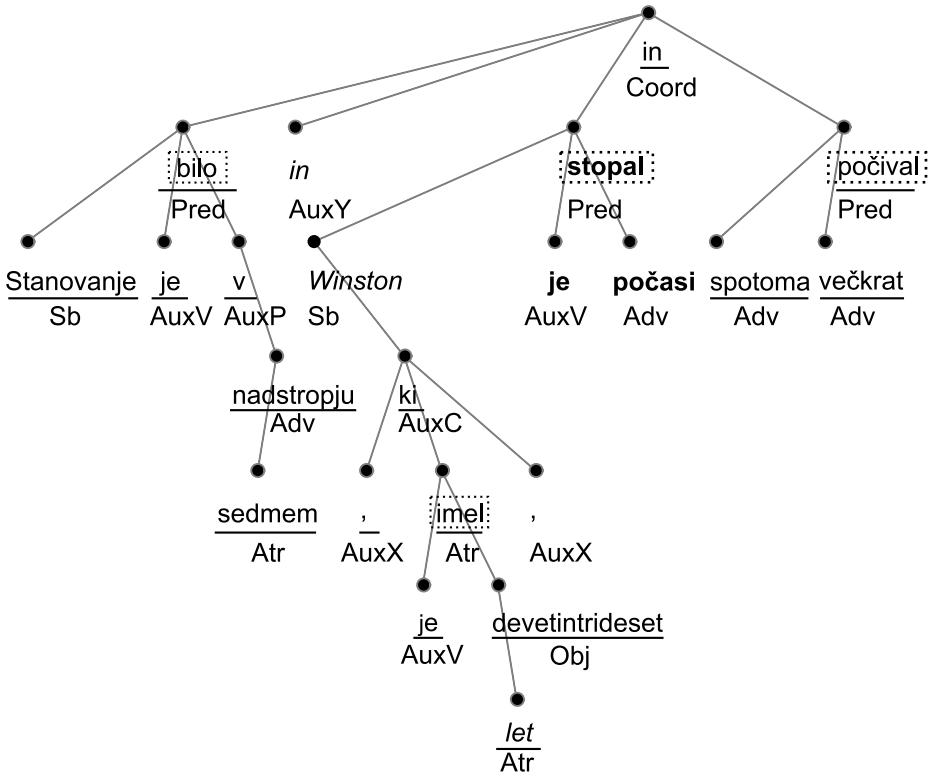
Stanovanje je bilo v sedmem nadstropju [in] *Winston* [, ki] je imel devetintrideset let [,] **je stopal počasi** [in] se vmes večkrat ustavljal.

Slika 5.6: Segmentacija povedi. Mejniki med segmenti so označeni z oglatimi oklepaji.

5.3.2 Učni primeri za razčlenjevalnike ter pozitivni učni primeri za klasifikator skupin glavnih besed

Priprava učnih množic za različne razčlenjevalne modele in pridobivanje pozitivnih primerov za klasifikatorje skupin glavnih besed potekata hkrati. Algoritem z iterativnim postopkom razgradi drevo in s tem izlušči učne primere.

V prvem koraku iteracije učni algoritem najprej doda celotno odvisnostno drevo povedi



Slika 5.7: Drevo s primeri glagolskih segmentov kot pozitivnih in negativnih primerov. Koreni poddreves stavkov so v pikčastih okvirjih.

v učno množico začetnega razčlenjevalnega modela U_{r_z} (slika 5.8a). Nato algoritem poišče korenska vozlišča vseh enostavnih stavkov in naštevanj, t.j. takih, ki ne vsebujejo drugih vrinjenih stavkov in naštevanj (slika 5.8b, na sliki 5.8a označeni s kvadratkom). Algoritem nato drevo povedi obreže (odrezana poddrevesa prikazuje slika 5.8c, na sliki 5.8a so mesta obrezovanja označena s križci). Če gre za poddrevesa naštevanj, prirednih stavkov ali podrednih stavkov tipa 2, algoritem odreže celotno poddrevo. Pri poddrevesih podrednih stavkov tipa 1 odreže le poddrevo glavnega glagola stavka. Poddrevesa doda v učni množici U_{r_s} oziroma U_{r_n}. V primeru naštevanj algoritem iz poddrevesa izlušči še pozitivni učni primer za klasifikator glavnih besed in ga doda v množico $U^+_{k_n}$. Končni rezultat obrezovanja je drevo na sliki 5.8č, ki predstavlja vhodni podatek za naslednjo iteracijo. Algoritem se zaključi, ko ni več možno najti nobenega stavka ali naštevanja.

5.3.3 Negativni učni primeri za klasifikator glavnih besed naštevanj

Negativnih primerov za klasifikator glavnih besed v drevesnici SDT ni možno poiskati neposredno, saj niso posebej označeni. V tem primeru se uporabi algoritem za iskanje

Algoritem 12 Priprava učnih množic za razčlenjevalne modele ter pridobivanje pozitivnih primerov za klasifikator glavnih besed naštevanj.

Vhod:

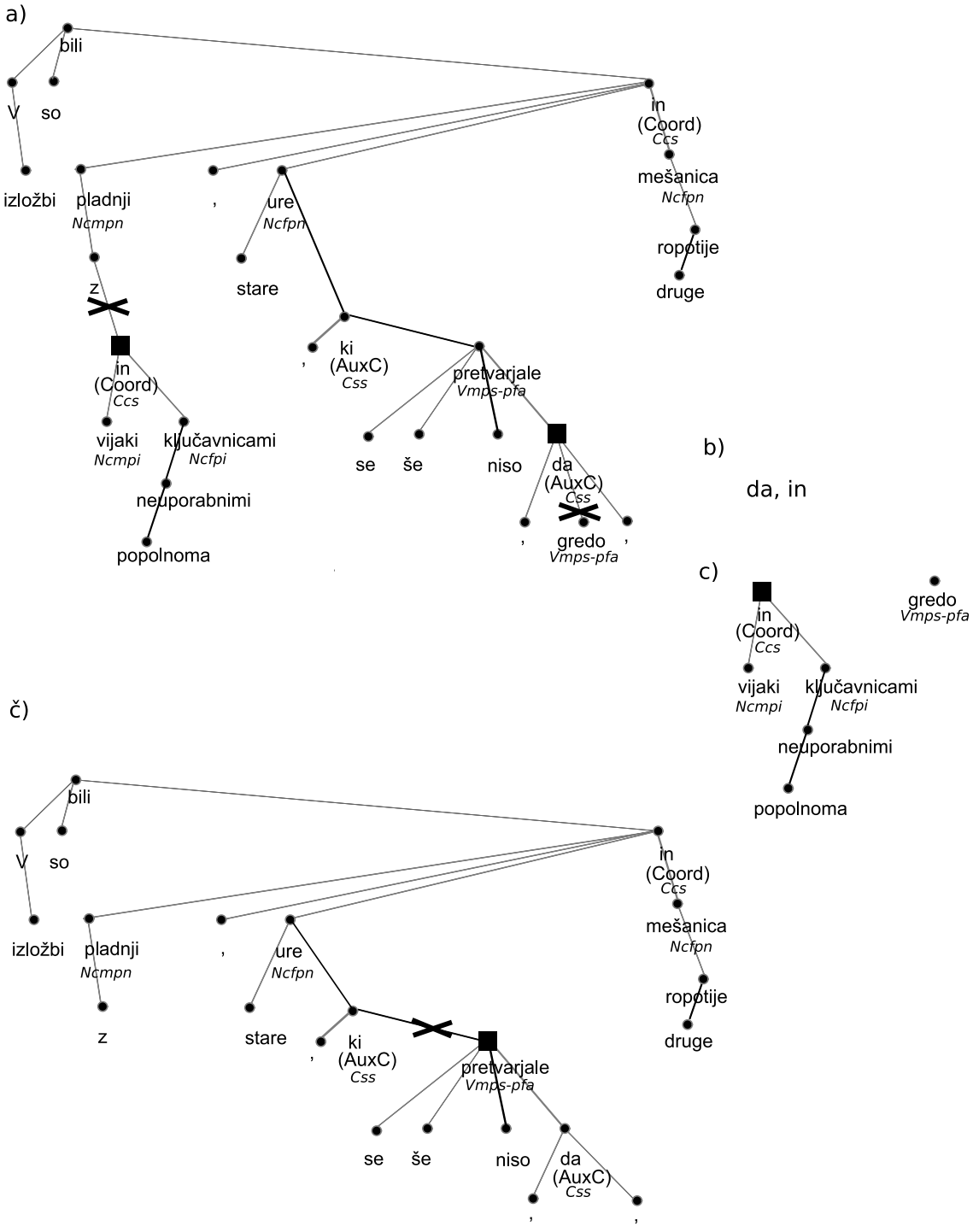
- t: odvisnostno drevo povedi.
- $np(y)$: funkcija, ki vrne seznam korenov enostavnih naštevanj in stavkov v drevesu y.
- $p(y, \nu)$: funkcija, ki obreže drevo y pri vozlišču ν. Izjema: če je vozlišče koren podrednega stavka tipa 1, funkcija obreže drevo pri določnem glagolu, ki je otrok korena. Funkcija vrne poddrevo.
- $mls(y)$: funkcija vrne seznam pozitivnih učnih primerov za klasifikator glavnih besed, ki jih izlušči iz poddrevesa y.

Izhod:

- $U_{r_z}, U_{r_s}, U_{r_n}, U^+_{k_n}$: učne množice za razčlenjevalne modele in množica pozitivnih učnih primerov za klasifikator glavnih besed naštevanj

```
1: while true do
2:     dodaj t v U_{r_z}
3:     N := np(t)
4:     if N = () then
5:         exit
6:     end if
7:     for all ν ∈ N do
8:         t_s := p(t, ν)
9:         if t_s je poddrevo naštevanja then
10:            dodaj t_s v U_{r_n}
11:            K := mls(t_s)
12:            dodaj K v U^+_{k_n}
13:        else
14:            dodaj t_s v U_{r_s}
15:        end if
16:    end for
17: end while
```

skupine glavnih besed naštevanj, opisan v razdelku 5.1.1, s tem da se najdenih skupin ne filtrira s strojnim klasifikatorjem. Iz skupin, ki ustrezajo pogojem hevrističnih pravil, se izloči tiste, ki so bile osnova za pozitivne primere (glej razdelek 5.3.2). Iz preostalih skupin besed se izlušči negativne primere in se jih doda v množico $U^-_{k_n}$. Postopek je zapisan v algoritmu 13. Na sliki 5.9 sta dve besedi, ki ustrezata hevrističnima praviloma A in B, vendar ne predstavljata skupine glavnih besed naštevanja.



Slika 5.8: Potek priprave učnih množic. Drevesa na levi strani so učni primeri za začetni razčlenjevalni model. Drevesa na desni strani sodijo v učne množice stavčnega razčlenjevalnega modela in modela za razčlenjevanje naštevanj. Iz poddreves naštevanj algoritem izlušči še učne primere za klasifikator glavnih besed naštevanj.

Algoritem 13 Algoritem za iskanje negativnih učnih primerov za klasifikator glavnih besed naštevanj.

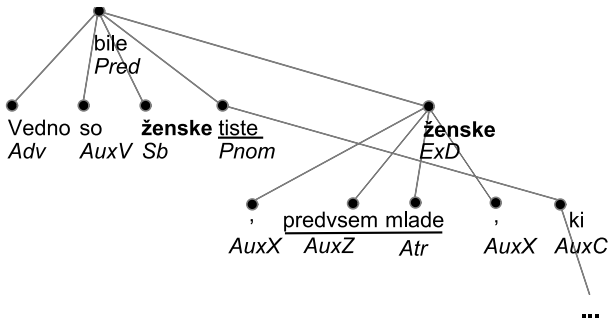
Vhod:

- t: odvisnostno drevo povedi
- $hg(y)$: funkcija, ki v drevesu y najde skupine besed, ki ustrezajo hevrističnima praviloma A in B (glej razdelek 5.1.1), ter vrne seznam učnih primerov.
- $U^+_{k_n}$: množica pozitivnih učnih primerov za klasifikator glavnih besed

Izhod:

- $U^-_{k_n}$: množica negativnih učnih primerov za klasifikator glavnih besed naštevanj

1: $P := hg(y)$
2: **for all** $p \in P$ **do**
3: **if** $p \notin U^+_{k_n}$ **then**
4: dodaj p v $U_{k_n}-$
5: **end if**
6: **end for**



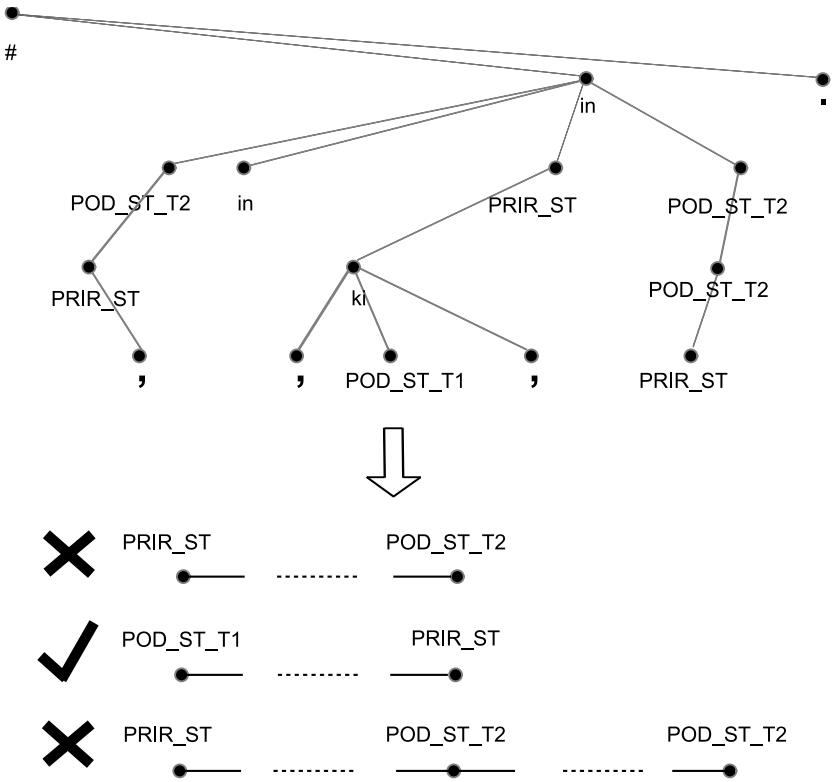
Slika 5.9: Negativni učni primer klasifikatorja skupin glavnih besed. Glavni besedi sta izpisani krepko. Skupini besed A in B, iz katerih izvirajo informacije za atributni opis primera, sta podčrtani.

5.4 Razčlenjevalnik s pravili

Razčlenjevalnik s pravili uporabimo v drugi faz in sicer takrat, kadar algoritem ARISiN zazna napake, ki jih osnovni razčlenjevalni model naredi pri gradnji začetnega drevesa. Odprava napak poteka pri drevesih, ki vsebujejo samo vozlišča ločil, veznikov in meta pojavnic. Po detekciji napake razčlenjevalnik s pravili zgradi novo začetno drevo z dvema prehodoma preko vseh pojavnic.

5.4.1 Zaznavanje napak

Algoritem ARISiN najprej analizira začetno drevo, s tem da pregleda vse poti, ki se začnejo v listih in končajo v korenu drevesa, kot je prikazano na sliki 5.10. Iz poti izloči vsa vozlišča besed in ločil, tako da ostanejo samo še vozlišča meta pojavnic. Če obstaja vsaj ena izmed poti, ki se ne konča z vozliščem meta pojavnice prirednega stavka ('PRIR_ST'), algoritem ARISiN to obravnava kot napako. Podredni stavek namreč ne more biti glavni stavek povedi, niti ne more biti v koordinaciji z drugimi prirednimi stavki, kot kaže primer na sliki 5.10. V primeru napake algoritem ARISiN začetno drevo zavrže.



Slika 5.10: Prikaz analize začetnega drevesa. V spodnjem delu slike so prikazane nekatere poti v drevesu, ki se začnejo v listih in končajo v korenu. Algoritem upošteva samo vozlišča meta pojavnic. Poti, ki nakazujeta napako v drevesu, sta označeni s križcem.

5.4.2 Novo začetno drevo – prvi prehod

Algoritem 14 v prvem prehodu obdela meta pojavnice prirednih stavkov. Če je v povedi samo ena taka pojavnica, jo razčlenjevalnik postavi neposredno pod tehnično vozlišče (slika 5.11a). V nasprotnem primeru ustvari novo poddrevo, ki ga postavi neposredno pod tehnično vozlišče. Koren poddrevesa, ki predstavlja koren priredja stavkov, je vozlišče

ločila ali prirednega veznika tik pred zadnjo meta pojavnico prirednega stavka. Meta pojavnice in pojavnice mejnikov tik pred njimi postanejo otroci korena priredja stavkov (slika 5.11b).

Algoritem 14 Razčlenjevalnik s pravili, prvi prehod.

Vhod:

- τ: tehnično vozlišče, koren odvisnostnega drevesa povedi.
- *nodes*[]: seznam vozlišč vseh pojavnic.
- *cooClsMetaNd*[]: seznam vozlišč vseh meta pojavnic prirednih stavkov.
- *createClsCoord(cooClsMetaNd)*: funkcija, ki izdela drevo priredja, kjer vozlišča meta pojavnic predstavljajo stavke. Vrne koren priredja stavkov.
- *append(ν, μ)*: funkcija, ki postavi vozlišče μ pod vozlišče ν.
- *size(list)*: funkcija, ki vrne dolžino seznama.
- *subConj(ν)*: vrne pravilno, če je vozlišče ν podredni veznik in napačno sicer.
- *subMetaNd(ν)*: funkcija, vrne pravilno, če vozlišče ν predstavlja meta pojavnico podrednega stavka.
- *clsMetaNd(ν)*: funkcija, ki vrne pravilno, če vozlišče ν predstavlja meta pojavnico stavka.

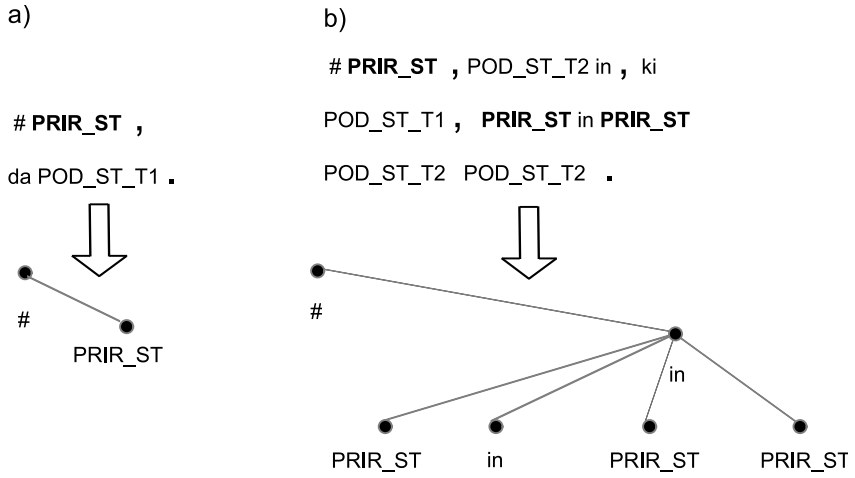
Izhod:

- Delno odvisnostno drevo s tehničnim vozliščem τ.

1: **if** $size(cooClsMetaNd) = 1$ **then**
2: $append(\tau, cooClsMetaNd[0])$
3: **else**
4: $\rho := createClsCoord(cooClsMetaNd)$
5: $append(\tau, \rho)$
6: **end if**

5.4.3 Novo začetno drevo – drugi prehod

V drugem prehodu razčlenjevalnik v odvisnostno drevo vključi preostala vozlišča (glej algoritem 15). Ko naleti na podredni veznik, ga doda na najbližje vozlišče meta pojavnice na levi, če vmes ni vejice ali prirednega veznika. Če takega vozlišča na levi ni, podredni veznik postane otrok najbližjega vozlišča meta pojavnice prirednega stavka na desni (beseda 'ki' na sliki 5.12a). Vozlišča meta pojavnic podrednih stavkov tipa 1 algoritem postavi pod podredne veznike levo od njih (slika 5.12b); tudi vejice neposredno pred podrednimi vezniki razčlenjevalnik v drevo postavi pod podredne veznike. Vozlišča meta pojavnic podrednih stavkov tipa 2 algoritem postavi pod najbližjo meta pojavnico na levi;



Slika 5.11: Prvi prehod razčlenjevalnika s pravili. Meta pojavnice prirednih stavkov, katere razčlenjevalnik najprej postavi v drevo, so izpisane krepko.

če ta ne obstaja, jo postavi pod najbližjo meta pojavnico na desni (slika 5.12c).

Algoritem 15 Razčlenjevalnik s pravili, drugi prehod.

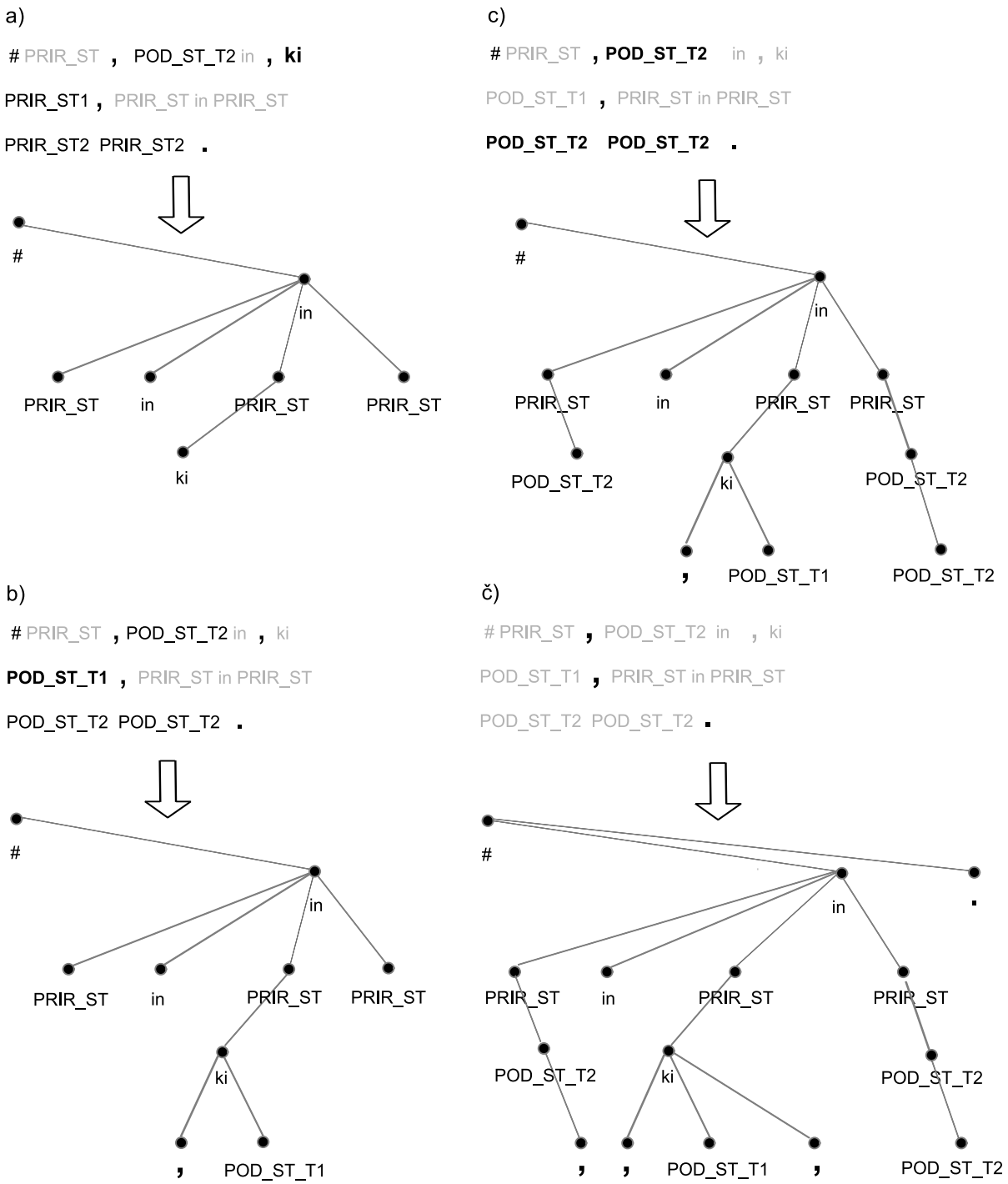
Vhod: Glej algoritem 14

Izhod:

- Končno odvisnostno drevo s tehničnim vozliščem τ.

1: $\zeta = cooClsMetaNd[0]$
2: **for all** $\nu \in nodes$ **do**
3: **if** $subConj(\nu)$ **or** $subMetaNd(\nu)$ **then**
4: $append(\zeta, \nu)$
5: **end if**
6: **if** $subConj(\nu)$ **or** $clsMetaNd(\nu)$ **then**
7: $\zeta := \nu$
8: **end if**
9: **end for**

Ko je drugi prehod zaključen, razčlenjevalnik vsa ostala ločila postavi pod najbližja vozlišča meta pojavnic na levi, razen pike na koncu povedi, katero postavi neposredno pod tehnično vozlišče. S tem je gradnja začetnega drevesa končana (slika 5.12č).



Slika 5.12: Drugi prehod razčlenjevalnika s pravili. Pojavnice, ki so že v drevesu, so izpisane v sivi barvi. Pojavnice, ki jih razčlenjevalnik v trenutnem koraku postavi v drevo, so izpisane krepko.

Poglavje 6

Ovrednotenje algoritma ARISiN

Poglavje je posvečeno ovrednotenju algoritma ARISiN. V prvem delu opisuje izvor in strukturo učno/testnih podatkov ter predstavlja način merjenja uspešnosti razčlenjevalnih algoritmov. Sledi še opis poskusov za ovrednotenje algoritma ARISiN v primerjavi z obstoječimi razčlenjevalniki.

6.1 Metode ovrednotenja, prečno preverjanje

Za ovrednotenje točnosti delovanja algoritmov, ki se učijo iz strukturiranih, označenih primerov (nadzorovano učenje), kot v primeru algoritma ARISiN, je v navadi uporaba empiričnih metod. To pomeni, da je en del množice označenih primerov, ki so na voljo, namenjen za učenje. Drugi del je namenjen ovrednotenju in sicer tako, da se izhod algoritma, ki obdela primere, primerja z ročnimi oznakami primerov. Večje kot je ujemanje, točnejši je algoritem. Pomembno je tudi, da je testna množica zadosti velika, saj je le tako možno dobiti statistično signifikantne ocene točnosti.

Če je število označenih primerov majhno, nastane problem, kako zagotoviti primerno velikost testne množice; če se prevelik del podatkov izloči iz procesa učenja, ocena točnosti delovanja algoritma ne bo smiselna, saj se iz manjše učne množice algoritem slabše uči. V takem primeru je običajna uporaba metode prečnega preverjanja, kar pomeni, da se množico označenih primerov razdeli na n enako velikih delov. Opraviti je treba n poskusov, pri čemer je vsakič eden od delov v vlogi testne množice, tako da vsi pridejo natanko enkrat na vrsto, medtem ko ostali deli skupaj predstavljajo učno množico. Končno oceno točnosti predstavlja povprečje doseženih rezultatov preko vseh poskusov prečnega preverjanja.

6.2 Opis učnih in testnih podatkov, mere točnosti

Za ovrednotenje algoritma ARISiN je bila uporabljena Slovenska odvisnostna drevesnica (SDT). Drevesnica SDT vsebuje 55.208 pojavnic, od tega 38.646 pojavnic izvira iz prvega dela romana "1984" Georgea Orwella [66], preostali del pa iz korpusa SVEZ-IJS [27]. Drevesnica SDT je predstavljala vir za učno množico osnovnih razčlenjevalnih modelov in strojnih klasifikatorjev ter testno množico in zlati standard za ovrednotenje celotnega algoritma ARISiN.

Poskusi so potekali z desetkratnim prečnim preverjanjem, kar pomeni, da je bilo v vsaki ponovitvi 90 % podatkov namenjenih učenju, preostalih 10 % pa testiranju. Statistična signifikantnost rezultatov je bila ocenjena z vzorčenimi t-testi (angl. resampled t-tests), ki jih predlagata Witten in Frank [83].

Za izdelavo osnovnih razčlenjevalnih modelov sta bila uporabljena jezikovno neodvisna razčlenjevalnika MSTP [56] verzija 0.2 in Malt [62] verzija 0.4. Na tekmovanju na konferenci CoNLL 2006 sta dosegla prvi in drugi najboljši rezultat za slovenščino, zato sta bila uporabljena za razvoj osnovnih razčlenjevalnih modelov. Samostojni razčlenjevalnik MSTP predstavlja algoritem z največjo točnostjo razčlenjevanja za slovenščino do sedaj. Poskusi z algoritmom ARISiN so praviloma parni, enkrat z osnovnimi razčlenjevalnimi modeli MSTP, drugič z modeli razčlenjevalnika Malt. Atributni opis pri razčlenjevalniku Malt je sestavljen iz modelov Φ^m_{03}, Φ^w_{00} in Φ^d_{111} (glej razdelek 3.2.3).

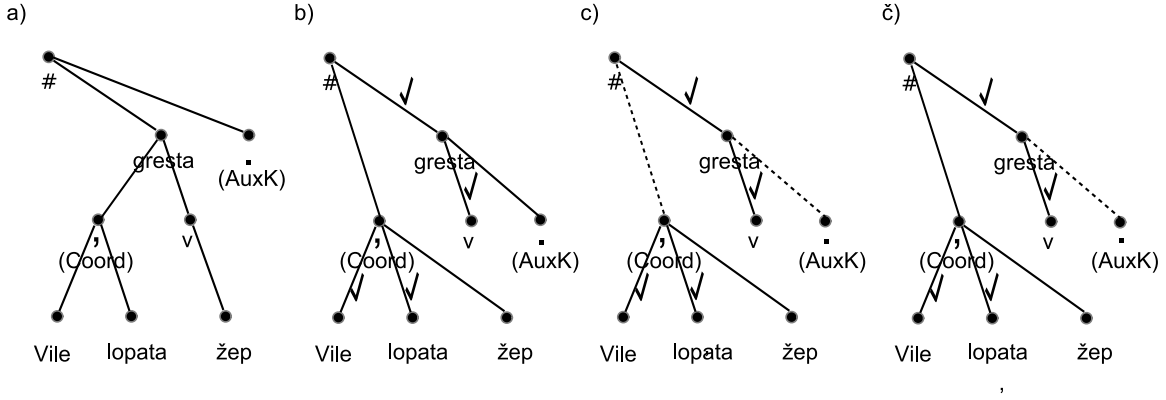
Točnost delovanja je bila ocenjena s primerjavo izhoda algoritmov s podatki v zlatem standardu, upoštevaje le strukturo drevesa, ne pa tudi določanja oznak povezav (angl. UAS, Unlabeled Attachment Score). Predstavljene so tri mere točnosti, ki se razlikujejo v tem, katere povezave pri primerjavi obravnavajo.

Definirajmo mere v okviru enega odvisnostnega drevesa. Naj bo drevo $T_a = (V, E_a, L_a)$ izhod avtomatskega razčlenjevalnega algoritma. Naj bo drevo $T_z = (V, E_z, L_z)$ drevo iz zlatega standarda, ki označuje isto poved. Naj bo $M_a \subseteq E_a$ množica obravnavanih povezav v drevesu T_a, definirana za vsako mero točnosti posebej. Naj bo $L_a \subseteq M_a$, tako da velja $\forall(i,j) \in L_a : (i,j) \in E_z$, pri čemer je i indeks nadrejene pojavnice, j pa indeks podrejene pojavnice vozlišč povezave. To je množica pravilnih povezav v drevesu T_a, kjer se nahajajo samo take povezave, ki so tudi v drevesu T_z iz zlatega standarda. Naj bo $p(t)$ funkcija, ki vrne 'pravilno', če je pojavnica t ločilo, in 'napačno' sicer, $l[(i,j)]$ pa funkcija, ki vrne oznako povezave (i,j) v zlatem standardu.

Definicija 18. Mera V je definirana kot količnik $\frac{|L_a|}{|E_a|}$.

Definicija 19. Mera L je definirana kot količnik $\frac{|L_a|}{|M_a|}$, pri čemer velja $\forall(i,j) \in M_a : \neg p(t_j)$.

Definicija 20. Mera C je definirana kot količnik $\frac{|L_a|}{|M_a|}$, pri čemer velja $\forall(i,j) \in M_a : \neg p(t_j) \vee l[(i,j)] = `Coord'$.



Slika 6.1: Primer ovrednotenja razčlenjevanja glede na različne mere točnosti. V oklepajih so zapisane oznake povezav med vozliščem in njegovim očetom.

Pri meri L se upošteva vse povezave. Pri meri B se upošteva povezave, kjer podrejeno vozlišče ni ločilo. Pri meri C se upošteva povezave, kjer podrejeno vozlišče ni ločilo, razen če ima povezava v zlatem standardu oznako 'Coord'. Ker ločila pri nadaljnji uporabi odvisnostnih dreves v splošnem ne igrajo pomembne vloge, jih ni v navadi upoštevati pri merah točnosti. Pri strukturi dreves, kot jo imata drevesnici SDT in PDT, so ločila, ki so koreni naštevanj ter priredij stavkov, izjema. Povezava v tako ločilo (oznaka 'Coord') določa položaj naštevanja oziroma priredja znotraj celotne strukture odvisnostnega drevesa, kar je bila motivacija za uporabo mere C pri poskusih. Na sliki 6.2 najdemo primere izračuna točnosti. Slika 6.2a prikazuje drevo iz zlatega standarda, na ostalih slikah je drevo, ki ga je izdelal avtomatski razčlenjevalni algoritem. Povezave, ki se ne nahajajo v množici M_a, so izrisane črtkano, povezave množice L_a so označene s kljukico. Slika 6.2b ponazarja mero V, glede na katero je točnost $\frac{4}{7}$. Točnost po meri L (slika 6.2c) je $\frac{4}{5}$ in točnost glede na mero C (slika 6.2č) je $\frac{4}{6}$.

6.3 Ovrednotenje različnih verzij algoritma ARISiN

Razdelek opisuje poskuse za ovrednotenje različnih verzij algoritma ARISiN, s katerimi je bilo izmerjeno, koliko k izboljšanju točnosti pripomorejo različne komponente algoritma. Rezultati so bili pridobljeni na delu drevesnice SDT, ki izvira iz romana "1984", kar se v tem razdelku obravnavama kot zlati standard za ovrednotenje algoritmov. Druga dimenzija, po kateri se razlikujejo poskusi, je uporaba razčlenjevalnika MSTP oziroma Malt za razvoj osnovnih razčlenjevalnih modelov. Rezultati z uporabo razčlenjevalnih modelov MSTP so prikazani v tabeli 6.1 z uporabo modelov Malt v tabeli 6.2. Po stolpcih so prikazane različne mere točnosti.

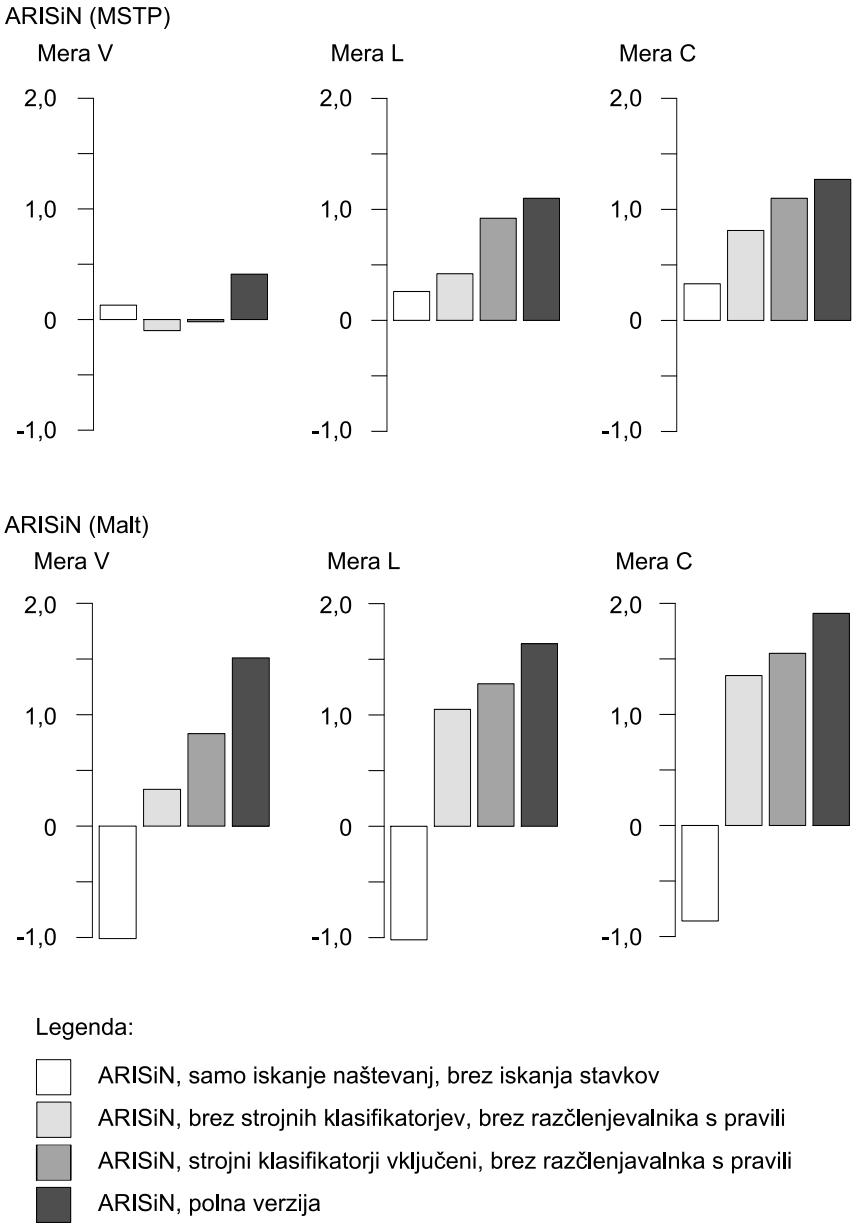
Prva dva poskusa sta bila namenjena ovrednotenju delovanja razčlenjevalnikov MSTP

in Malt brez dodatnih algoritmov za iskanje stavkov in naštevanj. To pomeni, da je učenje razčlenjevalnikov potekalo na originalnih podatkih iz drevesnice, brez predelave. Tako pridobljeni točnosti bosta nadalje služili kot izhodiščna rezultata za primerjavo z rezultati algoritma ARISiN. V drugih dveh poskusih je nastopal algoritem ARISiN, ki vključuje samo iskanje naštevanj, iskanje stavkov pa je izključeno. Pri tretjem paru poskusov je potekalo tudi iskanje stavkov, vendar brez uporabe strojnih klasifikatorjev (tako za iskanje naštevanj kot stavkov). Vse najdene kandidate za naštevanja in vse glagolske segmente je ARISiN reduciral v meta pojavnice. Prav tako ni bil uporabljen razčlenjevalnik s pravili za začetna drevesa. Za četrti par poskusov so bili strojni klasifikatorji vključeni za skupine glavnih besed naštevanj in glagolske segmente, razčlenjevalnik s pravili je bil izključen. V zadnjih dveh poskusih je bila uporabljena celotna verzija algoritma ARISiN, kot je opisana v razdelku 5. Rezultati v tabelah, ki so označeni z zvezdico, se statistično signifikantno razlikujejo od izhodiščnega rezultata vsaj z verjetnostjo 95 %. Na sliki 6.2 so prikazane razlike v točnosti med različnimi verzijami algoritma ARISiN ter izhodiščnimi rezultati.

Tabela 6.1: Točnost razčlenjevanja različnih verzij algoritma ARISiN z osnovnimi razčlenjevalnimi modeli MSTP v primerjavi z izhodiščnim rezultatom, ki ga je dosegel osnovni razčlenjevalnik MSTP brez izboljšav.

Razčlenjevalni algoritem	Mera V	Mera L	Mera C
Razčl. MSTP brez izboljšav, izhodiščni rezultat	79,35 %	80,82 %	80,24 %
ARISiN, samo iskanje naštevanj, brez iskanja stavkov	79,48 %	81,08 %	80,57 %
ARISiN, brez strojnih klas., brez razčl. s pravili	79,38 %	81,24 %	*81,05 %
ARISiN, strojni klas. vključeni, brez razčl. s pravili	79,46 %	*81,74 %	*81,34 %
ARISiN, polna verzija	79,76 %	*81,92 %	*81,51 %

Primerjava rezultatov glede na uporabo osnovnih razčlenjevalnikov pokaže, da z razčlenjevalnikom MSTP algoritem ARISiN dosega višje točnosti. Po pričakovanju je polna verzija algoritma ARISiN dala najboljše rezultate v obeh primerih. Uporaba razčlenjevalnika s pravili v polni verziji pripomore k nekoliko boljšemu rezultatu v primerjavi s četrtim poskusom. V tretjem poskusu, kjer niso bili uporabljeni strojni klasifikatorji, algoritem ARISiN naredi večje število napak pri iskanju stavkov in naštevanj – tu gre predvsem za napačne pozitivne primere. Rezultati drugega poskusa kažejo, da se pri uporabi osnovnih razčlenjevalnih modelov MSTP tudi samo z iskanjem naštevanj nekoliko izboljša točnost razčlenjevanja, medtem ko iskanje naštevanj v primeru uporabe razčlenjevalnih modelov Malt poslabša točnost. Primerjava različnih mer točnosti pokaže, da z upoštevanjem ločil algoritem ARISiN v manjši meri izboljša točnost, razen ko se upošteva samo ločila, ki so koreni poddreves naštevanj in stavčnih priredij. V nadaljnjih



Slika 6.2: Sprememba točnosti, ki jo dosegajo različne verzije algoritma ARISiN glede na izhodiščni rezultat.

Tabela 6.2: Točnost razčlenjevanja različnih verzij algoritma ARISiN z osnovnimi razčlenjevalnimi modeli Malt v primerjavi z izhodiščnim rezultatom, ki ga je dosegel osnovni razčlenjevalnik Malt brez izboljšav.

Razčlenjevalni algoritem	Mera V	Mera L	Mera C
Razčl. Malt brez izboljšav, izhodiščni rezultat	69,75 %	74,03 %	73,28 %
ARISiN, samo iskanje naštevanj, brez iskanja stavkov	*68,74 %	*73,01 %	*72,42 %
ARISiN, brez strojnih klas., brez razčl. s pravili	70,08 %	*75,08 %	*74,63 %
ARISiN, strojni klas. vključeni, brez razčl. s pravili	*70,58 %	*75,31 %	*74,83 %
ARISiN, polna verzija	*71,26 %	*75,67 %	*75,19 %

razdelkih bo podrobno predstavljena analiza posameznih komponent algoritma ARISiN.

6.4 Podrobna analiza komponent algoritma ARISiN

Razdelek podaja podrobno analizo delovanja polne verzije algoritma ARISiN. Najprej je opisana je analiza, kako algoritem obravnava naštevanja in različne vrste stavkov, nato je prikazana primerjava razčlenjevalnika s pravili z osnovnimi razčlenjevalnimi modeli.

6.4.1 Analiza obravnave naštevanj

Analiza obravnave naštevanj je potekala na nivoju skupin glavnih besed s primerjavo avtomatsko izdelanih odvisnostnih drevesih ter dreves zlatega standarda. Najprej je bila opravljena analiza števila v celoti oziroma delno najdenih skupin glavnih besed. V zlatem standardu se nahaja 615 naštevanj predlogov, samostalnikov in pridevnikov, torej takih, ki jih obravnava algoritem ARISiN. Predmet analize so bili štirje različni algoritmi: osnovni MSTP, ARISiN z osnovnimi razčlenjevalnimi modeli MSTP, osnovni Malt in ARISiN z osnovnimi razčlenjevalnimi modeli Malt. Rezultati so predstavljeni v tabeli 6.3. V prvem stolpcu je število skupin glavnih besed v avtomatsko izdelanih drevesih, ki so identične tistim v drevesih zlatega standarda. V drugem stolpcu je število skupin, v katerih so razčlenjevalniki našli le del glavnih besed. V tretjem stolpcu je število skupin, v katerih nekatere najdene glavne besede nimajo vloge glavnih besed v zlatem standardu. Rezultati za posamezne stolpce so predstavljeni še v diagramih na sliki 6.3.

Rezultati kažejo, da je po številu najdenih skupin, ki so identične tistim v zlatem standardu, najuspešnejši algoritem ARISiN z osnovnimi razčlenjevalnimi modeli MSTP. Boljši je od osnovnega razčlenjevalnika MSTP, kar se sklada z rezultati, predstavljenimi v drugi vrstici tabele 6.1. Osnovni razčlenjevalnik Malt deluje po tem kriteriju bolje kot

Tabela 6.3: Število najdenih skupin glavnih besed naštevanj v avtomatsko izdelanih drevesih.

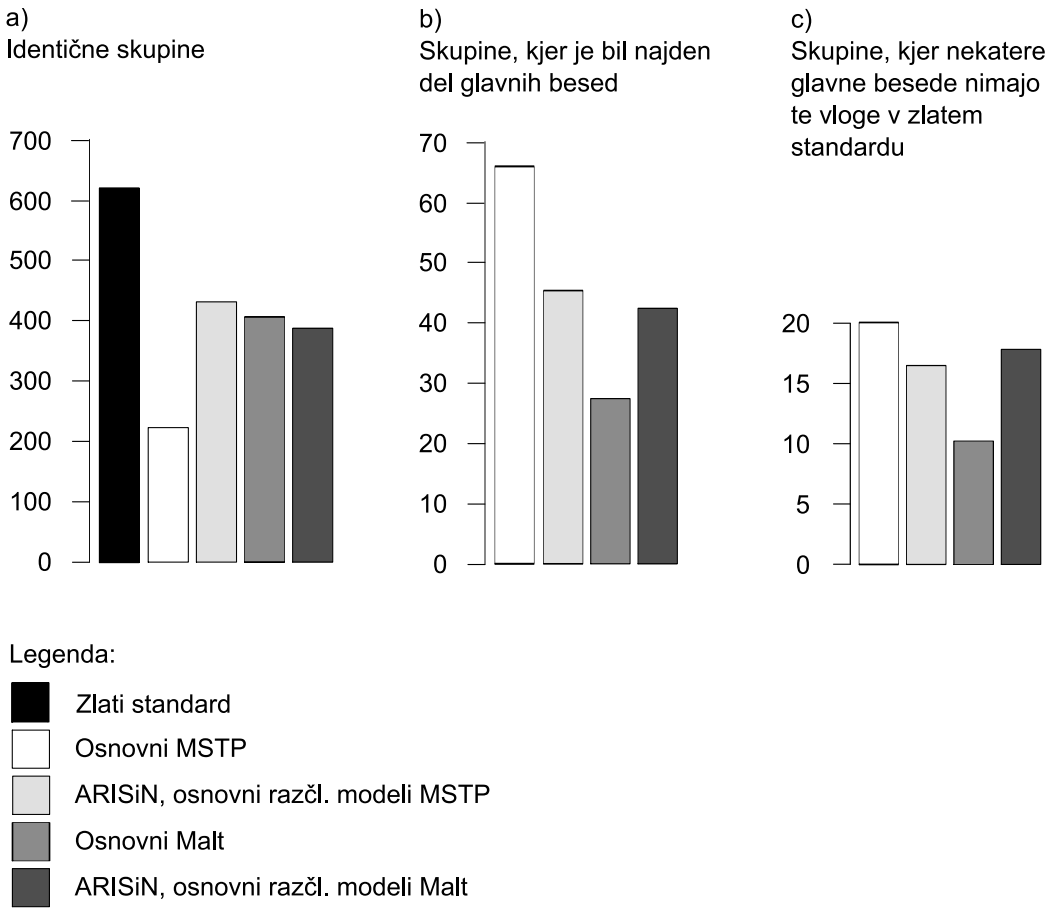
Razčlenjevalni algoritem	Identične skupine	Del glavnih besed	Glavne besede izven zlatega standarda
Osnovni MSTP	220	66	20
ARISiN (+ osn. MSTP)	427	45	17
Osnovni Malt	419	28	10
ARISiN (+ osn. Malt)	388	42	18

algoritem ARISiN z osnovnimi razčlenjevalnimi modeli Malt; po ostalih dveh kriterijih je osnovni razčlenjevalnik Malt celo najboljši izmed vseh, kar razlaga boljšo točnost razčlenjevanja v primerjavi z algoritmom ARISiN, ki vključuje samo iskanje naštevanj (glej drugo vrstico tabele 6.2).

Iskanje in redukcija naštevanj lahko pomagata povišati točnost razčlenjevanja, če z redukcijo napačnih zaporedij besed algoritem ARISiN ne naredi več škode kot koristi z redukcijo pravih naštevanj. Do določene mere je torej manj pomembno visoko število identično najdenih skupin glavnih besed, kot pa majhno število delno najdenih skupin ter še posebej skupin, ki vsebujejo besede, ki v zlatem standardu nimajo vloge glavnih besed. Pri delno najdenih skupinah osnovni razčlenjevalni modeli v drugi fazi sicer lahko še dodajo manjkajoče besede v poddrevo naštevanj, medtem ko v zadnjem primeru algoritem ARISiN v poddrevo nepopravljivo vsili napačne besede.

V sklopu analize delovanja algoritma ARISiN smo ročno pregledali 200 naključno izbranih avtomatsko izdelanih dreves. Sliki 6.4 in 6.5 predstavljata primere konkretnih razčlenitev dveh povedi. V prvem primeru je v povedi eno naštevanje pridevnikov z glavnima besedama ‘jasen’ in ‘mrzel’. Od vseh algoritmov to naštevanje najde samo osnovni razčlenjevalnik Malt, ki pri razčlenjevanju tudi sicer naredi najmanj napak. Algoritem ARISiN v obeh primerih najprej išče naštevanja samostalnikov in nápak reducira zaporedje besed ‘dan in ura’. Segment levo od besede ‘in’ je neglagolski, zato glavni besedi zadoščata pogoju III hevrističnega pravila B (glej razdelek 5.1.1), čeravno je beseda ‘in’ v tem primeru meja med stavkoma. Ko v naslednjem prehodu najde še besedi ‘mrzel’ in ‘jasen’ kot kandidata za glavne besede, ravno pogoj III prepreči redukcijo, saj je poved sestavljena samo še iz dveh glagolskih segmentov.

V drugem primeru algoritem ARISiN z obema vrstama osnovnih razčlenjevalnih modelov pravilno razčleni poved. Algoritem ARISiN v prvi iteraciji prve faze najprej najde naštevanje ‘starih, cunjastih’, v drugi iteraciji pa še naštevanje ‘kuhanem zelju in NAST predpražnikih’, kar poenostavi delo razčlenjevalnemu modelu pri gradnji začetnega drevesa. Drevo osnovnega razčlenjevalnika MSTP vsebuje pogosto napako,

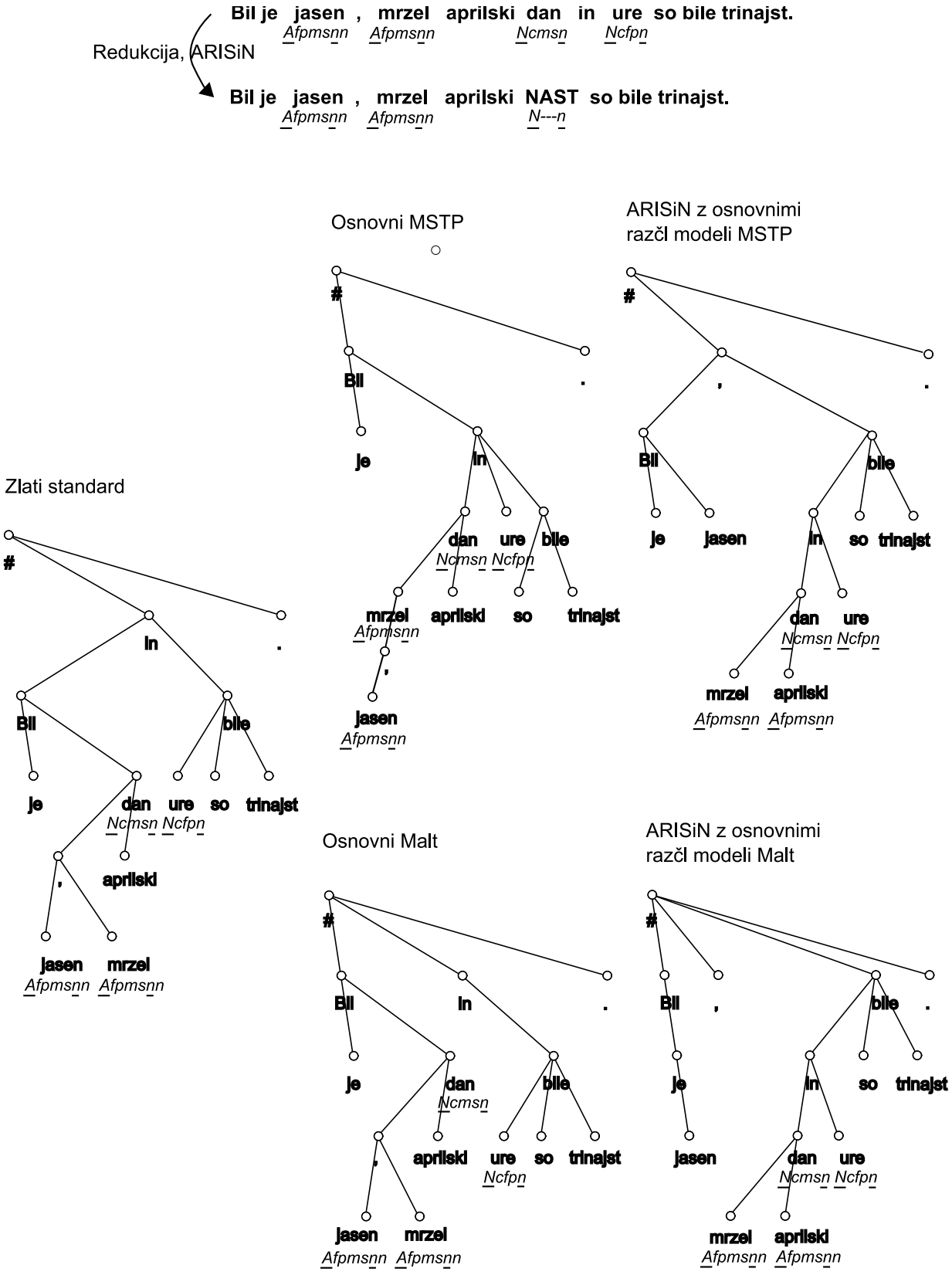


Slika 6.3: Diagrami prikazujejo število najdenih skupin glavnih besed. Na diagramu a) je s prvim stolpcem predstavljeno še celotno število skupin glavnih besed v zlatem standardu.

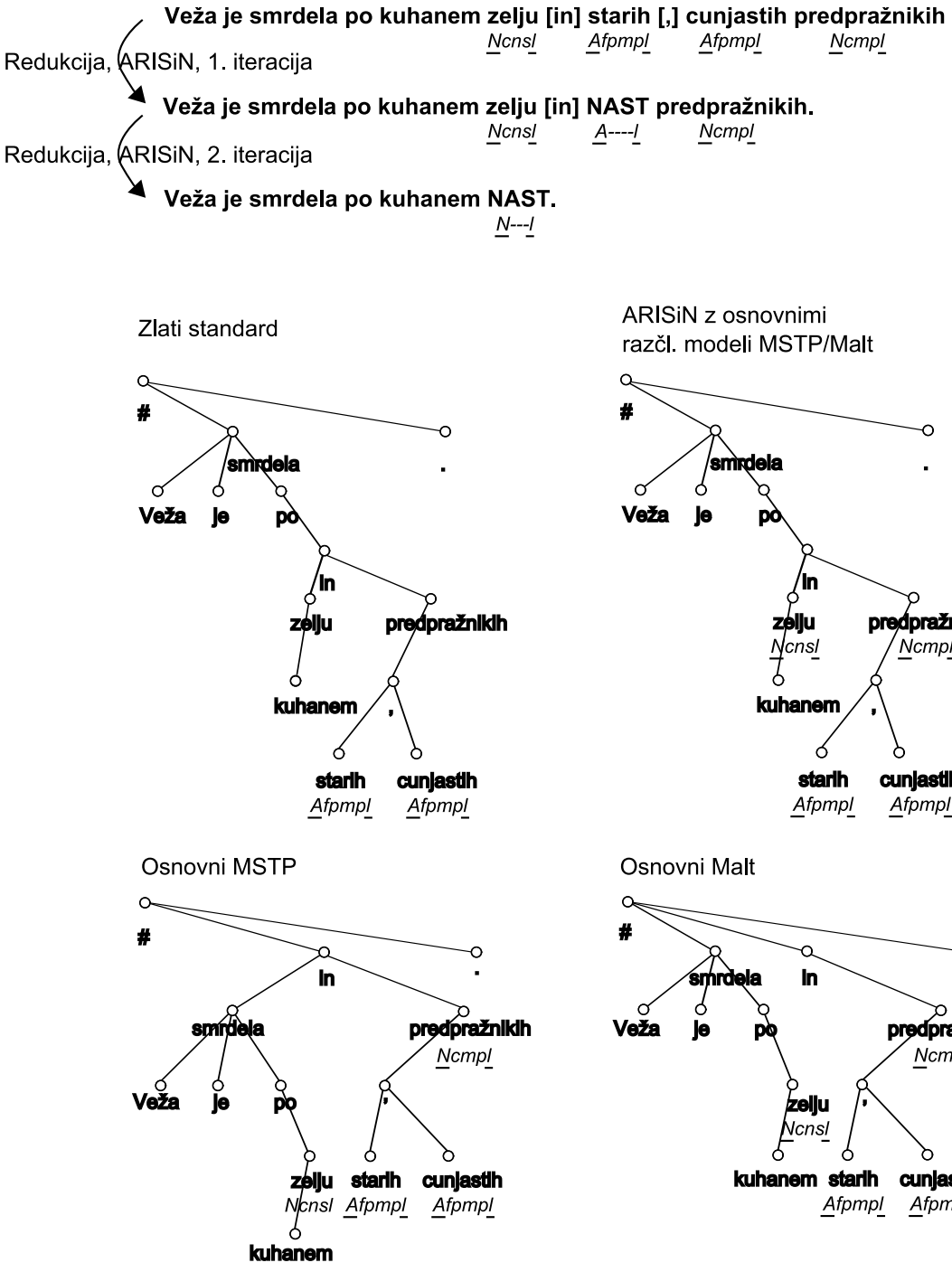
ko razčlenjevalnik v koordinacijo postavi dve besedi, ki nista iste vrste: 'smrdela' in 'predpražnikih'. Tipična napaka osnovnega razčlenjevalnika Malt, kot je prikazana na sliki, je, da pod tehnično vozlišče postavi poleg glavnega glagola še druga vozlišča.

Ker prostor ne dopušča predstavitve večjega števila konkretnih primerov, omenimo še nekaj pogostih težav algoritma ARISiN. Izkaže se, da kompleksna ugnezdena naštevanja za algoritem predstavljajo trd oreh. V primeru, da je naštevanje znotraj stavka, ki vsebuje pomožni glagol in deležnik, naštevanja ne bo mogoče najti, če je vejica oziroma priredni veznik med pomožnikom in deležnikom – algoritem bo besede levo in desno razumel kot dva glagolska segmenta in primer ne bo zadostil pogoju III hevrističnega pravila B. Kot neprilagodljiv se algoritem ARISiN izkaže tudi, kadar imajo besede napačne MSD oznake.

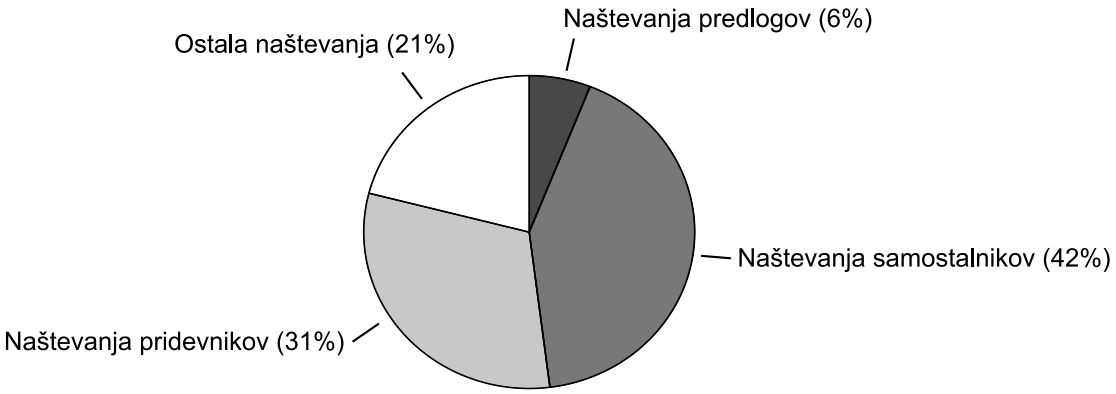
Naslednja analiza je bila posvečena obravnavi razčlenjevalnih algoritmov po različnih besednih vrstah naštevanj. Diagram na sliki 6.6 predstavlja deleže različnih vrst (predlogi, samostalniki, pridevniki, ostalo) glede na celotno število naštevanj v drevesih zlatega standarda.



Slika 6.4: Primer razčlenitve povedi z različnimi algoritmi. V besedilu v zgornjem delu slike so mejniki med segmenti označeni z oglatimi oklepaji. MSD oznake pod besedami so izpisane ležeče, oznake besednih vrst in sklonov so podčrtane.



Slika 6.5: Primer razčlenitve povedi z različnimi algoritmi. V besedilu v zgornjem delu slike so mejniki med segmenti označeni z oglatimi oklepaji. MSD oznake pod besedami so izpisane ležeče, oznake besednih vrst in sklonov so podčrtane. Obe razčlenitvi algoritma ARISiN sta enaki, zato je prikazano samo eno drevo.



Slika 6.6: Deleži različnih vrst naštevanj.

Tabela 6.4: Število najdenih skupin glavnih besed naštevanj v avtomatsko izdelanih drevesih, razčlenjeno glede na vrsto naštevanj.

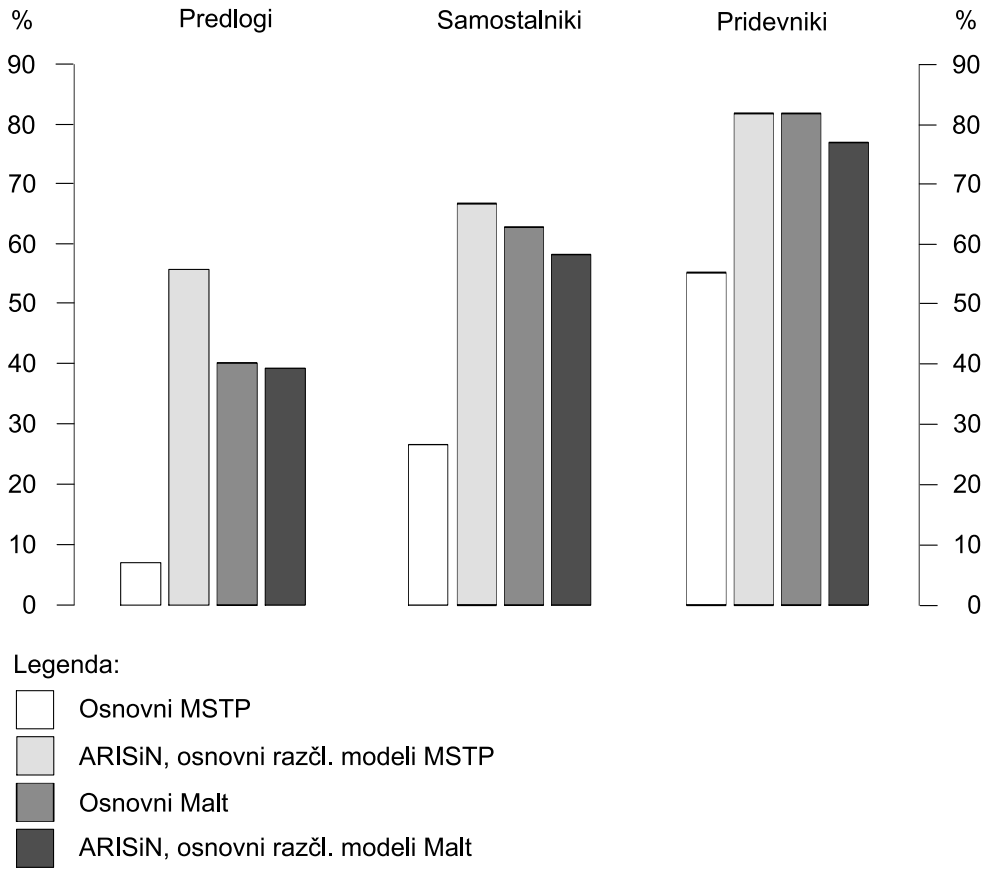
Razčlenjevalni algoritem	Predlogi	Samostalniki	Pridevniki
Osnovni MSTP	7 %	27 %	55 %
ARISiN (+ osn. MSTP)	56 %	67 %	82 %
Osnovni Malt	40 %	62 %	82 %
ARISiN (+ osn. Malt)	39 %	58 %	77 %

Tudi ta analiza je bila osredotočena na skupine glavnih besed v avtomatsko izdelanih odvisnostnih drevesih. V tabeli 6.4 zapisani rezultati pomenijo, kolikšen delež skupin določene vrste je določeni algoritem glede na zlati standard našel, upoštevaje le skupine, ki so bile identične s tistimi v zlatem standardu. Rezultati so grafično predstavljeni na sliki 6.7.

Kot je bilo pričakovati, so bili najboljši rezultati doseženi pri naštevanjih pridevnikov, ki so najenostavnejša vrsta naštevanj. Najslabši rezultati so bili doseženi za naštevanja predlogov, ki predstavljajo najbolj kompleksen tip izmed obravnavanih naštevanj.

Omeniti velja, da algoritem ARISiN poleg naštevanj najde tudi apozicije, ki sicer niso vključene v pričujočo analizo. Predstavljene so s poddrevesi enake strukture kot naštevanja, le da imajo drugo oznako povezave, ki vodi v koren poddrevesa. Zelo podobna je tudi struktura vmesnih besed med glavnimi besedami apozicije, zato za iskanje apozicij algoritem ARISiN deluje praktično enako dobro, kot za iskanje naštevanj in tako pozitivno vpliva na povečanje točnosti razčlenjevanja.

Ob primerjavi algoritmov se pokaže, da je pri osnovnem razčlenjevalniku Malt največja razlika med vrstami naštevanj. Izstopa tudi zelo nizek delež najdenih naštevanj predlogov pri osnovnem razčlenjevalniku MSTP.



Slika 6.7: Diagrami za različne razčlenjevalne algoritme prikazujejo delež najdenih skupin glavnih besed (identičnih tistim v zlatem standardu) glede na vse skupine določene vrste.

6.4.2 Analiza obravnave stavkov

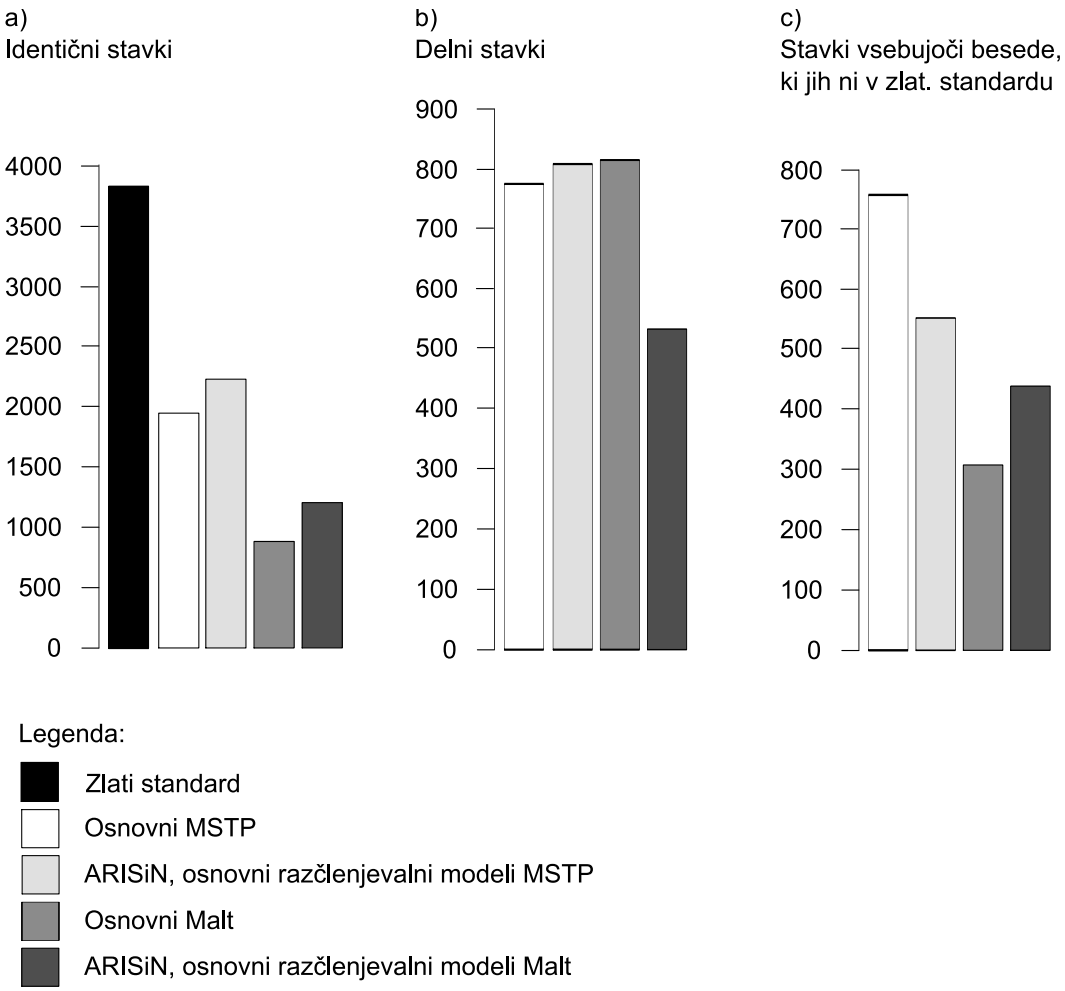
Razdelek predstavlja, kako različni razčlenjevalni algoritmi obravnavajo stavke. Po strukturi je podoben prejšnjemu razdelku o naštevanjih. Tudi tu je bila analiza opravljena s primerjavo avtomatsko izdelanih dreves s tistimi iz zlatega standarda.

V prvi analizi so prišli pod drobnogled v celoti in delno najdeni stavki za vsak razčlenjevalni algoritem posebej. Obravnavani stavki izvirajo iz dvo- ali večstavčnih povedi; v takih povedih v zlatem standardu se nahaja 3718 stavkov. Pri primerjavi so bile upoštevane samo besede, ne pa tudi ločila. Rezultati so prikazani v tabeli 6.5. V prvem stolpcu je število stavkov, ki so identični v avtomatsko izdelanih drevesih in v zlatem standardu. V drugem stolpcu je število primerov, v katerih so razčlenjevalni algoritmi našli samo del besed v stavku. V zadnjem stolpcu je število primerov, v katerih so razčlenjevalni algoritmi stavku dodali še besede, ki vanj ne sodijo glede na zlati standard. Grafično so rezultati prikazani še z diagrami na sliki 6.8.

Rezultati pri identičnih stavkih so v enakem odnosu kot dosežena točnost

Tabela 6.5: Število najdenih stavkov v avtomatsko izdelanih drevesih.

Razčlenjevalni algoritem	Identični stavki	Delni stavki	Stavki, vsebujoči besede, ki jih ni v zlat. standardu
Osnovni MSTP	1961	780	763
ARISiN (+ osn. MSTP)	2233	814	553
Osnovni Malt	860	822	308
ARISiN (+ osn. Malt)	1401	536	443

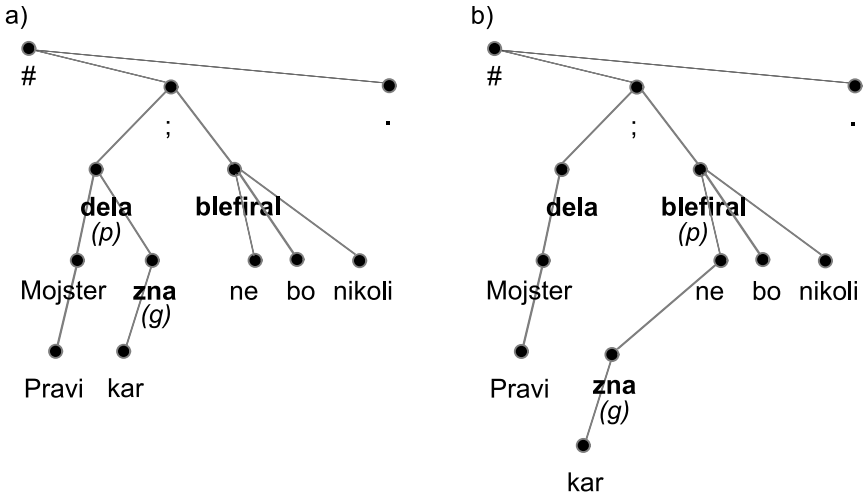


Slika 6.8: Diagrami za različne razčlenjevalne algoritme prikazujejo število najdenih stavkov. V diagramu a) je s prvim stolpcem predstavljeno še celotno število stavkov v zlatem standardu, upoštevajoč samo povedi, ki vsebujejo več kot en stavek.

razčlenjevanja (glej tabeli 6.1 in 6.2, primerjava izhodiščnih rezultatov in polne verzije algoritma ARISiN). Naj gre za uporabo osnovnih razčlenjevalnih modelov MSTP ali Malt, algoritem ARISiN vedno doseže boljše rezultate kot osnovni razčlenjevalnik. Razčlenjevalnik Malt bistveno slabše obravnava stavčno strukturo kot algoritem MSTP. Pri številu delno najdenih stavkov je algoritem ARISiN z osnovnimi razčlenjevalnimi modeli MSTP nekoliko slabši (večje število delno najdenih stavkov) kot osnovni razčlenjevalnik MSTP, medtem ko v primeru osnovnih razčlenjevalnih modelov Malt algoritem ARISiN doseže bistveno boljše rezultate kot osnovni razčlenjevalnik. Situacija je obratna pri številu stavkov, ki vsebujejo dodatne, nedovoljene besede, kjer pri uporabi razčlenjevalnika MSTP algoritem ARISiN doseže boljše rezultate (manjše število nepravilnih stavkov), pri uporabi razčlenjevalnika Malt pa slabše.

Ravno tako kot pri naštevanjih je pomembno, da algoritem ARISiN pri iskanju stavkov ne zagreši prevelikega števila takih napak, kjer bi v stavek vsilil napačne besede, kar bi lahko izničilo učinek pravilno najdenih stavkov.

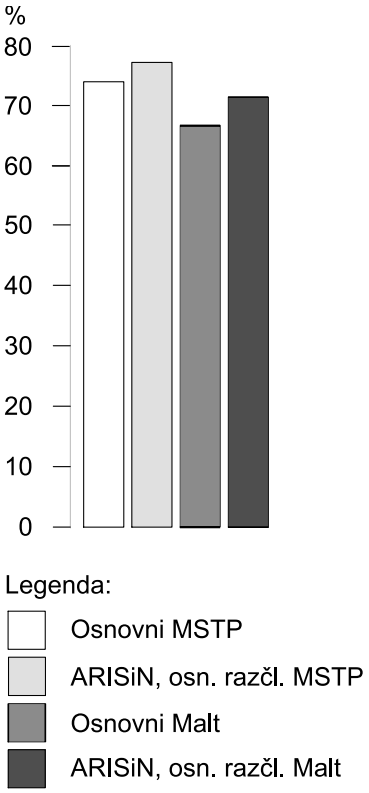
Naslednja analiza je bila namenjena točnosti določanja medstavčne strukture. Stavke v odvisnostnem drevesu lahko identificiramo z njihovimi glavnimi glagoli. Algoritem za analizo je za vsak glavni glagol g poiskal, kateri od ostalih glavnih glagolov p je njegov najbližji predhodnik. V primeru, da je bil glavni glagol g v glavnem stavku povedi, je za predhodnika p določil tehnično vozlišče. Nato je za vse razčlenjevalne algoritme izmeril, kakšen je delež glavnih glagolov g s pravilnimi najbližjimi predhodniki p. Na sliki 6.9a ima glagol 'zna' pravilnega najbližjega predhodnika 'dela', medtem ko je na sliki 6.9b najbližji predhodni določni glagol 'blefiral' napačen.



Slika 6.9: Primera pravilnega (drevo a) in nepravilnega (drevo b) predhodnika glavnega glagola. Vsi glavni glagoli so izpisani krepko. Opazovani glagol je označen z g, njegov najbližji glavni glagol – predhodnik pa označen s p.

Rezultati so predstavljeni v tabeli in na sliki 6.10. Zopet lahko ugotovimo, da so rezultati v enakem odnosu kot pri točnosti razčlenjevanja (glej tabeli 6.1 in 6.2). Algoritem ARISiN vedno doseže boljše rezultate v primerjavi z osnovnim razčlenjevalnikom istega tipa, ki je bil v algoritmu uporabljen. Prav tako uporaba razčlenjevalnika Malt daje slabše rezultate kot uporaba razčlenjevalnika MSTP.

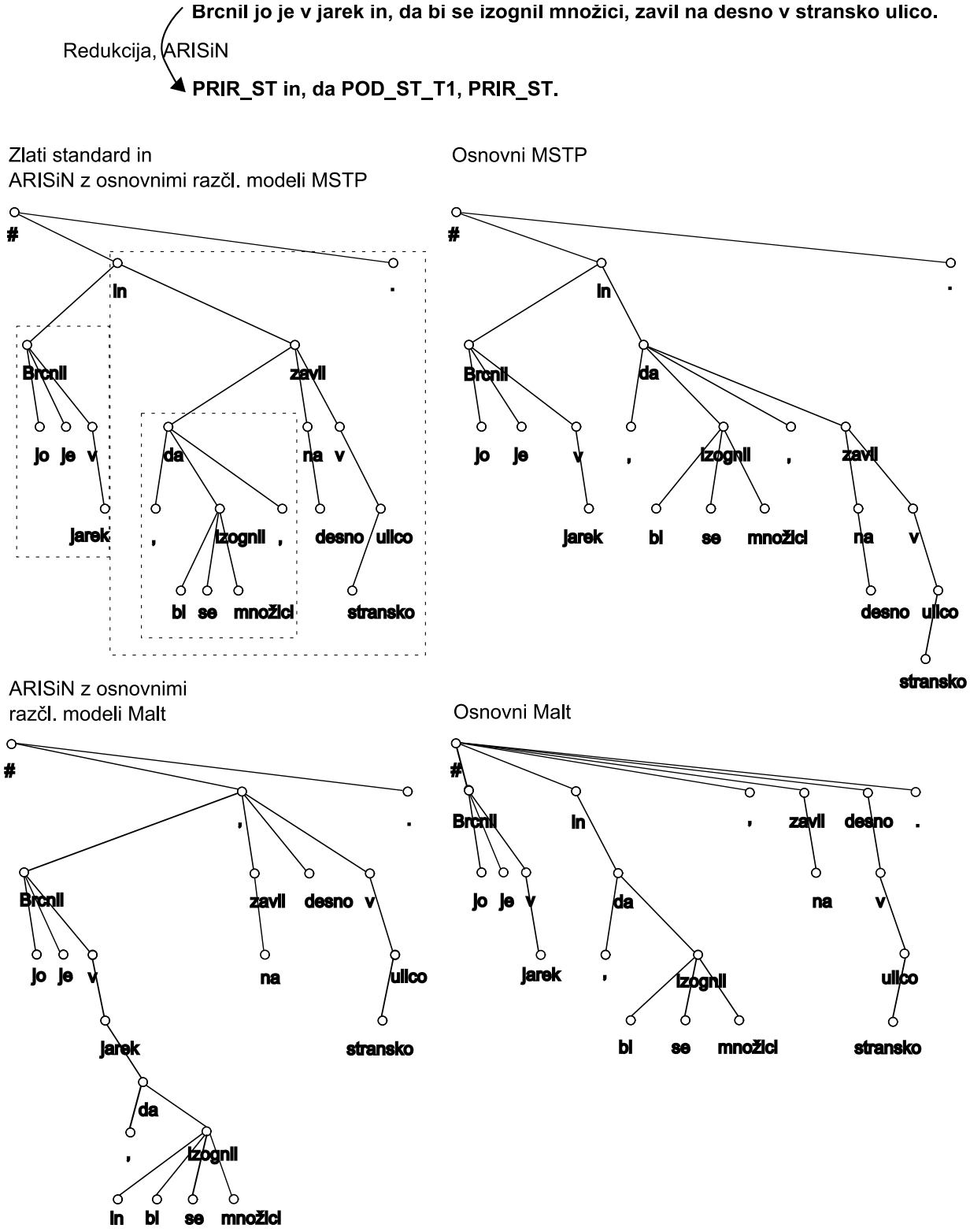
Razčlenjevalni algoritem	Delež glavnih glagolov s pravimi predhodniki
Osnovni MSTP	74%
ARISiN (+ osn. MSTP)	77%
Osnovni Malt	67%
ARISiN (+ osn. Malt)	72%



Slika 6.10: Delež glavnih glagolov, ki imajo v odvisnostnem drevesu pravi glavni glagol za predhodnika.

Naslednja primera izvirata iz množice 200 ročno pregledanih povedi in njihovih avtomatskih razčlenitev. V primeru na sliki 6.11 algoritem ARISiN najde vse tri stavke v povedi, kar omogoči, da z uporabo osnovnih razčlenjevalnih modelov MSTP dobimo popolnoma točno razčlenitev. Osnovni razčlenjevalnik MSTP ne prepozna, da je v povedi priredje dveh stavkov in postavi podredni stavek v koordinacijo s prvim prirednim stavkom. Algoritem ARISiN z osnovnimi razčlenjevalnimi modeli Malt sicer prepozna, da sta glagola 'Brcnil' in 'zavil' v koordinaciji, vendar podredni stavek postavi v odvisnost od prvega prirednega stavka namesto od drugega. Osnovni razčlenjevalnik Malt popolnoma zgreši strukturo drevesa in, kar je zelo značilno za take primere, precejšnje število besed postavi neposredno pod tehnično vozlišče.

V drugem primeru (sliki 6.12 in 6.13) ima algoritem ARISiN precejšnje težave pri



Slika 6.11: Primer razčlenitve povedi z različnimi razčlenjevalnimi algoritmi. Poddrevesa stavkov so v drevesu iz zlatega standarda označena s črtkanimi kvadrati, podredni stavek ‘, da bi se izognil množici,’ je vrinjen v drugi priredni stavek.

prepoznavanju stavkov, predvsem zaradi velikega števila segmentov v povedi. V prvi fazi algoritmu uspe reducirati le zadnji podredni stavek. Zalogaj je v drugi fazi pri gradnji odvisnostnega drevesa pretežak tako z uporabo osnovnih razčlenjevalnih modelov MSTP kot Malt. Slabo se odrežeta tudi samostojna osnovna razčlenjevalnika, ki ravno tako ne prepoznata prave medstavčne strukture v povedi.

Kot pri analizi razčlenjevanja povedi z naštevanji tudi pri stavkih povzemimo ročni pregled 200 naključno izbranih avtomatsko izdelanih dreves. Algoritem ARISiN je manj uspešen pri iskanju stavkov v povedih, v katerih je veliko neglagolskih segmentov in vrinjenih stavkov. Težave nastanejo tudi tam, kjer imajo besede MSD oznako za priredni veznik, a ne predstavljajo mejnika med segmenti. Algoritem ARISiN deluje tudi slabše nad povedmi, kjer v koordinaciji s prirednim stavkom nastopajo pojavnice, ki ne sestavljajo celotnega stavka, recimo v primerih opuščanja besed. Kot pri iskanju naštevanj tudi pri iskanju stavkov algoritmu ARISiN povzročajo preglavice napačne MSD oznake besed.

Sledi še analiza razčlenjevalnih algoritmov glede na različne vrste stavkov. Diagram na sliki 6.14 prikazuje, v kakšnem razmerju je število različnih vrst stavkov (priredni stavki, podredni stavki tipa 1 in tipa 2) v zlatem standardu. V avtomatsko izdelanih drevesih je bil analiziran delež identično najdenih stavkov (samo dvo- ali večstavčne povedi, samo primerjava besed, brez ločil) glede na vse stavke v zlatem zlati standardu. Rezultati so predstavljeni v tabeli 6.6 in z diagramom na sliki 6.15.

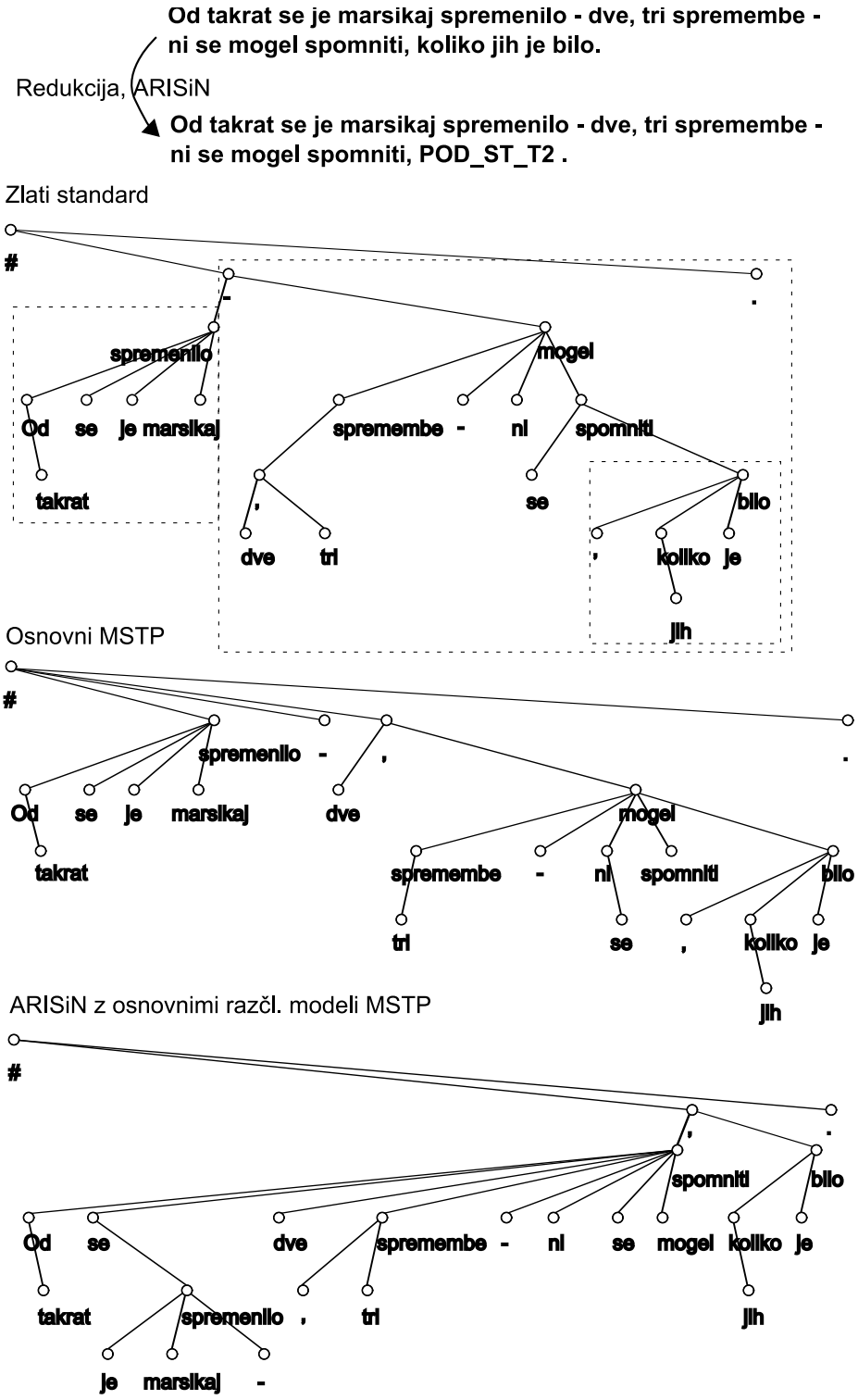
Tabela 6.6: Število najdenih stavkov v avtomatsko izdelanih drevesih, razčlenjeno glede na vrsto stavkov.

Razčlenjevalni algoritem	Priredni	Podredni tip 1	Podredni tip 2
Osnovni MSTP	46 %	67 %	36 %
ARISiN (+ osn. MSTP)	55 %	68 %	40 %
Osnovni Malt	11 %	29 %	25 %
ARISiN (+ osn. Malt)	37 %	46 %	28 %

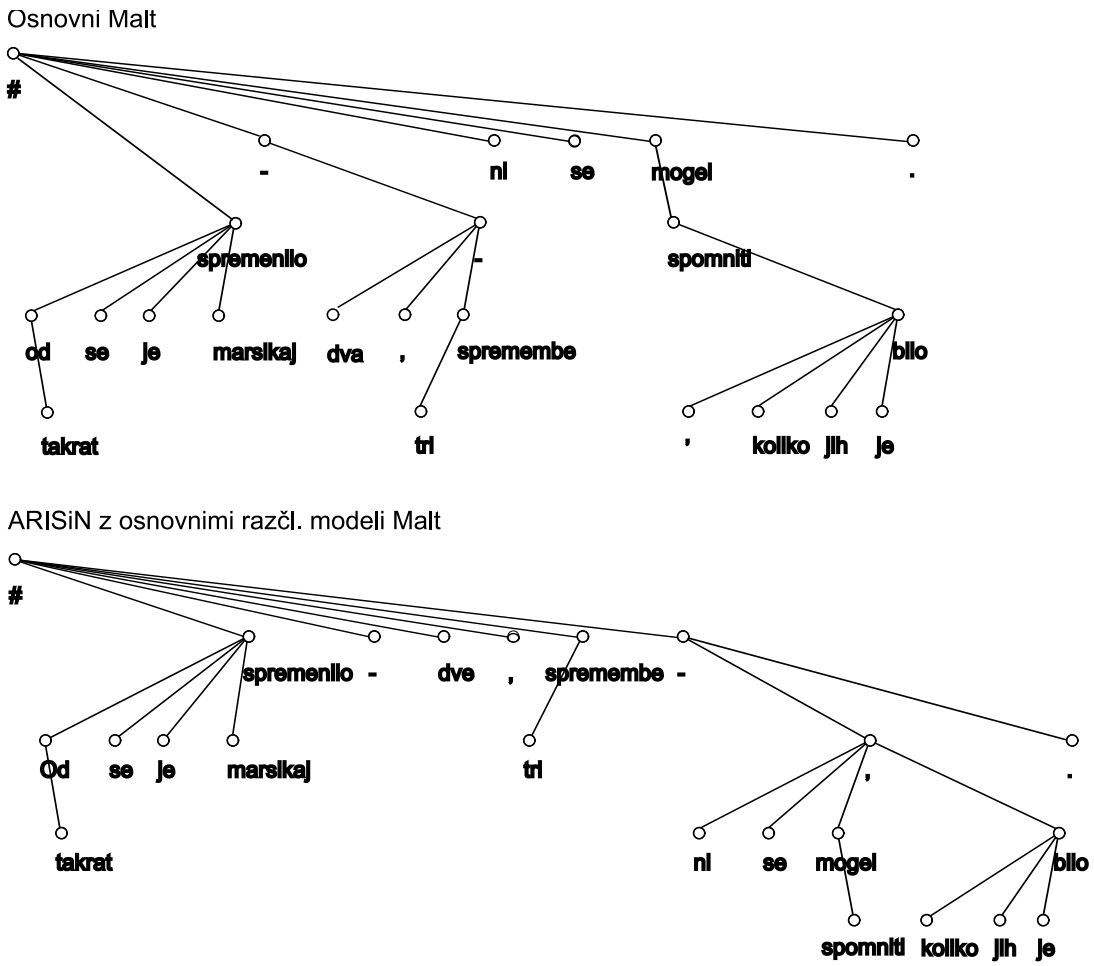
Vsi algoritmi dosegajo najboljše rezultate pri podrednih stavkih tipa 1. Začetek teh stavkov nedvoumno zaznamuje prisotnost podrednega veznika, kar olajša iskanje te vrste stavkov. Algoritmi se slabše odrežejo pri prirednih stavkih, v katerih se velikokrat nahajajo vrinjene druge stavki, kar otežkoči učinkovito iskanje te vrste stavkov. Večina algoritmov doseže najslabše rezultate pri podrednih stavkih tipa 2.

6.4.3 Analiza razčlenjevanja glede na število stavkov v povedi

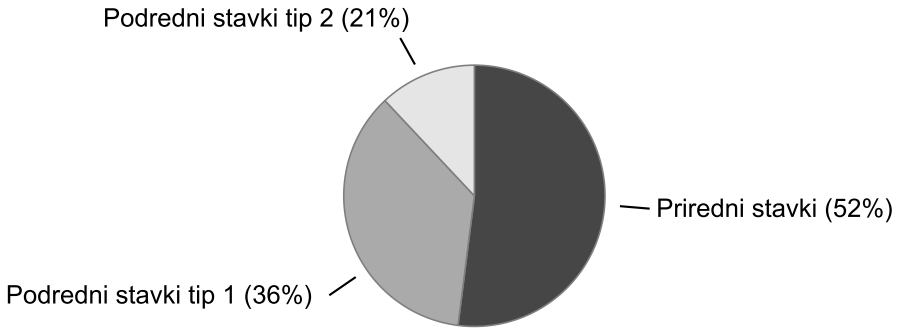
Razdelek predstavlja primerjavo točnosti razčlenjevanja med algoritmom ARISiN in osnovnima razčlenjevalnikoma glede na število stavkov v povedi. Testne množice iz



Slika 6.12: Primer razčlenitve povedi (razčlenjevalnik MSTP). Poddrevesa stavkov so v drevesu iz zlatega standarda označena s črtkanimi kvadrati.



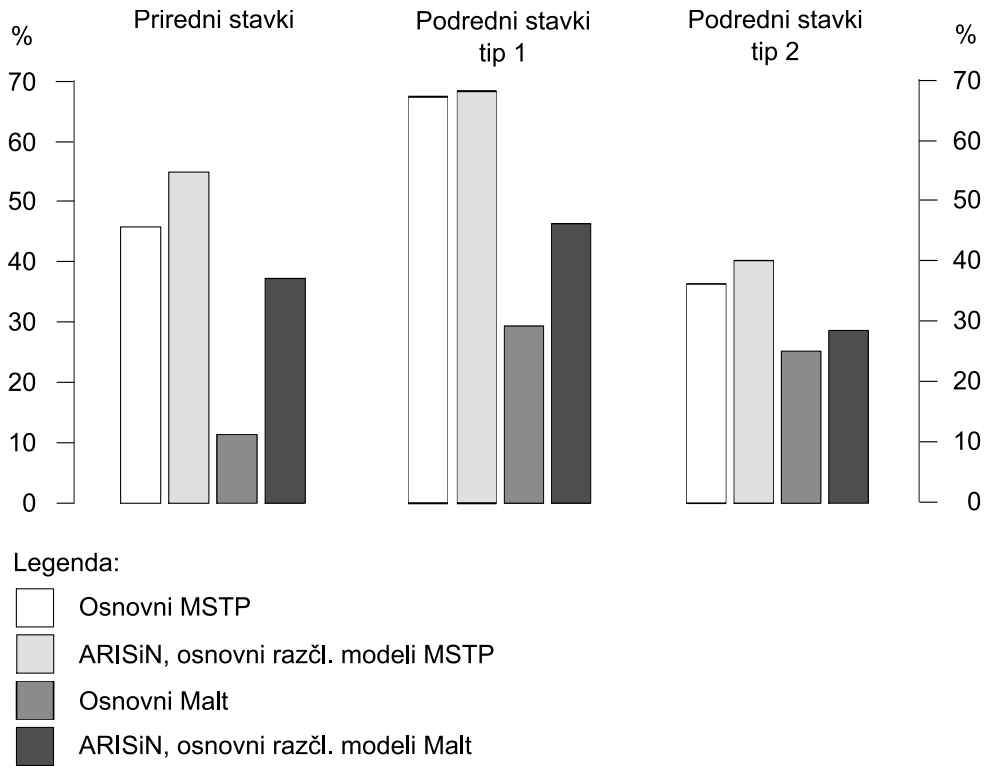
Slika 6.13: Primer razčlenitve povedi (razčlenjevalnik Malt).



Slika 6.14: Deleži različnih vrst stavkov.

poskusov s polno verzijo algoritma ARISiN, opisanih v razdelku 6.3, so bile razdeljene na šest podmnožic, v katerih so bili različni tipi povedi:

- povedi brez glagolov,



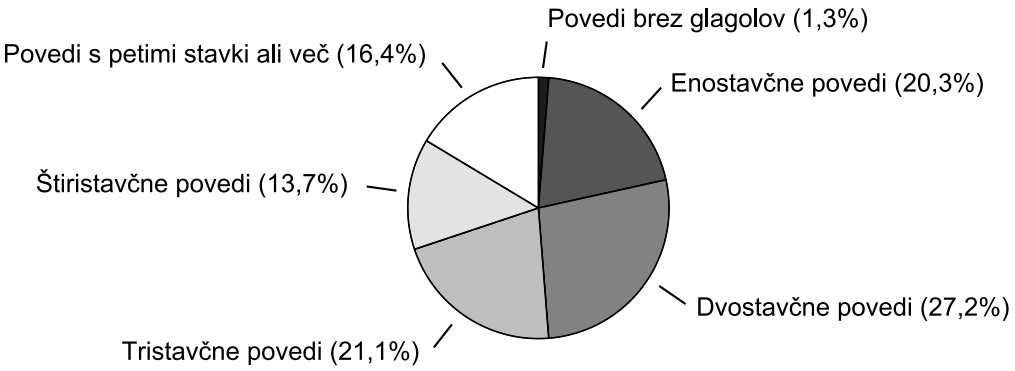
Slika 6.15: Diagrami za različne razčlenjevalne algoritme prikazujejo delež najdenih stavkov, identičnih glede na zlati standard, glede na vse stavke določene vrste.

- eno-, dvo-, tri- in štiristavčne povedi,
- povedi s petimi stavki ali več.

Kot enostavčno poved obravnavamo tako, ki vsebuje samo eno kompleksno glagolsko obliko (glavni glagol z morebitnimi pomožniki), dvostavčna poved ima dve kompleksni glagolski obliki, itn. Za vsako podmnožico posebej je bila izmerjena točnost razčlenjevanja. Delež različnih tipov povedi je predstavljen z diagramom na sliki 6.16.

Rezultati so predstavljeni v tabeli 6.7. Po vrsticah so rezultati razporejeni glede na različne razčlenjevalne algoritme, po stolpcih so razporejeni za vsak tip povedi posebej. V tabeli je zapisana mera točnosti C. Na diagramih na sliki 6.17 je prikazana razlika točnosti v odstotnih točkah med algoritmom ARISiN in ustreznim osnovnim razčlenjevalnikom kot izhodiščnim rezultatom.

Med rezultati preseneča razlika točnosti med algoritmom ARISiN z osnovnimi razčlenjevalnimi modeli MSTP in izhodiščnim rezultatom osnovnega razčlenjevalnika MSTP, izmerjena na enostavčnih povedih, ki znaša 2,47 odstotne točke. Kako je to možno, če znotraj enostavčnih povedi iskanje stavkov nima vpliva? Na enostavčnih povedih sta bila opravljena dodatna poskusa z dvema različnima verzijami algoritma



Slika 6.16: Deleži različnih tipov povedi.

Tabela 6.7: Točnost (mera C) razčlenjevanja glede na število stavkov v povedi.

	0	1	2	3	4	>=5
Osnovni MSTP	72,05 %	83,42 %	81,78 %	80,56 %	78,13 %	75,72 %
ARISiN (+ osn. MSTP)	72,53 %	85,89 %	82,89 %	81,66 %	78,98 %	76,44 %
Osnovni Malt	62,41 %	76,94 %	74,88 %	72,54 %	71,40 %	69,50 %
ARISiN (+ osn. Malt)	58,79 %	76,16 %	77,19 %	76,20 %	74,02 %	71,68 %

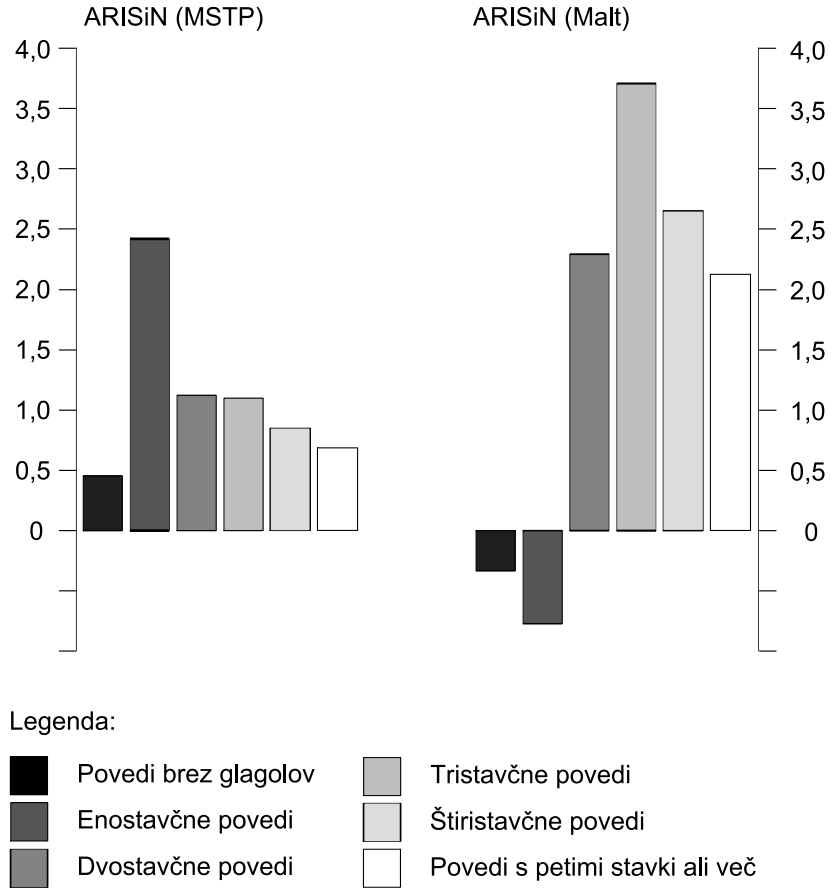
ARISiN v primerjavi z osnovnim razčlenjevalnikom:

- Poskus A: algoritem ARISiN, osnovni razčlenjevalni modeli MSTP, z izključenim iskanjem naštevanj. Dosežena točnost: 83,66 %.

- Poskus B: algoritem ARISiN, osnovni razčlenjevalni modeli MSTP, iskanje naštevanj vključeno. Dosežena točnost: 86,45 %.

Razlika 3,03 odstotne točke med rezultatom poskusa B in izhodiščnim rezultatom (83,42 %) je statistično signifikantna, medtem ko razlika 0,24 odstotne točke med rezultatom poskusa A in izhodiščnim rezultatom ni. S tem je nejasnost razčiščena tudi pri uporabi polne verzije algoritma ARISiN; pri enostavčnih povedih ima iskanje naštevanj glavno vlogo pri dvigu točnosti razčlenjevanja.

Pri rezultatih v tabeli 6.7 je opaziti dva trenda: pri uporabi osnovnih razčlenjevalnih modelov MSTP se doprinos k točnosti z uporabo algoritma ARISiN zmanjšuje od enostavčnih povedi naprej. Pri uporabi osnovnih razčlenjevalnih modelov Malt povišanje točnosti doseže vrh pri tristavčnih povedih. Predvsem pri povedih brez glagolov in pri enostavčnih povedih je opazen negativen vpliv iskanja naštevanj.



Slika 6.17: Razlika točnosti razčlenjevanja različnih tipov povedi med algoritmom ARISiN in ustreznim osnovnim razčlenjevalnikom v odstotnih točkah.

6.4.4 Analiza razčlenjevalnika s pravili

Razdelek predstavlja analizo delovanja razčlenjevalnika s pravili. Prvi korak v drugi fazi algoritma ARISiN je izdelava začetnega odvisnostnega drevesa z začetnimi razčlenjevalnimi modeli. Drevesa, v katerih so le meta pojavnice, vezniki in ločila, poimenujmo enostavna začetna drevesa. Algoritem ARISiN lahko v njih zazna določene napake, požene razčlenjevalnik s pravili in zgradi novo verzijo začetnega drevesa.

Analize so bile osredotočene na primerjavo starih verzij enostavnih začetnih dreves, v katerih so bile ugotovljene napake, in novih verzij teh dreves. Delež enostavnih začetnih dreves med vsemi začetnimi drevesi je 35 %. Od vseh enostavnih začetnih dreves algoritem ARISiN zazna napake v 17% dreves, izdelanih z razčlenjevalnikom MSTP, in v 25% primerih dreves, izdelanih z razčlenjevalnikom Malt.

Primerjave avtomatsko izdelanih enostavnih začetnih dreves z zlatim standardom ni mogoče izvesti na klasičen način z merami točnosti V, L in M kot pri meritvah v prejšnjih razdelkih. Algoritem ARISiN v prvi fazi naredi napake in pri nekaterih povedih število

meta pojavnic ni pravilno glede na zlati standard. Problematična so tudi ločila, saj nekatera algoritem ARISiN reducira, a jih po zlatem standardu ne bi smel.

Ovrednotenje točnosti razčlenjevanja pri enostavnih začetnih drevesih je zaradi zgoraj navedenih razlogov potekalo po novem postopku. Meta pojavnice v avtomatsko izdelanih drevesih algoritem za izmero točnosti najprej zamenja z glavnimi glagoli stavka, ki jih meta pojavnice predstavljajo. Avtomatsko izdelana enostavna začetna drevesa in drevesa v zlatem standardu nato transformira, tako da v njih ostanejo le še glavni glagoli in koreni stavčnih priredij. Vozliščem, ki izgubijo očeta s transformacijo, kot novega očeta določi najbližje prednike, ki ostanejo v transformiranih drevesih. Mera točnosti razčlenjevanja je delež vozlišč v transformiranem avtomatsko izdelanem drevesu, ki imajo pravilnega očeta glede na transformirano drevo iz zlatega standarda.

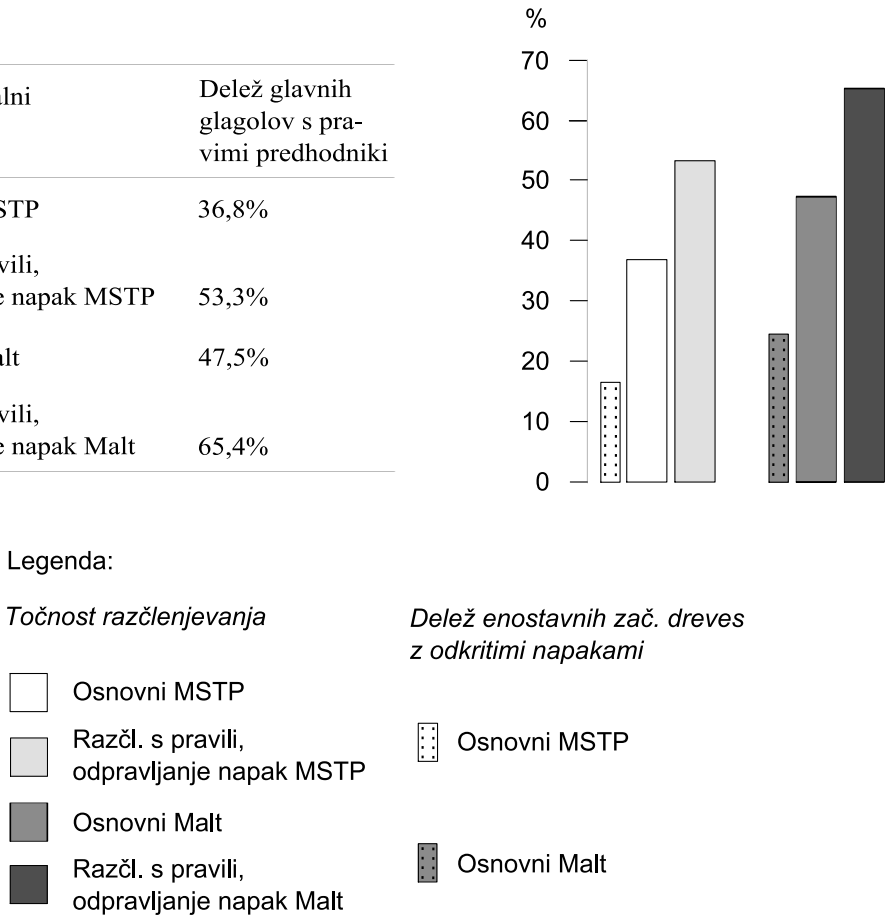
Tabela in slika 6.18 predstavljata točnost razčlenjevanja različnih algoritmov. Rezultati so bili izmerjeni na tistih enostavnih začetnih drevesih, v katerih so bile ugotovljene napake. Z medsebojno primerjavo osnovnih razčlenjevalnikov se izkaže, da v drevesih z napakami manjšo točnost doseže razčlenjevalnik MSTP, a je delež vseh enostavnih dreves z ugotovljenimi napakami pri razčlenjevalniku MSTP manjši. Primerjava med rezultati razčlenjevalnika s pravili in osnovnima razčlenjevalnikoma nedvomno kaže na večjo točnost prvega. Primerjava točnosti razčlenjevanja polne verzije algoritma ARISiN ter verzije brez razčlenjevalnika s pravili (zadnja in predzadnja vrstica v tabelah 6.1 in 6.2) pokažeta, da za relativno majhen napredek gledano v celoti ni razlog slabo delovanje razčlenjevalnika s pravili, ampak majhno število dreves, nad katerimi je možno uporabiti razčlenjevalnik s pravili.

Po ročnem pregledu 100 avtomatsko izdelanih začetnih dreves smo našli dve tipični napaki osnovnih razčlenjevalnih modelov MSTP in Malt. V nekaterih drevesih razčlenjevalnik postavi meta pojavnice podrednih stavkov neposredno pod tehnično vozlišče, kar bi pomenilo, da bi podredni stavki igrali vlogo glavnih stavkov v povedi. Drugi tip napake je koordinacija meta pojavnice podrednega in prirednega stavka, vendar je v tem primeru treba omeniti, da to v nekaterih izjemnih primerih ni napaka; takrat napako naredi razčlenjevalnik s pravili, ki takih primerov ne zmore pravilno obravnavati.

Sliki 6.19 in 6.20 predstavljata razčlenitvi dveh enostavnih začetnih dreves. V prvem primeru razčlenjevalnik s pravili pravilno določi hierarhijo med meta pojavnicami (nekatera ločila niso postavljena na pravilnih mestih, vendar ta ne vplivajo na mero točnosti opisano zgoraj). Osnovni razčlenjevalnik MSTP nápak postavi v koordinacijo prvi priredni veznik in drugo meta pojavnico, osnovni razčlenjevalnik Malt obe omenjeni pojavnici postavi pod tehnično vozlišče. Za preostale meta pojavnice osnovna razčlenjevalnika določita pravilno hierarhijo.

V drugem primeru razčlenjevalnik s pravili naredi napako, s tem da drugi meta pojavnici priredi napačno ime ‘POD_ST_T2’, kajti v stavku, ki ga predstavlja ta meta

Razčlenjevalni algoritem	Delež glavnih glagolov s pravimi predhodniki
Osnovni MSTP	36,8%
Razčl. s pravili, odpravljanje napak MSTP	53,3%
Osnovni Malt	47,5%
Razčl. s pravili, odpravljanje napak Malt	65,4%



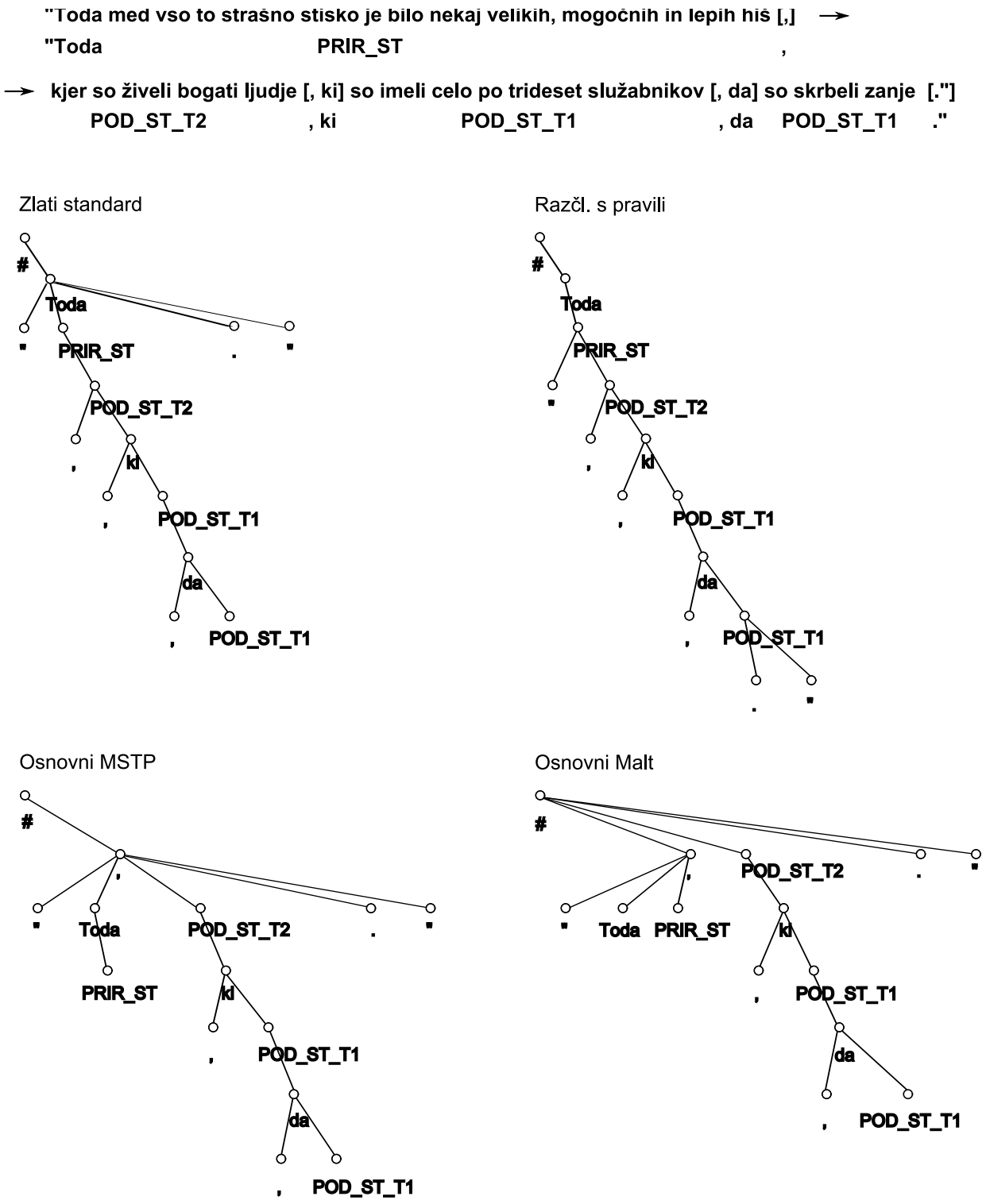
Slika 6.18: Točnost razčlenjevanja enostavnih začetnih dreves.

pojavnica, najde oziralni zaimek 'kje'. To prepreči, da bi razčlenjevalnik prepoznal koordinacijo prvih dveh meta pojavnic. Prav tako razčlenjevalnik s pravili ne prepozna dvonivojske koordinacije stavkov (prve tri meta pojavnice), kar je sicer njegova splošna pomanjkljivost. Osnovna razčlenjevalnika sicer pravilno določita koordinacijo prvih dveh meta pojavnic; vendar ravno tako ne prepoznata dvonivojskega priredja in tudi popolnoma zgrešita hierarhijo zadnjih treh meta pojavnic.

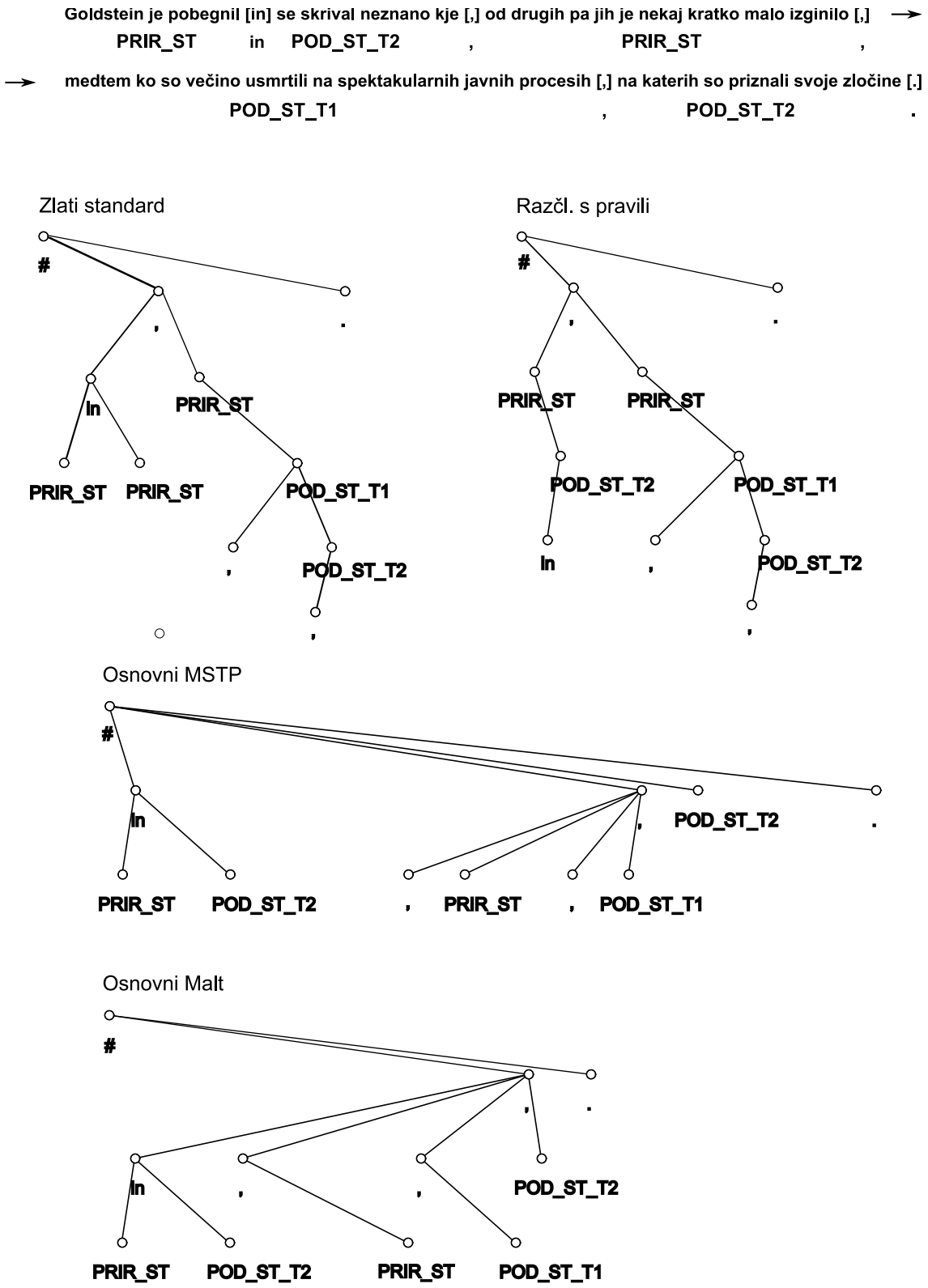
6.5 Spreminjanje velikosti učne množice

Razdelek predstavlja dve seriji poskusov, namenjenih oceni točnosti razčlenjevanja glede na količino in tip učnih podatkov. V obeh serijah poskusov je bila uporabljena polna verzija algoritma ARISiN:

1. Poskusi, v katerih se v učnih množicah postopno spreminja število povedi. Za učne in testne podatke je služilo besedilo, ki pripada delu drevesnice SDT izvirajočem iz



Slika 6.19: Primeri razčlenitev enostavnega začetnega drevesa. Zgoraj na sliki je prikazano, kako algoritem ARISiN reducira stavke v meta pojavnice. Mejniki med stavki so označeni z oglatimi oklepaji. Vmesni korak, v katerem algoritem reducira naštevanje 'velikih, mogočnih in lepih', ni prikazan. Poved je zapisana v dveh vrsticah.



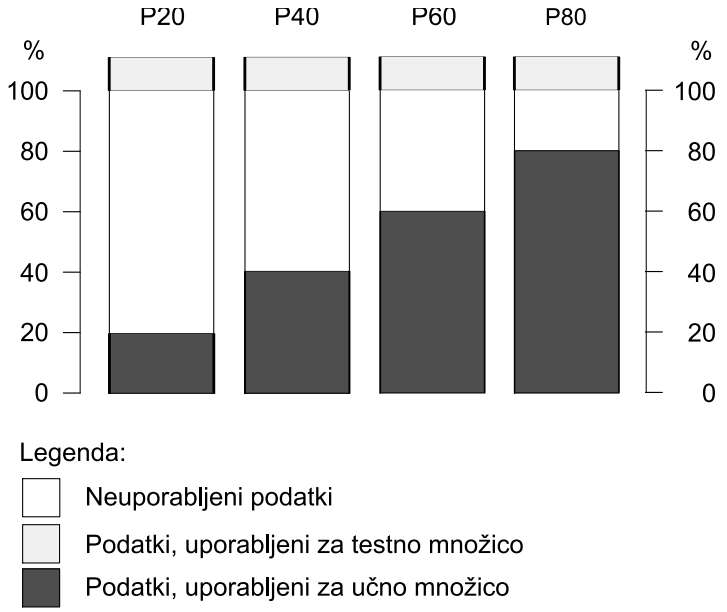
Slika 6.20: Primeri razčlenitev enostavnega začetnega drevesa. Zgoraj na sliki je prikazano, kako algoritem ARISiN reducira stavke v meta pojavnice. Mejniki med stavki so označeni z oglatimi oklepaji.

romana “1984”.

2. Poskusi, v katerih učne in testne množice vsebujejo vse podatke iz drevesnice SDT, torej tudi tiste, ki izvirajo iz korpusa SVEZ-IJS.

6.5.1 Postopno spreminjanje količine učnih podatkov

Vsak izmed štirih parov poskusov iz prve serije je kopija originalnih poskusov z osnovnima razčlenjevalnikoma in s polno verzijo algoritma ARISiN, ki so opisani v razdelku 6.3, z izjemo spreminjanja učne množice. V vsaki ponovitvi prečnega preverjanja je učna množica vsebovala le 20, 40, 60 oziroma 80 odstotkov podatkov, kot prikazuje slika 6.21. Testna množica je vedno ostala ista kot pri originalnem poskusu.



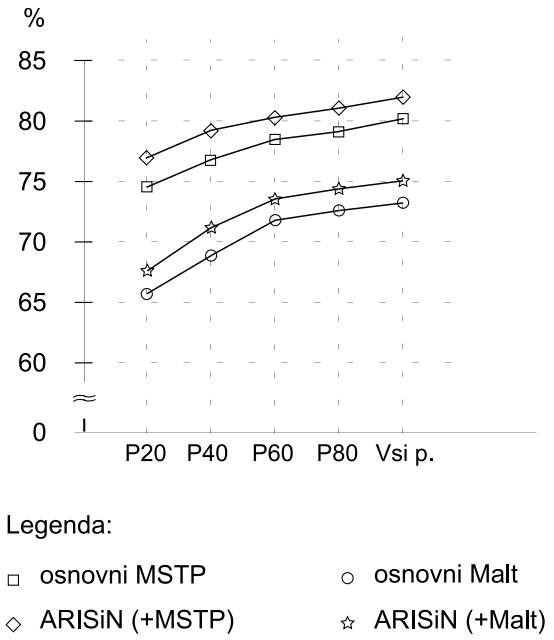
Slika 6.21: Diagrami prikazujejo razmerje podatkov, uporabljenih za učno in testno množico. Številke v odstotkih so zapisane tako, da prikažejo delež podatkov, uporabljenih za učno množico, pri čemer so kot celota (100 %) vzeti le podatki, ki lahko služijo učni množici, brez testnih podatkov. Nad diagrami so zapisane oznake poskusov.

Rezultati, predstavljeni v tabeli 6.8, so po vrsticah razvrščeni glede na razčlenjevalne algoritme in po stolpcih za vsak poskus posebej. Grafično so prikazani še na sliki 6.22. Razliko med točnostjo algoritma ARISiN in ustreznim izhodiščnim rezultatom osnovnega razčlenjevalnika predstavljajo diagrami na sliki 6.23. Vse razlike (primerjava rezultatov znotraj enega stolpca oziroma vrstice) so statistično signifikantne vsaj z verjetnostjo 95 %.

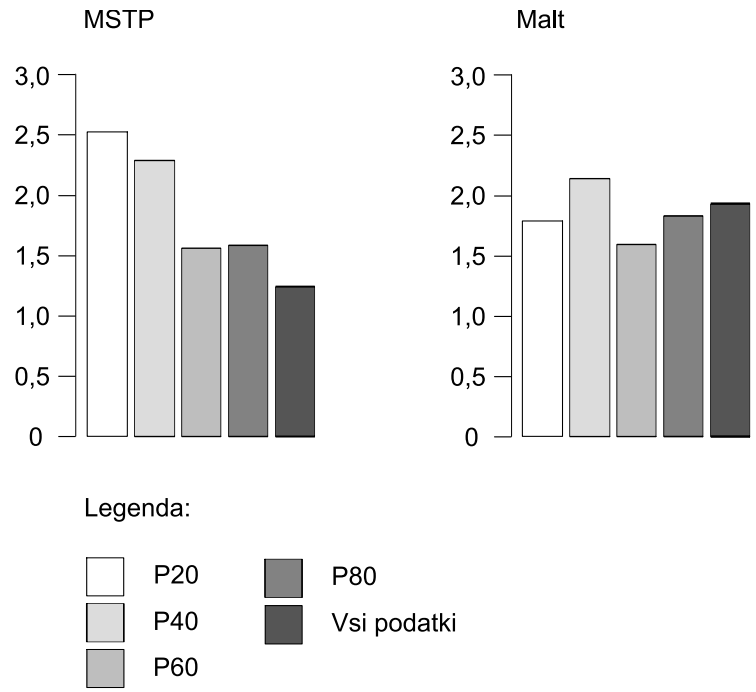
Pričakovano je bilo, da točnost razčlenjevalnikov narašča z večanjem učne množice. Ta trend izkazujejo prav vsi razčlenjevalni algoritmi. Gradient se sicer zmanjšuje pri

Tabela 6.8: Točnost razčlenjevanja glede na spremenljivo količino učnih podatkov, predstavljena z mero C. Zadnji stolpec tabele za lažjo primerjavo vsebuje rezultate iz tabel 6.1 in 6.2.

	P20	P40	P60	P80	Vsi podatki
Osnovni MSTP	74,58 %	76,89 %	78,67 %	79,27 %	80,24 %
ARISiN (+ osn.b MSTP)	77,11 %	79,19 %	80,26 %	80,87 %	81,51 %
Osnovni Malt	65,88 %	68,79 %	71,77 %	72,59 %	73,28 %
ARISiN (+ osn. Malt)	67,65 %	70,99 %	73,43 %	74,37 %	75,19 %



Slika 6.22: Točnost razčlenjevanja glede na spremenljivo količino učnih podatkov.



Slika 6.23: Diagrama za različne poskuse prikazujeta razlike točnosti med algoritmom ARISiN in osnovnima razčlenjevalnikoma, ki predstavljata izhodiščna rezultata.

večjih količinah učnih podatkov, vendar lahko sklenemo, da bi še dodatno povečanje učne množice lahko še znatno pripomoglo k boljši točnosti razčlenjevanja.

Kar se tiče razlike točnosti med algoritmom ARISiN in ustreznim osnovnim razčlenjevalnikom, je bilo pričakovati, da se bo manjšala z naraščanjem velikosti učne množice. Nekaj znanja o jezikovnih zakonitostih je namreč zapisanega v hevrističnih pravilih algoritma ARISiN, kar je neodvisno od spreminjanja učne množice. Pričakovati je bilo, da bosta z večjo količino učnih podatkov osnovna razčlenjevalnika pridobila del tega znanja. Rezultati to domnevo potrjujejo le pri uporabi razčlenjevalnika MSTP, medtem ko se naša pričakovanja ne skladajo z rezultati pri uporabi razčlenjevalnika Malt.

6.5.2 Učna množica – celotna drevesnica SDT

V drugi seriji poskusov je učno-testno množico predstavljala celotna drevesnica SDT, torej del, ki izvira iz romana "1984", in tudi del, ki je bil vzet iz korpusa SVEZ-IJS. Motivacija za ločitev poskusov, v katere je bil vključen tudi del SVEZ-IJS iz drevesnice SDT, je bilo občutno manjše število stavkov v dvo- ali večstavčnih povedih v tem delu drevesnice ter večje število brezglagolskih povedi glede na celotno število povedi:

- del iz romana "1984": 1999 povedi, od tega 70 brezglagolskih, 3614 stavkov v dvo-ali večstavčnih povedih.

- del iz korpusa SVEZ-IJS: 820 povedi, od tega 352 brezglagolskih, 764 stavkov v dvo-ali večstavčnih povedih.

Ciljnih povedi za iskanje stavkov je v SVEZ-IJS delu drevesnice bistveno manj, medtem ko je število naštevanj glede na število povedi približno v enakem razmerju tako v besedilu iz romana "1984" kot v tistem iz korpusa SVEZ-IJS.

Opravljena sta bila dva poskusa s polno verzijo algoritma ARISiN, vsakič z drugimi osnovnimi razčlenjevalnimi modeli (tabela 6.9, drugi stolpec). Rezultata drugih dveh poskusov z osnovnima razčlenjevalnikoma služita kot izhodišče za ovrednotenje algoritma ARISiN (tabela 6.9, prvi stolpec). V tabeli je vpisana mera točnosti C. Obe razliki med algoritmom ARISiN in ustreznim osnovnim razčlenjevalnikom sta statistično signifikantni vsaj z verjetnostjo 95 %.

Zaradi neugodne razporeditve stavkov v besedilu iz korpusa SVEZ-IJS je bilo pričakovati manjše izboljšanje točnosti z algoritmom ARISiN kot pri poskusih, narejenih zgolj na podatkih iz romana "1984" (slika 6.2, tabeli 6.1 in 6.2). Rezultati se skladajo s pričakovanji, vendar je možno to delno razumeti tudi kot nadaljevanje trenda povečevanja količine učnih podatkov, kar je bilo opisano v prejšnjem razdelku. Primerjava posameznih rezultatov z rezultati istih razčlenjevalnih algoritmov v tabeli 6.8 in na sliki 6.22 pokaže, da je točnost posameznih razčlenjevalnikov večja, razen pri algoritmu ARISiN z osnovnimi razčlenjevalnimi modeli MSTP.

Tabela 6.9: Rezultati poskusov na celotni drevesnici SDT.

	Osnovni razčl.	ARISiN	Razlika(odst. točke)
MSTP	80,34 %	81,06 %	0,72
Malt	73,91 %	75,52 %	1,61

Poglavje 7

Zaključek

V zadnjem poglavju so povzeti rezultati raziskav doktorske disertacije ter predstavljeni izvirni prispevki znanosti in možne nadaljnje poti raziskovanja za izboljšanje algoritmov.

7.1 Izsledki raziskav

Rezultati poskusov kažejo, da osnovni razčlenjevalniki brez predznanja o jeziku manj uspešno odkrivajo določene strukture v besedilu in da dekompozicija kompleksnih problemov pri razčlenjevanju v manjše podprobleme pozitivno vpliva na povečanje točnosti izgradnje odvisnostnih dreves.

Razliki točnosti po meri C (glej tabeli 6.1 in 6.2) znašata 1,27 odstotne točke pri nadgradnji osnovnega razčlenjevalnika MSTP s polno verzijo algoritma ARISiN ter 1,91 odstotne točke v primeru razčlenjevalnika Malt. Algoritem ARISiN z nadgradnjo razčlenjevalnika MSTP oziroma Malt delež napačno določenih povezav zmanjša z 19,76 % na 18,49 % oziroma s 26,72 % na 24,81 %. To pomeni zmanjšanje števila napak za 6,4 % pri razčlenjevalniku MSTP in 9,2 % pri razčlenjevalniku Malt.

Algoritem ARISiN vpliva samo na določene jezikovne strukture – stavke in naštevanja. V drevesnici SDT, ki je služila za preizkus algoritma, je 17 % povedi enostavčnih oziroma brezglagolskih, ki hkrati ne vsebujejo naštevanj; pri teh povedih algoritem ARISiN ne spreminja točnosti razčlenjevanja. Podrobna analiza iskanja naštevanj in stavkov, skeleta glavnih glagolov in delovanje razčlenjevalnika s pravili v razdelku 6.4 bolj kontrastno prikaže prednosti algoritma ARISiN v primerjavi z osnovnimi razčlenjevalniki.

Točnost razčlenjevalnika MSTP je večja v primerjavi z točnostjo razčlenjevalnika Malt, tudi z nadgradnjo z algoritmom ARISiN razmerje ostane enako. Prednost razčlenjevalnika Malt je hitrost delovanja, saj je njegova časovna zahtevnost le $O(n)$, pri čemer je n število pojavnic v povedi, v primerjavi s zahtevnostjo $O(n^2)$ pri razčlenjevalniku MSTP. Časovna zahtevnost algoritmov za iskanje stavkov in naštevanj je $O(n)$, kar pomeni, da nadgradnja z algoritmom ARISiN ne zvišuje časovne zahtevnosti v primerjavi z osnovnima

razčlenjevalnikoma.

7.2 Izvirni prispevki znanosti

Glavni izvirni prispevek k znanosti je algoritem ARISiN za iskanje stavkov in naštevanj v slovenskem besedilu.

Kot izvirne prispevke doktorske disertacije lahko obravnavamo nove algoritme za iskanje stavkov in naštevanj v slovenskem besedilu. Najpomembnejši rezultat je povečanje točnosti skladenjskega razčlenjevanja slovenskega jezika. Opisano podrobneje so dosežki naslednji:

- Nove metode za iskanje stavkov. Algoritem ARISiN iskanje stavkov modelira na nov način v primerjavi z do sedaj poznanimi metodami, ki se osredotočajo predvsem na iskanje stavčnih mej. Metode delujejo kot nadgradnja mehanizma segmentacije povedi.
- Nove metode za iskanje naštevanj. V dosedanji literaturi tudi za druge jezike skoraj ni prispevkov, ki obravnavajo to jezikovno strukturo.
- S pomočjo hevurističnih pravil in strojnega učenja je bilo v postopek skladenjskega razčlenjevanja vgrajeno znanje o specifikah slovenskega jezika. S tem so bili jezikovno neodvisni razčlenjevalniki nadgrajeni s predznanjem o slovenščini.
- Algoritmi temeljijo na načrtnem izkoriščanju informacij, ki jih ponuja visoka pregibnost slovenskega jezika.

7.3 Nadaljnje raziskave

Dobro zasnovano raziskovalno delo se nikoli ne zaključi, saj novi odgovori odpirajo nova vprašanja. Veljalo bi razširiti raziskave v naslednje smeri:

- Algoritem ARISiN nekoliko neprilagodljivo obravnava naštevanja in stavkov: bodisi določi zaporedje pojavnic kot stavek oziroma naštevanje ali ne. Namesto tega bi bilo verjetno bolje z mehkejšimi omejitvami vplivati na delovanje osnovnih razčlenjevalnikov. Pri razčlenjevalniku MSTP bi lahko povečali uteži ustreznim povezavam grafa povedi in tako dopustili algoritmu za iskanje maksimalnega vpetega drevesa, da najde najprimernejšo razčlenitev. Pri razčlenjevalniku Malt bi informacije o stavčni strukturi lahko vnesli z novimi atributi pri strojni klasifikaciji.
- Poleg iskanja naštevanj predlogov, samostalnikov in pridevnikov bi lahko še razširili nabor besednih vrst pri iskanju skupin glavnih besed.

- V strojnih klasifikatorjih bi lahko uporabili dodatne atribute. Na primer, pri naštevanjih bi za opis v atributnem modelu lahko uporabili še pojavnice, ki se nahajajo levo od najbolj leve ter desno od najbolj desne glavne besede naštevanja.
- S hevrističnimi pravili bi bilo možno v algoritem vključiti dodatno predznanje o slovenščini.
- Pri razčlenjevalniku s pravili se ponuja še precej možnosti nadgradnje in izboljšav, saj trenutna verzija predpostavlja preveč uniformno medstavčno strukturo znotraj povedi. Razčlenjevalnik ne upošteva večnivojske strukture prirednih stavkov, ne obravnava ustrezno brezlgagolskih stavkov itd.

Našteli smo le nekaj pomembnejših idej kako nadaljevati z delom; med raziskovanjem se je porodilo še precej drugih. Idej je vedno občutno več, kot je časa za njihovo udejanjenje.

Poglavje 8

Dodatek

8.1 MSD oznake

Za lažje razumevanje je v tabelah razložen pomen najpomembnejših pozicij v MSD oznakah.

Besedna vrsta
pozicija 1

V	glagol
N	samostalnik
A	pridevnik
R	prislov
S	predlog
C	veznik

Vrsta veznika
pozicija 2

c	priredni
s	podredni

Oblika glagola
pozicija 3

i	indikativ
p	deležnik

Sklon

Predlog, pozicija 3,
samostalnik, pozicija 5,
pridevnik, pozicija 6.

n	imenovalnik
g	rodilnik
d	dajalnik
a	tožilnik
l	mestnik
i	orodnik

Število

Samostalnik, pozicija 4,
pridevnik, pozicija 5.

m	moški
f	ženski
n	srednji

Spol

Samostalnik, pozicija 3,
pridevnik, pozicija 4.

s	ednina
p	množina
d	dvojina

8.2 Slovarček

Predstavljamo slovensko-angleški slovarček strokovnih izrazov, uporabljenih v disertaciji. Za nekatere smo prevod poiskali mi, ostali se nahajajo na seznamu, ker se v slovenščini le poredko pojavljajo, oziroma zato, da bi se izognili dvoumju.

dekompozicijska predstavitev	constituency representation
odvisnostna predstavitev	dependency representation
drevesnica	treebank
Slovenska odvisnostna drevesnica	SDT, Slovene Dependency Treebank
nadrejeno vozlišče, oče	head, governor
podrejeno vozlišče, otrok	dependent, modifier
predhodnik	ancestor
potomec	descendant
pojavnica	token
meta pojavnica	meta token
morfološka oznaka, MSD oznaka	morphosyntactic description tag, MSD-tag
glagolska, samostalniška itd. fraza	verb, noun, etc. phrase
naštevanje	intraclausal coordination
glavna beseda naštevanja	conjunct head
stavčno priredje	interclausal coordination
razčlenjevalnik	parser
razčlenjevalnik s pravili	rule-based parser
iskanje stavkov	clause splitting, clause identification
iskanje naštevanj	intraclausal coordination identification
obrezovanje	pruning

8.3 Seznam kratic

Kratica	Pomen
ARISiN	Algoritem za razčlenjevanje z iskanjem stavkov in naštevanj
MSTP	Maximum spanning tree parser
MSD	Morphosyntactic description tag
SDT	Slovene dependency treebank
PDT	Prague dependency treebank
LFG	Lexical functional grammar
GPSG	Generalized phrase strucure grammar
TAG	Tree adjoining grammar
HPSG	Head-driven phrase structure grammar
WG	Word grammar
FGD	Functional generative description
MTT	Meaning-text theory
CDG	Constraint dependency grammar
WEKA	Waikato environment for knowledge analysis
CFG	Context-free grammar
PCFG	Probabilistic context-free grammar

8.4 Zahvale

Ob začetku doktorskega študija je bilo računalniško jezikoslovje zame popolnoma nepoznan svet znanosti. Takrat sta mi bila v pomoč doc. dr. Tomaž Erjavec in mentor dr. Tomaž Šef, saj sem si v pogovorih z njima ustvaril splošno sliko o jezikovnih tehnologijah. Glede na njune izkušnje mi še vedno marsikakšen njun nasvet omogoči nova spoznanja o področju, ki ga raziskujem.

Za raziskovalca je na začetku najteže najti problem, ki bi bil primeren za doktorsko disertacijo; biti mora zadosti zahteven, da je vreden doktorata, vendar ne preveč, da je še obvladljiv. Ta korak sem naredil v času obiska Inštituta za formalno in uporabno jezikoslovje Karlove univerze v Pragi, kjer sem se spoznal z dr. Zdeněkom Žabokrtskim. On mi je osvetlil trenutno stanje raziskav v skladenjskem razčlenjevanju, s pomočjo njegovih nasvetov se je izoblikovala osnovna ideja doktorske disertacije.

Pri uvajanju v pisanje znanstvenih publikacij mi je že od začetka študija največ časa posvetil mentor prof. dr. Matjaž Gams. Tudi sicer je nadzoroval moje raziskovalno delo in me s predlogi usmerjal pri poskusih, ki sem jih izvajal v okviru doktorskih raziskav.

Tekom študija sem pred sodelavci Odseka za inteligentne sisteme na Institutu "Jožef Stefan" in člani Laboratorija za umetno inteligenco Fakultete za računalništvo in informatiko Univerze v Ljubljani večkrat opravil predstavitve, kako napreduje moje delo. Njihovi komentarji in predlogi so pomembno vplivali na potek mojega dela.

Literatura

[1] Anne Abeillé (ur.). *Treebanks: Building and Using Parsed Corpora*. Kluwer Academic Publishers, 2003.

[2] Steven P. Abney. Rapid Incremental Parsing with Repair. V *Proceedings of the 6th New OED Conference*, strani 1–9, 1990.

[3] Leonard Bloomfield. *Language*. The University of Chicago Press, 1933.

[4] Joan Bresnan. *Lexical-Functional Syntax*. Blackwell Publishing, 2000.

[5] Sabine Buchholz in Erwin Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. V *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, strani 149–164, 2006.

[6] Xavier Carreras in Lluís Màruez. Boosting Trees for Clause Splitting. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, strani 73–75, 2001.

[7] Glenn Carroll in Eugene Charniak. Two Experiments on Learning Probabilistic Dependency Grammars from Corpora, Technical Report TR-92. Department of Computer Science, Brown University, 1992.

[8] Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[9] Noam Chomsky. On Certain Formal Properties of Grammars. *Information and Control*, 2(2):137–167, 1959.

[10] Noam Chomsky. *Aspects of the Theory of Syntax*. The MIT Press, 1965.

[11] Noam Chomsky. *Lectures on Government and Binding*. Foris Publications, 1981.

[12] Noam Chomsky. *The Minimalist Program*. The MIT Press, 1995.

[13] Yoeng-jin Chu in Tseng-hong Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[14] Michael Collins, Jan Hajič, Lance Ramshaw in Cristoph Tillmann. A Statistical Parser for Czech. V *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, strani 505–512, 1999.

[15] Koby Crammer, Ryan McDonald in Fernando Pereira. New large margin algorithms for structured prediction. V *Learning with Structured Outputs Workshop (NIPS)*, 2004.

[16] Koby Crammer in Yoram Singer. Ultraconservative Online Algorithms for Multiclass Problems. *The Journal of Machine Learning Research*, 3:951–991, 2003.

[17] Walter Daelemans, Jakub Zavrel, Ko van der Sloot in Antal van den Bosch. Tilburg Memory Based Learner, version 5.1, Reference Guide, ILK research group technical report series no. 04-02. ILK Research Group, 2004.

[18] Hervé Déjean. Using ALLiS for Clausing. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, strani 1–3, 2001.

[19] Hervé Déjean. Learning Rules and Their Exceptions. *Journal of Machine Learning Research*, 2:669–693, 2002.

[20] Sašo Džeroski, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdeněk Žabokrtský in Andreja Žele. Towards a Slovene Dependency Treebank. V *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, strani 1388–1391, 2006.

[21] David Dowty. On the semantic content of the notion 'thematic role'. V Partee B. H. Chierchia, G. in R. Turner, uredniki, *Properties, Types and Meaning. Volume II: Semantic Issues*, strani 69–130. Reidel, 1989.

[22] Jason M. Eisner. An Empirical Comparison of Probability Models for Dependency Grammar, Technical Report, IRCS-96-11. Institute of Research in Cognitive Science, University of Pennsylvania, 1996.

[23] Jason M. Eisner. Three New Probabilistic Models for Dependency Parsing: An Exploration. V *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, strani 340–345, 1996.

[24] Eva I. Ejerhed. Finding clauses in unrestricted text by finitary and stochastic methods. V *Proceedings of the second conference on Applied natural language processing*, strani 219–227, 1988.

[25] Eva I. Ejerhed. Finite State Segmentation of Discourse into Clauses. *Natural Language Engineering*, 2(4):355–364, 1996.

[26] Tomaž Erjavec. MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. V *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, strani 1535–1538, 2004.

[27] Tomaž Erjavec. The English-Slovene ACQUIS Corpus. V *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, strani 2138–2141, 2006.

[28] Charles J. Fillmore. The case for case. V E. W. Bach in R. T. Harms, uredniki, *Universals in Linguistic Theory*, strani 1–88. Holt, Rinehart and Winston, 1986.

[29] Yoav Freund in Robert E. Schapire. Experiments with a New Boosting Algorithm. V *International Conference on Machine Learning (ICML)*, strani 148–156, 1996.

[30] Matjaž Gams. *Weak Intelligence: Through the Principle and Paradox of Multiple Knowledge*. Nova Science, 2001.

[31] Gerald Gazdar, Ewan Klein, Geoffrey Pullum in Ivan A. Sag. *Generalized Phrase Structure Grammar*. Blackwell Publishing, 1985.

[32] Jan Hajič. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. V *Issues of Valency and Meaning*, strani 106–132. Karolinum, 1998.

[33] Jan Hajič, Barbora Vidova Hladka, Jarmila Panevová, Eva Hajičová, Petr Sgall in Petr Pajas. *Prague Dependency Treebank 1.0*. LDC 2001T10, 2001.

[34] Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, Beáta Megyesi, Mattias Nilsson in Markus Saers. Single Malt or Blended? A Study in Multilingual Parser Optimization. V *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning (EMNLP-CoNLL)*, strani 933–939, 2007.

[35] James Hammerton. Clause Identification with Long Short-Term Memory. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, strani 61–63, 2001.

[36] Mary P. Harper in All A. Helzerman. Extensions to Constraint Dependency Parsing for Spoken Language Processing. *Computer Speech and Language*, 9:187–234, 1995.

[37] Deirdre Hogan. Empirical measurements of lexical similarity in noun phrase conjuncts. V *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, strani 149–152, 2007.

[38] Tomaš Holán in Zdeněk Žabokrtský. Combining Czech Dependency Parsers. V *Proceedings of the Ninth International Conference on Text, Speech and Dialogue (TSD)*, strani 95–102, 2006.

[39] Richard Hudson. *Word Grammar*. Blackwell Publishing, 1984.

[40] Richard Hudson. *English Word Grammar*. Blackwell Publishing, 1990.

[41] Ray S. Jackendorf. *Semantic Interpretation in Generative Grammar*. MIT Press, 1972.

[42] Aravind K. Joshi. Tree-Adjoining Grammars. V G. Rozenberg in A. Salomaa, uredniki, *Handbook of Formal Languages. Volume 3: Beyond Words*, strani 69–123. Springer, 1987.

[43] Aravind K. Joshi. Tree Adjoining Grammars: How much Context Sensitivity is Required to Provide Reasonable Structural Descriptions? V D. Dowty, L. Karttunen in A. Zwicky, uredniki, *Narural Language Parsing*, strani 206–250. Cambridge University Press, 1987.

[44] Ron Kaplan in Joan Bresnan. Lexical-Functional Grammar: a Formal System for Grammatical Representation. V J. Bresnan, urednik, *The Mental Representation of Grammatical Relations*, strani 173–281. MIT Press, 1982.

[45] Vladislav Kuboň, Markéta Lopatková, Martin Plátek in Patrice Pognan. Segmentation of complex sentences. V *Proceedings of the 9th International Conference on Text, Speech and Dialogue (TSD)*, strani 151–158, 2006.

[46] Taku Kudo in Matsumoto Yuji. Japanese Dependency Analysis Using Cascaded Chunking. V *Proceedings of the 6th Workshop on Computational Language Learning (CoNLL)*, strani 63–69, 2002.

[47] Nina Ledinek. *Površinskoskladenjsko označevanje korpusa Slovene Dependency Treebank, diplomsko delo*. Filozofska fakulteta, Univerza v Ljubljani, 2005.

[48] Vilson J. Leffa. Clause Processing in Complex Sentences. V *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, strani 937–943, 1998.

[49] Mitchell P. Marcus, Beatrice Santorini in Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[50] Domen Marinčič, Matjaž Gams in Tomaž Šef. Parsing Aided by Intraclausal Coordination Detection. V *Proceedings of The 6th International Workshop on Treebanks and Linguistic Theories (TLT)*, strani 79–84, 2007.

[51] Domen Marinčič, Matjaž Gams in Mitja Lušrek. Knowledge vs. simulation for bidding in tarok. *Informatica (Ljubljana, Slovenija)*, 30(4):467–476, 2006.

[52] Domen Marinčič, Tea Tušar, Matjaž Gams in Tomaž Šef. Analysis of automatic stress assignment in Slovene. *Informatica (Vilnius, Litva)*, sprejeto v objavo.

[53] Hiroshi Maruyama. Structural Disambiguation with Constraint Propagation. V *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL)*, strani 31–38, 1990.

[54] Takehiko Maruyama, Hideki Kashioka, Tadashi Kumano in Hozumi Tanaka. Development and Evaluation of Japanese Clause Boundaries Annotation Progam. *Journal of Natural Language Processing (v japonščini)*, 11(3):39–68, 2004.

[55] Ryan McDonald, Kevin Lerman in Fernando Pereira. Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. V *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, strani 216–220. 2006.

[56] Ryan McDonald, Fernando Pereira, Kiril Ribarov in Jan Hajič. Non-projective Dependency Parsing Using Spanning Tree Algorithms. V *Proceedings of the Joint Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, strani 523–530, 2005.

[57] Igor Mel'čuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, 1988.

[58] Wolfgang Menzel in Ingo Schröder. Decision Procedures for Dependency Parsing Using Graded Constraints. V *Proceedings of the Workshop on Processing of Dependency-Based grammars*, strani 78–87, 1998.

[59] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[60] Antonio Molina in Ferran Pla. Clause Detection Using HMM. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, strani 70–72, 2001.

[61] Antonio Molina in Ferran Pla. Shallow Parsing Using Specialized HMMs. *Journal of Machine Learning Research*, 2:595–613, 2002.

[62] Joakim Nivre. *Inductive Dependency Parsing*. Springer, 2006.

[63] Joakim Nivre, Johan Hall, Sandra Kuebler, Ryan McDonald, Jens Nilsson, Sebastian Riedel in Deniz Yuret. The CoNLL 2007 Shared Task on Multilingual Dependency Parsing. V *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning (EMNLP-CoNLL)*, strani 915–932. 2007.

[64] Tomohiro Ohno, Shigeki Matsubara, Hideki Kashioka, Takehiko Maruyama in Yasuyoshi Inagaki. Incremental Dependency Parsing of Japanese Spoken Monologue Based on Clause Boundaries. V *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (ACL)*, strani 169–176, 2006.

[65] Constantin Orasăn. A Hybrid Method for Clause Splitting in Unrestricted English Texts. V *Proceedings of the International Conference on Artificial and Computational Intelligence For Decision, Control and Automation In Engineering and Industrial Applications (ACIDCA)*, strani 129–134, 2000.

[66] George Orwell. *Nineteen Eighty-Four*. 1949.

[67] Harris V. Papageorgiou. Clause Recognition in the Framework of Alignment. V R. Mitkov in N. Nicolov, uredniki, *Recent Advances in Natural Language Processing*, strani 417–425. John Benjamins Publishing Company, 1997.

[68] Jon D. Patrick in Ishaan Goyal. Boosted Decision Graphs for NLP Learning Tasks. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, strani 58–60, 2001.

[69] Carl J. Pollard in Ivan A. Sag. *Information-Based Syntax and Semantics*. CLSI Publications, University of Chicago Press, 1987.

[70] Carl J. Pollard in Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. CLSI Publications, University of Chicago Press, 1994.

[71] Georgiana Puscasu. A Multilingual Method for Clause Splitting. V *Proceedings of the Seventh Annual CLUK Research Colloquium*, strani 199–206, 2004.

[72] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 2003.

[73] Petr Sgall, Eva Hajičová in Jarmila Panevová. *The Meaning of the Sentence in its Pragmatic Aspects*. Reidel, 1986.

[74] Stanley Starosta. *The Case for Lexicase*. Pinter Publishers, 1988.

[75] Robert E. Tarjan. Finding Optimum Branchings. *Networks*, 7:25–35, 1977.

[76] Lucien Tesnière. *Éléments de syntaxe structurale*. Editions Klincksieck, 1959.

[77] Erik F. Tjong Kim Sang. Memory-Based Clause Identification. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, strani 67–69, 2001.

[78] Erik F. Tjong Kim Sang. Memory-Based Shallow Parsing. *Journal of Machine Learning Research*, 2:559–594, 2002.

[79] Erik F. Tjong Kim Sang in Hervé Déjean. Introduction to the CoNLL-2001 Shared Task: Clause Identification. V *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL)*, 2001.

[80] Jože Toporišič. *Slovenska slovnica*. Založba obzorja, 2004.

[81] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[82] Wen Wang in Mary P. Harper. A Statistical Constraint Dependency Grammar (CDG) Parser. V *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, strani 42–49, 2004.

[83] Ian H. Witten in Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition*. Morgan Kaufmann Publishers, 2005.

[84] Hiroyasu Yamada in Yuji Matsumoto. Statistical Dependency Analysis with Support Vector Machines. V *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, strani 195–206, 2003.

Objave za izpolnjevanje pogojev za dokončanje doktorskega študija:

Domen Marinčič, Matjaž Gams in Mitja Lušrek. Knowledge vs. simulation for bidding in tarok. *Informatica* (Ljubljana, Slovenija), 30(4): 467–476, 2006, revija indeksirana v bazi Inspec.

Domen Marinčič, Tea Tušar, Matjaž Gams in Tomaž Šef. Analysis of automatic stress assignment in Slovene. *Informatica* (Vilnius, Litva), sprejeto v objavo, revija indeksirana v bazi SCI Expanded.