Usporedba alata za zaključivanje u deskriptivnim logikama

Pavić, Hrvoje

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Pavić

USPOREDBA ALATA ZA ZAKLJUČIVANJE U DESKRIPTIVNIM LOGIKAMA

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Pavić

Matični broj: 35197929–R

Studij: ***Organizacija poslovnih sustava***

USPOREDBA ALATA ZA ZAKLJUČIVANJE U DESKRIPTIVNIM LOGIKAMA

DIPLOMSKI RAD

Mentorica:

Prof. dr. sc. Sandra Lovrenčić

Varaždin, rujan 2024.

Hrvoje Pavić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom radu opisane su deskriptivne logike, od samih početaka, pa do korištenja danas. Pojašnjeni su osnovni i prošireni jezici deskriptivnih logika uz primjere. Detaljno je objašnjeno kako funkcionira Tableau algoritam za $\mathcal{ALC}$, te je kreirana vlastita ontologija u alatu Protégé, kako bi se moglo provesti zaključivanja nad njom. Provedba zaključivanja je glavna tema ovog rada, jer je cilj usporediti alate koji provode zaključivanje. Napravljena je usporedna analiza alata za zaključivanje, koji su dostupni u alatu Protégé. Podaci za usporednu analizu su dobiveni nakon provjere zaključivanja nad vlastitom ontologijom.

Ključne riječi: deskriptivne logike, ontologije, baze znanja, zaključivanje, alati za zaključivanje, Protégé, Pellet, HermiT, jcel, ELK

Sadržaj

1. Uvod

Razvoj tehnologije i znanja u računalnoj znanosti rezultirao je pojavom alata za prikaz i obradu informacija i znanja. Deskriptivne logike postale su bitan alat u području umjetne inteligencije, pogotovo u domenama ontologija i semantičkog weba. One pružaju formalni okvir za opisivanje i strukturiranje složenih domena znanja, omogućujući preciznu definiciju pojmova i odnosa između njih.

Jedan od najvažnijih jezika za prikaz znanja u deskriptivnim logikama je $\mathcal{ALC}$. On čini osnovu mnogih proširenih sustava deskriptivnih logika i ključan je za razumijevanje rada s ontologijama. U ovom radu posebna je pažnja posvećena Tableau algoritmu, koji se koristi za automatsko zaključivanje (ili rezoniranje) u deskriptivnim logikama.

Ontologije, kao formalni modeli specifičnih područja znanja, igraju veliku ulogu u semantičkom webu. OWL 2 (Web Ontology Language) jedan je od najkorištenijih jezika za definiranje ontologija, a alati kao Protégé omogućuju izradu i manipulaciju ontologijama, na pregledan način.

U ovom radu analizirani su različiti alati za automatsko zaključivanje (engl. *reasoners*), konkretno Pellet, HermiT, jcel i ELK, s posebnim naglaskom na njihove performanse. Također, prikazana je izrada vlastite ontologije u programu Protégé i primjena navedenih alata na njoj.

Motivacija za odabir ove teme proizlazi iz mog interesa za kreiranje ontologije u alatu Protégé i rješavanja Tableau algoritma. S njima sam se upoznao na kolegiju Baze znanja i semantički web, te mi se činilo zanimljivim proširiti znanje u ovom području.

2. Metode i tehnike rada

U ovom radu nakon proučene literature korištena je metoda sekundarne analize podataka za pisanje teoretskog dijela o deskriptivnim logikama. Također, za izradu vlastite ontologije korišten je program Protégé. Ontologija je izrađena na temu MMORPG online igre *RuneScape*. U svrhu validiranja konzistentnosti ontologije korišteni su alati za zaključivanje.

3. Deskriptivne logike

Deskriptivne logike (u nastavku DL, engl. *Description Logics*) su formalizmi koji se koriste za prikaz znanja unutar određene domene, a omogućuju precizno definiranje koncepata i odnosa među njima.

DL su skup logičkih jezika koji olakšavaju prikaz strukturiranog znanja i omogućuju zaključivanje na temelju tog znanja. Pružaju temelje za ontologije i baze znanja, što ih čini ključnima u područjima poput umjetne inteligencije i semantičkog weba.

U ontologijama koncepti predstavljaju klase unutar domene, dok odnosi ili uloge opisuju veze između tih koncepata. Individue su osnovni entiteti koji predstavljaju konkretne objekte ili instance (u radu se koriste naizmjenično izrazi individua i instanca) koncepata unutar domene. Uloge povezuju individue s konceptima ili s drugim individuama, omogućujući modeliranje složenih odnosa u domeni.

Za primjer modeliranja znanja u knjižnici koristeći DL mogu se definirati koncepti i odnosi kako bi se strukturirale informacije u knjižnici. Koncepti mogu biti knjiga, čitatelj i autor. Knjiga predstavlja sve knjige u knjižnici, čitatelj predstavlja sve korisnike knjižnice koji posuđuju knjige, a autor predstavlja sve autora knjiga koje su dostupne u knjižnici. Odnosi mogu biti 'napisao' i 'posudio'. Odnos između koncepata knjiga i autor je 'napisao', dok je 'posudio' odnos između knjige i čitatelja. „Dječak koji je volio jelene“ je primjer instance koncepta knjiga, Samuel Bjork koncepta autor, a Ivan Horvat koncepta čitatelj.

Korištenjem DL, može se formalno definirati navedene koncepte i odnose korištenjem terminološke komponente ($u\ nastavku\ TBox, engl. Terminological\ Box$) i asertivne komponente ($u\ nastavku\ ABox, engl. Assertional\ Box$). $TBox$ sadrži definicije koncepata i uloga dok $ABox$ sadrži tvrdnje o individuama i njihovim pripadnostima konceptima ili ulogama.

$TBox$ predstavlja terminološki dio baze znanja gdje se definira hijerarhiju koncepata i odnose među njima. U $TBox$-u navodimo kako su koncepti međusobno povezani, koristeći deklaracije i definicije. U primjeru knjižnice, može se definirati koncept "Inaktivan čitatelj" kao podređeni koncept koncepta "Čitatelj" s dodatnim svojstvom da nije posudio niti jednu knjigu u zadnjih godinu dana. Ovo se formalno može izraziti:

$$Inaktivan\ čitatelj \equiv Čitatelj \sqcap posudioManjeOd(1, Knjiga)$$

Najčešće $TBox$ dopušta samo jednu definiciju za svaki koncept kako bi se izbjegle kontradikcije i omogućilo konzistentno zaključivanje. Također, definicije trebaju biti acikličke, što znači da koncept ne smije biti definiran samim sobom ili preko lanca drugih koncepata koji se vraćaju na njega.

S druge strane, $ABox$ sadrži informacije o konkretnim individuama i njihovoj pripadnosti konceptima ili ulogama. $ABox$ uključuje tvrdnje poput:

$Čitatelj(IvanHorvat)$

$Knjiga(DječakKojiJeVolioJelene)$

$posudioKnjigu(IvanHorvat, DječakKojiJeVolioJelene)$

$napisaoKnjigu(SamuelBjork, DječakKojiJeVolioJelene)$

$Autor(SamuelBjork)$

Navedene tvrdnje pružaju informacije o objektima u domeni knjižnice. $TBox$ pruža strukturu i odnose između koncepata, dok $ABox$ pruža podatke o individuama. Ova podjela također olakšava održavanje baze znanja, jer se terminološkim promjenama može upravljati u $TBox$-u bez izmjene konkretnih podataka u $ABox$-u. Kombinirajući informacije iz $ABox$-a i definicije iz $TBox$-a, može se izvršiti zaključivanje.

Precizno definiranje koncepata i odnosa između njih je ključno za razvoj baza znanja, iz kojih je moguće zaključivati nove informacije na temelju postojećih podataka. Da bi baza znanja bila kreirana, prvo je potrebno identificirati sve ključne koncepte i odnose unutar odabrane domene, zatim se korištenjem formalizama DL oni definiraju. Potreban je unos konkretnih instanci koncepata i odnosa te cjelokupnu bazu znanja validirati, kako bi se osiguralo da baza znanja točno predstavlja stvarni svijet. Primjer zaključivanja bi bio da spoznajmo ako je Ivan Horvat posudio knjigu „Dječak koji je volio jelene“, a ta knjiga napisana je od autora Samuel Bjork, tada je moguće zaključiti da čitatelj Ivan Horvat čita knjigu od autora Samuel Bjork. Dakle, na temelju eksplicitno definiranih podataka, da je čitatelj posudio knjigu i da je ta knjiga napisana od strane autora, može se zaključiti da čitatelj čita knjigu koju je napisao autor. Tako je izvedeno implicitno znanje, jer podaci nisu bili direktno uneseni, već su logički zaključeni iz postojećih odnosa.

Neke od prednosti izvođenja implicitnog znanja domene su automatizacija i personalizacija. Automatizirano zaključivanje štedi vrijeme i smanjuje mogućnost grešaka prilikom ručnog unosa podataka. Personalizacija se može objasniti na primjeru knjižnice, gdje kreiranje sustava preporuke, olakšava preporuku buduće knjige za čitanje korisniku, na temelju autora knjiga koje je korisnik čitao do sada. Ovakav proces zaključivanja omogućava knjižnicama da bolje razumiju interese korisnika i optimiziraju usluge prema njihovim potrebama.

DL također pružaju osnove za standardne jezike poput OWL (Web Ontology Language), koji se koristi za izradu ontologija na semantičkom webu. Tako je omogućena interoperabilnost između drugačijih sustava i olakšana razmjena i ponovna uporaba znanja.
Deskriptivne logike su odlučivi dio logike prvog reda. Odlučivost znači da za svaki problem koji se postavi unutar odabranog sustava, postoji algoritam koji će uvijek završiti (neće zapeti u beskonačnoj petlji) te će uvijek dati točan odgovor, bio on pozitivan ili negativan. Proširimo postojeći primjer s novim konceptom 'Aktivan čitatelj', koji predstavlja sve čitatelje koji su posudili više od 10 knjiga godišnje, te novim odnosom 'je' koji definira da čitatelj jest aktivan

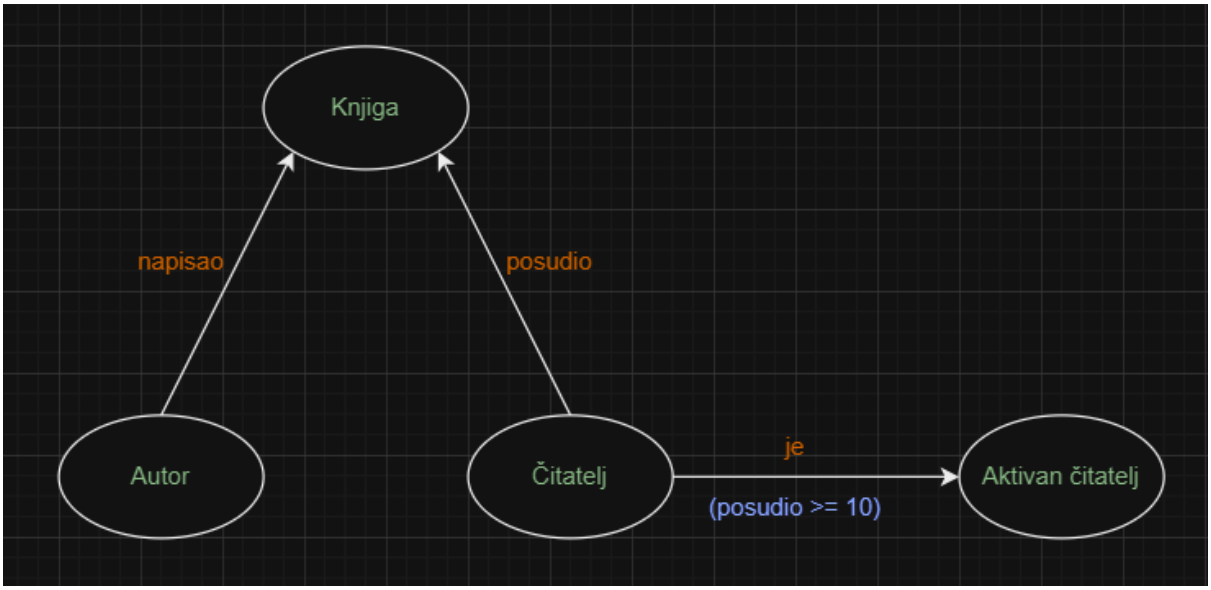
čitatelj. Moguće je kreirati algoritam koji provjerava posudbe svakog čitatelja, te na temelju definiranog uvjeta, ako se uvjet zadovolji, čitatelju je dodjeljen status 'Aktivan čitatelj'. Algoritam završava s pozitivnim odgovorom (da, čitatelj je aktivan) ili negativnim (ne, čitatelj nije aktivan), i to unutar konačnog perioda. Svojstva odlučivosti i konačnog vremena izvršenja algoritama za zaključivanje su važna za praktičnu primjenu DL u stvarnim sustavima, gdje je važno imati jamstvo da će proces zaključivanja završiti unutar razumljivog vremenskog okvira.

Različite varijante DL pružaju drugačije razine izražajnosti, koliko će detaljno i precizno informacije biti opisane. Neke proširene DL podržavaju dodatne konstruktore kao što su ograničenja kardinalnosti, inverzne uloge i tranzitivne uloge, čime je omogućeno bogatije modeliranje, ali i povećana složenost zaključivanja. Pri odabiru odgovarajuće DL važan je balans između izražajnosti jezika i učinkovitosti algoritama za zaključivanje. [1]

3.1. Preteče deskriptivnih logika

Deskriptivne logike razvijene su iz semantičkih mreža i okvira. Semantičke mreže su strukture koje prikazuju skupove objekata i njihove relacije, dok su okviri metodologije za strukturiranje informacija.

Semantičke mreže, koje su se pojavile 1960-ih godina grafički se mogu prikazati s pomoću čvorova (koji predstavljaju koncepte) i poveznica (koje predstavljaju odnose između koncepata). Također, svaki čvor može sadržavati jednostavne atribute koji dodatno opisuju taj koncept. Mreže omogućuju jednostavan prikaz odnosa između koncepata, ali imaju ograničenja u izražajnosti i formalnoj semantici. [8] Na Slici 1. može se vidjeti primjer semantičke mreže s navedenim primjerom knjižnice.



Slika 1: Semantička mreža (vlastita izrada, draw.io)

Okviri, uvedeni 1970-ih godina, su strukture za prikaz objekata i njihovih svojstava, koristeći hijerarhiju i nasljeđivanje. Omogućuju organizaciju znanja putem skupova atributa i vrijednosti, što olakšava modeliranje složenijih domena. Međutim, okviri također imaju ograničenja u formalnoj semantici i ne pružaju dovoljnu podršku za automatizirano zaključivanje. [9]

Primjer okvira je prikazan u nastavku. Dodani su jednostavni atributi za određene koncepte, npr. za koncept 'Knjiga' atribut naziv i godinu te autora, koji je također koncept te ima svoje atribute ime i listu knjiga koje je napisao. Čitatelj ima atribute ime i broj posudbi. Također čitatelj je ujedno i aktivan čitatelj ako je zadovoljen uvjet da je posudio 10 ili više knjiga. Koncept 'Aktivan čitatelj' nasljeđuje sve atribute od čitatelja, uz vlastito proširenje s atributom koji definira da je aktivan, u slučaju da je zadovoljen navedeni uvjet.

Knjiga
    imaSvojstvo - naziv (niz znakova)
    imaSvojstvo - godina (broj)
    imaSvojstvo - imaAutora (Autor)
Autor
    napisao - Knjiga
    imaSvojstvo - ime (niz znakova)
    imaSvojstvo - djela (lista knjiga)
Čitatelj
    posudio - Knjiga
    imaSvojstvo - ime (niz znakova)
    imaSvojstvo - brojPosudbi (broj)
    je - Aktivan čitatelj ako brojPosudbi >= 10
Aktivan čitatelj
    Nasljeđuje atribute iz Čitatelj
    imaSvojstvo - status (enumeracija: aktivan)

U navedenim primjerima prikazano je kod koncepta 'Aktivan čitatelj', da nasljeđuje atribute iz koncepta 'Čitatelj' te je prikazan uvjet zbog kojeg 'Čitatelj' postaje 'Aktivan čitatelj', no takva svojstva u tradicionalnom semantičkim mrežama i okvirima nije bilo jednostavno implementirati, te je stvorena potreba za razvojem naprednijeg sustava.

Ograničenjima semantičkih mreža i okvira potaknut je razvoj deskriptivnih logika. Jedan od glavnih problema bio je nedostatak formalne semantike, nisu postojali precizno definirani matematički ili logički temelji koji bi jasno specificirali značenje koncepata, odnosa i pravila zaključivanja, što je otežavalo precizno zaključivanje i provjeru konzistentnosti baze znanja.

Na primjer, ako postoji koncept "Ptica" povezan s konceptom "Može letjeti" preko veze "ima svojstvo", nije jasno mogu li sve ptice letjeti ili samo neke. Bez formalne semantike, sustav ne može pouzdano zaključiti da li konkretan primjer, poput "Noj", može letjeti ili ne. Ovo otežava automatizirano zaključivanje i može dovesti do nekonzistentnih baza znanja.

Nedostatak standardiziranih pravila za nasljeđivanje svojstava i rješavanje proturječnosti doveo je do nejasnoća u definiranju kako se svojstva prenose kroz hijerarhiju koncepata. To je često rezultiralo neočekivanim zaključcima. Na primjer, ako koncept "Čitatelj" ima ograničenje od najviše 5 posudbi, a koncept "Aktivan čitatelj", koji nasljeđuje "Čitatelj", ima definirano ograničenje od 10 posudbi, nije jasno je li za "Aktivnog čitatelja" maksimalan broj posudbi 5 ili 10, da bi bio tako definiran, kao "Aktivnog čitatelja".

Tradicionalne semantičke mreže i okviri pružali su osnovne metode za reprezentaciju znanja, međutim pojavili su se problemi koji su ograničili njihovu upotrebljivost i efikasnost. Objašnjeno je na jednostavnom primjeru kako se radi zaključivanje na temelju mreže, međutim, ako veze između čvorova nisu precizno definirane, to dovodi do nejasnoća prilikom zaključivanja, što je bio problem kod tradicionalnih semantičkih mreža. Mnoge semantičke mreže nisu podržavale složene logičke konstrukte poput ograničenja (∃, ∀) i brojčanih ograničenja (≥, ≤), što je ograničilo njihovu sposobnost za precizno modeliranje složenih domena i odnosa između koncepata. [2]

Kako bi se prevladali navedeni problemi, te pružila preciznija reprezentacija znanja, razvijen je KL-ONE sustav. Prvi put je implementiran 1977. godine, te se koristi u AI području. KL-ONE je bio jedan od prvih sustava koji je uveo formalnu semantiku u reprezentaciju znanja, pružajući precizne definicije koncepata i odnosa među njima. Koristi specifičan pristup nazvan "epistemološka razina" za upravljanje konceptima kao što su "opis", "atribut", "koncept", "uloga", "nasljeđivanje" i "instanciranje", koji su bili tretirani na neformalan način u navedenim sustavima.

KL-ONE koristi jasno definirane semantičke uloge, eliminirajući nejasnoće povezane sa semantičkim mrežama. Koncepti u KL-ONE su hijerarhijski organizirani, omogućavajući nasljeđivanje svojstava od nadređenih koncepata, što olakšava proširivanje i održavanje baze znanja. Za razliku od tradicionalnih semantičkih mreža i okvira, KL-ONE podržava složene logičke konstrukte poput egzistencijalnih i univerzalnih ograničenja (∃, ∀) i brojčanih ograničenja (≥, ≤). Time je omogućeno precizno modeliranje složenih odnosa između koncepata. [3] Na primjer, KL-ONE može definirati da 'Aktivan čitatelj' mora imati najmanje 10 posuđenih knjiga.

Danas se KL-ONE koncepti koriste u području umjetne inteligencije. Mnoge ideje iz KL-ONE sustava integrirane su u moderne ontološke jezike poput OWL-a, koji se koriste za reprezentaciju složenih domena znanja na semantičkom webu. Razvoj KL-ONE sustava označio je korak naprijed u prikazu znanja, pružajući formalnu semantiku i alate za

modeliranje. Time je omogućeno rješavanje problema nejasnoća i nedosljednosti prisutnih u semantičkim mrežama i okvirima te su postavljeni temelji za daljnji razvoj deskriptivnih logika i naprednih sustava za zaključivanje. [4] Zbog prednosti nad semantičkim mrežama i okvirima, KL-ONE postaje preteča deskriptivnih logika (DL).

3.2. Uporaba deskriptivnih logika

Jedna od glavnih prednosti deskriptivnih logika je njihova sintaksa, koja je intuitivna i bliska prirodnom jeziku. DL jezici koriste jednostavne i lako razumljive konstruktore poput konjunkcije (⊓), disjunkcije (⊔), negacije (¬), egzistencijalno (∃) i univerzalno (∀) ograničenje. Bez potrebe za dubokim tehničkim znanjem, korisnici mogu lagano razumjeti i koristiti DL jezike za opisivanje složenih domena. [5] Primjer lako razumljivog i intuitivnog izraza je 'Ivan Horvat posudio Dječak koji je volio jelene'. Ovaj se izraz može formalizirati u DL na sljedeći način:

$Čitatelj(IvanHorvat)$ → Ivan Horvat pripada konceptu Čitatelj.

$Knjiga(DječakKojiJeVolioJelene)$ → Dječak koji je volio jelene pripada konceptu Knjiga.

$posudio(IvanHorvat, DječakKojiJeVolioJelene)$ → Postoji relacija "posudio" između Ivan Horvat i Dječak koji je volio jelene.

Ovi jednostavni izrazi omogućuju jasno definirane činjenice u domeni knjižnice, što olakšava zaključivanje i obradu informacija.

Također, još jedna od prednosti DL jezika je ta da se mogu implementirati koristeći grafičke prikaze. Vizualizacija DL ontologija putem dijagrama klasa, hijerarhija i odnosa pomaže korisnicima da bolje razumiju strukturu i dinamiku domene. Grafičkim prikazom olakšano je identificiranje veza između koncepata i moguće je lakše ukazati na potencijalne nedosljednosti u bazi znanja. Vizualni prikaz koncepata i njihovih odnosa pomaže prilikom obrađivanja složene domene. [6] Protégé je primjer alata koji omogućuje izradu i pregled ontologija koristeći grafička sučelja.

Izrada ontologije je jedan od čestih tipova grafičkog prikaza implementacije DL jezika. Ontologije definiraju skup koncepata i odnosa koji postoje u promatranoj domeni, pružajući formalnu specifikaciju znanja. Korištenjem DL jezika za izradu ontologija osigurana je preciznost i mogućnost automatiziranog zaključivanja, što je ključno za aplikacije koje zahtijevaju pouzdano i učinkovito upravljanje znanjem. [10]

Ontologija je formalni, eksplicitni opis koncepata (klasa) koji mogu sadržavati specifične atribute (svojstva) i ograničenja nad svojstvima. Dodavanjem instanci klasa ontologiji, kreirana je baza znanja. Instance su konkretni primjeri koncepata koji sadrže vrijednosti svojstava. Na ontologiju se može gledati kao neku vrstu okvira bez sadržaja, a dodavanjem realnog sadržaja toj ontologiji, kreirana je baza znanja. Ontologija definira pravila i strukture, dok baza znanja

sadrži podatke koji su u skladu s definiranim pravilima. Klase su središnji element većine ontologija, one opisuju koncepte definirane u domeni. Mogu biti organizirane hijerarhijski, gdje potklase nasljeđuju svojstva od natklasa. [11] Klasa knjiga predstavlja sve knjige, dok bi knjiga 'Dječak koji je volio jelene' bila specifična instanca klase knjiga. Na primjer, moguće je podijeliti klasu svih knjiga na beletristiku (romani, pripovjetke...) i publicistiku (novine, časopisi...). Također, klasa knjiga se može podijeliti prema žanrovima poput fantastike, kriminalistike i horora. Ova podjela omogućuje preciznije kategoriziranje knjiga i olakšava pretraživanje i zaključivanje unutar baze znanja. Svojstva opisuju karakteristike klasa i instanci: knjiga "Dječak koji je volio jelene" napisana je od autora Samuela Bjorka te je izdana 2018. godine. Dva svojstva opisuju knjigu u ovom primjeru, a to su autor i godina izdanja. Sve instance klase knjiga, i njezinih potklasa, imaju svojstvo autor čija je vrijednost instanca klase autor. Sve instance klase autor imaju svojstvo napisao koje se odnosi na sve knjige (instance klase knjiga i njezinih potklasa) koje je autor napisao. Dvosmjerni odnos između autora i knjiga omogućuje lako pronalaženje svih djela određenog autora ili identificiranje autora neke knjige. Ontologije izrađene pomoću deskriptivnih logika omogućuju automatsko zaključivanje, kao što je spomenuto. Na primjer, ako je "Dječak koji je volio jelene" knjiga iz žanra kriminalistike, a Ivan Horvat voli kriminalističke knjige, sustav može zaključiti da bi Ivan Horvat mogao biti zainteresiran za posudbu ove knjige.

Ontologije se najčešće pohranjuju u formatima OWL (Web Ontology Language) i RDF (Resource Description Framework). OWL je jezik za izradu i razmjenu ontologija na semantičkom webu, koji se temelji na deskriptivnim logikama. [12] RDF je okvir za opisivanje resursa na webu putem subjekta, predikata i objekta, omogućujući strukturirano i međusobno povezano predstavljanje podataka. [13] U ovom radu izrađena je vlastita ontologija, na temu „*RuneScape*“ (MMORPG video igre), te je provedeno zaključivanje nad njom.

Govoreno je o zaključivanju u kontekstu knjižnice, gdje je korišten primjer poput određivanja statusa aktivnog čitatelja na temelju broja posuđenih knjiga. Taj primjer pokazao je kako se može izvesti implicitno znanje iz eksplicitno definiranih podataka. Zaključivanje je proces izvođenja novih informacija iz postojećih podataka korištenjem formalnih logičkih metoda. Da bi se takvo zaključivanje automatiziralo i primijenilo na složenije baze znanja, koriste se specijalizirani alati za zaključivanje.

Alati za zaključivanje omogućuju automatizirano zaključivanje, provjeru konzistentnosti, klasifikaciju koncepata i druge analitičke funkcije. Koriste formalne metode kako bi izveli nove informacije iz postojećih podataka u ontologijama. Poznati alati su : ELK, Pellet, HermiT, jcel, Ontop i mnogi drugi. [7] Ovi alati implementiraju razne algoritme zaključivanja i podržavaju određene dijelove OWL standarda. Optimizirani su za specifične zadatke i tipove ontologija. Različiti alati postoje jer su optimizirani za drugačije scenarije korištenja i vrste ontologija. Na primjer, neki alati su dizajnirani za rad s velikim i složenim

ontologijama koje zahtijevaju napredne algoritme za zaključivanje (npr. HermiT), dok su drugi optimizirani za brzinu i efikasnost u realnom vremenu s jednostavnijim ontologijama (npr. ELK). HermiT koristi napredne tehnike poput hypertableau algoritma za učinkovito zaključivanje nad vrlo izražajnim ontologijama. [16] ELK je alat za zaključivanje optimiziran za OWL 2 EL profil, pružajući vrlo brzo zaključivanje nad velikim ontologijama s ograničenom izražajnošću. [17] U kontekstu deskriptivnih logika i alata za zaključivanje, izražajnost se odnosi na sposobnost jezika ili ontologije da precizno i detaljno modelira složene koncepte, odnose i ograničenja unutar neke domene. Također, neki ali za zaključivanje podržavaju specifične jezike ili formate ontologija, kao što je OWL 2 EL (npr. jcel), dok drugi pružaju širu podršku za cijeli spektar OWL jezika (npr. Pellet). Pellet je alat koji u potpunosti podržava OWL 2 DL i omogućuje provjeru konzistentnosti, klasifikaciju i druge napredne funkcije. [18] Raznolikost alata omogućuje korisnicima odabir onih koji najbolje odgovara njihovim potrebama u pogledu performansi, izražajnosti i kompatibilnosti s određenim standardima. [19]

U nastavku rada detaljnije će se obraditi navedeni alati za zaključivanje, njihove karakteristike i usporedba njihovih performansi i efikasnosti u zaključivanju.

4. Jezici deskriptivnih logika

Atributivni jezik ($\mathcal{AL}$) je osnovni jezik deskriptivnih logika, koji služi za gradnju svih drugih, složenijih jezika. Ovaj jezik prvi puta je spomenut kao minimalni jezik koji ima praktičnu primjenu [Schmidt-Schauß i Smolka, 1991]. Koncepti ovog osnovnog jezika su:

1) Atomarni koncept (A) najosnovniji je koncept koji predstavlja osnovnu klasu entiteta bez ikakvih dodatnih ograničenja.
2) Univerzalni koncept ($\top$) predstavlja klasu koja uključuje sve entitete u domeni, bez izuzetaka.
3) Prazan koncept ($\bot$) predstavlja klasu koja ne sadrži niti jedan entitet.
4) Atomarna negacija ($\neg A$) negira atomarni koncept A.
5) Presjek koncepata ($C \sqcap D$) kombinira koncepte tako da se uključe samo entiteti koji pripadaju u obje klase.
6) Univerzalno ograničenje ($\forall R.C$) znači da svaki entitet povezan s ulogom R mora biti dio koncepta C.
7) Ograničeno egzistencijalno ograničenje ($\exists R.\top$) znači da postoji barem jedna veza uloge R prema bilo kojem entitetu.

Korištenjem osnovnih $\mathcal{AL}$ koncepata mogu se graditi složeni opisi koncepata koji precizno definiraju klasu entiteta, npr. klase student i profesor:
$Student\ je\ Osoba \sqcap \exists sluša.Kolegij$, a $Profesor\ je\ Osoba \sqcap \forall predaje.Kolegij$.
Može se primijetiti kako $\mathcal{AL}$ nema definirana pravila za disjunkciju, niti univerzalnu negaciju, samo atomarnu. Kako se stvorila potreba za modeliranjem složenih domena znanja, to je postao problem te je bilo potrebno proširiti ovaj osnovni jezik. [1]

4.1. Temeljni jezik deskriptivne logike

Jezik koji uvodi disjunkciju i univerzalnu negaciju je $\mathcal{ALC}$ (engl. *Attributive Language with Complements*), temeljni jezik deskriptivnih logika koji se koristi za zaključivanje.
Koncepti $\mathcal{ALC}$ jezika su:

1) Konjunkcija ($C \sqcap D$) predstavlja entitete koji su u C i D.
2) Disjunkcija ($C \sqcup D$) predstavlja entitete koji su ili u C ili u D.
3) Negacija ($\neg C$) predstavlja entitete koji nisu u C.
4) Univerzalno ograničenje ($\forall R.C$) znači da svi entiteti povezani u ulozi R moraju biti u C.
5) Egzistencijalno ograničenje ($\exists R.C$) znači da postoji barem jedan entitet u ulozi R koji je u C.

Prethodni primjer student/profesor, u $\mathcal{ALC}$ može se iskazati preko znaka ekvivalencije ≡.
$Student \equiv Osoba \sqcap \exists sluša.Kolegij$, student je osoba koja je upisana na barem jedan kolegij.
$Profesor \equiv Osoba \sqcap \forall predaje.Kolegij$, profesor je osoba koja predaje kolegije svugdje gdje je to definirano ulogom $predaje$.
$\neg Student$, svi entiteti koji nisu studenti.
$\mathcal{ALC}$ predstavlja značajno proširenje u odnosu na $\mathcal{AL}$ jezik, međutim kako s vremenom $\mathcal{AL}$ nije bio dovoljan, tako nije ni $\mathcal{ALC}$, pa su razvijena proširenja $\mathcal{ALC}$ jezika. [1]

4.2. Proširenja deskriptivne logike $\mathcal{ALC}$

Svako proširenje omogućuju bolje modeliranje odnosa i ograničenja unutar same logike. Neka od najznačajnijih proširenja $\mathcal{ALC}$ logike su:

1) Tranzitivne uloge (S) → Omogućuju prenošenje odnosa između entiteta u više koraka, primjer Maja ima potomka Niku, a Nika ima potomka Daria, zaključak je da Maja ima potomka Daria.
2) Hijerarhija uloga (H) → Omogućuje definiranje hijerarhija među ulogama, jedna uloga je podskup druge uloge.
3) Kardinalnost (N) → Definira broj veza koje entitet može imati prema drugima u ulozi.
4) Brojevna ograničenja (Q) →Koristi se kod definiranja više brojčanih ograničenja, u jednom izrazu manje i više od.
5) Funkcionalne uloge (F) → Definira da uloga može imati samo jednu vrijednost za svaku instancu, primjer osoba može imati samo jednu majku.
6) Inverzne uloge (I) → Definiraju relacije u suprotnom smjeru, Ivan jeKupio loptu, inverzno bi bilo da je Lopta kupljenaOd Ivana.
7) Podatkovne vrijednosti (D) → Omogućen rad s konkretnim podatkovnim tipovima kao što su integer, string, double...
8) Nominalni koncepti (O) →Omogućuje uključivanje konkretnih, imenovanih entiteta unutar logike.

Neka od proširenja koja koriste alati za zaključivanje, a koje ćemo usporediti te detaljnije obraditi u nastavku, kod same obrade alata, su: SHOIN(D), SROIQ(D), EL+, EL i QL. [1]

5. Tableau algoritam

Tableau algoritmi su tehnike provjere zadovoljavanja tvrdnje primjenom pravila za kreiranje modela, ili njegove reprezentacije. Koriste se dugi niz godina u područjima logike prvog reda (FOL) i modalnih logika (ML). Od kasnih 90-ih godina 20. stoljeća koriste se i u područjima DL-a. Mnogi DL alati za zaključivanje implementiraju Tableau tehnike, na primjer Racer, FaCT++, Pellet, HermiT ili jcel. HermiT je primjer alata koji implementira poboljšani Tableau algoritam, Hypertableau. Prije provedbe usporedbe navedenih alata za zaključivanje, u ovom poglavlju fokus je na Tableau algoritam za $\mathcal{ALC}$, temeljni jezik deskriptivnih logika, i prikaz kako se koristi za testiranje zadovoljivosti $\mathcal{ALC}$ koncepata. [20]

Prvo će se opisati algoritam koji odlučuje o zadovoljivosti $\mathcal{ALC}$ koncepata, a zatim na konkretnim primjerima pojasniti opisano.

Za zadani $\mathcal{ALC}$ koncept , algoritam pokušava izgraditi konačni grafički prikaz modela I koji zadovoljava , odnosno sadrži element takav da je $\in$. Prilikom izrade grafičke reprezentacije modela I, koncepti čine čvorove grafa, a uloge lukove. Počevši s $:= \{()\}$ algoritam primjenjuje pravila transformacije sve dok se ne zadovolje sva ograničenja ($ABox$ tvrdnje se promatraju kao ograničenja) ili dok se ne otkrije kontradikcija. Svakom primjenom pravila ili se dodaju novi čvorovi ili se postojeći čvorovi proširuju. Graf je zadovoljiv čim je jedna od $ABox$ tvrdnji otvorena, odnosno ne sadrži kontradikciju.

Prije same primjene pravila transformacije, kao prvi postupak provođenja procedure Tableau algoritma, potrebno je da bude u negacijskoj normalnoj formi (NNF). [23] logički izraz je u NNF ako se negacija pojavljuje samo uz atomarne elemente izraza, a u izrazu se koristi samo negacija (¬), disjunkcija (∨) ili konjunkcija (∧). Pravila koja primjenjujemo kako bi logički izraz bio u NNF su:

Tablica 1: Pravila za NNF

$\neg\neg C$	$\equiv$	C
$\neg(C \sqcup D)$	$\equiv$	$\neg C \sqcap \neg D$
$\neg(C \sqcap D)$	$\equiv$	$\neg C \sqcup \neg D$
$\neg(\forall R.C)$	$\equiv$	$\exists R.\neg C$
$\neg(\exists R.C)$	$\equiv$	$\forall R.\neg C$

i $nnf()$ su ekvivalentni, odnosno $=$ u svakoj interpretaciji I. [21]

U prvom koraku kreira se inicijalni graf za , koji sadrži samo jedan čvor i nema lukova. Zatim se primjenjuju pravila transformacije sve dok se ne dođe do pod-grafa (čvora unutar inicijalnog grafa; svaki čvor se može gledati kao zaseban graf) u kojem nema kontradikcije i ne postoje više pravila koja se mogu primijeniti. Ako u svim granama postoji kontradikcija, algoritam završava i početna tvrdnja kojase dokazuje je potvrđena. Kontradikcija se događa

kada se za određeni koncept C naiđe na njegovu negaciju, odnosno u istoj grani grafa se nalaze $\{C, \neg C\}$. Graf je dovršen kada se na njega ne može primijeniti više niti jedno pravilo transformacije.

5.1. Transformacijska pravila

Prilikom provođenja algoritma, izraze u $nnf()$ je potrebno transformirati, na temelju pravila za transformaciju konjunkcije, disjunkcije, egzistencijalno i univerzalno ograničenje.

5.1.1. Pravilo za konjunkciju

Tablica 2: Pravilo za konjunkciju

$\sqcap - Pravilo$:		
Ako	Značenje	Primjer
$(C_1 \sqcap C_2)(x) \in A$ i $\{C_1(x), C_2(x)\} \nsubseteq A$	Klasa i instancirana s individuom x je dio $ABox(A)$, ali jedan ili oba dijela konjunkcije (x) i (x) još nisu eksplicitno dodani u $ABox(A)$.	U $ABox(A)$ je prisutna (x), a postoji tvrdnja $(C_1 \sqcap C_2)(x)$, onda se prijenjuje pravilo i dodaje se (x) u $ABox(A)$. Ako su (x) i (x) već prisutni u $ABox(A)$ onda ovo pravilo neće biti primijenjeno.
Onda	Značenje	Primjer
$:= A \cup \{C_1(x), (x)\}$	Moguće je proširiti postojeći $ABox(A)$ s klasama i od individue x	Akcija koja proizlazi iz pravila je da se u $ABox(A)$ doda (x) i $C_2(x)$, jer konjunkcija znači da oba koncepta moraju biti istinita za individuu x.

[22]

5.1.2. Pravilo za disjunkciju

Tablica 3: Pravilo za disjunkciju

$\sqcup - Pravilo$:		
Ako	Značenje	Primjer

$(C_1 \sqcup C_2)(x) \in A$ i $\{C_1(x), C_2(x)\} \cap A = \emptyset$	Klasa ili instancirana s individuom x je dio $ABox(A)$, ali niti jedan od dijelova disjunkcije (x) i (x) još nije eksplicitno dodan u $ABox(A)$, odnosno presjek (x) i (x) i $ABox(A)$ je prazan skup.	Ako je u $ABox(A)$ tvrdnja $(C_1 \sqcup C_2)(x)$, ali ni $C_1(x)$ ni $C_2(x)$ nisu dodani u $ABox(A)$, pravilo dodaje jednu od tih tvrdnji (ali ne obje) u $ABox(A)$.
Onda	Značenje	Primjer
$:= A \cup \{C_1(x)\}$, $:= A \cup \{C_2(x)\}$	Prilikom primjene pravila, dolazi do grananja u stablu. Postojeći $ABox(A)$ proširuje se na dva načina: $C_1(x)$ se dodaje u jednu granu, a (x) u drugu, za individuu x. Postoji nedeterministički izbor (odabire se samo jedna tvrdnja) koju tvrdnju dodati.	Akcija koja proizlazi iz pravila znači da se proširuje $ABox(A)$ dodavanjem ili (x) ili (x), ali nikako oboje. Taj izbor je nedeterministički, pa će se kasnije vidjeti hoće li na kraju koje grane $ABox(A)$ biti zadovoljen.

[22]

5.1.3. Pravilo za egzistencijalno ograničenje

Tablica 4: Pravilo za egzistencijalno ograničenje

$\exists - Pravilo$:		
Ako	Značenje	Primjer
$(\exists R.C)(x) \in A$, ali nema individue z takve da su $C(z)$ i $R(x,z)$ u A	Klasa C povezana s individuom x egzistencijalnom relacijom R zahtijeva postojanje individue z za koju vrijedi $C(z)$ i $R(x,z)$, te takva individua još ne postoji u $ABox$-u.	Ako je u $ABox(A)$ prisutno egzistencijalno ograničenje $(\exists R.C)(x)$, ali još uvijek ne postoji individua z koja zadovoljava $C(z)$ i relaciju $R(x,z)$, mora se dodati nova individua y koja zadovoljava $C(y)$ i s pomoću relacije $R(x,y)$ biti povezana s individuom x.
Onda	Značenje	Primjer
$:= A \cup \{C(y),\ R(x,y)\}$, gdje je y nova individua koja se ne pojavljuje u $ABox$-u.	Dodaje se nova individua y u $ABox$ i tvrdi da je y dio klase C te da postoji relacija R između x i y. Tako se zadovoljava egzistencijalno ograničenje.	Akcija koja proizlazi iz pravila je da se u $ABox$ doda nova individua y, za koju se tvrdi da je $C(y)$ istinito i da relacija $R(x,y)$ postoji. Ovo

		proširuje graf dodavanjem novog čvora i relacije.

[22]

Razlika između z i y je ta da se s z označava moguća postojeća individua koju već tražimo u $ABox$-u, a y označava novu individuu koja se dodaje u $ABox$ ako takva individua već ne postoji.

5.1.4. Pravilo za univerzalno ograničenje

Tablica 5: Pravilo za univerzalno ograničenje

$\forall - Pravilo$:		
Ako	Značenje	Primjer
$(\forall R.C)(x) \in A$ i $R(x,y) \in A$, ali $C(y) \notin A$	Klasa C mora vrijediti za svaku individuu y koja je povezana s x putem relacije $R(x,y)$, ali $C(y)$ još nije dodan u $ABox$.	Ako x ima relaciju R s y, a postoji univerzalna tvrdnja da sve individue povezane s x putem R moraju zadovoljiti C, ali $C(y)$ nije dodan u $ABox$, tada pravilo zahtijeva dodavanje $C(y)$.
Onda	Značenje	Primjer
$:= A \cup \{C(y)\}$	Dodaje se tvrdnja $C(y)$ u $ABox$, jer je C koncept koji mora vrijediti za sve individue y koje su povezane s x putem relacije $R(x,y)$.	Akcija koja proizlazi iz pravila je da se doda tvrdnja $C(y)$ za individuu y, jer je univerzalno ograničenje $\forall R.C$ primijenjena na sve individue y povezane s x.

[22]

5.1.5. TBox pravilo

Dosadašnje objašnjenje Tableau algoritma za $\mathcal{ALC}$ odnosilo se na rad s $ABox$-om, gdje se koriste konkretne činjenice o pojedinačnim instancama, poput tvrdnji da je "Lajka pas" ili "Miško je mačka". U takvim slučajevima, Tableau algoritam provjerava zadovoljavaju li ove tvrdnje odgovarajuće koncepte bez znanja o međusobnim odnosima tih koncepata. Međutim, u praksi, često se radi s dodatnim pravilima koja definiraju odnose među konceptima – takva pravila se pohranjuju u $TBox$-u.

$TBox$ sadrži hijerarhije i odnose između koncepata, kao što su "Svi psi su životinje" ili "Svi kućni ljubimci imaju vlasnike", dok $ABox$ sadrži konkretne činjenice o instancama tih koncepta. Proširenje Tableau algoritma na $TBox$ omogućava da se uzmu u obzir ove relacije prilikom zaključivanja, čime algoritam postaje sposobniji za složenije zadatke, jer može koristiti i pravila iz $TBox$-a, a ne samo pojedinačne instance iz $ABox$-a.

Za rješavanje situacija koje uključuju $TBox$, potrebno je uvesti novo pravilo u set pravila za transformacije, $\sqsubseteq - Pravilo$. Ovo pravilo se koristi za primjenu odnosa između klasa unutar $TBox$-a, poput pravila da je "Pas podskup Životinja". Dok $ABox$ grafovi često imaju oblik jednostavnih stabala, u slučaju kada uključujemo $TBox$, dobivamo grafove koji su složeniji, u obliku šuma (više stabala) i koji koriste dodatne relacije i hijerarhije između koncepta.

Tablica 6: TBox pravilo

$\sqsubseteq - Pravilo$:		
Ako	Značenje	Primjer
$C_1(x) \in A$ i $\sqsubseteq \in T$	Ako je individua x član klase , a iz TBox-a se zna da je potklasa od , tada se dodaje tvrdnja da je x član klase .	Ako je u $ABox$-u prisutna tvrdnja $Pas(Lajka)$, a iz TBox-a se zna da $Pas \sqsubseteq Životinja$, tada se dodaje $Životinja(Lajka)$ u $ABox$.
Onda	Značenje	Primjer
$:= A \cup \{C_2(x)\}$	$ABox$ se proširuje tvrdnjom da je individua x član klase , jer je $\sqsubseteq$ iz TBox-a.	Ako je $Pas(Lajka)$ istinito, a zna se da su svi psi životinje ($Pas \sqsubseteq Životinja$), tada se u $ABox$ dodaje $Životinja(Lajka)$.

Potrebno je osigurati prekid algoritma kako bi se spriječilo beskonačno generiranje novih čvorova u grafu. To se postiže mehanizmom blokiranja. [23] Algoritam detektira cikluse u grafu prepoznavanjem čvorova s istim oznakama i zaustavlja daljnje grananje za te čvorove. Ova tehnika sprječava beskonačne petlje u grafu, koje mogu nastati zbog egzistencijalnog ograničenja ili hijerarhijskih odnosa u $TBox$-u. Pretpostavka je da imamo $TBox$: $Roditelj \sqsubseteq \exists imaDijete.Roditelj$, ova tvrdnja znači da svaki roditelj mora imati dijete koje je također roditelj, te da postoji $ABox$: $Roditelj(Marko)$, tvrdnja da je Marko roditelj. Kada se krene provoditi Tableau algoritam početni $ABox$ bi bio $= \{Roditelj\}$. Primjenom $\exists - pravila$ na $Roditelj(Marko)$, po $TBox$-u se zna da Marko mora imati dijete koje je roditelj, dakle dodaje se nova individua Ivan i tvrdi da $imaDijete(Marko, Ivan)$ i $Roditelj(Ivan)$. Dobiva se da $= \{Roditelj(Marko), imaDijete(Marko, Ivan), Roditelj(Ivan)\}$, a kako je Ivan roditelj opet bi se morala primijeniti $\exists - pravila$. Bez blokiranja ovaj bi proces generirao beskonačan broj novih

individua, jer svaki roditelj mora imati dijete koje je također roditelj. Ovaj postupak osigurava da se algoritam prekida kada se pojave ciklusi u hijerarhijskim strukturama, npr. kod roditelja i djece.

5.2. Tableau algoritam kroz primjer

Sada će biti prikazati kako Tableau algoritam funkcionira u praksi. Kroz konkretan primjer, primijeniti će se pravila transformacije na definirane tvrdnje i pokazati kako se dolazi do zaključka. Proces koji će se proći osnova je za ono što svaki alat za zaključivanje koristi prilikom provjere konzistentnosti i zaključivanja nad deskriptivnim logikama.

Primjer započinje definiranjem sustava pravila, odnosno $TBox$-a, koji sadrži klasične opise i relacije:

$Životinja \sqsubseteq Entitet$, (Svaka životinja je entitet)
$Osoba \sqsubseteq Entitet$, (Svaka osoba je entitet)
$Pas \equiv Životinja \sqcap Laje$, (Pas je životinja koja laje)
$Mačka \equiv Životinja \sqcap \neg Pas$, (Mačka je životinja koja nije pas)
$Štene \equiv Pas \sqcap Mlad$, (Štene je mlad pas)$PasLjubimac \equiv Pas \sqcap \exists imaVlasnika.Osoba$, (Pas ljubimac je pas koji ima vlasnika koji je osoba)
$MačkaLjubimac \equiv Mačka \sqcap \exists imaVlasnika.Osoba$, (Mačka ljubimac je mačka koja ima vlasnika koji je osoba)
$KućniLjubimac \equiv PasLjubimac \sqcup MačkaLjubimac$, (Kućni ljubimac je pas ljubimac ili mačka ljubimac)
$OtacŠteneta \equiv PasLjubimac \sqcap \exists imaMladunče.Štene$, (Otac šteneta je pas ljubimac koji ima mladunče koje je štene)
$OdanOtacŠteneta \equiv OtacŠteneta \sqcap \exists brineSeZa.Štene$, (Odan otac šteneta je otac šteneta koji brine za svoje štene)

Nakon što je $TBox$ definiran sada je moguće provjeriti određene tvrdnje, a to se radi koristeći Tableau algoritam. Tvrdnja koja će se u ovom prvom primjeru provjeriti: Je li svaki odan otac šteneta automatski kućni ljubimac, odnosno $OdanOtacŠteneta \sqsubseteq KućniLjubimac$?

Tableau algoritam će negirati ovu tvrdnju i pokušati dokazati kontradikciju. Je li moguće pronaći odanog oca šteneta koji nije kućni ljubimac, odnosno $OdanOtacŠteneta \sqsubseteq KućniLjubimac \rightarrow OdanOtacŠteneta \sqcap \neg KućniLjubimac$? Ako se ispostavi da je ovakva situacija nezadovoljiva, odnosno da vodi do kontradikcije, može se zaključiti da svaki odan otac šteneta mora biti ujedno i kućni ljubimac.

Prvi korak je priprema, odnosno potrebno je odrediti $nnf()$. Izraze u tvrdnji koja se dokazuje $= OdanOtacŠteneta \sqcap \neg KućniLjubimac$ potrebno je razraditi korak po korak, odnosno do atomarnih komponenti koje su definirane u $TBox$-u.

1) $OdanOtacŠteneta \equiv OtacŠteneta \sqcap \exists brineSeZa.Štene$

2) $OdanOtacŠteneta \equiv PasLjubimac \sqcap \exists imaMladunče.Štene \sqcap \exists brineSeZa.Štene$

3) $OdanOtacŠteneta \equiv Pas \sqcap \exists imaVlasnika.Osoba \sqcap \exists imaMladunče.Štene \sqcap \exists brineSeZa.Štene$

4) $OdanOtacŠteneta \equiv Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba \sqcap \exists imaMladunče.(Životinja \sqcap Laje \sqcap Mlad) \sqcap \exists brineSeZa.(Životinja \sqcap Laje \sqcap Mlad)$

1) $KućniLjubimac \equiv PasLjubimac \sqcup MačkaLjubimac$

2) $KućniLjubimac \equiv (Pas \sqcap \exists imaVlasnika.Osoba) \sqcup (Mačka \sqcap \exists imaVlasnika.Osoba)$

3) $KućniLjubimac \equiv (Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap \neg Pas \sqcap \exists imaVlasnika.Osoba)$

4) $KućniLjubimac \equiv (Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap \neg(Životinja \sqcap Laje) \sqcap \exists imaVlasnika.Osoba)$

5) $KućniLjubimac \equiv (Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba)$

6) $\neg KućniLjubimac \equiv \neg(Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcap \neg(Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba)$

7) $\neg KućniLjubimac \equiv (\neg Životinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)$

$nnf() = Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba \sqcap \exists imaMladunče.(Životinja \sqcap Laje \sqcap Mlad) \sqcap \exists brineSeZa.(Životinja \sqcap Laje \sqcap Mlad) \sqcap (\neg Životinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)$

Određen je $nnf()$, što je početni $ABox$: $= \{nnf()(x)\}$

$= \{(Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba \sqcap \exists imaMladunče.(Životinja \sqcap Laje \sqcap Mlad) \sqcap \exists brineSeZa.(Životinja \sqcap Laje \sqcap Mlad) \sqcap (\neg Životinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba))(x)\}$

Prvo se primjenjuje $\sqcap - pravilo$; pravilo za konjunkciju na cijeli .

$= \cup \{(Životinja)(x), (Laje)(x), (\exists imaVlasnika.Osoba)(x), (\exists imaMladunče.(Životinja \sqcap Laje \sqcap Mlad))(x), (\exists brineSeZa.(Životinja \sqcap Laje \sqcap Mlad))(x), (\neg Životinja \sqcup \neg Laje \sqcup$

$\forall imaVlasnika.\neg Osoba)(x),\ (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup$
$\forall imaVlasnika.\neg Osoba)(x)\}$

Zatim se primjenjuje $\exists - pravilo$; pravilo za egzistencijalno ograničenje na $(\exists imaVlasnika.Osoba)(x)$ iz .

$= \cup \{(imaVlasnika(x,y),\ (Osoba)(y))\}$

Opet se primjenjuje pravilo za egzistencijalno ograničenje, na sljedeći izraz iz .

$= \cup \{(imaMladunče(x,z),\ (Životinja \sqcap Laje \sqcap Mlad)(z))\}$

Zatim se primjenjuje pravilo za konjunkciju na A_3. kojom rješavamo do kraja izraz iz prethodnog koraka .

$= \cup \{(Životinja)(z),\ (Laje)(z),\ (Mlad)(z))\}$

Nastavlja se dalje, i opet primjenjuje pravilo za egzistencijalno ograničenje.

$= \cup \{(brineSeZa(x,v),\ (Životinja \sqcap Laje \sqcap Mlad)(v))\}$

Ponovno se provodi pravilo za konjunkciju.

$= \cup \{(Životinja)(v),\ (Laje)(v),\ (Mlad)(v))\}$

Dolazi se do prvog grananja, jer se primjenjuje $\sqcup - pravilo$; pravilo za disjunkciju na .

$= \cup \{(\neg Životinja)(x)\}$ **kontradikcija** u grani

U prvoj grani postoji kontradikcija, jer u postoji $(Životinja)(x)$, a sada u $(\neg Životinja)(x)$. Ova grana se zatvara i prebacuje se na iduću.

$= \cup \{(\neg Laje \sqcup \forall imaVlasnika.\neg Osoba)(x)\}$

Opet se izraz grana, odnosno primjenjuje se pravilo za disjunkciju na .

$= \cup \{(\neg Laje)(x)\}$ **kontradikcija** u grani

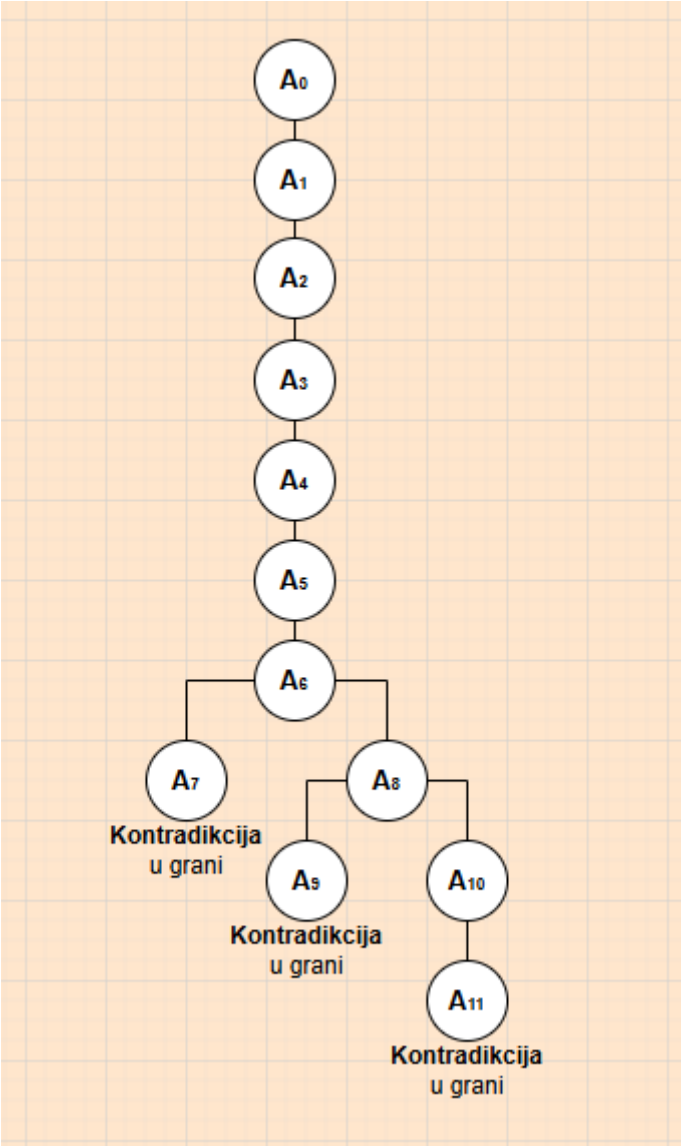
U novoj grani postoji kontradikcija, jer $(Laje)(x)$ postoji u , a sada u $(\neg Laje)(x)$. Grana se zatvara i prebacuje se na iduću.

$= \cup \{(\forall imaVlasnika.\neg Osoba)(x)\}$

Sada se primjenjuje $\forall - pravilo$; pravilo za univerzalno ograničenje na .

$= \cup \{(\neg Osoba)(y)\}$ **kontradikcija** u grani

Kako $(Osoba)(y)$ postoji u , a sada u $(\neg Osoba)(y)$ grana se zatvara zbog kontradikcije. Zatvorene su sve grane, u svakoj grani se nalazi kontradikcija, nema slobodnih interpretacija, tako da se zaključuje kako početni izraz $OdanOtacŠteneta \sqcap \neg KućniLjubimac$ **je nezadovoljiv**. To znači da početna tvrdnja $OdanOtacŠteneta \sqsubseteq KućniLjubimac$ je zadovoljiva, odnosno da je potvrđeno kako svaki odan otac šteneta ujedno i je kućni ljubimac. Vizualni prikaz ovog primjera vidljiv je na Slici 2.



Slika 2: Grananje za prvi primjer (vlastita izrada, draw.io)

U prvom primjeru postoji kontradikcija u svakoj grani, međutim prikazati će se još par primjera u kojima možda to neće biti slučaj. Iduća tvrdnja će provjeriti je li svaki kućni ljubimac automatski i mačka ljubimac, odnosno $KućniLjubimac \sqsubseteq MačkaLjubimac$? Postupak je isti kao i u prethodnom primjeru, pitanje je postoji li kućni ljubimac koji nije mačka ljubimac: $KućniLjubimac \sqsubseteq MačkaLjubimac \rightarrow KućniLjubimac \sqcap \neg MačkaLjubimac$. Potrebno je odrediti početni $ABox$, iz prethodnog primjera je već pojednostavljen $KućniLjubimac$, pa će se taj izraz iskoristiti za ovaj primjer, a pojednostavljen je i izraz $MačkaLjubimac$, samo ga je potrebno negirati.

1) $KućniLjubimac \equiv (Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba)$

1) $\neg MačkaLjubimac \equiv \neg(Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba)$

2) $\neg MačkaLjubimac \equiv \neg Životinja \sqcup \neg(\neg Životinja \sqcup \neg Laje) \sqcup \forall imaVlasnika.\neg Osoba$

3) $\neg MačkaLjubimac \equiv \neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba$

$nnf() = ((Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba)) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)$

$= \{ (((Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba)) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba))(x)\}$

$= \cup \{((Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba) \sqcup (Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba))(x),\ (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)(x)\}$

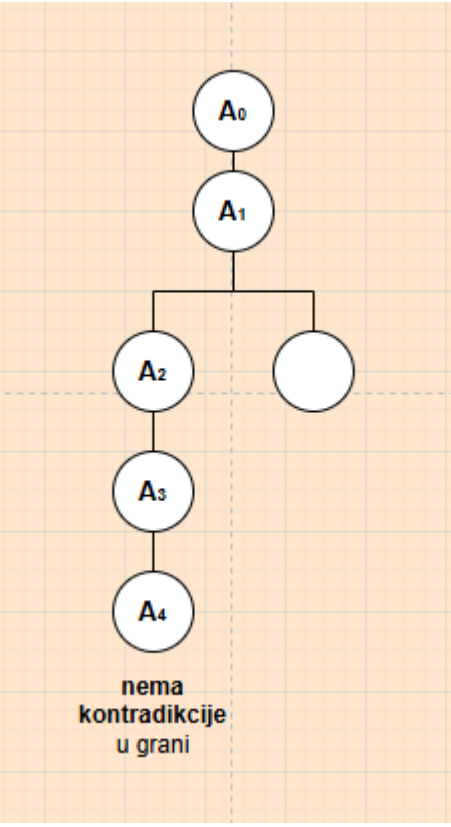
$= \cup \{(Životinja \sqcap Laje \sqcap \exists imaVlasnika.Osoba)(x)\}$

$= \cup \{(Životinja)(x),\ (Laje)(x),\ (\exists imaVlasnika.Osoba)(x)\}$

$= \cup \{imaVlasnika(x,y),\ (Osoba)(y)\}$ **nema kontradikcije** u grani

U nema kontradikcije u grani što znači da je identificirana moguća interpretacija koja zadovoljava sve uvjete unutar te grane. Dakle, postoji interpretacija koja omogućuje da su svi koncepti i relacije unutar konzistentni i mogući.

Algoritam ovdje završava, čim postoji grana u kojoj nema kontradikcije, zaključuje se da postoji barem jedna interpretacija za slučaj koji se provjerava, tako se može zaključiti da je izraz $KućniLjubimac \sqcap \neg MačkaLjubimac$ zadovoljiv, što ujedno znači da početna tvrda $KućniLjubimac \sqsubseteq MačkaLjubimac$ nije zadovoljiva. Primjer grananja je vidljiv na Slici 3.



Slika 3: Grananje za drugi primjer (vlastita izrada, draw.io)

Kako je dokazano da je tvrdnja za prethodni primjer nezadovoljiva, bilo bi zanimljivo provjeriti suprotnu tvrdnju, odnosno $MačkaLjubimac \sqsubseteq KućniLjubimac$? Želimo vidjeti je li svaka mačka ljubimac ujedno i kućni ljubimac. U kontekstu ovog sustava ($TBox$-a) i prethodnog primjera, ova tvrdnja bi trebala vrijediti. Početni korak je isti, potrebno je vidjeti vrijedi li negirana tvrdnja, odnosno postoji li mačka ljubimac koja ujedno i nije kućni ljubimac, $MačkaLjubimac \sqsubseteq KućniLjubimac \rightarrow MačkaLjubimac \sqcap \neg KućniLjubimac$? Izrazi $MačkaLjubimac$ i $\neg KućniLjubimac$ su već definirani u prethodnim primjerima.

1) $MačkaLjubimac \equiv Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba$

1) $\neg KućniLjubimac \equiv (\neg Životinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)$

$nnf(C_0) = Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba \sqcap (\neg Životinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)$

$= \{(Životinja \sqcap (\neg Životinja \sqcup \neg Laje) \sqcap \exists imaVlasnika.Osoba \sqcap (\neg Životinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba) \sqcap (\neg Životinja \sqcup (Životinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba))(x)\}$

$= \cup \{(\check{Z}ivotinja)(x),\ (\neg\check{Z}ivotinja \sqcup \neg Laje)(x),\ (\exists imaVlasnika.Osoba)(x),\ (\neg\check{Z}ivotinja \sqcup \neg Laje \sqcup \forall imaVlasnika.\neg Osoba)(x),\ (\neg\check{Z}ivotinja \sqcup (\check{Z}ivotinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)(x)\}$

$= \cup \{(\neg\check{Z}ivotinja)(x)\}$ **kontradikcija** u grani

$= \cup \{(\neg Laje)(x)\}$

U nema kontradikcije, tako da se algoritam nastavlja u ovoj grani.

$= \cup \{imaVlasnika(x,y),\ (Osoba)(y)\}$

$= \cup \{(\neg\check{Z}ivotinja)(x)\}$ **kontradikcija** u grani

U postoji kontradikcija koja se već ponovila, moguće je da se neka kontradikcija pojavi više puta, pogotovo kada je inicijalna tvrdnja definirana u korijenu grafa, u .

$= \cup \{(\neg Laje \sqcup \forall imaVlasnika.\neg Osoba)(x)\}$

$= \cup \{(\neg Laje)(x)\}$

$A_8 = A_7 \cup \{(\neg\check{Z}ivotinja)(x)\}$ **kontradikcija** u grani

$A_9 = A_7 \cup \{((\check{Z}ivotinja \sqcap Laje) \sqcup \forall imaVlasnika.\neg Osoba)(x)\}$

$A_{10} = A_9 \cup \{(\check{Z}ivotinja \sqcap Laje)(x)\}$

$A_{11} = A_{10} \cup \{(\check{Z}ivotinja)(x),\ (Laje)(x)\}$ **kontradikcija** u grani

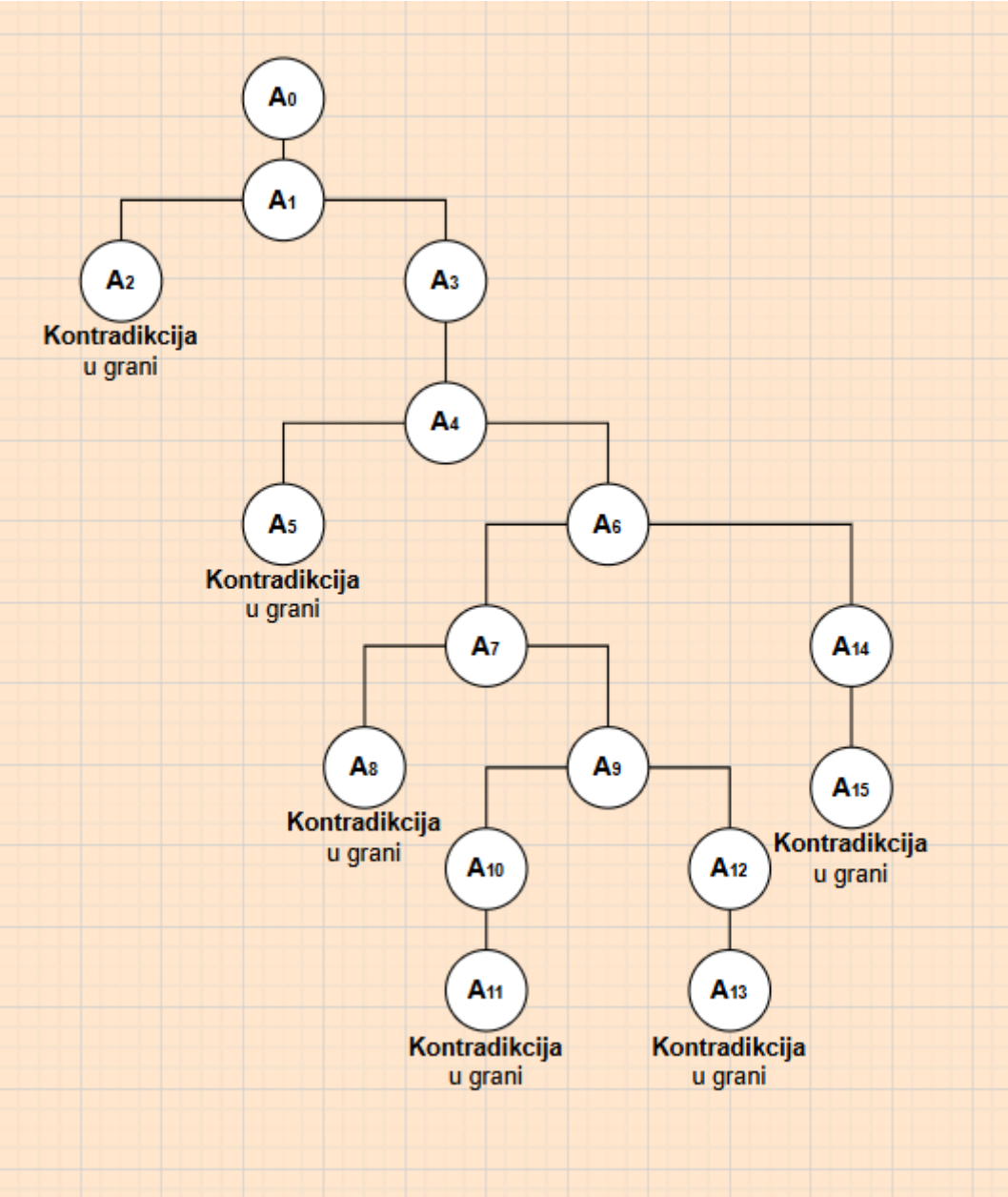
$A_{12} = A_9 \cup \{(\forall imaVlasnika.\neg Osoba)(x)\}$

$A_{13} = A_{12} \cup \{(\neg Osoba)(y)\}$ **kontradikcija** u grani

$A_{14} = A_6 \cup \{(\forall imaVlasnika.\neg Osoba)(x)\}$

$A_{15} = A_{14} \cup \{(\neg Osoba)(y)\}$ **kontradikcija** u grani

Nema slobodnih grana, u svakoj se nalazi kontradikcija, tako da se zaključuje kako je tvrdnja koja se provjeravala $MačkaLjubimac \sqcap \neg KućniLjubimac$ nezadovoljiva, odnosno zaključak je da $MačkaLjubimac \sqsubseteq KućniLjubimac$ vrijedi. Svaka mačka kućni ljubimac je ujedno i kućni ljubimac. Ovaj zaključak se slaže sa zaključkom koji je dobiven u drugom primjeru. Vizualni prikaz se može vidjeti na slici grananja za ovaj primjer, Slika 4.



Slika 4: Grananje za treći primjer (vlastita izrada, draw.io)

Kroz ova 3 primjera pokazano je kako funkcionira Tableau algoritam za $\mathcal{ALC}$, koristeći konjunkciju, disjunkciju, egzistencijalno i univerzalno ograničenje. Međutim, u deskriptivnim logikama moguće se susresti i s proširenjima koja omogućuju rad s numeričkim ograničenjima, $\mathcal{ALCN}$ i $\mathcal{ALCQ}$. U nastavku će biti objašnjeno kako Tableau algoritam radi s navedenim proširenjima.

$\mathcal{ALCN}$ proširenje omogućuje korištenje numeričkih ograničenja u konceptima. Primjer bi bio da se uvede numeričko pravilo kako svaki pas mora imati dva mladunca koji pripadaju konceptu štene, odnosno $(\geq 2 ima Mladunče. Štene)(Pas)$. Za ovaj primjer inicijalni $ABox$ je:
$= \{Pas(x), (\geq 2 ima Mladunče. (Životinja \sqcap Laje \sqcap Mlad))(x)\}$

Tableau će provjeriti ovu tvrdnju, a kako tvrdnja kaže da pas mora imati najmanje 2 mladunca, kreirat će dvije nove individue i y_2, te ih povezati s relacijom $imaMladunče$. Prošireni $ABox$ će onda izgledati :

$=\cup \{imaMladunče(x, y_1), imaMladunče(x, y_2), (\check{Z}ivotinja \sqcap Laje \sqcap Mlad)(y_1), (\check{Z}ivotinja \sqcap Laje \sqcap Mlad)()\}$

$=\cup \{(\check{Z}ivotinja)(y_1), (Laje)(y_1), (Mlad)(y_1), (\check{Z}ivotinja)(y_2), (Laje)(y_2), (Mlad)()\}$

U ovom primjeru za $\mathcal{ALCN}$ dodano je minimalno ograničenje, međutim moguće je raditi ili s maksimalnim ograničenjem, ali ne s oba odjednom. Za rad s oba ograničenja koristi se $\mathcal{ALCQ}$. Sada će se za $\mathcal{ALCQ}$ pokazati kako bi izgledala kombinacija minimalnim i maksimalnih uvjeta, odnosno primjer koji će biti prikazan je $(\geq 1 imaMladunče.\check{S}tene)(Pas) \wedge (\leq 3 imaMladunče.\check{S}tene)(Pas)$. Ovim primjerom ograničen je broj mladunaca psa na najmanje 1 i najviše 3. Inicijalni $ABox$ je:

$= \{Pas(x), (\geq 1 imaMladunče.(\check{Z}ivotinja \sqcap Laje \sqcap Mlad))(x), (\leq 3 imaMladunče.(\check{Z}ivotinja \sqcap Laje \sqcap Mlad))(x)\}$

Tableau algoritam u prvom koraku će kreirati barem jednog mladunca, što je propisano minimalnim ograničenjem:

$=\cup \{imaMladunče(x, y_1), (\check{Z}ivotinja \sqcap Laje \sqcap Mlad)(y_1)\}$

$=\cup \{(\check{Z}ivotinja)(y_1), (Laje)(y_1), (Mlad)(y_1)\}$

U idućem koraku Tableau provjerava može li dodati još individua, što je propisano maksimalnim ograničenjem:

$=\cup \{imaMladunče(x, y_2), (\check{Z}ivotinja \sqcap Laje \sqcap Mlad)(y_2), imaMladunče(x, y_3), (\check{Z}ivotinja \sqcap Laje \sqcap Mlad)(y_3)\}$

$=\cup \{(\check{Z}ivotinja)(y_2), (Laje)(y_2), (Mlad)(y_2), (\check{Z}ivotinja)(y_3), (Laje)(y_3), (Mlad)(y_3)\}$

Kada bi se sada proveo idući korak gdje se pokuša dodati još individua, došlo bi do kontradikcije, jer je dodano između 1 i 3 mladunčadi.

$=\cup \{imaMladunče(x, y_4), (\check{Z}ivotinja \sqcap Laje \sqcap Mlad)(y_4)\}$

$=\cup \{(\check{Z}ivotinja)(y_4), (Laje)(y_4), (Mlad)(y_4)\}$ **kontradikcija** u grani [24]

Još neka proširenja za koja će biti pokazani primjeri kako Tableau algoritam radi su $\mathcal{ALCR}+, \mathcal{ALCF}, \mathcal{ALCI}$.

$\mathcal{ALCR}+$ je proširenje koje omogućuje tranzitivne uloge. Ako je neka relacija tranzitivna, a postoji relacija između individua x i y, te između y i z, onda mora postojati i relacija između x i z. Inicijalni $ABox$ je:

$= \{Roditelj(x, y), Roditelj(y, z)\}$

Ako je $Roditelj(x, y)$ tranzitivna uloga, primijenilo bi se tranzitivno pravilo:

$=\cup \{Roditelj(x, z)\}$ [25]

$\mathcal{ALCF}$ je proširenje koje omogućuje definiranje funkcionalnih uloga, što znači da individua može biti povezana samo s jednom individuom preko te uloge. Primjer $ImaSupružnika(x, y)$ je funkcionalna uloga, što znači da individua x može imati najviše jednog supružnika. Ako je inicijalni $ABox$:

$= \{Osoba(x), Osoba(y), Osoba(z), ImaSupružnika(x, y)\}$

Ako se pokuša dodati druga relacija:

$=\cup \{ImaSupružnika(x, z)\}$

Nastat će kontradikcija, jer $ImaSupružnika$ je funkcionalna uloga pa se individua x ne može povezati s drugom individuom preko te uloge. [26]

$\mathcal{ALCI}$ je proširenje koje omogućuje inverzne uloge. Ako postoji relacija između x i y, onda je moguće definirati i relaciju u suprotnom smjeru y i x. Ako je inicijalni $ABox$:

$= \{ImaRoditelja(x, y)\}$

Dodavanje inverzne uloge bi bilo :

$=\cup \{ImaDijete(y, x)\}$ [27]

6. Izrada ontologije

U prvom poglavlju ontologije su definirane kao strukture koje omogućuju formalno i organizirano predstavljanje znanja u različitim područjima, s pomoću klasa i relacija između njih. Ontologije se najčešće implementiraju s pomoću jezika kao što su OWL (Web Ontology Language) i RDF (Resource Description Framework). U nastavku je opisana izrada ontologije koja je razvijena za potrebe usporedbe alata za zaključivanje,

6.1. OWL 2

OWL 2 je najnovija verzija jezika za izradu ontologija. Nudi niz proširenja u odnosu na ranije verzije OWL-a, kao što su ograničenja na svojstvima i definiranje složenijih relacija. Koristi deskriptivnu logiku za definiranje entiteta i njihovih odnosa, čineći ga idealnim za modeliranje kompleksnih domena znanja. Kroz ovaj jezik moguće je precizno definirati odnose između pojmova te izvršavati složena zaključivanja.

OWL 2 ontologije mogu se prikazati kao RDF grafovi, gdje su entiteti prikazani kao čvorovi, a relacije kao lukovi između tih čvorova. Sintaksa koju moraju podržavati svi OWL 2 alati je RDF/XML, koja pruža standardizirani način za pohranu i razmjenu OWL 2 podataka između alata i aplikacija, osiguravajući interoperabilnost. Alternativne sintakse su Turtle, OWL/XML i Manchester Syntax, koja se koristi u alatu Protégé 4 (i novijim verzijama). Svaka od ovih sintaksi ima svojih prednosti, OWL/XML ontologije je lakše obraditi s pomoću XML alata, dok Manchester Syntax omogućuje lakše čitanje i pisanje DL ontologija. [28]

Standardi povezani s RDF, RDF Schema (RDFS), OWL te ostalim tehnologijama definirani su od strane World Wide Web Consortiuma (W3C). Svi ovi standardi, uključujući detaljne specifikacije i opise, dostupni su na službenim stranicama W3C-a. [40]

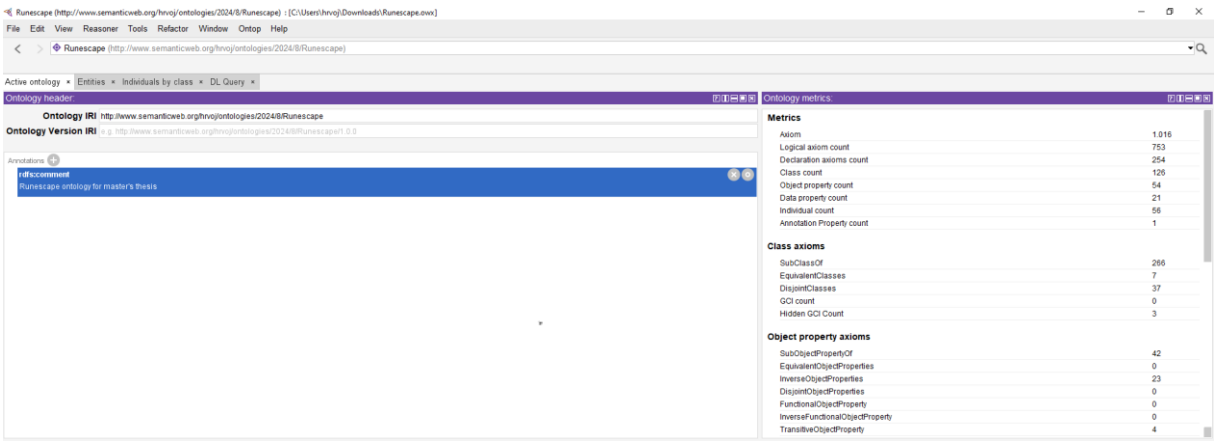
6.2. Protégé

Za izradu ontologije u ovom radu koristit će se alat Protégé. Protégé je besplatan alat razvijen na Stanfordu koji omogućuje izradu i uređivanje ontologija u OWL formatu. Podržava grafičku vizualizaciju ontologija i omogućuje jednostavno upravljanje klasama, svojstvima i individuama. Verzija Protégé alata korištenog za razvoj ontologije u ovom radu je 5.6.1. [29]

Tema kreiranja ontologije je vezana za MMORPG online igru, *RuneScape*, koja prati autora ovog rada otkada je prvi puta koristio internet, u Osnovnoj školi Mladost, prije 17 godina.

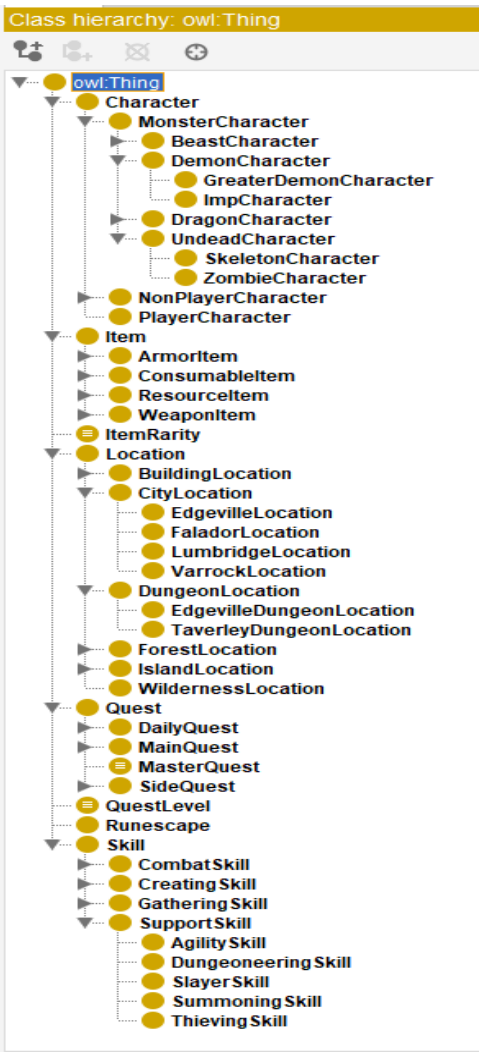
Prvi korak izrade ontologije je definiranje IRI-a, u prvoj kartici (engl. *Tab*) koja se otvori, *Active ontology*. IRI (Internationalized Resource Identifier) označava (glavni *identifier*) model

podataka koji će biti kreiran u ovoj ontologiji, odnosno služi za jednostavno identificiranje resursa na web stranici ili u ontologiji. [38] Također u prvoj kartici je moguće dodati komentar, gdje se obično definira o kakvoj ontologiji je riječ, s pomoću *rdfs:comments*. Prikaz početnog zaslona vidljiv je na Slici 5.



Slika 5: Protege početak (snimka zaslona)

Glavni elementi koji čine OWL ontologije su klase. Klase se definiraju u kartici *Entities*. Može se primijetiti da već postoji definirana jedna klasa, *owl:Thing*. To je glavna klasa, odnosno sve klase koje će biti kreirane u bilo kojoj ontologiji će biti potklase od *owl:Thing* (drugdje se može nazivati i Entity). Dakle, klase se mogu organizirati u hijerarhiju natklasa i potklasa, poznatu i kao taksonomija, koja najčešće ima oblik stabla. Višestruko nasljeđivanje se često događa između klasa, a alati za zaključivanje su ti koji omogućuju automatski izračun odnosa natklase i potklase. Prikaz taksonomije klasa koje se koriste u ovoj ontologiji može se vidjeti na Slici 6:



Slika 6: Taksonomija ontologije (snimka zaslona)

Klase su podijeljene na:

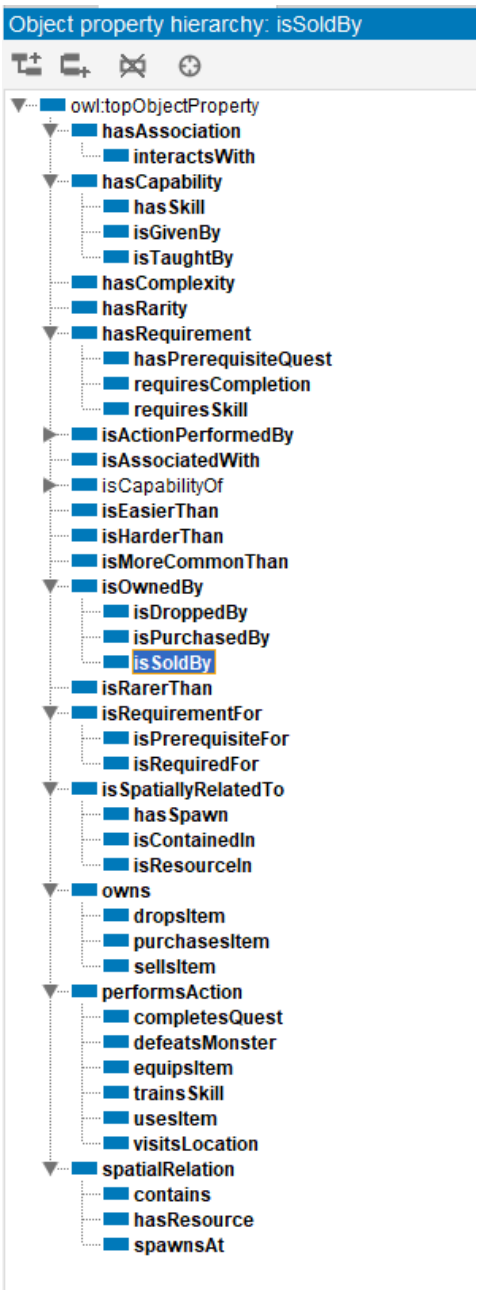
- *Character* → Likovi unutar igre su: NPC (non-player character), čudovišta ili igrač. Igračem se upravlja, u cilju razvijanja. Razvijanje se između ostalog postiže s interakcijom između ostalih likova.
- *Item* → Predmet se može nositi, može biti prodan, može se dobiti od uništavanja čudovišta, može biti kupljen...
- *Location* → Lokacije unutar igre, a mogu biti gradovi, tamnice, ali i zgrade kao banka ili dućan. Na lokacijama se nalaze razni likovi.
- *Quest* → Zadaci koji se rješavaju kako bi se moglo napredovati. Postoje određen preduvjeti prije nego je moguće riješiti pojedini zadatak. Također rješavanje jednostavnijih zadataka vodi ka rješavanju težih i složenijih zadataka.

- *Skill* → Vještine koje igrač ima, a razvijaju se učestalim ponavljanjem istih radnji, npr. ubijanjem čudovišta, ili rješavanjem zadataka. Podijeljene su u borbene i razvojne kategorije.

Bitno je spomenuti da kada se kreiraju ove klase, označeno je da su one razdvojene (engl. *disjoint)*. To znači da neka individua koja će biti kasnije kreirana, može biti član samo jedne od međusobno razdvojenih klasa. Dakle ne postoji individua koja je ujedno igrač i lokacija, to nije moguće.

Ovo je realna reprezentacija prave igre, ali nije potpuna, jer RuneScape je veoma razvijena igra, stara preko 20 godina. Cijela ontologija ove igre je ogromna, a prikazan je jedan manji uzorak kreiran za potrebe ovog diplomskog rada.

Nakon što su klase definirane, potrebno je definirati veze između njih. Unutar kartice *Entities* nalazi se kartica *Object properties*. Slično kao i kod klasa, glavno objektno svojstvo je već definirano, *owl:topObjectProperty*. Objektna svojstva su grupirana po kategorijama, a vidljiva su na Slici 7.



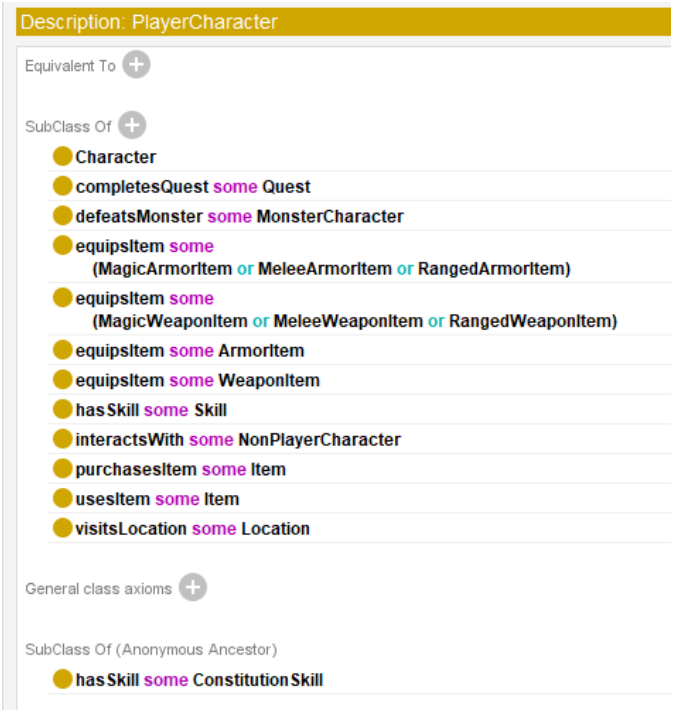
Slika 7: Objektna svojstva (snimka zaslona)

Ova svojstva definiraju veze između klasa. Primjer svojstva je *visitsLocation*, veza između igrača i lokacije. Još jedno svojstvo koje se može nadovezati na *visitsLocation*, je *defeatsMonster*. Nakon što igrač dođe na određenu lokaciju, tamo se može nalaziti čudovište. Veza između igrača i čudovišta je baš svojstvo *defeatsMonster*. Nakon što igrač porazi čudovište, ono izvodi *dropsItem*. To je veza između čudovišta i predmeta. Ovo je primjer kako igrač, došavši na lokaciju, nakon što porazi čudovište, može doći u posjed novog predmeta. Većina karakteristika koje su definirane za objekta svojstva u ovoj ontologiji su asimetričnost i irefleksivnost. Asimetričnost znači da ako je $R(x,y)$ istina, onda $R(y,x)$ nije. Irefleksivnost znači da individua ne može biti povezana sama sa sobom kroz ovo svojstvo. Također je za

većinu objektnih svojstava kreiran odgovarajući inverz, ako postoji *sellsItem*, postoji i *isSoldBy*. Ostala svojstva definiranu su u poglavlju o proširenjima deskriptivnih logika. Kod objektnih svojstava je bitno odrediti domenu i doseg svakog svojstva. Za navedene primjere domena i doseg su:

- *visitsLocation* → Domena: *Character*, Doseg: *Location*, Svojstvo: Asimetričnost i Irefleksivnost
- *defeatsMonster* → Domena: *Character*, Doseg: *MonsterCharacter*, Svojstvo: Irefleksivnost (ovdje je zanimljivo spomenuti kako asimetričnost nije definirano svojstvo, jer je moguće da i igrač i čudovište umru u istom trenu, odnosno da igrač porazi čudovište, ali isto tako da čudovište porazi igrača)
- *dropsItem* → Domena: *MonsterCharacter*, Doseg: *Item*, Svojstvo: Asimetričnost i Irefleksivnost
- *sellsItem* → Domena: *NonPlayerCharacter*, Doseg: *Item*, Svojstvo: Asimetričnost i Irefleksivnost
- *isSoldBy* → Domena: *Item*, Doseg: *NonPlayerCharacter*, Svojstvo: Asimetričnost i Irefleksivnost

Nakon što su kreirane klase i objektna svojstva, te su objektnim svojstvima definirani domena i doseg, uvode se ograničenja nad klasama na temelju definiranih objektnih svojstava. Konkretan primjer može se vidjeti na Slici 7a.



Slika 7a: Ograničenja (snimka zaslona)

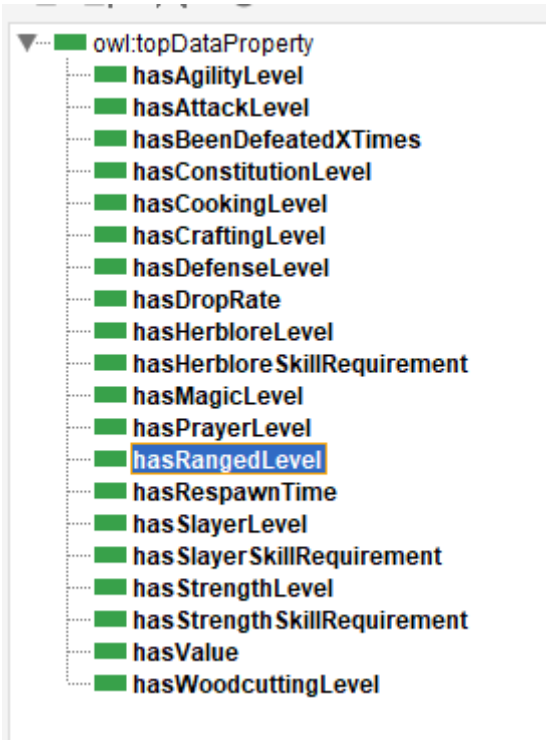
Klasi *PlayerCharacter* kreirana su egzistencijalna ograničenja, koja se označavaju s ključnom riječi *some*. Neki od primjera su:

- *completesQuest* some *Quest* → ograničenje koje definira da klasa *PlayerCharacter* može završiti neki zadatak
- *equipsItem* some (*MagicWeaponItem* or *MeleeWeaponItem* or *RangedWeaponItem*) → ograničenje koje definira da se klasa *PlayerCharacter* može opremiti s predmetom koji je oružje iz 3 različite kategorije
- *visitsLocation* some *Location* → ograničenje koje definira da klasa *PlayerCharacter* može posjetiti neku lokaciju

U ovom primjeru nema korištenja univerzalnog ograničenja, međutim primjer univerzalnog ograničenja bio bi na klasi MonsterCharacter:

- *isDefeatedBy* only *PlayerCharacter* → ograničenje koje definira da čudovište može biti poražena samo od strane igrača. Ključna riječ koja se koristi za univerzalno ograničenje je *only*.

Zatim su kreirana podatkovna svojstva u kartici Data properties (također ima glavno podatkovno svojstvo owl:topDataProperty), vidljiva na Slici 8.



Slika 8: Podatkovna svojstva (snimka zaslona)

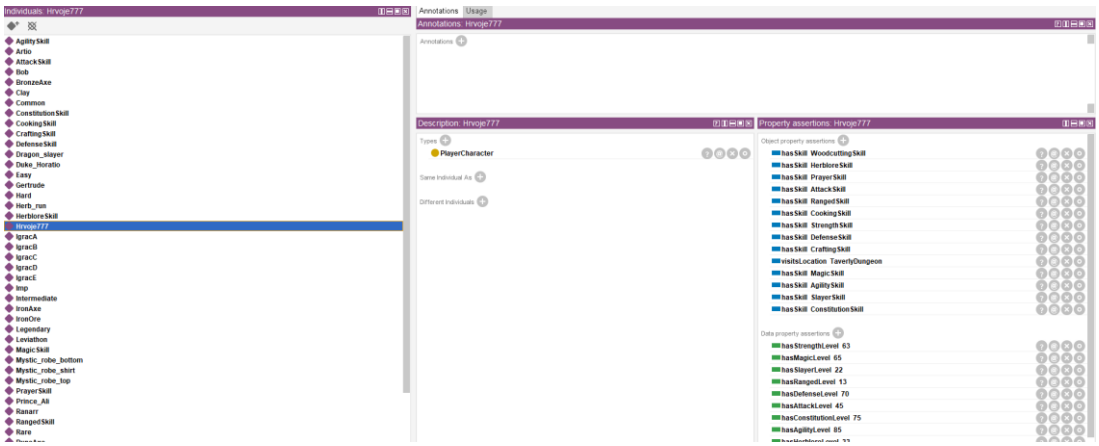
Podatkovna svojstva definiraju da neka individua, ima npr. *hasRangedLevel* 20, što znači da ima Range level 20, s time da naravno ta individua mora biti povezana s objektom *hasSkill* s klasom *RangeSkill*. Ograničenja podatkovnih svojstava su ili string ili integer u ovom slučaju, s time da je bitno naglasiti kako je moguće i definirati složena ograničenja, u slučaju da level ne može biti manji od 1 i veći od 99: $xsd{:}\,integer[>= 1, <= 99]$. Konkretni primjeri kreiranih podatkovnih svojstava su:

- *hasBeenDefeatedXTimes*, Domena: *MonsterCharacter*, Doseg: *xsd:integer* → Ovo svojstvo sugerira koliko je puta koje čudovište bilo uništeno
- hasWoodcuttingLevel, Domena: PlayerCharacter, Doseg: *xsd:integer[>=1, <=99]* → Ovo svojstvo sugerira da je igrač dosegao određenu razinu sječe drveća, od 1 do 99.
- *hasHerbloreSkillRequirement*, Domena: *HerbRunQuest*, Doseg: *xsd:integer[>=25, <=99]* → Ovo svojstvo sugerira da zadatak *HerbRunQuest* ima preduvjet kako bi moglo biti izvršeno, a to je da *HerbloreSkill* mora biti barem razina 25.

Dodavanjem individua cijela ontologija dobiva smisao, jer su tada konkretne instance klasa kreirane i dodijeljena su im objektna i podatkovna svojstva. Primjer instance je glavni igrač od autora rada, *Hrvoje777*:

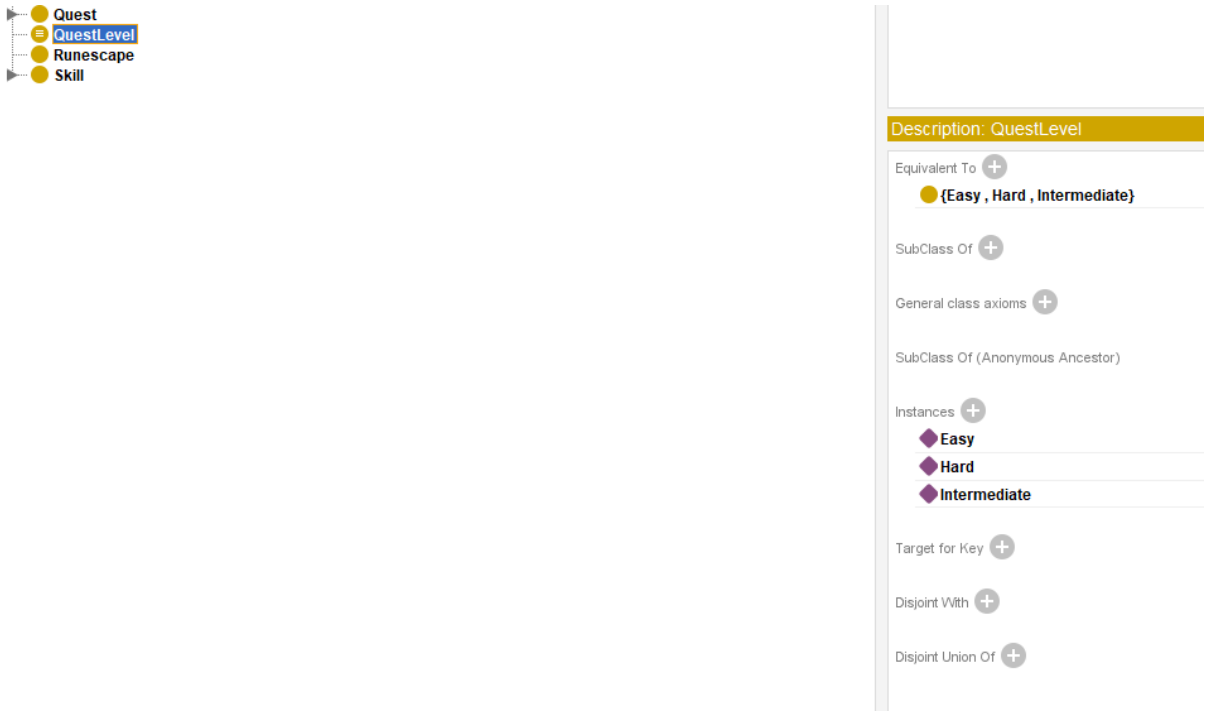
- *hasSkill WoodcuttingSkill* → objektno svojstvo
- *visitsLocation TaverlyDungeon* → objektno svojstvo
- *hasSkill SlayerSkill* → objektno svojstvo
- *hasWoodcuttingLevel 87* → podatkovno svojstvo
- *hasSlayerLevel 22* → podatkovno svojstvo

Pokretanjem alata za zaključivanje prikažu se i ostala svojstva koja su definirana na drugim individuama, a imaju veze s individuom *Hrvoje777*, prikazano na Slici 9.



Slika 9: Instance (snimka zaslona)

Također je moguće kreirati prebrojivu (enumeriranu) klasu. To je klasa koja ima definirana svojstva, odnosno kada se zna da neko svojstvo može imati samo nekoliko vrijednosti, korisno je kreirati ovakvu klasu. U primjeru na Slici 10, prikazano je kako QuestLevel može imati vrijednost samo 3 ishoda: *Easy*, *Hard* ili *Intermediate*. Također je za svaki ishod kreirana instanca tog ishoda, te objektno svojstvo *isHarderThan* koje definira koji *QuestLevel* je teži za završiti. Također, kada se definira da je *Intermediate isHarderThan Easy* i *Harder isHarderThan Intermediate*, alat za zaključivanje će pri pokretanju odmah zaključiti da je *Hard isHarderThan Easy*, iako to nije eksplicitno navedeno.



Slika 10: Enumerirana klasa (snimka zaslona)

[39]

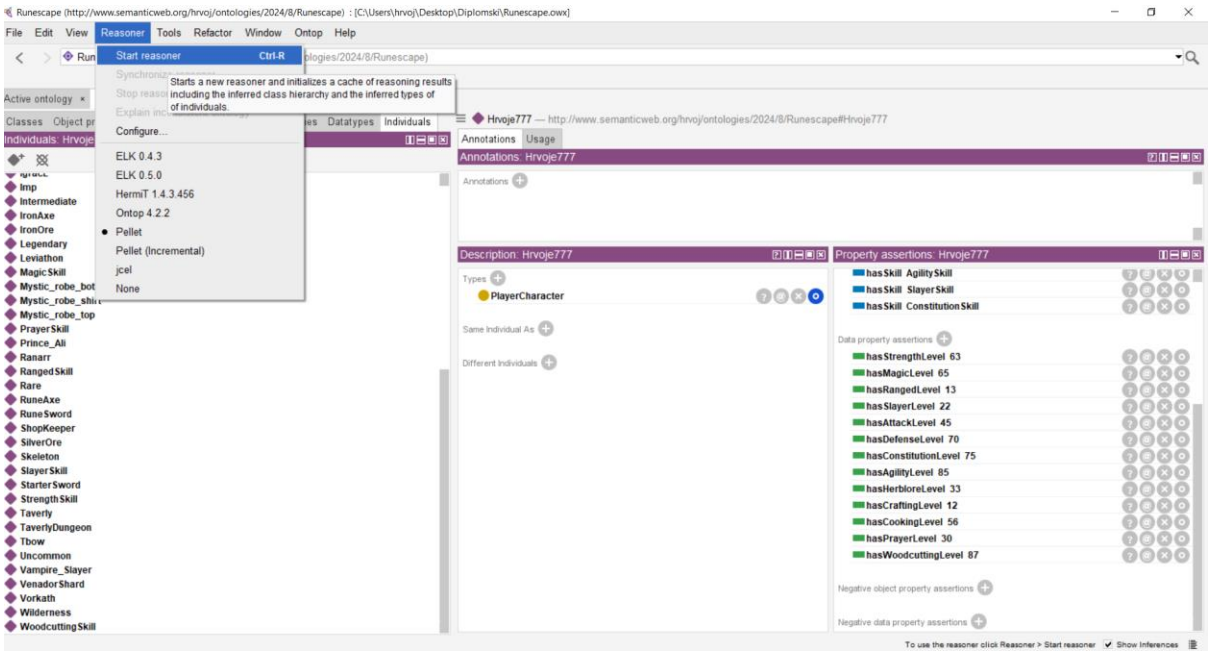
7. Alati za zaključivanje

U ovom poglavlju opisani su alati za zaključivanje te rezultati njihovog pokretanja na izrađenoj ontologiji. Alati koji su korišteni su : Pellet, HermiT, jcel i ELK. Prvo je tekstualno opisan svaki alat, a zatim je prikazano što točno alat radi na razvijenoj ontologiji. Na kraju poglavlja, tekstualno objašnjenje je prikazano tablici radi lakšeg pregleda i usporedbe. Dio kriterija za usporedbu određen je na temelju razmišljanja: vrijeme obrade alata za zaključivanje i potrošnja energije uređaja na kojem se pokreće alat. Ostali kriteriji preuzeti su iz izvora koji su također uspoređivali pojedine alate. [30] [41] Naravno, svi zaključci koji su doneseni su potkrijepljeni tekstom ili slikom provedbe zaključivanja nad razvijenom ontologijom.

7.1. Pellet

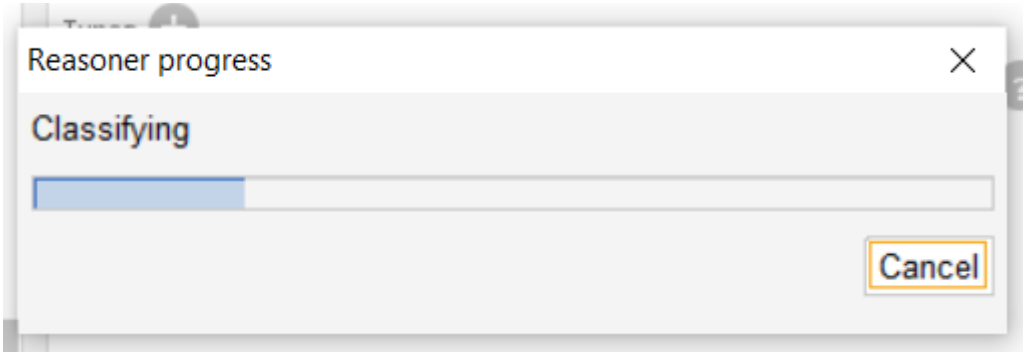
Pellet je alat otvorenog koda za zaključivanje temeljen na Javi koji koristi tableau algoritam, razvijen od grupe Mind Swap. Podržava napredne i složene varijante deskriptivnih logika, što znači da može raditi s vrlo složenim ontologijama te donositi točne zaključke u složenim domenama. [31] Također je prvi alat koji je podržavao puni OWL DL SHOIN(D) standard, odnosno set logičkih konstruktora: mogućnost da relacija bude tranzitivna (S), hijerarhiju odnosa (H), reference na određene individue (O), omogućuje inverzne odnose (I), ograničava broj članova klase (N) i podržava rad s različitim tipovima podataka (D). [14] Podržava i proširenje OWL standarda, OWL 2, čija je logička osnova SROIQ(D) gdje su tranzitivne relacije označene sa (S), mogućnost da dva odnosa budu međusobno disjunktna (R) te u odnosu na SHOIN(D) umjesto ograničenja brojeva (N), podržava kardinalnost, odnosno mogućnost postavljanja preciznih ograničenja na broj odnosa koji pojedinac može imati (Q). Zaključivanje se može odvijati kroz klasično OWL-API sučelje i Jena sučelje. [32] Jena je Java framework koji se koristi za manipulaciju RDF podacima i OWL ontologijama.

Prvi korak je pokrenuti Pellet alat unutar Protégé-a, kao na Slici 11.



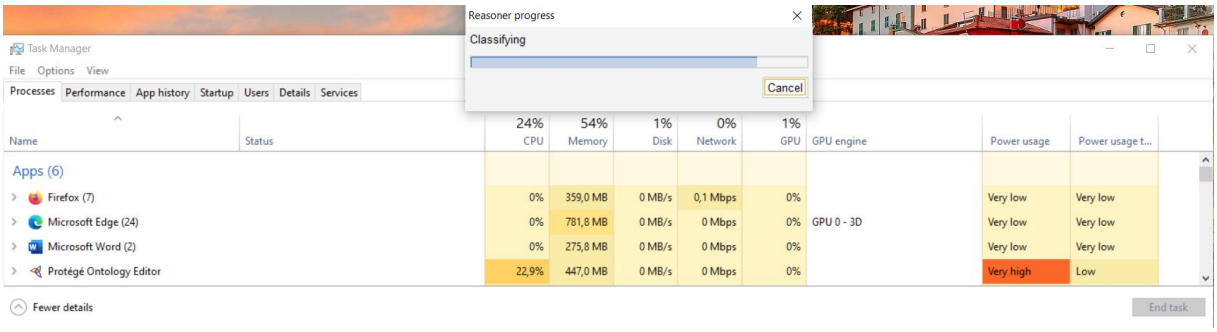
Slika 11: Pokretanje Pallet (snimka zaslona)

Zatim se pokreće alat, odnosno provodi se Tableau algoritam kako bi se odredila zadovoljivost ontologije (Slika 12):



Slika 12: Rad alata za zaključivanje (snimka zaslona)

Za vrijeme trajanja procesa zaključivanja može se primijetiti kako je potrebno izvjesno vrijeme da se ono provede (nije instantno) te je potrošnja energije na samom uređaju dosta visoka (potrebno je dosta snage da se zaključivanje izvrši), vidljivo na Slici 13.



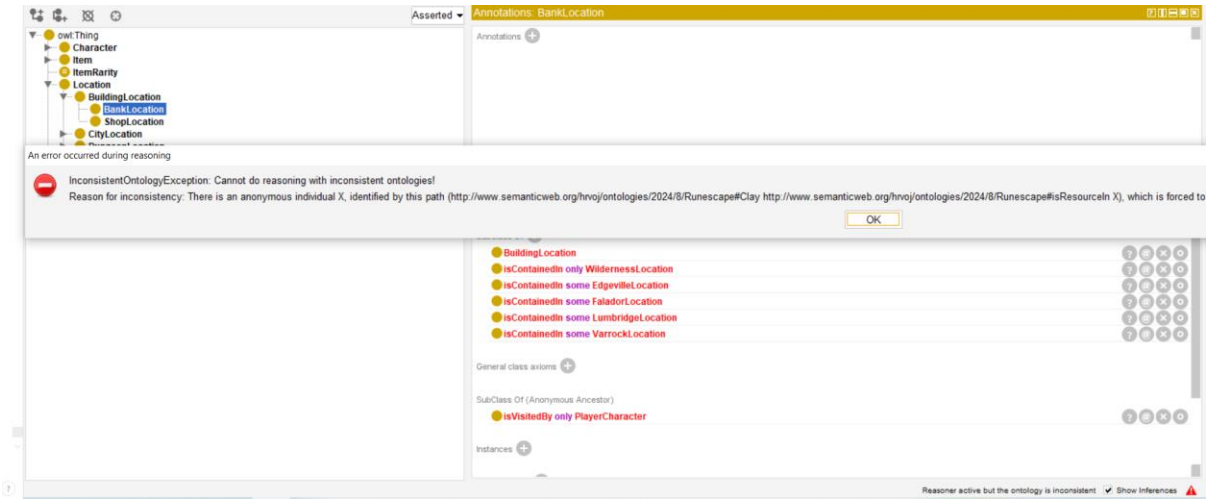
Slika 13: Potrošnja energije (snimka zaslona)

Nakon što je proces zaključivanja gotov, nije dobivena nikakva greška, a sama potvrda o završetku zaključivanja može se iščitati u bilješci (engl. *log*) od Protégéa, Slika 14.

Slika 14: Vrijeme Pellet (snimka zaslona)

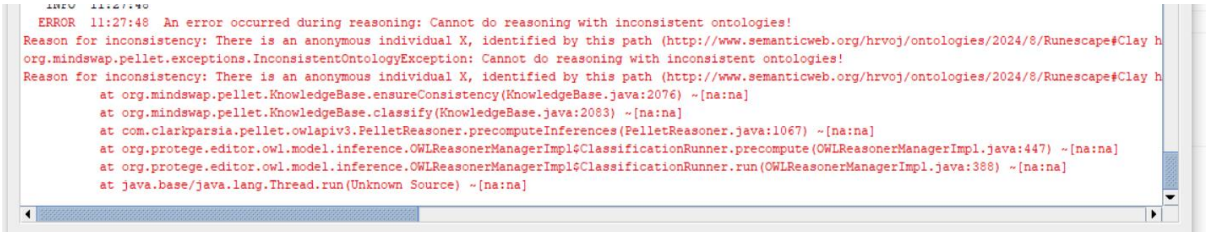
Kao što se može vidjeti, vrijeme potrebno da Pellet završi zaključivanje bilo je 77 sekundi.

Kako pri završetku nije bilo nikakvih grešaka, može se zaključiti da je ontologija konzistentna. Primjer kako bi izgledala pogreška da je ontologija nekonzistentna, vidljivo je na Slici 15.



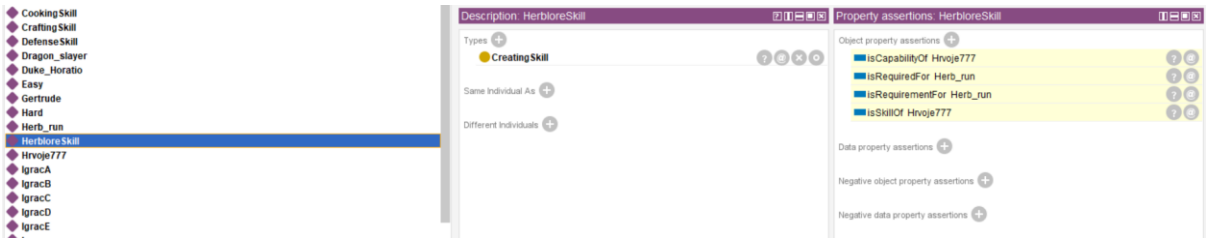
Slika 15: Inkozistentnost (snimka zaslona)

U ovom slučaju namjerno je dodano pravilo da se banka nalazi samo u *Wildernessu*, a prije toga je postojalo više pravila da banka može biti na različitim lokacijama. Naravno da banka ne može onda imati pravilo da je samo na jednoj lokaciji, a ujedno i na više lokacija. U tom slučaju Pellet javlja ovakvu grešku nekonzistentnosti. Prikaz greške prikazan je na Slici 16.:



Slika 16: Poruka pogreške (snimka zaslona)

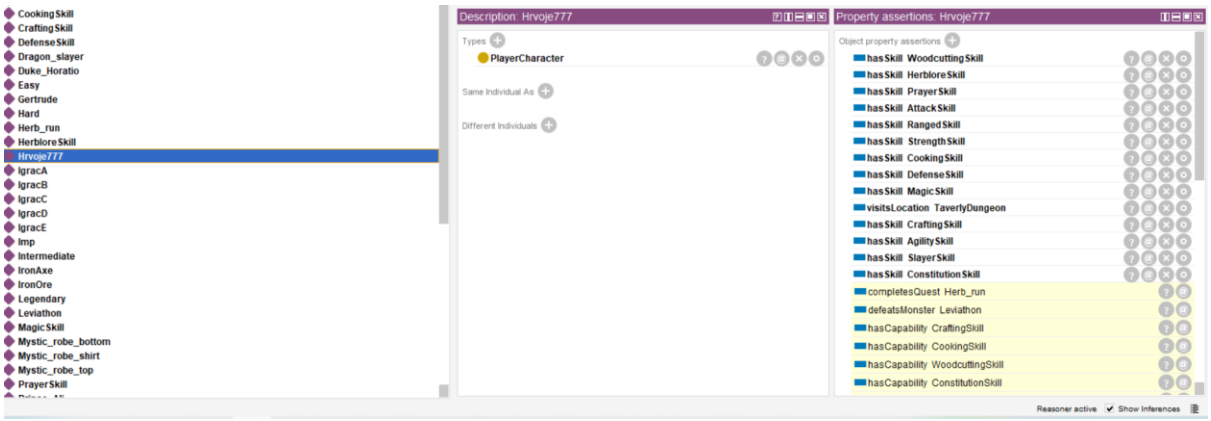
Sljedeća bitna stvar koja se može primijetiti su izvedeni zaključci (engl. *inferred*). Izvedeni zaključci su prikazani u žutoj boji, kao na Slici 17.:



Slika 17: Izvedeni zaključci (snimka zaslona)

Vidi se da *HerbloreSkill* ima samo definiran tip, ali ništa više od toga. Bez pokretanja alata za zaključivanje, sve što je u žutoj boji ne bi bilo jasno vezano uz instancu, jer nije

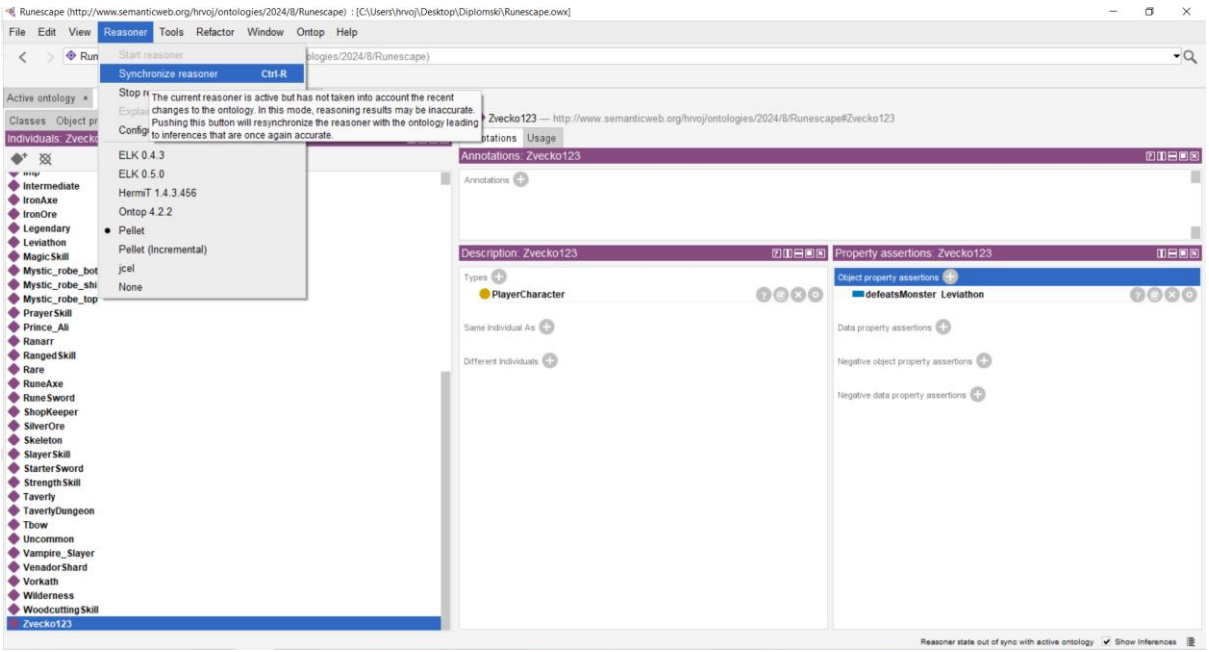
eksplicitno definirano. Međutim, kako je u instanci *Herb_run* definirano da *requiresSkill HerbloreSkill*, a svojstvo objekta *requiresSkill* ima svoj inverz *isRequiredFor*, alat dolazi do zaključka da postoji veza i iz smjera *HerbloreSkill-a* prema *Herb_run-u*, iako ona nije striktno definirana. Ista stvar je i s definiranjem *HerbloreSkill* na instanci *Hrvoje777*, vidljivo na Slici 18.:



Slika 18: Instance Hrvoje777 (snimka zaslona)

I u ostalim primjerima, može se vidjeti da je Pellet izveo sve moguće zaključke koji nisu eksplicitno definirani. Te značajke ontologije su ispravnost (engl. *Soundness*) i potpunost (engl. *Completeness*).

Za vrijeme rada alata moguće je kreirati nove individue i definirati nova pravila za njih. Nakon što su pravila kreirana, potrebno je Sinkronizirati alat, kao na Slici 19.:



Slika 19: Sinkronizacija (snimka zaslona)

Kreirana je nova individuu Zvecko123, dodijeljen joj je tip i definirano da defeatsMonster Leviathon. Nakon toga pokrenuto je sinkroniziranje alata i vrijeme da se zaključivanje završi bilo je isto kao na samom početku. Zatim je maknut novokreirani tip i objektno svojstvo te ponovno pokrenuo sinkroniziranje, vrijeme je opet bilo duže, oko 75 sekundi. Međutim, vrijeme je takvo jer je dodana veze na novo kreiranu individuu. Ali samim dodavanjem nove individue vrijeme sinkroniziranja je instantno. To je potvrđeno brisanjem individue Zvecko123, nakon što su maknute sve veze, i vrijeme sinkroniziranja je bilo instantno, a prikazano je na Slici 20.:

```
INFO  12:01:31  ------------------------------ Deleting entities ------------------------------
INFO  12:01:31  Generating changes to remove 1 entities
INFO  12:01:31  Generated 1 changes to remove 1 entities in 0 ms
INFO  12:01:31  Applied 1 changes in 6
INFO  12:01:31
INFO  12:01:32  ------------------------------ Running Reasoner ------------------------------
INFO  12:01:32  Pre-computing inferences:
INFO  12:01:32      - class hierarchy
INFO  12:01:32      - object property hierarchy
INFO  12:01:32      - data property hierarchy
INFO  12:01:32      - class assertions
INFO  12:01:32      - object property assertions
INFO  12:01:32      - data property assertions
INFO  12:01:32      - same individuals
INFO  12:01:32  Ontologies processed in 0 ms by Pellet
INFO  12:01:32
```

Slika 20: Dodavanje individue (snimka zaslona)

Stoga se može zaključiti da samo dodavanje novih individua i brisanje istih, bez kreiranja veza, ne pokreće cijeli alat ispočetka, odnosno klasifikacija ontologije je efikasno ažurirana bez potrebe za zaključivanjem cijele ontologije.

7.2. HermiT

HermiT je alat otvorenog koda za zaključivanje napisan u Java programskom jeziku, koji dijeli istu logičku osnovu SROIQ(D) kao i Pellet. Ovaj alat pruža snažnu podršku za OWL API, omogućujući jednostavnu integraciju s Protégéom i drugim OWL-baziranim alatima. Međutim, za razliku od Pelleta, HermiT ne nudi podršku za Jena framework. [33]

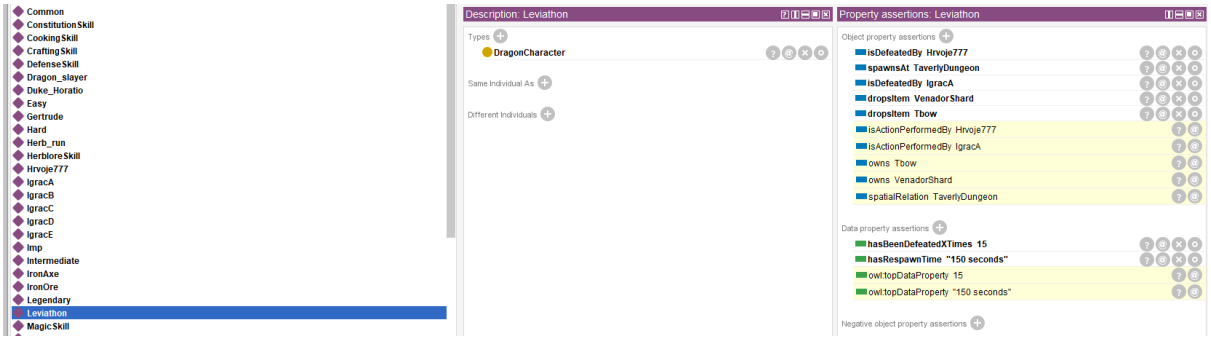
Nakon pokretanja HermiT alata u Protégé-u odmah je jasno kako je puno brži od Pelleta. HermiT je završio sa zaključivanjem nakon manje od 5 sekundi, što je vidljivo sa Slike 21.

```
INFO  15:07:04  ------------------------------ Running Reasoner ------------------------------
INFO  15:07:04  Pre-computing inferences:
INFO  15:07:04      - class hierarchy
INFO  15:07:04      - object property hierarchy
INFO  15:07:04      - data property hierarchy
INFO  15:07:04      - class assertions
INFO  15:07:04      - object property assertions
INFO  15:07:04      - data property assertions
INFO  15:07:04      - same individuals
INFO  15:07:09  Ontologies processed in 4616 ms by HermiT
INFO  15:07:09
```

Slika 21: HermiT vrijeme (snimka zaslona)

Sukladno tome, potrošnja energije za vrijeme zaključivanja je bila neprimjetna. Razlog tome je što HermIt koristi hypertableau algoritam koji je posebno optimiziran za brže zaključivanje. Smanjuje nepotrebna grananja i optimizira memorijsku upotrebu, uz pomoć predmemoriranja kako bi se ponovno koristili već izračunati zaključci.

Zaključene vrijednosti označene su za vrijeme rada alata žutom bojom, tako da vrijedi potpunost i ispravnost, vidljivo na Slici 22.



Slika 22: HermiT zaključci (snimka zaslona)

Kod dodavanja novih entiteta, cijeli algoritam se ponovno aktivira, tako da nema inkrementalnog dodavanja i brisanja. Cijeli izvještaj dodavanja novih entiteta i micanja vidljiv je na Slici 23.

Slika 23: HermiT dodavanje/brisanje individua (snimka zaslona)

7.3. Jcel

Jcel je OWL 2 EL alat za zaključivanje, koji koristi OWL API. Napisan je u Javi, a održava ga Julian Mendez. [34]

Prilikom pokretanja jcel alata na trenutnoj ontologiji u Protégé-u odmah se javlja greška, jer trenutna ontologija nije prilagođena na OWL 2 EL profil, odnosno njena je izražajnost puno veća. OWL 2 EL je jedan od profila (uz OWL 2 QL i OWL 2 RL) u okviru OWL 2 standarda za deskriptivne logike. Profil je posebno definiran za rad s vrlo velikim ontologijama koje definiraju ogroman broj klasa i svojstava.

Podržani konstrukti OWL 2 EL su: egzistencijalno ograničenje, samoograničenje, enumeracija, presjek klasa i podrška za aksiome. Međutim, postoje i ograničenja koja isključuju univerzalno ograničenje, ograničenja kardinalnosti, negaciju klasa i disjunkciju, inverzne odnose i drugo. Na temelju ovoga jasno je da ontologija u ovom radu ne može raditi s jcel alatom bez smanjenja izražajnosti. [35]

Da bi se jcel alat pokrenuo potrebno je prilagoditi određene postavke koje su definirane u trenutnoj ontologiji, odnosno potrebno je trenutno ontologiju pojednostaviti. Kako bi se pokazalo da se javljaju greške za definirane elemente ontologije, može se krenuti pojednostavljivati ontologiju.

Prva greška koja se javlja je da funkcionalno svojstvo nije podržano, prikaz je na Slici 24.:

Slika 24: jcel prva greška (snimka zaslona)

Nakon što je isključeno funkcionalno svojstvo na podatkovnom svojstvu hasCraftingLevel, alat za zaključivanje je ponovno pokrenut i dobivena je nova greška, prikazano na Slici 25.:

Slika 25: jcel druga greška (snimka zaslona)

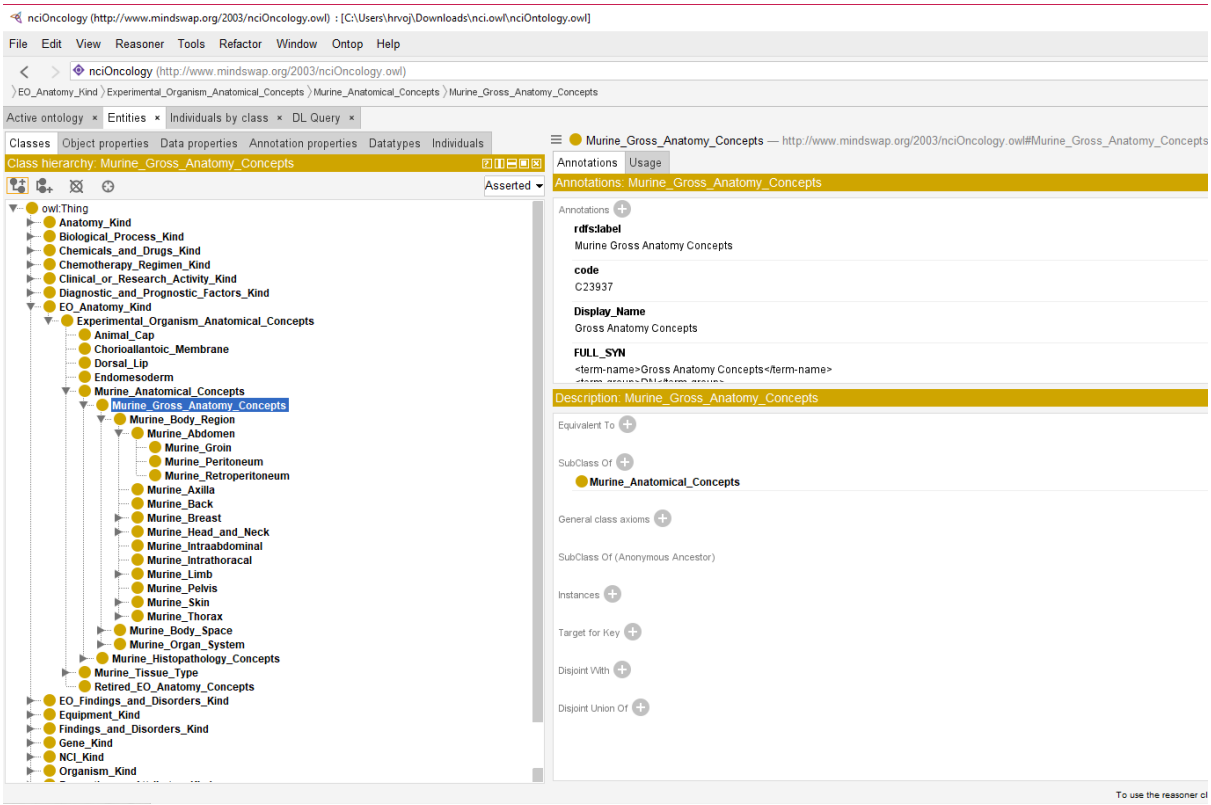
Isključena je asimetrična karakteristika iz objektnog svojstva spawnsAt, te zatim ponovno pokrenut alat i dobivena nova greška, vidljivo na Slici 26.:

Slika 26: jcel treća greška (snimka zaslona)

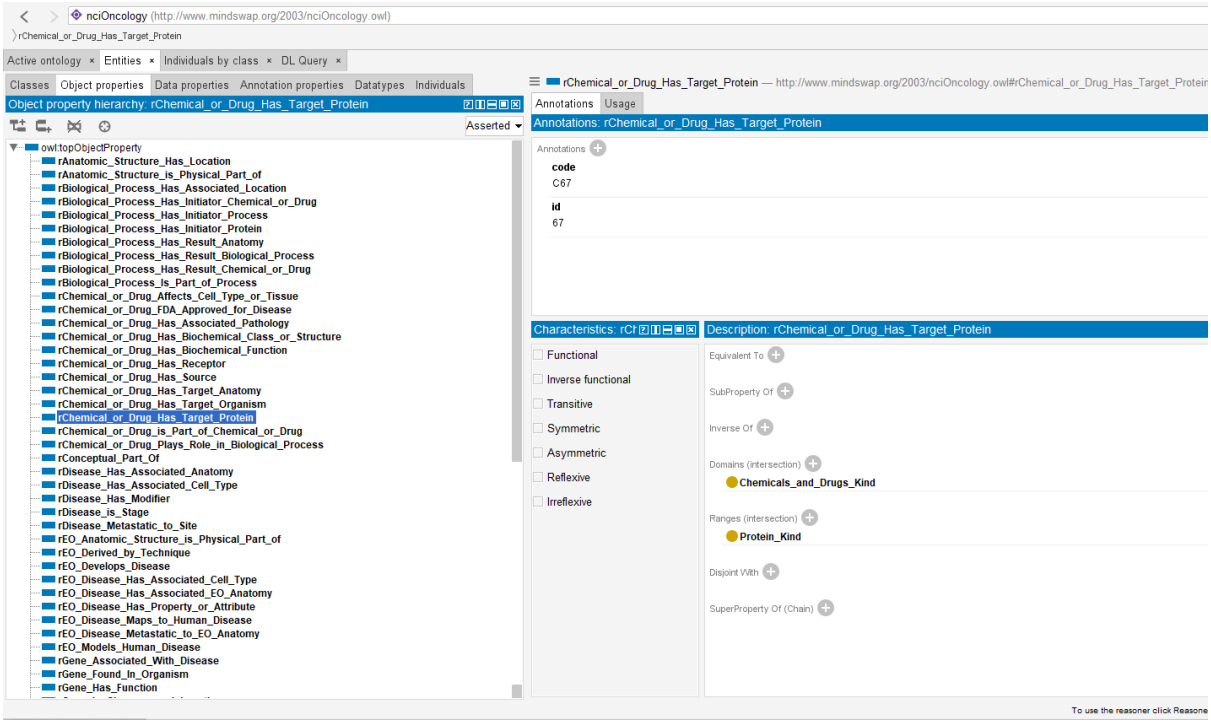
Složeni uvjet da hasWoodcuttingLevel mora biti integer između 1 i 99 nije podržan, moguće je samo definirati da treba biti integer, xsd:integer, a ne zadati od do vrijednosti koje smije poprimiti. To je potrebno promijeniti za svako podatkovno svojstvo koje ima složenu definiciju dosega. Također za svako podatkovno svojstvo je isključena funkcionalna karakteristika i javljena je greška kod objektnih svojstava da irefleksivnost nije podržana, te je i tu karakteristiku potrebno isključiti za svako objektno svojstvo, kao i asimetričnost.

Zbog svih grešaka koje se javljaju, kako bi se testirao ovaj alat, preuzet je besplatan primjer ontologije sa jcel Github stranice. Primjer koji je korišten je NCI Thesaurus ontologija[1]. Ontologija ima definirane samo klase (Slika 27.) i objektna svojstva (Slika 28.):

[1] https://sourceforge.net/projects/jcel/files/ontologies/nci.owl.zip/download



Slika 27: jcel uvozna ontologija klase (snimka zaslona)



Slika 28: jcel uvozna ontologija objektna svojstva (snimka zaslona)

Ovo je velika ontologija, s mnogo podataka, ali bez previše veza. Bilo je potrebno manje od 3 sekunde da alat odredi da je ontologija konzistentna. Vrijeme trajanja je prikazano na Slici 29.

```
INFO  21:54:56
INFO  21:55:00  ------------------------------ Running Reasoner ------------------------------
INFO  21:55:01  Pre-computing inferences:
INFO  21:55:01      - class hierarchy
INFO  21:55:01      - object property hierarchy
INFO  21:55:01      - data property hierarchy
INFO  21:55:01      - class assertions
INFO  21:55:01      - object property assertions
INFO  21:55:01      - data property assertions
INFO  21:55:01      - same individuals
INFO  21:55:03  Ontologies processed in 2511 ms by jcel
```

Slika 29: jcel vrijeme (snimka zaslona)

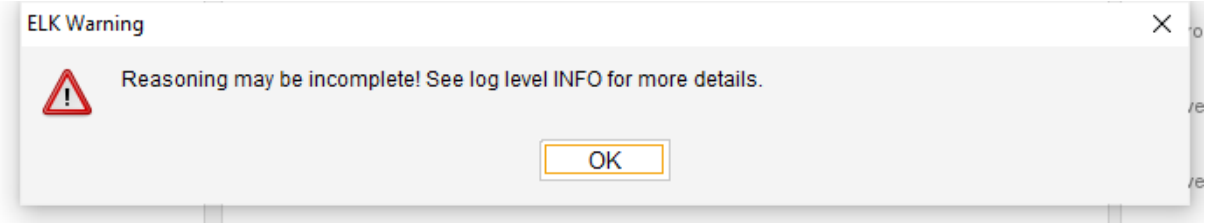
Kako je konzistentnost ontologije brzo završena, potrošnja energije nije bila visoka, međutim trenutni rad s ovakvom ontologijom troši dosta memorije, jer je riječ o stvarno velikoj ontologiji, tako da će se za potrošnju pretpostaviti da je srednje jačine.

Jcel koristi algoritam završetka temeljen na pravilima (engl. *rule-based completion algorithm*). Dok alat radi nema nikakvih zaključenih vrijednosti, jer se pojavi bug kada se prijeđe u prikaz dobiven zaključivanjem, tako da se nije mogla provjeriti ispravnost i potpunost. Također, za vrijeme rada alata bilo je moguće definirati nove klase ili individue bez da je nastala ikakva greška, a i za to je trebala sekunda manje, tako da je moguće inkrementalno dodavanje, međutim kada su se brisale pojavila su se upozorenja, no klasa ili individua je normalno obrisana. [36]

7.4. ELK

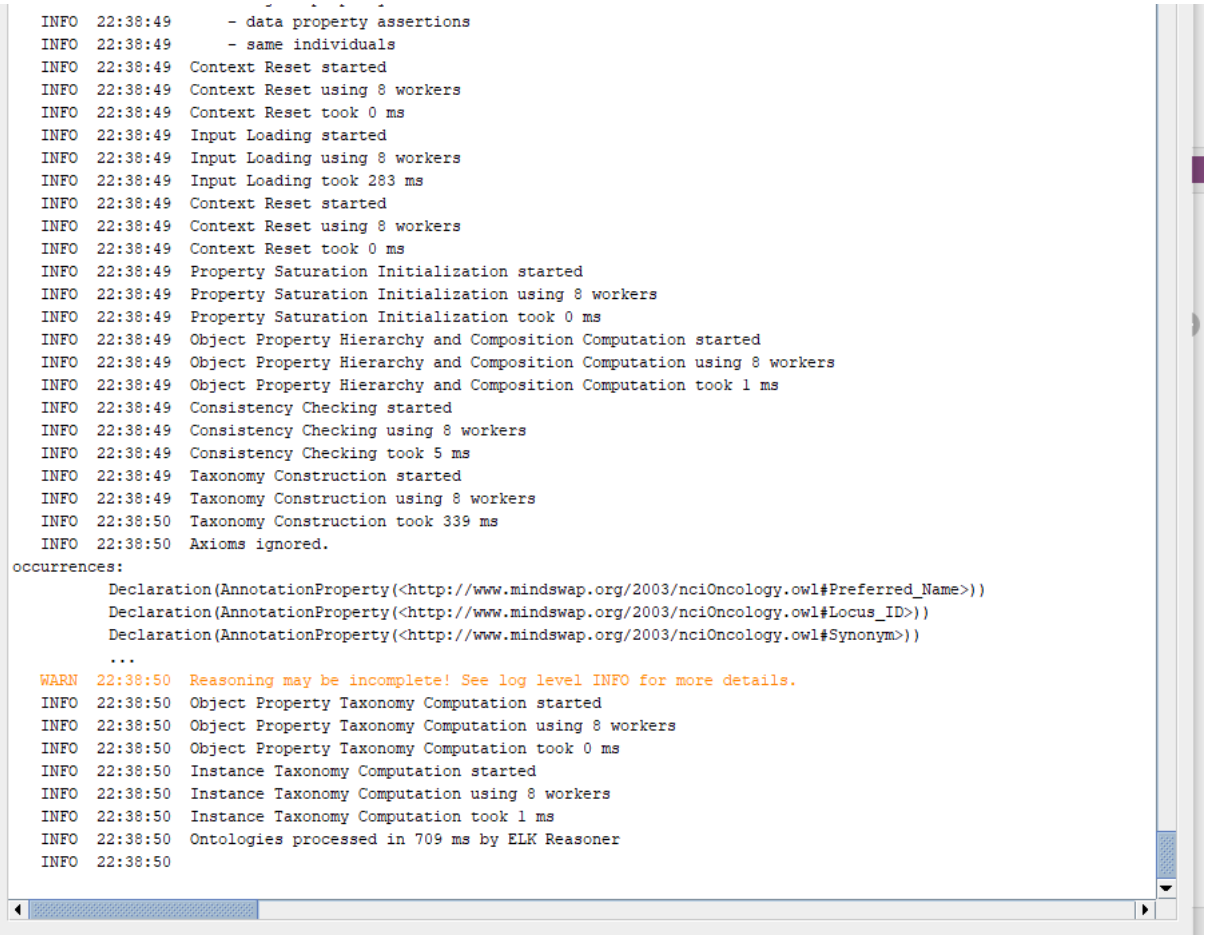
ELK je alat za zaključivanje napisan u Javi, posebno dizajniran za OWL 2 EL profil. Implementira optimizirani, polinomijalni, ciljano usmjereni postupak zaključivanja, koji je idealan za velike ontologije s mnogo klasa i svojstava. Koristi višestruke jezgre procesora za ubrzano zaključivanje. Također generira objašnjenja za logičke zaključke, kao što će biti vidljivo u dokumentu bilješke. [37]

Pokretanjem ELK alata u Protégéu za ontologiju koja se koristi i za jcel, dobivamo ELK upozorenje koje je prikazano na Slici 30.:



Slika 30: ELK upozorenje (snimka zaslona)

Međutim, ako se pogleda bilješka (Slika 31.), može se primijetiti da nema nikakvih grešaka i da je ELK obradio zaključivanje ontologije u manje od sekunde:



Slika 31: ELK log (snimka zaslona)

Također, može se vidjeti mnogo više detalja nego kod prethodnih ontologija, koliko je trebalo za input loading, ili provjeru konzistentnosti. Također se vidi da su aksiomi ignorirani, što je i očekivano od OWL 2 EL.

Ako se odabere opcija da se vide izvedeni zaključci, nema nikakvih pogrešaka kao kod jcel alata. Tako da se može zaključiti da je ontologija ispravna i potpuna. Također, inkrementalno dodavanje i brisanje je moguće.

Kao i kod jcel, potrošnja energija je niska, jer je provjera alata završila vrlo brzo, ali kako je velika ontologija aktivna onda se troši dosta memorije.

7.5. Usporedba

Za usporedbu alata za zaključivanje, razmotreni su kriteriji koji utječu na njihovu učinkovitost, pouzdanost i primjenjivost u različitim scenarijima:

- Metodologija predstavlja osnovni pristup koji alat koristi za izvođenje zaključivanja
- Logička osnova određuje vrstu logičkog sustava na kojima alat operira
- Vrijeme obrade mjeru brzinu kojom alat izvršava zaključivanje
- Potrošnja energije je danas relevantan pojam, u kontekstu održivosti računalnih sustava
- Ispravnost osigurava da zaključci dobiveni od strane alata budu točni
- Potpunost garantira da alat može prikazati sve zaključke iz eksplicitno definiranih podataka
- Inkrementalno dodavanje i brisanje omogućuju fleksibilnost u radnu s dinamičkim ontologijama, gdje se često dodaju ili uklanjaju novi pojmovi i relacije
- Programski jezik utječe na integraciju alata s postojećim sustavima
- OWL API i Jena određuju interoperabilnost alata za zaključivanje s alatima za upravljanje ontologijama
- Dostupnost omogućava korisnicima da slobodno pristupaju i doprinose razvoju alata

Tablica 7: Usporedba alata za zaključivanje

Kriterij	*Pellet*	*HermiT*	*jcel*	*ELK*
Metodologija	Tableau algoritam	Hypertableau algoritam	Rule-based completion algorithm	Consequence-based procedure
Logička osnova	SROIQ(D)	SROIQ(D)	OWL 2 EL	OWL 2 EL
Vrijeme obrade	77 sec (3.8 sec*)	4.6 sec (2.5 sec*)	2.5 sec	0.7 sec
Potrošnja energije	Jako visoka (Niska**)	Niska (Niska**)	Srednja	Srednja
Ispravnost	DA	DA	N/A	DA
Potpunost	DA	DA	N/A	DA
Inkrementalno dodavanje	DA	NE	DA	DA
Inkrementalno brisanje	DA	NE	DA	DA
Programski jezik	Java	Java	Java	Java
OWL API	DA	DA	DA	DA
Jena	DA	NE	NE	NE

Dostupnost	otvoreni kod	otvoreni kod	otvoreni kod	otvoreni kod

* Vrijeme za zaključivanje kod neizražajne ontologije koja se koristila kod alata jcel i ELK

** Potrošnja energije za zaključivanje kod neizražajne ontologije koja se koristila kod alata jcel i ELK

Na temelju dobivenih rezultata vidi se da ELK alat najbrže radi, ali naravno u području svog djelovanja. Hypertableau algoritam koji koristi HermiT djeluje brzo u području svoje logike, odnosno na brz i efikasan način dolazi do zaključka. Sve alate je moguće koristiti u Protégéu, te je zanimljivo da su svi napisani u Javi. Pellet ima najduže vrijeme zaključivanja, ali to radi na ispravan način, te omogućuje dosta detaljan uvidu u sam rezultat zaključivanja. HermiT nema mogućnost inkrementalnog dodavanja i brisanja, dok Pellet ima, no je li ta razlika bitna kada HermiT barem na ovom primjeru odradi zaključivanje 16x brže?

Kako nisu svi alati testirani na izrađenoj ontologiji, jer je prekompleksna za jcel i ELK, alati Pellet i HermiT su naknadno testirani na preuzetoj ontologiji, koja se koristila za testiranje alata jcel i ELK. Riječ je o neizražajnoj ontologiji, te su alati Pellet i HermiT završili zaključivanje za 3.8, odnosno 2.5 sekundi. Očekivano je alat HermiT bio brži, zbog korištenja Hypertableau algoritma. Međutim, vidljivo je da alat Pellet isto tako dosta brzo dolazi do zaključka, mnogo brže nego kada se definiraju svojstva kao na izrađenoj ontologiji. Potrošnja energije je bila niska u oba slučaja. Na temelju dobivenih rezultata za neizražajnu ontologiju, jasno je da ELK ima najmanje vrijeme obrade, međutim HermiT i Pellet koji su nešto sporiji ipak imaju mogućnost zaključivanja nad složenijim ontologijama.

8. Zaključak

U ovom radu razvijena je vlastita ontologija u alatu Protégé kako bi se nad njom mogla provoditi zaključivanja s različitim alatima za zaključivanje. Alati za zaključivanje koji su se koristili su Pellet, HermiT, jcel i ELK. Svi alati dolaze kao ugrađene opcije uz sam Protégé.

Razvijena ontologija imala je dosta definiranih svojstava, koje jcel i ELK ne mogu obraditi, izlaze iz njihove domene, tako da je za njihovu usporedbu korištena besplatna verzija ontologije s interneta. Međutim, razvijena ontologija je bila odličan primjer za usporedbu između alata Pellet i HermiT.

Pellet ima svoje prednosti nad HermiT-om, kao što je mogućnost dodavanja i brisanja individua u ontologiji te ponovna validacija konzistentnost, bez da se validira cijela ontologija, dok se kod HermiT-a validira uvijek cijela. Međutim Pellet je mnogo sporiji i potrošnja energije je mnogo. HermiT taj proces zaključivanja odrađuje mnogo brže, prvenstveno jer ne koristi Tableau algoritam, već Hypertablea algoritam.

ELK i jcel su svakako još brži alati za zaključivanje, pogotovo ELK koji radi u paralelnom radu, međutim oni su pogodni za velike ontologije koje nisu složene. Ontologije koje imaju mnogo klasa i objektnih svojstava su idealne da se validiraju pomoću jednog od ova 2 alata, no velika mana im je ta što ne mogu validirati složene ontologije kao Pellet i HermiT.

Alat za zaključivanje koji je odradio najbolji posao, u sklopu razvijene *RuneScape* ontologije za ovaj diplomski rad, jest HermiT.

9. Popis literature

[1] D. Baader, M. Calvanese, D. McGuinness, D. Nardi, i P. Patel-Schneider, "The Description Logic Handbook," Cambridge, UK: Cambridge Univ. Press, 2003.

[2] W. A. Woods, "What's in a Link: Foundations for Semantic Networks," Hillsdale, NJ: Erlbaum, 1975.

[3] R. J. Brachman i M. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," *Artificial Intelligence*, vol. 25, no. 1, pp. 1-65, 1985.

[4] N. K. Sondheimer, R. M. Weischedel, i R. J. Bobrow, "Semantic Interpretation Using KL-ONE," u *Proceedings of the 6th International Joint Conference on Artificial Intelligence (IJCAI)*, 1984, str. 123-128.

[5] "Description Logic Primer," *arXiv preprint arXiv:1201.4089*, 2012. [Online]. Dostupno: https://arxiv.org/pdf/1201.4089 [Pristupljeno: 10. srpnja 2024.].

[6] N. Noy i D. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Knowledge Systems Laboratory, str. 32, 2001.

[7] "List of Reasoners," [Online]. Dostupno: http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/ [Pristupljeno: 10. srpnja 2024.].

[8] J. F. Sowa, "Semantic Networks," [Online]. Dostupno: http://www.jfsowa.com/pubs/semnet.pdf [Pristupljeno: 12. kolovoza 2024.].

[9] M. Minsky, "A Framework for Representing Knowledge," [Online]. Dostupno: https://courses.media.mit.edu/2004spring/mas966/Minsky%201974%20Framework%20for%20knowledge.pdf [Pristupljeno: 10. rujna 2024.].

[10] A. Gomez-Perez, M. Fernández-López i O. Corcho, "Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web," London, UK: Springer, 2004.

[11] N. Guarino, "What is an Ontology?" [Online]. Dostupno: https://iaoa.org/isc2012/docs/Guarino2009_What_is_an_Ontology.pdf. [Pristupljeno: 5. lipnja 2024.].

[12] M. Horridge i S. Bechhofer, "The OWL API: A Java API for OWL Ontologies," 1. siječnja 2011., str. 11–21.

[13] "RDF 1.1 Concepts and Abstract Syntax," [Online]. Dostupno: https://www.w3.org/TR/rdf11-concepts/. [Pristupljeno: 5. lipnja 2024.].

[14] I. Horrocks i P. Patel-Schneider, "Reducing OWL Entailment to Description Logic Satisfiability," 2004. [Online]. Dostupno: https://ssrn.com/abstract=3199027 ili http://dx.doi.org/10.2139/ssrn.3199027. [Pristupljeno: 10. rujna 2024.].

[16] B. Glimm, I. Horrocks, B. Motik, G. Stoilos i Z. Wang, "HermiT: an OWL 2 reasoner," *Journal of Automated Reasoning*, vol. 53, br. 3, str. 245–269, 2014.

[17] "ORE 2012," [Online]. Dostupno: https://iccl.inf.tu-dresden.de/w/images/2/20/Ore2012_paper10.pdf. [Pristupljeno: 10. rujna 2024.].

[18] "Pellet: A practical OWL-DL reasoner," [Online]. Dostupno: http://www.inf.ufsc.br/~fernando.gauthier/EGC6006/material/Aula%206/Pellet%20A%20practical%20OWL-DL%20reasoner.pdf. [Pristupljeno: 12. rujna 2024.].

[19] B. Parsia, N. Matentzoglu, R. Gonçalves, B. Glimm i A. Steigmiller, "The OWL Reasoner Evaluation (ORE) 2015 Competition Report," *Journal of Automated Reasoning*, vol. 59, str. 10–100, 2017. doi: 10.1007/s10817-017-9406-8.

[20] "ALC Tableau Algorithm," [Online]. Dostupno: http://www.kr.tuwien.ac.at/education/dekl_slides/ws12/alcTableau.4nup.pdf. [Pristupljeno: 12. kolovoza 2024.].

[21] "Negation Normal Form," [Online]. Dostupno: https://www.sciencedirect.com/topics/computer-science/negation-normal-form. [Pristupljeno: 7. kolovoza 2024.].

[22] F. Baader, "Tableaux Algorithms," [Online]. Dostupno: https://lat.inf.tu-dresden.de/~baader/Talks/Tableaux2000.pdf. [Pristupljeno: 10. lipnja 2024.].

[23] "ALC Tableau Algorithm," [Online]. Dostupno: https://cw.fel.cvut.cz/old/_media/courses/a4m33rzn/alc-tableau-algorithm.pdf. [Pristupljeno: 6. kolovoza 2024.].

[24] F. Baader i U. Sattler, "Tableaux Algorithms for Description Logics," 2000. [Online]. Dostupno: https://www.tu-dresden.de. [Pristupljeno: 12. kolovoza 2024.].

[25] I. Horrocks i P. Patel-Schneider, "Optimizing Description Logic Subsumption," 1998. [Online]. Dostupno: https://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/1998/HoPa98c.pdf. [Pristupljeno: 13. kolovoza 2024.].

[26] "A Novel DL Tableau Algorithm," [Online]. Dostupno: https://arxiv.org/pdf/1105.5446. [Pristupljeno: 1. rujna 2024.].

[27] P. Patel-Schneider, "A Knowledge Representation System," *Springer*, 2012, pp. 123-134. [Online]. Dostupno: https://link.springer.com/chapter/10.1007/978-3-642-32891-6_17. [Pristupljeno: 1. rujna 2024.].

[28] "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax," [Online]. Dostupno: https://www.w3.org/TR/owl2-overview/#ref-owl-2-specification. [Pristupljeno: 4. rujna 2024.].

[29] "Protégé," [Online]. Dostupno: https://protege.stanford.edu/. [Pristupljeno: 4. rujna 2024.].

[30] S. Bechhofer, R. Volz, and P. Lord, "Pellet: A Practical OWL-DL Reasoner," [Online]. Dostupno: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9091e269a2cf7a44b46681b3de3ca489a36ad243. [Pristupljeno: 12. rujna 2024.].

[31] "Pellet GitHub Repository," [Online]. Dostupno: https://github.com/stardog-union/pellet/blob/master/README.md. [Pristupljeno: 12. rujna 2024.].

[32] "Pellet Reasoner Report," [Online]. Dostupno: http://tinman.cs.gsu.edu/~raj/8711/sp11/presentations/pelletReport.pdf. [Pristupljeno: 12. rujna 2024.].

[33] "HermiT: An OWL 2 Reasoner," [Online]. Dostupno: http://www.hermit-reasoner.com/. [Pristupljeno: 12. rujna 2024.].

[34] "Jcel: A Reasoner for Lightweight Description Logic," [Online]. Dostupno: http://julianmendez.github.io/jcel/. [Pristupljeno: 13. rujna 2024.].

[35] "OWL 2 Profiles: OWL 2 EL," [Online]. Dostupno: https://www.w3.org/TR/owl2-profiles/#OWL_2_EL. [Pristupljeno: 5. rujna 2024.].

[36] J. Mendez, "Me-ORE: Ontology Reasoning Evaluation," [Online]. Dostupno: https://iccl.inf.tu-dresden.de/w/images/d/d8/Me-ORE12.pdf. [Pristupljeno: 13. rujna 2024.].

[37] "ELK Reasoner GitHub Repository," [Online]. Dostupno: https://github.com/liveontologies/elk-reasoner/blob/main/README.md. [Pristupljeno: 14. rujna 2024.].

[38] W3C, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)," [Online]. Dostupno: https://www.w3.org/TR/owl2-syntax/. [Pristupljeno: 19. rujna 2024.].

[39] M. DeBellis, *A Practical Guide to Building OWL Ontologies: Using Protégé 5.5 and Plugins*, [Online]. Dostupno: https://www.michaeldebellis.com/post/new-protege-pizza-tutorial/. [Pristupljeno: 12. rujna 2024.].

[40] W3C, "World Wide Web Consortium," [Online]. Dostupno: https://www.w3.org/. [Pristupljeno: 19. rujna 2024.].

[41] T. Gardiner, D. Tsarkov, and I. Horrocks, "Framework for an Automated Comparison of Description Logic Reasoners," Proceedings of the International Semantic Web Conference (ISWC), Manchester, UK, 2009

10. Popis slika

11. Popis tablica

12. Prilozi

1) Runescape.owl
2) Runescape.properties