

# Obrada prirodnog jezika - chatbot programi

---

**Kokan, Roko**

**Master's thesis / Diplomski rad**

**2021**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:217:017085>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2024-11-30**



*Repository / Repozitorij:*

[Repository of the Faculty of Science - University of Zagreb](#)



**SVEUČILIŠTE U ZAGREBU**  
**PRIRODOSLOVNO–MATEMATIČKI FAKULTET**  
**MATEMATIČKI ODSJEK**

Roko Kokan

**OBRADA PRIRODNOG JEZIKA -**  
**CHATBOT PROGRAMI**

Diplomski rad

Voditelj rada:  
prof. dr. sc. Luka Grubišić

Zagreb, travanj, 2020.

Ovaj diplomski rad obranjen je dana \_\_\_\_\_ pred ispitnim povjerenstvom u sastavu:

1. \_\_\_\_\_, predsjednik
2. \_\_\_\_\_, član
3. \_\_\_\_\_, član

Povjerenstvo je rad ocijenilo ocjenom \_\_\_\_\_.

Potpisi članova povjerenstva:

1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_

*Svojoj obitelji, prijateljima, mentoru i svima koji su mi pomogli na studentskom putu da ostvarim željeni cilj stjecanja znanja i primanja diplome kao potvrde za zalaganje i trud.*

# Sadržaj

<b>Sadržaj</b>	<b>iv</b>
<b>Uvod</b>	<b>4</b>
<b>1 Kratki povijesni pregled</b>	<b>5</b>
Obrada prirodnog jezika temeljena na pravilima . . . . .	7
Statistička obrada prirodnog jezika . . . . .	7
<b>2 Obrada teksta - tokenizacija</b>	<b>9</b>
Što je riječ . . . . .	10
Tokenizacija u drugim jezicima . . . . .	12
<b>3 Morfološka analiza</b>	<b>14</b>
Korjenovanje . . . . .	14
Lematizacija . . . . .	17
Gramatičko tagiranje . . . . .	17
<b>4 Sintaktička analiza</b>	<b>26</b>
Beskontekstna gramatika . . . . .	28
Parsanje u beskontekstnim gramatikama[32] . . . . .	32
<b>5 Chatbot programi</b>	<b>46</b>
Što su chatbot programi . . . . .	47
Studijski primjer . . . . .	50
<b>6 Zaključak</b>	<b>57</b>
<b>Bibliografija</b>	<b>59</b>

# Uvod

Obrada prirodnog jezika (engl. *natural language processing* - NLP) jest područje računarske znanosti i umjetne inteligencije koje omogućava računalima analizu i razumijevanje ljudskih (prirodnih) jezika. Pod terminom ljudski jezik ili prirodni jezik podrazumijeva se jezik koji ljudi koriste za svakodnevnu, međusobnu komunikaciju. Kao kontrast stoje "umjetni jezici" u koje spadaju npr. programski jezici i matematičke notacije.

Prirodni jezici su se razvijali generacijskim prenošenjem "s koljena na koljeno" i zbog toga se jako teško mogu svesti pod skup eksplicitnih pravila. Kao posljedica toga, ne postoje kompajleri ni interpreteri za prirodne jezike poput, primjerice, hrvatskog ili engleskog jezika, dok za umjetne postoje. Stoga, cilj ovog područja jest omogućiti računalima obavljanje korisnih zadataka koji uključuju ljudski jezik, poput komunikacije čovjek-računalo, ili poboljšanja komunikacije čovjek-čovjek ili nekakve korisne obrade jezika/govora.

A obrada jezika jest doista posvuda. Činjenica da računalo "razumije" naš jezik postala je sveprisutna i uobičajena u našoj svakodnevici i toliko smo se priviknuli na nju da možda i ne zamjećujemo da je upravo računalna obrada jezika zaslužna za mnoge stvari koje nas okružuju. Recimo, internetski pretraživač može pružiti značajnije rezultate ako se indeksi internetskih stranica ili dokumenti čuvaju tako da uzimaju u obzir značenje tekstova. Standardni alat kod svih internet pretraživača te tipkovnica za pametne telefone jest automatsko popunjavanje koje koristi NLP za dovršetak misli prilikom pretraživanja. Mnogi programi za obradu teksta, proširenja za internet pretraživače te uređivači teksta sadrže pravopisne i gramatičke provjernike. Računalni programi koji omogućuju oblik interakcije (čavrljanja) između čovjeka i stroja (tzv. *chatbot* programi) koriste obradu prirodnog jezika kako bi generirali odgovor korisniku. NLP protočne strukture koje generiraju tekst ne samo da mogu kreirati kratke odgovore korisniku kod *chatbot* programa te virtualnih pomoćnika, već i mogu kreirati mnogo dulje odlomke teksta. Pa tako *Associated Press* koristi NLP kao robot-novinara za pisanje gotovih članaka o financijskim vijestima i izvještaja o sportskim događajima. Programi mogu i sastavljati vremensku prognozu koja će zvučati slično onome kako bi je vaš meteorolog sastavio. Imamo i filtere za neželjenu poštu koji su pomogli e-poštama da preteknu telefon i telefaks kao glavne komunikacijske kanale u 90-ima. Nadalje, oko 20% tweet-ova vezanih za predsjedničke izbore u SAD-u 2016. sastavljeni su od strane *chatbot*-ova[8]. Na stranicama koje imaju službu za korisnike možete čak raz-

govarati s takvim vrstama programa. NLP sustavi mogu generirati i više od običnih objava na društvenim mrežama. Mogu razvijati dugometrane filmove te ostavljati recenzije proizvoda na npr. *Amazonu*. Štoviše, većina recenzija za kupljene proizvode i odgledane filmove nisu napisani ljudskim rukama. Kada *chatbot* programima dodamo još dodatani atribut, prepoznavanje glasa, tada dobijemo *Siri* i *Google Assistant*[18]. Također, NLP sustavi mogu zamijeniti tajnice u kompanijama i tako odgovarati na pozive, organizirati događaje, zakazati susrete, komunicirati s klijentima preko e-pošte. Mogu čak pripomoći u marketinškim kampanjama generirajući reklame.

Search	Web	Documents	Autocomplete
Editing	Spelling	Grammar	Style
Dialog	Chatbot	Assistant	Scheduling
Writing	Index	Concordance	Table of contents
Email	Spam filter	Classification	Prioritization
Text mining	Summarization	Knowledge extraction	Medical diagnoses
Law	Legal inference	Precedent search	Subpoena classification
News	Event detection	Fact checking	Headline composition
Attribution	Plagiarism detection	Literary forensics	Style coaching
Sentiment analysis	Community morale monitoring	Product review triage	Customer care
Behavior prediction	Finance	Election forecasting	Marketing
Creative writing	Movie scripts	Poetry	Song lyrics

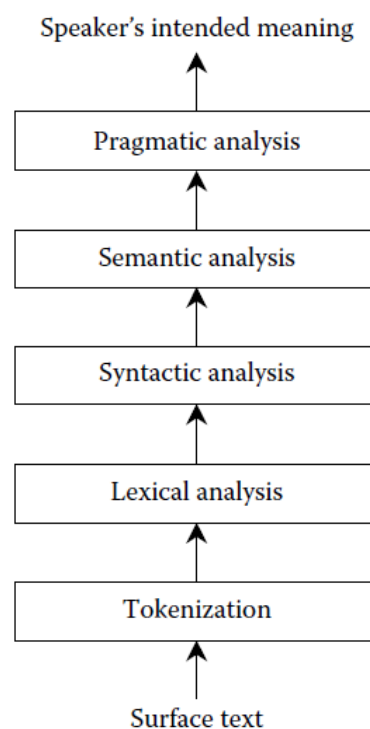
Slika 0.1: Primjene obrade prirodnog jezika

Možda sve prethodno nabrojano zvuči kao obični proces nad podacima koje računalo i inače ionako izvršava. I donekle je to točno. No, ono što suštinski razlikuje aplikaciju koja obrađuje jezik od one koja obrađuje sirove systemske podatke jest poznavanje i razumijevanje jezika. Uzmimo primjer *Unixove wc* systemske naredbe. Ona se koristi za prebrojavanje bitova, riječi i linija u nekoj tekstualnoj datoteci. Kada tom naredbom želimo prebrojati bitove i linije tada to uistinu jest obično izvršavanje računalnih instrukcija nad podacima. S druge strane, ako želimo izbrojati koliko riječi sadrži tekstualna datoteka tada prije ikakvog izvršavanja prvo moramo znati što je to zapravo riječ[16]. I još važnije, računalo to isto mora znati. Dakle, trebamo ga u nekom smislu "naučiti" prepoznavati riječi. Odnosno, potrebno mu je nekakvo znanje o prirodnom jeziku da bi tu operaciju/zadatak mogao izvršiti.

Upravo zato pri obradi prirodnog jezika veliku ulogu ima jezikoslovlje ili lingvistika. Logična posljedica te uloge jest da metode proučavanja strukture jezika iskoristimo u obradi prirodnog jezika na računalima. Koraci koje poduzimamo pri obradi prirodnog jezika redom uključuju:[24]

- leksičku analizu - identificiranje i analizu strukture riječi

- sintaktičku analizu - analizu strukture riječi u rečenici (gramatika)
- semantičku analizu - identificiranje značenja riječi i teksta
- integraciju diskursa - analizu značenja rečenice u odnosu na prethodnu rečenicu
- pragmatičku analizu - ponovnu interpretaciju onoga što se reklo kako bi se identificiralo ono što se mislilo reći.



Slika 0.2: Koraci pri analizi i obradi prirodnog jezika

U ovom radu ćemo proučiti svaki od tih koraka detaljnije, vidjeti podzadatke koje svaka od tih analiza obuhvaća te algoritme koji implementiraju svaku od tih vrsta analiza na računalima. Potom ćemo spomenuti razne primjene tih vrsta analiza koje smo obradili od kojih ćemo naglasak staviti na chatbot programe, kao što i sam naslov rada ukazuje. Uzevši u obzir veliku raznolikost chatbot funkcionalnosti, moguća kompetencija nekog konkretnog programa te vrste jest velika. Ipak, smisleno je vidjeti i konkretnu primjenu. Stoga će se studijskim primjerom pokazati konstruktivna korisnost chatbota u jednom realnom



problemu (vidjevši implementaciju ključnih dijelova), ali i pokazati neiscrpnost daljnjih mogućnosti koje takav program ima.

Prvo ćemo započeti s kratkim povijesnim pregledom razvijanja obrade prirodnih jezika kroz godine.

# Poglavlje 1

## Kratki povijesni pregled

Za mnoge, sposobnost računala da obrađuje i razumije prirodne jezike približno dobro kao čovjek signaliziralo je dolazak "inteligentnih" strojeva. Korijen tog uvjerenja jest u tome da je ljudski jezik jako povezan s našom općom kognitivnom sposobnošću. Među prvima koji je to primijetio bio je Alan Turing. U svom poznatom radu "Računalna tehnologija i inteligencija"[29] iz 1950. godine, Turing je uveo nešto što mi danas nazivamo "Turingov test". On je obrazložio da na pitanja "Što to znači da je stroj ili računalo inteligentno" ili "Što znači da računalo misli" nije moguće dati odgovor zbog nedovoljno preciznih i egzaktnih definicija pojmova "stroj", "računalo", "inteligencija" i "misliti". Umjesto toga, predložio je test, igru, u kojoj bi se preko računalne obrade jezika omogućilo provjeru inteligencije računala. Ako računalo pobjedi igru, smatralo bi se inteligentnim. U toj igri, tri su sudionika: dva čovjeka i računalo. Jedan čovjek je u ulozi ispitivača i on pokušava dokučiti na temelju njihovih odgovora koji je od preostala dva sudionika računalo. Zadaća računala bi, dakle, bila "prevariti" ispitivača tako da on povjeruje da je osoba, a ne računalo. Zadaća drugog sudionika bi bila da uvjeri ispitivača da je on čovjek i da je drugi sudionik računalo. Iz tog rada, imamo primjer interakcije kakvu je Turing imao na umu.

- **Q:** Please write me a sonnet on the subject of the Forth Bridge.
- **A:** Count me out on this one. I never could write poetry.
- **Q:** Add 34957 to 70764.
- **A:** (Pause about 30 seconds and then give as answer) 105621.
- **Q:** Do you play chess?
- **A:** Yes.

- **Q:** I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?
- **A:** (After a pause of 15 seconds) R-R8 mate.

Turing je predviđao da će krajem 20. stoljeća računalo s 10 GB memorije imati vjerojatnost uspjeha od 30% u zavaravanju ispitivača nakon 5 minuta ispitivanja.

Nedugo, 14 godina nakon, pokazala se relevantnost onoga što je Turing iznio u svom radu zbog razvijenog programa pod nazivom *ELIZA*, napisanog od strane Josepha Weizenbauma. Taj sustav je simulirao psihoterapeuta i bio je jedan od najranijih sustava za obradu jezika sposoban za ograničenu interakciju s čovjekom. Interakcija je bila ograničena zbog vrlo oskudnog vokabulara.

Ipak, bez korištenja ikakvih informacija o ljudskom razmišljanju i emocijama, *ELIZA* je mogla pružiti interakciju nalik ljudskoj. Kada "pacijent" nadiđe njezinu vrlo malenu bazu znanja, *ELIZA* bi mogla pružiti generički odgovor, primjerice odgovarajući na "Boli me glava" pitanjem "Zašto kažete da Vas boli glava?" Slijedi primjer jednog takvog razgovora:

- **User:** You are like my father in some ways.
- **ELIZA:** WHAT RESEMBLANCE DO YOU SEE
- **User:** You are not very aggressive but I think you don't want me to notice that.
- **ELIZA:** WHAT MAKES YOU THINK I AM NOT AGGRESSIVE
- **User:** You don't argue with me.
- **ELIZA:** WHY DO YOU THINK I DON'T ARGUE WITH YOU
- **User:** You are afraid of me.
- **ELIZA:** DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU

Uspjeh ovog jednostavnog programa u ovom području je zbog činjenice da zapravo ne treba znati gotovo ništa za imitirati psihoterapeuta ovakvog tipa. Kao što kaže sam Weizenbaum, ovo je jedan od načina pristupa te struke gdje slušatelj u dijalogu smije djelovati kao da ne zna ništa na svijetu. *ELIZA* je jako važna za Turingove ideje jer mnogi ljudi koji su komunicirali s *ELIZOM* su i dalje vjerovali da ih stvarno razumije i da shvaća njihove probleme. Weizenbaum zapaža da su mnogi od tih ljudi i dalje vjerovali u Elizu čak i nakon što im je objašnjen način rada. Ono o čemu je Turing pisao, 14 godina kasnije je Weizenbaum praktički pokazao kao naznaku i početak moderne računarske znanosti kakvu danas poznajemo. Ostaje vidjeti dvije glavne paradigme koje su usmjeravale povijesni razvoj prirodne obrade jezika.

## Obrada prirodnog jezika temeljena na pravilima

Ova paradigma započela je 1950-ih godina na temelju ranijeg Turingovog rada i modela Turingovog stroja. Taj rad potaknuo je McCulloch-Pitts istraživanje o neuronima[20]. Za svoj pojednostavljeni model neurona držali su da bi mogao postati model po kojem računala nešto izračunavaju. Slijede radovi Kleenea iz 1951. i 1956. godine o konačnim automatima i regularnim izrazima. Na temelju Shannonovog rada iz 1948. godine u kojem je primijenio vjerojatnosne metode diskretnih Markovljevih lanaca na automate u jeziku, Chomsky je dobio ideju da modele strojeva s konačno mnogo stanja iskoristi za karakterizaciju gramatike. To je vodilo do razvitka teorije formalnih jezika - teorije koja je koristila algebru i teoriju skupova kao alate za definiranje formalnih jezika preko niza simbola. Ovo uključuje beskontekstne gramatike, koje je prvi definirao Chomsky. Neovisno o njemu, ponovno su otkrivene od strane Backusa 1959.godine i Naura 1960.godine u svome opisu programskog jezika *ALGOL*[22].

Već početkom 1960-ih polako nastaje i druga paradigma - stohastička (statistička). Prva paradigma nastavlja u smjeru Chomskyjevog rada i generativne sintakse te u smjeru mnogih istraživanja i radova na području algoritama za parsiranje. Inicijalno su to bili dinamički algoritmi s "Odozgo prema dolje" (eng. *Top-down*) i "Odozdo prema gore" (eng. *Bottom-up*) pristupom. Početkom 1970-ih nastaju nove dvije paradigme koje se nastavljaju na prethodnu. Prva je paradigma bazirana na logici koja je započeta radom Colmerauer i njegovih kolega na Q-sustavima i metamorfoznoj gramatici - što je bila preteča programskom jeziku Prolog. Paralelno, i neovisno, za to vrijeme Kayov rad na funkcionalnoj gramatici 1979. godine, i malo nakon Bresnanov i Kaplanov rad 1982. godine na leksičko-funkcionalnoj gramatici, pokazali su važnost unifikacije. I time je polako započela druga paradigma - razumijevanje prirodnog jezika (engl. *natural language understanding*).

Početak je popraćen Winogradovim SHRDLU sustavom koji je simulirao rad robota u svijetu kocaka za djecu. Program je mogao prihvatiti naredbe zadane u prirodnom jeziku, poput "Pomakni crvenu kocku na manju, zelenu kocku". Roger Schank, njegovi kolege i studenti (često se nazivaju engl. *Yale School*) izgradili su cijelu seriju programa za razumijevanje jezika koji bi strukturirali znanje iz stvarnog svijeta u podatke koji su razumljivi računalu. To je niz radova: Schank i Albelson, 1977; Schank i Riesbeck, 1981; Cullingford, 1981; Wilensky, 1983; Lehnert, 1977.

## Statistička obrada prirodnog jezika

Sve do 1980-ih, većina sustava za obradu prirodnog jezika bazirala se na kompleksnim skupovima ručno pisanih pravila. Kasnih 1980-ih dolazi do revolucije u obradi prirodnog jezika s uvođenjem algoritama strojnog učenja za obradu jezika. Za ovo su zaslužni postojan rast računalne moći (tzv. Mooreov zakon) i postupno popuštanje dominacije Chomskyjevih teorija lingvistike čije su teoretičarske pretpostavke obeshrabrivale uporabu lingvistike

korpusa koja je temelj pristupa obradi jezika temeljenog na strojnom učenju. Neki od najranije korištenih algoritama strojnog učenja, kao što su stabla odluke, generiraju sustave čvrstih ako-onda pravila sličnih postojećim ručno pisanim pravilima.

Međutim, označavanje vrsta riječi u rečenicama uvodi uporabu skrivenih Markovljevih modela u područje obrade prirodnog jezika i istraživanja se sve više usredotočuju na statističke modele koji stvaraju meke probabilističke granice temeljene na povezivanju težina sačinjenih od realnih brojeva sa značajkama iz ulaznih podataka. Cache jezični modeli, na kojima se danas temelje mnogi sustavi za raspoznavanje govora, primjer su takvih statističkih modela. Takvi modeli su u pravilu robusniji kad rade s nepoznatim ulaznim podacima, posebice onima koji sadrže pogreške (što je vrlo često kod stvarnih podataka), i daju pouzdanije rezultate kad su integrirani u velik sustav koji se sastoji od brojnih podzadataka.

Mnogi rani uspjesi zabilježeni su u području strojnog prevođenja, posebice zahvaljujući radu IBM Research-a, gdje su razvijeni znatno složeniji statistički modeli. Ovi sustavi koristili su višejezične korpuse tekstova koje su sastavili Kanadski parlament i Europska Unija kao rezultat zakona koji zahtijevaju prevođenje svih vladinih odluka na sve službene jezike. Međutim, većina drugih sustava ovisi o korpusima teksta koji su izgrađeni specifično za zadatak na koji se ti sustavi primjenjuju, što je veliko ograničenje ovakvih sustava. Kao rezultat toga, velik broj istraživanja bavi se metodama koje omogućavaju efektivnije učenje na temelju ograničenih količina podataka.

Nakon kratkog povijesnog pregleda, slijedi analiza obrade prirodnog jezika. Kada pričamo o analizi obrade, treba reći kako će se u ovom radu većinom staviti naglasak na analizi obrade hrvatskog i engleskog jezika.

## Poglavlje 2

# Tokenizacija

Tekst u elektroničkom obliku nije ništa drugo nego niz znakova (karaktera). Znakovi mogu biti: slova abecede, brojevi, interpunkcijski znakovi, bjeline, znak za kraj reda, itd. Ti znakovi sami po sebi gotovo ništa ne predstavljaju u pisanom obliku (nemaju intrinzično značenje) i prije bilo kakve daljnje analize teksta, potrebno je taj tekst pretvoriti u jezikoslovne jedinice.

Taj jednostavan koncept prirodne obrade jezika gdje tekst pokušavamo raščlaniti u neke smislenije segmente naziva se tokenizacija (opojavničenje). A svaki segment zasebno nazivamo *token* (pojavnica). Neke smislene jezikoslovne jedinice su: riječi, sintaktičke grupe, rečenice, paragrafi... Kada pišemo tekst našim prirodnim jezikom tada nam prirodno dolazi razdvajati napisane riječi razmakom. Prema tome, prikladno je da se prvenstveno fokusiramo na riječi. Tokenizacija naizgled djeluje jednostavna u jezicima poput hrvatskog i engleskog. Primjerice, u programskom jeziku Python vrlo se lako razdvoji tekst po razmacima jer postoji već ugrađena metoda `.split()` koja prima odvajač (separator) kao parametar po kojem želimo rečenicu razdvojiti:

- **Ulaz:** Tolstoj je napisao: „Sve sretne obitelji nalik su jedna na drugu, svaka nesretna obitelj nesretna je na svoj način.”

- **Izlaz:** Tolstoj je napisao: „Sve sretne obitelji nalik su jedna na drugu, svaka nesretna obitelj nesretna je na svoj način.”

Međutim, već u ovom primjeru vidimo kako postoje pravopisni znakovi koji su spojeni s riječima. Postavlja se pitanje što sa znakovima, hoćemo li ih promatrati odvojeno? Sigurno da odgovor ovisi o problemu kojim se bavimo, pa čak i o prirodnom jeziku kojeg obrađujemo. No, ako i izuzmemo ostale jezike u kojima je tokenizacija razmakom nemoguća (kineski, japanski, ..), situacija, čak ni tada, nije baš tako jednostavna.

## Što je riječ

Naime, već smo vidjeli kako se u gornjem primjeru pravopisni znakovi nalaze odmah kraj riječi i nema razmaka između njih. Može se nametnuti ideja potpunog zanemarivanja pravopisnih znakova u tokenizaciji. Ipak, ta ideja nije preporučljiva jer oni mogu ukazati na kontekst riječi unutar rečenice/teksta. Naprimjer, dvotočka i navodnici nam ukazuju na upravni govor i to nam ukazuje da netko izgovara nekakvu rečenicu. Točka može ukazivati na kraj rečenice i slično.

Jedan od pristupa je zadržati one znakove koje označavaju kraj rečenica, ali sve unutar-rečenične znakove zanemariti. Ni to nije baš mudar pristup. To vidimo na primjeru rečenice: *Upisala se na studij 2005./2006., a diplomirala je 2009./2010.*. Ovdje jedna riječ sama *2005./2006.* sadrži tri unutar-rečenična znaka i kada bi ih zanemarili, ta riječ bi izgubila svoj smisao.

Dakle, pitanje "što podrazumijevamo pod riječ" je poprilično kompleksno. Jezikoslovci ponekad tvrde kako se definicija riječi razlikuje ovisno o kontekstu, kao što su morfosintaktičke riječi i fonološke riječi. Koju bi definiciju onda trebao koristiti računarski jezikoslovac? Kučera i Francis su 1967. predložili da se riječ u engleskom jeziku definira kao uzastopni niz slovo-brojčanih znakova (alfanumerički znakovi) s bjelinom sa svake strane koji može sadržavati: spojnice, polunavodnike i nikakve druge pravopisne znakove[2]. Nažalost, i to se pokazalo nedostatnom definicijom, pa čak i za praktične svrhe. Brojevi i novčani iznosi tretiraju se kao riječi, a po prethodnoj definiciji izraz *600.000 kuna* ne spada pod riječ. Stvari postanu još kompliciranije kada analiziramo nekakav tekst s internetskih stranica pa riječi postanu čak i kombinacije pravopisnih znakova koje predstavljaju osjećajnike (engl. *emoticon*). S obzirom na to kako postoje mnoge iznimke u definiranju riječi isključivo preko bjelina, fokusirat ćemo se na par glavnih problema.

## Točke

Već smo ustanovili kako riječi nisu uvijek okružene bjelinama nego je često na riječ "privezana" točka, zarez, dvotočka, trotočka. Iako većina točaka označava kraj izjavne rečenice, postoje one koje imaju drugo značenje:

1. iza rednih brojeva napisanih arapskim i rimskim brojkama:
  - Završila sam 2. razred.
  - Petar Krešimir IV. jedan je od najslavnijih hrvatskih vladara.
2. iza većine kratica: br., dr. sc., i sl., itd., npr., prof., tj.
3. iza primjera koji je izjavna rečenica u rečenici koja ga objašnjava: *U rečenicama Marko jede jabuku., Učitelji su ga pohvalili. i Brod je potopljen. subjekti su imenice Marko, učitelji i brod.*

4. ispred drugih pravopisnih znakova:
  - Sretna 2013.!
  - Možeš li doći do 15.?
5. između podataka u pojedinim sustavima pisanja bibliografskih jedinica: *Vukojević, Luka. 2009. Izražavanje posljedičnih odnosa u hrvatskome standardnom jeziku. Institut za hrvatski jezik i jezikoslovlje. Zagreb.*

### Izostavnik

Izostavnik ili apostrof (engl. *Apostrophe*) je pravopisni znak posebno karakterističan za engleski jezik. U hrvatskom jeziku se koristi samo u situacijama u kojima se označuje da je nešto izostavljeno u brzome ili neformalnemu govoru: *Drž'te ga!, Je l' došao?.* U engleskom jeziku se pojavljuje kada se riječi (glagoli) skraćuju "gutanjem" slova kao što imamo u primjerima *I'll* i *isn't* ili kada želimo naglasiti posjedovanje: *in one month's time*. Teško je odrediti na koji način promatrati ovakve lekseme. S jedne strane po gornjoj definiciji spada u riječ, no naša nam intuicija govori kako su to zapravo dvije riječi. U gornjim primjerima tako *I'll* zapravo predstavlja *I will*, dok *isn't* predstavlja *is not*. Neki procesi obrade, ovisno o problemu kojeg rješavamo, to "shvaćaju" kao jednu riječ, a drugi pak kao dvije. I jedan i drugi pristup imaju svoje mane. Prvom pristupu je mana što prestaje vrijediti tradicionalno sintaksno pravilo ( $S \rightarrow NP VP$ ) i to narušava gramatičku strukturu. Drugom pristupu je problem što separacijom dobijemo čudne izraze 's i n't.

### Spojnicica

Spojnicica (–) pravopisni je znak koji se piše bez bjelina slijeva i zdesna. Premda nema bjelina ni s jedne strane, postavlja se pitanje predstavlja li niz znakova koji sadrži spojnicu jednu riječ ili dvije. Situacija nam ponovno govori - ovisi. I to ponajviše o tome u koju smo je svrhu koristili jer se spojnicom piše:

1. dvije sastavnice od kojih se prva ne sklanja, koje označuju jedan pojam, a svaka ima svoj naglasak: *aperitiv-bar, beta-inačica, džin-tonik, zemlja-zrak, ping-pong*
2. dvije sastavnice od kojih je prva oznaka, simbol, kratica ili broj:
  - a-mol, e-pošta, H-dur, i-sklonidba, T-profil, x-faktor
  - pH-vrijednost, Rh-faktor
  - 18-karatni, 22-inčni, 8-bitni, 25-godišnjica, 2,8-postotni, 18-godišnjak
  - n-ti, n-torka, x-ti



3. najmanje dvije sastavnice od kojih se samo zadnja sklanja, koje označuju najmanje dva pojma, a svaka ima svoj naglasak: *englesko-njemačko-hrvatski, fizičko-kemijski, Michelson-Morleyev, političko-ekonomski, splitsko-dalmatinski*
4. riječi nastale povezivanjem dviju oprečnih sastavnica: *hoćeš-nećeš, kakav-takav, lijevo-desno, manje-više, povuci-potegni*
5. riječi nastale povezivanjem brojevnih riječi ili imenica i brojevnih riječi kojima se izražava približna vrijednost: *dva-tri, tisuću-dvije, čašica-dvije, dan-dva, sat-dva*
6. najmanje dvije nesamostalne sastavnice: *cakum-pakum, ho-ruk, šuć-muć, tip-top*
7. usklici koji se ponavljaju ili spajaju: *av-av, kuc-kuc, z-z-z, bim-bam-bom, klik-klak, tik-tak*
8. padežni oblici i tvorenice u kojima je prva sastavnica broj ili pokrata riječ koja se piše velikim slovima nastala kraćenjem najčešće višerječnoga naziva ili imena: *EU-a, INA-e, NATO-a, 30-ak, 80-ih, PMF-ovac*
9. riječ koje se dio prenosi u novi redak

*Tokenizer* bi zapravo trebao pokrivati sve slučajeve. Slučaj 9. bismo mogli riješiti gledajući samo spojnice koje se nalaze na kraju linije, maknuti ih te spojiti dijelove riječi s kraja i početka nove linije. Ni to rješenje nije savršeno jer ukoliko bi imali neku od složenica kao zadnju riječ u liniji, u tom slučaju bi se pisala samo jedna spojnica, a ne dvije (jedna zbog nove linije, jedna zbog složenice). Kod nekih od preostalih slučajeva je očito, naprimjer 2. slučaj gdje svakako ne želimo razdvajati jedno slovo od riječi i još time izgubiti izvorni smisao. U slučajevima 4. i 5. svakako želimo razdvajati s obzirom na to da bi u protivnom vokabular mogao znatno porasti, dok u 8. slučaju želimo razdvojiti i odbaciti padežni oblik.

Ovdje su obrađeni neki od glavnih problema koje pravopisni znakovi mogu predstavljati u tokenizaciji. Postoje i mnogi drugi pravopisni znakovi poput dvotočke, trotočke, zagrada, kosih crta, itd. koje ovdje nismo detaljno obradili.

## Tokenizacija u drugim jezicima

Jezici kao što su kineski, japanski i tajski ne koriste bjeline kao graničnu podjelu za odrediti što je riječ.

U kineskom, primjerice, riječi se sastoje od niza znakova koji se zovu *hanzi*. Svaki od tih znakova generalno predstavlja morfem i izgovara se kao jedan slog. Riječi se u prosjeku

sastoje od 2.4 takva znaka. Jednostavan algoritam koji nam pomaže u segmentaciji kineskog jezika, a podloga je i za mnoge druge kompleksnije algoritme, jest pohlepna verzija algoritma koji se naziva MAXMATCH algoritam.

Rad algoritma zahtijeva rječnik riječi iz jezika. Algoritam započinje postavljanjem pokazivača na početak nekakve zadane riječi. Pokazivač nastavlja "hodati" po nizu znakova tražeći najdulji podniz koji se nalazi i u rječniku. Ako ne postoji takav podniz, tada uzima samo prvo slovo, kreirajući riječ od jednog slova. Algoritam tako radi iterativno svaki puta krenuvši od te nove pozicije pokazivača nakon što smo pronašli riječ unutar rečenice.

Kako bi se pomoglo vizualizirati taj algoritam, Palmer je 2000. godine iskazao primjer analogije unutar engleskog jezika, primijenivši algoritam na *thetabledownthere*. Algoritam bi, dakle, prvo pronašao riječ *theta* kao najdulju riječ koja se nalazi u nizu znakova i u rječniku. Pokazivač bi nastavio od nove pozicije i tako iterativno pronašao redom *bled*, *own*, *there*. Kao što vidimo to nije točan rezultat zbog toga što rečenica *theta bled own there* nije smisljena. Rezultat je trebao biti *the table down there*. Algoritam puno bolje funkcionira za kineski jezik nego za engleski. Kao razlog tome možemo pripisati što algoritam ne radi dobro za jezike koji sadrže duge riječi što je u našem primjeru bio slučaj. No, primjena u kineskom jeziku također sadrži ima par slabih točaka, poput: rad na dosad nepoznatim riječima (onih kojih nema u rječniku).

---

#### Algoritam 1 MAXMATCH

---

**Ulazni podaci:** rečenica *rec* i rječnik *dict*

**Izlazni podaci:** lista riječi

```

if rečenica rec prazna then
    return prazna lista
end if
for  $i \leftarrow \text{length}(rec)$  to 1 do
    prva_ri_jec  $\leftarrow$  prvih  $i$  znakova rečenice rec
    ostatak  $\leftarrow$  ostatak rečenice
    if uRječniku(prva_ri_jec, dict) then
        return list(prva_ri_jec, MAXMATCH(ostatak, dict))
    end if
end for
prva_ri_jec  $\leftarrow$  prvi znak rečenice rec
ostatak  $\leftarrow$  ostatak rečenice
return list(prva_ri_jec, MAXMATCH(ostatak, dict))

```

---

Postoje natjecanja na godišnjoj bazi (engl. *bakeoff annual competition*) u algoritmima za segmentaciju kineskog jezika algoritmima, i oni obično koriste sekvencijalne modele trenirane uz nadzirano strojno učenje na ručno segmentiranim skupovima za treniranje.

## Poglavlje 3

# Morfološka analiza

Većina podataka na računalima se čuva u tekstu. Kako bismo tim podacima brzo i jednostavno pristupali, moramo razvijati tehnologije pogodne pretraživanju teksta. Fiktivan primjer: zamislimo da iz mnogo novinskih članaka (koje imamo spremljene na računalo), želimo izvući neke članke o određenoj temi, recimo bankarstvu. Za pretragu nije dosta samo pretražiti pojam *bankarstvo* već i pojmove *bankarstvu*, *bankarstvom*, *bankari*, *banci*, *banka*, ... Drugim riječima, trebala bi nam u nekom smislu morfološka normalizacija riječi *bankarstvo*.

Razlikujemo dva osnovna pristupa morfološkoj normalizaciji: lematizacija i korjenovanje. Lematizacija je postupak kojim se pronalazi kanonski oblik riječi, tj. natuknički oblik u rječniku ili *lema*. Lematizator bi oblike *bankarstvo*, *bankarstvu*, *bankarstvom*, *bankari* trebao svesti na lemu *bankar*. Korjenovanje je pak postupak kojim se uklanjaju afiksi iz različitih oblika riječi kako bi se pronašao „korijen” koji je zajednički svim oblicima. Tako dobiven „korijen” češće je zapravo jednak osnovi riječi (ne treba biti riječ sama po sebi), pa se neki autori koriste nazivom *pseudokorijen*.

### Korjenovanje

Kao što smo rekli, u ovom pristupu morfološke normalizacije želimo pronaći ”korijen” zajednički svim oblicima i on ne mora nužno biti smisljena riječ. Računarsko rješenje za ovaj pristup morfološkoj normalizaciji nazivamo korjenovatelj. Primjer: Kod riječi *župni*, *župnog*, *župnim*, *župa*, *župe*, *župi*, *župana*, *županom*, *županje*, *župe*, *banka*, korjenovatelj bi trebao izbaciti pseudokorijen *žup*.

Treba istaknuti da korjenovanje može obuhvatiti i flektivnu i derivacijsku morfološku varijaciju, tako da može dovesti do različitih stupnjeva normalizacije. Ako korjenovanje obuhvaća samo flektivnu morfološku varijaciju, govorimo o slabome (konzervativnome) korjenovanju, a ako obuhvaća i derivacijsku morfološku varijaciju, govorimo o jakome (agresivnome) korjenovanju. Postupak bi slaboga korjenovanja oblike *članstvo*, *članstva*,

*članstava* i *članstvima* trebao svesti na pseudokorijen *članstv*, a postupak jakoga korjenovanja iste bi oblike trebao svesti na pseudokorijen *član*[8].

### Pristupi korjenovanju

Pristupe morfološkoj normalizaciji možemo podijeliti na: korjenovanje temeljeno na pravilima, hibridno korjenovanje i nenadzirane pristupe korjenovanju. Korjenovanje temeljeno na pravilima čine algoritmi temeljeni na pravilima koji pronalaze pseudokorijen riječi primjenom niza ručno kodiranih pravila odsijecanjem afiksa (najčešće sufiksa). Hibridno korjenovanje također upotrebljava pravila odsijecanja sufiksa, ali smanjuje broj pogrešaka korištenjem rječnika. Nenadzirani pristupi korjenovanju rječnike ili pravila korjenovanja induciraju automatski iz neoznačenih korpusa.

Najraširenije od navedenih pristupa jest korjenovanje temeljeno na pravilima. Prednost ove vrste u prvome je redu njegova jednostavnost i brzina izrade. Za razliku od ostalih pristupa, za korjenovanje temeljeno na pravilima nisu potrebni ni leksički resursi kao što su morfološki leksikoni, ali ni specijalizirano znanje o strojnome učenju.

### Algoritmi za korjenovatelje u engleskom jeziku

**Porterov algoritam:** Porterov algoritam baziran je na ideji da su sufiksi u engleskom jeziku formirani kombinacijom manjih i jednostavnijih sufiksa. Sastoji od pet stadija koji se odvijaju jedan za drugim. Jedan od najučinkovitijih i najčešće korištenih algoritama za korjenovanje engleskoga jezika zbog svoje brzine i jednostavnosti. Algoritam je poprilično dugačak i star (1980. godina).

- **Primjer:** Jedno od pravila glasi  $EED \rightarrow EE$  i ono znači "ako riječ sadrži barem jedan samoglasnik i suglasnik te sufiks EED, promijeni sufiks u EE.  
Tako npr. 'agreed' postaje 'agree'.
- **Prednosti algoritma:** Daje najbolje rezultate u usporedbi s ostalim algoritmima za korjenovatelja i ima nisku stopu pogreške.
- **Mane algoritma:** Pseudokorijen često nije smisljena riječ engleskog jezika.

**Lovinsonov algoritam:** Prvi algoritam za korjenovanje riječi bio je Lovinsonov. Ime je dobio po Julie Beth Lovinson i njezinom radu iz 1968. godine. Algoritam je poprilično tehničke vrste i znatno je veći od Porterovog zbog velike liste sufiksa. Sadrži 294 različita nastavka, 29 uvjeta i 35 transformacijskih pravila. Algoritam radi tako da je svaki nastavak povezan s jednim uvjetom. Na početku tražimo najduži nastavak koji zadovoljava svoj uvjet te ga potom mičemo. U drugom koraku primjenjujemo transformacijska pravila.

- **Primjer:** *sitting* → *sitt* → *sit*
- **Prednosti algoritma:** Brži je od Porterovog te pokriva nepravilne glagole i sličan tip riječi.
- **Mane algoritma:** Relativno spor.

**Snowball algoritam:** Mnogo implementacija Porterovog algoritma napisane su i distribuirane kroz vrijeme. Međutim, svaka od tih implementacija je sadržavala suptilne greške. Kao rezultat, takav korjenovatelj nije dosegao svoj potencijal. Da bi se eliminirale te greške, Martin Porter je napravio preinaku algoritma, kojeg možemo praktički zvati Porter2 algoritam. Nastavio je svoj rad i kroz idućih par godina nastao je Snowball - radno okruženje za pisanje korjenovatelja. U sklopu toga izrađen je unaprijeđeni korjenovatelj za engleski jezik, ali i za neke druge jezike.

- **Primjer:** *sportingly* → *sport*
- **Prednosti algoritma:** Zbog preinaka koje su dodane u odnosu na Porterov, ima puno veću brzinu.
- **Mane algoritma:** Znatno je agresivniji i jači od Porterovog što može biti mana.

**Lancasterov algoritam:** Lancasterov algoritam je vrlo agresivan algoritam. Nadalje, krajnje riječi koje dobijemo Porterovim algoritmom (čak i Snowball) su najčešće razumljive, dok s ovim algoritmom, naročito s kraćim riječima, dobijemo prilično nejasan pseudokorijen. Također, ovaj algoritam je najbrži od do sada navedenih te će mnogo skratiti radni set riječi.

Sve ovisi o potrebi zadatka koji algoritam želimo koristiti. Postoje još neki (manje bitni) koje ovdje nismo naveli (Dawsonov, Krovetzov, ...).

- **Primjer:** *caring* → *car*
- **Prednosti algoritma:** Najbrži algoritam, te može znatno smanjiti radni set riječi ako je potrebno i to želimo.
- **Mane algoritma:** Najagresivniji među algoritama.

### Korjenovatelj u hrvatskom jeziku

U radu Ivana Pandžića *Oblikovanje korjenovatelja za hrvatski jezik*[8], vidimo neka dostupna rješenja za morfološku normalizaciju hrvatskog jezika. Ističe se rad Ljubešića i drugih iz 2007.godine. Do te navedene godine izrađena su samo dva korjenovatelja za hrvatski jezik i to za pretraživanje informacija. Prvi takav osmislila je dr.sc. Tomislava Lauc napisavši pravila za imeničke i pridjevske paradigme. Taj korjenovatelj je postigao preciznost od 90,26% (bio je testiran na leksikonu, a ne na korpusu). Drugi korjenovatelj, izrađen u svrhu pretraživanja tekstova Narodnih novina Republike Hrvatske, napravio je Dobrica Pavlinušić. Učinkovitost tog korjenovatelja nije moguće procijeniti s obzirom na to da nije bio kvantitativno ispitan. Još preostaje spomenuti rad Šnajder i Bašić iz 2009. godine[31] koji opisuju korjenovanje temeljeno na udaljenosti znakovnih nizova.

### Lematizacija

Lematizacija, za razliku od korjenovanja, reducira riječ na njezin osnovni oblik. Taj osnovni, kanonski oblik mora biti riječ koja je dio jezika i nazivamo je lema. U hrvatskom jeziku lema neke imenice je njezin nominativ jednine, a lema glagola infinitiv. Primjerice, neki od oblika leme *hodati* su: *hodam*, *hodao*, *hodamo* itd. U engleskom jeziku lematizacija, za razliku od korjenovanja, pronalazi lemu čak i kod nepravilnih izraza. Naprimjer, kada imamo nepravilni glagol u prošlom vremenu *was*, lema je *be*. Ponekad nije moguće jednoznačno odrediti lemu neke riječi. Tako u primjeru riječi *vode* imamo sljedeće leme: *vod*, *voda*, *voditi*. Iz prethodnog vidimo kako nam je potrebno najprije odrediti vrstu riječi ovisnu o kontekstu kako bi mogli jednoznačno odrediti lemu riječi. Taj proces jednoznačnog određivanja vrste riječi nazivamo gramatičko tagiranje.

### Gramatičko tagiranje

U sklopu morfološke analize treba još obraditi gramatičko tagiranje. Ukratko, gramatičko tagiranje (engl. *part-of-speech tagging*) jest pridruživanje oznake za vrstu riječi pojavnica (*token*) u korpusu[6]. Zbog čega je taj proces bitan? Naime, već je više puta u ovom radu spomenuto (čak je i iz same strukture rada očito) kako se računalna obrada prirodnog jezika odvija sekvencijalno. Svaki korak nam služi kao podloga u svrhu daljnje obrade. Kako je idući korak parsiranje gramatike koje ne bi bilo moguće bez označenih pojava, gramatičko tagiranje svrstavamo u potproces morfološke analize, iako je znatno drukčiji od korjenovanja i lematizacije koji se bave afiksima. Osim što je bitno za daljnju analizu, tagiranje je generalno korisno zato što vrste riječi otkrivaju mnogo informacija o samoj riječi i riječima koje su joj susjedne. Naprimjer, označivači za engleski jezik imaju najčešće različite oznake za posvojne zamjenice (*my*, *your*, *his*, *her*, *its*) i osobne zamjenice (*I*, *you*, *he*, *me*). Saznanjem je li riječ posvojna ili osobna zamjenica možemo pretpostaviti koja će

se riječ vrlo vjerojatno pojaviti iduća unutar rečenice jer je posvojna zamjenica najčešće popraćena imenicom, dok je osobna najčešće glagolom.

Sad se nameće jedno prirodno pitanje? Koje oznake sve mogu postojati u gramatičkom tagiranju? I kako ih odrediti? S lingvističke strane gledišta, zajednički je konsenzus da postoje tri glavne kategorije riječi: imenice, glagoli i pridjevi (Pustet, 2003.). Ovaj minimalni skup od tri kategorije se smatra univerzalnim. Osim te glavne leksičke skupine kategorija, lingvisti predlažu i neke sekundarne skupine (priloge, prijedloge,..) , te potkategorije primarnih kategorija (Anderson, 1997; Taylor, 2003). Spomenute potkategorije najčešće uključuju distinkciju na morfosintaktičkoj razini, recimo po rodu (muški, ženski), broju(jednina, množina), vremenu (glagol u prvom licu jednine..) i sl. Hrvatski jezik je drukčiji od engleskog pa je logično da će i označivači imati različite oznake za riječi. Dok engleski sadrži neke kategorije koje hrvatski nema, primjerice posebnu za odrednice (*a, the,..*), tako u hrvatskom jeziku razlikujemo veći skup glavnih kategorije. U hrvatskom jeziku razlikujemo promjenjive i nepromjenjive riječi. Promjenjive su: imenice, glagoli, pridjevi, zamjenice i brojevi. A nepromjenjive su: prilozi, prijedlozi, usklici, veznici i čestice. Potkategorije u hrvatskom postoje samo za promjenjivu vrstu riječi.

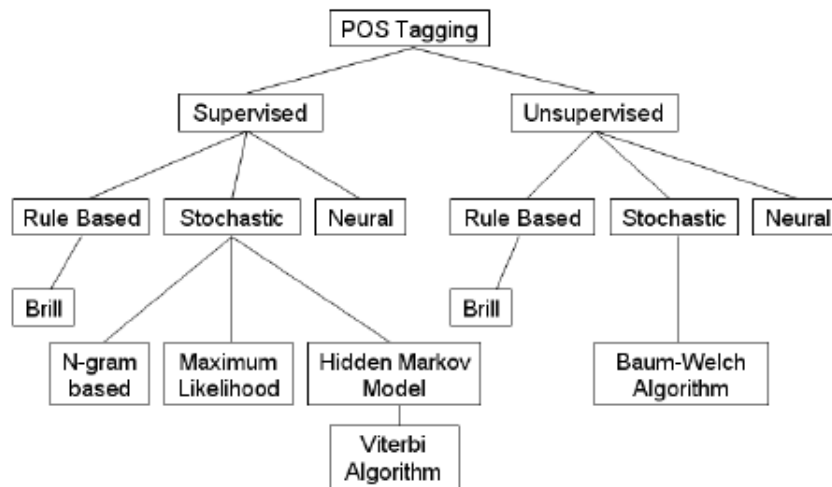
Kako računalna obrada jezika zahtijeva i praktičnu analizu, ne možemo kod određivanja kategorija pristupati isključivo s lingvističke perspektive. Zbog toga se označivači često razlikuju u veličini skupa kategorija, ovisno o potrebi. Zgodno je navesti par najpoznatijih označivača (za engleski jezik). Svakako vrijedi istaknuti *TAGGIT* kojeg su razvili Greene i Rubin 1971. godine. Bio je osmišljen za osnovno označavanje Brown korpusa. Sastojao 87 kategorije/oznaka za označavanje. Nakon toga se, kasnih 1970-ih, pojavio *CLAWS1* označivač koji je razvijen na Sveučilištu Lancaster za potrebe označavanja Lancaster-Oslo-Bergen korpusa. Imao je 135 oznaka od kojih je 23 bilo baznih. Vrijedi možda još samo spomenuti Penn Treebank and Wall Street Journal korpus s 48 oznaka.

Dolazimo do dijela kad ćemo definirati problem i vidjeti različite pristupe u izradi označivača. Postoji mnogo algoritama za ovaj problem. Mi ćemo vidjeti neke glavne ideje. Tako razlikujemo označivače temeljene na pravilima, probabilističke metode, te označivače bazirane na transformacijama. I ako pričamo o modernijim pristupima, neuronske mreže se također koriste.

### **Pristup izradi označivača**

Neka je  $W = w_1, w_2, \dots, w_n$  rečenica sastavljena od  $n$  riječi. Zadatak označivača je pronaći skup oznaka  $T = t_1, t_2, \dots, t_n$ , gdje je  $t_i$  gramatička oznaka za riječ  $w_i$ ,  $1 \leq i \leq n$ , što je točnije moguće[14].

Da bi odabrao odgovarajuću oznaku, označivač se koristi morfološkom strukturom, ali i kontekstom. U tu svrhu, označivač često uzima u obzir i okolne riječi koje se nalaze oko riječi čiju oznaku želi odrediti. Tako, koristeći se informacijama koje mu donosi taj kontekst riječi, pokušava što točnije odrediti. Recimo, ako je zadana riječ  $w_i$ , tada ćemo



Slika 3.1: Pristupi u izradi modela za označivače[13]

uzeti u obzir redom  $w_{i-2}, t_{i-2}, w_{i-1}, t_{i-1}, w_i$ . Većina se označivača koristi samo lijevim kontekstom (onim koji se nalazi lijevo od zadane riječi), no ima primjera gdje se koristi i desni kontekst. Prirodno pitanje koje dolazi jest zbog čega se ne uzme veći kontekst? Što je veći kontekst, bolje će se odrediti vrsta riječi. Slijedi primjer iz kojeg će se to očigledno vidjeti:

*the girls can..*

Vidimo kako u ovom primjeru ovaj uzeti kontekst još uvijek nije dovoljan za jednoznačno odrediti predstavlja li *can* u toj rečenici pomoćni glagol (npr. *the girls can do it*) ili glagol (npr. *the girls can the food*). Kada bi koristili širi kontekst

*The man who saw the girls can..*

jasno bi bilo da je *can* u ovom slučaju pomoćni glagol. Ipak, prevelik kontekst bi jako povećao "obujam" problema i pretvorio ga u problem još veće složenosti. Štoviše, uzimajući u obzir i tako relativno "malen" kontekst, dobivaju se rezultati visoke točnosti.

### Označivač temeljen na pravilima

Prvi algoritmi razvijeni za automatsko dodjeljivanje gramatičkih oznaka bazirani su na arhitekturi koja se odvija u dva stadija. Prvi stadij koristi rječnik kako bi svakoj riječi dodijelio potencijalnu listu oznaka. (Klein i Simmons, 1963). Drugi stadij koristi veliki skup ručno napisanih pravila (riječ je imenica, a ne glagol ako ispred nje stoji odrednica) kako bi gornju potencijalnu listu sveo na samo jednog kandidata tj. kako bi dodijelio jedinstvenu



oznaku svakoj riječi i riješio se višeznačnosti (Greene and Rubin, 1971). Moderni pristupi označivačima imaju sličnu arhitekturu, no rječnici i skup pravila je znatno veći.

Primjer ovakvog označivača je *ENGTWOL*. *ENGTWOL* je dvo-razinski morfološki analizator za engleski jezik koji za svaku riječ u tekstu daje u prosjeku dvije alternativne oznake. Taj označivač koristi princip gramatike ograničenja (engl. *Constraint Grammar*; *CG* pa se još ovaj označivač zna nazivati i *EngCG ENGTWOL*. Sadržavao je oko 56 000 korigiranih engleskih riječi (s tim da su se homonimi brojali kao odvojene riječi). Svaki od tih ulaza bio je označen kao što možemo vidjeti na primjeru iz slike 3.2.

Word	POS	Additional POS features
smaller	ADJ	COMPARATIVE
entire	ADJ	ABSOLUTE ATTRIBUTIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog's	N	GENITIVE SG
furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	IMPERATIVE VFIN
show	V	PRESENT -SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

Slika 3.2: Primjer jednog dijela EngCG označivača

Te oznake su se poslije koristile u pravilima i za većinu se da zaključiti iz kratice što predstavljaju. Tako je *SG* zapravo *singular* ili jednina, *-SG3* zapravo *third-person-singular* ili treće lice jednine, *ABSOLUTE* zapravo *non-comparative adjective* ili pridjevi koji se ne kompariraju itd.

U prvom stadiju, označivač prolazi kroz gore navedeni leksikon riječi i iz njega iščitava sve moguće oznake za neku riječ. U primjeru rečenice *Pavlov had shown that salivation*, označivač bi vratio listu mogućih oznaka

*Pavlov* **PAVLOV** *N NOM SG PROPER*  
*had* **HAVE** *V PAST VFIN SVO*  
**HAVE** *PCP2 SVO*  
*shown* **SHOW** *PCP2 SVOO SVO SV*  
*that* *ADV*  
*PRON DEM SG*  
*DET CENTRAL DEM SG*  
*CS*  
*salivation* *N NOM SG*

Vidimo kako za sve riječi ima višeznačne oznake. Potom bi slijedio drugi stadij u kojem bi pomoću pravila (ima ih otprilike 3,744) eliminirali sve višeznačne oznake koje su nedosljedne s kontekstom. Naprimjer, očito bi bilo kako *had* ovdje nije *past participle*, pa bi eliminirali PCP2 oznaku[16]. Na kraju, analognim postupkom i primjenom pravila, ostala bi na kraju jedinstvena oznaka za svaku pojedinu riječ rečenice.

### HMM Označivač

Označivač temeljen na pravilima s vremenom se počeo zamjenjivati stohastičkim modelima. Jedan od glavnih razloga tome bio je uvjet da pravila pišemo ručno što osim vremena zahtijeva i lingvističko znanje. Većina ljudi smatra kako su ovakvi modeli robusniji i superiorniji naspram onih temeljenih na ručno napisanim pravilima. Ovakvi modeli najčešće koriste Markovljeve lance, pa ćemo se većinom na njih i koncentrirati i označivač se zbog toga naziva HMM (engl. *hidden Markov model*) označivač.

Problem smo već prije formalno definirali. Dakle, neka je  $W = w_1, w_2, \dots, w_n$  rečenica sastavljena od  $n$  riječi. Zadatak označivača bi onda trebao biti pronaći skup oznaka  $T = t_1, t_2, \dots, t_n$ , gdje je  $t_i$  gramatička oznaka za riječ  $w_i$ ,  $1 \leq i \leq n$ , što je točnije moguće.

Točnost u ovakvoj vrsti modela možemo definirati kao maksimalna vrijednost vjerojatnosti:

$$P(T | W). \quad (3.1)$$

Po Bayesovom modelu, imamo nadalje

$$P(T | W) = \frac{P(W | T)P(T)}{P(W)}. \quad (3.2)$$

Stoga, problem pronalaska maksimalne točnosti oznaka možemo zapisati ovako

$$\begin{aligned} \arg \max_T P(T | W) &= \arg \max_T \frac{P(W | T)P(T)}{P(W)} \\ &= \arg \max_T P(W | T)P(T) \end{aligned} \quad (3.3)$$

Član  $P(W)$  smo izostavili jer je isti za sve  $T$ .

Kako je nezgodno direktno izračunati ova dva člana, jednadžba zahtijeva još pojednostavljenja. U prvom članu  $P(W | T)$  vidimo kako je svaka riječ neovisna o drugim riječima i nizovima oznaka, već samo ovisi o svojoj oznaci, pa možemo zapisati na sljedeći način:

$$\begin{aligned} P(W|T) &= P(w_1 \dots w_n | t_1 \dots t_n) \\ &= \prod_{i=1}^n P(w_i | t_1 \dots t_n) \\ &= \prod_{i=1}^n P(w_i | t_i) \end{aligned} \quad (3.4)$$

A drugi član možemo pojednostaviti na temelju činjenice kako svaka oznaka ovisi o posljednjih  $k$  oznaka:

$$\begin{aligned} P(T) &= P(t_1 \dots t_n) \\ &= P(t_1)P(t_2 | t_1)P(t_3 | t_1 t_2) \dots P(t_n | t_1 \dots t_{n-1}) \\ &= \prod_{i=1}^n P(t_i | t_{i-1} \dots t_{i-k}) \end{aligned} \quad (3.5)$$

Spomenuli smo već kako se za kontekst najčešće uzimaju dvije riječi iz lijevog konteksta, odnosno kad je  $k = 2$  i takav model zazivamo model trojavnice (engl. *trigram model*).  $P(W | T)$  vjerojatnost možemo nazivati leksička vjerojatnost jer se odnosi na leksičku formu riječi, a  $P(T)$  tranzicijsku vjerojatnost jer računa odnose između oznaka. Konačno, koristeći 3.4 i 3.5, problem možemo zapisati kao :

$$\arg \max_T \prod_{i=1}^n P(w_i | t_i)P(t_i | t_{i-1} \dots t_{i-2}). \quad (3.6)$$

Korisno je vidjeti na konkretnom primjeru[16] kako HMM označivač. Neka imamo rečenicu engleskog jezika za primjer:

*Secretariat is expected to race tomorrow.*

Očito je kako neke od tih riječi mogu imati višeznačnu oznaku. Recimo, riječ *race* može inače biti (osim glagola VB) i imenica. To vidimo u rečenici *People continue to inquire the reason for the race for outer space.*. Ovdje se očigledno vidi kako *race* podrazumijeva imenicu utrka. Nadalje, naš HMM označivač bi trebao izbaciti sljedeći rezultat kao krajnji ishod označavanja.

$$\underbrace{NNP}_{Secretariat} \quad \underbrace{BEZ}_{is} \quad \underbrace{VBN}_{expected} \quad \underbrace{TO}_{to} \quad \underbrace{VB}_{race} \quad \underbrace{NR}_{tomorrow}.$$

Zadatak označivača je da od svih mogućih hipotetskih kombinacija oznaka (kojih ima mnogo) izabere onu najtočniju. Iako više riječi u našem primjeru ima višeznačnu oznaku (*expected*), mi ćemo se fokusirati na riječ *race* i uzeti ćemo bez smanjenja općenitosti da ostale riječi imaju jednoznačnu oznaku. Naime, postupak bi se jednako razvijao samo uz mnogo više kombinacija. Dakle, "pravimo" se kao je cilj označivača u ovom primjeru dodijeliti oznaku VB ili NN riječi *race*. U tu svrhu ćemo koristiti formulu 3.6. Iz formule vidimo kako je potrebno izračunati sve vjerojatnosti oblika  $P(w_i | t_i)$  i  $P(t_i | t_{i-1} \dots t_{i-2})$ . Kako dobiti te vjerojatnosti? Direktno iz korpusa. Računanjem frekvencija unutar konkretnog korpusa možemo vidjeti koliko su "česte" neke pojavnice. U našem konkretnom slučaju, prvi član  $P(w_i | t_i)$  zapravo predstavlja pitanje: "Koliko je vjerojatno da za neku danu oznaku  $t_i$ , neka riječ  $w_i$  ima baš tu oznaku  $t_i$ ?". Ovdje treba pripaziti "smjer" pitanja. Nije dana riječ pa mi gledamo kolika je vjerojatnost za neku oznaku nego za danu oznaku gledamo kolika je vjerojatnost da neka riječ ima baš tu oznaku. Odnosno,  $P(race | NN)$ ,  $P(race | VB) = ?$  Te vjerojatnosti lako dobijemo gledajući u korpus uz pomoć brojanja:

$$P(race | NN) = \frac{\text{broj pojavljivanja (race,NN) u korpusu}}{\text{broj pojavljivanja (NN) u korpusu}}$$

Na jednak način dobijemo i  $P(race | VB)$ . Koristeći Brownov korpus u ovom primjeru, dobit ćemo da su ti konkretni brojevi  $P(race | NN) = 0.00057$  i  $P(race | VB) = 0.00012$ . Kod drugog člana  $P(t_i | t_{i-1} \dots t_{i-2})$ , računanje ide slično. Instancirano, tražimo  $P(VB | TO, VBN)$  i  $P(NN | TO, VBN)$ . Vjerojatnosti dobijemo po istom principu - direktnim gledanjem u korpus i brojanjem frekvencije ponavljanja:

$$P(VB | TO, VBN) = \frac{\text{broj pojavljivanja (VB,TO,VBN) u korpusu}}{\text{broj pojavljivanja (TO,VBN) u korpusu}}$$

Vjerojatnost  $P(NN | TO, VBN)$  doznajemo na identičan način. Na kraju, sve dobivene vjerojatnosti pomnožimo i usporedimo krajnju vjerojatnost. Označivač izbacuje onu kombinaciju oznaka koja ima veću krajnju vjerojatnost.

### Viterbijev algoritam

U prošlom primjeru smo vidjeli kako HMM označivači pronalaze skup oznaka koje najbolje opisuju danu rečenicu. Međutim, čak i kod tog danog primjera smo ograničili broj mogućih kombinacija oznaka na samo dvije. To je, istina, bilo i zbog potrebe lakšeg računanja, no je li generalno "pametno" da HMM označivač mora proći po baš svim mogućim kombinacijama prije nego zaključi koja kombinacija je optimalna? Takav pristup bi značio eksponencijalnu složenost problema.

Valja primijetiti da neki skup oznaka  $t_1, t_2 \dots t_n$  ako je optimalan, nužno sadrži i podskup oznaka  $t_1, t_2 \dots t_i$  koji je također optimalan za prvih  $i$  riječi. Drugim riječima, nije moguće da za prvih  $i$  riječi budu dodijeljene oznake tako da za njih oznake nisu optimalne, a da na kraju za svih  $n$  riječi skup bude optimalan. Ako bi postojao neki drugi optimalni podskup za prvih  $i$  riječi tada bi on nužno bio dio krajnjeg optimalnog skupa. Ovo svojstvo se inače naziva svojstvo optimalne podstrukture i on jedan od dva uvjeta da problem bude rješiv metodom dinamičkog programiranja. To sve ukazuje na to kako je ipak moguće poboljšanje po pitanju složenosti i kako nije nužno proći po svim kombinacijama. Štoviše, optimalno rješenje za  $n - 1$  je i optimalno rješenje za  $n$ .

Algoritam koji koristi metodu dinamičkog programiranja i rješava naš problem naziva se Viterbijev algoritam. Intuicija za Viterbijev algoritam glasi ovako: Za zadanu oznaku pronađi najtočnije rješenje gledajući prethodne oznake. Tako se osiguravamo da uvijek čuvamo lokalni optimum i na kraju on rezultira globalnim optimumom.

Dakle, cilj je pronaći  $\arg \max_T P(t_1 \dots t_n | w_1 \dots w_n)$ . Definiramo sljedeću varijablu [19]

$$\delta_{\tilde{t}}(j) = \arg \max_{t_1 \dots t_j} P(t_1 \dots t_{j-1} \tilde{t} | w_1 \dots w_j)$$

U ovoj varijabli  $\delta_{\tilde{t}}(j)$  zapravo čuvamo prvih  $j - 1$  točnih oznaka za prvih  $j - 1$  riječi, ako  $j$ -ta riječ po redu ima oznaku  $\tilde{t}$ . Treba paziti, ovo nisu prvih  $j - 1$  točnih oznaka, već prvih  $j - 1$  oznaka uz uvjet da je iduća oznaka  $\tilde{t}$  (trebalo bi još vidjeti od svih takvih koji skup je najtočniji).

- **Inicijalizacija:**  $\delta_{\tilde{t}}(1) = P(\tilde{t} | w_1)$ . Vjerojatnost da rečenica počinje s riječju koja ima  $\tilde{t}$  oznaku.
- **Korak:**

$$\delta_{\tilde{t}}(j + 1) = \max_{\tilde{t}_1, \dots, \tilde{t}_N} \delta_{\tilde{t}_i}(j) \cdot P(t_{j+1} = \tilde{t} | t_j = \tilde{t}_i) \cdot P(w_j | t_{j+1} = \tilde{t})$$

$$\psi_{\tilde{t}}(j + 1) = \arg \max_{\tilde{t}_1, \dots, \tilde{t}_N} \psi_{\tilde{t}_i}(j) \cdot P(t_{j+1} = \tilde{t} | t_j = \tilde{t}_i) \cdot P(w_j | t_{j+1} = \tilde{t})$$

- **Terminacijski uvjet i rješenje:** Neka je  $\hat{t}_1 \dots \hat{t}_n$  točno rješenje. Njega dobijemo na sljedeći način:

$$\hat{T}_{n+1} = \arg \max_{\tilde{t}_1, \dots, \tilde{t}_N} \delta_{\tilde{t}_i}(n+1)$$

$$\hat{t}_{n+1} = \psi_{\hat{T}_{n+1}}(n+1).$$

# Poglavlje 4

## Sintaktička analiza

Ljudi su proučavali gramatiku jezika od davnina. Prve gramatike datiraju još od prije 2000 godina. U ovom poglavlju pokazat će se važnost formalne gramatike u računalnoj sintaktičkoj analizi, preciznije kod parsiranja.

Riječ *sintaksa* dolazi od latinske riječi *syntaxis* što znači slaganje ili skladnja (skladno uređivanje). Sintaksa je formalizirani termin onoga što ionako već znamo svi, da jezici imaju svoje pravilnosti i da poredak i način kako slažemo riječi i rečenične jedinice nije proizvoljan nego postoje neka pravila koja smo trebali naučiti. Da to nije tako, jezik se uopće ne bi mogao onda ni učiti. Stroga definicija sintakse, preuzeta sa središnje hrvatske leksikografske ustanove, jest:

”dio gramatike koji proučava ustrojstvo i funkcije gramatičkih konstrukcija, odnosno jedinica većih od riječi. Prema veličini gramatičkih konstrukcija može se razlikovati sintaksa fraze ili sintagmatika (proučava slaganje riječi u skupine riječi ili izraze, odnosno u sintagme) i sintaksa rečenice (bavi se ustrojstvom jednostavnih i složenih rečenica).”[1]

U navedenoj definiciji piše ”proučava ustrojstvo i funkcije gramatičkih konstrukcija većih od riječi”. Kakve su to gramatičke konstrukcije? Dosad smo se većinom bavili riječima. Sad ćemo vidjeti kako se riječi mogu slagati u skupinu riječi - sintagme. Budući da je naglasak trenutno na sintagmama, glavne rečenične jedinice (sastavnice) nam više nisu riječi. Odsada rečenicu proučavamo kao niz sintagmatskih sastavnica. Primjerice, rečenica *Ivan je zaspao* može se raščlaniti u dvije sastavnice, imenski izraz (imensku sintagmu) *Ivan* te glagolski izraz (glagolsku sintagmu) *je zaspao*. Te sastavnice međusobno imaju hijerarhijski odnos. Zbog toga bi se sastavnica *je zaspao* mogla još rastaviti na nove dvije sastavnice *je* i *zaspao* (što će biti poslije itekako bitno za parsiranje).

Koje se sve vrste sintagmi koriste u obradi? To, naravno ovisi o jeziku, ali i o autoru. Za engleski jezik se često navodi 7 kategorija sintagmi najčešćih:

- **imenska sintagma:** Obuhvaća sve od početka imenske fraze pa do njezine glavne imenice.  
Primjer: *big brown bear*
- **glagolska sintagma:** Uključuje pomoćne glagole, modalne glagole i završava s glavnim glagolom/glagolskim pridjevom.  
Primjer: *can be seen*
- **infinitivna sintagma:** Glagoli u infinitivu uključno sa 'to'.  
Primjer: *to drive*
- **sintagma prezenta participa ili gerunda:** Glavni glagol koji završava na 'ing'.  
Primjer: *talking*
- **pridjevska sintagma:** Započinje s prilogom, a završava s glavnim pridjevom.  
Primjer: *completely silent*
- **priložna sintagma:** Priložne sintagme s više riječi.  
Primjer: *really fast*
- **sintagma participa prošlog:** Svi postnominalni i sekundarni glagolski pridjevi koji su morfološki participi prošli.  
Primjer: *bored*

Sasvim slično možemo navesti i za hrvatski jezik:

- **imenska sintagma:** Obuhvaća sve od početka imenske fraze pa do njezine glavne imenice.  
Primjer: *moj maleni dječak*
- **glagolska sintagma:** Uključuje sva jednostavna i složena vremena, njihove negacije i eventualnu povratnu česticu 'se'.  
Primjer: *bih se bio javio*
- **atributna sintagma:** Uključuje dvije imenice od kojih je jedna atribut drugoj te pridjeve koji pobliže opisuju navedene imenice. Svi se slažu u rodu, broju i padežu.  
Primjer: *veliko srce malog čovjeka*
- **apozicijska sintagma:** Uključuje dvije imenice od kojih je jedna apozicija drugoj te njihove pridjeve koji ih pobliže opisuju. Svi se slažu u rodu, broju i padežu.  
Primjer: *naše otkupljenje*
- **prijedložna sintagma:** Uključuje jedan prijedlog i imensku, atributnu ili apozicijsku sintagmu s kojom se slaže u padežu.  
Primjer: *u tom prekrasnom prvom susretu*



## Beskontekstna gramatika

Spomenuto je kako se jezik ponaša po nekakvim svojim pravilnostima te kako bez tih pravila jezik se uopće ne bi mogao učiti. Mogu li se kako ta pravila formalizirati? Naime, ranih 1900-ih rodila se ideja o formaliziranom modelu tih jezičnih pravilnosti. Tu ideju su Chomsky (1956.), a kasnije i Backus (1959.) i Naur (1960.) sproveli u djelo u obliku matematičkog modela kojeg su nazvali beskontekstna (bezokolinska) gramatikama, kontekstno neovisna gramatika ili jednostavno BKG (engl. *context-free grammar* ili *CFG*).

Kontekstno neovisne gramatike se smatraju okosnicama formalnih modela sintakse prirodnih jezika. Zbog toga su i dovoljno moćne da opišu sintaksu programskih jezika. Dapače, sintaksa većine programskih jezika i jest specificirana koristeći kontekstno neovisne gramatike. S druge strane, kontekstno neovisne gramatike su dovoljno jednostavne da dozvole izgradnju učinkovitih algoritama parsiranja koji, za dani niz sastavnica, određuju može li se (i kako) taj niz generirati iz gramatike. Glede prirodnih jezika, ova vrsta formalne gramatike nam omogućava da imamo precizan mehanizam za opisivanje metoda pomoću kojih se gramatičke konstrukcije tj. fraze (ponajviše rečenice) raščlanjuju na manje blokove. Zbog toga ovu gramatiku ponekad opisuju kao gramatika fraznih struktura.

Beskontekstna gramatika sastoji se od skupa pravila ili izvoda, te leksikona. Jedno pravilo iskazuje jedan način kako se određene frazne strukture mogu međusobno grupirati. Pravilo pišemo po sljedećem principu: Pravilo je niz simbola razdvojenih razmakom i sadrži točno jedan poseban simbol "strelicu prema desno" ( $\rightarrow$ ) koja doznachava raščlambu. Simboli se dijele na *terminale* (predstavljaju riječi iz jezika) i *varijable* (*gramatička oznaka definirana u prošlom poglavlju*). S lijeve strane strelice nalazi se točno jedna varijabla, odnosno neterminalni simbol, a s desne strane se nalazi lista varijabli i/ili terminala (lista mora imati barem jedan član, a može imati i više).

Sad će se navesti jedan primjer prethodno opisanih pravila. U tu svrhu koristimo imensku sintagmu NP iz engleskog jezika, za koju smo rekli da obuhvaća sve od početka imenske fraze pa do njezine glavne imenice. Prikaz njezine raščlambe bi izgledao ovako:

$NP \rightarrow Det\ Nominal$	NP se sastoji ili od odrednice (Det) i nominala
$NP \rightarrow ProperNoun$	ili je vlastita imenica (ProperNoun)
$Nominal \rightarrow Noun$	Nominal je ili opća imenica (Noun)
$Nominal \rightarrow Nominal\ Noun$	ili niz općih imenica

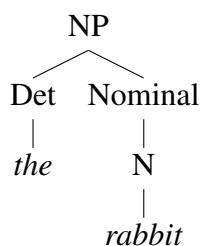
Kada imamo niz od  $n$  pravila ovog tipa  $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_n$ , možemo ih kraće zapisati kao  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ . Slijedi da prethodnu raščlambu NP imenske sintagme možemo zapisati jednostavnije:

$$NP \rightarrow Det \textit{Nominal} \mid \textit{ProperNoun}$$

$$\textit{Nominal} \rightarrow \textit{Noun} \mid \textit{Nominal Noun}$$

Slično vidimo i za ostale kategorije sintagmi, npr.  $VP \rightarrow \textit{Verb NP}$  Primijetimo još u gornjem primjeru pravila hijerarhijsku strukturu - VP sintagma se raščlanjuje na NP sintagmu i "nešto", pa se u nekom smislu NP sintagma nalazi na "nižoj razini" od VP sintagme.

Beskontekstne gramatike mogu djelovati dvojako. Mogu se smatrati kao mehanizam koji generira rečenice ili kao mehanizam koji anotira strukture u danoj rečenici. Kao generator ona instancira varijable na konkretne terminale. Recimo, ako uzmemo sintagmu *the rabbit* za primjer, onda kažemo da se ona može "izvesti" pomoću pravila za NP sintagmu. Naime, znamo da sastavnica *the* ima oznaku *Det* i da sastavnica *rabbit* ima oznaku *Noun*. Također smo upoznati s pravilima koja kažu  $NP \rightarrow Det \textit{Nominal}$  i  $\textit{Nominal} \rightarrow \textit{Noun}$ . Taj skup pravila potreban da bi se to realiziralo inače nazivamo izvod. Izvodi se inače prezentiraju takozvanim *parsnim stablima* odnosno *stablama izvoda*.



Ostaje još za spomenuti *startni simbol*. U formalnoj definiciji gramatike, koju ćemo poslije vidjeti, stoji kako je jezik gramatike skup svih stringova koji se mogu izvesti tom gramatikom iz startnog simbola. S obzirom na to da se najčešće rečenice anotiraju ili izvode, stoji kako taj startni simbol predstavlja rečenicu. Označava se sa slovom *S*. Recimo, pravilo za *S* bi izgledalo ovako:  $S \rightarrow NP VP$ .

Osim pravila, beskontekstna gramatika se sastoji i od leksikona. Slijedi primjer jednog takvog leksikona.

<i>Noun</i>	→	<i>love   cake   morning   flights   happiness   lion   eyes   children   ...</i>
<i>Verb</i>	→	<i>is   like   want   ate   slept   saw   help   love   ...</i>
<i>Adjective</i>	→	<i>first   big   tiny   fat   rich   old   checked   gold   ...</i>
<i>Pronoun</i>	→	<i>I   me   you   yourself   he   she   him   we   ...</i>
<i>Proper-Noun</i>	→	<i>John   Roko   Zagreb   Croatia   Himalaja   Google   Titanic   ...</i>
<i>Determiner</i>	→	<i>a   an   the   this   that   my   his   many   much   ...</i>
<i>Preposition</i>	→	<i>from   to   on   near   with   before   until   ...</i>
<i>Conjunction</i>	→	<i>and   or   but   yet   so   for   ...</i>

Kada uz leksikon pridružimo gore opisani skup pravila neke gramatike, možemo generirati rečenice. Sve rečenice zajedno sačinjavaju nekakav jezik  $\mathcal{L}$ .

$$S \rightarrow NP VP$$

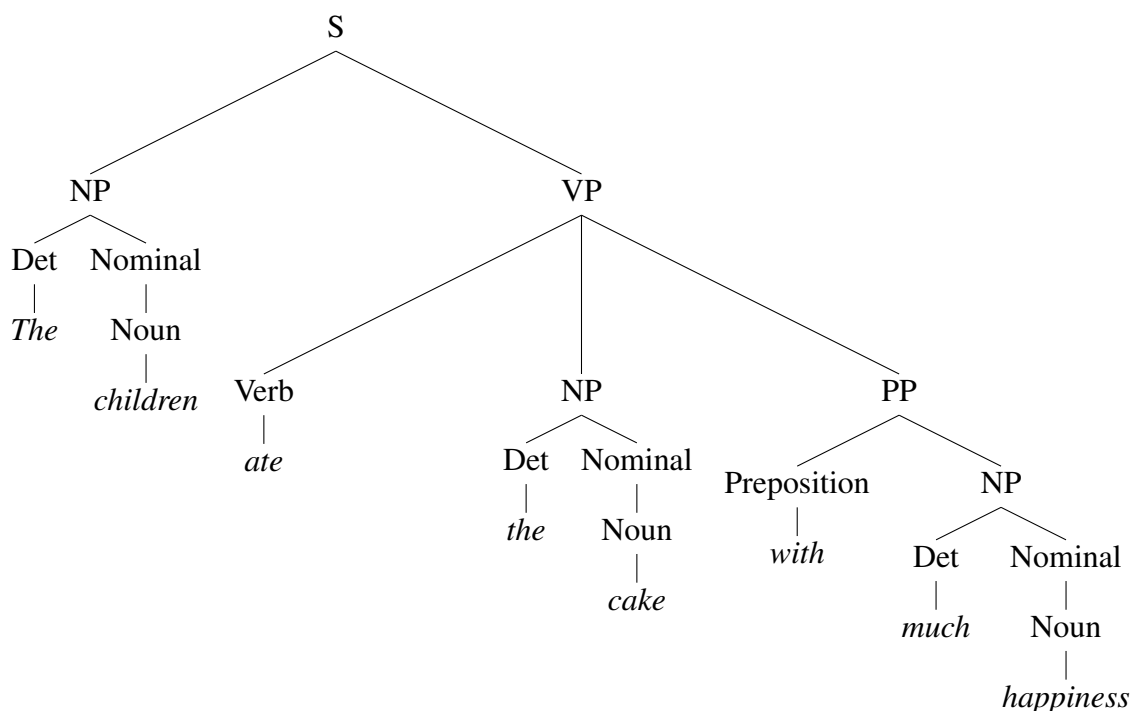
$$NP \rightarrow \begin{array}{l} \textit{Pronoun} \\ | \\ \textit{Proper-Noun} \\ | \\ \textit{Det Nominal} \end{array}$$

$$Nominal \rightarrow \begin{array}{l} \textit{Nominal Noun} \\ | \\ \textit{Noun} \end{array}$$

$$VP \rightarrow \begin{array}{l} \textit{Verb} \\ | \\ \textit{Verb NP} \\ | \\ \textit{Verb NP PP} \\ | \\ \textit{Verb PP} \end{array}$$

$$PP \rightarrow \textit{Preposition NP}$$

Jedna rečenica koja se nalazi u jeziku  $\mathcal{L}$  jest primjerice *The children ate the cake with much happiness*. Izvodi, kao što je već spomenuto, se prikazuju uz pomoć parsnih stabala. Stoga, parsno stablo za ovu rečenicu možemo vidjeti na idućoj slici.



### Formalna definicija gramatike

U prethodnoj sekciji smo opisivali uvođenje matematičkih modela gramatike kako bi formalizirali jezične pravilnosti. Za to su potrebne i formalne definicije, prije svega gramatike, ali i svih ostalih potrebnih pojmova kako bi gramatiku i pojmove vezane uz nju definirali.

**Definicija 4.0.1.** *Abeceda je (obično fiksna) konačan neprazan skup  $\Sigma$  koji se sastoji od znakova koje zovemo slova. Elemente skupa  $\Sigma$  obično označavamo s  $\alpha$ .*

**Definicija 4.0.2.** *Riječ nad  $\Sigma$  jest niz od konačno mnogo (nula ili više) znakova iz  $\Sigma$ .*

**Napomena 4.0.3.** *Treba imati na umu kako slova abecede i riječi iz jezika, zapravo mogu biti bilo kakvi simboli kao npr. brojevi, znamenke, slova, riječi.. Ne moraju nužno biti slova i riječi u našem smislu kakvom smo navikli. Tako u našem slučaju slova su zapravo riječi, a riječi su rečenice odnosno klauzule.*

**Definicija 4.0.4.** *Beskontekstna gramatika  $\mathcal{G}$  je 4-torka  $\mathcal{G} = (V, \Sigma, \longrightarrow, S)$  (naziva se još i beskontekstna gramatika  $\mathcal{G}$  nad abecedom  $\Sigma$ ) gdje je:*

- $V$ : Konačan skup varijabli (neterminala), disjunktan sa  $\Sigma$  tj.  $V \cap \Sigma = \emptyset$ .
- $\Sigma$ : Konačan neprazan skup čije elemente nazivamo znakovi ili terminali i obično ih označavamo s  $\alpha$ .

- $\longrightarrow$ : konačnu binarnu relaciju na  $(V \cup \Sigma)^*$ , čije elemente zovemo pravila (odnosno produkcije). Pravilo je oblika  $A \longrightarrow B$ , gdje je  $A \in V$ , a  $B \in (V \cup \Sigma)^*$ , odnosno string sastavljen od varijabli i terminala.
- $S$ : istaknuti element  $S \in V$  koji nazivamo početna varijabla.

Uz pomoć formalne definicije gramatike, doći ćemo do jezika kojeg te iste gramatike generiraju (kao u gornjem primjeru kod jezika  $\mathcal{L}$ ). Na za to nam je još potrebno par definicija. Prvenstveno se misli tu na izvode riječi.

**Definicija 4.0.5.** Neka je  $\mathcal{G} = (V, \Sigma, \longrightarrow, S)$  beskontekstna gramatika. Za riječi  $\alpha$  i  $\beta \in (V \cup \Sigma)^*$  kažemo da  $\alpha$  "daje"  $\beta$  ( $\alpha$  "izvodi"  $\beta$ ) i označavamo s  $\alpha \Rightarrow \beta$ , ako i samo ako postoje  $\gamma_l$  i  $\gamma_r \in (V \cup \Sigma)^*$  takvi da je  $\alpha = \gamma_l A \gamma_r$  i  $\beta = \gamma_l B \gamma_r$ , gdje je  $A \longrightarrow B$  pravilo iz binarne relacije  $\longrightarrow$ .

**Definicija 4.0.6.** Izvod u  $\mathcal{G}$  je konačan niz riječi  $\alpha_0, \alpha_1, \dots, \alpha_n$  nad  $(V \cup \Sigma)^*$ , takav da je  $\alpha_0 = S$ , te  $\alpha_{i-1} \Rightarrow \alpha_i$  za sve  $i \in c$ . Zovemo ga "izvod za  $\alpha$ ", gdje je  $\alpha$  neka riječ iz  $(V \cup \Sigma)^*$ , ako je  $\alpha_n = \alpha$ . Kraće pišemo  $S \longrightarrow^* \alpha$ .

**Definicija 4.0.7.** Kažemo da  $\mathcal{G}$  izvodi riječ  $\alpha$  nad  $\Sigma$  ako postoji izvod za  $\alpha$  u  $\mathcal{G}$ .

**Definicija 4.0.8.** Kažemo da  $\mathcal{G}$  generira jezik  $L(\mathcal{G}) := \{\alpha \in \Sigma^+ \mid \mathcal{G} \text{ izvodi } \alpha\}$ .

**Definicija 4.0.9.** Stablo je konačna skupina čvorova organiziranih u neku rekurzivnu strukturu. Svako stablo je konačni niz  $(k, s_1, s_2, \dots, s_n)$  za  $n \in \mathbb{N}_0$ , gdje je  $k$  čvor zvan korijen, a  $s_1, s_2, \dots, s_n$  su stabla zvana glavna podstabla. Korijene od  $s_i$  zovemo djecom čvora  $k$ .

**Definicija 4.0.10.** Stablo parsanja iz BKG  $\mathcal{G}$  je stablo u čijem korijenu je početna varijabla, a svaki čvor je ili znak bez djece, ili varijabla čija djeca (može ih biti 0) čitana redom predstavljaju desnu stranu jednog pravila za tu varijablu. Stablo parsiranja za riječ  $\alpha$  je stablo parsiranja čiji znakovi čitani redom su upravo znakovi riječi  $\alpha$ .

**Definicija 4.0.11.** Očito,  $\alpha \in L(\mathcal{G})$  ako i samo ako postoji takvo stablo.

## Parsanje u beskontekstnim gramatikama[32]

Temeljem prošle sekcije zaključujemo kako se parsno stablo i parsanje nalaze u centru beskontekstnih gramatika. Naime, zanima nas ekspresivnost jezika, koliko i koje sve riječi se nalaze u jeziku kojeg generira neka beskontekstna gramatika? A zadnja napisana definicija 4.0.7 kaže kako je dosta ustvrditi postoji li parsno stablo za neku danu riječ. Stoga će se sada posebna pažnja pridati parsanju.

Naziv parsanje preuzet je iz engleskog jezika (*to parse*), a oba posljedično crpe iz latinskoga<sup>1</sup> *pars orationis*, što u doslovnome prijevodu znači vrsta riječi. Također, latinska

<sup>1</sup><https://www.dictionary.com/browse/parse>

riječ (*pars part*) znači dio. Osim termina *parsanje*, može se koristiti i termin *parsiranje*, koji je jednakovrijedan terminu *parsanje*.

Već smo vidjeli kako izgleda parsno stablo za neku danu riječ iz jezika. No, nameće se pitanje, može li biti više parsnih stabala za istu riječ? I ako da, predstavlja li to problem?

Postoji poznata šala američkog komičara Groucha Marxa gdje je jednom prilikom rekao: *"One morning I shot an elephant in my pajamas. How he got into my pajamas I'll never know."* Ova šala upravo je smiješna zbog toga što prirodno ovu rečenicu shvaćamo na jedan intuitivan i logičan način, dok komičar to koristi i pridodaje drukčije, pomalo apsurdno značenje rečenici. No, šala upravo jest u tome što svi znamo da se zaista, iako je apsurdan, rečenica mogla shvatiti i na taj drugi način. Drugim riječima, kao da ista rečenica ima više značenja. To odmah nagoviješta i više parsnih stabala. Potrebno je formalizirati ove pojmove koje su se intuitivno objasnili.

**Definicija 4.0.12.** *Kada za neku riječ  $\alpha \in L(\mathcal{G})$  postoji više parsnih stabala kažemo da je takva riječ, odnosno njezina gramatika - višeznačna.*

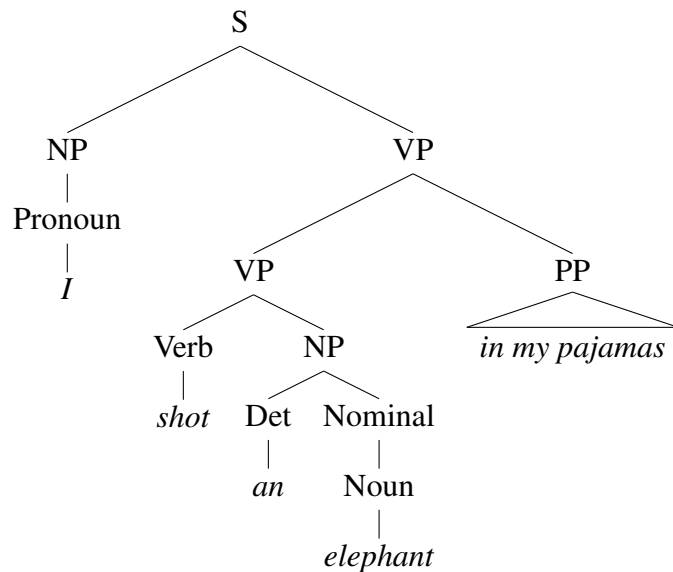
**Definicija 4.0.13.** *Riječ  $\alpha \in L(\mathcal{G})$  je  $\mathcal{G}$ -višeznačna ako postoji više stabala parsiranja iz  $\mathcal{G}$  za  $\alpha$ .*

**Definicija 4.0.14.** *BKG  $\mathcal{G}$  je višeznačna ako postoji  $\mathcal{G}$ -višeznačna riječ iz  $L(\mathcal{G})$ .*

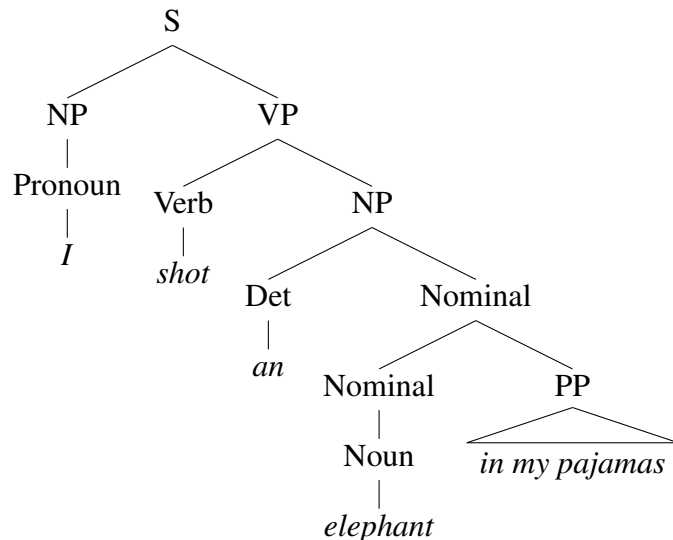
**Definicija 4.0.15.** *Beskontekstni jezik  $L$  je višeznačan ako je svaka BKG za  $L$  višeznačna.*

**Napomena 4.0.16.** *"Jednoznačna" (riječ, gramatika ili pak jezik) znači negaciju pojma "višeznačna".*

Rečenica koju smo gore spomenuli zaista jest višeznačna. Primjerom će se pokazati kako zaista ima dva stabla parsiranja. Prvi način na koju smo mogli shvatiti rečenicu jest onaj intuitivan, a to je da je neki čovjek ujutro, kada se probudio, u svojoj pidžami, upucao slona. Vidimo parsno stablo za taj način.



Drugi način jest ovaj iz šale: Čovjek, jedno jutro kada se probudio, upucao je slona koji je bio obučen u njegovu pidžamu.



Kada računalo parsira neku gramatiku, sigurno ne želimo da vrati spektar nekih rješenja nego jedinstveno rješenje. Prema tome, višeznačnost predstavlja problem koji nazivamo sintaktička dvoznačnost (eng. *syntactic ambiguity*). Može se još nazivati i strukturna dvoznačnost ili dvosmislenost. Razlikujemo više vrsta dvosmislenosti koje se pojavljuju, a najčešće su: dvosmislenost priljepljivanja (eng. *attachment ambiguity*) i koordinacijska dvosmislenost (eng. *coordination ambiguity*).

Sintaktičko uklanjanje dvosmislenosti je proces kojim sustavi za obradu prirodnog jezika odabiru točno jedno parsiranje od nekoliko mogućih.

### Chomskyjeva normalna forma

Ovdje ćemo prikazati rješenje gore spomenutog problema normalizacijom gramatike.

Gramatiku je potrebno "normalizirati". I to na takav način da dvije gramatike ostanu ekvivalentne.

**Definicija 4.0.17.** *Kažemo da su dvije gramatike slabo ekvivalentne ako generiraju isti skup riječi, a jako ekvivalentne ako su slabo ekvivalentne i dodjeljuju istu fraznu strukturu svakoj rečenici.*

U ovom slučaju je potrebna jaka ekvivalencija. Definirat ćemo sada normalnu formu za koju će se poslije pokazati da ima jaku ekvivalenciju sa svakom beskontekstnom gramatikom.

**Definicija 4.0.18.** *Za gramatiku  $\mathcal{G} = (V, \Sigma, \longrightarrow, S)$  kažemo da je u Chomskyjevoj normalnoj formi (kratko: da je Chomskyjeva) ako je svako njeno pravilo jednog od dva oblika:*

- $A \longrightarrow BC$  (širenje), gdje je  $\{A, B, C\} \subseteq V$ , te  $S \notin \{B, C\}$ .
- $A \longrightarrow \alpha$  (čitanje), gdje je  $A \in V$  varijabla, a  $\alpha \in \Sigma$  znak.

Chomskyjeva normalna forma (eng. *Chomsky normal form*, kraće CNF) stavlja ne-kakve dodatne uvjete na gramatiku čuvajući jednaku ekspresivnost i moć gramatike. Ono što se dobilo stavljanjem dodatnih uvjeta jest izbjegavanje dvosmislenosti.

Postoji i dodatan benefit osim rješenja sintaktičke dvosmislenosti. CNF donosi gornju granicu složenosti parsiranja. Za bilo koju riječ  $\alpha$  duljine  $n$ , izvod ide u najviše  $2n - 1$  koraka:  $n - 1$  kako bi odredili duljinu  $n$  i dodatnih  $n$  koraka kako bi varijable zamijenili terminalima. U najgorem slučaju izračunavamo svih mogućih  $2n - 1$  koraka. Iako složenost ostaje eksponencijalna, barem znamo da je problem rješiv. Srećom, postoji algoritam CKY (poslije će se detaljno obraditi) koji ovo izračunava u mnogo manjoj složenosti,  $\mathcal{O}(n^3)$ .

Sada je potrebno pokazati ekvivalentnost tj. da se svaka gramatika može pretvoriti u Chomskyjev normalan oblik.

**Teorem 4.0.19.** *Svaka BKG  $\mathcal{G}$  se može pretvoriti u gramatiku  $\mathcal{G}'$  u Chomskyjevom normalnom obliku koja joj je "skoro" ekvivalentna:*

$$L(\mathcal{G}') = L(\mathcal{G}) \setminus \{\varepsilon\}$$

*Dokaz.* Da bi dokazali ovu tvrdnju koristit ćemo konstruktivni dokaz pokazavši na takav način konstrukcijsku metodu pretvaranje općenite gramatike  $\mathcal{G}$  u gramatiku  $\mathcal{G}'$  u Chomskyjevom normalnom obliku. Dokaz se odvija u 4 faze (koraka):



- **Prva faza:** Ako se početna varijabla  $S$  pojavljuje na desnoj strani nekog pravila, dodamo novu varijablu  $S'$ , dodamo novo pravilo  $S \rightarrow S'$  te proglasimo  $S'$  novom početnom varijablom.
- **Druga faza:** Svakom znaku  $\alpha \in \Sigma$  koji se pojavljuje na desnoj strani nekog pravila ali **ne sam** (dakle ne u pravilu čitanja), dodijelimo novu varijablu  $N_\alpha$ , te u gramatiku dodamo pravilo  $N_\alpha \rightarrow \alpha$ . Zatim sve takve pojave od  $\alpha$  zamijenimo s  $N_\alpha$ .
- **Treća faza:** Za svako pravilo čija desna strana ima  $n \geq 3$  simbola (nakon 2. faze oni moraju biti varijable), oblika  $A \rightarrow V_1 V_2 \dots V_n$ , dodamo nove varijable  $A_1, \dots, A_n$ , i to pravilo zamijenimo s  $n - 1$  "ulančanih" pravila širenja.

$$A \rightarrow V_1 A_1, A_1 \rightarrow V_2 A_2, A_2 \rightarrow V_3 A_3, \dots, A_{n-2} \rightarrow V_{n-1} V_n.$$

- **Četvrta faza:** Cilj ove faze jest ukloniti sva  $\varepsilon$ -pravila (sve  $\varepsilon$ -produkcije).  $\varepsilon$ -pravilo nazivamo ono pravilo oblika

$$A \rightarrow \varepsilon; \text{ gdje } A \text{ nije } S', \text{ odnosno nije startni simbol.}$$

Kako bi eliminirali sva pravila ovog oblika, treba utvrditi koje sve varijable generiraju  $\varepsilon$ . Ako postoji pravilo oblika:

$$A \rightarrow \varepsilon, \text{ tada kažemo da } A \text{ generira } \varepsilon.$$

$$A \rightarrow V_1 V_2 \dots V_n, \text{ gdje svaki } V_i \text{ generira } \varepsilon, \text{ tada kažemo da i svaki } A \text{ generira } \varepsilon.$$

Tada u svakom pravilu oblika  $A \rightarrow V_1 V_2 \dots V_n$  "obrišemo" one  $V_i$  koje generiraju  $\varepsilon$ . Nakon toga, na koncu, obrišemo i sva  $\varepsilon$ -pravila.

□

**Napomena 4.0.20.** *Gramatika u Chomskyjevu normalnom obliku produkcijama može opisivati samo binarna stabla (ona kod kojih se u svakom grananju stvaraju nužno dvije nove grane, osim ako se radi o listu stabla).*

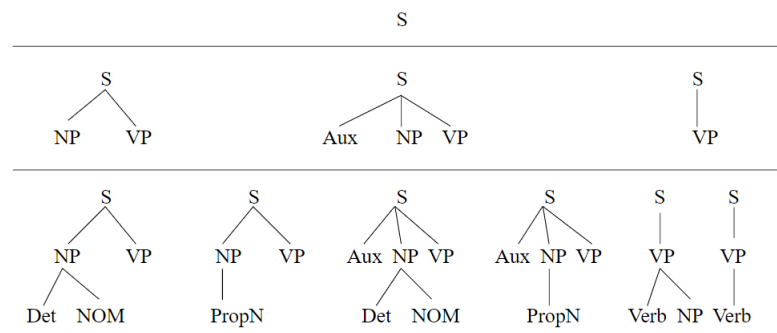
Formalizirali smo pojam gramatike i uveli neke pojmove uz to, kao naprimjer stablo parsiranja. Ipak, modeliranje gramatike i njezin formalni zapis ostaje na deklarativnom formalizmu trenutno. I time u nijednom trenutku nije navedeno kako to neko konkretno stablo parsiranja pronaći i odrediti. Još nijednom se nije spomenulo kako zapravo - parsirati gramatiku. Zato sada slijedi opis par različitih pristupa parsiranju i na kraju implementacija algoritma koji parsira gramatiku.

**Različiti pristupi parsiranja beskontekstne gramatike**

Postoje različiti pristupi kako parsirati beskontekstnu gramatiku i njezina pravila. Načelno se svrstavaju u tri kategorije, no u ovom radu će se opisati samo dvije:

1. Prema načinu sučeljavanja ulaznoga niza riječi s produkcijama zadane formalne gramatike razlikuje se:
  - Silazno parsiranje odnosno parsiranje od vrha prema dnu parsnoga stabla (eng. *top-down parsing*).
  - Uzlazno parsiranje odnosno parsiranje od dna prema vrhu parsnoga stabla (eng. *bottom-up parsing*).
  
2. Prema načinu čitanja ulazne rečenice:
  - Usmjereno parsiranje odnosno čitanje niza riječi s lijeva na desno (eng. *directional parsing*).
  - Neusmjereno parsiranje odnosno čitanje niza riječi nekim drugim slijedom koji pogoduje izvršavanju algoritma (eng. *non-directional parsing*).

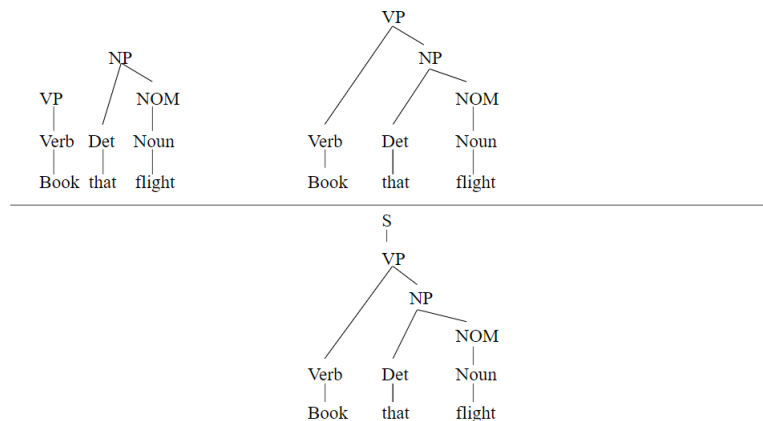
Kod silaznoga parsiranja (eng. *top-down parsing*) nastoji se oponašati postupak izvođenja formalnom gramatikom od početnoga nezavršnog simbola prema listovima parsnoga stabla, pa se stoga – budući da ide od vrha parsnoga stabla prema njegovu dnu – takva analiza i naziva silaznom.



*Input:* Book that flight

Slika 4.1: Top-down parsing.

Obrnuto, uzlazno parsanje (eng. *bottom-up parsing*) nastoji rekonstruirati postupak izvođenja od posljednje produkcije unatrag, odnosno od listova parsnoga stabla pa do početnoga nezavršnog simbola.



Slika 4.2: Bottom-up parsing.

Kod neusmjerenoga parsanja (eng. *directional parsing*) način pristupanja ulaznomu nizu podređen je cilju izgradnje parsnoga stabla. Čitava ulazna rečenica stoga mora biti u memoriji parsera kako bi on njezinim riječima mogao pristupati neograničenim redosljedom, odnosno neusmjerenom. Postoji i silazna i uzlazna inačica neusmjerenoga parsanja. Dok usmjereni parseri (eng. *non-directional parsing*) čitaju ulazni niz riječi redom, s lijeva na desno, pa stoga za njihov rad nije potrebno čitavu rečenicu pohraniti u memoriju. Sve poznate metode usmjerenoga parsanja modeliraju se i izvode u obliku parsnih automata. S obzirom na formalnu gramatiku, također postoji uzlazna i silazna inačica[32].

Algoritam koji se najčešće rabi kod izvedbe parsiranja beskontekstnih gramatika naziva se *CKY parser*. Taj parser spada u skupinu uzlazno neusmjerenih parsera. U idućoj sekciji slijedi opis i implementacija tog algoritma.

### CKY parsiranje

Parser CKY jest algoritam za parsanje koji kao ulaz uzima beskontekstnu gramatiku  $\mathcal{G}$  i jedan niz riječi  $\alpha_1, \dots, \alpha_n$ , odnosno jednu rečenicu-kandidat za koju treba odrediti stablo parsiranja (ili stabla parsiranja) prema zadanoj gramatici. Za potrebe parsera CKY ulazna gramatika mora biti u Chomskyjevu normalnom obliku.

CKY algoritam koristi princip dinamičkog programiranja pristupajući ulaznomu nizu riječi, a izvršavajući uzlazno parsiranje. Rad algoritma najčešće se prikazuje s pomoću dvo-dimenzionalne kvadratne matrice, nazovimo je  $M$  (donjetrokutaste ili gornjetrokutaste),

čije su dimenzije određene duljinom ulaznoga niza, odnosno brojem riječi u rečenici. Tako ćemo za ulaznu rečenicu od  $n$  riječi, imat ćemo gornjetrokutastu (donjetrokutastu) matricu  $M$  reda  $(n + 1)$ .

Svaki element matrice  $M[i, j]$  sadrži skup varijabli koje predstavljaju sve frazne strukture koje sačinjavaju riječi počevši od  $i$ -te riječi pa sve do  $j$ -te. Dakle, element matrice  $M[0, n]$  predstavlja cijelu rečenicu. Ako nakon izvršenja algoritma sadrži  $S$ , tada postoji stablo parsiranja (barem jedno). Gdje nam je potrebno da gramatika treba biti Chomskyjeva? Zbog napomene 4.0.20 vidimo da svaka varijabla ima dva djeteta u stablu parsiranja, pa onda znamo da za element na poziciji  $(i, j)$  mora postojati  $k$ ,  $i \leq k \leq j$  koji razdjeljuje tu fraznu strukturu na dva dijela:  $[i, k]$  i  $[k, j]$ . Kako taj  $k$  mora postojati, a algoritam radi dinamički, onda znamo kako će svaki element matrice  $M[i, j]$  biti popunjen (ukoliko pravila zaista generiraju neku fraznu strukturu) i osigurano je kako ćemo u datom trenutku imati sve potrebne informacije.

Kao konkretan primjer rada ovog algoritma, uzmimo beskontekstnu gramatiku  $\mathcal{G}$  s početnom varijablom  $S$  definiranu kao:

- $V = \{S, A\}$ , disjunktan sa  $\Sigma$  tj.  $V \cap \Sigma = \emptyset$ .
- $\Sigma = \{a, b\}$ .
- konačnu binarnu relaciju na  $(V \cup \Sigma)^*$ , čije elemente zovemo pravila (odnosno produkcije).
  1.  $S \rightarrow S S$
  2.  $S \rightarrow A A$
  3.  $S \rightarrow b$
  4.  $A \rightarrow A S$
  5.  $A \rightarrow A A$
  6.  $A \rightarrow a$

Sada možemo pokazati kako bi izgledala matrica  $M$  koju bi vratio CKY parser parsirajući gramatiku  $\mathcal{G}$ .

		a	a	b	b
a	A	S, A	S, A	S, A	
a		A	A	A	
b			S	S	
b				S	

Algoritam 2 opisuje CKY prepoznavanje (a ne parsiranje!). Matrica ima indekse od 0 do  $n$ , a lista riječi od 1 do  $n$ .

---

**Algoritam 2** CKY PARSER
 

---

**Ulazni podaci:** niz riječi *rijeci* i gramatika  $\mathcal{G}$

**Izlazni podaci:** matrica  $M$

Stvori matricu  $M$  reda  $(n + 1)$

**for**  $j \leftarrow 1$  **to**  $\text{length}(\text{rijeci})$  **do**

**for all**  $A \mid A \leftarrow \text{rijeci}[j] \in \mathcal{G}$  **do**

$M[j - 1, j] \leftarrow M[j - 1, j] \cup A$

**end for**

**for**  $i \leftarrow j - 2, 0$  **do**

**for**  $k \leftarrow i + 1$  **to**  $j - 1$  **do**

**for all**  $A \mid A \leftarrow BC \in \mathcal{G}$  and  $B \in M[i, k]$  and  $C \in M[k, j]$  **do**

$M[i, j] \leftarrow M[i, j] \cup A$

**end for**

**end for**

**end for**

**end for**

**return**  $M$

---

Spomenuto je već kako algoritam treba jednostavno pronaći varijablu  $S$  unutar elementa  $M[0, n]$  kako bi mogli reći da postoji stablo parsiranja. To nam pak ne daje parsno stablo, već samo vraća kao rezultat postoji li parsno stablo ili ne. Parsno stablo rekonstruira se iz rada algoritma CYK, odnosno iz traga ostavljenoga uparivanjima lijevih i desnih strana produkcija – svako uparivanje stvara jednu roditeljsku granu stabla i dvije njoj podređene grane na razini ispod.

### Statističko parsiranje

Jedan od glavnih razloga zbog kojeg se uvodi potreba za ovakvim vidom parsiranja jest sintaktičko uklanjanje dvosmislenosti. Zašto statističko parsiranje? Očito možemo izračunati vjerojatnost svakog od stabala parsiranja neke riječi i "vratiti" onog koji ima najveću vjerojatnost.

**Definicija 4.0.21.** *Vjerojatnosna beskontekstna gramatika  $\mathcal{G}$  je 4-torka  $\mathcal{G} = (V, \Sigma, \longrightarrow, S)$  (naziva se još i vjerojatnosna beskontekstna gramatika  $\mathcal{G}$  nad abecedom  $\Sigma$ ) gdje je:*

- $V$ : Konačan skup varijabli (neterminala), disjunktan sa  $\Sigma$  tj.  $V \cap \Sigma = \emptyset$ .

- $\Sigma$ : Konačan neprazan skup čije elemente nazivamo znakovi ili terminali i obično ih označavamo s  $\alpha$ .
- $\longrightarrow$ : konačnu binarnu relaciju na  $(V \cup \Sigma)^*$ , čije elemente zovemo pravila (odnosno produkcije). Pravilo je oblika  $A \longrightarrow \alpha [P]$ , gdje je  $A \in V$ , a  $\alpha \in (V \cup \Sigma)^*$ , odnosno string sastavljen od varijabli i terminala, te a  $P$  vjerojatnost  $P(\alpha | A)$  gdje je  $P \in [0, 1]$
- $S$ : istaknuti element  $S \in V$  koji nazivamo početna varijabla.

Dakle, vjerojatnosna BKG se razlikuje od "obične" što svako pravilo dolazi s kondicionalnom vjerojatnošću  $P$  gdje je  $P: R \longrightarrow [0, 1]$ . Odnosno,  $P$  predstavlja vjerojatnost da će se varijabla  $A$  proširiti u  $\alpha$ . Vjerojatnost prema tome možemo zapisivati kao  $P(A \longrightarrow \alpha)$ . Preciznije,  $P(A \longrightarrow \alpha | A)$ . Prema tome, vjerojatnost shvaćamo kao vjerojatnost da će varijabla  $A$  proširiti u  $\alpha$  uz uvjet da već imamo  $A$ . Ili, ako baš želimo, možemo zapisati i ovako:  $P(RHS | LHS)$ .

Ako gledamo sva moguća proširenja varijable  $A \in V$ , tada mora vrijediti da je suma svih pravila jednaka 1. Formalno,

$$\sum_{\alpha} P(A \longrightarrow \alpha) = 1. \quad (4.1)$$

Sada ćemo vidjeti primjer jedne takve gramatike. Znamo kako se gramatika sastoji od leksikona i produkcija, stoga slijede primjeri.

*Noun*  $\longrightarrow$  *love* [0.30] | *cake* [0.10] | *dinner* [0.10] | *flight* [0.40] | *children* [0.10]  
*Verb*  $\longrightarrow$  *is* [0.20] | *like* [0.15] | *want* [0.15] | *ate* [0.15] | *be* [0.05] | *book* [0.30]  
*Pronoun*  $\longrightarrow$  *I* [0.40] | *me* [0.15] | *you* [0.40] | *yourself* [0.05]  
*Proper-Noun*  $\longrightarrow$  *Roko* [0.40] | *Croatia* [0.60]  
*Determiner*  $\longrightarrow$  *a* [0.20] | *an* [0.10] | *the* [0.60] | *that* [0.10]  
*Preposition*  $\longrightarrow$  *from* [0.30] | *to* [0.30] | *on* [0.25] | *near* [0.15]

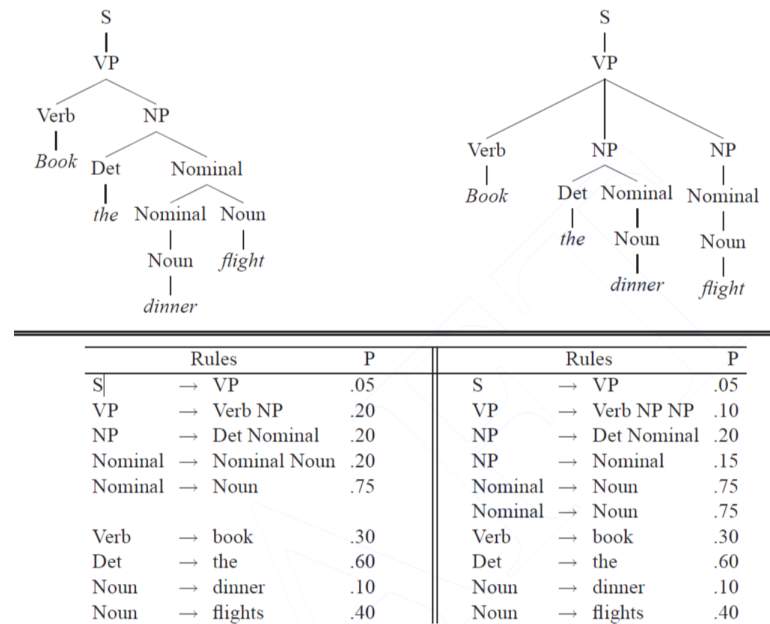
$S$	$\longrightarrow$	$NP VP$	[0.95]
$S$	$\longrightarrow$	$VP$	[0.05]
$NP$	$\longrightarrow$	$Pronoun$	[0.35]
		$Proper-Noun$	[0.3]
		$Nominal$	[0.15]
		$Det Nominal$	[0.2]
$Nominal$	$\longrightarrow$	$Nominal PP$	[0.05]
		$Noun$	[0.75]
		$Nominal Noun$	[0.2]
$VP$	$\longrightarrow$	$Verb$	[0.4]
		$Verb NP$	[0.2]
		$Verb NP PP$	[0.15]
		$Verb NP NP$	[0.1]
		$Verb PP$	[0.25]
$PP$	$\longrightarrow$	$Preposition NP$	[1.0]

Vjerojatnosna CFG dodjeljuje vjerojatnost svakom stablu parsiranja  $T$  neke rečenice ili rečenične klauzule  $S$ . Uzmimo naprimjer rečenicu: "Book the dinner flights." Intuitivno značenje rečenice jest da se rezerviraju karte za let koji servira večeru. Neintuitivan značenje bi bilo da se rezerviraju karte za let u ime večere. Očito je da ta dva značenja nisu jednako vjerojatna. Ostaje pokazati konkretnom računicom.

Ipak, potrebno je prije toga opisati kako se odvija ta računica. Kako izračunati vjerojatnost nekog parsnog stabla? Dakle, potrebno je izračunati vjerojatnost:

$$P(T, S) = \prod_{i=1}^n P(RHS_i | LHS_i). \quad (4.2)$$

Rezultat gore definirane vjerojatnosti  $P(T, S)$  jest uparena vjerojatnost nekog parsiranja i rečenice. No, ona je zapravo jednaka samoj vjerojatnosti parsiranog stabla. Slijedi kratki dokaz.



Slika 4.3: Dva parsna stabla iz gornjeg primjera s njihovim pripadajućim vjerojatnostima

$$P(T, S) = P(T)P(S | T). \tag{4.3}$$

Ali kako stablo parsiranja sadrži sve riječi rečenice  $S$ , imamo da je  $P(S | T) = 1$ . Dakle,

$$P(T, S) = P(T)P(S | T) = P(T). \tag{4.4}$$

Kao posljedicu, imamo da se vjerojatnost svakog parsnog stabla može računati kao produkt vjerojatnosti svih korištenih pravila u tom parsnom stablu. Konkretno,

$$P(T_{lijevi}) = 0.05 \times 0.20 \times 0.20 \times 0.20 \times 0.75 \times 0.30 \times 0.60 \times 0.10 \times 0.40 = 2.2 \times 10^{-6}.$$

$$P(T_{desni}) = 0.05 \times 0.10 \times 0.20 \times 0.15 \times 0.75 \times 0.75 \times 0.30 \times 0.60 \times 0.10 \times 0.40 = 6.1 \times 10^{-7}$$

Očekivano, od dva parsna stabla (značenja) rečenice "Book the dinner flights", veću vjerojatnost ima ono intuitivnije značenje i njegovo pripadajuće parsno stablo. Prema tome, algoritam za dvosmislenost vjerojatnosne beskontekstne gramatike bi vratio  $P(T_{lijevi})$  kao rješenje.



Međutim, ovo sve iz prethodnog primjera potrebno je formalizirati. Treba formalizirati intuiciju kako je biranje stabla s najvećom vjerojatnošću, ispravan način rješenja sintaktičke dvosmislenosti.

Neka imamo sva parsna stabla neke rečenice  $S$ . Tu rečenicu  $S$  zovemo "plod" (eng. *yield*) parsnog stabla od  $S$ . Dakle, od svih parsnih stabala koji imaju rečenicu  $S$  kao plod, algoritam za uklanjanje sintaktičke dvosmislenosti odabire ono stablo s najvećom vjerojatnošću.

$$\tilde{\mathcal{T}}(S) = \arg \max_{T:S = \text{yield}(T)} P(T | S) \quad (4.5)$$

Prema definiciji,  $P(T | S)$  možemo zapisati po definiciji pa dobijemo:

$$\tilde{\mathcal{T}}(S) = \arg \max_{T:S = \text{yield}(T)} \frac{P(T, S)}{P(S)} \quad (4.6)$$

S obzirom na to da maksimiramo po svim parsnim stablima iste rečenice,  $P(S)$  će biti jednak za sva parsna stabla, pa ga možemo zanemariti:

$$\tilde{\mathcal{T}}(S) = \arg \max_{T:S = \text{yield}(T)} P(T, S) \quad (4.7)$$

Nadalje, kako smo već pokazali da je  $P(T, S) = P(T)$ , konačno imamo:

$$\tilde{\mathcal{T}}(S) = \arg \max_{T:S = \text{yield}(T)} P(T) \quad (4.8)$$

### Vjerojatnosni CKY algoritam

Prema zadnjoj napisanoj formuli 4.8, tražimo stablo s najvećom vjerojatnošću.

Algoritam koji ćemo koristiti će biti modifikacija standardnog algoritma za parsiranje stabla, CKY algoritma. Kao i kod tog algoritma, pretpostavljamo da je gramatika Chomskyjeva, inače algoritam ne bi mogao raditi.

Prisjetimo se kratko prvo "običnog" CKY algoritma. Svaka produkcija koju je moguće iskoristiti, zapisuje se u dvodimenzionalnu matricu  $M$ . Za danu rečenicu duljinu  $n$ , matrica jest reda  $(n + 1)$ . Svaki element matrice  $M[i, j]$  sadrži skup svih mogućih produkcija koje obuhvaćaju sve frazne strukture koje su sačinjene od riječi, počevši od  $i$ -te riječi pa sve do  $j$ -te.

U vjerojatnosnoj inačici malo je drukčije. Svaka od tih mogućih sastavnih fraznih struktura zapravo konstituira treću dimenziju matrice, veličine  $\text{card}(V)$ . Ova treća dimenzija zapravo predstavlja neterminale koji bi bili inače bili zapisani u element matrice  $M[i, j]$ . Vrijednost matrice  $M[i, j, A]$  sada više nije skup varijabli nego vjerojatnost da će

baš varijabla  $A$  biti izvedena nekom produkcijom koja na desnoj strani ima fraznu strukturu sačinjenu od riječi između  $i$ -te pozicije i  $j$ -te pozicije.

Implementacija preinačenog algoritma sada izgleda ovako:

---

**Algoritam 3** VJEROJATNOSNI CKY PARSER
 

---

**Ulazni podaci:** niz riječi  $ri\ jeci$  i gramatika  $\mathcal{G}$

**Izlazni podaci:** stablo parsiranja  $\tilde{T}$  s najvećom vjerojatnošću i pripadna vjerojatnost

Stvori matricu  $M$  reda  $(n + 1) \times (n + 1) \times \mathbf{card}(V)$

**for**  $j \leftarrow 1$  **to**  $length(ri\ jeci)$  **do**

**for all**  $A \mid A \leftarrow ri\ jeci[j] \in \mathcal{G}$  **do**

$M[j - 1, j, A] \leftarrow P(A \leftarrow ri\ jeci[j])$

**end for**

**for**  $i \leftarrow j - 2, 0$  **do**

**for**  $k \leftarrow i + 1$  **to**  $j - 1$  **do**

**for all**  $A \mid A \leftarrow BC \in \mathcal{G}$  and  $M[i, k, B] \geq 0$  and  $M[k, j, C] \geq 0$  **do**

**if**  $M[j - 1, j, A] \leq P(A \leftarrow BC) \times M[i, k, B] \times M[k, j, C]$  **then**

$M[j - 1, j, A] \leftarrow P(A \leftarrow BC) \times M[i, k, B] \times M[k, j, C]$

$back[i, j, A] \leftarrow \{k, B, C\}$

**end if**

**end for**

**end for**

**end for**

**end for**

**return**

---

## Poglavlje 5

# Chatbot programi

U prethodnim poglavljima opisana je detaljno obrada prirodnog jezika na računalima. Postoje razne praktične primjene takve obrade. Poput: gramatičkih provjernika, pretrage podataka, raznih vrsta tražilica, marketinga, glasovnih pomoćnika (engl. *voice assistant*), filtriranja e-pošte, sentimentalne analize ... Iako su neke od ovih primjena već spomenute u uvodu, glavni fokus ovoga poglavlja će biti neka druga primjena - tzv. *chatbot* programi. Kako izgledaju općenito chatbot programi?

Sigurno je gotovo svatko danas čuo za Siri, IBM Watson, Google Allo, itd. Glavna ideja takvih programa jest da pomažu korisnicima da budu produktivniji postajući posrednicima. To čine tako da omogućuju korisnicima da se manje brinu oko dohvaćanja informacija te načina kako dohvatiti određeni podatak. Spadaju pod područje umjetne inteligencije jer s vremenom mogu postajati sve inteligentniji kako se koristeći susreću s mnogo raznovrsnijim skupovima podataka na kojima se mogu trenirati. Primjerice, sigurno je jednom korisniku naporno imati mnogo korisničkih imena, lozinki na raznim web aplikacijama različite namjene. Zamislimo kada bi postojao nekakav program koji može naručivati hranu, obavljati kupovinu, kupovati ulaznice, rezervirati karte i smještaj, a korisnik ne bi trebao nikakvo korisničko ime i lozinku za to.

Postoje servisi koje sve ove zadatke obavljaju, a da nisu chatbot programi (web stranica, aplikacija), stoga se ne smije na chatbot programe gledati kao isključivo na sučelja pogodna za obavljanje zadataka. Razlog zbog kojeg chatbot program ima prednost naspram konvencionalnih metoda koje rješavaju sve nabrojane probleme (i mnoge druge) jest njegova odlike višenamjensva i kao takav može služiti kao osobni asistent. To kao posljedicu ima uštedu vremena i novca.

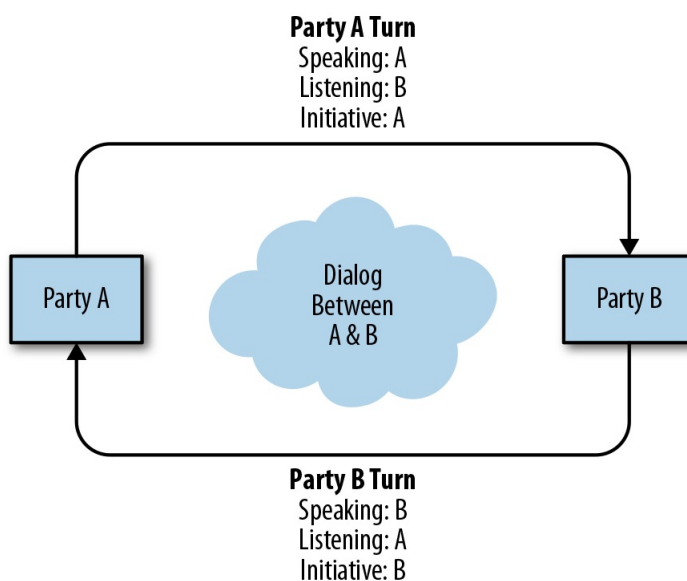
Zbog čega chatbot program zahtijeva obradu prirodnog jezika? Chatbot treba biti dovoljno inteligentan da "shvati" korisnički zahtjev i što korisnik treba, a taj upit najčešće dolazi izražavajući se prirodnim jezikom.

## Što su chatbot programi

Claude Shannon i Warren Weaver, pioniri informacijske teorije i strojnog prevođenja, 1940-ih su razvili komunikacijski model[26] toliko utjecajan da se njime čak i danas služimo za razumijevanje razgovora. U njihovom modelu, komunikacija se svodi na niz šifriranja i transformacija dok poruka putuje kroz komunikacijski kanal od početnog izvora do željene destinacije. Interakcija se odvija u unaprijed fiksiranom vremenskom intervalu. Tijekom komunikacije, svaki od sudionika može ili slušati ili govoriti. Efektivna komunikacija se smatra ona u kojoj u svakom trenutku točno jedan sudionik govori, a svi ostali slušaju.

Možda se nama iz naše ljudske perspektive čini nepotrebna uopće potreba za uvođenjem nekakvog modela komunikacije jer nam se nameće samo od sebe, intuitivno i prirodno. No, takav pristup ima ogromne računarske implikacije i koristi. Također, zar nije i u uglađenijim ljudskim dijalozima potreban nekakav model? U svakoj saborskoj sjednici ili debati bilo kakvog tipa postoji točno određeno vrijeme koliko se smije govoriti i kada. I uvijek je jedan govornik dok ostali slušaju, kao u našem modelu[10].

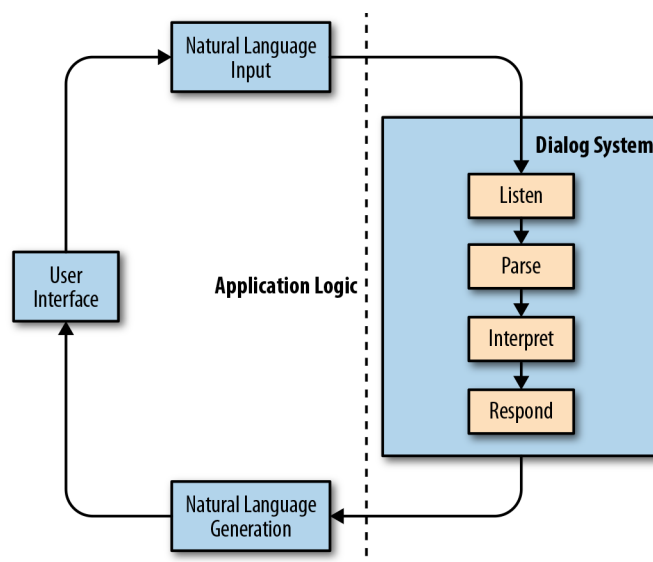
Kako bi se postigla istinska komunikacija, potrebno je da se govornici međusobno izmjenjuju.



Slika 5.1: Struktura komunikacije.

Na temelju prethodnog, chatbot program se može definirati kao program koji sudjeluje u komunikaciju baziranoj na gore opisanom modelu, čiji je cilj interpretirati tekst ili govor i generirati prikladan odgovor. Poradi toga se chatbotovi oslanjaju na heuristike i tehnike

strojnog učenja kako bi postigli željene rezultate. Na slici 5.2 se vidi arhitektura jednog chatbot programa.



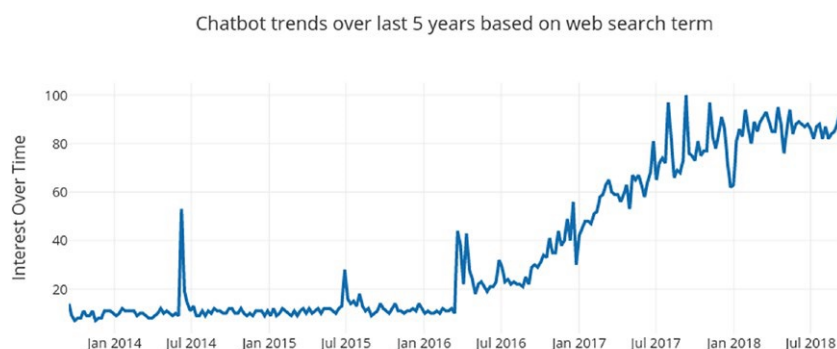
Slika 5.2: Arhitektura chatbot programa.

Arhitektura se sastoji od dvije glavne komponente. Prva je sučelje preko kojeg korisnik šalje svoje upite (npr. mikrofoni kod govora ili aplikacijsko programsko sučelje) i preko kojeg se šalju odgovori (zvučnik ili mobilno sučelje). Druga komponenta jest interni sustav za dijalog koji interpretira primljeni tekst te generira odgovore.

### Popularnost chatbot programa

Iako chatbotovi nisu nova stvar nastala prije par godina, nego postoje već relativno dulje vrijeme, doživjeli su u zadnje vrijeme ogroman eksponencijalni rast u popularnosti, što se vidi na slici 5.3.

Jedan od glavnih razloga te činjenice su njihova jednostavnost. Chatbotovi nisu kompleksni softveri, štoviše to su softveri koje najčešće može koristiti bilo tko. U izgradnji chatbot programa ulazi se sa sviješću kako će taj softver koristiti ljudi svih dobnih skupina, preferencija i informatičke pismenosti. Suprotno, kada se koristi bilo koji drugi softver, najčešće postoji barem nekakva faza prilagodbe, te se postupno treba navikavati kako koristiti taj softver optimalno. Kod chatbotova nije tako. Ako neka osoba zna pričati s drugom osobom tada će vrlo vjerojatno znati komunicirati i s chatbot programom[25]. Prema tome, za očekivati je konstantni trend rasta potražnje za takvim programima.



Slika 5.3: Rast u popularnosti chatbot programa.

Osim olakšavanja svakodnevice korisnicima, chatbotovi su jako korisni u poslovnom svijetu. Više je razloga:

- **Pristupačnost:** Pristupačniji su nego ljudi i zbog toga npr. potrošač ne mora izbirati broj i čekati korisničku podršku, već u par klikova može dobiti željene informacije.
- **Efikasnost:** Korisnik može sjediti kući ili biti u uredu i svejedno uspjeti npr. provjeriti status svoje naručene hrane ili aplicirati se za kredit.
- **Dostupnost:** Dostupni su 24 sata dnevno, 7 dana tjedno. Programi ne poznaju umor kao ljudi, ne pitaju godišnji odmor. Isti zadatak uvijek obavljaju s jednakom efikasnošću, premda to stalno bio slučaj.
- **Skalabilnost:** Mogu rukovoditi puno korisnika odjednom. Upiti ako i stižu od strane tisuću različitih korisnika, svi dobijaju uslugu odjednom. Nema redova za čekanje.
- **Smanjenje troškova:** Rad postojećeg programa iziskuje minimalno troškova, dok ljudska snaga iziskuje mnogo resursa.
- **Moć opažanja:** Odjel za korisnike možda ne može zapamtiti ponašanja svojih korisnika i pamti sve detalje. Chatbot to može, i uz pomoć strojnog učenja može vršiti čak i analizu.

Stoga, kad se vide svi argumenti pobrojani ne treba čuditi kako chatbotovi u različitim poslovnim granama donose ogroman profit. Tako je primjerice, tvrtka za online prodaju kozmetičkih i modnih dodataka *Asos* zabilježila porast od gotovo 300% uvođenjem Facebook Messenger chatbota.

## Studijski primjer

### Opis problema

Kao kod gore spomenutog primjera *Asosa*, ideja studijskog primjera je bila napraviti interaktivni chatbot program koja bi pomagao pri online kupovini u hrvatskim prodavaonica, u ovom konkretnom primjeru prodavaonici sportske obuće i odjeće - *Sport Vision*. Taj interaktivni chatbot bi komunicirao s korisnikom i bio mu asistent pri kupnji. Poseban fokus bi bio na tzv. *recommender* sustavu koji bi kupcu, odnosno korisniku predlagao slične proizvode na temelju nekog odabranog. Iako potpuna integracija takvog sustava (ona nakon koje bi se program mogao izravno koristiti) nadmašuje opseg ovog rada, svejedno će se studijskim primjerom pokazati kako bi izgledali ključni dijelovi takvog programa i kako izgleda implementacija tih dijelova. Naglasak će biti na treniranju neuronske mreže za takav *recommender* sustav.

### Skup podataka

Skup podataka za treniranje neuronske mreže su prikupljeni direktno s web stranice *Sport Visiona*. Podaci su zapravo slike sportskih proizvoda (sportska obuća i odjeća) i dohvaćene su metodom *web scrapinga*. Program za *web scraping* pisan je u programskom jeziku *Python*, te su još kao dodatna pomoć poslužili već postojeći besplatni *web scraping* alati s ugrađenim grafičkim sučeljem u svrhu jednostavnijeg korištenja. Od takvih alata, ponajviše se misli na *ParseHub*[4]. Preuzeto je preko 10000 slika koji su služili kao podaci na kojima se testirala mreža. Na slici vidimo kako izgleda primjer jedne slike:

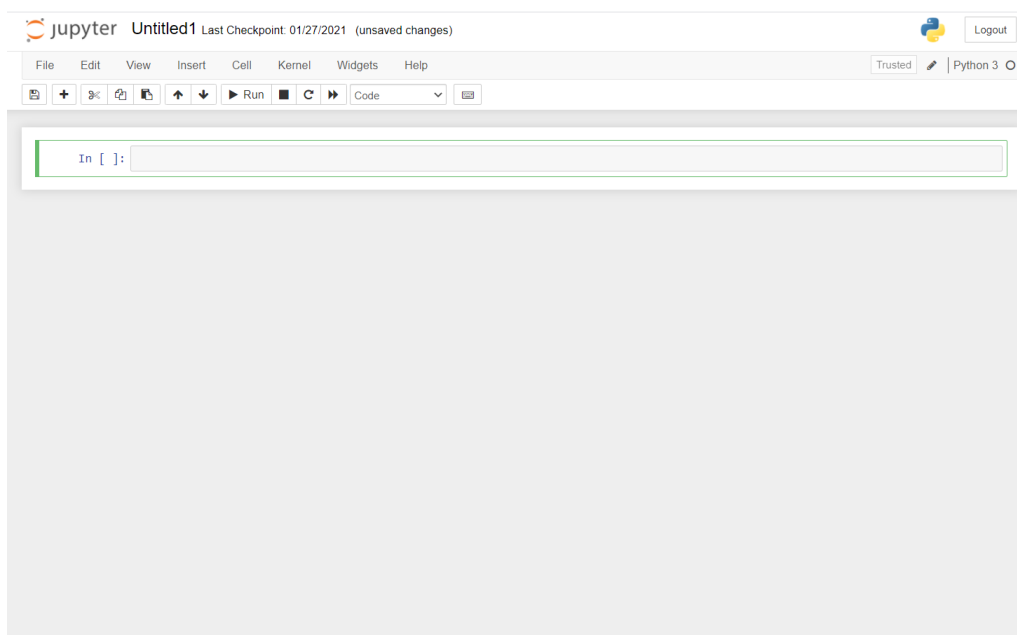


Slika 5.4: Primjer jedne slike iz skupa podataka.

Prije treniranja mreže bilo je još potrebno malo urediti slike (promijeniti nazive te ih klasificirati po odgovarajućim datotekama).

## Treniranje mreže

Glavno radno okruženje za razvijanje i treniranje neuronske mreže bio je **Jupyter Notebook**. Jupyter Notebook je okruženje za prošireni rad u Python programskom jeziku koji omogućuje razvoj, dokumentiranje i izvođenje koda, kao i komuniciranje rezultata u web aplikaciji. Kombinira dvije komponente: web aplikaciju (alat temeljen na pregledniku za interaktivno upravljanje datotekama koje koriste popratne tekstove, matematičke i računske operacije i sadržajno bogati medijski izlaz) te Notebook dokumente (reprezentacije cjelokupnog sadržaja vidljivog u web aplikaciji). Iako se koristi u pregledniku, Jupyter Notebook je sasvim siguran jer se pokreće na vlastitom računalu koje postaje poslužitelj. Izgled sučelja za rad u Jupyter Notebooku prikazan je na slici 5.5. Jupyter Notebook bilježnice (ili skraćeno engl. *notebook*) sastoje se od više ćelija (označeno zelenom bojom na slici) koje sadrže dijelove koda. Svaka od ćelija može se zasebno izvršiti, a s ostalim ćelijama unutar iste bilježnice dijele memoriju odnosno prostor varijabli i metoda.



Slika 5.5: Izgled rada u Jupyter Notebook dokumentu.

Prije početka rada bilo je potrebno odrediti sve Python biblioteke potrebne za treniranje mreže. Glavna biblioteka koja se koristila zove se *Keras*, odnosno *TensorFlow*. **Keras** je Python biblioteka za duboko učenje. Ili preciznije, aplikacijsko programsko sučelje (API) visoke razine za neuronske mreže, pisano u Pythonu i izgrađeno nad TensorFlowom. Razvijen je s fokusom na omogućavanje brzog eksperimentiranja te se koristi za jednostavan



i brz razvoj prototipa, izgradnju konvolucijskih i povratnih neuronskih mreža te njihovih kombinacija te za neprimjetno izvođenje na procesorskim i grafičkim jedinicama. Karakteriziraju ga principi razumljivosti, modularnosti, jednostavnog proširivanja i rada s Pythonom. **TensorFlow** je s druge strane, programska biblioteka otvorenog tipa kreirana 2015. godine od strane GoogleBrain tima i najčešće je korištena za strojno učenje. U ovom radu, dakle, koristio se Keras koji pokreće TensorFlow koji mu u pozadini izvršava operacije i izračunavanja.

```
1 from keras.applications.imagenet_utils import preprocess_input
2 from keras.preprocessing.image import load_img, img_to_array
3 from keras.models import Model
4 from keras.applications import vgg16
5
6 from PIL import Image
7 import os
8 import pandas as pd
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from sklearn.metrics.pairwise import cosine_similarity
```

Kod 5.1: Biblioteke potrebne za rad

Tip mreže koji smo razvijali u ovom radu zove se **VGG16**. VGG16 jest model konvolucijske neuronske mreže predložen od strane K. Simonyana i A. Zissermana sa sveučilišta u Oxfordu u svom radu pod nazivom *Very Deep Convolutional Networks for Large-Scale Image Recognition*[28]. Mreža se trenirala tjednima i model ostvaruje točnost od 92.7% na skupu podataka pod imenom *ImageNet*. *ImageNet* jest skup podataka koji se sastoji od gotovo 15 milijuna označenih slika visoke rezolucije koje su klasificirane u otprilike 22000 kategorija. Svaka slika jest dimenzije  $256 \times 256$ . VGG kao ulazni podatak prima slike  $224 \times 224$  piksela.

```
1 images_path = "images/sportvision/" # put do foldera sa slikama
2 images_width, images_height = 224, 224 # dimenzije slika
3
4 convolutional_network_vgg = vgg16.VGG16(weights='imagenet')
```

Kod 5.2: Dimenzije slika i instanciranje vgg16 mreže

**Napomena 5.0.1.** Parametar `weights='imagenet'` određuje da model ne uči nove težine, već da ih ostavi ovakvima kakve su u originalnom modelu.

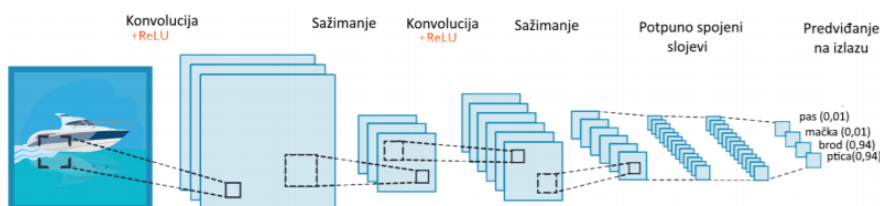
Ti ulazni podaci tj. slike se nadalje prosljeđuju skrivenim slojevima (engl. *layers*) koje mreža sadrži. Ona u skrivenim slojevima razlikuje dvije vrste slojeva: konvolucijske (engl. *convolution*) slojeve i slojeve sažimanja (engl. *pooling*). Zbog svoje arhitekture ova vrsta mreže je dobra u prepoznavanju obrazaca i uzoraka, zbog čega se i koristi za

analizu slika. Pod obrasce i uzorke se misli na rubove, oblike, teksture i objekte. Za prepoznavanje obrasca se koriste takozvani filtri, a što mreža "ide dublje" u analizi slike, to joj je lakše prepoznati detalje na proizvodima na kojima učimo mrežu. Filter je ništa drugo nego obična matrica kvadratnih dimenzija koja prolazi svaki dio slike u koracima (engl. *stride*) i tako detektira razne objekte. U običnim neuronskim mrežama učenje mreže se postizalo izmjenom težinskih faktora, dok se kod konvolucijskih mreža uče težine unutar filtra.

1	2	-1
0	1	2
-2	-1	1

Slika 5.6: Primjer filtra.

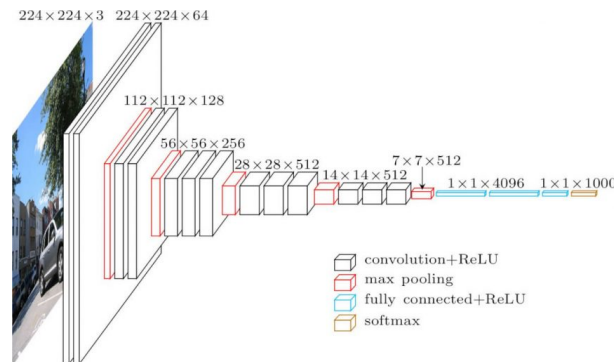
Nakon jednog ili više konvolucijskih slojeva, dolazi do sloja sažimanja. Oni se koriste s ciljem smanjivanja rezolucije. Na taj način smanjuju osjetljivosti na sitne pomake značajki na slici. Na slici 5.7 vidimo općeniti primjer rada konvolucijske neuronske mreže. Ulaz u mrežu je slika i nakon nje slijedi niz konvolucijskih slojeva i slojeva sažimanja. Zatim slijede potpuno spojeni slojevi i na kraju izlazni sloj koji pokazuje procjenu mreže.



Slika 5.7: Konvolucijska neuronska mreža.

VGG16 mreža je posebna jer sadrži veliki broj hiperparametara i u konvolucijskim slojevima koristi filtere dimenzija 3x3 s korakom 1. Slojevi sažimanja koriste filtere dimenzija 2x2 s korakom 2. Na slici 5.8 vidimo detaljniji prikaz arhitekture ovakve mreže. Vidljiva je njezina konstrukcija koja završava s dva potpuno spojena sloja i izlaznim slojem koji predstavlja *softmax* funkciju. Softmax funkcija daje predviđanje u postocima za svako od ponuđenih rješenja. VGG16 ima preko 138 milijuna parametara što ju čini veoma sporom,

a naučene težine zauzimaju puno memorije, no vrlo se lako može implementirati Kerasom i ima veliku preciznost, pa je zbog toga korištena u ovom radu.



Slika 5.8: Arhitektura VGG16 mreže.

U programskom kodu to bi izgledalo ovako:

```

1 features = Model(inputs=convolutional_network_vgg.input, outputs=
  convolutional_network_vgg.get_layer("fc2").output)
2
3 # Slojevi konvolucijske neuronske mreze
4 features.summary()
5 # potrebno je slike pretvoriti u 'numpy array' kako bi mogli baratati
  sirovim podacima
6 # PIL (width, height, channel) vs. Numpy (height, width, channel)
7 numpy_slika = img_to_array(testna_slika)
8
9 # batch format
10 image_batch = np.expand_dims(numpy_slika, axis=0)
11 print('Batch:', image_batch.shape)
12
13 # predprocesuiranje slike kako bi je prilagodili za konvolucijsku mrezu
14 procesuirana_slika = preprocess_input(image_batch.copy())
15
16 # dobavi features
17 image_features = features.predict(procesuirana_slika)
18
19 print("Broj featurea:", image_features.size)
20 image_features

```

Kod 5.3: Biblioteke potrebne za rad

Nadalje, ovaj postupak je potrebno ponoviti za sve slike. To činimo uz pomoć *for*-petlje prolazeći po svim slikama koje su spremljene i očitane iz zadane datoteke u kojoj se nalaze.

```

1 slike = [images_path + x for x in os.listdir(images_path) if "jpg" in x]
2 ucitaneSlike = []
3
4 for img in slike:
5     trenutna_slika = img
6     slika = load_img(trenutna_slika, target_size=(224, 224))
7     numpy_slika = img_to_array(slika)
8     image_batch = np.expand_dims(numpy_slika, axis=0)
9
10    ucitaneSlike.append(image_batch)
11
12 all_images = np.vstack(ucitaneSlike)
13
14 procesuirana_slika = preprocess_input(all_images.copy())

```

Kod 5.4: Određivanje značajki za svaku sliku proizvoda

Potom računamo sličnost slika. To računamo uz pomoć kosinusne sličnosti. Ta metoda je već ugrađena u Python biblioteci *sklearn.metrics.pairwise*.

```

1 cosSimilarities = cosine_similarity(image_features)
2
3 # spremi rezultate
4 cos_similarities_df = pd.DataFrame(cosSimilarities, columns=slike, index
   =slike)
5 cos_similarities_df.head()

```

Kod 5.5: Računanje koeficijenta sličnosti između zadanog proizvoda i svih ostalih

Kao rezultat dobijemo spremljene koeficijente sličnosti. Kako bi korisniku predložili neke slične proizvode sada je potrebno uzeti  $n$  slika koje imaju najviši koeficijent. U ovom radu je odabrano  $n = 2$ . Dakle, 2 najbližnje slike vraćamo kao rezultat.

```

1 number_of_similar_images = 2 # broj slicnih slika
2
3 najslicnije_slike = cos_similarities_df[slika].sort_values(ascending=
   False)[1:number_of_similar_images+1].index
4 najslicnije_slike_score = cos_similarities_df[slika].sort_values(
   ascending=False)[1:number_of_similar_images+]

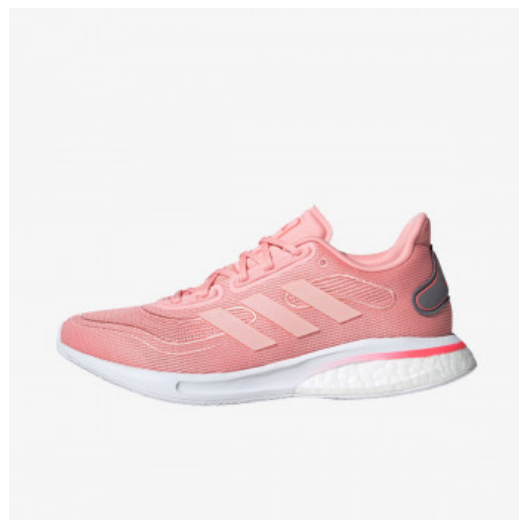
```

Kod 5.6: Spremanje prvih 5 najbližnjih slika

Sada je potrebno testirati program. Kao testnu sliku, odabrat ćemo sliku sa već navedenim proizvod sa slike 5.4. Njega ubacujemo kao ulazni podatak u našu istreniranu mrežu. Izlazni podatak će nam biti dva najbližnja proizvoda. Na idućoj slici vidimo što program izbacuje. Zaista se vidi sa slika kako ljudskim okom dosta slično izgledaju zadani primjer te programski rezultat.



(a) Koeficijent sličnosti: 0.921169



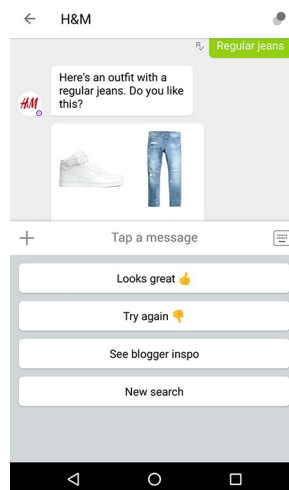
(b) Koeficijent sličnosti: 0.8934374

Slika 5.9: Dva najbližnja proizvoda.

## Poglavlje 6

### Zaključak

Daljnje mogućnosti nastavka rada na ovaj prijašnji program su neiscrpne. Ovaj program se može integrirati unutar nekakvog personaliziranog chatbota. Taj chatbot možemo postaviti kao dio nekakve postojeće platforme kao npr. Messenger bot ili Slack bot. Postoji mogućnost ugrađivanja unutar nekog razvojnog okvira poput *Dialogflow* (postoje i razni drugi razvojni okviri). Može također, biti ugradbeni dio nekakvog virtualnog asistenta na toj web-aplikaciji, kao što pokazuje primjer sa slike 6.1. Opcija postoji, između ostalog, da bude dio nekakve nespomenute mobilne aplikacije ili web aplikacije.



Slika 6.1: Primjer virtualnog asistenta.

Ne samo integracija, nego funkcionalnost, pa čak i izgled tog potencijalnog chatbota mogu jako varirati. To jako dobro pokazuje slika ispod uspoređujući neke chatbotove koji su u funkciji.

Templates:	chatfuel	Botify	OnSquad	Motion AI	Flow XO
<b>Gallery</b>	●	●	●	●	●
<b>Text card</b>	●	●	●	●	●
<b>Quick reply</b>	●	●	●	●	●
<b>Image</b>	●	●	●	●	●
<b>Inline video</b>	●	●	●	●	●
<b>Multiple languages support</b>	●	●	●	●	●
<b>Emoji recognition</b>	●	●	●	●	●
<b>Stickers recognition</b>	●	●	●	●	●
<b>Image recognition</b>	●	●	●	●	●
<b>Persistent menu</b>	●	●	●	●	●
<b>Polls / Feedback</b>	●	●	●	●	●
<b>Delay / Typing icon</b>	●	●	●	●	●
<b>Share button</b>	●	●	●	●	●
<b>Call button</b>	●	●	●	●	●
<b>E-commerce / Buy button</b>	●	●	●	●	●
Platforms:					
<b>Facebook Messenger</b>	●	●	●	●	●
<b>Kik</b>	●	●	●	●	●
<b>Skype</b>	●	●	●	●	●
<b>Telegram</b>	●	●	●	●	●
<b>Slack</b>	●	●	●	●	●
<b>WeChat</b>	●	●	●	●	●

Slika 6.2: Usporedba chatbot funkcionalnosti.

Dakle, chatbot programi su nova višenamjenska sučelja čija implementacija može biti riješena na mnoge načine u različite svrhe, bilo poslovne ili korisničke. Mogu kreirati nove usluge ili nadograditi postojeće. Stoga, vrijedi istaknuti kako su oni već sada sadašnjost koja nas okružuje, a sudeći po trendu rasta i naša su budućnost.

# Bibliografija

- [1] *sintaksa*, 2021, <https://enciklopedija.hr/Natuknica.aspx?ID=56165>.
- [2] 8. *Pravopisni znakovi*, <https://pravopis.hr/kategorija/pravopisni-znakovi/46/>.
- [3] *Dialogflow*, <https://cloud.google.com/dialogflow/docs/>.
- [4] *A free web scraper that is easy to use*, <https://www.parsehub.com/>.
- [5] *Keras vs Tensorflow: Must Know Differences!*, <https://www.guru99.com/tensorflow-vs-keras.html>.
- [6] *Pojmovnik - Hrvatski mrežni rječnik*, <http://ihjj.hr/mreznik/page/pojmovnik/6/>.
- [7] *VGG16 – Convolutional Network for Classification and Detection*, <https://neurohive.io/en/popular-networks/vgg16/>.
- [8] *View of Social bots distort the 2016 U.S. Presidential election online discussion: First Monday*, <https://firstmonday.org/article/view/7090/5653>.
- [9] James Allen, *Natural Language Understanding*, Addison Wesley Longman, 1987 (engleski).
- [10] Benjamin Bengfort, Rebecca Bilbro i Tony Ojeda, *Applied Text Analysis with Python*, O'Reilly, 2018 (engleski).
- [11] Steven Bird, Ewan Klein i Edward Loper, *Natural Language Processing With Python*, O'Reilly, 2009 (engleski).
- [12] Alexander Clark, Chris Fox i Shalom Lappin, *The Handbook of Computational Linguistics and Natural Language Processing*, Wiley-Blackwell, 2010 (engleski).



- [13] Fahim Muhammad Hasan, *Comparison of Different Pos Tagging Techniques for Some South Asian Languages*, Diplomski rad, Department of Computer Science and Engineering of BRAC University, <https://core.ac.uk/download/pdf/61799956.pdf>, 2006.
- [14] Nitin Indurkha i Frederick J. Damerau, *Handbook of Natural Language Processing*, Chapman and Hall/CRC, 2010 (engleski).
- [15] Grant S. Ingersoll, Thomas S. Morton i Andrew L. Farris, *Taming Text: How to Find, Organize, and Manipulate It*, Manning, 2013 (engleski).
- [16] Dan Jurafsky i James H. Martin, *Speech and Language Processing: An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Dorling Kindersley Pvt, Ltd., 2014 (engleski).
- [17] Rashid Khan i Anik Das, *Build Better Chatbots*, Apress, 2018 (engleski).
- [18] Hobson Lane, Hannes Max Hapke i Cole Howard, *Natural Language Processing in action: Understanding, analyzing, and generating text with Python*, Manning Publications Co., 2019 (engleski).
- [19] Christopher D. Manning i Hinrich Schutze, *Foundations of Statistical Natural Language Processing*, MIT, 1999 (engleski).
- [20] Warren S. McCulloch i Walter H. Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, (1943), <http://www.cse.chalmers.se/~coquand/AUTOMATA/mcp.pdf> (engleski).
- [21] Ruslan Mitkov, *The Oxford Handbook of Computational Linguistics*, Oxford University Press, 2003 (engleski).
- [22] Peter Naur i J. W. Backus, *Report on the Algorithmic Language ALGOL 60*, (1960), [http://www.softwarepreservation.org/projects/ALGOL/report/Algol60\\_report\\_CACM\\_1960\\_June.pdf](http://www.softwarepreservation.org/projects/ALGOL/report/Algol60_report_CACM_1960_June.pdf) (engleski).
- [23] Ivan Pandžić, *Oblikovanje Korjenovatelja Za Hrvatski Jezik*, Oct 2015.
- [24] Ena Pribisalić, *Obrada prirodnog jezika*, Diplomski rad, Sveučilište J.J. Strossmayera u Osijeku, Odjel za matematiku, <http://www.mathos.unios.hr/mdjumic/uploads/diplomski/PRI14.pdf>, 2019.
- [25] Sumit Raj, *Building Chatbots with Python*, Apress, 2019 (engleski).
- [26] Claude Shannon i Warren Weaver, *A Mathematical Theory of Communication*, (1948), <http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf> (engleski).

- [27] Amir Shevat, *Designing Bots: Creating Conversational Experiences*, O'Reilly, 2017 (engleski).
- [28] Karen Simonyan i Andrew Zisserman, *Very Deep Convolutional Networks For Large-scale Image Recognition*, (2015), <https://arxiv.org/pdf/1409.1556.pdf> (engleski).
- [29] Alan Turing, *Computing Machinery and Intelligence*, (1950), <https://www.csee.umbc.edu/courses/471/papers/turing.pdf> (engleski).
- [30] Kristina Vuković, *Model Parsera Za Hrvatski Jezik*, Doktorska disertacija, Sveučilište u Zagrebu, Filozofski fakultet, Odsjek za informacijske znanosti, <https://www.bib.irb.hr/388476>, Apr 2009.
- [31] Jan Šnajder i D. B. Bašić, *String Distance-Based Stemming of the Highly Inflected Croatian Language*, (2009), <https://aclanthology.org/R09-1074.pdf> (engleski).
- [32] Željko Agić, *Pristupi Ovisnom Parsanju Hrvatskih Tekstova*, Doktorska disertacija, Sveučilište u Zagrebu, Filozofski fakultet, Odsjek za informacijske znanosti, <https://www.bib.irb.hr/587199>, Jul 2012.

# Sažetak

U ovom radu detaljno je opisana obrada prirodnog jezika na računalima. Poseban naglasak se stavio na sekvencijalne faze kojima se najčešće obavlja prirodna obrada jezika. Ulazni podatak računalne obrade jest nekakav tekst u prirodnom jeziku. Potom kreću redom faze: tokenizacija, leksička analiza, sintaktička analiza i semantička analiza. Svaka od tih faza je detaljnije opisana s različitim alatima i procesima koji se koriste unutar nje.

# Summary

This thesis describes in detail the processing of the natural language on computers. Special emphasis has been placed on sequential stages that are the most common in language processing. The input data of the computer processing is some kind of text in the natural language. Then proceed in sequence: tokenization, lexical analysis, syntax analysis and semantic analysis. Each of these phases is described in more detail with the different tools and processes used within it.

# Životopis

Zovem se Roko Kokan i rođen sam 28.06.1995 godine u Splitu. Odrastao sam u mjestu Kaštel Sućurcu, gdje sam i pohađao Osnovnu školu kneza Mislava. Osnovnu školu sam završio 2009. godine, te sam po završetku upisao prirodoslovno-matematičku gimnaziju u Splitu (tzv. MIOC). Tijekom srednjoškolskog obrazovanja, sudjelovao sam na brojnim natjecanjima: županijska natjecanja iz fizike i logike, te državna natjecanja iz matematike (svake godine). Godine 2013. završavam srednju školu i potom upisujem preddiplomski studij Matematika na Prirodoslovno matematičkom fakultetu u Zagrebu. Nakon tri godine završavam preddiplomski studij te potom uzimam godinu mirovanja zbog zdravstvenih problema. Nakon toga 2017. godine upisujem diplomski studij Računarstvo i matematika na istom fakultetu na kojem sam završio preddiplomski studij. Tijekom diplomskog studija, radio sam dvije i pol godine studentski posao kao C++ inženjer u perspektivnoj hrvatskoj firmi koja se zove Photomath.