TakeLab

Laboratorij za analizu teksta i inženjerstvo znanja
Text Analysis and Knowledge Engineering Lab
Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
Unska 3, 10000 Zagreb, Hrvatska

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1528

Semantička analiza matematičkih zadataka iskazanih riječima

Tena Perak

Zagreb, srpanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 3. ožujka 2017.

Predmet: **Analiza i pretraživanje teksta**

DIPLOMSKI ZADATAK br. 1528

Pristupnik: **Tena Perak (0036472453)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Semantička analiza matematičkih zadataka iskazanih riječima**

Opis zadatka:

Strojno razumijevanje matematičkih zadataka iskazanih riječima novo je i zanimljivo istraživačko područje na presjecištu obrade prirodnog jezika, ekspertnih sustava i simboličkog izračunavanja, sa značajnim primjenama u obrazovanju i podučavanju. Sa stajališta obrade prirodnog jezika, problem je posebno izazovan jer iziskuje preciznu semantičku analizu teksta koji je često vrlo kratak i koji se isprepliće s matematičkim izrazima. Jedan od izazova jest određivanje cilja zadatka, odnosno akcije koja se očekuje od onoga tko zadatak rješava.
U okviru diplomskoga rada potrebno je osmisliti sustav za semantičku analizu matematičkih zadataka iskazanih riječima. Osmisliti semantički prikaz teksta zadatka i matematičkih izraza koji se u njemu pojavljuju prikladan za određivanje cilja matematičkog zadatka. Osmisliti i razviti sustav za predviđanje cilja matematičkog zadatka temeljen na pravilima, metodama statističkog strojnog učenja ili kombinaciji ovih dviju metoda. Prikupiti ili pripremiti prikladan, ručno označen skup podataka matematičkih zadataka iskazanih riječima na engleskome jeziku. Na tom skupu provesti eksperimentalno vrednovanje rada sustava te načiniti detaljnu analizu pogrešaka sustava te statističku obradu rezultata. Radu priložiti izvorni i izvršni kod razvijenog sustava, skupove podataka i programsku dokumentaciju te citirati korištenu literaturu.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 29. lipnja 2017.

Mentor:

Izv. prof. dr. sc. Jan Šnajder

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Veliko Hvala Leonu Rotimu za svu pruženu pomoć pri modeliranju i implementaciji zadatka.

SADRŽAJ

1. Uvod **1**

2. Semantička analiza teksta **2**

2.1. Semantičko parsiranje 3

2.1.1. Tipovi reprezentacija 3

2.1.2. Metode parsiranja 6

2.2. Zaključivanje 10

3. Semantička analiza matematičkih zadataka **12**

3.1. Skup podataka 13

3.2. Formalna reprezentacija matematičkih zadataka 15

3.2.1. Tipovi 15

3.2.2. Gramatika 18

3.2.3. Usporedba sa semantičkim jezikom DOL 20

3.3. Generiranje akcija za matematički rješavač 21

3.3.1. Usporedba sa semantičkim jezikom DOL 26

4. Rezultati **27**

5. Zaključak **34**

Literatura **35**

A. Popis matematičkih zadataka **40**

1. Uvod

Automatizirano rješavanje matematičkih zadataka iskazanih riječima aktualna je tema istraživanja u području umjetne inteligencije i inteligentnih sustava. Računala su od samih početaka bila brža i uspješnija u izračunavanju vrijednosti matematičkih izraza u odnosu na čovjeka, no već najjednostavniji problemi iskazani riječima računalu predstavljaju izazov. S druge strane, ljudi razumiju tekst zadatka bez imalo napora, no tekstualni zadaci često su kamen spoticanja učenicima već u osnovnoj školi. Takvi zadaci zahtijevaju prevođenje teksta u simbolički zapis matematičkog znakovlja koji olakšava njihovo rješavanje, no taj korak nekoj djeci predstavlja nepremostivi problem.

Cilj ovog rada je razviti semantički analizator specifičnih tekstnih zadataka – matematičke formule popraćene tekstnim opisom načina njihovog rješavanja – koji na izlazu generira jednu ili više naredbi za simbolički rješavač Photomath. Ideja za temu rada proizašla je iz suradnje s istoimenom tvrtkom Photomath Inc. koja razvija mobilnu aplikaciju za rješavanje matematičkih izraza. Aplikacija podržava rješavanje širokog spektra zadataka i omogućuje definiranje akcija od strane korisnika (programera), no nedostaje mu sustav koji bi ih generirao samostalno iz teksta. Npr., za ulazni niz $x + 2y = 5$ nije jasno po kojoj varijabli bi rješavač trebao prikazati rješenje. Ovaj sustav trebao bi omogućiti da, ukoliko tekst zadatka glasi "*Solve the equation for* y.", Photomath generira samo rješenje po varijabli y, dok drugu mogućnost eliminira.

Implementirani semantički analizator sastoji se od dvije povezane cjeline: semantičkog parsera i modula zaključivanja. Semantički parser iz tokeniziranog teksta gradi semantičku reprezentaciju u obliku stabla pomoću domenski-specifičnog SML (engl. *Semantic Math Language*) jezika, a modul zaključivanja na temelju te reprezentacije generira naredbe za Photomath. Oba sustava koriste simbolički pristup semantičkoj analizi teksta zbog nedostatka većeg skup podataka za razvoj statističkih modela te radi zadržavanja mogućnosti finijeg kontroliranja sustava kroz unos znanja.

U poglavlju 2 dan je kratak pregled područja, a u poglavlju 3 detaljno je opisan razvijeni sustav i targetirani skup podataka. Rezultati modela popraćeni komentarima o uviđenim poteškoćama i daljnjem razvoju rješenja navedeni su u 4. poglavlju.

2. Semantička analiza teksta

Područje računalne semantike bavi se pitanjem automatizacije procesa konstruiranja značenjskih reprezentacija izraza prirodnog jezika (engl. *meaning representation*) te postupka rasuđivanja, odnosno donošenja odluka na temelju njih. Zaključivanje se provodi nad formalnim reprezentacijama teksta jer su izrazi govornog jezika često višeznačni, nepotpuni te imaju slobodnu formu – jezik dozvoljava različite načine iskazivanja semantički ekvivalentnog sadržaja. Semantička se analiza stoga može promatrati kroz dva međuovisna problema: Kako automatizirati generiranje semantičke reprezentacije teksta te kako ih koristiti da se automatizirano donose zaključi na temelju njih. U (Blackburn i Bos, 2005) detaljno su opisane osnovne tehnike semantičke analize teksta popraćene implementacijom u programskom jeziku Prolog.

Područje primjene je široko, a uključuje automatizirano odgovaranje na pitanja, izvlačenje relacija (engl. *relation extraction*), kontroliranje robota i računalnih sustava, sučelje prema bazama podataka i sl.

Neki od glavnih izazova semantičke analize jesu nedefiniranost opsega (engl. *scope ambiguities*) te određivanje referenta zamjenica (engl. *anaphora resolution*). Problem opsega varijabli javlja se kod sintaksno jednoznačnih izraza koje možemo semantički interpretirati na različite načine, pa nemamo indikator za odbacivanje nekog od mogućih značenja. Nadalje, zamjenice se često koriste u tekstu radi izbjegavanja opetovanog ponavljanja imena i naziva objekata. Dok je čovjeku jednostavno odrediti na koga ili što se zamjenica odnosi, računalo na neki način mora donijeti odluku koji će joj poznati entitet pridružiti.

U nastavku su pobliže opisane poznate metode parsiranja teksta u formalni zapis te zaključivanja pomoću njih. Iako u ovom radu ne koristim predikatnu logiku, ona čini okosnicu većine postojećih istraživanja na području računalne semantike, pa je opisana relativno detaljno skupa s pripadajućim postupkom zaključivanja.

2.1. Semantičko parsiranje

Semantičko parsiranje podrazumijeva proces mapiranja rečenice prirodnog jezika u formalnu reprezentaciju njezinog značenja. Semantički parser koristi unaprijed definirani skup logičkih konstanti (ontologiju) za konstrukciju reprezentacije značenja rečenice, što je ujedno i razlog nedostatka potpunosti (engl. *completeness*) ovog pristupka. Odabir reprezentacije centralno je i ključno pitanje jer ograničava mogućnosti sljedećeg koraka analize – zaključivanja.

Jedan od glavnih zadataka semantičkog parsiranja jest osigurati sto veću pokrivenost modela, tj. mogućnost parsiranja različitih jezičnih izraza, neovisno o domeni primjene. (Cai i Yates, 2013; Kwiatkowski et al., 2013) primjerice koriste veliku količinu podataka (bazu znanja *Freebase*) za razvoj semantičkih parsera otvorene domene koji se mogu koristiti za automatsko procesiranje upita u vrlo različitim sustavima. Ovaj rad ne dotiče se problema pokrivenosti jezika jer je tema domenski uska, pa je leksikon potreban za mapiranje zadataka zapravo poprilično ograničen.

U potpoglavlju 2.1.2 dan je pregled postojećih postupaka semantičkog parsiranja, a u 2.1.1 detaljnije su opisani neki od najčešće korištenih oblika formalnih reprezentacija teksta.

2.1.1. Tipovi reprezentacija

Logičke forme

Logičke forme jesu formule logičkog jezika koje na formalan i koncizan način opisuju značenje teksta. Pogodne su za korištenje prilikom semantičke analize jer se zaključivanje nad njima može raditi pomoću poznatih sustava za dokazivanje teorema.

Logika prvog reda ili predikatna logika definirana je svojim formalnim jezikom koji sadrži sljedeće konstrukte:

- konstante
- varijable
- predikate
- kvantifikatore – $\forall, \exists$

Predikati povezuju varijable i konstante te nose semantičku vrijednost, a kvantifikatori služe za *vezanje* varijabli. Varijablu koja nije u dosegu niti jednog od kvantifikatora zovemo *slobodna varijabla*. Npr., u formuli:

$$\forall x P(x) \wedge Q(x)$$

x je vezana varijabla samo u prvom predikatu, odnosno, ekvivalentna je s formulom:

$$\forall x P(x) \wedge Q(y)$$

Primjer prevođenja jednostavne rečenice prirodnog jezika "Svi kvadrati su pravokutnici." glasi:

$$\forall x(kvadrat(x) \rightarrow pravokutnik(x))$$

gdje su *kvadrat* i *pravokutnik* predikati logike, a varijabla x je univerzalnim kvantifikatorom vezana u oba predikata.

Ranije spomenuti problem dvosmislenost opsega varijabli pojavljuje se kod generiranja izraza predikatne logike. Na primjer, rečenicu *"Svaki muškarac voli ženu."* moguće je interpretirati na 2 načina: tvrdnjom da svi muškarci vole istu žensku osobu

$$(\exists x \forall y((žena(x) \wedge muškarac(y)) \rightarrow voli(y, x))$$

ili da svaki voli neku drugu

$$(\forall y(muškarac(y) \rightarrow \exists x(žena(x) \wedge voli(y, x)))\,.$$

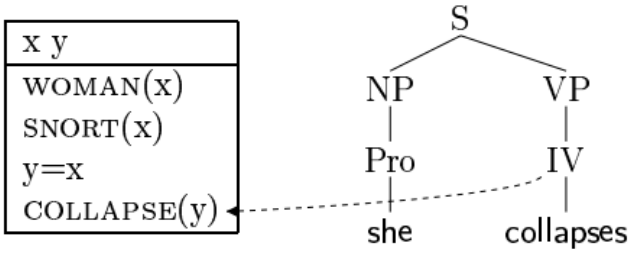
Jedan od načina rješavanja ovog problema je eliminiranje jedne od reprezentacija uz pomoć statističkih informacija izvađenih iz skupa podataka.

Reprezentiranje teksta logičkim formulama prvog reda i zaključivanje temeljem njih moguće je na prirodan način ostvariti u programskom jeziku Prolog.

Struktura reprezentacije diskursa

Reprezentacija nekoliko uzastopni rečenica često neće biti ispravna ako se radi kao konjunkcija logičkih formula svake zasebne rečenice. To je tako jer interpretacija svake rečenice ovisi o njenim prethodnicama, tj. o kontekstu u kojem je izračena. DRS struktura (engl. *Discourse Representation Structure*) predstavlja prirodan način za pamćenje tog konteksta. Postupak generiranja reprezentacije rečenice gradi tzv. *kutije* (engl. *box*) u koje zapisuje poznate entitete, informacije o njihovim atributima i načinu na koji su međusobno povezani. Stoga, kada naiđe na zamjenicu, iz konteksta prethodnih rečenica može pomoću unaprijed definiranih pravila odrediti na koji entitet se ta zamjenica odnosi (Eijck, 2005).

Primjer strukture koja odgovara paru rečenica "*A woman snorts. She collapses.*" prikazan je na slici 2.1. U zaglavlju *kutije* spremljene su dvije poznate varijable – x odgovara imenici *woman*, a y zamjenici *she*. Algoritam povezuje ta dva entiteta ($y = x$) jer prilikom nailaska na zamjenicu postoji samo jedan entitet na koju se ona



Slika 2.1: Primjer DRS strukture

može referencirati. Ideja slična ovom korištena je u modulu zaključivanja (opisanom u 3. poglavlju) za razrješavanje nepoznatih entiteta. Opisani primjer na engleskom je jeziku jer je preuzet iz knjige (Blackburn i Bos, 2005).

Cijela teorija iza reprezentacije DRS nastala je radi potrebe za rješavanjem problema anaforičkih zamjenica (zamjenice koje se referenciraju na prethodno spomenute entitete), glagolskih vremena i pretpostavki, odnosno implicitnog znanja izostavljenog iz rečenice. Jedna od glavnih prednosti ovog zapisa u odnosu na logiku prvog reda je prirodan način baratanja s tzv. *donkey* rečenicama (Geach, 1980), rečenicama koje sadrže *donkey* zamjenice – zamjenice semantički vezane za kvantificiranu frazu (npr. "svaki čovjek", "neki auti") ali sintaksno slobodne. Poznati primjer takve rečenice je "*Svaki seljak koji posjeduje magarca tuče ga.*" gdje problem nastaje pri rezoluciji osobne zamjenice *ga*.[1] Pri izgradnji logičke formule varijablu koju pridjeljujemo imenici "magarac" vežemo egzistencijalnim kvantifikatorom jer se u rečenici nalazi u akuzativu jednine. Dobivena logička forma $\forall x(seljak(x) \wedge \exists y(magarac(y) \wedge posjeduje(x,y))) \rightarrow tuce(x,y)$ netočna je jer ostavlja varijablu y slobodnu u posljednjem predikatu. Struktura reprezentacije diskursa problem vezanja zamjenice rješava uvođenjem novog *diskursnog referenta* za svaku imeničku frazu te svojstvom *dostupnosti* (engl. *availability*) tih referenata bez potrebe za vezanjem kvantifikatorima.

Grafovi i stabla

Grafovi su strukture pogodne za opisivanje odnosa entiteta, a stabla omogućuju jednostavan zapis hijerarhijske strukture između objekata. Primjer grafičkog zapisa čini Apstraktna reprezentacija značenja (engl. *Abstract meaning representation*) (Banarescu et al., 2013) kao jezik za opis semantičke reprezentacije koji svaku rečenicu zapisuje kao usmjereni aciklični graf. Jezik je svojevrsna apstrakcija sintaksnih reprezentacija na način da tekstne izraze istog značenja, a različitog izričaja, reprezentira identičnim

[1] Originalna rečenica na engleskom jeziku glasi "*Every farmer who owns a donkey beats it.*"

grafovima. Primjerice, sintagme “dječak je uništio loptu“ i “dječakovo uništenje lopte“ imat će jednake grafove formalnih reprezentacija.

Jezik DOL (engl.*DOlphin Language*) opisan u (Shi et al., 2015) primjer je jezika koji rečenice preslikava u stablastu strukturu čiji su čvorovi različitih tipova. Sastoji se od konstanti, razreda i funkcija, a za parsiranje koristi kontekstno neovisnu gramatiku nastojeći zadržati što više informacija iz ulaznog teksta. Konstante predstavljaju specifične objekte, npr. brojeve i znakovne nizove (“Taylorov red“), a razredi čine kategorije entiteta zajedničkih svojstava. Primjerice, svi geometrijski likovi obuhvaćeni su klasom `math.geometric_shape`, pa se za određene likove definira hijerarhija:

$$\texttt{math.rectangle} \subseteq \texttt{math.geometric\_shape}$$
$$\texttt{math.square} \subseteq \texttt{math.rectangle}$$

Posebna vrsta razreda jesu parametrizirani razredi slični parametriziranim klasama u jeziku C++. Primjerice:

$$\texttt{t.list<math.number, 2, } +\infty\texttt{>}$$

predstavlja listu od minimalno dva broja.

Funkcije služe za izgradnju stabla spajanjem više instanci razreda opisivanjem odnosa među njima. Svaka funkcija određena je imenom, tipovima parametara te vlastitim povratnim tipom. Jezik DOL razlikuje tri vrste funkcija: imenične, glagolske i modificirajuće funkcije. Imenične funkcije koriste se za predstavljanje imeničnih fraza, glagolske za reprezentaciju predikata, a modificirajuće kao dodatni argumenti imeničnih i glagolskih funkcija, koji mijenjaju njihovu semantiku. Posljednja vrsta funkcija najčešće zamjenjuje pridjeve, priloge i prijedloge. Tablica 2.1 sadrži nekoliko primjera definicija funkcija, njihovih gramatičkih pravila te semantičkih interpretacija.

Ovaj oblik formalne reprezentacije poslužio je kao inspiracija i vodilja za ovaj rad prvenstveno radi obrade slične tematike te ugrađenog mehanizma provjere tipova prilikom izgradnje semantičkog stabla.

2.1.2. Metode parsiranja

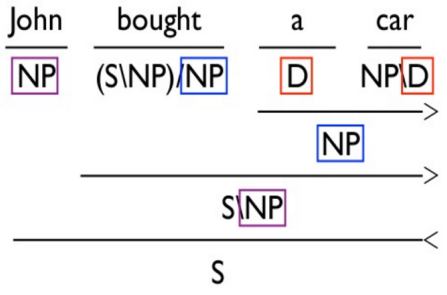
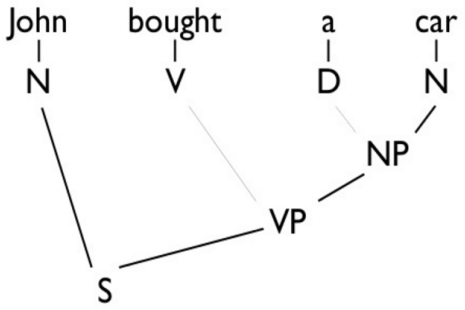
Najjednostavniji postupak semantičkog parsiranja čini označavanje objekata semantičkim ulogama (engl. *semantic role labeling*). Metoda određenim entitetima u tekstu pridjeljuje jednu od unaprijed definiranih oznaka ili ih uči statističkim postupcima iz podataka (Gildea i Jurafsky, 2002). Složeniji oblici parsiranja prevode rečenicu u njezinu logičku formu pomoću gramatike ili postupka podudaranja predložaka (engl. *template matching*).

nf.math.sum!1	
Definicija	nf.math.sum!1: $1:math.expression; $2:math.expression return type: math.expression
Gramatičko pravilo	nf.math.sum!1($1, $2) → $1 plus $2 | $1 added to $2
Interpretacija	nf.math.sum!1($1, $2) -> X($1) + X($2)

vf.be.equ	
Definicija	vf.be.equ: $1:quantity.generic; $2:quantity.generic return type: t.vf
Gramatičko pravilo	vf.be.equ($1, $2) → $1 be equal to $2 | $1 equal $2 | $1 be $2
Interpretacija	vf.be.equ($1, $2) -> X($1) = X($2)

mf.number.even	
Definicija	mf.number.even: return type: t.mf.adj
Gramatičko pravilo	mf.number.even → even
Interpretacija	mf.number.even -> X($↑) % 2 = 0

Tablica 2.1: Primjeri funkcija u jeziku DOL

CCG	CFG
	
John := *NP* *buy* := *(S\NP)/NP* *a* := *D* *car* := *NP\D*	$S \rightarrow NP\ VP$ $V \rightarrow buy$ $VP \rightarrow V\ NP$ $D \rightarrow a$ $NP \rightarrow D\ N$ $N \rightarrow John \mid car$ NP → N

Tablica 2.2: Usporedba gramatika CCG i CFG

Za preslikavanje teksta na logičke forme razvijene su metode nadziranog (Zettlemoyer i Collins, 2012; Kwiatkowski et al., 2010; Wong i Mooney, 2006) i polunadziranog učenja iz parova pitanja i odgovora (Berant et al., 2013; Berant i Liang, 2014). Većina statističkih postupaka uči preslikavanje teksta na logičke forme korištenjem gramatike CCG (engl. *Combinatory Categorical Grammar*). Gramatiku CCG čini leksikon parova riječi i fraza te njihovih semantičkih i sintaktičkih informacija, te univerzalna kombinacijska pravila za njihovu manipulaciju (kombiniranje). Ova je gramatika prikladna za semantičko parsiranje jer ima jasno mapiranje između sintaksne i semantičke reprezentacije entiteta. Svakoj riječi pridružena je kategorija – primitivna kategorija (*S, N, NP, PP* i sl.) ili funkcija koja definira redoslijed kombiniranja određenih tipova. Npr., funkcija *S/NP* prima argument tipa NP s desne strane i vraća objekt tipa S. Funkcije ujedno definiraju na koji se način generira tip duljeg znakovnog niza kombiniranjem tipova pojedinih riječi, što je zapravo direktna veza na njegovu logičku reprezentaciju. Tablica 2.2 prikazuje usporedbu parsiranja rečenice "*John bought a car.*" gramatikom CCG i gramatikom CFG kao i pripadne definicije (Choi, 2017). Iako djeluju vrlo slično, gramatika CCG je leksikalizirana (engl. *lexicalized*), što znači da koristi manji broj gramatičkih pravila, a sintaksne informacije za definiciju jezika određuju leksičke kategorije pojedinih riječi. Glavna pravila gramatike CCG čine primjene funkcija, kompozicijska pravila i proširivanje tipova (engl. *type raising*), čija je formala definicija dana u tablici 2.3 (Steedman, 2000).

Primjena funckija	$X/Y\ Y \rightarrow X$
	$Y\ X\backslash Y \rightarrow X$
Kompozicijsko pravilo	$X/Y\ Y/Z \rightarrow_B X/Z$
	$Y\backslash Z\ X\backslash Y \rightarrow_B X\backslash Z$
Proširivanje tipova	$X \rightarrow_T Y/(Y\backslash X)$
	$X \rightarrow_T Y\backslash(Y/X)$

Tablica 2.3: Pravila kombiniranja tipova gramatike CCG

Zanimljiv pristup semantičkom parsiranju čini uporaba modela prevođenja (Wong i Mooney, 2006, 2007; Andreas et al., 2013) gdje se formalna reprezentacija značenja teksta interpretira kao ciljni jezik. Za razliku od klasičnog modela prevođenja čiji izlaz ne mora u potpunosti odgovarati oznaci primjera u skupu podataka (jer ga interpretira čovjek), formalna reprezentacija teksta zahtjeva apsolutnu točnost. Stoga rezultate navedenih modela treba uzeti sa zadrškom jer postojeća evaluacijska metrika nije mjerodavna za procjenu njihove stvarne dobrote te skoro sigurno daje preoptimistične procjene.

U posljednje vrijeme pojavila su se istraživanja generiranja značenjske reprezentacije teksta uz pomoć neuronskih mreža (Jia i Liang, 2016; Dong i Lapata, 2016). Ovi modeli su u suštini također translacijski modeli, tj., uče poznate arhitekture za prevođenje (*seq2seq LSTM*) da prirodni jezik "prevode" u formalnu reprezentaciju. Očiti nedostatak ovih pristupa je nemogućnost dodavanja apriornog znanja u model te potreba za velikim količinama označenih podataka. Za rješavanje prvog problema (Jia i Liang, 2016) iz skupa za učenje induciraju generativni model pomoću kojeg uzorkuju nove primjere (*data recombination*) te na taj način model uče određena svojstva jezika (slično proširivanju skupa podataka (engl. *data augmentation*). Na primjer, ako iz originalnog upita "*Koje države graniče s Teksasom?*", *Teksas* zamijenimo nekom drugom državom, generiraju se primjeri pomoću kojih mreža može naučiti da je značenje ostatka rečenice neovisno o imenu države u pitanju (tj., model uči svojstvo uvjetne nezavisnosti). (Dong i Lapata, 2016) koriste klasičnu koder-dekoder povratnu mrežu s tim da dekoder na izlazu ne generira niz znakova već hijerarhijsku strukturu stabla (jer je logičku formu zapravo moguće prikazati kao stablo). Neuronske mreže privlače sve više pažnje radi jednostavne generalizacije parsera na različite jezike i domene primjene, no polunadzirano učenje (npr. iz parova pitanja i odgovora) još uvijek predstavlja velik izazov. Nedavna istraživanja (Liang et al., 2016; Guu et al., 2017)

pokazuju potencijal za učenje logičkih formi putem podržanog učenja (engl. *reinforcement learning*), ali ne dokumentiraju usporedbu s tradicionalnim modelima razvijenim na Freebase bazi podataka (Berant et al., 2013; Berant i Liang, 2014).

2.2. Zaključivanje

Postupak automatiziranog donošenja odluka i zaključaka iz logičke reprezentacije prirodnog jezika prvenstveno ovisi o odabiru te reprezentacije. Kako je ranije rečeno, zaključivanje na temelju logičkih formi provodi se jednom od metoda teorije dokaza (engl. *proof theory*). Za predikatnu logiku moguće je koristiti generaliziranu metodu rezolucije koja je posebno pogodna za automatizirano izvođenje na računalu.

Metoda rezolucije

Zaključivanje nad logičkim formama provodimo dokazivanjem zadovoljivosti izraza, odnosno dokazivanjem kontradiktornosti njegove negacije. Metoda rezolucije praktična je jer višestruko primjenjuje samo jedno pravilo logičkog zaključivanja – generalirizani modus ponens.

Pri dokazivanju tvrdnji logike prvog reda potrebno je ostvariti mehanizam za supstituciju vezanih varijabli. Varijable vezane univerzalnim kvantifikatorom moguće je zamijeniti bilo kojom poznatom konstantom tijekom provođenja rezolucije (unifikacija predikata), dok se konkretizacija egzistencijalnih varijabli mora prethodno provesti (kažemo da se izraz normalizira). Postupak je poznat pod nazivom *skolemizacija*, a provodi se uvođenjem novih, još nekorištenih konstanti (Bašić i Šnajder, 2008).

Rad metode demonstriran je na sljedećem primjeru, a čitatelja se upućuje na (Bašić i Šnajder, 2008) za više informacija o provedbi metode za generalniji slučaj.

Neka su nam poznate sljedeće dvije tvrdnje:

Ako je sustav inteligentan, onda razumije tekst.

Photomath ne razumije tekst.

i želimo dokazati istinitost tvrdnje

Photomath nije inteligentan sustav.

Prevođenjem rečenica u formalni zapis dobivamo sljedeće:

$$\forall x(inteligentan(x) \rightarrow razumije\_tekst(x))$$

$$\neg razumije\_tekst(Photomath)$$

$$\neg inteligentan(Photomath)$$

Sređivanjem sustava i negiranjem tvrdnje koju dokazujemo sustav je spreman za primjenu rezolucijskih pravila:

$$\neg inteligentan(x) \vee razumije\_tekst(x)$$
$$\neg razumije\_tekst(Photomath)$$
$$inteligentan(Photomath)$$

Rezolucijom prva dva izraza dobivamo sustav kontradiktornih klauzula:

$$\neg inteligentan(Photomath)$$
$$inteligentan(Photomath)$$

te zaključujemo da je izvorna tvrdnja o neinteligentnosti Photomatha istinita.

Postupak zaključivanja nad izrazima predikatne logike neće uvijek dati odgovor o istinitosti, odnosno neistinitosti tvrdnje koju dokazuje. Logika prvog reda je poluodlučiva, što znači da će postupak dokazivanja istinitosti tvrdnje završiti u konačnom vremenu samo ako je ona istinita, a u suprotnom postupak zaključivanja možda nikada neće generirati povratnu vrijednost.

Zaključivanje nad grafičkim strukturama

Ne postoji standardizirana metoda zaključivanja za grafove i stablaste reprezentacije. S obzirom da se radi o grafičkim strukturama, postupak nužno mora uključivati obilazak čvorova i njihove transformacije. Primjerice, interpretiranje stabla DOL kreće od njegovih listova te gradi matematičke izraze u *postorder* obilasku.

3. Semantička analiza matematičkih zadataka

Semantička analiza matematičkih zadataka iskazanih riječima specifičan je potproblem semantičke analize teksta. Domena jezika korištena u takvim zadacima je razmjerno uska te test zadatka standardno sadrži svega nekoliko rečenica. Problemi kao što su nejednoznačnost opsega (engl. *scope ambiguity*) te leksička pokrivenost iščezavaju jer se matematički zadaci u pravilu iskazuju jednostavnim frazama. Međutim, evaluacije postojećih modela ukazuju na visoku složenost zadatka jer trenutno najbolja (engl. *state of the art*) rješenja nisu niti blizu komercijalne uporabe (Huang et al., 2016).

Prvi pokušaji razvoja automatiziranog rješavača tekstnih zadataka uključuju interaktivni sustav STUDENT, koji matematički izraz gradi podudaranjem uzoraka nad transformiranim rečenicama (Bobrow, 1968), njemu sličan sustav CARPS koji uvodi stabla za internu reprezentaciju rečenica (Charniak, 1968) te program WORDPRO koji simulira psihološke procese koji se odvijaju u mozgu prilikom rješavanja jednostavnih aritmetičkih zadataka (Fletcher, 1985). Međutim, radovi koji opisuju te sustave ne navode njihove empirijske rezultate, pa je njihov potencijal upitan.

Sva aktualna istraživanja na ovu temu moguće je sistematizirati u dvije skupine: simboličke i statističke metode. Statistički pristupi opisani u (Kushman et al., 2014; Zhou et al., 2015) izvlače uzorke jednadžbi iz skupa podataka za učenje koje onda pokušavaju primijeniti na nove zadatke tijekom parsiranja. Hosseini et al. (2014) pak prvo u rečenici pronalaze glagol i klasificiraju ga u jednu od 7 kategorija (dodavanje, oduzimanje i sl.) na temelju koje određuju strukturu izlazne jednadžbe, dok Koncel-Kedziorski et al. (2015) treniraju diskriminativni model za generiranje stabla jednadžbi na skupu neoznačenih primjera (parovi zadatak-rješenje).

Testiranjem na većem datasetu Huang et al. (2016) pokazali su da postojeće statističke metode ne profitiraju od povećanja broja primjera dostupnih tijekom učenja, što je suprotno očekivanju. Također, usporedba postojećih modela teško je provediva jer

svaki pokriva specifični skup zadatka. Tako Hosseini et al. (2014) podržavaju samo zadatke koji se svode na zbrajanje i oduzimanje varijabli i brojeva, dok model opisan u (Koncel-Kedziorski et al., 2015) rješava i zadatke koji uključuju operacije množenja i dijeljenja, no samo one s jednom varijablom.

Simbolički pristupi također koriste kategorizaciju glagola i podudaranje uzoraka za preslikavanje rečenica na matematičke izraze, no bez pomoći strojnog učenja. Ove metode bile su popularnije u prvotnim istraživanjima, a od novijih radova ističe se već spomenuti pristup semantičkom parsiranjem opisan u (Shi et al., 2015). Iako je fokus rada na poprilično različitom skupu podataka u usporedbi s mojim, ovaj simbolički pristup pokazao je dobre empirijske rezultate zbog čega je poslužio kao glavna motivacija za izgradnju ovog sustava.

Skup podataka prema kojem je sustav razvijan opisan je u 3.1, dok je postupak parsiranja dan u 3.2, a metoda zaključivanja u 3.3 zajedno s detaljnom usporedbom s jezikom DOL.

3.1. Skup podataka

Cilj ovog rada jest ostvariti sustav koji za određeni podskup zadataka s riječima generira jednu ili vise naredbi rješavaču Photomatha. S obzirom na težinu općenitijeg zadatka rješavanja zadataka iz kojih je potrebno generirati sustav jednadžbi (Huang et al., 2016), u ovom radu fokus je stavljen na tekstne zadatke koji u sebi sadrže matematički izraz koji je potrebno riješiti te neke dodatne informacije o postupku njegovog rješavanja. Rješavač Photomatha često na ulazu dobije izraz za koji nije jasno što korisnik očekuje na izlazu. Primjerice, za izraz $2x^2 + 5x + 3$ nije jasno treba li se ulaz faktorizirati ili shvatiti kao funkcija i izračunati nultočke ili nacrtati njen graf. Osim toga, najjednostavnije linearne jednadžbe mogu ovisiti o nekoliko varijabli ili parametara, pa rješavač ne može samostalno znati po kojoj varijabli korisnik očekuje rješenje. Zadaci takvog tipa redovito su popraćeni kratkim tekstnim opisom koji uklanja problem dvosmislenosti, a Photomath podržava unos naredbi od strane programera za usmjeravanje toka rješavanja matematičkih izraza. Neke od trenutno podržanih naredbi su: *simplify_wrapper* koji označava da je ulazni izraz potrebno proslijediti direktno modulu istog imena, *eq_linear_solve, var* poziva modul za rješavanje linearnih jednadžbi po varijabli *var*, a *factorise_call* prosljeđuje ulaz na faktorizacijski modul.

Dakle, na izlazu semantičkog analizatora želimo generirati parove (E, A), gdje je E matematički izraz (engl. *Expression*), a A akcija – uputa rješavaču. Pri definiciji zadatka nije u potpunosti bilo jasno koje točno zadatke bi ovaj sustav trebao moći riješiti.

Zadatak	Akcije
The area of a trapezoid is $A = \frac{1}{2} * h * (b_1 + b_2)$. Solve for b_1.	$A = \frac{1}{2} * h * (b_1 + b_2)$, eq_solve, b_1
Factor $6x^2 - 216$.	$6x^2 - 216$, factorise_call
Which expression best represents the value of x in $y = mx + b$?	$y = mx + b$, eq_solve, x
What is the positive solution of $\|2x + 8\| = 19$?	$\|2x + 8\| = 19$, eq_solve, x {ARG0}, get_positive
What are the numbers $1.9, \frac{5}{4}, -1.2$ and $\sqrt{3}$ in order from greatest to least?	$list(1.9, \frac{5}{4}, -1.2, \sqrt{3})$, order_decreasing

Tablica 3.1: Primjeri zadataka

Iz knjiga (Saxon, 1998) i (Charles et al., 2011) izvađeno je 50 primjera na engleskom jeziku prema kojima je sustav oblikovan. Tablica 3.1 sadrži nekoliko primjera, a u dodatku A dan je popis svi zadataka kao i očekivani izlaz sustava. Odabrane zadatke karakterizira kratkoća – svi su sačinjeni od jedne do dvije rečenice. Također, svi u sebi sadrže matematički izraz, matricu ili listu brojeva (za zadatke koji traže usporedbu niza brojeva). Sustav je rađen za zadatke na engleskom jeziku radi veće iskoristivosti – većina korisnika Photomath aplikacije koristi englesku verziju, Uz to, za engleski jezik postoji nekoliko sakupljenih skupova podataka i referentnih radova za usporedbu pokrivenosti modela.

Čitatelju se možda može učiniti da je za ovako odabran skup podataka problem moguće riješiti jednostavnijim pristupom (vjerujem da bi obična detekcija ključnih riječi bila dovoljna). Opravdanje za odabranu metodu leži u modularnosti i ideji da se za jednom napravljen sustav, unosom novih komponenti te dodatnog znanja kroz konstrukte jezika SML, njegova domena može proširiti na kompleksnije zadatke. Dodatna vrijednost koja se može ostvariti na ovaj način je mogućnost objašnjavanja pojedinih koraka zaključivanja korisniku, što jednostavniji modeli ne mogu.

Pretprocesiranje

Tekst zadatka potrebno je pripremiti za parsiranje. Zadatak se dijeli na rečenice te se svaka rečenica tokenizira. Tokeni se lematiziraju i određuju im se POS (engl. *part of speech*) i NER (engl. *named entity recognition*) oznake koje parser koristi za određivanje vrste pojedinih tokena. Za obradu teksta korišteni su alati razvijeni na sveučilištu

Stanford.[1] Matematički izrazi koji se nalaze u tekstu moraju biti obilježeni znakovima "`$$`" s obje strane, te im je nakon obrade rečenica za vrijednost leme potrebno postaviti znakovni niz prefiksiran s "*math_expression*".

3.2. Formalna reprezentacija matematičkih zadataka

Kao što je ranije rečeno, za generiranje formalnih reprezentacija značenja ulaza razvijen je poseban jezik SML.[2][3] Za parsiranje teksta zadataka napisana je kontekstnoneovisna gramatika (Chomsky, 1956) pomoću koje se Earleyjevim algoritmom parsiranja (Earley, 1970) gradi značenjska reprezentacija rečenica.

Jezik SML sastoji se od tri komponente: tipova, funkcija te gramatičkih pravila. Svi tipovi i gramatička pravila definiraju su u tekstnim datotekama i dinamički se učitavaju prilikom izvršavanja programa. U nastavku su detaljno opisane karakteristike jezika – tipovi i pravila za izgradnju semantičkih stabala.[4]

3.2.1. Tipovi

Rečenica prirodnog jezika se jezikom SML prikazuje kao stablo čiji je svaki čvor odreden tipom. Postoje dvije vrste tipova: razredi i funkcije. Krajnji čvorovi stabla – čvorovi bez djece – uvijek su razredi, dok je tip unutrašnjeg čvora uvijek funkcija.

Razredi definiraju sve entitete koji se mogu naći u tekstu (različiti objekti, pridjevi, prilozi i sl.). Slično programskim jezicima, razrede je moguće nasljeđivati, a podržano je i višestruko nasljeđivanje. Na primjer, definicija klase `math.centroid` kao potklasi razreda `math.point` i `class.attribute` dana je s:

```
math.centroid : math.point & class.attribute.
```

Funkcije su zamišljene kao tip koji povezuje dijelove rečenice u hijerarhijsku strukturu. Svaka je funkcija strogo tipizirana što znači da je definirana imenom, brojem parametara i njihovim tipovima te vlastitim povratnim tipom. Primjerice, definicija funkcije koja semantički predstavlja potenciranje dva broja može se zapisati na sljedeći način:

```
math.power($1:math.number, $2:math.number) : math.number
```

[1] https://nlp.stanford.edu/software/

[2] Sustav za učitavanje jezika iz definicije gramatike i parsiranje ulaznog teksta razvio je Leon Rotim.

[3] SML jezik nije povezan s funkcijskim jezikom Standard ML.

[4] Implementaciju mogućnosti definicije jezika napisao je Leon Rotim u jeziku C++, a definicija konkretne gramatike ostvarena je samostalno.

Funkcija imena `math.power` prima dva argumenta, oba tipa `math.number` te vraća instancu istog tipa. Prefiks *"math"* u imenu tipova ne omogućuje nikakvu posebnu funkcionalnost već se koristi za sistematizaciju kao pomoć programeru.

Mehanizmom nasljeđivanja klasa omogućena je apstrakcija funkcija. Ako definiramo hijerarhiju nekoliko razreda na sljedeći način:

```
math.expression
math.geometric_shape : math.expression
math.triangle : math.geometric_shape
math.circle : math.geometric_shape
math.ellipse : math.geometric_shape
math.rectangle : math.geometric_shape
math.polygon : math.geometric_shape

math.point : math.geometric_shape
math.curve : math.geometric_shape
math.line : math.curve
math.arc : math.curve
math.asymptote : math.line
math.hyperbola: math.curve
math.parabola : math.curve
math.inflection_point : math.point
```

onda je primjerice funkciju za povezivanje bilo koje krivulje ili geometrijskog lika i njegove formule moguće definirati kao:

```
math.shape_formula($1 : math.geometric_shape,
                   $2 : math.expression_formula) :
                   math.expression_formula
```

Zbog hijerarhije nasljeđivanja funkcionalnost je ostvarena samo jednom funkcijom, dakle eliminirano je višestruko kopiranje za svaki pojedini tip. Još jedan mehanizam koji smanjuje potrebu za kopiranjem semantički ekvivalentnih funkcija jesu *parametrizirani razredi* (engl. *template classes*) – mehanizam sličan parametriziranim klasama u programskim jezicima. Parametrizirani razred definiran je imenom te trima parametrima: tipom čvora koji enkapsulira te minimalnim i maksimalnim brojem čvorova tog tipa. Jedini trenutno postojeći parametrizirani tip u SML jeziku je *lista* definirana s

`list<T, minarg, maxarg>`. Ovaj razred služi za povezivanje nekoliko objekata istog tipa u jednu cjelinu što između ostalog omogućuje pisanje generalnih funkcija. Npr. funkcija zbrajanja definirana kao:

```
math.sum($1:math.expression, $2:math.expression) :
                                        math.expression
math.sum2($1:list<math.expression, 2, inf>) :
                                        math.expression
```

pokriti će sve slučajeve zbrajanja brojeva bez obzira na broj pribrojnika. Parametriziranom klasom je stoga eliminirana potreba za definiranjem funkcije zbrajanja za svaki mogući broj operanada (primjer preuzet iz (Shi et al., 2015)).

Kao što je ranije rečeno, funkcije služe za povezivanje čvorova i izgradnju semantičkog stabla. Ideja je parsiranjem sačuvati što više informacija iz ulaznog teksta i odgoditi odluku o relevantnosti podataka kasnijem stadiju zaključivanja. Radi toga se ne odbacuju članovi, zamjenice (npr. čuvaju se i upitne zamjenice) i ostali determinatori (ili odrednice, engl. *determiners*). Za parsiranje imeničnih fraza koje ih sadrže koriste se tzv. *funkcije omotači* definirane u datoteci `wrapper_fun.txt`. Ove funkcije imaju posebnu semantiku koja nije odmah jasna iz njihove definicije. Primjerice, funkcija `wf.this():_;` naizgled ne prima niti jedan argument i nema definiranu povratnu vrijednost. Ovakva sintaksa koristi se za definiciju funkcija koje samo "prenose" tip parametra koji primaju – dakle, primaju jedan argument bilo kojeg tipa i njegov tip vraćaju kao povratnu vrijednost. Ukoliko se za određeni razred želi onemogućiti omatanje ovim funkcijama, potrebno je pri njegovoj definiciji dodati oznaku "`@W`". Na primjer, naredba

```
gen.adj @W
```

definira razred pridjeva. S obzirom da se pridjev može dodati ispred svake imenične fraze potreban je mehanizam koji se s tim zna nositi. Sustav koji zna razriješiti rečenicu "*Zbroji brojeve 3 i 5.*", a za matematički izraz ekvivalentna rečenica "*Zbroji plave brojeve 3 i 5.*" nailazi na probleme, nije previše koristan. Ovaj problem riješen je upravo prethodno definiranim razredom `gen.adj` i njegovom posebnom funkcijom omotačem `wf.adj()`. Za pridjeve čija nam je vrijednost zanimljiva i/ili utječe na izgradnju semantičkog stabla definiramo posebne funkcije (u datoteci `adjectives.txt`).

Pri definiranju funkcije istoj je moguće odrediti prioritet. Prioriteti se koriste prilikom parsiranja za donošenje odluke koje stablo – od nekoliko sintaksno ispravnih generiranih stabala – proslijediti modulu zaključivanja. Oznaka prioriteta navodi se nakon povratnog tipa funkcije odvojeno točka-zarezom:

```
adj.positive() : _ ; 5.
```

Ako nije naveden, podrazumijeva se prioritet iznosa 10, a veći broj predstavlja viši prioritet.

3.2.2. Gramatika

Semantičko stablo rečenice prethodno definiranih tipova gradi se prema pravilima kontekstno-neovisne gramatike. Pravila dakle s lijeve strane imaju jedan "znak" – u ovom slučaju tip čvora.

Pravila razreda

Pravila za generiranje čvorova razreda najčešće su jednostavnog oblika jer klase opisuju entitete iz stvarnog svijeta. Primjerice, pravila vezana za geometrijske likove iz gornjeg primjera zapisujemo na sljedeći način:

```
math.trianlge -> triangle_L
math.circle -> circle_L
```

gdje sufiks *_L* označava lematizirani oblik riječi. Prvo pravilo će za svaku riječ čija je lema *triangle* (*triangle, triangles, Triangle i sl.*) stvarati čvor tipa `math.triangle`.

Sva postojeća pravila generirana su ručno, no jasno je da se pisanje pravila za definiciju razreda može lako automatizirati.

Pravila funkcija

Gramatička pravila pridružena funkcijama kompleksnija su utoliko što moraju omogućavati povezivanje ulaznih tokena i argumenata funkcija. Pravilo "pali" samo ako referencirani tokeni tipom odgovaraju deklariranim argumentima. U nastavku je dano nekoliko primjera opisanih pravila:

```
math.calculate($1) -> calculate_L {$1} | evaluate_L {$1}

vf.be.equ($1, $2) -> {$1} be_L equal to {$2}
                   | {$1} equal to {$2}
                   | {$1} be_L {$2}
                   | {$1} be_L given by {$2}
                   | {$1} be_L given as {$2}.
```

Znak "|" (logičko *ILI*) koristi se za kombiniranje više pravila u jedno. Prvo pravilo iz gornjeg primjera može se na ekvivalentan način zapisati na sljedeći način:

```
math.calculate($1) -> calculate_L {$1}
math.calculate($1) -> evaluate_L {$1},
```

odnosno, ukoliko se za znakovne nizove koristi izvorni tekst umjesto lema (funkcionalnost trenutno nije dodana u parser):

```
math.calculate($1) -> [calculate | evaluate] {$1}.
```

Funkcije bez argumenata (tj. s jednim argumentom neodređenog tipa) koriste posebnu sintaksu pravila. Na primjer, za funkciju koja bilo koji čvor "omotava" pridjevom *positive* definiramo pravilo:

```
adj.positive($1 : _) -> positive_L {$1}.
```

Prioritet funkcije definira se samo na jednom mjestu, dakle ne navodi se ponovno u gramatičkim pravilima.

Stroga tipiziranost funkcija djelomično rješava težak problem semantičkog parsiranja kojeg karakterizira nejednoznačno sintaktičko parsiranje rečenice. Problem nastaje jer samo jedan rezultat semantički ima smisla, a pravila gramatike generiraju nekoliko sintaktički ispravnih stabala te parser ne može odrediti (bez nekog dodatnog znanja) koji je ispravan. Definiranje funkcija nad specifičnim tipovima eliminira ovaj problem jer znamo točno koji ulazni niz povezujemo u podstablo unutar rečenice. Medutim, to još uvijek ne znači da na izlazu parsera dobivamo jedno stablo – nego samo da je svako generirano stablo sukladno definiranom jeziku.

Prepoznavanje *tokena*

U matematičkim zadacima često se pojavljuju brojevne konstante i matematički izrazi. Za njihovo parsiranje napisan je poseban mehanizam podudaranja tokena (engl. *token matcher*). U datoteci `SMTokenMatcherBuilder.cpp` navedeni su svi C++ razredi za podudaranje tokena te tekstne oznake za njihovo korištenje u gramatičkim pravilima. Primjerice, determinatore će u tekstu prepoznati `DeterminerMatcher` provjerom POS (engl. *Part Of Speech*) oznake ulaznog tokena. Također, jedna od neophodnih klasa je `MathExpressionMatcher`, koja prepoznaje matematički izraz unutar teksta na temelju oznake leme (u pretprocesiranju rečenica potrebno je za tokene matematičkih formula kao lemu postaviti znakovni niz prefiksa *"math_expression"*).

Referenciranje na određene *token matchere* u pravilima vrši se prefiksiranjem ključne riječi klase znakom '`#`'. Na primjer, pravilo koje stvara čvor tipa `gen.adj` dano je s:

```
gen.adj -> #ADJ | #ADJ {$gen.adj}.
```

Oznaka `#ADJ` efektivno zamjenjuje sve pridjeve koje prihvaća `SMAdjectiveMatcher`. Drugi dio pravila rekurzivne strukture omogućuje parsiranje niza uzastopnih pridjeva. U nastavku je dano još nekoliko primjera korištenja opisanog mehanizma:

```
t.determiner -> #DT
math.number -> #NUMCONST | #NUMBER
math.expression_formula -> #MATHEXPR
```

Korisnik na jednostavan način može dodati novo prepoznavanje posebnih tokena implementacijom apstraktne klase `TokenMatcher`, odnosno njegove metode `bool match(const CFG::Token& token)`, koja za uspješno podudaranje ulaznog argumenta vraća vrijednost `true`, a u suprotnom vrijednost `false`.

Odabir jedinstvenog semantičkog zapisa

Algoritam zaključivanja nad stablom opisan u 3.3 dovoljno je kompleksan i ako barata samo jednim stablom. Zbog toga je odlučeno heuristički odabrati jedan izlaz parsera, a ostale odbaciti. Odabir se vrši rangiranjem prema formuli:

$$score(T) = \frac{\sum_{i=1}^{k} len(T_i) \cdot score(T_i)}{\sum_{i=1}^{k} len(T_i)}$$

gdje je T stablo čiju dobrotu računamo, T_i njegovo podstablo, a $len(T_i)$ duljina ulaznog niza tokena kojeg to podstablo obuhvaća. Rangiranje stabala preuzeto je iz (Shi et al., 2015).

3.2.3. Usporedba sa semantičkim jezikom DOL

Većina jezičnih konstrukata preuzeta je iz (Shi et al., 2015) uz poneka pojednostavljenja i modifikacije. Dok jezik DOL sadrži tri tipa čvorova (konstante, razrede i funkcije), jezik SML ne koristi konstante, već su svi objekti omotani u razrede. Između razreda moguće je definirati hijerarhijsku strukturu, a funkcije su strogo tipizirane. DOL podržava postojanje istoimenih funkcija s različitim ulaznim parametrima (engl. *function overloading*), što je izbačeno u jeziku SML jer nije pretjerano korisno, a stvara probleme modulu zaključivanja. Kroz pisanje pravila došla sam do zaključka da većina funkcija ima samo jednu definiciju, a one koje je moguće definirati za različiti skup parametara zahtijevaju tek dvije ili tri definicije. Nadalje, u jeziku DOL postoje tri vrste funkcija: imenične, glagolske i modifikacijske. Svaka vrsta ima posebnu

primjenu i semantiku. Jezik SML definira jedinstven tip funkcija. Za modifikacijske funkcije spomenute u izvornom radu nije dano dovoljno informacija o načinu na koji se one parsiraju, pa su one izostavljene u jeziku SML. Direktorij `functions` sadrži datoteku `modifiers.txt` što bi moglo zbuniti čitatelja. Funkcije definirane tamo najobičnije su funkcije izdvojene u zasebnu datoteku zbog svoje posebne semantike. Također, prilikom parsiranja ne radi se nikakvo zaključivanje niti povezivanje entiteta i razrješavanje anaforičnih izraza. DOL entitetima dodjeljuje oznake varijable, dok taj korak ne postoji u ovom sustavu. Konačno, DOL sadrži puno više razreda i funkcija jer je cilj autora bio osmisliti jezik otvorene domene. Jeziku SML moguće je dodati tipove ručno ili automatizirano bez previše problema, međutim to nije napravljeno jer u ovom stadiju razvoja to nije bio prioritet.

3.3. Generiranje akcija za matematički rješavač

Modul zaključivanja ili razrješavač (engl. *resolver*) implementiran je kao zasebna cjelina odvojena od parsera. Semantičku reprezentaciju koju generira parser rješavač prvo prilagođava sebi za lakše korištenje – ulazno stablo čiji su čvorovi tipa `SMNode` prevodi u instance `SMRInputNode` klase te im pridružuje korisnički definirane transformacije. Ovaj korak može proširiti svoju funkcionalnost, pa npr. razrješavač već tu može donijeti odluku o odbacivanju čvorova nekog tipa koji se smatra nebitnim za generiranje akcija Photomath rješavaču. Konkretno, svi čvorovi tipa `gen.adj` odbacuju se prilikom prevođenja ulaznog stabla jer su semantički nebitni.

Modul zaključivanja prevodi ulazno stablo (čvorovi tipa `SMRInputNode`) u izlazno stablo tipa `SMRNode`. Svaki objekt tipa `SMRNode` ima svoj tip (enumeracija `SMRNodeType`), znakovni niz iz ulaznog teksta kojem je pridružen, a može imati djecu i atribute. Atributi su također instance razreda `SMRNode`, a služe za pobliže opisivanje roditeljskog čvora. Primjerice, čvor tipa *SMR_FORMULA_NODE* koji predstavlja matematički izraz iz originalnog teksta zadatka kao atribut ima pridruženo stablo tog izraza. Formulu izraza ne dodajemo vršnom čvoru kao dijete jer ono to semantički nije – u suprotnom nastaje problem u algoritmu prolaska kroz stablo jer se formula ne gleda kao cjelina (što ona jest), već se ulazi u elemente izraza i pokušava ih se zasebno razriješiti.

Akcije

Krajnji izlaz modula zaključivanja je lista stabala čiji su čvorovi instance `SMRNode` klase posebnog *akcijskog* tipa. Podržani tipovi `SMRNode` objekata popisani su u datoteci `SMRNodeType.hpp`, a tzv. *akcijski* tipovi su oni tipovi iz kojih se može generirati ulaz za Photomath rješavač. U trenutku pisanja ovog rada Photomath prepoznaje svega nekoliko iskoristivih naredbi: faktorizaciju (*factorise_call*), pojednostavljivanje izraza (*simplify_wrapper*) te rješavanje linearne jednadžbe po specificiranoj varijabli (*eq_linear_solve*). Njima su redom pridružene sljedeće vrijednosti `SMRNodeType` enumeracije : *SMR_FACTORISE_NODE*, *SMR_SIMPLIFY_NODE* i *SMR_SOLVE*.

Bez obzira na to, modulu zaključivanja moguće je dodati dodatne tipove radi semantičkog razrješavanja pojedinih zadataka, no za njih kompletni sustav neće generirati postupak rješavanja.

Akcije se generiraju kroz proces transformacije ulaznih stabala. U nastavku su opisane pojedine komponente sustava te algoritam prolaska po stablu.

Transformacije

Prevođenje iz ulaznog u izlazni tip stabla ostvareno je kroz mehanizam transformacija koje se definiraju u tekstualnim datotekama, slično gramatičkim pravilima jezika SML. Transformaciju čine kreator i niz uvjeta, a sintaksa za njeno definiranje je sljedeća:

```
tip(a, b, ...) : kreator(...)
        if uvjet(...)
```

gdje "..." označava varijable transformacije, odnosno djecu čvora tipa *tip*. Referenciranje djece vrši se preko jednoslovnih varijabli (za razliku od sintakse jezika SML) jer daje jednostavniji zapis. Za pojedini tip moguće je definirati više transformacija, a prioritet pri izvršavanju ima ona koja je navedena prije u datoteci. S obzirom da većina razreda jezika SML predstavlja entitet iz stvarnog svijeta (pridruženi su imenicama, formulama ili brojevima) za njih nije potrebno pisati transformacije već im se implicitno pridružuje transformacija koja stvara objekt tipa *SMR_ENTITY_NODE* i sprema ga u kontekst. Funkcije međutim moraju imati pridruženu transformaciju jer su kompleksnije strukture i imaju posebnu semantiku.

Kontekst

Kontekst je centralni objekt algoritma zaključivanja implementiran u datoteci `SMRContext.cpp`. Njime se prenose informacije o trenutnim varijablama i stvorenim entite-

tima. Pamćenje entiteta važno je jer njihovo navođenje u tekstu nije uvijek u redoslijedu koji želimo te je ponekad potrebno referencirati se na neki prethodni objekt kako bi se ispravno razlučilo značenje zadatka. Lista entiteta se tijekom izvođenja konstantno nadopunjuje (bez brisanja), dok se lista varijabli stalno ažurira – vrijednosti varijabli konteksta prilikom obrade određenog čvora odgovaraju stablima generiranim iz njegove djece (u ispravnom redoslijedu).

Kreatori

Ideja za izgradnju akcija korištenjem kreatora preuzeta je iz rješavača Photomath. Photomath koristi kreatore za modifikaciju ulaznog stabla dok ovaj sustav gradi izlazno stablo iz temelja po uzoru na ulazno. Isti kreatori mogu se koristiti u različitim transformacijama, što značajno reducira potrebu za kopiranjem slične funkcionalnosti za različite tipove ulaznih čvorova. Dodavanje novih kreatora u jezik radi se implementacijom apstraktne klase `SMRCreator`, odnosno njezine metode `create`.

Općenito, kreatore možemo podijeliti u dvije skupine: jednostavni i složeni. Jednostavni su oni koji omogućavaju referenciranje varijabli transformacije, njihove djece ili atributa te stvaranje čvorova nekog tipa. Kôd za njihovo generiranje nalazi se u datoteci `SMRCreatorFactory.cpp`. Složeni kreatori su kompleksniji samo po svojoj semantici – ti kreatori obavljaju neki dodatni posao. Primjerice, kreiraju stablo matematičkog izraza ili spremaju entitet u kontekst razrješavača. Popis svih implementiranih kreatora nalazi se u datoteci `SMRComplexCreatorFactory.cpp` kao i znakovni nizovi kojima ih se referencira u definicijama transformacija. Primjeri nekih složenih kreatora su `add_entity(a)`, koji sprema čvor `a` u listu entiteta trenutnog konteksta, `tail(a)`, koji vraća posljednje dijete čvora `a` te `create_formula()`, koji pomoću Photomath-a parsira ulazni niz koji sadrži matematički izraz, prevodi ga u stablo `SMRNode` tipa i postavlja kao atribut vršnom čvoru.

U nastavku je dano nekoliko definicija transformacija uz pojašnjenje korištene sintakse.

```
adj.greatest(a) : max(a)
how_many(a) : count(a)
```

Prvo pravilo definira da funkcija `adj.greatest` uvijek generira čvor tipa *SMR_MAX_NODE* čije je jedino dijete transformirano dijete ulaznog čvora. Slično, drugo pravilo stvara čvor *akcijskog* tipa *SMR_COUNT_NODE* uz prosljeđivanje djeteta s ulaza.

```
vf.find(a) : a
adj.last(a) : tail(a)
```

Pravilo pridruženo tipu `vf.find` ne stvara nikakav novi čvor, već samo prosljeđuje već generirano dijete ulaznog čvora. Pretpostavka ove transformacije je da je parametar te funkcije već sam po sebi akcija za matematički rješavač. Drugo pravilo koristi složeni kreator `tail` čija funkcionalnost semantički odgovara značenju funkcije `adj.last` (u prijevodu, "posljednji").

```
adv.increasing : ``increasing``
adj.lateral(a) : modify_entity(a, ``lateral``)
```

Pomoću kreatora atributa (`SMRAttributeCreator`) moguće je kreirati objekte tipa *SMR_ATTRIBUTE_NODE* iz konstantnih znakovnih nizova. Semantika tih čvorova je posebna – oni imaju značenje tek kada su pridruženi nekom drugom čvoru kao njegov atribut. Kreator `modify_entity` zamišljen je kao generalniji modifikator ulaznog objekta, no trenutno podržava samo dodavanje atributa.

```
nf.list(a) : add_entity(ttt_list__system(a))
        if is_formula(a[0])
        if is_relation(a[0]{0})
```

Ovo pravilo transformacije sadrži dva uvjeta provjere argumenta. Oznakom `a[0]` dohvaća se prvo dijete objekta `a`, a oznakom `a[0]{0}` prvi atribut tog djeteta. Dakle, za referenciranje djece koriste se uglate, a za dohvat atributa vitičaste zagrade (indeksiranje počinje od nule). Također, na ovom primjeru možemo vidjeti kako je moguće ostvariti ulančavanje kreatora. Prvo se izvodi složeni kreator `ttt_list_system` koji mijenja tip objekta `a` iz *SMR_LIST_NODE* u *SMR_SYSTEM_NODE* ("ttt" je skraćeno za *type to type*). Izlaz tog kreatora prosljeđuje se na ulaz drugom složenom kreatoru `add_entity` koji ga zapisuje u listu entiteta globalnog konteksta.

```
math.line_through_points(a, b) :
                    eq(frac(sub(y, a[1]), sub(b[1], a[1])),
                    frac(sub(x, a[0]), sub(b[0], a[0])))
```

Na ovom primjeru možemo vidjeti kao je moguće generirati formulu za jednadžbu pravca kroz dvije točke `a` i `b`. Trenutno je podržana samo prefiksna sintaksa definiranja tipova, a za varijable je moguće koristiti samo znakove `x`, `y` i `z`.

```
math.solve_for(a) : solve(_relation, a)
```

Posljednje transformacijsko pravilo demonstrira mehanizam korištenja nepoznatih entiteta. Ulazna funkcija za argument prima varijablu po kojoj je potrebno riješiti neki (trenutno nepoznati) izraz. Zapis `_relation` označava da prvi argument čvora tipa

SMR_SOLVE_NODE treba biti neki čvor relacijskog tipa (*SMR_EQUALS_NODE*, *SMR_LT_NODE*, i sl.). Ovo pravilo stvara čvor *SMR_UNKNOWN_ENTITY_NODE*, koji algoritam zaključivanja u nekom trenu mora zamijeniti konkretnim entitetom.

Uvjeti izvođenja

Uvjeti kod definicije transformacije navode se nakon ključne riječi "`if`" , a služe za ograničavanje njenog izvršavanja, odnosno omogućuju postojanje većeg broja različitih transformacija za čvorove istog tipa. Svi postojeći uvjeti popisani su u datoteci `SMRConstraintFactory.cpp`. Trenutno su podržani samo uvjeti koji provjeravaju podudaranje tipa ulaznog čvora. Uvjet oblika `is_entity_[SM_type]` provjerava odgovara li tip predanog argumenta tipu *SMR_ENTITY_NODE* čija je vrijednost znakovnog niza (varijabla `strVal_`) jednaka *SM_type*. Ograničenje definirano s `is_[type]` radi jednostavnu provjeru tipa argumenta, a ograničenje oblika `is_unknown_[type]` provjerava je li argument tipa *SMR_UNKNOWN_ENTITY* i njegovo prvo dijete tipa *type*.

Uz ovaj oblik ograničenja, pojedini kreatori mogu definirati vlastite uvjete. Primjerice, kreatori koji pristupaju nekom djetetu prosljeđenog argumenta moraju provjeriti da to dijete stvarno postoji. Jedan od načina dojave greške bio bi vratiti `NULL` vrijednost pri izlasku iz `create` metode, no provjera uvjeta neposredno prije izvršavanja kreatora čišći je način obrane od rušenja sustava jer ne zahtijeva nikakve posebne provjere povratne vrijednosti kreatorskih funkcija. Primjer definiranja ovakvih uvjeta izvođenja može se vidjeti u datoteci `SMRChildCreator.cpp`.

Algoritam zaključivanja

Pseudokod na slici 3.1 opisuje glavnu ideju generiranja akcija. Algoritam rekurzivno prolazi stablom te iz ulaznih čvorova tipa `SMRInputNode` stvara čvorove tipa `SMRNode`. Generiranje izlaznog stabla počinje od vršnih čvorova (razreda u SML jeziku) i gradi se prema gore. Nakon što razriješi cijelo podstablo trenutnog čvora, algoritam ažurira stanje konteksta te pronalazi prvo transformacijsko pravilo koje se može izvršiti – ono pravilo čije uvjete zadovoljava trenutno stanje konteksta. Kreator pronađene transformacije kreira objekt tipa `SMRNode`. Ako je novonastali objekt *akcijskog* tipa, algoritam provjerava postoji li u podstablu objekta čvor nepoznatog entiteta (*SMR_UNKNOWN_ENTITY_NODE*). U jednostavnijem slučaju kada su svi entiteti stabla poznati objekti, pamtimo ga kao izlaznu akciju modula. Ako pak postoje nerazriješeni čvorovi, traži se poznati entitet u kontekstu podudarajućeg tipa te ukoliko

```
razrijesi(stablo, kontekst):
    za svako dijete d : stablo:
        v = razrijesi(d)
        varijable.dodaj(v)

    azuriraj varijable konteksta
    za svaku transformaciju t : stablo.transformacije():
        ako (t.uvjeti_zadovoljeni()):
            cvor = t.izgradi(kontekst)
        ako (cvor.tip == akcija):
            ako (!razrijesen(cvor)):
                cvor = razrijesi_nepoznate_entitete(cvor, kontekst)
            ako (razrijesen(cvor)):
                azuriraj akcije konteksta

        vrati cvor
```

Slika 3.1: Pseudokod algoritma generiranja akcija

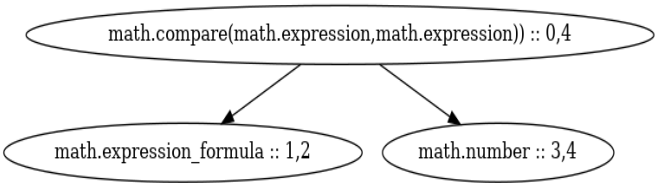
on postoji vrši se supstitucija dvaju čvorova i trenutni čvor ponovno spremamo u listu akcija. U slučaju da ne uspijemo pronaći supstituirajući entitet, trenutni čvor pamtimo u listi nerazriješenih akcija te ga još jednom pokušavamo razriješiti na kraju prolaska kroz cijelo stablo.

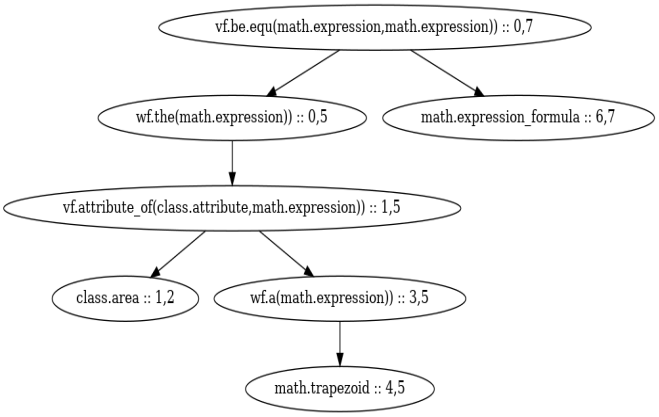
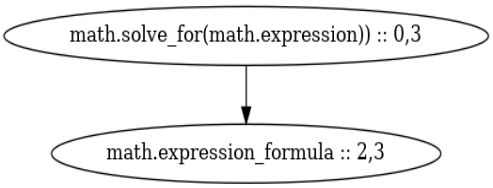
3.3.1. Usporedba sa semantičkim jezikom DOL

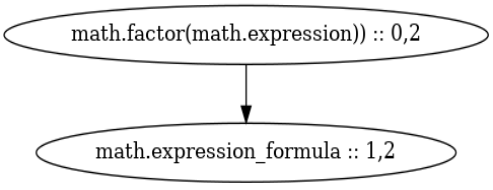
Slično algoritmu opisanom u originalnom radu, ovaj algoritam gradi izlazno stablo odozdo prema gore. Jedna bitna razlika jest cilj koji se generira – DOL modul zaključivanja gradi stablo jednadžbe, a upravo opisani sustav pronalazi uputu za rješavanje već poznatog matematičkog izraza. Slično originalnom radu koji za izlaz modula pamti sve generirane jednadžbe i nejednadžbe, ovaj sustav pamti izraze čiji tip može biti akcija za izvršavanje u rješavaču Photomatha. Autori jezika DOL ostavili su nejasnim na koji način primjenjuju semantičku interpretaciju funkcija. Primjeri koji su opisani u radu su trivijalni i ostaje otvoreno pitanje na koji način obrađuju funkcije čija interpretacija nije u potpunosti jasna (većina funkcija je ovakve prirode jer se radi o obradi teksta).

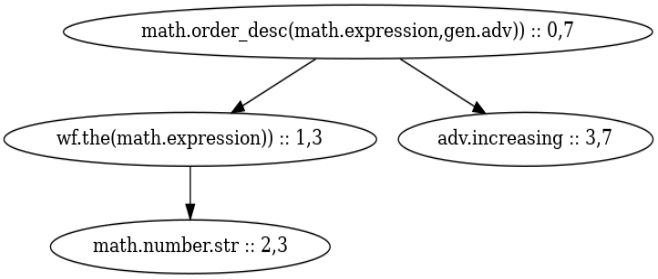
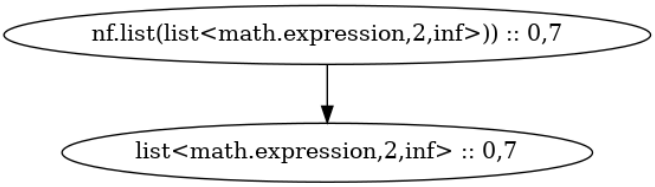
4. Rezultati

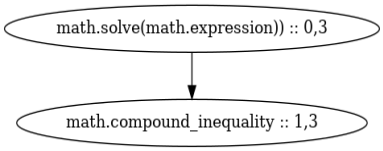
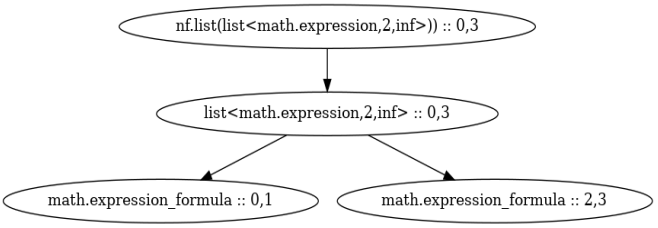
Kao što je ranije rečeno, sustav je oblikovan prema skupu od samo 50 primjera koji nije podržan u cijelosti. Razvoj gramatike i modula zaključivanja pokazao se dosta velikim pothvatom – što zbog konceptualnih problema osmišljavanja prikaza rezultata i njegovog generiranja, što zbog implementacijskih problema. Također, skup podataka trebao je biti mnogo veći i sakupljen od strane Photomath tima, no označavanje podataka za ovaj zadatak do sada još nije provedeno. Imajući to na umu, odustalo se prvotne namjere razvijanja kompletnog sustava i testiranja na ispitnom skupu – što bi dalo neko mjeru dobrote ovog rada – već je fokus stavljen na razvoj funkcionalnosti jezika i omogućavanju daljnjeg unosa znanja za pokrivanje šireg spektra zadataka. Rezultat rada zapravo je već opisan u prethodnom poglavlju, a ovdje navodim nekoliko primjera zadataka za koje je uneseno znanje i koje semantički parser ispravno interpretira. Za svaki zadatak dan je njegov tekstni opis, pripadna semantička reprezentacija (izlaz parsera) te izlaz modula zaključivanja u tekstualnom obliku (originalni izraz i akcija za Photomath). Čvorovi semantičkog stabla prikazuju svoj tip i indekse prvog i posljednjeg tokena u izvornoj rečenici koji obuhvaćaju.

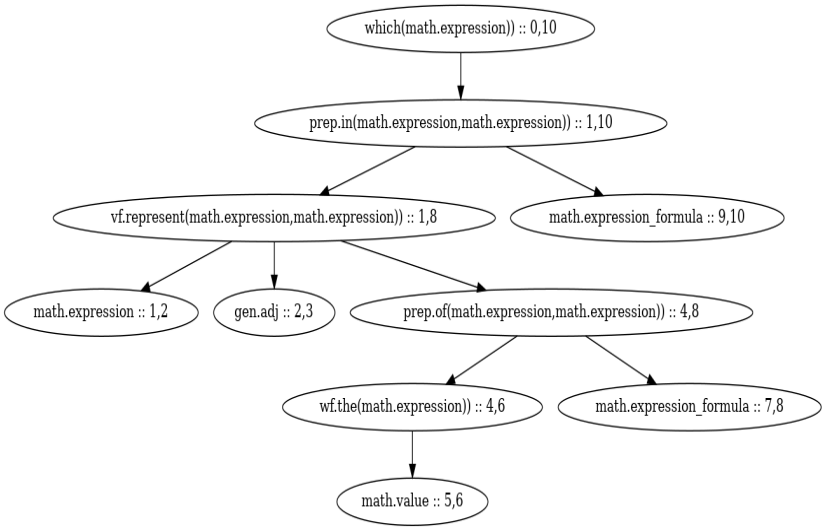
Zadatak 1	
Ulaz	Compare $\sqrt{26}$ and 4.9.
Semantička reprezentacija	
Izlaz sustava	compare($root(2, 26), 4.9$)
Akcije	(nije definirano)

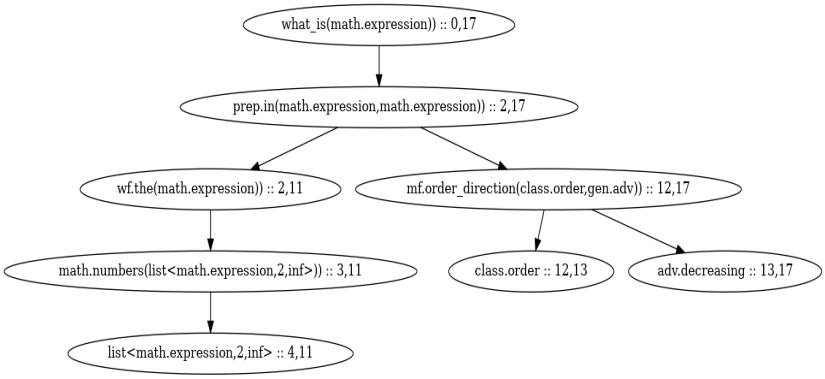
Zadatak 2	
Ulaz	The area of a trapezoid is $A = frac(1, 2) * h * (b_1 + b_2)$. Solve for b_1. b_1
Semantička reprezentacija (1. rečenica)	
Semantička reprezentacija (2. rečenica)	
Izlaz sustava	solve($A = frac(1, 2) * h * (b_1 + b_2), b_1$)
Akcije	$A = frac(1, 2) * h * (b_1 + b_2), b_1$, `eq_solve`, b_1

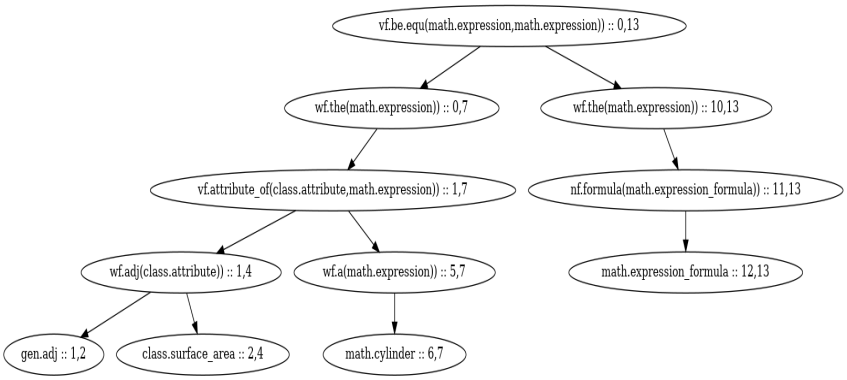
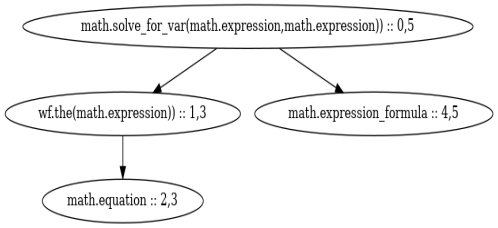
Zadatak 3	
Ulaz	Factor $6x^2 - 216$.
Semantička reprezentacija	
Izlaz sustava	factorise($6x^2 - 216$)
Akcije	$6x^2 - 216$, `factorise_call`

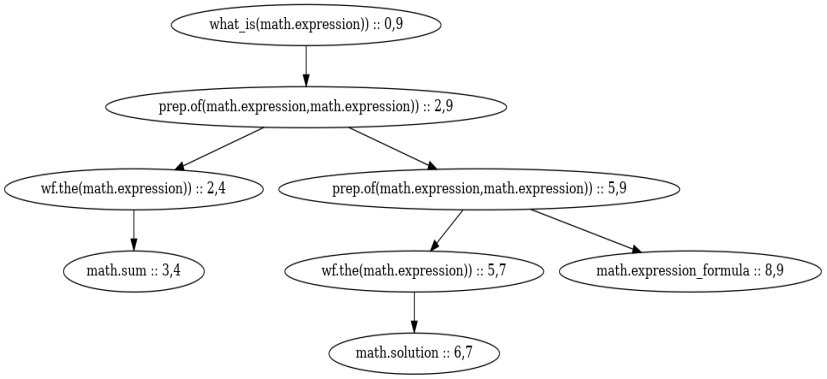
Zadatak 4	
Ulaz	Order the numbers from least to greatest. $-1.5, -0.5, -\sqrt{2}, -1.4$
Semantička reprezentacija (1. rečenica)	
Semantička reprezentacija (2. rečenica)	
Izlaz sustava	order($list < -1.5, -0.5, -\sqrt{2}, -1.4 >$, increasing)
Akcije	(nije definirano)

Zadatak 5	
Ulaz	Solve compound inequality. $2x > -10$ and $9x < 18$
Semantička reprezentacija (1. rečenica)	
Semantička reprezentacija (2. rečenica)	
Izlaz sustava	solve($system(2x > -10, 9x < 18)$)
Akcije	$system(2x > -10, 9x < 18)$

Zadatak 6	
Ulaz	Which expression best represents the value of x in $y = mx + b$?
Semantička reprezentacija	
Izlaz sustava	solve($y = mx + b$, x)
Akcije	$y = mx + b$, `eq_solve`, x

Zadatak 7	
Ulaz	What are the numbers 1.9, $\frac{5}{4}$, -1.2 and $\sqrt{3}$ in order from greatest to least?
Semantička reprezentacija	
Izlaz sustava	order($list < 1.9, frac(5, 4), -1.2, root(2, 3) >$, decreasing)
Akcije	(nije definirano)

Zadatak 8	
Ulaz	The lateral surface area of a cylinder is given by the formula $S = frac(2, \pi * r * h)$. Solve the equation for r.
Semantička reprezentacija (1. rečenica)	
Izlaz sustava	solve($S = frac(2, \pi * r * h)$, r)
Semantička reprezentacija (2. rečenica)	
Izlaz sustava	solve($S = frac(2, \pi * r * h)$, r)
Akcije	$S = frac(2, \pi * r * h)$, `eq_solve`, r

Zadatak 9	
Ulaz	What is the sum of the solutions of $\|2x + 4\| - 6 = 8$?
Semantička reprezentacija	
Izlaz sustava	solve($abs(2x + 4) - 6 = 8$), sum_list($ARG(0)$)
Akcije	(nije definirano)

Komentari i prijedlozi za daljnji rad

Pozorni je čitatelj mogao primijetiti nekoliko problema s kojima se suočava razvijeni sustav. Prvo, prilikom parsiranja ulaznog niza znakova, parser se jako oslanja na točnost POS, odnosno NER oznaka za određivanje pripadnosti tokena određenim tipovima. Drugo, modul zaključivanja na ulaz dobiva samo jednu semantičku reprezentaciju po rečenici zadatka, a parser ih generirano vrlo mnogo. Sve greške načinjene tijekom pretprocesiranja i parsiranja propagiraju se kroz sustav, a ne postoji nikakav mehanizam oporavka.

Nadalje, gramatika semantičkog parsera pisana je ručno što nije nimalo jednostavan zadatak i podložan je greškama. Za kvalitetniji semantički parser potrebno je sastaviti označeni skup podataka dostatan za učenje prioriteta pojedinih gramatičkih pravila. U trenutku pisanja ovog rada, jezik SML broji 125 razreda, 56 funkcija te 193 gramatičkih pravila za njihovo generiranje.

Algoritam zaključivanja i strukture koje se koriste sigurno će biti potrebno dodatno prilagoditi kako će korisnik nailaziti na nove zadatke. Primjerice, već sada je jasno da strukturu konteksta treba nadograditi za kvalitetnije razrješavanje nepoznatih entiteta (jedna od mogućnosti je entitetima dati oznaku vidljivosti, slično strukturama DRS). Također, trenutna implementacija sve generirane akcijske čvorove šalje na izlaz, što će biti potrebno promijeniti ukoliko se želi pokriti širi spektar zadataka. Na primjer, zadatak "*The sum of two numbers equals 10. What is the value of the second number if the first one is 5?*" stvorit će čvor tipa *SMR_ADD_NODE* tijekom obrade prve rečenice, a taj je tip interno označen kao *akcijski*.

Jedan veliki problem cijelog projekta je nedovoljna definiranost ciljanog skupa podataka koji se želi podržati. Inicijalna ideja bila je omogućiti rješavaču Photomatha prepoznavanje jednostavnih uputa za rješavanje jednadžbi. Problem se javlja sa određivanjem što točno jesu te "jednostavne upute". Stvar je jasna za jednostavne zadatke tipa "*Find the inverse of a matrix ...*" koji jasno definiraju akciju ophođenja s matematičkim izrazom. Međutim, tako definiran problem ima svega nekolicinu mogućih varijanti ulaznih zadataka i generalno nije previše zanimljiv. S druge pak strane, za imalo "složenije" zadatke drastično se mijenja semantika problema. Na primjer, zadatak "*For* $f(x) = -4x + 1$*, what is the output for the given input? x = 0*" za pravilno razrješavanje zahtjeva parsiranje matematičkog izraza i shvaćanje njegove semantike (moramo znati da se radi o funkciji jer inače "izlazna vrijednost" (engl. *output*) nema nikakvog smisla).

S druge strane, neki problemi karakteristični za semantičko parsiranje nisu se po-

javili u implementaciji ovog sustava. Primjerice, anaforičke fraze česte su pojave u prirodnom jeziku, no rijetko se koriste u izražavanju matematičkih zadataka. Problemi vezani za referenciranje entiteta pojavljuju se kod zadataka s 2 ili više rečenica, gdje je potrebno pronaći objekt na koji se odnosi neka funkcija u globalnom kontekstu zadatka. Za proširivanje sustava na algebarske zadatke iz kojih je potrebno graditi sustave jednadžbi – a koje vrve zamjenicama – potrebno je osmisliti neki mehanizam njihovog razrješavanja. Jedna mogućnost je npr. unošenje dodatnog znanja o muškim i ženskim imenima za pomoć pri konkretiziranju osobnih zamjenica.

Motivacija za korištenje kreatora za definiranje transformacija nad semantičkim stablom krije se u njihovoj modularnosti. Ono što u sustavu još nije implementirano (a vrlo lako može biti) jesu generički kreatori koji bi se mogli koristiti za definiranje uvjeta transformacija ili manipuliranje argumentima ulaznih čvorova. Semantika takvih kreatora slična je funkcijama višeg reda u funkcijskim jezicima (npr. `map`, `fold`, `find` i sl.)

Konačno, simbolički pristup semantičkoj analizi pogodan je iz dva razloga: omogućuje definiranje egzaktnih pravila zaključivanja, što je u suglasnosti s rješavanjem matematičkih zadataka, te nudi relativno jednostavno proširenje sustava koje za korake zaključivanja krajnjem korisniku nudi razlog njihovog izvođenja. To je potencijalno značajan edukacijski sadržaj za aplikaciju Photomath.

5. Zaključak

Problem automatiziranog rješavanja tekstnih matematičkih zadataka zaokuplja stručnjake još od začetka ideje umjetne inteligencije. Kako zadovoljavajuće rješenje problema još uvijek nije na vidiku, u ovom radu razmotren je podskup izvornog problema – fokus je stavljen na matematičke zadatke iskazane riječima koji u tekstu već sadrže formulu matematičkog izraza popraćenu dodatnim informacijama koje specificiraju metodu ili varijablu rješavanja.

U ovom radu implementirana je prilagodba metode simboličkog pristupa semantičkom parsiranju opisane u (Shi et al., 2015). Izvorni cilj bio je sakupiti veći skup podataka te unijeti potrebno znanje u model za uspješno parsiranje i zaključivanje nad njegovim rezultatom. Međutim, zadatak u startu nije bio u potpunosti određen, bilo je potrebno istražiti mogućnosti i postojeće pristupe rješavanja tekstualnih zadataka te na temelju toga osmisliti i implementirati rješenje. Većina rada prati ideje spomenutog rada prilagođene za postojeći skup podataka te za modul zaključivanja koristi neke jezične konstrukte simboličkog jezika rješavača Photomath. Rad detaljno opisuje sve postojeće mogućnosti jezika SML kao i prostor za njihovo proširivanje i uvođenje novih funkcionalnosti. Da bi sustav bilo moguće testirati na novom skupu podataka, prethodno je nužno unijeti znanje kroz tipove jezika SML i njihova gramatička pravila te transformacijske postupke. Rad sustava demonstriran je na nekoliko radnih primjera.

Ovako definiran sustav nije skalabilan – sve "zanimljivo" znanje potrebno je unijeti ručno. Proširivanje funkcionalnosti na druge jezike zahtijeva definiranje cijele nove gramatike jer ne postoji način da se dio posla automatizira procesom učenja iz podataka. Međutim, jednom napisani modul zaključivanja ovisan je samo o semantici razreda i funkcija, što ga čini neovisnim o odabiru jezika. Nadalje, trebalo bi sastaviti veći skup zadataka označenih sa semantičkom reprezentacijom. Iz oznaka značenjske reprezentacije mogao bi se naučiti bolji sustav rangiranja generiranih stabala od postojećeg. Također, za proširivanje domene modela potrebno je pokriti puno više imenica i glagola kroz tipove jezika SML korištenjem nekih od javno dostupnih baza (npr. VerbNet ili WordNet).

LITERATURA

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers, 2014. The Association for Computer Linguistics. ISBN 978-1-937284-72-5. URL http://aclweb.org/anthology/P/P14/.

Jacob Andreas, Andreas Vlachos, i Stephen Clark. Semantic Parsing as Machine Translation, 2013.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, i Nathan Schneider. Abstract Meaning Representation for Sembanking, 2013.

Bojana Dalbelo Bašić i Jan Šnajder. *UMJETNA INTELIGENCIJA - Zakljucivanje uporabom propozicijske i predikatne logike*. Sveučilište u Zagrebu, 2008. ISBN 978-953-184-149-8.

Jonathan Berant i Percy Liang. Semantic Parsing via Paraphrasing. U *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers* DBL (2014), stranice 1415–1425. ISBN 978-1-937284-72-5. URL http://aclweb.org/anthology/P/P14/P14-1133.pdf.

Jonathan Berant, Andrew Chou, Roy Frostig, i Percy Liang. Semantic Parsing on Freebase from Question-Answer Pairs. U *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, stranice 1533–1544. ACL, 2013. ISBN 978-1-937284-97-8. URL http://aclweb.org/anthology/D/D13/D13-1160.pdf.

Patrick Blackburn i Johan Bos. *Representation and Inference for Natural Language:*

A First Course in Computational Semantics. Center for the Study of Language and Information, Stanford, CA, 2005. ISBN 978-1-57586-496-9.

D.G. Bobrow. Natural language input for a computer problem-solving system. *Semantic Information Processing*, stranice 146–226, 1968.

Qingqing Cai i Alexander Yates. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. U *In Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

Randall Charles, Barisa Hall, Dan Kennedy, Allan E. Bellman, Sadie Chavis Bragg, William G. Handlin, Stuart J. Murphy, i Grant Wiggins. *Algebra 2 Common Core, Student Edition*. Prentice Hall, 2011. ISBN 0133186024.

E. Charniak. CARPS, A PROGRAM WHICH SOLVES CALCULUS WORD PROBLEMS. Technical report, Cambridge, MA, USA, 1968.

Jinho D. Choi. Combinatory Categorical Grammar, 2017. URL `https://github.com/emory-courses/cs571/wiki/Schedule`.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956. `http://www.chomsky.info/articles/195609--.pdf` – last visited 14^{th} January 2009.

Li Dong i Mirella Lapata. Language to Logical Form with Neural Attention. *CoRR*, abs/1601.01280, 2016. URL `http://arxiv.org/abs/1601.01280`.

Jay Earley. An Efficient Context-free Parsing Algorithm. *Commun. ACM*, 13(2):94–102, Veljača 1970. ISSN 0001-0782. doi: 10.1145/362007.362035. URL `http://doi.acm.org/10.1145/362007.362035`.

Jan van Eijck. Discourse Representation Theory. 2005. URL `http://homepages.cwi.nl/~jve/papers/05/drt/drt.pdf`.

Charles R. Fletcher. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571, 1985. ISSN 1532-5970. doi: 10.3758/BF03207654. URL `http://dx.doi.org/10.3758/BF03207654`.

Peter Thomas Geach. *Reference and Generality: An Examination of Some Medieval and Modern Theories*. Cornell University Press, 1980.

Daniel Gildea i Daniel Jurafsky. Automatic Labeling of Semantic Roles. *Comput. Linguist.*, 28(3):245–288, Rujan 2002. ISSN 0891-2017. doi: 10.1162/089120102760275983. URL http://dx.doi.org/10.1162/089120102760275983.

Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, i Percy Liang. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. *CoRR*, abs/1704.07926, 2017. URL http://arxiv.org/abs/1704.07926.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, i Nate Kushman. Learning to Solve Arithmetic Word Problems with Verb Categorization. U *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, stranice 523–533, 2014. URL http://aclweb.org/anthology/D/D14/D14-1058.pdf.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, i Wei-Ying Ma. How well do Computers Solve Math Word Problems? Large-Scale Dataset Construction and Evaluation. U *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. URL http://aclweb.org/anthology/P/P16/P16-1084.pdf.

Robin Jia i Percy Liang. Data Recombination for Neural Semantic Parsing. *CoRR*, abs/1606.03622, 2016. URL http://arxiv.org/abs/1606.03622.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, i Siena Ang. Parsing Algebraic Word Problems into Equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/692.

Nate Kushman, Luke Zettlemoyer, Regina Barzilay, i Yoav Artzi. Learning to Automatically Solve Algebra Word Problems. U *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers* DBL (2014), stranice 271–281. ISBN 978-1-937284-72-5. URL http://aclweb.org/anthology/P/P14/P14-1026.pdf.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, i Mark Steedman. Inducing Probabilistic CCG Grammars from Logical Form with Higher-order Unification. U *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, stranice 1223–1233, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1870658.1870777.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, i Luke Zettlemoyer. *Scaling semantic parsers with on-the-fly ontology matching*, stranice 1545–1556. Association for Computational Linguistics (ACL), 2013. ISBN 9781937284978.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, i Ni Lao. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. *CoRR*, abs/1611.00020, 2016. URL http://arxiv.org/abs/1611.00020.

Saxon. *Algebra 1 : Homeschool Kit*. Saxon Publishers, 1998. ISBN 1565771230.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, i Yong Rui. Automatically Solving Number Word Problems by Semantic Parsing and Reasoning. U Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, i Yuval Marton, urednici, *EMNLP*, stranice 1132–1142. The Association for Computational Linguistics, 2015. ISBN 978-1-941643-32-7. URL http://dblp.uni-trier.de/db/conf/emnlp/emnlp2015.html#ShiWLLR15.

Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0-262-19420-1.

Yuk Wah Wong i Raymond J. Mooney. Learning for Semantic Parsing with Statistical Machine Translation. U *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, stranice 439–446, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. doi: 10.3115/1220835.1220891. URL http://dx.doi.org/10.3115/1220835.1220891.

Yuk Wah Wong i Raymond J. Mooney. Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. U *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, Prague, Czech Republic, June 2007. URL http://nn.cs.utexas.edu/?wong:acl07.

Luke S. Zettlemoyer i Michael Collins. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. *CoRR*, abs/1207.1420, 2012. URL http://arxiv.org/abs/1207.1420.

Lipu Zhou, Shuaixiang Dai, i Liwei Chen. Learn to Solve Algebra Word Problems Using Quadratic Programming. U *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, stranice 817–822, 2015. URL http://aclweb.org/anthology/D/D15/D15-1096.pdf.

Dodatak A
Popis matematičkih zadataka

Zadatak	Akcije
The area of a trapezoid is $A = \frac{1}{2} * h * (b_1 + b_2)$. Solve for b_1.	$A = \frac{1}{2} * h * (b_1 + b_2)$, eq_solve, b_1
Factor $6x^2 - 216$.	$6x^2 - 216$, factorise_call
A rectangle has a length of $2x + 3$ and a width of $x - 4$. Find the area of the rectangle.	$(2x + 3) * (x - 4)$, simplify_wrapper
What is the y-intercept of the line that passes through the points $(-4, 4)$ and $(2, -5)$?	$y - 4 = \frac{-5-4}{2-(-4)} * (x - (-4))$, substitute($x = 0$); { ARG0 }, eq_solve, y
Solve the equation using the Quadratic Formula. $6x^2 - 10x + 3 = 0$.	$6x^2 - 10x + 3 = 0$, eq_quad_cs
Simplify the expression bellow. $(-6y^{-4})^5$	$(-6y^{-4})^5$, simplify_wrapper
What is the solution to $\frac{2n+8}{3} = \frac{n+7}{2}$?	$\frac{2n+8}{3} = \frac{n+7}{2}$, eq_solve, n
Which is the 7th number in this pattern? $8, 13, 18, 23, ...$	$x = 3 + 5y$, substitute($y = 7$); { ARG0 }, eq_solve, x
What is the value of r when $s = -1$, $t = 4$, and $u = \frac{1}{5}$? $r = 3s^2 + 5(t - 2u)$	$r = 3s^2 + 5(t - 2u)$, substitute($s = -1$); { ARG0 }, substitute($t = 4$); { ARG0 }, substitute($u = \frac{1}{5}$); { ARG0 }, eq_solve, x
Order the numbers from least to greatest. $-1.5, -0.5, -\sqrt{2} - 1.4$	$(-1.5, -0.5, -\sqrt{2} - 1.4)$, order_increasing
Solve compound inequality. $2x > -10$ and $9x < 18$	$system(2x > -10, 9x < 18)$

Zadatak	Akcije
Evaluate each expression for $x = -4$ and $y = 3$. $x - 2y + 3$, $x + \frac{x}{y}$	$x - 2y + 3$, substitute($x = -4$); { ARG0 }, substitute($y = 3$); { ARG0 }, calculate; $x + \frac{x}{y}$, substitute($x = -4$); { ARG0 }, substitute($y = 3$); { ARG0 }, calculate
Which expression best represents the value of x in $y = mx + b$?	$y = mx + b$, eq_solve, x
What is the positive solution of $\|2x + 8\| = 19$?	$\|2x + 8\| = 19$, eq_solve, x {ARG0}, get_positive
If p is an integer, what is the least possible value of p in the following inequality? $\|3p - 5\| <= 7$	$\|3p - 5\| <= 7$; { ARG0 }, round_floor
Evaluate $3t(t + 2) - 3t^2$ for $t = 19$.	$3t(t + 2) - 3t^2$, substitute($t = 19$); { ARG0 }, calculate
The lateral surface area of a cylinder is given by the formula $S = \frac{2}{\pi * r * h}$. Solve the equation for r.	$S = \frac{2}{\pi * r * h}$, eq_solve, r
How many negative solutions does $2\|3x - 6\| <= 6$ have?	$2\|3x - 6\| <= 6$, eq_solve, x; { ARG0 }, get_negative; { ARG0 }, count
For which value of a does $4 = a + \|x - 4\|$ have no solution?	$4 = a + \|x - 4\|$, domain, a; { ARG0 }, inverse
What is the sum of the solutions of $\|2x + 4\| - 6 = 8$?	$\|2x + 4\| - 6 = 8$; { ARG0 }, sum
What is the value of $4x^2 + 2x - 1$ when $x = \frac{3}{4}$? Express the answer as a decimal.	$4x^2 + 2x - 1$, substitute($x = \frac{3}{4}$); { ARG0 }, calculate_float
What is the coefficient of b in the simplified form of the expression $-8(a - 3b) + 2(-a + 4b + 1)$?	$-8(a - 3b) + 2(-a + 4b + 1)$, simplify_wrapper; { ARG0 }, coefficient, b

Zadatak	Akcije
For $f(x) = -4x + 1$, what is the output for the given input? a) -2, b) 0	$f(x) = -4x + 1$, substitute($x = -2$); { ARG0 }, calculate; $f(x) = -4x + 1$, substitute($x = 0$); { ARG0 }, calculate
Determine whether y is a function of x. $y^2 = 3x - 7$	$y^2 = 3x - 7$, eq_solve, y
If $f(x) = -3x + 7$ and $g(x) = -7x + 3$. What is the value of $f(-3) - g(-3)$	$f(x) = -3x + 7$, substitute($x = -3$); { ARG0 }, calculate; $g(x) = -7x + 3$, substitute($x = -3$); { ARG0 }, calculate
What are the numbers 1.9, $\frac{5}{4}$, -1.2 and $\sqrt{3}$ in order from greatest to least?	(1.9, $\frac{5}{4}$, -1.2, $\sqrt{3}$), order_decreasing
What number is a solution of both $\|x - 3\| = 2$ and $\|9 - x\| = 8$?	$\|x - 3\| = 2$, eq_solve, x; $\|9 - x\| = 8$, eq_solve, x; ({ ARG0 },{ ARG1 }), intersection
Find the slope of the line through given pair of points. $(1, 6)$ and $(8, -1)$	$y - 6 = \frac{-1-6}{8-1} * (x - 1)$, eq_solve, y; { ARG0 }, coefficient, x
Find the slope and y-intercept of a line. $-\frac{1}{3} * x + \frac{2}{3} * y = \frac{5}{3}$	$-\frac{1}{3} * x + \frac{2}{3} * y = \frac{5}{3}$, eq_solve, y; { ARG0 }, coefficient, x; { ARG1 }, substitute($x = 0$); { ARG0 }, calculate
Write an equation of the line in standard form with integer coefficients. $y = \frac{1}{2} * x - 2$	$y = \frac{1}{2} * x - 2$,implicit_form
The line $(y - 1) = \frac{2}{3}(x + 1)$ contains point (a, -3). What is the value of a?	$(y - 1) = \frac{2}{3}(x + 1)$, substitute($y = 3$); { ARG0 }, calculate
What is the equation of a line parallel to $y = x$ that passes through the point $(0, 1)$?	$y - 1 = 1 * (x - 0)$

Zadatak	Akcije
Suppose y varies directly with x. If x is 30 when y is 10, what is x when $y = 9$?	$y - 10 = 1 * (x - 30)$; { ARG0 }, substitute($y = 9$); { ARG0 }, eq_solve, x
For which value of b would the equation $3\|x - 2\| = b - 6$ have infinitely many solutions?	$\frac{b-6}{3} > 0$, eq_solve, b
If $4(x + 2) - 2(x - 10) = 0$, what is the value of x?	$4(x + 2) - 2(x - 10) = 0$, eq_solve, x
Solve using matrix. $system(6x + 3y = -15, 2x + 4y = 10)$	$system(6x + 3y = -15$, $2x + 4y = 10)$, system_by_matrix
Determine whether the function $f(x) = 0.25(2x - 15)^2 + 150$ has a maximum or a minimum value.	$f(x) = 0.25(2x - 15)^2 + 150$, find_optimum
Rewrite $y = -2x^2 + 35$ in vertex form.	$y = -2x^2 + 35$, line_vertex_form
Which is the composition $f(g(x))$, if $f(x) = -x - 3$ and $g(x) = 7 + 5x$?	$f(x) = -x - 3$, substitute($x = 7 + 5x$); { ARG0 }, simplify_wrapper
Solve the equation by factoring. $x^2 - 10x + 25 = 0$	$x^2 - 10x + 25 = 0$, factorise_solve
Solve system by elimination. $system(2x + y = 4, 3x - y = 6)$	$system(2x + y = 4, 3x - y = 6)$, system_by_elimination
How many different real solutions are there for $2x^2 - 3x + 5 = 0$?	$2x^2 - 3x + 5 = 0$, eq_solve, x; { ARG0 }, get_real; { ARG0 }, count
Find the sum and product of the roots of the equation. $x^2 - 2x + 3 = 0$	$x^2 - 2x + 3 = 0$, { ARG0 }, sum; { ARG1 }, product
Find a value of a for which the line $y = x + a$ separates the parabolas $y = x^2 - 3x + 2$ and $y = -x^2 + 8x - 15$.	(unknown)

Zadatak	Akcije
What is the axis of symmetry for the graph of the quadratic equation $y = -3x^2 - 12 + 12x$?	$y = -3x^2 - 12 + 12x$, find_optimum; { ARG0 }, x_axis
What is the coefficient of the x-term of the factorization of $25x^2 + 20x + 4$?	$25x^2 + 20x + 4$, factorise_call; { ARG0 }, coefficient, x
Compare the two numbers. Use > or < . $16, \sqrt{16}$	$(16, \sqrt{16})$, compare
Write number as a percent. 0.5	$0.5 * 100\%$
Compare $\sqrt{26}$ and 4.9.	$(\sqrt{26}, 4.9)$, compare
What is the number of x-interscepts of the parabola with equation $y = 6x^2 - 4x - 3$?	$y = 6x^2 - 4x - 3$; { ARG0 }, count

Tablica A.1: Ulaz u sustav i očekivani izlaz

Semantička analiza matematičkih zadataka iskazanih riječima

Sažetak

Rad opisuje simbolički pristup problemu automatiziranog rješavanja tekstnih matematičkih zadataka koji u sebi sadrže gotove formule matematičkih izraza i tekst koji pobliže opisuje način na koji ih treba rješavati. Razvijeni sustav svaku rečenicu zadatka prevodi u semantičku reprezentaciju stabla te njegovim obilaskom generira naredbe za izvršavanje od strane rješavača Photomath. Za semantičku analizu zadataka razvijena je arhitektura domensko-specifičnog jezika SML (engl. *Semantic Math Language*) definiranog tipovima, kontekstno-neovisnom gramatikom i transformacijskim pravilima za generiranje izlaznih naredbi. Funkcionalnost modela isprobana je na nekolicini primjera, no za valjanu evaluaciju jezik SML potrebno je nadopuniti dodatnim znanjem.

Ključne riječi: obrada prirodnog jezika, semantička reprezentacija, domensko-specifični jezik, tekstni matematički zadaci, Photomath

Semantic Analysis of Math Word Problems

Abstract

This thesis describes a symbolic approach to the problem of automatically solving math word problems containing formulae of mathematical expressions and supplement text describing the desired procedure of solving them. The system translates each sentence of the math word problem into semantical representation using trees. By traversing the trees in postorder fashion the system generates instructions for Photomath solver. For the semantic analysis of word problems an architecture of domain-specific SML language (abbr. of Semantic Math Language) is developed. The language consists of user-defined types, context-free grammar, and transformation rules for the generation of output instructions. The functionality of the model is tested on a few examples, but for its proper evaluation further work on the SML language is needed.

Keywords: natural language processing, meaning representation, domain-specific language, math word problems, Photomath