Logičko programiranje i razumijevanje prirodnog jezika

Šoljić, Filip

SVEUČILIŠTE U ZAGREBU

FAKULTET HRVATSKIH STUDIJA

Filip Šoljić

LOGIČKO PROGRAMIRANJE I RAZUMIJEVANJE PRIRODNOG JEZIKA

ZAVRŠNI RAD

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET HRVATSKIH STUDIJA

ODSJEK ZA FILOZOFIJU

FILIP ŠOLJIĆ

LOGIČKO PROGRAMIRANJE I RAZUMIJEVANJE PRIRODNOG JEZIKA

ZAVRŠNI RAD

Mentor: doc. dr. sc. Sandro Skansi

Zagreb, 2020.

Sadržaj

1. Uvod

Kao što sam naslov sugerira, tema mog završnog rada biti će iznošenje nekih od izazova i zahtjeva s kojima se susreće obrada prirodnog jezika od strane računarstva i kako tu od koristi mogu biti logika i filozofija jezika. Strukturarno, rad će biti podijeljen na dva dijela, prvi dio će se baviti programskim paradigmama, s naglaskom na logičkoj paradigmi, odnosno Prologom kao programskim jezikom, točnije, što je on, njegova povijest i kako funkcionira, dok će se drugi dio baviti Prologom i obradom prirodnog jezika, na koji način Prolog pristupa toj tematici, a na koji način neki drugi obrasci poput Deep Learning-a, te koji su prednosti a koji nedostaci svakog od modela i kako bi tu eventualo filozofija jezika mogla pomoći kada nijedan od modela nema adekvatno riješenje ili čak pravilno postavljen model problema.

Svekolika informatizacija i neporecivi tehnološki napredak stubokom mijenjaju, ne samo tržište rada već i sam način života, te tjera čovječanstvo na sve ubrzaniji adaptaciju i aklimatizaciju na nove industrijske i tehnološke paradigme. Upravo toj paradigmi je imanentan svojevrsni utilitarizam, te donekle postavljeno pitanje smisla i svrhovitosti određenih područja koja ne hvataju korak sa tehnološkim dostignućima, naročito društvene i humanističke znanosti čij je filozofija dio. Upravo je akademsko bavljenje filozofijom često puta dovedeno u pitanje, te proglašeno za anakronizam i arhaizam kojem jednostavno nema mjesta u digitalnom dobu. Implicitna namjera mog rada je pokazati kako su takva razmišljanja pogrešna, posebno ako uzmemo u obzir da je logika, kao filozofska disciplina, temelj modernog računarstva, te da su filozofi i posrednog filozofija itekako relevantni u 21. stoljeću.

Filozofija kao akademska disciplina u današnjem dobu ima svojih mana i nedostataka, određena krutost, strogi formalizam i ekskluzivnost znanosti znaju biti zapreka u interdisciplinarnoj suradnji sa drugim granama znanosti ili područjima djelatnosti, no ono što je itekako pozitivno i vrijedno u bavljenju filozofijom su alati kojim vas filozofija obdari, prvenstveno u vidu kognitivnih procesa, kritičkog prosuđivanja, načina razmišljanja koji su od inteligibilnog značaja na današnjem tržištu rada, koji sve manje zahtjeva formalno obrazovanje a sve više određeni mentalni sklop, način razmišljanja i riješavanja problema koji u velikoj mjeri koincidira sa onim što filozofija nudi kao akademska disciplina danas.

2. **Programske paradigme**

Proje nego što se detaljnije upustim u analizu logičkog programiranja kroz programski jezik Prolog, pokušati ukratko objasniti programske paradigme i razložiti nekoliko temeljnih. Programska paradigma, laički rečeno je način pisanja kompjuterskog programa. Najopćenitija podjela programskih paradigmi je na deklarativnu i na imperativnu. Obje paradigme služe za rješavanje algoritamskih problema. U svijetu programiranja, algoritmi su set naredbi koje programski jezik izvršava kako bi riješio određeni zahtjev. Deklarativnu paradigmu krasi rješavanje problema na višoj razini apstrakcije prilikom čega nije potrebno mijenjanje stanja memorije dok kod imperativne paradigme to je slučaj.

2.1. Proceduralna paradigma

Proceduralna programska paradigma ,kao što joj i ime govori, temelji se na korištenju procedura , gdje se svaka procedura može pozvati u bilo kojem trenutku izvršavanja programa uključujući pozivanje unutar drugih procedura i čak unutar sebe same. Pripada imperativnoj paradigmi. Velika prednost ovakve vrste paradigme je mogućnost da se program promatra kao skup modula ili potprograma,odnosno dobra organizacija koda i laka pretraga po njemu. Nedostatak takve paradigme je što može bit teška za razumijevanje i naknadno održavanje[1]. Proceduralnu paradigma je podržana od brojnih programskih jezika, od FORTRAN-a koji još od 1958. omogućava izradu korisnički definiranih procedura, Pascala, jezika C pa i kod aktualnijih poput Pythona i C++ .[2]

[1] *Progopedia*, »Paradigm: Procedural«
[2] VujoševićJaničić i Tošić, 2008, str. 69

2.2. Objektno-orijentirana paradigma

Kod Objektno-orijentirana paradigme, koja pripada pod imperativne paradigme, je zanimljivo da glavnu razliku čini filozofija razvoja programa, jer je to paradigma koja sadrži i sve elemente proceduralne paradigme ali koja je proširena konceptima klasa i njihovih objekata. [3]Objektno-orijentirana paradigma ili OOP u nastavku, se u suštini temelji na objektu. U tim objektima su sadržani podaci koji se nazivaju atributi ili svojstva. Suština OOP-a je da objekti mogu biti u inerakciji jedan sa drugima, pomoću procedura i na taj način mijenjati svojstva, biti ovisni jedni o drugima i nasljeđivati jedni drugi. C# je Microsoftov programski jezik koji je dobar primjer takvog jezika.

Neki od temeljnih koncepata objektno-orijentirane paradigme su:

- Enkapsulacija i apstrakcija
- Podtipovi – odnos kompatibilnosti koji se temelji na funkcionalnosti nekog objekta
- Nasljeđivanje – mogućnost pomnog korištenja neke metode koja se prethodno može definirati za neki drugi objekt ili za neku drugu klasu
- Dinamički odabir metoda
- Polimorfizam

[3] Li i Henry, 1993, str. 53

2.3. Logička paradigma

Logička programska paradigma se temelji na formalnoj logici, odnosno na logici prvog reda, a to je zapravo skup rečenica koje sadrže pravila i činjenice koje su vezane za neki problem. Za logičko programiranje su se uvelike koristili Prolog, jezik koji je centralni pojam ovoga rada i Datalog. U navedenim jezicima, pravila su se pisala u obliku rečenica: G:-T1,…, Tn. Gdje se G naziva glavom pravila, a T1, …, Tn tijelom pravila. Činjenice su pravila koja nemaju tijelo i jednostavno se pišu u obliku: G.[4]

Logičko programiranje spada pod deklarativnu paradigmu. Bit logičkog programiranja je programiranje sa relacijama i zaključivanjem.Programer je zaslužan za određivanje logičkih odnosa bez specificiranja kako će se primeti pri pravila i zaključivanja. Programski jezici logičke paradigme su multi paradigmatski programski jezici koji se sastoje od funkcionalne paradigme programiranja viseg reda a (engl. higher-order programming) i logičke programske paradigme (nedeterminističko programiranje i unifikacija).[5] Programska logika sastoji se od:

- Aksioma - definiranje činjenica o objektima
- Pravila - definiranje načina zaključivanja novih činjenica
- Izjava - definiranje teorema

Pošto se logičko programiranje temelji na logici prvog reda valja spomenuti njegove elemente :

- Abeceda
- Uvjeti definirani preko abecede
- Dobro formirana formula definirana preko abecede

[4] *Wikipedia*, »Logic programming«
[5] Vujošević-Janičić i Tošić, 2008, str. 75

Abeceda se sastoji od simbola i dijeli se u dva podskupa: skup logičkih simbola i skup nelogičkih simbola koji su specifični za određenu interesnu domenu. Skup logičkih simbola tako sadrži sljedeće elemente:

- Logičke veze: ¬ negacija, ∧ konjunkcija, ∨ disjunkcija, ⇒ implikacija, ⇔ ekvivalencija
- Predložene konstante koje su istinite ili lažne [6]

Kako se logičko programiranje temelji na logici prvog reda trebali bi odvojiti trenutak da prikažemo i njenu sintaksu koja je ovom prilikom opisana njenom abecedom:

Abeceda A jezika L(RP) računa predikata je unija sljedećih skupova simbola:

- A1 = {ci : i ∈ I ⊆ ℕ} – najviše prebrojiv skup konstanti, gdje je ℕ skup svih prirodnih brojeva
- A2 = {fI : I ∈ L ⊆ ℕ } – najviše prebrojiv skup funkcija svih konačnih kratnosti
- A3 = {Pk: k ∈ K ⊆ ℕ } – najviše prebrojiv skup predikata svih konačnih kratnosti
- A4 = {¬ , ∧ , ∨ , ⇒ ,⇔} – skup logičkih veznika
- A5 = {∀ , ∃} – skup kvantifikatora „za svaki“ i „postoji neki“
- A6 = {(,)} – skup lijeva i desna zagrada [7]

[6] Gabbrielli, Martini, 2010, str. 374
[7] Čubrilo, 1989, str. 72

Ono što logičko programiranje čini pogodnim za upoznavanje svijeta programiranja od strane nekoga tko se prije nije s tim susretao, je lako razumljiva sintaksa utemeljena na formalno je logici. Tako bi primjerice osobe koje su se bavile logikom prvog reda, odnosno formalnom logikom, mogle napraviti tranziciju u svijet programiranja i utilizirati znanja i alate koje su stekli u formalnoj izobrazbi u društvenim znanostima, točnije filozofiji. Ono što bi se moglo istaknuti kao nedostatak logičkog programiranja je loša organizacija koda i ne prikladan je za prikaz proceduralnog znanja.

3. **Prolog**

Prolog je deklarativan programski jezik koji spada pod logičku programsku paradigmu, temelji se na logici prvog reda. Od samih početaka od tamo 1970-ih, Prolog je bio jezik mnogih programera kada je simbolička nenumerička obrada u pitanju, gdje su neki od područja primjene:

- relacijske baze podataka
- logika
- apstraktno rješavanje problema
- razumijevanje prirodnog jezika
- dizajniranju automatizacij
- rješavanje simbličkih jednadžbi
- analiza biokemijske strukture
- razna područja prirodne inteligencije[8]

[8] Clokcsin, Melish - Programming in Prolog, str 1

Ono što je specifično za programiranje u Prologu je način rješavanja zadanog problema. Dok je u konvencionalnim[9] programskim jezicima uobičajeno propisivanje koraka ili sekvenci, odnosno algoritama, koje računalo mora izvršiti kako bi riješilo problem, u Prologu se to rješava opisivanjem poznatih činjenica i veza u zadanom problemu[10] što zahtijeva još veću razinu apstraktnog razmišljanja od strane osobe koja koristi Prolog.

3.1. Objekti i Relacije

Prilikom rješavanja problema u Prologu, bitno je znati prepoznati, odnsno odrediti objekte i relacije među tim objektima. Primjerice, kada kada kažemo " Ivan je vlasnik knjige" , mi zapravo određujemo relaciju, "vlasništvo" između dva individualna objekta, "Ivana " i " knjige". Nadalje, ta relacija ima specifičan redoslijed, Ivan je vlasnik knjige, knjiga nije vlasnik Ivana. Kada pitamo "Je li Ivan vlasnik knjige" mi pokušavamo otkriti vrstu relacije između dva objekta. Kao što vidimo rješavanje problema se svodi na računalo da sazna informacije o objektima i relacijama koji deriviraju iz našeg programa.[11]

Bitno napomenuti da ne da ne treba brkati objekte u prologu i objekte u objektno-orijentiranoj paradigmi koju smo ranije spominjali. dok su objekti u objektno-orijentiranoj paradigmi strukture podataka koje komuniciraju jedne s drugima na temelju inherentnih svojstva koje su specifične toj programskoj paradigmi (apstraktcija, nasljeđivanje, enkapsulacija, polimorfizam) , Prolog koristi objekt na drugačiji način, naime objekti su stvari koje možemo reprezentirati koristeći termine, odnosno objekt u Prologu je istoznačan terminu objekta u formalnom logici. Strukture podataka u Prologu su uniformne, one su temelj svih podataka koje čine program u Prologu a nazvane su termin. Prolog može djelovati kao prvi korak prema ultimativnom cilju logičkog programiranja zbog načina na koji je program u

[9] Clokcsin, Melish- Programming in Prolog, str 3
[10] Clokcsin, Melish- Programming in Prolog, str 6
[11] Clokcsin, Melish- Programming in Prolog, str 11

Prologu konstruiran. Program se sastoji od seta klauzula , gdje je svaki set ili činjenica o danim informacijama pravilo kako da se rješenje odnosi prema danima činjenicama.[12]

Kada bih transponirao ovaj način rješavanja problema u filozofsko-lingvističke sfere, odnosno opisivanja relacije između objekata u svakodnevnom govoru, onda bi gore navedeni primjer djelovao još intuitivnije. Primjerice, rečenica " Dvije osobe su sestre ako su obje ženskog spola i ako imaju iste roditelje" nam govori kako možemo otkriti kako se netko može smatrat sestrama: jednostavno provjeri jesu ženskog spola i jesu li u rodu. Iako se čini da su pravila iznimno simplificirana, jer u stvarnom životu biti sestrama može značiti puno više od gore navedenih pravila, ona su ipak prihvatljiva definicija, jer na kraju krajeva, nijedna definicija ma kakava god ona bila, nam ne može otkriti sve o nečemu.[13] Ali kako u programiranju, tako i u stvarnom životu, prilikom riješavanja određenih problema, nama ne trebaju sve informacije o objektima koji su dio problema, već samo ona informacije koje su relevantne i doprinose što efikasnijem riješavanju tog problema.

3.2. Programiranje u Prologu

Programiranje u Prologu se sastoji od :

- Činjenica
- Pravila
- Pitanja

Mi odredimo neke činjenice o objektima i njihovim relacijama da bi nakon toga definirali neka pravila o objektima i njihovim relacijama i na kraju postavili pitanja o objektima i njihovim relacijama. Koristeći primjer sa sestrama, mi možemo provjeriti jesu li neki objekti sestre tako što ćemo reći Prologu naše pravilo o tome što konstituira vezu "biti sestra"među dva objekta. Zatim postavimo pitanje jesu li Marija i Ivana sestre. Prolog će pretražiti kroz sve što smo mu mi naveli o ta dva objekta (Marija i Ivana) i vratiti nam odgovor da ili ne.

[12] Clokcsin, Melish- Programming in Prolog, str 18
[13] Clokcsin, Melish- Programming in Prolog, str 29

Stoga bismo mogli gledati na Polog kao skladište činjenica i pravila koje onda koristi kako bi odgovorio na pitanja. U Prologu, kolekcija činjenica se naziva “baza”. Prolog naravno može učiniti puno više od odgovarati sa da ili ne na određena pitanja. Prolog omogućuje da se na temelju “uskladištenih” činjenica i pravila pronađu načini kako da se donesu zaključci preko jedna činjenice do druge, a sve utemeljeno na deduktivnoj logici. Ono što također krasi Prolog je interaktivnost slična konverzaciji, između osobe i računala. Osoba preko tipkovnice unese određene činjenice i pravila glede određenog problema koji želi riješiti. Nakon što postavi prava pitanja, Prolog će iznjeti potrebne odgovore na zaslonu računala.

3.3. Činjenice

Najbolji način objašnjavanja činjenica u Prologu je preko jednostavnih primjera. Ako hoćemo reći Prologu da se “ Ivanu sviđa Ana”, onda uočavamo da se pva činjenica sastoji od dva objekta, “Ivana” i “Marije” i relacije među njima, “sviđa”. Zapisivanje ovakve izjave u Prologu, standardnom formom bi izgledalo ovako:

```
sviđa(ana, ivan).
```

Kao što vidimo relacija je napisana prva, dok su objekti razdvojeni zarezom, i nalaze se u uglatim zagradama. Imena relacije i objekata su napisana malim početnim slovima i i točka dolazi na kraju izjave. Kada definiramo relaciju između objekata koristeći činjenice, moramo voditi računa o redoslijedu objekata u zagradama. U gore navedenom primjeru, objekt koji se nekome sviđa je drugi po redu. Iako je taj redoslijed proizvoljan, mi smo ga odredili, tog redosljeda se nadalje moramo držati jer bi drugačiji redoslijed dao drugačije značenje, kada bi obrnuli redosljed, to bi značilo da Ana je ta kojoj se sviđa Ivan a ne obrnuto kako smo naveli na početku. Odabir imena objekata i relacija je također proizvoljan, jednako tako smo umjesto ana i ivana mogli stavit a i b a relaciju imenovati sa c, pa bi taj isti iskaz izgledao ovako:

```
c(a,b).
```

Neki od primjera činjenica u Prologu i njihovog tumačenja u prirodnom jeziku:

vrijedno(dijamant). Dijamant je vrijedan.

žensko(klara). Klara je žensko.

vlasnica(Klara, dijamant). Klara je vlasnica dijamanta.

otac(petar, marija). Petar je Marijin otac.

daje(petar, knjiga, klara). Petar daje knjigu Mariji.

Problem koji možemo uočiti u gore navedenim izrazima i njihovom značenju je referiranje, odnosno denotiranje. Kao govornici hrvatskog jezika, intuitivno nam je tumačiti vlastita imena u kontekstu koji nam je zadan, no što je sa nazivima koja ne denotiraju nešto sasvim razvidno na prvu. Takve riječi logičari zovu "non-count word" ili nebrojena riječ na hrvatskom. Naime neka riječ može imati više značenja i samim time može bit problematična prilikom tumačenja u danom kontekstu. Zato bi programeri, oni koji imenuju objekte trebali biti konzistentni prilikom imenovanja kako bi se takvi problemi sveli na minimum.

U primjerima :

sviđa(ana, ivan).

žensko(klara).

imena objekata u zagradi se zovu argumenti dok se relacija naziva predikat. Što bi značilo da je "sviđa" predikat koji ima dva argumenta, dok je "žensko" predikat koji ima jedan argument.

3.4. Pitanja

Sljedeći korak, nakon upisivanja činjenica u program, je postavljanje pitanja o tim činjenicama. Sintaksa postavljanja pitanja u Prologu izgleda ovako:

?- igra(petar, nogomet).

Prolog provjera gore postavljeno pitanje pomoću unifikacije činjenica u pitanju. Prolog će proći kroz svoju bazu i potražit će činjenice koje odgovaraju činjenicama u postavljenom pitanju. Činjenice će biti unificirane ako su njihovi predikati isti (podudaraju se sintakticki) i ako im je broj argumenata isti. Što bi primjerom bilo izraženo na sljedeći način:

1. Prvo bi unijeli neke činjenice u naš program:

sviđa(ivi, jabuka).

sviđa(kati, filip).

sviđa(maji, knjiga).

sviđa(nevenu, knjiga).

sviđa(luki, italija).

1. Nakon toga bi mogli postaviti sljedeća pitanja Prologu , i Prolog bi mogao odgovoriti:

?- sviđa(ivi, lubenica).

no

?- sviđa(kati, karlo).

no

?- sviđa(maji, knjiga).

yes

Važna napomena je da *no* u gore navedenim slučajevima znači kako ništa ne unificira činjenice sa pitanjem, a ne da znači *false* u booleanskom smislu koje je svojstveno za druge vrste programskih paradigmi. Također, do sada navedene karakteristike i funkcionalnosti djeluju suviše rudimentarno, nimalo uzbudljivo ako se uzme u obzir da smo do sada samo unijeli neke činjenice u bazu i postavili neka jednostavna pitanja na koje nam je Prolog odgovorio. Bilo bio puno korisnije kada bi nam Prolog mogao dati odgovor na pitanje tipa *Koje objekte voli Klara?* Da bi dobili odgovor na pitanje, moramo uvesti *varijable* u igru.

3.5. Varijable

U Prologu uz opciju da možemo nazvati objekta određenim nazivljem, također možemo uzeti neodređene termine tipa X da označavaju objekte koje mi nismo imenovali. Varijabla može biti *instancirana* ili *ne instancirana*. Varijabla je instancirana kada postoji objekt koji je varijabla imenovala. Varijabla nije instancirana kada stoji na mjestu nekog objekta koji još nije poznat.[14] Sintaksa varijable u Prologu je ta da svaka varijabla počinje velikim slovom, kako bi ih Prolog distingvirao od imena objekata. Primjer upotrebe varijabli u Prologu:

?- sviđa(Ivan, *nešto što se ivanu sviđa).*

To bi značilo da će Prolog proći kroz svoju bazu činjenica i tražiti objekt koji predstavlja varijabla. Samo što bi se zapravo taj izraz u Prologu zapisao na sljedeći način:

?- sviđa(Ivan, X*).*

Prevedeno na prirodni jezik, pitanje glasi *Ima li išta što se Ivanu sviđa?* Ako bi u bazi činjenica bili neki objekti koje se Ivanu sviđaju, tada bi Prolog odgovorio:

X = knjiga

U slučaju da ima više objekata u bazi koje se sviđaju Ivanu, tada će varijabla biti instancirana na mjestu prvog objekta koji nađe, ovisno po redoslijedu kojim je taj objekt unesen u bazu

[14] Clokcsin, Melish- Programming in Prolog, str 33

činjenica. Kada naiđe na prvi objekt, tada će ispisati vrijednost varijable na ekran i čekati daljne upute. Ako korisnik stisne ENTER tipku ili “Return key” znači da je zadovoljan odgovorom i da program ne treba više tražiti po bazi. U slučaju da korisnik želi vidjeti koje se još stvari sviđaju Ivanu, onda će stisnuti točku-zarez na tipkovnici ili “ semicolon” nakon čega ENTER tipku, što će rezultirati daljnjom iteracijom po bazi činjenica, od mjesta gdje je program instanciran zadnju varijablu prije nego što je nastavio potragu. Što bi u samom programu izgledalo ovako:

?- sviđa(Ivan, X*).* naše pitanje

X* = *knjige; prvi odgovor. Utipkamo točku-zarez i ENTER tipku

X* = *filmovi; drugi odgovor. Ponovno utikamo točku-zarez i ENTER tipku

no nema više odgovora.

3.6. Konjukcije

Konjukcije su, laički rečeno , logički operatori koji nam pomažu da opišemo kompleksnije relacije među objektima. Uzmimo na primjer sljedeću bazu:

sviđa(ana, čokolada).

sviđa(petra, vino).

sviđa(alen, vino).

sviđa(ivan, maja).

Sljedeći izraz u Prologu:

?- sviđa(ivan, maja), likes(maja, ivan).

se prevodi kao *Sviđa li se Maja Ivanu ?* **i** *Sviđa li se Ivan Maji ?* gdje **i** kao konjukt izražava kako nas zanima konjukcija ova dva pitanja, odnosno izraz u cjelini, koji je istinit samo ako su oba pitanja istinita. Nazovimo svaki od ovih pitanja cilj. Prolog kada dobije ovakav upit, pretražuje bazu činjenica kako bi zadovoljio oba cilja jer u ovom slučaju oba moraju biti istinita kako bi izraz bio istinit.

3.7. Pravila

U Prologu, pravila se odnose na situacije kada želimo reći da činjenice ovise o grupi drugih činjenica. Pravilo je generalna izjava o objektima i njihovim relacijama. Primjerice, Fred je ptica ako je Fred životinja i ako Fred ima perje.[15] Pravila zapravo definiraju nove relacije na temelju postojećih. U prirodnom jeziku koristimo riječ "ako" kako bi izrazili pravilo ili uvjet.:

"Otići ću na more ako dobijem godišnji."

U Prologu, pravilo se sastoji od *glave* i *tijela*:

(glava) (ako) (tijelo)

odlazak(X, more) :- dobitak(X, godišnji).

Gdje su glava i tijelo spojeni simbolom ":-" koji se čita kao *ako.* Glava pravila opisuje koju činjenicu pravilo namjera definirati, dok tijelo opisuje koja konjukcija ciljeva mora biti zadovoljena, jedna za drugom ako ih ima više, da bi glava bit istinita. Još jedna stvar o kojoj treba voditi računa je doseg ili "scope" varijable X. Doseg varijable X u gornjem iskazu je sve do točke, što znači da će se instanciranje varijable X odnositi jednako i u glavi i u tijelu pravila. Kao što smo vidjeli do sada,postoje dva načina za dobivanje informacija o danom

[15] Clokcsin, Melish- Programming in Prolog, str 39

predikatu u Prologu. Može se dobiti putem činjenica i pravila, gdje je najčešće predikat definiran mješavinom činjenica i pravila. To se zove klauzulom za predikate.[16]

S ovim dijelom smo prošli većinu temeljnih dijelova Prologa, poput utvrđivanja činjenica o objektima, postavljanja pitanja o tim činjenicama, korištenje varijabli i razumijevanje njihovih dosega, konjukciju, prikazivanju relacije u obliku pravila/uvjeta. To sve su mali gradivni blokovi s kojima je već moguće pisati korisne programe i manipulirati jednostavnim bazama podataka. U sljedećem dijelu moga rada pokušati prikazati neke od primjena Prologa, u obradi prirodnog jezika i u umjetnoj inteligenciji.

4. Prolog i obrada prirodnog jezika

Po pitanju obrade prirodnog jezika, generalni konsenzus je da je najpopularniji alat metoda Dubokog Učenja ili "Deep learning-a". Deep Learning, u nastavku DL, je postao popularan zahvaljujući svojem naprednom obrascu prepoznavanja koji su pogodni za rad sa skalabilnim podacima. Može detektirati ne-linearne obrasce sa količinom varijabli koje bi čovjeku bilo iznimno teško, ako ne i nemoguće. Dakle, pregnantni zaključak bi bio da su prednosti DL-a prepoznavanje obrazaca i prevođenje istih u predviđanja sa visokom stopom uspješnosti.

Kritike DL-a su usmjerene prema načinu na koji su postignuća proizvedena.DL treba enormno velike količine podataka za "trening" kako bi generirale pouzdane rezultate. Iako ima nekih metoda za reduciranje zahtjeva za trening podacima , kao npr. "transfer learning", i dalje postoji veliki broj slučajeva kada ogromne količine podataka nisu dostupne. Još veće ograničenje DL-a je to što njegovi modeli ne razumiju nužno to što analiziraju. Možda mogu prepoznati idealan obrazac za Obradu Prirodnog Jezika, u nastavku OBJ, ili za sisteme prepoznavanje slika, ali je DL ograničen u svojoj sposobnosti da prepozna značaj obrasca ili da izvuče nekakav zaključak iz njih. Dok s druge strane, Prolog je dizajniran da razumije i kreira zaključke na

[16] Clokcsin, Melish- Programming in Prolog, str 46

temelju podataka koje analizira. Na primjer, programi temeljeni na Prologu mogu brzo analizirati podatke značajnih količina, poput onih koji se koriste na burzi. Programi mogu označiti koncepte na koje se analitičari mogu fokusirati, čime umnogome smanjuju vrijeme koje bi ti isti analitičari potrošili čitajući i analizirajući te materijale, i to u puno većem obujmu nego što bi to ljudi mogli.

Još jedan dobar primjer korištenja Prologa u privatnom sektoru je kompanija Kyndi, start-up u Silicijskoj Dolini, gdje je njihov software napisan u Prologu, gdje navode kako je jedna od prednost Prologa upravo u činjenici da kroz procesuiranje činjenica i koncepata, pokušava riješiti zadatke koji nisu uvijek dobro definirani. Kyndi je uspio, sa vrlo malo trening podataka, automatizirat generiranje činjenica, koncepta i zaključaka. Konkretnije, Kyndi je uspio istrenirati 10 do 30 znanstvenih radova, sa rasponom stranica od 10 do 50 svaki. Jednom kada je "istreniran", njihov software može identificirati čak i koncepte, ne samo riječi. Ono što je također impozantno u njihovom software-u, je brzina izvršavanja zadatka. Prosječnom čovjeku bi trebala otprilike godina dana da analizira tisuću podužih znanstvenih dokumenata. Kyndy-evom software treba sedam sati za tih istih tisuću dokumenata. Cilj tog start-up-a je povećanje produktivnosti ljudske analize, no mogućnosti primjene su neslućene, jer takva vrsta analiziranja i zaključivanja u poljima gdje analiza podataka igra odlučujuću uloga, može biti revolucionarna u punom smislu te riječi.[17]

Iako je javnost trenutno usmjerena na DL, polje umjetne ili strojne inteligencije uvijek je uključivalo čitav dijapazon raznih tehnologija i tehnika koje rade zajedno. Mješavina strojnog učenja (statističkog i obraznog podudaranja), biheviorizma (odgovor na podražaj) i analitika temeljena na pravilima (pristup simboličke logike) je dala do sada najsnažniju metodu za analiziranje kompleksnih podataka. Upravo zbog svojih specifičnih pristupa, Prolog i DL, odnosno prednosti koje iz tih istih proizlaze, kombiniranjem ta dva pristupa potencijal za njihovu promjenu je uistinu izvanredan. U pogledu primjene zajedničke primjene u farmaceutskoj industriji, DL bi mogao identificirati koji su podaci relevantni za određene pokusne periode, kao i koje značajke tih podataka ih čine relevantnim. Prolog bi mogao napravit inteligentne zaključke o tome kako se ti podaci primjenjuju na specifično farmaceutsko svojstvo. Tako što zapravo

[17] *AI Business,* »Is Deep Learning too superficial?«

razumije značaj podataka u relaciji sa željenim poslovnim ciljevima, Prolog bi mogao indicirati na koje elemente se treba fokusirati prilikom određenog perioda.

Ono što je razvidno iz gore navedene analize je da interdisciplinarni, kompozitni pristup u polju strojne inteligencije trenutno polučuje najbolje rezultate jer nadilazi pojedinačne slabosti dok umnaža komparativne prednosti svakog od pristupa, koje onda daju puno bolje rezultate nego da svaki pristup pokuša riješiti dani problem sam od sebe.

5. **Zaključak**

Logičko programiranje ,te Prolog kao reprezentativni primjerak jezika koji se temelji na takvoj paradigmi, su itekako relevantni u današnje doba, iako počeci Prologa sežu u 70-e godine prošlog stoljeća. Relevantni su zbog inherentni svojstava kojih sam se djelomice dotaknuo u ovome radu, a koja su tražena i vrijedna na tržištu rada i koja moga olakšati pristup nekome tko se do sada nije bavio programiranjem ali imao je doticaja sa simboličkom logikom. Nekoga poput osobe koja studira ili je studirala filozofiju. Dok se ulazak u polje IT tehnologija, njihovog razvoja i poslovne primjene, više ne promatrra kroz prizmu formalnog obrazovanja, koje sve više gubi značaj eliminacijskog uvjeta koji razdvaja "inicirane" od ostalih, te se sve više oslanja na sposobonost usvajanja novih vještina, logičkog razmišljanja, prilagodbe i timskog rada, koje su itekako kongruentne sa temeljnim postulatima akademskog izučavanja filozofije, naročito logike. Jedan od problema s kojima se susreću studenti filozofije, nakon što završe fakultet je zasićeno i relativno malo tržište, gdje vrlo mali postotak diplomiranih filozofa nađe posao usko vezan uz područje za koje su se i školovali. No ono što bi ih trebalo motivirati je činjenica kako uz malo truda mogu napraviti tranziciju u neko drugo područje, gdje je potražnja puno veća , tržište koje je dinamično i zahtjeva cijeloživotno učenje i edukaciju jer je brzina kojom nove tehnologije zamijenjuju stare gotovo eksponencijalna, tržište u kojem će utilizirati sve ono što su stekli tokom obrazovanja, na način da će im to biti ujedno i komparativna prednost u određenim situacijama i temelj za daljni razvoj i stjecanje novih znanja i vještina. Jedan od od eklatatnih primjera je gore navedena kompanija Kyndi, koja koristi Prolog a koji je iznimno intuitivan

programski jezik, pogotov nekome tko se bavio logikom prvog reda, te bi uz relativno mali period učenja i prakse, mogao samouvjereno izaći na tržište rada i pronaći svoje mjesto pod suncem, u kompanijama poput Kyndi ili neke slične.

6. BIBLIOGRAFSKI PODACI

1. Clocksin,W., Mellish, C. (2003.) *Programming in Prolog: using the ISO standard* , New York: Springer-Verlag Berlin.

2. Wikipedia, *Logic programming*, (Stranica posjećena: 30. travnja 2020) , https://en.wikipedia.org/wiki/Logic_programming.

3. Progopedia, *Paradigm: Procedural*, (Stranica posjećena: 27. travnja 2020), http://progopedia.com/paradigm/procedural/.

4. Vujošević-Janičić, M., Tošić, D. (2008), »The role of programming paradigms in the first programming courses«, U Marjanović, M., Kadelburg, Z. (ur.) *The Teaching of Mathematics*, pp. 63-83. Beograd: Društvo matematičara Srbije.

5. Gabbrielli, M., Martini, S. (2010) *Programming Languages: Principles and Paradigms*. London: Springer Verlag-London Ltd.

6. Čubrilo, M. (1989*) Matematička logika za ekspertne sustave*. Zagreb: Informator.

7. Li, W., Henry, S. (1993) Maintenance Metrics for the Object-Oriented Paradigm. *U Proceedings First International Software Metrics Symposium*, pp. 52-60. Baltimore.

8. Daly, C. (2018.) »Is Deep Learning too superficial?«, *AI Business* , https://aibusiness.com/document.asp?doc_id=760684&site=aibusiness.

9. Lohr, S. (2018.) »Is There a Smarter Path to Artificial Intelligence? Some Experts Hope So«, *The New York Times*, https://www.nytimes.com/2018/06/20/technology/deep-learning-artificial-intelligence.html?emc=edit_th_180621&nl=todaysheadline.