SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Goran Brajdić

DEDUKTIVNE BAZE PODATAKA

Diplomski rad

Voditelj rada:
prof. dr. sc Robert Manger

Zagreb, 2016.

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

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem svima koji su svojim savjetima, prijedlozima i podrškom pridonijeli izradi ovog diplomskog rada. Posebno se zahvaljujem svom mentoru prof.dr.sc Robertu Mangeru na pomoći, vodstvu i suradnji tijekom izrade ovog diplomskog rada.

Sadržaj

Uvod

Baze podataka postale su sastavni dio našeg svakodnevnog života. S njima se susrećemo na bankomatima, skeniranjem proizvoda na blagajnama supermarketa, traženjem telefonskih brojeva putem interneta, posuđivanjem knjiga u knjižnicama ili pak pri traženju informacija o avionskim letovima i putovanjima željeznicom. Zbog velikih količina podataka koji se moraju pohranjivati i kojima se mora moći manipulirati, većina modernih kompanija ne bi uopće mogla funkcionirati bez neke vrste sustava za upravljanje bazama podataka.

Zbog stalnog povećanja količine podataka prirodno se javlja i potreba za velikim povećanjem prostora na koji te podatke moramo pohraniti. Stoga bi bilo korisno kada bi mogli reducirati količinu prostora potrebnu za pohranu tih informacija. No kako to možemo učiniti? Jedan dobar način je da koristimo deduktivne baze podataka.

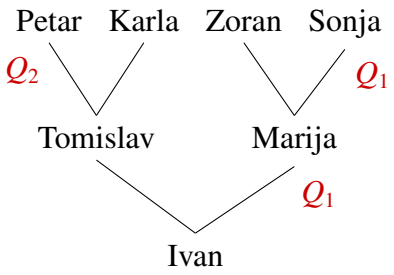
Deduktivna baza podataka je baza podataka koja može koristiti pravila kako bi izvela nove podatke koji su implicitno zastupljeni u već postojećim podacima. Direktna prednost toga je da se može potencijalno reducirati prostor potreban za skladištenje podataka. No postoje i druge možda ne tako očite prednosti:

- Korištenjem pravila možemo pohranjivati nove vrste podataka koji se ne mogu pohraniti korištenjem konvencionalnih metoda, primjerice rekurzivno definirani podaci, ili neodređeni podaci, primjerice znamo da objekt može imati jedno ili više svojstava, no nismo sigurni koje točno svojstvo ima.

- Deduktivne baze podataka bazirane su na principu matematičke logike kao područje logičkog programiranja. Obično se realiziraju u *Datalogu* ili nekom proširenju *Dataloga* (*LogiQL*, *DES*), Takvi jezici su specifično definirani za implementaciju deduktivnih baza podataka, podskup su logičkog programskog jezika *Prolog* te omogućuju postavljanje složenijih rekurzivnih upita kakvi se ne mogu izraziti u standardnom *SQL*-u.

Pretpostavimo da imamo pohranjene podatke o nekom obiteljskom stablu kao na slici 1 koje nam je važno za pronalazak genetskih predispozicija za neku bolest *X*. Postoji rizik

za dobivanje neke bolesti X ukoliko su određene varijante gena Q_1 i Q_2 naslijeđene od roditelja. Podaci koji su nam potrebni su imena djece, imena svih roditelja i poznata posjedovanja specifičnih varijanti gena (”Da” ukoliko se kod roditelja manifestira gen Q_1 ili Q_2, ”Ne” ukoliko se kod roditelja ne manifestira gen Q_1 ili Q_2). Ti podaci predstavljaju osnovne činjenice koje moraju biti pohranjene u bazi podataka (tablica 1).



Slika 1: Obiteljsko stablo

BolestX			
Ime	Roditelj	Q_1	Q_2
Ivan	Tomislav	*Ne*	*Ne*
Ivan	Marija	*Da*	*Ne*
Tomislav	Petar	*Ne*	*Da*
Tomislav	Karla	*Ne*	*Ne*
Marija	Zoran	*Ne*	*Ne*
Marija	Sonja	*Da*	*Ne*

Tablica 1: Tablica nasljedstva i posjedovanja specifičnih gena Q_1 i Q_2

Kako bismo postavili sljedeći upit da dobijemo korektan odgovor? Postoji li rizik da Ivan dobije genetsku bolest X? U SQL-u mogli bismo postaviti upit za roditelje s genetskim predispozicijama:

$$(SELECT\ Ime\ FROM\ BolestX\ WHERE\ Q_1 =' Da')\ INTERSECT$$
$$(SELECT\ Ime\ FROM\ BolestX\ WHERE\ Q_2 =' Da')$$

No, što ako Ivan može naslijediti gene od svih predaka? Primjerice može se dogoditi da se kod Tomislava ne manifestira gen Q_2, zato jer je gen Q_2 kod Tomislava možda latentan, tj. poznato je da nekada geni preskoče generaciju pa je moguće da Ivan svejedno naslijedi taj gen. Tada je očito potrebna ekstenzija modela gdje bi trebali pohranjivati

(*Ime*, *Predak*, Q_1, Q_2). No to nije samo ekstenzija modela, već se javlja i potreba za promjenom sheme baze podataka. Također, zbog takve ekstenzije potrebno je i promijeniti sam sadržaj baze podataka (jer ne znamo ništa o tome tko su predci neke osobe), za što je potrebno mnogo više prostora za pohranu.

Sada je sasvim jasno da relacijske baze podataka neće biti primarni izbor za naš skup problema. Razlog tome je što je znanje koje one reprezentiraju statičko znanje dano tablicama i izvedeno znanje dano mehanizmom pogleda koje je ponovno na neki način statičko. Jer kao što vidimo u našem primjeru ne možemo izvesti sve pretke. Da bi to mogli potrebne su nam rekurzije, odnosno sljedeća deduktivna pravila:

- Svaka osoba ima ime, roditelja i genetičke predispozicije.
- Svaki roditelj osobe je ujedno i predak te osobe.
- Svi roditelji predaka su predci.
- Za sve osobe postoji rizik da obole od bolesti X, ako neki predak ima varijantu gena Q_1 i neki predak ima varijantu gena Q_2.

Ovim pravilima sada možemo izvesti sve pretke neke osobe, te možemo dobiti odgovor na pitanje postoji li rizik da neka osoba oboli od bolesti X, što nije bilo moguće ukoliko bi koristili upite relacijske baze podataka bez rekurzija.

Ideja korištenja pravila kako bismo izveli znanje nije nova. Automatizirane dedukcije bile su veliko područje istraživanja tokom 60-tih godina prošlog stoljeća i razvoji u tom području korišteni su u logičkom programiranju, deduktivnim bazama podataka i ekspertnim sustavima. Prvi jezik za realizaciju baza podataka bio je Datalog. Datalog je razvijen 1987. godine od strane Hervé Gallairea i Jack Minkera. Koristio se prvo kao temelj za teoriju ekspertnih sustava tijekom osamdesetih godina prošlog stoljeća, a potom i kao temelj za teoriju i razvoj deduktivnih baza podataka. Ekspertni sustavi funkcioniraju na sličan način kao i deduktivne baze podataka koristeći pravila kako bi izveli novo znanje. No postoje i određene razlike, ekspertni sustavi orijentirani su više prema pravilima, dok su deduktivne baze podataka orijentirane prema podacima. Odnosno, deduktivne baze podataka orijentirane su više ka pohrani i manipulaciji velikih količina podataka, dok su ekspertni sustavi orijentirani ka pohrani i manipulaciji velikih količina pravila.

Deduktivne baze podataka sve do današnjice nisu doživjele veliki komercijalni uspjeh kao zasebni sustav, već su i dalje pretežno popularne i vladaju relacijske baze podataka. No njihovu potrebu prepoznali su mnogi sustavi za upravljanje relacijskim bazama podataka (Oracle, DB2, MSSQL, MySQL,...), pa je tako i SQL3 standardom uvedena mogućnost postavljanja upita koji koriste rekurzije.

Kako logika prvog reda predstavlja teorijsku bazu za razvoj i realizaciju deduktivnih baza podataka, njome ćemo se baviti u prvom poglavlju. Tu ćemo navesti neke važne i poznate pojmove i rezultate logike prvog reda i uvesti nekoliko novih pojmova kao što su Herbrandov svemir, Herbrandova baza te Herbrandove interpretacije koji će imati ključnu ulogu u semantici Datalog jezika. Drugo poglavlje ima za cilj opisati svojstva Datalog jezika, njegovu sintaksu i semantiku, klasificirati razne programe pisane u Datalogu, te ga povezati sa relacijskom algebrom i njenim svojstvima. Zatim raznim tehnikama optimizacije odrediti efikasne načine evaluacije programa pisanih u Datalog jeziku. Treće i posljednje poglavlje bavi se jednom od implementacija Datalog jezika i realizacijom jedne konkretne baze podataka. Tu ćemo vidjeti kako možemo u jednom konkretnom softveru realizirati deduktivnu bazu podataka i postavljati razne rekurzivne upite nad takvom bazom.

Poglavlje 1

Logika prvog reda

Centralna ideja logike prvog reda je formalno deducirati iz nekog skupa činjenica koje izjave su istinite, a koje su lažne. Naravno da bismo to mogli prvo trebamo definirati što je istinito a što je lažno. Za razliku od pojmova logike i propozicionalne logike, logika prvog reda uvodi koncept **predikata**, stoga se često još naziva i predikatna logika prvog reda. Predikate koristimo za grupiranje individualnih entiteta u tipove, tako ćemo primjerice izjavu "Sokrat je filozof" pisati kao predikat *filozof(Sokrat)*. Još jedan koncept koji uvodi logika prvog reda su univerzalni i egzistencijalni kvantifikatori.

Razlog zašto se oslanjamo na logiku prvog reda je zato što je logika prvog reda potpuno formalizirana. Da bismo mogli formalizirati logiku prvog reda prva stvar koju trebamo jest formalni jezik koji razmatra kako izgleda ispravna logička izjava, tj. određuje se što je ispravna logička izjava, a što nije ispravna logička izjava sa gledišta sintakse. Nakon što odredimo kako ispravna logička izjava izgleda moramo odrediti da li je ta izjava istinita ili je lažna, dakle moramo je nekako interpretirati sa semantičke točke gledišta.

Nakon što imamo sintaktički ispravne izjave koje možemo interpretirati, želimo ih iskoristiti kako bi generirali novo znanje. Odnosno želimo sustav koji može raditi dedukciju i generirati novo znanje. To će biti naš Datalog sustav koji će nam omogućiti realizaciju deduktivne baze podataka.

U ovom poglavlju dane su osnovne definicije i rezultati logike prvog reda koji će nam biti potrebni u ostalim poglavljima te na temelju kojih ćemo graditi naš deduktivni sustav, odnosno našu deduktivnu bazu podataka.

1.1 Sintaksa logike prvog reda

Definicija 1.1.1. *Alfabet nekog jezika logike prvog reda je unija sljedećih šest skupova:*

1. $\Gamma = \{c_k : k \in K \subseteq \mathbb{N}\}$*, neprazan skup čije elemente nazivamo* ***konstantski simboli****.*
2. $\Omega = \bigcup_{n\in\mathbb{N}} \Omega_n$ *, disjunktna unija konačnih skupova* Ω_n *n-arnih* ***funkcijskih simbola****.*
3. $\Pi = \bigcup_{n\in\mathbb{N}} \Pi_n$ *, disjunktna unija konačnih skupova* Π_n *n-arnih* ***predikatnih simbola****.*
4. $X = \{x_1, x_2, x_3, \ldots\}$*, prebrojiv skup čije elemente nazivamo* ***individualne varijable****.*
5. $\{(\ ),\}$*, skup* ***pomoćnih simbola*** *(lijeva i desna zagrada, te zarez).*
6. $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \forall, \exists\}$*, skup* ***logičkih simbola*** *koje redom nazivamo: negacija, disjunkcija, konjunkcija, kondicional, bikondicional, univerzalni i egzistencijalni kvantifikator.*

Specifični jezik logike prvog reda definiramo kao uređenu četvorku $\mathcal{L} = (\Gamma, \Omega, \Pi, X)$*.*

Napomena 1.1.2. *Prilikom pisanja raznih riječi alfabeta logike prvog reda nećemo se strogo držati samo simbola iz alfabeta. Tako ćemo obično umjesto individualnih varijabli* x_i *pisati znakove* $x, y, z, \ldots$ *Umjesto predikatnih simbola pisat ćemo znakove* $P, Q, R, \ldots$*, umjesto funkcijskih simbola pisat ćemo* $f, g, h, \ldots$*, te umjesto konstantskih simbola* c_k *pisat ćemo* $a, b, c, \ldots$

Definicija 1.1.3. ***Term*** *je riječ jezika* $\mathcal{L}$ *definirana sljedećom induktivnom definicijom:*

1) svaki konstantski simbol iz skupa Γ *je term;*

2) svaka individualna varijabla iz skupa X je term;

3) ako je $f \in \Omega_n$ *n-arni funkcijski simbol i* $t_1, \ldots, t_n$ *su termi, tada je* $f(t_1, \ldots, t_n)$ *term;*

4) riječ je term ako i smo ako je nastala primjenom konačno mnogo puta pravila 1)*,* 2) *i* 3)*.*

Term koji ne sadrži individualne varijable nazivamo ***osnovni term****. Skup svih terma jezika* $\mathcal{L}$ *označavamo sa* $T_{\mathcal{L}}$*.*

Definicija 1.1.4. *Definiramo skup svih* ***atomarnih formula*** *jezika* $\mathcal{L}$ *kao*

$$A_{\mathcal{L}} := \{p(t_1, ..., t_n) : p \in \Pi_n,\ t_1, ..., t_n \in T_{\mathcal{L}}\}.$$

Atomarna formula koja se sastoji samo od osnovnih terma naziva se ***osnovna atomarna formula****.*

Primjer 1.1.5. *Neka je zadan jezik* $\mathcal{L} = (\Gamma, \Omega, \Pi, X)$ *na sljedeći način:*

- *Konstantski simboli* $\Gamma = \{Sokrat,\ Hektor,\ a,\ b,\ zeleno,\ crveno,\ plavo\}$.
- *Predikatni simboli* $\Pi = \{Filozof(x),\ Ratnik(x),\ roditelj(x, y)\}$.
- *Funkcijski simboli* $\Omega = \{znanje(x)\}$.
- *Individualne varijable* $X = \{x,\ y,\ z\}$.

– *Tada je* $T_{\mathcal{L}} = \{Sokrat,\ Hektor,\ a,\ b,\ zeleno,\ crveno,\ plavo,$
$x,\ y,\ z,\ znanje(Sokrat),\ znanje(Hektor),\ znanje(x), \ldots\}$.

– *Skup atomarnih formula* $A_{\mathcal{L}} = \{Filozof(Sokrat),\ Filozof(Hektor), \ldots,$
$Filozof(z), Filozof(znanje(Sokrat)), \ldots, Ratnik(Sokrat), \ldots\}$.

Definicija 1.1.6. *Pojam* ***formule*** *definiran je sljedećom induktivnom definicijom:*

1) svaka atomarna formula je formula;

2) ako su W_1 *i* W_2 *formule, tada su i* $\neg W_1,\ W_1 \vee W_2,\ W_1 \wedge W_2,\ W_1 \rightarrow W_2,\ W_1 \leftrightarrow W_2$ *također formule;*

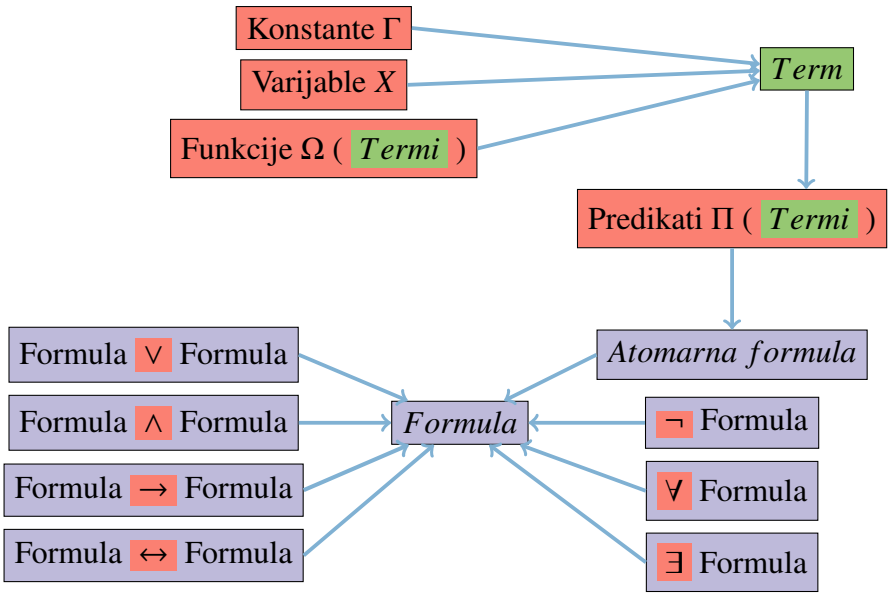
3) ako je W formula i x varijabla, tada su riječi $\forall xW$ *i* $\exists xW$ *također formule;*

4) riječ je formula ako i samo ako je nastala primjenom konačno mnogo puta pravila 1), 2) *i* 3).

Napomena 1.1.7. *Kako bi izbjegli nagomilavanje zagrada u formulama, usvajamo sljedeću hijerarhiju među logičkim simbolima:*

1. $\forall,\ \exists$
2. $\neg$
3. $\vee$
4. $\wedge$
5. $\rightarrow,\ \leftrightarrow$

Ako su F i W oznake za istu formulu tada pišemo $F \equiv W$*, i govorimo da su formule F i W jednake. Znak* $\equiv$ *nije znak alfabeta nekog jezika logike prvog reda već je pomoćni, tj. meta-simbol.*



Slika 1.1: Dijagram sintakse

Definicija 1.1.8. *Svaka varijabla koja se pojavljuje u formuli je* ***slobodna*** *ili* ***vezana****. Za bilo koju formulu W možemo definirati tri skupa:*

- ***vars(W)****, skup koji sadrži varijable formule W.*
- ***free(W)****, skup koji sadrži sve slobodne varijable formule W.*
- ***bound(W)****, skup koji sadrži sve vezane varijable formule W*

Također vrijedi $vars(W) = free(W) \cup bound(W)$.

Definicija 1.1.9. *Za bilo koju formulu W, skupove* $free(W)$ *i* $bound(W)$ *definiramo rekurzivno:*

- $free(W) := vars(W)$ *i* $bound(W) := \emptyset$*, ako je W atomarna formula.*
- $free(\neg W) := free(W)$ *i* $bound(\neg W) = bound(W)$
- $free(W_1 \wedge W_2) := free(W_1) \cup free(W_2)$ *i* $bound(W_1 \wedge W_2) := bound(W_1) \cup bound(W_2)$*, analogno vrijedi i za* $\vee$, $\rightarrow$, *i* $\leftrightarrow$.
- $free(\forall x(W)) := free(W) \setminus \{x\}$ *i* $bound(\forall x(W)) := bound(W) \cup \{x\}$*, analogno vrijedi i za* $\exists$.

Definicija 1.1.10. *Kažemo da je formula W* ***zatvorena*** *ako vrijedi* $free(W) = \emptyset$*, inače kažemo da je formula* ***otvorena****.*

Primjer 1.1.11. *Primjeri definicija 1.1.8, 1.1.9, i 1.1.10:*

- $F_1 \equiv P_1()$ *je zatvorena, jer* $free(F_1) = \emptyset$
- $F_2 \equiv P_2(x_1,\ x_2)$ *je otvorena, jer* $free(F_2) = \{x_1,\ x_2\}$
- $F_3 \equiv \forall x_1(P_2(x_1,\ x_2))$ *je otvorena, jer* $free(F_3) = \{x_2\}$
- $F_4 \equiv \exists x_2 \forall x_1(P_2(x_1,\ x_2))$ *je zatvorena, jer* $free(F_4) = \emptyset$
- $F_5 \equiv P_3(x_1) \wedge P_3(x_2)$ *je otvorena, jer* $free(F_5) = \{x_1,\ x_2\}$
- $F_6 \equiv Riba(Nemo)$ *je zatvorena, jer* $free(F_6) = \emptyset$
- $F_7 \equiv Riba(x_1)$ *je otvorena, jer* $free(F_7) = \{x_1\}$
- $F_8 \equiv \forall x,\ y\ (Riba(x) \wedge Jezero(y)\ \rightarrow\ voli(x,\ y))$ *je zatvorena, jer* $free(F_8) = \emptyset$

Napomena 1.1.12. *Vezane varijable vrijede samo u dosegu njima odgovarajućih kvantifikatora, tj. ista individualna varijabla može se pojaviti u formuli više puta neovisno o prethodnima.*

Primjer 1.1.13. *Formula* $F \equiv \forall x_1\ (P_3(x_1)) \wedge P_2(x_1)$ *je otvorena, i* $free(F) = \{x_1\}$ *i* $bound(F) = \{x_1\}$*. Prvi* x_1 *je neovisan o ostalim* x_1*. Ovakve formule su često zbunjujuće, stoga je dobra praksa preimenovati sva pojavljivanja vezane varijable izvan njenog dosega.*

Definicija 1.1.14. *Za formulu W za koju vrijedi* $free(W) \cap bound(W) = \emptyset$ *kažemo da je* ***ispravljena formula****.*

Napomena 1.1.15. *Od sada pa nadalje promatrati ćemo samo* ***ispravljene formule****.*

Definicija 1.1.16. *Za svaku formulu W za koju vrijedi* $free(W) = \{x_1,\ x_2, \ldots,\ x_n\}$ *definiramo operacije zatvorenja:*

- ***Univerzalno zatvorenje:*** $\forall x_1,\ \forall x_2, \ldots,\ \forall x_n\ (W)$ *ili kraće* $\forall x_1,\ x_2, \ldots,\ x_n\ (W)$
- ***Egzistencionalno zatvorenje:*** $\exists x_1,\ \exists x_2, \ldots,\ \exists x_n\ (W)$ *ili kraće* $\exists x_1,\ x_2, \ldots,\ x_n\ (W)$

1.2 Semantika logike prvog reda

U prethodnom odjeljku vidjeli smo kako se formiraju sintaktički ispravne formule. U ovom odjeljku vidjet ćemo kako interpretirati jezik $\mathcal{L} = (\Gamma, \Omega, \Pi, X)$. Odnosno vidjet ćemo kako interpretacijama možemo odrediti semantiku jezika $\mathcal{L}$.

Definicija 1.2.1. *Interpretacija je uređena četvorka $I = (U,\ I_C,\ I_F,\ I_P)$ gdje je:*

- *U, neprazan skup objekata, entiteta i koncepata koji se odnose na našu trenutnu primjenu. Skup U nazivamo **domena diskursa**.*
- *$I_C : \Gamma \to U$, preslikavanje svih konstantskih simbola u elemente domene diskursa.*
- *I_F, preslikava svaki $f \in \Omega_n$ u n-arnu funkciju, tj.*
  $I_F(f) : \underbrace{U \times \ \ldots \ \times U}_{n-arna\ funkcija} \ \to \ U$.
- *I_P, preslikava svaki $p \in \Pi_n$ u n-arni predikat, tj. $I_P(p) \subseteq U \times \ldots \times U$.*

*Osim interpretacije jezika $\mathcal{L}$ definiramo i **zamjenu varijable** $\rho : X \ \to \ U$. Sada svaki term $t \in T_{\mathcal{L}}$ može biti interpretiran u odnosu na interpretaciju $I_{\mathcal{L}}$ i zamjenu ρ. Da bi to bilo moguće koristimo **evaluaciju terma** $I^*_\rho(t) =: t_I \in U$, gdje je t_I rezultat interpretacije od t.*

Definicija 1.2.2. ***Evaluacija terma** $I^*_\rho(t)$ definirana je sljedećim pravilima:*

- *$t_I \coloneqq I_C(t)$, ako je t konstantski simbol, tj. $t \in \Gamma$*
- *$t_I \coloneqq \rho(t)$, ako je t individualna varijabla, tj. $t \in X$*
- *$t_I \coloneqq I_F(f)(I^*_\rho(t_1),\ I^*_\rho(t_2), \ldots,\ I^*_\rho(t_n))$, ako je t term oblika $f(t_1, \ldots,\ t_n)$.*

Definicija 1.2.3. *Neka je dana zamjena ρ na skupu $\{x_1, \ldots\ x_n\} \subseteq X$ i vrijednosti domene diskursa $\{d_1, \ldots\ d_n\} \subseteq U$, tada definiramo **modificiranu zamjenu** na sljedeći način:*

$$\rho(x_1|d_1, \ldots, x_n|d_n)(y) = \begin{cases} d_i, & ako\ y = x_i \\ \rho(y), & inače \end{cases} \tag{1.1}$$

Definicija 1.2.4. ***Evaluacija formule*** *F jezika $\mathcal{L}$ je funkcija $I_\rho : F_{\mathcal{L}} \to \{0,\ 1\}$ za koju vrijedi:*

1. *Ako je W* ***atomarna formula*** *koja se sastoji od n-arnog predikata P:*

$$I_\rho(W) := \begin{cases} 1, & ako\ (I^*_\rho(t_1), \ldots, I^*_\rho(t_n)) \in I_P(P) \\ 0, & inače \end{cases}$$

2. *Logički simboli $\neg$, $\vee$, $\wedge$, $\to$, $\leftrightarrow$ interpretiraju se kao funkcije na skup $\{0,\ 1\}$ koristeći odgovarajuće vrijednosti funkcija $I_\neg$, $I_\wedge$, $I_\vee$, $I_\wedge$, $I_\to$, $I_\leftrightarrow$ dane tablicom 1.1. Sada se evaluacija formule $I_\rho : F_{\mathcal{L}} \to \{0,\ 1\}$ može proširiti na sljedeći način.*

   Ako su W_1, i W_2 ***formule****:*

$$\begin{aligned} &I_\rho(\neg W_1) := I_\neg(I_\rho(W_1)) \\ &I_\rho(W_1 \wedge W_2) := I_\wedge(I_\rho(W_1), I_\rho(W_2)) \\ &I_\rho(W_1 \vee W_2) := I_\vee(I_\rho(W_1), I_\rho(W_2)) \\ &I_\rho(W_1 \to W_2) := I_\to(I_\rho(W_1), I_\rho(W_2)) \\ &I_\rho(W_1 \leftrightarrow W_2) := I_\leftrightarrow(I_\rho(W_1), I_\rho(W_2)) \end{aligned}$$

3. *$\exists x\ (W)$: ako postoji bilo koji element domene diskursa za koji se formula W evaluira na 1 (kao istinita), tada je cijela izjava istinita, u protivnom je izjava lažna.*

$$I_\rho(\exists x(W)) := \begin{cases} 1, & ako\ 1 \in \{I_{\rho(x|d)}(W)\ :\ d \in U\} \\ 0, & inače \end{cases}$$

4. *$\forall x\ (W)$: ako postoji bilo koji element domene diskursa za koji se formula W evaluira na 0 (kao lažna), tada je cijela izjava lažna, u protivnom je izjava istinita.*

$$I_\rho(\forall x(W)) := \begin{cases} 0, & ako\ 0 \in \{I_{\rho(x|d)}(W)\ :\ d \in U\} \\ 1, & inače \end{cases}$$

W_1	W_2	$I_\neg(W_1)$	$I_\wedge(W_1,\ W_2)$	$I_\vee(W_1,\ W_2)$	$I_\to(W_1,\ W_2)$	$I_\leftrightarrow(W_1,\ W_2)$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Tablica 1.1: Evaluacija složenih formula

Napomena 1.2.5. *U prethodnoj definiciji implicitno koristimo tzv.* ***pretpostavku zatvorenog svijeta*** *(eng. closed world assumption), tj. sve što nije eksplicitno element domene diskursa ne postoji. Domena diskursa enumerira sve postojeće stvari. Suprotna pretpostavka naziva se* ***pretpostavka otvorenog svijeta****. Što bi se desilo kada bi koristili pretpostavku otvorenog svijeta pri evaluaciji primjerice atomarnih formula?*
Neka je zadan jezik $\mathcal{L} = (\Gamma, \Omega, \Pi, X)$ *na sljedeći način:*
$\Gamma := \{'Sokrat', 'Hektor'\},\ \Omega := \{\},\ \Pi := \{Filozof\},\ X := \{\}.$

Neka je dana sljedeća interpretacija: $I = (U,\ I_C,\ I_F,\ I_P)$ *gdje je:*
$U = \{Filozof\ Sokrat, Ratnik\ Hektor\}$
$I_C : \Gamma \to U,\ \{'Sokrat' \mapsto Filozof\ Sokrat,\ 'Hektor' \mapsto Ratnik\ Hektor\}$
$I_P(Filozof) = \{(Filozof\ Sokrat)\} \subseteq U$

Dakle naša domena diskursa sadrži filozofa Sokrata koji je filozof i reprezentiran je simbolom $'Sokrat'$*, te Ratnika Hektora, koji nije filozof te je reprezentiran simbolom 'Hektor'. Provjerimo istinitost sljedećih formula:*

$Filozof('Sokrat')$ *– istina, jer znamo da je Sokrat filozof.*
$Filozof('Hektor')$ *– laž, jer znamo da Hektor nije filozof.*
$Filozof('Aristotel')$ *– ovisi o pretpostavci*
Pretpostavka zatvorenog svijeta *– laž jer Aristotel nije element naše domene diskursa, stoga ne može biti filozof.*
Pretpostavka otvorenog svijeta *– ne znamo, jer Aristotel nije element domene diskursa, možda je filozof, a možda nije. Formula se ne može evaluirati.*
Mi ćemo očito koristiti pretpostavku zatvorenog svijeta, jer to je upravo ono što želimo imati u našoj deduktivnoj bazi podataka. Relevantni su samo podaci koji se nalaze u bazi, ono što je izvan baze za nas ne postoji, te se evaluira kao laž.

Primjer 1.2.6. *Neka je zadan jezik* $\mathcal{L} = (\Gamma, \Omega, \Pi, X)$*, gdje je:* $\Gamma = \{a,\ b\},\ \Omega = \{f,\ g\},\ \Pi = \{P\},\ X = \{x,\ y,\ z\}$*. Da li je formula* $F \equiv P(f(a),\ g(b, x))$ *istinita? Treba nam interpretacija. Neka je* $I = (U,\ I_C,\ I_F,\ I_P)$ *zadana na sljedeći način:*

$$U = \mathbb{N}$$

$$I_C : \Gamma \to U,\ \{a \mapsto 5,\ b \mapsto 3\}$$

$$I_F(f) : U \to U,\ n \mapsto n^2$$

$$I_F(g) : U \times U \to U,\ (n,\ m) \mapsto n + m$$

$$I_P(P) = \{(n,\ m) \in \mathbb{N}^2\ :\ n < m\} \subseteq U \times U$$

Tada imamo $5^2 < 3 + x$. *No još uvijek ne možemo znati da li je formula istinita. Treba nam zamjena varijable* ρ.

- *Za* $\rho : X \to U$, $\{x \mapsto 1\}$, $5^2 < 3 + 1$ *nije istina jer* $(25,\ 4) \notin \{(n,\ m) \in \mathbb{N}^2\ :\ n < m\}$.
- *Za* $\rho : X \to U$, $\{x \mapsto 99\}$, $5^2 < 3+99$ *je istina jer* $(25,\ 102) \in \{(n,\ m) \in \mathbb{N}^2\ :\ n < m\}$.

Još interesantnije pitanje koje bi mogli postaviti je: za koje zamjene varijable ρ *je formula* F *istinita? Ovakvi upiti predstavljati će osnovu logičkog programiranja, čime ćemo se baviti u sljedećem odjeljku.*

1.3 Modeli

U ovom odjeljku pokušati ćemo dati odgovor na sljedeće pitanje. Koja interpretacija i zamjena varijable čine formulu istinitom? Postoji neograničeno mnogo interpretacija i zamjena varijabli koje bi smo mogli isprobati. To bi očito bila loša taktika, budući da želimo izgraditi deduktivni sustav koji bi se morao izvršavati u konačnom vremenu. Stoga umjesto da isprobavamo sve interpretacije, ideja je da koristimo interpretacije koje su reprezentanti za cijelu klasu svih interpretacija. No postoje li takve interpretacije, i ako postoje za koje tipove formula? Pokazat ćemo da za klauzule (određene tipove zatvorenih formula) tzv. **Herbrandove interpretacije** će biti upravo ti reprezentanti. One će biti od ključnog značaja za izgradnju deduktivnog sustava, odnosno naše deduktivne baze podataka.

Definicija 1.3.1. *Kažemo da je interpretacija* I ***model*** *za zatvorenu formulu* W *ako se formula* W *evaluira kao istinita u odnosu na interpretaciju* I.
Kažemo da je interpretacija I ***model*** *za skup zatvorenih formula* $\boldsymbol{W}$, *ako je* I *model za svaku formulu* $W \in \boldsymbol{W}$.

Primjer 1.3.2. *Promotrimo sljedeća dva primjera:*

- $W \equiv \forall x \exists y\ (P(x,\ y))$
  - *Neka je* I *interpretacija koja preslikava predikat* P *na relaciju* $<$ *na skupu* $\mathbb{N}$. *Tada je* I *model od* W. *Formula* W *se naziva još i* ***činjenica*** *u odnosu na interpretaciju* I.
- $W \equiv \exists x \forall y\ (P(x,\ y))$
  - *Neka je* I *ista interpretacija kao i u gornjem primjeru. Tada interpretacija* I *nije model za formulu* W.

Definicija 1.3.3. *Za skup zatvorenih formula* $\boldsymbol{W}$ *kažemo da je* ***ispunjiv****, ako* $\boldsymbol{W}$ *ima model. Za skup zatvorenih formula* $\boldsymbol{W}$ *kažemo da je* ***neispunjiv*** *ukoliko nema niti jedan model.*

Definicija 1.3.4. *Kažemo da su zatvorene formule W_1, i W_2 **logički (semantički) ekvivalentne** (u oznaci $W_1 \Leftrightarrow W_2$) ako je $I(W_1) = I(W_2)$, za svaku interpretaciju I.*

Definicija 1.3.5. *Za zatvorenu formulu W kažemo da je **logička (semantička) posljedica** skupa zatvorenih formula $\mathbf{W}$, ako je svaki model skupa formula $\mathbf{W}$, također i model za formulu W. Tu činjenicu kraće zapisujemo $\mathbf{W} \models W$. Čitamo $\mathbf{W}$ logički (semantički) povlači W.*

Lema 1.3.6. *Neka je $\mathbf{W}$ skup zatvorenih formula i W zatvorena formula, tada vrijedi: $\mathbf{W} \models \neg W$ ako i samo ako je $\mathbf{W} \cup \{W\}$ neispunjiv skup.*

Dokaz. Pretpostavimo prvo da je formula $\neg W$ semantička posljedica skupa formula $\mathbf{W}$. Tada je svaka interpretacija I koja je model za $\mathbf{W}$ ujedno i model za $\neg W$, stoga I ne može biti model za W pa samim time ni za $\mathbf{W} \cup \{W\}$. Po definiciji 1.3.3 skup $\mathbf{W} \cup \{W\}$ je neispunjiv. Pretpostavimo sada da je skup $\mathbf{W} \cup \{W\}$ neispunjiv, te neka je interpretacija I model za skup formula $\mathbf{W}$. Tada zbog pretpostavke da je skup $\mathbf{W} \cup \{W\}$ neispunjiv, slijedi da I ne može biti model za formulu W, dakle I je tada model za $\neg W$. Po definiciji 1.3.5 vrijedi da je formula $\neg W$ semantička posljedica skupa formula $\mathbf{W}$, tj. $\mathbf{W} \models \neg W$. □

Napomena 1.3.7. *Da je skup zatvorenih formula $\mathbf{W}$ neispunjiv možemo dokazati tako da nađemo jednu formulu W iz tog skupa tako da $\neg W$ slijedi iz preostalih formula tog skupa.*

Definicija 1.3.8. *Za skup zatvorenih formula $\mathbf{W}$ kažemo da je univerzalan (u oznaci $\models \mathbf{W}$), ako je svaka interpretacija model za $\mathbf{W}$. Formule skupa $\mathbf{W}$ tada se nazivaju **tautologije**, tj. formula je tautologija ako je istinita za svaku interpretaciju.*

Primjer 1.3.9. *Tautologije možemo koristiti za izvođenje logičkih ekvivalencija, koje onda možemo koristiti kao sljedeća pravila transformacije.*

- $A \Leftrightarrow \neg\neg A$
- $A \wedge B \Leftrightarrow B \wedge A,\ A \vee B \Leftrightarrow B \vee A$
- $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C),\ A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$
- $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B, \neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$
- $A \rightarrow B \Leftrightarrow \neg A \vee B,\ A \wedge B \Leftrightarrow \neg(A \rightarrow (\neg B)),\ A \vee B \Leftrightarrow (\neg A \rightarrow B)$
- $A \leftrightarrow B \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$
- $\forall x\,(P(x)) \Leftrightarrow \neg\exists x\,(\neg P(x)),\ \exists x\,(P(x)) \Leftrightarrow \neg\forall x\,(\neg P(x))$

Definicija 1.3.10. *Skup **literala** $L_{\mathcal{L}}$ sastoji se od svih atomarnih formula $A \in A_{\mathcal{L}}$, te odgovarajućih negiranih atomarnih formula $\neg A$. Atomarne formule nazivamo **pozitivni literali**. Negirane atomarne formule nazivamo **negativni literali**. Ukoliko neka atomarna formula ne sadrži varijable, tada se naziva **osnovni literal**.*

Definicija 1.3.11. *Univerzalno zatvorenje disjunkcije literala nazivamo **klauzula**, tj. $\forall\ (L_1 \vee L_2 \vee \ldots \vee L_n),\ L_i \in L_{\mathcal{L}}$.*
*Klauzula koja sadrži najviše jedan pozitivni literal naziva se **Hornova klauzula**.*
*Hornova klauzula bez pozitivnog literala naziva se **ciljna klauzula**.*
*Hornova klauzula sa točno jednim pozitivnim literalom naziva se **definitna klauzula**.*
*Hornova klauzula sa jednim pozitivnim literalom i bez negativnih literala naziva se **činjenična klauzula**.*

Napomena 1.3.12. *Hornove klauzule igraju ključnu ulogu u deduktivnim bazama podataka. Naime, vidjet ćemo da je logički program zapravo skup Hornovih klauzula. Sjetimo se sljedećeg pravila transformacije u primjeru 1.3.9:*

$$\neg A \vee B \Leftrightarrow A \rightarrow B$$

Tada možemo uočiti da definitne Hornove klauzule zapravo predstavljaju implikaciju, tj.

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_{n-1} \vee A_n \Leftrightarrow (A_1 \wedge A_2 \wedge \ldots \wedge A_{n-1}) \rightarrow A_n$$

Dakle definitne Hornove klauzule su zapravo implikacije sa skupom preduvjeta $\{A_1, \ldots, A_{n-1}\}$ i jednim zaključkom A_n. Ako nemamo zaključka, tada imamo ciljnu klauzulu, ili mogli bismo reći upit. Ako imamo samo jedan pozitivan literal i nemamo negativnih literala tada imamo činjeničnu klauzulu, ili jednostavnije činjenicu.

Definicija 1.3.13. ***Herbrandov svemir (domena)** $U_{\mathcal{L}}$ jezika $\mathcal{L}$ sastoji se od svih osnovnih terma. **Herbrandova baza** $B_{\mathcal{L}}$ sastoji se od svih osnovnih atomarnih formula koje su formirane tako da kao argumente imaju elemente Herbrandovog svemira.*

Primjer 1.3.14. *Neka je zadan jezik $\mathcal{L} = (\Gamma, \Omega, \Pi, X)$ na sljedeći način:*
$\Gamma = \{a,\ b\},\ \Omega = \{f,\ g\},\ \Pi = \{P\},\ X = \{\}$.
Herbrandov svemir $U_{\mathcal{L}}$ sastoji se od svih osnovnih terma, tj. onih koji se mogu generirati koristeći samo funkcijske i konstantske simbole.

$$U_{\mathcal{L}} = \{a,\ f(a),\ g(a),\ f(f(a)),\ f(g(a)), \ldots\} \cup \{b,\ f(b),\ g(b),\ f(f(b)),\ f(g(b)), \ldots\}$$

Herbrandova baza $B_{\mathcal{L}}$ sastoji se od svih osnovnih atomarnih formula koje možemo generirati koristeći elemente Herbrandovog svemira.

$$\begin{aligned} B_{\mathcal{L}} =& \{P(a),\ P(f(a)),\ P(g(a)),\ P(f(f(a))),\ P(f(g(a))), \ldots\} \cup \\ & \{P(b),\ P(f(b)),\ P(g(b)),\ P(f(f(b))),\ P(f(g(b))), \ldots\} \end{aligned}$$

Definicija 1.3.15. *Herbrandova interpretacija je uređena četvorka* $I = (U,\ I_C,\ I_F,\ I_P)$ *gdje je:*

- $U = U_{\mathcal{L}}$*, kao domenu diskursa uzimamo Herbrandov svemir.*
- $I_C(c) \coloneqq c$*, svaki konstantski simbol* $c \in \Gamma$ *interpretiramo kao njega samoga.*
- $I_F(f) : U_{\mathcal{L}} \times \ldots \times U_{\mathcal{L}} \to U_{\mathcal{L}},\ f(t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)$*, svaki funkcijski simbol* $f \in \Omega$ *interpretiramo kao njega samog.*

Svaki entitet jezika $\mathcal{L}$ *preslikavamo na ekvivalentan simbol Herbrandovog svemira.*

Primjer 1.3.16. *Neka je jezik* $\mathcal{L}$ *zadan kao u primjeru 1.3.14.*
Hebrandova interpretacija tada evaluira term $f(g(a))$ *u* $f(g(a)) \in U_{\mathcal{L}}$*. Za danu zamjenu varijable* $\rho(x) = g(f(b))$*, term* $f(x)$ *evaluira se na* $f(g(f(b))) \in U_{\mathcal{L}}$*. Dakle, trebamo imati na umu da term* $f(g(a))$ *i element Herbrandovog svemira* $f(g(a))$ *nisu isti, premda izgledaju isto. Prvi je samo simbol, dok drugi nešto znači.*

Napomena 1.3.17. *Primijetimo u definiciji 1.3.15 da su* U, I_C *i* I_F *isti za sve Herbrandove interpretacije. Herbrandove interpretacije jedino se razlikuju u odnosu na* ***interpretaciju predikata*** I_P*. Za jezik dan kao u primjeru 1.3.14, za dve različite Herbrandove interpretacije* $P(a)$ *može biti istinita u jednoj, a lažna u drugoj. Stoga Herbrandove interpretacije mogu biti definirane navođenjem svih atomarnih formula iz Herbrandove baze koje se evaluiraju kao istinite. Herbrandova interpretacija može se identificirati sa podskupom Herbrandove baze i obratno.*

Definicija 1.3.18. *Za Herbrandovu interpretaciju koja je model za skup zatvorenih formula* $\boldsymbol{W}$ *kažemo da je* ***Herbrandov model****.*

Primjer 1.3.19. *Neka je dana sljedeća formula:*

$$W \equiv \forall x,\ y\ (voli(x,\ y) \to voli(y, x))$$

Tada za implicitno zadani jezik $\mathcal{L}$*:*

$I_1 = \{voli(Romeo,\ Julija),\ voli(Julija,\ Romeo)\}$, I_1 *je Herbrandov model.*

$I_2 = \{voli(Romeo,\ Julija),\ voli(Julija,\ William)\}$, I_2 *nije Herbrandov model.*

Lema 1.3.20. *Za dani skup klauzula* $\boldsymbol{W}$ *vrijedi:*
$\boldsymbol{W}$ *ima model ako i samo ako* $\boldsymbol{W}$ *ima Herbrandov model.*
Skup $\boldsymbol{W}$ *je neispunjiv ako i samo ako* $\boldsymbol{W}$ *nema Herbrandov model.*

Dokaz. Za dokaz leme 1.3.20 vidi [2]. □

Napomena 1.3.21. *Iz leme 1.3.20 vidimo da se svi simboli klauzule ili skupa klauzula mogu interpretirati na čisto sintaktički način. Ako postoji sintaktička mogućnost da se zadovolji klauzula ili skup klauzula (odnosno da klauzula ili skup klauzula bude ispunjiv) tada će također postojati i neka (više ili manje korisna) semantička interpretacija. Koristeći lemu 1.3.20 možemo testirati neispunjivost skupa* $\boldsymbol{W} \cup \{W\}$ *(sjetimo se moramo pokazati da* $\boldsymbol{W} \models \neg W$*). Ali sada moramo pokazati samo egzistenciju ili neegzistenciju jednog Herbrandovog modela, umjesto testiranja svih postojećih modela. Naravno uz dozu opreza, jer lema 1.3.20 vrijedi samo za klauzule, ne generalno sve zatvorene formule, no to je upravo ono što će nam i trebati. Dakle da zaključimo ovo poglavlje. Cilj nam je izgraditi deduktivnu bazu podataka. Da bismo to mogli trebaju nam pravila kako bismo mogli dobiti novo znanje iz podataka u bazi, a to znači da moramo konstruirati Herbrandove interpretacije. Ako su te Herbrandove interpretacije ujedno i model, tada ćemo ih moći iskoristit za daljnju evaluaciju i optimizaciju upita kao što ćemo vidjeti u sljedećem poglavlju.*

Poglavlje 2

Datalog i deduktivne baze podataka

Ovo poglavlje posvećeno je formalnom proučavanju jezika Datalog. Datalog je jezik specifično definiran za realizaciju deduktivnih baza podataka. To je jezik koji omogućuje postavljanje složenijih rekurzivnih upita kakvi se ne mogu izraziti u standardnom SQL-u. Sintaktički je sličan deklarativnom programskom jeziku Prolog, no ipak postoje određene razlike. Mogli bismo reći da je jezik Datalog podskup Prologa budući da je Datalog jezik namijenjen za užu upotrebu odnosno realizaciju deduktivnih baza podataka. Jedno od osnovnih svojstava i prednost jezika Datalog je da sigurni i stratificirani Datalog program staje za svaki upit na konačnom skupu pravila i činjenica.

Datalog je razvijen 1978. godine od strane Hervé Gallairea i Jack Minkera u sklopu znanstvenog istraživanja baza podataka. Koristio se kao temelj za teoriju ekspretnih sustava tijekom osamdesetih godina prošlog stoljeća. Danas se koristi u širokom spektru raznih znanstvenih i stručnih disciplina. Čak je i SQL3 standard prepoznao važnost dedukcije u relacijskim bazama podataka te je uveo upite koji omogućuju rekurzije.

2.1 Sintaksa Dataloga

Relacijske baze podataka razlikuju dva jezika:

- **Jezik za opis podataka** (eng. data description language – DDL) omogućuje stvaranje shema i pogleda (eng. views).
- **Jezik za manipulaciju podataka** (eng. data manipulation language – DML) omogućuje održavanje podataka i postavljanje upita.

U klasičnim relacijskim bazama podataka ta dva jezika uklopljena su u jedan jezik SQL.

U **deduktivnim bazama podataka** nemamo striktne podjele na logički dio (DDL) i dio za manipulaciju podataka (DML) već su podaci i upiti specificirani logičkim formulama.

Svaki predikat je napisan Hornovim klauzulama oblika:

- $\forall\ (L_1 \vee L_2 \vee \cdots \vee L_n),\ L_i \in L_{\mathcal{L}}$ gdje postoji najviše jedan pozitivan literal L_j

Logičko programiranje uvodi malo drugačiju notaciju Hornovih klauzula:

- $L_j \leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_n.$ gdje '$\leftarrow$' predstavlja implikaciju, ',' konjunkciju, dok '.' predstavlja kraj klauzule.

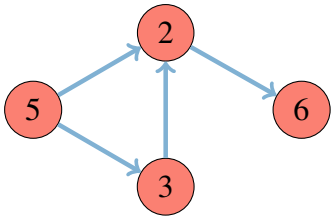
Deduktivna baza podataka sastoji se od činjenica i pravila:

- Skup činjenica koji nazivamo **ekstenzijska baza podataka** (eng. extensional database – EDB) – ako se u činjenicama ne nalaze funkcije tada skup može biti pohranjen kao jednostavna relacijska baza podataka.
- Skup pravila koji nazivamo **intenzijska baza podataka** (eng. intentional database – IDB) – odražava ideju pogleda u relacijskim bazama podataka, ali povrh toga dopušta i rekurzije.

Definicija 2.1.1. ***Klauzula baze podataka*** *(eng. DB-clause) definirana je na sljedeći način:*

- $A \leftarrow L_1, \ldots, L_n.$*, gdje su* $A \in A_{\mathcal{L}}$ *atomarna formula, a* $L_i \in L_{\mathcal{L}}$ *literali. Atomarnu formulu A nazivamo* ***glava****, a* $L_1, \ldots, L_n$ ***tijelo*** *klauzule baze podataka. Alternativno, klauzulu baze podataka možemo pisati i kao* $A :- L_1, \ldots, L_n$.
- *Klauzule baze podataka gdje je* $n > 0$ *nazivamo* ***pravila****.*
- *Klauzule baze podataka gdje je* $n = 0$ *i A osnovna atomarna formula nazivamo* ***činjenice****.*
- *Klauzule baze podataka čije se tijelo sastoji samo od pozitivnih literala nazivamo* ***definitne*** *klauzule baze podataka.*

Primjer 2.1.2. *Najvažnije svojstvo Dataloga je mogućnost korištenja rekurzija. Jedno od osnovnih pitanja u teoriji grafova je postoji li put od jednog do drugog čvora u grafu. Odnosno može li se određeni čvor dohvatiti iz nekog čvora. Neka je dan usmjereni graf na slici 2.1.*



Slika 2.1: Usmjereni graf

Tada sve usmjerene putove, odnosno bridove možemo zapisati sljedećim činjenicama.

$$\left.\begin{array}{l} brid(3,2). \\ brid(2,6). \\ brid(5,2). \\ brid(5,3). \end{array}\right\} \text{činjenice (EDB)}$$

Kako bismo mogli dobiti sve putove u grafu trebaju nam pravila. Primjerice putove od čvora 5 *do čvora* 6 *i od čvora* 3 *do čvora* 6 *možemo dobiti uvođenjem sljedeća dva pravila.*

$$\left.\begin{array}{l} put(X,Y) :- brid(X,Y). \\ put(X,Y) :- brid(X,Z), put(Z,Y). \end{array}\right\} \text{pravila (IDB)}$$

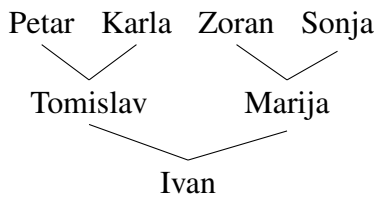
Prvo pravilo nam jednostavno kaže da ako postoji brid od jednog čvora do drugog, da onda postoji i put između ta dva čvora. Drugo pravilo nam kaže da ako postoji brid od čvora X do čvora Z, te put od tog istog čvora Z do čvora Y da tada postoji put od X do Y. Drugo pravilo je očito rekurzivno jer generira znanje (putove) iz prethodnih iteracija istog predikata.

Definicija 2.1.3. ***Definicija predikatnog simbola*** *p je skup činjenica i pravila Datalog programa gdje se p pojavljuje u glavi pravila. Definiciju predikatnog simbola označavamo kao def(p).*

Napomena 2.1.4. *U Datalog programu može se javiti problem s varijablama koje se nalaze u glavama pravila. Tako npr. pravilo $p(X) :- r(Y)$ je problematično jer ne znamo što znači. Znači li da ako postoji zamjena varijable Y koja čini $r(Y)$ istinitim da je onda i $p(X)$ istinit za sve moguće zamjene varijable X ili samo za $X = Y$? Da bi izbjegli takve dvosmislenosti uvodimo sljedeću restrikciju.*
Restrikcija*: u Datalog programu sve varijable koje se nalaze u predikatu glave pravila moraju se uvijek pojaviti i u nekom literalu tijela pravila. Sličan problem mogao bi se pojaviti i ako bi konstanta u predikatu glave pravila ovisila o varijablama u tijelu pravila npr. $p(a) :- r(X)$, stoga takva pravila zabranjujemo.*

Definicija 2.1.5. ***Upit*** *(query) baze podataka definiramo kao:* $?L_1, \ldots, L_n$*, gdje su:* $L_i \in L_{\mathcal{L}},\ n > 0$*. Alternativni zapisi su:* $\leftarrow L_1, \ldots, L_n$ *ili* $: - L_1, \ldots, L_n$*.*
Upit koji se sastoji samo od pozitivnih literala naziva se ***definitni upit****.*
Definitni upit gdje je $n = 1$ *naziva se* ***Datalog upit****.*

Primjer 2.1.6. *Neka je dano sljedeće obiteljsko stablo:*



Slika 2.2: Obiteljsko stablo

Prikažimo stablo 2.2 skupom činjenica i pravila kao deduktivnu bazu podataka.

$$roditelj(Tomislav, Ivan).\ roditelj(Marija, Ivan).$$
$$roditelj(Petar, Tomislav).\ roditelj(Karla, Tomislav).$$
$$roditelj(Zoran, Marija).\ roditelj(Sonja, Marija).$$
$$žensko(Marija).\ žensko(Sonja).\ žensko(Karla).$$

$$baka(X, Y) :- roditelj(X, Z),\ roditelj(Z, Y),\ žensko(X).$$

Ukoliko bi nas zanimalo tko je Ivanova baka, tada bismo postavili sljedeći Datalog upit: $?baka(X, Ivan)$*. Odgovor Dataloga bio bi* $baka(Karla, Ivan)$*. i* $baka(Sonja, Ivan)$*.*
O samom načinu evaluacije upita biti će više riječi u narednim odjeljcima.

Napomena 2.1.7. *Zbog jednostavnosti u Datalogu su neki češće korišteni aritmetički predikati* $\{<, \leq, >, \geq, =, \neq\}$ *unaprijed definirani za korištenje u tijelima pravila kao npr.* $znamenka(X) :- prirodniBroj(X),\ X \leq 9$*. Isto vrijedi i za jednostavne aritmetičke funkcije* $\{+, -, * /\}$*. npr.* $suma(X, Y, Z) :- Z = X + Y$*.*

2.2 Klase Datalog programa i stratifikacija

Definicija 2.2.1. *Ukoliko se predikatni simboli koji definiraju činjenice nikada ne pojavljuju u glavama pravila tada se skup klauzula baze podataka naziva* $\textbf{\textit{Datalog}}^{f,\ neg}$ ***program****.*
U ovisnosti o tome dozvoljavamo li funkcije ili negacije možemo razlikovati nekoliko klasa Datalog jezika:

- ***Datalog^{neg}*** *programi – ne sadrže funkcijske simbole*
- ***Datalog^{f}*** *programi – ne sadrže negativne literale.*
- ***Datalog*** *programi – ne sadrže niti negativne literale niti funkcijske simbole.*

Datalog programe možemo razlikovati i po njihovoj ovisnosti među predikatima.

Definicija 2.2.2. ***Graf predikatnih ovisnosti*** *nekog programa P sastoji se od:*

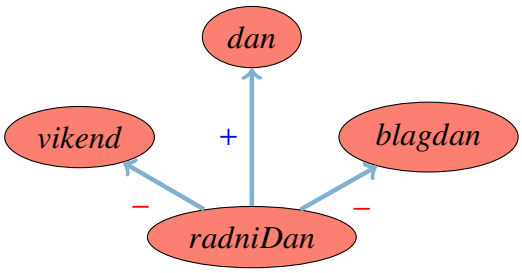
- ***Čvorova*** *– svaki predikatni simbol $p \in P$ predstavlja jedan čvor u grafu.*
- ***Usmjerenih bridova*** *– od čvora p do čvora q ako se q pojavljuje u tijelu pravila sa predikatom p u glavi.*
- *Brid je* ***negativan*** *ako se q pojavljuje u negativnom literalu, u protivnom je* ***pozitivan****.*

Definicija 2.2.3. *Kažemo da je program* ***hijerarhijski*** *ukoliko graf predikatnih ovisnosti tog programa ne sadrži cikluse. Ukoliko graf predikatnih ovisnosti nekog programa sadrži cikluse kažemo da je program* ***rekurzivan****. Za program kažemo da je* ***stratificiran*** *ukoliko se ciklusi u grafu predikatnih ovisnosti tog programa sastoje samo od* ***pozitivnih*** *bridova.*

Primjer 2.2.4. *Radi bolje ilustracije prethodne definicije 2.2.3 pogledajmo nekoliko primjera programa te njima odgovarajućih grafova predikatnih ovisnosti.*

a)

$$radniDan(X) :- dan(X),\ \neg vikend(X),\ \neg blagdan(X).$$

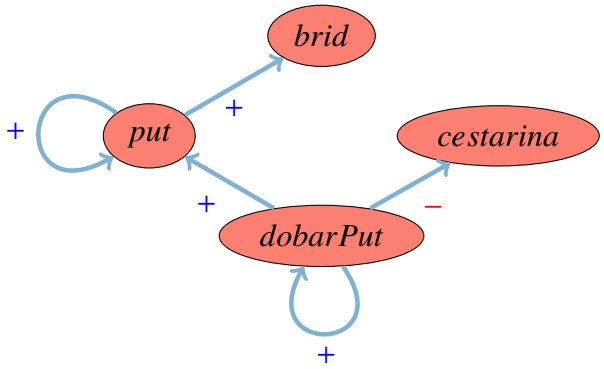


Slika 2.3: Graf predikatnih ovisnosti hijerarhijskog programa

Graf predikatnih ovisnosti ne sadrži cikluse pa je stoga program hijerarhijski.

b)

$$put(X,Y) :- brid(X,Y).$$
$$put(X,Y) :- brid(X,Z),\ put(Z,Y).$$
$$dobarPut(X,Y) :- put(X,Y),\ \neg cestarina(X,Y).$$
$$dobarPut(X,Y) :- dobarPut(X,Z),\ dobarPut(Z,Y).$$

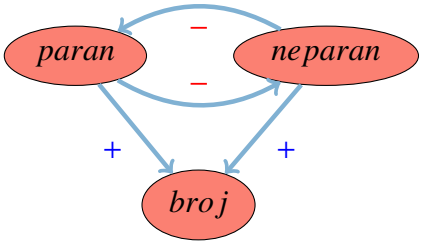


Slika 2.4: Graf predikatnih ovisnosti rekurzivnog i stratificiranog programa

Graf predikatnih ovisnosti sadrži cikluse od kojih ni jedan nije negativan, stoga je riječ o rekurzivnom i stratificiranom programu.

c)

$$parni(X) :- broj(X),\ \neg neparan(X).$$
$$neparan(X) :- broj(X),\ \neg paran(X).$$



Slika 2.5: Graf predikatnih ovisnosti rekurzivnog i nestratificiranog programa

Graf predikatnih ovisnosti sadrži negativne cikluse, stoga je riječ o rekurzivnom i nestratificiranom programu.

Definicija 2.2.5. ***Stratifikacija*** *nekog programa P je particija $P = P_1 \cup \ldots \cup P_n$ od P u dijelove programa (strate) tako da vrijedi sljedeće:*

- *Definicija svakog predikatnog simbola je podskup neke strate.*
- *Definicija predikatnog simbola koji se pojavljuje u pozitivnom literalu tijela klauzule baze podataka u P_i pripada particiji P_j, za $j \leq i$.*
- *Definicija predikatnog simbola koji se pojavljuje u negativnom literalu tijela klauzule baze podataka u P_i pripada particiji P_j, za $j < i$.*

Napomena 2.2.6. *Osnovna ideja stratifikacije je raslojavanje programa tako da su definicije predikata koji se pojavljuju u negativnim literalima uvijek prethodno dane u prijašnjim slojevima. Time efektivno isključujemo korištenje negacija unutar rekurzija. Može se dokazati da je program stratificiran, ako i samo ako ima stratifikaciju.*

Sada ćemo dati pseudokod algoritma za stratifikaciju **Datalog$^{\text{f, neg}}$** programa.

Algoritam 2.1: Pseudokod algoritma za stratifikaciju **Datalog$^{\text{f, neg}}$** programa.

```
Input: Datalog^{f, neg} program;
brojPredikata := 0
for each predikatni simbol p do
    stratum[p] := 1;
    brojPredikata := brojPredikata + 1
maxstratum := 1;
repeat
    prethodniMaxStratum := maxstratum;
    for each klauzulu baze podataka sa predikatom p u glavi do
        for each negativni literal tijela klauzule s predikatom q do
            stratum[p] := max(stratum[p], stratum[q] + 1);
        for each pozitivni literal tijela klauzule s predikatom q do
            stratum[p] := max(stratum[p], stratum[q]);
        maxstratum := max {stratum[p] : p je predikat};
until (maxstratum > brojPredikata) ∨ (maxstratum == prethodniMaxStratum);
if maxstratum > brojPredikata then
    return 'program nije stratificiran';
else
    for i = 1 to maxstratum do
        P_i := ⋃_{p ∈ i-tog stratuma} def(p);
```

Primjer 2.2.7. *Ovim primjerom ilustrirati ćemo rad algoritma za stratifikaciju 2.1. Neka nam je dan sljedeći intenzijski dio baze podataka kao ulazni Datalog^{neg} program:*

$$put(X,Y) :- brid(X,Y). \tag{2.1a}$$
$$put(X,Y) :- brid(X,Z),\ put(Z,Y). \tag{2.1b}$$
$$dobarPut(X,Y) :- put(X,Y),\ \neg cestarina(X,Y). \tag{2.1c}$$
$$dobarPut(X,Y) :- dobarPut(X,Z),\ dobarPut(Z,Y). \tag{2.1d}$$

***Inicijalizacija**:*

$$stratum[brid] := 1;$$
$$stratum[put] := 1;$$
$$stratum[cestarina] := 1;$$
$$stratum[dobarPut] := 1;$$
$$maxstratum := 1;$$

***Prva iteracija**:*

***Pravilo** 2.1a*
$stratum[put] := max(stratum[put],\ stratum[brid]) = 1$
***Pravilo** 2.1b*
$stratum[put] := max(stratum[put],\ stratum[brid]) = 1$
$stratum[put] := max(stratum[put],\ stratum[put]) = 1$
***Pravilo** 2.1c*
$stratum[dobarPut] := max(stratum[dobarPut],\ stratum[cestarina] + 1) = 2$
$stratum[dobarPut] := max(stratum[dobarPut],\ stratum[put]) = 2$
***Pravilo** 2.1d*
$stratum[dobarPut] := max(stratum[dobarPut],\ stratum[dobarPut]) = 2$
$maxstratum = 2$

***Druga iteracija:** nema promjene u maxstrata, odnosno na kraju druge iteracije $maxstrata = 2$ i uvjetom $prethodniMaxStrata == maxstrata$, algoritam izlazi iz petlje. $brojPredikata = 4$, stoga algoritam radi sljedeću stratifikaciju:*

$$P_1 = \{def(brid),\ def(put),\ def(cestarina)\};$$
$$P_2 = \{def(dobarPut)\};$$

Primjer 2.2.8. *Ovim primjerom ilustrirati ćemo rad algoritma za stratifikaciju 2.1 programa koji neće biti stratificiran. Neka je dan sljedeći intenzijski dio baze podataka:*

$$paran(X) :- broj(X),\ \neg neparan(X). \tag{2.2a}$$
$$neparan(X) :- broj(X),\ \neg paran(X). \tag{2.2b}$$

***Inicijalizacija**:*

$stratum[broj] := 1;$
$stratum[paran] := 1;$
$stratum[neparan] := 1;$
$maxstratum := 1$

***Prva iteracija**:*

Pravilo 2.2*a*
$stratum[paran] := max(stratum[paran],\ stratum[neparan] + 1) = 2$
$stratum[paran] := max(stratum[paran],\ stratum[broj]) = 2$
Pravilo 2.2*b*
$stratum[neparan] := max(stratum[neparan],\ stratum[paran] + 1) = 3$
$stratum[neparan] := max(stratum[neparan],\ stratum[broj]) = 3$
$maxstratum = 3$

***Druga iteracija**:*

Pravilo 2.2*a*
$stratum[paran] := max(stratum[paran],\ stratum[neparan] + 1) = 4$
$stratum[paran] := max(stratum[paran],\ stratum[broj]) = 4$
Pravilo 2.2*b*
$stratum[neparan] := max(stratum[neparan],\ stratum[paran] + 1) = 5$
$stratum[neparan] := max(stratum[neparan],\ stratum[broj]) = 5$
$maxstratum = 5$

Algoritam će pri završetku druge iteracije stati i izaći iz petlje zbog uvjeta $maxstratum > brojPredikata$, *jer je* $maxstratum = 5$ *dok je* $brojPredikata = 3$, *te će vratiti obavijest da program nije stratificiran.*

Napomena 2.2.9. *Može se dokazati da ako je program stratificiran, da tada algoritam 2.1 staje i nikada ne radi particiju veću od n, gdje je n broj predikata u programu. Također se*

može dokazati i da algoritam 2.1 korektno određuje je li $Datalog^{f,\ neg}$ program stratificiran ili ne. Očito je da ćemo htjeti pisati stratificirane programe kako bi izbjegli čudna i pomalo besmislena pravila, kao u primjeru 2.2.8. Stratifikacija će biti od vrlo važnog značaja pri evaluaciji upita, što ćemo vidjeti u narednim odjeljcima.

2.3 Semantika Dataloga

U ovom odjeljku bavit ćemo se semantikom Dataloga, odnosno značenjem Datalog programa i načinom evaluacije upita. U početku ćemo se ograničiti na $Datalog^f$ programe, a kasnije ćemo proširiti teoriju i na $Datalog^{neg}$ programe. Podsjetimo se, program je dan skupom činjenica i pravila, odnosno Hornovih klauzula. Upit je dan ciljnom klauzulom. Skup klauzula baze podataka $\mathbf{W}$ i upit $Q \equiv L_1, \ldots, L_n$ nisu ispunjivi ako $\mathbf{W} \models \neg Q$, gdje je $\neg Q \equiv \exists(L_1 \wedge \ldots \wedge L_n)$. Stoga za evaluaciju programa i upita potreban nam je operator logičke posljedice. Odnosno, moramo pronaći neku interpretaciju koja je model. No zbog činjenice da se naš Datalog program sastoji od klauzula, možemo koristiti Herbrandove interpretacije. Herbrandove interpretacije interpretiraju konstante, funkcije i terme na čisto sintaktički način kao njih same. Razne Herbarndove interpretacije razlikuju se samo po tome koji elementi baze su istiniti, a koji su lažni. Stoga Herbrandovu interpretaciju možemo identificirati sa nekim podskupom Herbrandove baze. Konkretno, ideja je sljedeća:

- Sve dane činjenice koje su podskup Herbrandove baze interpretiraju se kao istinite.
- Pravila mogu propagirati istinite vrijednosti ako su sve premise istinite, tj. za dano Datalog pravilo $B :- A_1, \ldots, A_n$, B će se evaluirati kao istina ako je svaki $A_i,\ i \in \{1, \ldots, n\}$ istinit.

To nam omogućuje da definiramo **logičku (semantičku) posljedicu** Datalog programa **P**.

Definicija 2.3.1. *Kažemo da je osnovna činjenica W logička posljedica Datalog programa **P** u oznaci $\mathbf{P} \models W$ ako je svaki Herbrandov model od **P**, također i Herbrandov model za W.*

Primjer 2.3.2. *Pogledajmo sada jedan primjer vezan za interpretacije i modele. Pretpostavimo da imamo dva konstantska simbola $\{Sokrat,\ Hektor\}$ i dva predikatna simbola $\{ratnik(x),\ smrtnik(x)\}$. Neka je dan sljedeći program:*

$$ratnik(Hektor).$$
$$smrtnik(X) :- ratnik(X).$$

Tada možemo naći 16 interpretacija. Zbog pretpostavke zatvorenog svijeta svi x-evi u atomarnim formulama zamijenjeni su sa svim mogućim konstantama. Negativni atomi

mogu također jednostavno biti izostavljeni. No koja od 16 interpretacija će biti model našeg programa? Imamo činjenicu ratnik(Hektor). Stoga svi modeli je moraju sadržavati. Imamo također i pravilo smrtnik(X) : − ratnik(X). stoga svi modeli moraju sadržavati smrtnik(Hektor). Uzevši to u obzir imamo tri moguća modela za naš program.

$$\{ratnik(Hektor),\ smrtnik(Hektor)\} \tag{2.3a}$$

$$\{ratnik(Hektor),\ smrtnik(Hektor),\ smrtnik(S\,okrat)\} \tag{2.3b}$$

$$\{ratnik(Hektor),\ smrtnik(Hektor),\ ratnik(S\,okrat),\ smrtnik(S\,okrat)\} \tag{2.3c}$$

Primijetimo da osnovne atomarne formule ratnik(S okrat) i smrtnik(S okrat) ne utječu negativno na program, no s druge strane naši modeli ne trebaju te atomarne formule. Stoga koji model valja izabrati? Ima smisla da izaberemo najmanji model, odnosno onaj koji dobivamo presjekom svih modela. To je model 2.3a.

Definicija 2.3.3. *Kažemo da je Herbrandov model M* ***najmanji model*** *ako je $M \subseteq M'$, za sve druge Herbrandove modele M'.*

Lema 2.3.4. *Neka su dani Datalog^f program **P** i skup **M** svih Herbrandovih modela programa **P**. Tada je najmanji Herbranov model $\boldsymbol{M_P} := \cap \boldsymbol{M}$.*

Dokaz. Za dokaz leme 2.3.4 vidi [7]. □

Ukoliko promatramo *Datalog^f* program bez negacija tada uvijek postoji najmanji model.

Napomena 2.3.5. *Najmanji Herbrandov model $\boldsymbol{M_P}$ predstavlja namijenjenu semantiku programa **P**, odnosno on evaluira sve dane činjenice i pravila kao istinu, ali ništa više. Samo ono što je eksplicitno navedeno u programu je istina, sve ostalo smatramo lažnim. Stoga za evaluaciju semantike danog Datalog programa **P**, pronalaženje najmanjeg Herbrandovog modela je od ključne važnosti. No često pronalaženje najmanjeg Herbrandovog modela $\boldsymbol{M_P}$ koristeći presjeke modela kao u lemi 2.3.4 neće biti moguće budući da skup svih Herbrandovih modela **M** može biti beskonačan. Za logičke jezike kao što je Prolog deduktivni sustavi su korišteni za pronalaženje istinitih vrijednosti za elemente Herbrandovog svemira (npr. SLD rezolucija), što dovodi do značajnih problema u performansama. U Datalogu problem se rješava korištenjem jednostavnije* ***iteracije fiksne točke****. Iteracija fiksne točke će biti naš deduktivni sustav za Datalog za koji ćemo ubrzo pokazati da je adekvatan i potpun.*

Osnovna ideja iteracije fiksne točke:

- Počinjemo sa praznim podskupom I_0 Herbrandove baze logičkog jezika koji se koriti u programu **P**.
- Transformiramo skup I_n u skup I_{n+1}, tj. $I_{n+1} := T_{\mathbf{P}}(I_n) := T_{\mathbf{P}}^{n+1}(I_0) := T_{\mathbf{P}}(\ldots T_{\mathbf{P}}(T_{\mathbf{P}}(I_0)))$, gdje je $T_{\mathbf{P}}$ neko pravilo transformacije.

Znati ćemo da smo dosegnuli fiksnu točku kada je $I_{n+1} = I_n$.

Definicija 2.3.6. ***Elementarna produkcija*** *je preslikavanje $T_{\boldsymbol{P}} : 2^{B_{\mathcal{L}}} \to 2^{B_{\mathcal{L}}}$ koje preslikava element partitivnog skupa Herbrandove baze $B_{\mathcal{L}}$ u drugi element partitivnog skupa Herbrandove baze $B_{\mathcal{L}}$ na sljedeći način:*

- $T_{\boldsymbol{P}} : I \mapsto \{B \in B_{\mathcal{L}} \ : \ \text{posto ji osnovna instanca } B :- A_1, \ldots, A_n \text{ klauzule programa tako da je } \{A_1, \ldots, A_n\} \subseteq I\}$.

Napomena 2.3.7. *Osnovne instance su kvantifikatorski slobodne podformule formule u preneksnoj formi gdje su sve slobodne varijable zamijenjene nekim termom Herbrandovog svemira. Neka je primjerice dan Herbrandov svemir $U_{\mathcal{L}} = \{v_1,\ v_2,\ v_3, \ldots\}$.*

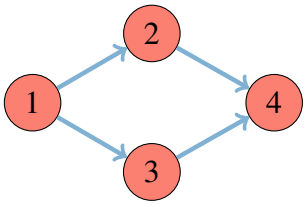
- *Klauzula programa $put(X, Y) :- brid(X, Y)$ znači da $\forall X, Y\ (put(X, Y) \vee \neg brid(X, Y))$.*
- *Osnovna instanca je $put(v_1,\ v_2) \vee \neg brid(v_1,\ v_2)$, sa zamjenama varijabli $X|_{v_1},\ Y|_{v_2}$.*

Primjer 2.3.8. *Neka je dan sljedeći program **P**.*

$$\left.\begin{array}{l} brid(1,2). \\ brid(1,3). \\ brid(2,4). \\ brid(3,4). \end{array}\right\} \text{činjenice}$$

$$put(X, Y) :- brid(X, Y). \tag{2.4a}$$

$$put(X, Y) :- brid(X, Z),\ put(Z, Y). \tag{2.4b}$$



Slika 2.6: Usmjereni graf

Ilustrirajmo iteraciju fiksne točke na danom primjeru.

- $I_0 := \{\}$.
- $I_1 := T_{\boldsymbol{P}}(I_0) = \{brid(1,2),\ brid(1,3),\ brid(2,4),\ brid(3,4)\}$, *elementi od* I_1 *su osnovne činjenice.*
- $I_2 := T_{\boldsymbol{P}}(I_1) = \{brid(1,2),\ brid(1,3),\ brid(2,4),\ brid(3,4),\ put(1,2),\ put(1,3),\ put(2,4),\ put(3,4)\}$, *ovdje smo primijenili pravilo 2.4a na sve elemente skupa* I_1. *Pravilo 2.4b se ovdje ne primjenjuje pošto u* I_1 *još nemamo putova, već imamo samo bridove.*
- $I_3 := T_{\boldsymbol{P}}(I_2) = \{brid(1,2),\ brid(1,3),\ brid(2,4),\ brid(3,4),\ put(1,2),\ put(1,3),\ put(2,4),\ put(3,4),\ put(1,4)\}$, *sada smo primijenili pravilo 2.4b da bi još dobili i put od vrha* 1 *do vrha* 4.
- $I_4 := T_{\boldsymbol{P}}(I_3) = I_3$, *nemamo više novih putova, dakle dosegnuli smo fiksnu točku.*

Napomena 2.3.9. *Za elementarnu produkciju* $T_{\boldsymbol{P}}$ *može se pokazati sljedeće:*

- $I_n \subseteq I_{n+1}$, *tj. unutar svake iteracije skup može samo rasti (evaluacija je monotona).*
- *postoji* $f \geq 0$ *tako da* $\forall m \geq f$ *vrijedi* $I_m = I_{m+1}$, f *se naziva fiksna točka. Vrijedi također* $\forall m < f$, $I_m \subset I_{m+1}$.
- I_f *se može poistovjetiti sa najmanjim Herbrandovim modelom* $\boldsymbol{M_P}$. I_f *nije samo neki skup elemenata Herbrandove baze, već ga možemo shvatiti kao minimalnu interpretaciju koja je konzistentna s programom* $\boldsymbol{P}$ *i koja je samim time model.*

Iteraciju fiksne točke možemo shvatiti kao deduktivni sustav. Program $\boldsymbol{P}$ *pruža aksiome, a jedino pravilo dedukcije je* $T_{\boldsymbol{P}}$. *Na taj način iteracija fiksne točke stvara izvedene osnovne činjenice unutar svake iteracije na čisto sintaktički način.*

Definicija 2.3.10. *Neka je dan program* $\boldsymbol{P}$. *Kažemo da osnovnu činjenicu* W *možemo izvesti iz programa* $\boldsymbol{P}$ *u oznaci* $\boldsymbol{P} \vdash W$ *ako je* $W \in \boldsymbol{P}$ *ili ako se* W *može dobiti primjenom pravila dedukcije* $T_{\boldsymbol{P}}$ *nakon konačnog broja iteracija.*

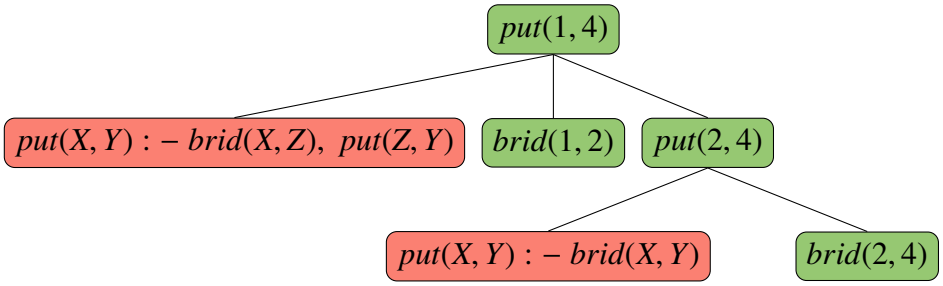
Za svaku izvedenu osnovnu činjenicu W možemo konstruirati **stablo dokaza**.

Definicija 2.3.11. ***Stablo dokaza*** *sastoji se od dva tipa čvorova.*

- ***Činjenični čvorovi*** *koji sadrže izvedene osnovne činjenice ili činjeničnu klauzulu iz programa P.*
- ***Čvorovi koji sadrže pravila*** *iz programa P.*

Stablo dokaza prikazuje minimalni broj pravila i činjenica koja su bila potrebna za izvod činjenice W. Sami W nalazi se u korijenu stabla koji je činjenični čvor. Svaki nivo stabla predstavlja jednu iteraciju. Najniži nivo predstavlja prvu iteraciju. A dubina stabla predstavlja broj potrebnih iteracija da bi izveli W. Ako se ista klauzula koristi više puta, tada se kopira.

Primjer 2.3.12. *Neka nam je dan isti primjer kao i u 2.3.8 Primjer jednog stabla dokaza za put od od vrha 1 do vrha 4.*



Slika 2.7: Stablo dokaza za $put(1,4)$

Alternativno, od vrha 1 *do vrha* 4 *mogli smo također doći i preko vrha* 3.

Teorem 2.3.13. *Neka je dan program* $\boldsymbol{P}$ *te osnovna činjenica W tada vrijedi:*
$\boldsymbol{P} \vdash W \Rightarrow \boldsymbol{P} \models W$.

Dokaz. Dokaz provodimo indukcijom po dubini stabla dokaza za W.

Baza:
Ako W ima stablo dokaza dubine 0, tada je $W \in \mathbf{P}$. Stoga W mora biti u svakom Herbrandovom modelu od $\mathbf{P}$ pa je $\mathbf{P} \models W$.

Korak indukcije:
Pretpostavimo da W ima stablo dokaza dubine $i+1$. Tada postoji pravilo $R \equiv B :- A_1, \ldots, A_n$ i neke osnovne činjenice $F_1, \ldots, F_n$ na nivou i tako da se W može izvesti u jednom koraku primjenom elementarne produkcije $T_{\mathbf{P}}$ na R i $F_1, \ldots, F_n$. Kako se činjenice $F_1, \ldots, F_n$ pojavljuju na nivou i, tada svaki od njih mora imati stablo dokaza *dubine* $\leq i$. Po pretpostavci indukcije imamo za $1 \leq k \leq n$, $\mathbf{P} \models F_k$, stoga za svaki Herbrandov model I imamo da je $F_k \in I$. Kako je $R \in \mathbf{P}$, tada je $W \in I$, pa vrijedi $\mathbf{P} \models W$. □

Teorem 2.3.14. *Neka je dan program* $\boldsymbol{P}$ *te osnovna činjenica W tada vrijedi:*
$\boldsymbol{P} \models W \Rightarrow \boldsymbol{P} \vdash W$.

Dokaz. Promotrimo sljedeći skup $infer(\mathbf{P}) := \{W \ : \ W$ je osnovna činjenica i $\mathbf{P} \vdash W\}$. Po definiciji 2.3.10 svaka činjenica $W \in \mathbf{P}$ je također element skupa $infer(\mathbf{P})$. Neka je $R \in \mathbf{P}$, $R \equiv B :- A_1, \ldots, A_n$ bilo koje pravilo iz $\mathbf{P}$. Pretpostavimo da za neku zamjenu ρ vrijedi $\mathbf{P} \vdash \rho(A_1), \ldots, \mathbf{P} \vdash \rho(A_n)$. Tada je i $\rho(B) \in infer(\mathbf{P})$ i $\mathbf{P} \vdash \rho(B)$. Stoga je $infer(\mathbf{P})$ Herbrandov model za $\mathbf{P}$. Po pretpostavci teorema vrijedi $\mathbf{P} \models W$. Stoga W se nalazi u svakom Herbrandovom modelu od $\mathbf{P}$ a posebno u $infer(\mathbf{P})$, dakle $\mathbf{P} \vdash W$. □

Korolar 2.3.15. *Neka je dan program $\boldsymbol{P}$ te osnovna činjenica W tada vrijedi:*
$\boldsymbol{P} \models W \iff \boldsymbol{P} \vdash W$.

Dokaz. Dokaz slijedi direktno iz teorema 2.3.13 i 2.3.14. □

Korolar 2.3.16. *Ukoliko je program $\boldsymbol{P}$ konačan tada je također i skup $infer(\boldsymbol{P})$ konačan.*

Dokaz. Dokaz korolara 2.3.16 dan je u [7]. □

Napomena 2.3.17. *Bilo koji Datalog model možemo reprezentirati i izračunati u konačnom prostoru i vremenu. Sada možemo pokazati da mora postojati fiksna točka koju se može dosegnuti. Sada ćemo dati naivni algoritam za izvršavanje upita.*

Naivni algoritam za izvršavanje upita:

- Neka je dan Datalog program $\mathbf{P}$ i upit $Q \equiv A_1, \ldots, A_n$
- Započnimo iteraciju fiksne točke na programu $\mathbf{P}$
  - Čim vrijedi da je $\mathbf{P} \vdash \neg Q$, vrati prazan skup kao rezultat.
  - Za svaku izvedenu osnovnu činjenicu W koja je osnovna instanca od Q, stavi W u rezultantni skup.
  - Ako smo dosegnuli fiksnu točku, vrati rezultantni skup.

Napomena 2.3.18. *Treba imati na umu da u svakom koraku iteracije kompletan skup trenutno poznatih osnovnih činjenica se također ponovo izračunava. Očito je takav način izvršavanja upita suboptimalan, stoga i naziv "naivni algoritam". U sljedećim odjeljcima vidjet ćemo bolji algoritam za izvršavanje upita.*

Primjer 2.3.19. *Neka je dan isti primjer kao u 2.3.8. Pretpostavimo da smo postavili sljedeći upit: $?p(1, X)$, odnosno u koje sve čvorove možemo doći iz čvora 1. Tada u drugoj iteraciji u rezultantni skup smo dodali $put(1, 2)$ i $put(1, 3)$, a u trećoj iteraciji u rezultantni skup smo dodali $put(1, 4)$. Kada smo dosegnuli fiksnu točku, odnosno kada je $I_4 := T_{\boldsymbol{P}}(I_3) = I_3 = \{brid(1, 2),\ brid(1, 3),\ brid(2, 4), brid(3, 4),\ put(1, 2),\ put(1, 3),\ put(2, 4),\ put(3, 4),\ put(1, 4)\}$, rezultantni skup sadrži sve vrhove koje možemo dosegnuti iz vrha 1, tj. $\{put(1, 2),\ put(1, 3),\ put(1, 4)\}$*

Napomena 2.3.20. *Treba imati na umu da smo do sada promatrali Datalog^f programe, bez negacije. No treba li nam uopće negacija? Može se pokazati da je Datalog^f Turing potpun. Što znači da koristeći Datalog^f možemo izračunati sve što možemo izračunati drugim programskim jezicima (C, Java, Pascal itd.). Dakle negacija nam nije nužna, no mogućnost izražavanja negacija bilo bi lijepo svojstvo jer bi mogli na intuitivniji način modelirati stvarni svijet u Datalog programima. No kod negacija se javljaju razni problemi. Ako nam je primjerice dana konstanta Hektor i program $filozof(X) :- \neg ratnik(X)$. Tada možemo dobiti dva modela $\{ratnik(Hektor)\}$ i $\{filozof(Hektor)\}$. Oba modela zadovoljavaju program, no presjek im je prazan. Stoga ne postoji najmanji Harbranov model, te iteracija fiksne točke ne radi dobro. Obično Datalog^{neg} programi nemaju najmanji Herbrandov model, ali zato mogu imati više* ***minimalnih modela****.*

Definicija 2.3.21. *Herbrandov model M je* ***minimalan Herbrandov model*** *ako ne postoji ni jedan drugi Herbrandov model M', takav da vrijedi $M' \subset M$. Namijenjena semantika se naziva* ***minimalna Herbrandova semantika****.*

Sljedeće pitanje koje se nameće je, ukoliko imamo više minimalnih modela, koji od njih je prikladan? Dakle, treba nam neki deterministički kriterij odluke za izbor prikladnog modela. Jedan dobar način bilo bi iskoristiti rezultate stratifikacije. Stratifikacija pokušava odrediti negativne ovisnosti predikata u programu, tako da predikate uredi hijerarhijski. Samo programi koji se mogu stratificirati mogu se i izvršiti. Ukoliko pogledamo pravilo $filozof(X) :- \neg ratnik(X)$, da bi ga evaluirali moramo testirati sve osnovne terme Herbrandovog svemira. U nekim slučajevima (npr *Datalog^f*) Herbrandov svemir može biti beskonačan. Stoga za pravilo $filozof(X) :- \neg ratnik(X)$ gdje imamo beskonačne ili izuzetno velike modele kažemo da je **nesigurno**. Nadalje, izbor modela za $filozof(X) :- \neg ratnik(X)$ je dvosmislen. Nemamo informaciju $ratnik(Hektor)$, stoga ne možemo ništa reći o tome da li je $filozof(Hektor)$ istina. Ta dva problema (sigurnost i dvosmislenost) možemo riješiti uvođenjem sljedećih ograničenja.
Ograničenja: ako se varijabla pojavljuje u negativnom literalu, tada se također mora pojaviti i u pozitivnim literalu u tijelu pravila. Ovo ograničenje se naziva **sigurna negacija**. Na taj način možemo prvo evaluirati pozitivne a zatim negativne literale. Da bismo to usvojili organizirano evaluaciju stratu po stratu.

Definicija 2.3.22. *Definiramo* ***relaciju prioriteta*** $<_P$ *na elementima Herbrandove baze $B_\perp$ na sljedeći način:*

> *$P(t_1, \ldots, t_n) <_P Q(s_1, \ldots, s_m)$, ako postoji negativni brid od P do Q u grafu predikatnih ovisnosti programa* ***P****.*

Definicija 2.3.23. *Neka su M_1 i M_2 modeli za program* ***P****. Tada kažemo da M_1* ***ima prednost*** *nad M_2 ($M_1 \leq M_2$) ako vrijedi sljedeće:*

- $M_1 = M_2$ *ili* $M_1 \neq M_2$ *i za sve* $A \in M_1 \setminus M_2$ *postoji* $B \in M_2 \setminus M_1$ *tako da vrijedi* $A <_{\boldsymbol{P}} B$.

Za model M kažemo da je ***savršen model*** *ako vrijedi* $M \leq M'$ *za sve Herbrandove modele* M' *programa* $\boldsymbol{P}$.

Teorem 2.3.24. *Za svaki Datalog*^{neg} *program postoji savršen model koji je ujedno i minimalni model.*

Dokaz. Za dokaz teorema 2.3.24 vidi [7]. □

Napomena 2.3.25. *Savršeni model koji je ujedno i minimalni je namijenjena semantika za Datalog*^{neg} *program. No još uvijek moramo popraviti iteraciju fiksne točke. Ideja je da modificiramo elementarnu produkciju* $T_{\boldsymbol{P}}$ *tako da radi na stratama programa* $\boldsymbol{P}$.

Definicija 2.3.26. ***Elementarna produkcija*** $T_{\boldsymbol{P}}^{J}$ *u ovisnosti o J je preslikavanje* $T_{\boldsymbol{P}}^{J} : 2^{B_{\mathcal{L}}} \to 2^{B_{\mathcal{L}}}$ *koje preslikava element partitivnog skupa Herbrandove baze* $B_{\mathcal{L}}$ *u drugi element partitivnog skupa Herbrandove baze* $B_{\mathcal{L}}$ *na sljedeći način:*

- $T_{\boldsymbol{P}}^{J} : I \mapsto \{B \in B_{\mathcal{L}} \ : \ postoji\ osnovna\ instanca\ B :- A_1, \ldots, A_n, \neg C_1, \ldots, \neg C_m$ *klauzule programa tako da je* $\{A_1, \ldots, A_n\} \subseteq I$ *i za svaki* $1 \leq i \leq m,\ C_i \notin J\}$.

Na osnovu novog pravila elementarne produkcije možemo definirati tzv. ***iteriranu iteraciju fiksne točke*** *koristeći stratifikaciju:*

Neka je $\boldsymbol{P}$ *stratificirani program, tj.* $\boldsymbol{P} = P_0 \cup \ldots \cup P_n$

$I_0 \coloneqq T^{\infty}_{P_0}(\{\})$, *normalna (pozitivna) iteracija fiksne točke samo sa* P_0

$I_1 \coloneqq T^{\infty}_{P_1 \cup P_0}(T^{I_0}_{P_1 \cup I_0})$

$\vdots$

$I_n \coloneqq T^{\infty}_{P_n \cup \ldots \cup P_0}(T^{I_{n-1}}_{P_n \cup I_{n-1}})$

Čim smo dosegnuli fiksnu točku n, tada I_n *nazivamo* ***iterirana fiksna točka*** *za program* $\boldsymbol{P}$.

Teorem 2.3.27. *Iterirana fiksna točka* I_n *je* ***minimalni i savršeni Herbrandov model*** *za program* $\boldsymbol{P}$.

Dokaz. Za dokaz teorema 2.3.27 vidi [1]. □

Primjer 2.3.28. *Ilustrirajmo iteriranu iteraciju fiksne točke na stratificiranom programu* $\boldsymbol{P} = P_1 \cup P_2$, *gdje su:*

$$P_1 = \{brid(1,2),\ brid(1,4),\ brid(2,4),\ brid(3,4),\ cestarina(1,2),\ put(X,Y) :- brid(X,Y),\ put(X,Y) :- brid(X,Z),\ put(Z,Y)\}$$

$$P_2 = \{dobarPut(X,Y) :- put(X,Y),\ \neg cestarina(X,Y),\ dobarPut(X,Y) :- dobarPut(X,Z),\ dobarPut(Z,Y)\}$$

Sada primjenjujemo $T_{neg,\ P}$ *na dio najniže strate programa.*

$$I_1 := T_{\boldsymbol{P}}^{\infty}(T_{neg,\ P_1}) = \{brid(1,2),\ brid(1,4),\ brid(2,4),\ brid(3,4),\ cestarina(1,2),\ put(1,2),\ put(1,4),\ put(2,4),\ put(3,4)\}$$

$$I_2 := T_{\boldsymbol{P}}^{\infty}(T_{neg,\ P_2 \cup I_1}^{I_1}) = \{brid(1,2),\ brid(1,4),\ brid(2,4),\ brid(3,4),\ cestarina(1,2),\ put(1,2),\ put(1,4),\ put(2,4),\ put(3,4),\ dobarPut(2,4),\ dobarPut(3,4),\ dobarPut(1,4)\}$$

2.4 Datalog i relacijska algebra

U ovom odjeljku preslikati ćemo $Datalog^{neg}$ u relacijsku algebru. To će nam omogućiti implementaciju *Datalog* koncepata unutar relacijske algebre. Ideja je da uzmemo $Datalog^{neg}$ program, prevedemo ga u relacijsku algebru, evaluiramo izraz relacijske algebre (sa nekim SQL kompajlerom) i vratimo rezultat. To nam omogućuje da iskoristimo sve prednosti već utvrđenih svojstava relacijske algebre kao što su primjerice optimizacija upita, transakcije (ACID svojstva), raspodjela opterećenja (eng. load balancing), distribucija baze podataka itd. Također za pouzdano skladištenje velikih količina podataka (npr. činjenica u ekstenzijskoj bazi podataka) relacijska algebra i SQL su tehnologije koje koriste većina današnjih aplikacija.

Kada koristimo relacijski model i relacijsku algebru pretpostavljamo sljedeće:

- Podaci odnosno činjenice su pohranjeni u više relacija.
- Relacija R nad nekim skupovima $D_1, \ldots, D_n$, je podskup njihovog Kartezijevog produkta, tj. $R \subseteq D_1 \times \ldots \times D_n$. Skupovi $D_1, \ldots, D_n$ su konačni i nazivaju se **domene**.
- Nad relacijama definiramo sljedeće algebarske operacije:

  Osnovne operacije relacijske algebre

  $\times$ – Kartezijev produkt

  σ – selekcija

  π – projekcija

$\cup$ – skupovna unija

$\setminus$ – skupovna razlika

Složene operacije relacijske algebre

$\bowtie$ – join (spajanje relacija), $R \bowtie S \equiv \sigma_{\Theta}(R \times S)$

$\ltimes$, $\rtimes$ – lijevi i desni semi join ($R \ltimes S \equiv \pi_{atr(R)}(R \bowtie S)$)

Napomena 2.4.1. *U ovom odjeljku koristiti ćemo varijante normalne relacijske algebre. Umjesto imena, na atribute se referenciramo njihovim brojevima (npr.* #1 *ili* #9*). Kada se referenciramo na relaciju u binarnim operacijama (npr. join) tada umjesto oznaka relacija možemo pisati* [*lijeva*] *i* [*desna*] *npr.* $(R \times S) \bowtie_{[lijeva].\#3=[desna].\#1} W$.

Primjer 2.4.2. *Pogledajmo primjer baze o studentima i kolegijima koje su upisali. Imamo tri relacije (Student, Upisao i Kolegij).*

id	*Ime i prezime*
1	Ivan Horvat
2	Marko Matić

Tablica 2.1: Relacija Student

id	*Kolegij*
1	Složenost algoritama
2	Računarski praktikum 3
3	Izračunljivost
4	Baze podataka

Tablica 2.2: Relacija Kolegij

id_studenta	*id_kolegija*
1	1
1	2
2	4
2	3
1	3

Tablica 2.3: Relacija Upisao

U terminima relacijske algebre ako bismo htjeli ispisati ime i prezime studenta sa vrijednosti id $= 1$.

- $\pi_{\#2}\sigma_{\#1=1}(Student)$

Ukoliko bismo željeli ispisati sve predmete koje sluša student sa vrijednosti id $= 2$.

- $\pi_{\#5}((\sigma_{\#1=2})(Student) \bowtie_{Student.\#1=Upisao.\#1} Upisao \bowtie_{[lijeva].\#2=[desna].\#1} Kolegij)$

Kao što je već rečeno u ovom odjeljku napravit ćemo preslikavanje $Datalog^{neg}$ programa u relacijsku algebru. Sljedeći jednostavni primjer ilustrira nam kako bi smo preslikali relacijsku algebru u $Datalog^{neg}$.

Primjer 2.4.3. *Preslikavanje relacijske algebre u* $Datalog^{neg}$.

- $\sigma_{\#2=5}(R) \longmapsto R(X, 5)$
- $\pi_{\#1}(R) \longmapsto R'(X) :- R(X, Y)$
- $R \times S \longmapsto RS(W, X, Y, Z) :- R(W, X),\ S(Y, Z)$
- $R \bowtie_{[lijeva].\#1=[desna].\#2} S \longmapsto RS(W, X, Y, Z) :- R(W, X),\ S(Y, Z),\ W = Z$
- $R \ltimes_{[lijeva].\#1=[desna].\#2} S \longmapsto RS(W, X) :- R(W, X),\ S(Y, Z),\ W = Z$
- $R \cup S \longmapsto \begin{matrix} R'(X, Y) :- R(X, Y) \\ R'(X, Y) :- S(X, Y) \end{matrix}$
- $R \setminus S \longmapsto R'(X, Y) :- R(X, Y),\ \neg S(X, Y)$

U nastavku ćemo napraviti implementaciju jednostavne iteracije fiksne točke u relacijskoj algebri. Razmatramo samo **sigurne Datalog^{neg} programe**. Stoga za sigurni $Datalog^{neg}$ program **P** i relacijsku bazu podataka cilj je pohraniti ekstenzijsku bazu podataka (EDB) u tablice, te kodirati intenzijsku bazu podataka (IDB) u prilagođenu relacijsku algebru, te u toj relacijskoj algebri realizirati elementarnu produkciju $T_{\mathbf{P}}$.

Neka je dan sigurni $Datalog^{neg}$ program **P** i relacijska baza podataka, tada:

- Svaki predikatni simbol $r_1, \ldots, r_m$ ekstenzijske baze podataka dodijeljen je relaciji $R_1, \ldots, R_m$, tj. predikatni simboli ekstenzijske baze podataka predstavljaju činjenice, stoga svaka činjenica ima svoju relaciju.
- Svaki predikatni simbol $q_1, \ldots, q_m$ intenzijske baze podataka dodijeljen je relaciji $Q_1, \ldots, Q_m$, ti predikati su definirani pravilima.
- Predikatni simboli $<$, $>$, $\leq$, $\geq$, $=$, $\neq$ dodijeljeni su redom hipotetskim relacijama $\mathbf{H} = \{LT,\ GT,\ LTE,\ GTE,\ EQ,\ NEQ\}$. Te relacije su beskonačno velike stoga ih nećemo pohranjivati u relacijskoj bazi podataka. Kasnije ćemo vidjeti da ih možemo i maknuti.

- Radi jednostavnosti uvodimo restrikciju da svaki predikat može biti definiran ili u ekstenzijskoj bazi podataka ili u intenzijskoj bazi podataka, odnosno predikati koji se koriste za definiciju činjenica ne smiju se pojaviti u glavi pravila.

Da bi preveli $Datalog^{neg}$ u relacijsku algebru prvo moramo preurediti pravila na sljedeći način:

- Sva pravila intenzijske baze podataka transformiramo tako da glava pravila sadrži samo varijable. To možemo postići tako da svaku konstantu u glavi pravila zamijenimo novom varijablom i dodamo literal koji veže tu novu varijablu sa svojom starom vrijednosti npr. $q(X, a) :- L_1, \ldots, L_n$ transformiramo u
  $q(X, Y) :- L_1, \ldots, L_n,\ Y = a$.

- Promijenimo redoslijed varijabli tako da je njihova sigurnost osigurana u prethodnim literalima tijela. Literal je nesiguran ako je potencijalno beskonačan.

  - Pravilo $R :- X = Y,\ p(X),\ q(Y)$ nije u dobrom redoslijedu jer sigurnost literala $X = Y$ nije osigurana prethodnim literalima. Imamo beskonačno mnogo mogućnosti za X da bude jednak Y.
  - Pravilo $R :- p(X),\ q(Y),\ X = Y$ je u dobrom redoslijedu jer $p(X)$ i $q(Y)$ ograničavaju moguće vrijednosti od X i Y.

- Također preuredimo pravila tako da pozitivni literali dolaze prije negativnih.

Svako pravilo $R :- L_1, \ldots, L_n$ je sada transformiramo u u relacijsku algebru na sljedeći način:

- Za svaki literal $L_1, \ldots, L_n$ odgovarajuća atomarna komponenta $A_i \equiv p_i(t_1, \ldots, t_m)$ je transformirana u relacijski izraz $\mathbf{E}_i$. $\mathbf{E}_i \equiv \sigma_\Theta(P_i)$, gdje je P_i relacija koja odgovara predikatu p_i. Kriterij selekcije Θ je konjunkcija uvjeta definirana na sljedeći način:

  - Za svaki t_i dodan je uvjet:

    $\#j = t_j$, ako je t_j konstantski simbol
    $\#j = \#k$, ako su t_j i t_k iste varijable

- Nakon transformacije literala sastavljamo tijelo izraza $\mathbf{F}$ sa lijeva na desno.

  - Inicijaliziramo privremeni izraz $\mathbf{F}_1 \coloneqq \mathbf{E}_1$

- Ovisno o varijablama u literalima izraze $\mathbf{F}_2$ do $\mathbf{F}_k$ generiramo na različiti način:

  - $\mathbf{F}_i \coloneqq \mathbf{F}_{i-1} \times \mathbf{E}_i$, ako L_i ne sadrži varijable iz prethodnih literala u tijelu, tj. $vars(L_i) \cap vars(\{L_1, \ldots, L_{i-1}\}) = \emptyset$. Ili ukratko, konjunkcija nepovezanih literala rezultira računanjem Kartezijevog produkta.

  - $\mathbf{F}_i := \mathbf{F}_{i-1} \bowtie_\Theta \mathbf{E}_i$, ako je L_i pozitivan literal i dijeli varijable sa prethodnim literalima u tijelu. Ili ukratko, konjunkcija povezanih pozitivnih literala rezultira računanjem spajanja relacija (join) koristeći povezane varijable kao uvjet spajanja.
  - $\mathbf{F}_i := \mathbf{F}_{i-1} \setminus (\mathbf{F}_{i-1} \ltimes_\Theta \mathbf{E}_i)$, ako je L_i negativni literal i dijeli varijable sa prethodnim literalima u tijelu. Ili ukratko, konjunkcija povezanih negativnih literala rezultira računanjem skupovne razlike uklanjanjem onih redaka tablice koji su povezani sa negativnim literalom.

- Ako u izrazu imamo beskonačnu hipotetsku relaciju $\mathbf{H} = \{LT,\ GT,\ LTE,\ GTE,\ EQ,\ NEQ\}$, tada:
  - Svako spajanje relacija (join) $\mathbf{E} \bowtie_\Theta H_i$ ili Kartezjev produkt $\mathbf{E} \times H_i$ za svaki "normalni" izraz $\mathbf{E}$ i $H_i \in \mathbf{H}$ zamjenjuju se prikladnim izrazom oblika $\pi(\sigma(\mathbf{E}))$.
- Kompletno pravilo $\mathbf{C} \equiv R :- L_1, \ldots, L_n$ transformiramo u izraz $eval(\mathbf{C}) := \pi_{glava(R)}(\mathbf{F})$

Primjer 2.4.4. *Ovim primjerom ilustriramo transformacije nekih pravila:*

1) $R(X,Y,Z) :- q(X,2),\ r(Y),\ Z = 3$*, u ovom pravilu imamo konjunkciju nepovezanih literala, što rezultira računanjem Kartezijevog produkta.*

- $\boldsymbol{F}_1 := \boldsymbol{E}_1 = \sigma_{\#2=2}Q,\ \boldsymbol{E}_2 = R,\ \boldsymbol{E}_3 = \sigma_{\#1=3}EQ$
- $\boldsymbol{F}_2 := (\sigma_{\#2=2}Q) \times R$
- $\boldsymbol{F}_3 := (\sigma_{\#2=2}Q) \times R \times \sigma_{\#1=3}EQ$
- $\boldsymbol{F} := \pi_{\#1,\#2,3}((\sigma_{\#2=2}Q) \times R))$

2) $R(X,Y) :- q(3,X),\ r(Y),\ X < Y$*, u ovom pravilu imamo konjunkciju povezanih pozitivnih literala, što rezultira računanjem spajanjem relacija (join) koristeći povezane varijable kao uvjet spajanja.*

- $\boldsymbol{F}_1 := \boldsymbol{E}_1 = \sigma_{\#1=3}Q,\ \boldsymbol{E}_2 = R,\ \boldsymbol{E}_3 = LT$
- $\boldsymbol{F}_2 := (\sigma_{\#1=3}Q) \times R$
- $\boldsymbol{F}_3 := ((\sigma_{\#1=3}Q) \times R) \bowtie_{[lijeva].\#2=[desna].\#1 \wedge [lijeva].\#3=[desna].\#2} LT$
- $\boldsymbol{F} := \sigma_{\#2<\#3}((\sigma_{\#1=3}Q) \times R)$*, algebarskom optimizacijom kasnije ćemo imati:*

  $\boldsymbol{F} := (\sigma_{\#1=3}Q) \bowtie_{\#2<\#3} R$

3) $R(X) :- q(X),\ \neg r(X)$*,u ovom pravilu imamo konjunkciju povezanih negativnih literala, što rezultira računanjem skupovne razlike uklanjanjem onih redaka tablice koji su povezani sa negativnim literalom.*

– $\boldsymbol{F}_1 := \boldsymbol{E}_1 = Q,\ \boldsymbol{E}_2 = R$

– $\boldsymbol{F} := Q \setminus (Q \ltimes_{Q.\#1=R.\#1} R)$

Da bi evaluirali jedan korak iteracije za dani intenzijski predikat q_i, moramo ujediniti sva povezna pravila, tj. $eval(q_i) := \bigcup_{C \in def(q_i)}(C)$. Sada elementarna produkcija $T_{\mathbf{P}}$ korespondira sa evaluacijom $eval(q_i)$. Upiti $Q \equiv p(t_1, \ldots, t_n)$ se transformiraju u relacijsku algebru na sličan način.

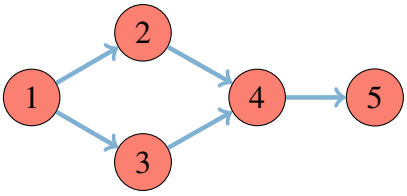
Sljedećim algoritmom izvodimo **iteraciju fiksne točke u relacijskoj algebri**.

1. Za svaki intenzijski predikat q_i stvori jednu tablicu.
2. Izvrši elementarnu produkciju $T_{\mathbf{P}}$ (tj. pokreni $eval(q_i)$ za svaki intenzijski predikat) i spremi privremeno rezultate.
   a) Ako su tablice rezultata iste veličine kao i predikatne tablice, dostigli smo fiksnu točku i možemo prijeći na korak 3.
   b) Zamijeni sadržaj tablice intenzijskih predikata sa sadržajem njima odgovarajućih privremenih tablica.
   c) Idi na korak 2.
3. Pokreni upit nad tablicama, kako bi dobio krajnji rezultat.

Primjer 2.4.5. *Ilustrirajmo ovim primjerom iteraciju fiksne točke u relacijskoj algebri. Neka nam je dan ekstenzijski i intenzijski dio baze podataka.*

$$\left.\begin{array}{l} brid(1,2). \\ brid(1,3). \\ brid(2,4). \\ brid(3,4). \\ brid(4,5). \end{array}\right\} \text{činjenice}$$

$$put(X,Y) :- brid(X,Y). \tag{2.5a}$$
$$put(X,Y) :- brid(X,Z),\ put(Z,Y). \tag{2.5b}$$



Slika 2.8: Usmjereni graf

Neka nam je krajnji cilj dobiti sve čvorove u koje možemo doći iz čvora 2*, tj. postavljamo upit* $?put(2, X)$.

Prije nego što započnemo s iteracijom fiksne točke, moramo prevesti Datalog pravila i upit u relacijsku algebru.

- $put(X, Y) :- brid(X, Y).$

  $\longmapsto \boldsymbol{F} := \pi_{\#1,\#2}(\sigma_{istina}(Brid)) = Brid$

- $put(X, Y) :- brid(X, Z),\ put(Z, Y).$

  $\longmapsto \boldsymbol{F} := \pi_{\#1,\#2}(\sigma_{istina}(Brid) \bowtie_{[lijeva].\#2=[desna].\#1} \sigma_{istina}(Put))$
  $= Brid \bowtie_{[lijeva].\#2=[desna].\#1} Put$

- $?put(2, X) = put(Y, X),\ Y = 2.$

  $\longmapsto \boldsymbol{F} := \sigma_{\#1=2}(Put)$

Na kraju ujedinjujemo sva povezana pravila:

- $eval(put) := Brid \cup Brid \bowtie_{[lijeva].\#2=[desna].\#1} Put$

Sve činjenice pohranimo u ekstenzijsku tablicu Brid, te staramo intenzijsku tablicu Put koja je u početku prazna.

#1	#2
1	2
1	3
2	4
3	4
4	5

Tablica 2.4: Brid

#1	#2

Tablica 2.5: Put

Sada izvršavamo elementarnu produkciju $eval(put) := Brid \cup Brid \bowtie_{[lijeva].\#2=[desna].\#1} Put$ *na trenutnim tablicama 2.4 i 2.5 te pohranimo rezultate u privremenu tablicu 2.6.*

#1	#2
1	2
1	3
2	4
3	4
4	5

Tablica 2.6: $Temp_{Put}$

Zamijenimo sav sadržaj tablice Put (koja je trenutno prazna) sa sadržajem privremene tablice $Temp_{Put}$.

#1	#2
1	2
1	3
2	4
3	4
4	5

Tablica 2.7: Brid

#1	#2
1	2
1	3
2	4
3	4
4	5

Tablica 2.8: Put

Cijeli postupak sada ponavljamo, tj. izvršavamo elementarnu produkciju $eval(put) := Brid \cup Brid \bowtie_{[lijeva].\#2=[desna].\#1} Put$ *na trenutnim tablicama 2.7 i 2.8 te pohranimo rezultate u privremenu tablicu 2.9.*

#1	#2
1	2
1	3
2	4
3	4
4	5
1	4
2	5
3	5

Tablica 2.9: $Temp_{Put}$

Zamijenimo sav sadržaj tablice Put sa sadržajem privremene tablice $Temp_{Put}$.

#1	#2
1	2
1	3
2	4
3	4
4	5

Tablica 2.10: Brid

#1	#2
1	2
1	3
2	4
3	4
4	5
1	4
2	5
3	5

Tablica 2.11: Put

Cijeli postupak sada ponavljamo, tj. izvršavamo elementarnu produkciju $eval(put) := Brid \cup Brid \bowtie_{[lijeva].\#2=[desna].\#1}$ *Put na trenutnim tablicama 2.10 i 2.11 te pohranimo rezultate u privremenu tablicu 2.12.*

#1	#2
1	2
1	3
2	4
3	4
4	5
1	4
2	5
3	5
1	5

Tablica 2.12: $Temp_{Put}$

Zamijenimo sav sadržaj tablice Put sa sadržajem privremene tablice $Temp_{Put}$.

#1	#2
1	2
1	3
2	4
3	4
4	5

Tablica 2.13: Brid

#1	#2
1	2
1	3
2	4
3	4
4	5
1	4
2	5
3	5
1	5

Tablica 2.14: Put

Cijeli postupak sada ponavljamo, tj. izvršavamo elementarnu produkciju $eval(put) := Brid \cup Brid \bowtie_{[lijeva].\#2=[desna].\#1} Put$ *na trenutnim tablicama 2.13 i 2.14 te pohranimo rezultate u privremenu tablicu 2.15.*

#1	#2
1	2
1	3
2	4
3	4
4	5
1	4
2	5
3	5
1	5

Tablica 2.15: $Temp_{Put}$

Tablica Put i privremena tablica $Temp_{Put}$ su jednake što znači da smo postigli fiksnu točku. Iteracija fiksne točke je završila. Sada nam jedino preostaje još izvršiti upit $\sigma_{\#1=2}(Put)$ nad tablicom 2.14. Dobivamo tablicu rezultata upita 2.16:

#1	#2
2	4
2	5

Tablica 2.16: *Tablica rezultata upita*

Dakle iz čvora 2 *možemo doći u čvorove* 4 *i* 5.

2.5 Metode evaluacije i optimizacije

U ovom odjeljku dan je pregled i opis evaluacijskih i optimizacijskih metoda kojima možemo evaluirati i optimizirati Datalog programe i upite. Nakon toga ćemo u narednim odjeljcima detaljnije obraditi neke od metoda evaluacije a potom i optimizacije koje su najvažnije i koje se najčešće koriste za Datalog.

Evaluacijske i optimizacijske metode možemo klasificirati po nekoliko kriterija:

- **Tehnike (strategije) pretraživanja** za evaluaciju Datalog programa – postoje dvije osnovne strategije.

  - **Bottom-up**: započinje sa danim činjenicama iz ekstenzijske baze podataka, potom generira sve moguće činjenice i odbacuje one koje ne odgovaraju upitu. Ovakva strategija odražava ideju ulančavanja unaprijed (eng. forward-chaining) gdje se počevši od činjenica krećemo unaprijed da bi deducirali sve što možemo. Primjer ovakve strategije je iteracija fiksne točke. Bottom-up strategija je dobra strategija za ograničene i manje scenarije. Kasnije ćemo dati bolji algoritam iteracije fiksne točke i bottom-up pristup kojim ćemo izbjeći generiranje nepotrebnih činjenica.
  - **Top-down**: započinjemo od upita, te generiramo stabla dokaza krećući se prema dolje sve dok ne dođemo do činjenica iz ekstenzijske baze podataka. Ovakva strategija odražava ideju ulančavanja unazad (eng. backward-chaining) gdje počevši od upita generiramo samo znanje i činjenice koje nam trebaju krećući se unazad. Top-down strategija je dobra strategija za veće i kompleksnije scenarije. Većina logičkih okruženja (npr. SLD rezolucija) koriste top-down strategiju. Također Prolog je jedan primjer logičkog jezika koji koristi SLD rezoluciju, odnosno evaluira svoje programe koristeći top-down strategiju.

- **Formalizmi za optimizaciju upita** – postoje dva ne ekskluzivna formalizma za optimizaciju upita.
  - **Logički formalizam**: Datalog program tretiramo kao skup logičkih pravila, te radimo optimizaciju nad tim skupom logičkih pravila.
  - **Algebarski formalizam**: pravila Datalog programa možemo transformirati u algebarske izraze, tako da intenzijski dio baze podataka odgovara sustavu algebarskih jednadžbi (ovo smo napravili u odjeljku 2.4 gdje smo Datalog transformirali u relacijsku algebru). Zatim možemo ponovo raditi nekakvu vrstu optimizacije u toj algebri.

- **Ciljevi optimizacije** – optimizacija se može odnositi na različite ciljeve.
  - **Prerada programa**: za danu specifičnu metodu evaluacije, Datalog program $\mathbf{P}$ možemo preraditi u semantički ekvivalentan program $\mathbf{P}'$. To ima za posljedicu da se program $\mathbf{P}'$ može izvršiti puno brže koristeći istu evaluacijsku metodu (algoritam).
  - **Optimizacija evaluacije**: pokušava poboljšati sam proces evaluacije, odnosno ne mijenjamo program već samo algoritam evaluacije. Može se kombinirati sa preradom programa kako bi se postigao još bolji rezultat.

- **Redoslijed obilaska** – optimizacija se može fokusirati na različite redoslijede obilaska.

  - **Pretraživanje u dubinu** (eng. depth-first): redoslijed literala u tijelu pravila može utjecati na izvođenje samog programa, u nekim općenitijim slučajevima (npr. Prolog), može čak utjecati na odlučivost. S druge strane moguće je ranije dobiti prve rezultate.
  - **Pretraživanje u širinu** (eng. breadth-first): cijela desna strana pravila evaluira se istovremeno. Zbog restrikcija u Datalogu to postaje skupovno orijentirana operacija pa je stoga vrlo pogodna za baze podataka.

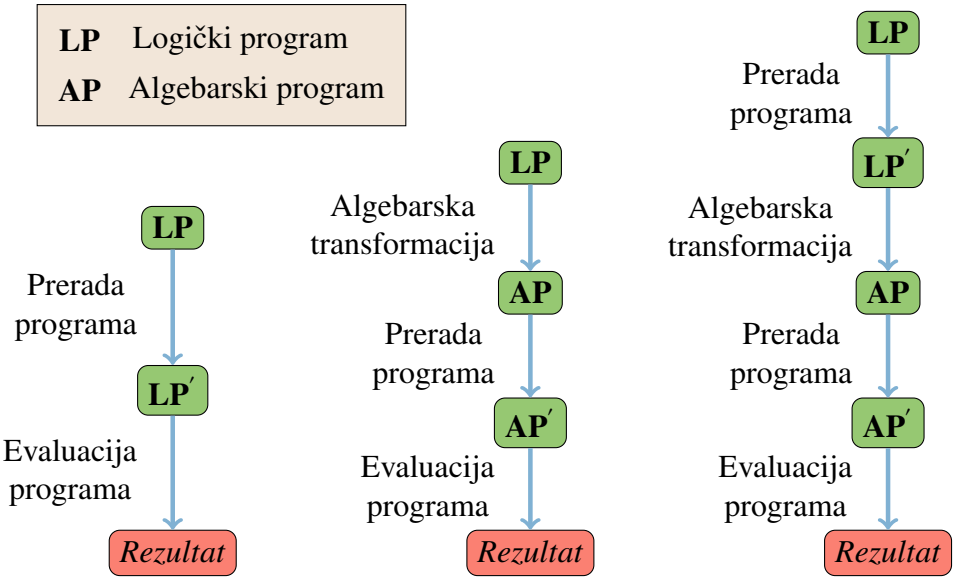
- **Pristup optimizaciji** – prilikom optimizacije moguća su dva pristupa.
  - **Sintaktički pristup**: odnosi se samo na sintaksu pravila. Primjerice, ograničimo varijable ovisno o strukturi cilja ili koristimo neku specijalnu evaluaciju ukoliko su sva pravila linearna.
  - **Semantiki pristup**: koristi vanjsko znanje tijekom evaluacije. Primjerice, ograničenja koja osiguravaju integritet (primarni ključ, strani ključ itd.)

- **Struktura** –odnosi se na orijentaciju koja nas više zanima, možemo se orijentirati prema pravilima ili prema ciljevima.

Kriterij	**Alternative**	
Tehnike pretraživanja	bottom-up	top-down
Formalizmi	logički	algebarski
Ciljevi optimizacije	prerada programa	optimizacija evaluacije
Redoslijed obilaska	pretraživanje u dubinu	pretraživanje u širinu
Pristup optimizaciji	sintaktički	semantički
Struktura	struktura pravila	struktura ciljeva

Tablica 2.17: Klasifikacija evaluacije i optimizacije

Optimizacijske tehnike mogu se kombinirati. Stoga su moguća različita izvršavanja prerada programa i evaluacijskih tehnika, kao na slici 2.9. Započnemo sa nekim logičkim programom **LP**, pa različitim kombinacijama prerade programa, transformacijama u algebarski program **AP** i nekom evaluacijskom tehnikom dobivamo program koji brže dolazi do rezultata.



Slika 2.9: Razne kombinacije optimizacijskih tehnika

Naravno, nisu baš sve kombinacije izvedive i smislene. Mi ćemo se narednom odjeljku posvetiti top-down evaluaciji sa naivnim evaluacijskim algoritmom i stablima traženja. A kasnije ćemo vidjeti i bottom-up evaluaciju sa semi-naivnim algoritmom (tzv. delta-iteracijama). Od optimizacijskih tehnika obraditi ćemo logičke i algebarske formalizme te na svakom od njih pogledati neku optimizaciju programa (magični skupovi i guranje-selekcije).

2.6 Top-down evaluacija

Odjeljak ćemo započeti sa jednostavnim top-down pristupom, osnovna ideja je sljedeća:

- Krenemo od upita $Q \equiv p(t_1, \ldots, t_n)$
- Generiramo iterativno sva stabla dokaza koja završavaju sa osnovnim instancama od Q i počinju sa osnovnim činjenicama. Iteriramo po dubini stabla. Kao pomoćnu strukturu podataka kreiramo sva moguća stabla traženja na trenutnoj dubini. Potom transformiramo stablo traženja u sva moguća stabla dokaza. Stanemo kada kada više ne možemo konstruirati nova stabla traženja odnosno stabla dokaza.

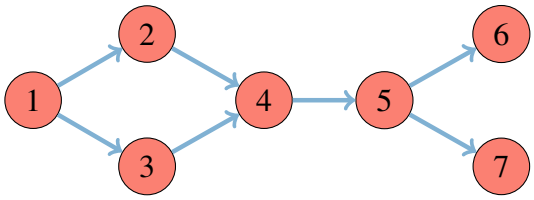
Stablo traženja je generičko stablo dokaza koje je u određenoj mjeri parametrizirano. Stabla dokaza mogu se generirati iz stabala traženja. Čvorove stabla traženja koji su listovi nazivamo **podciljni čvorovi**. Korijen stabla traženja naziva se **ciljni čvor**.

Sljedećim primjerom 2.6.1 na nekom Datalog programu sa činjenicama demonstrirati ćemo top-down pristup evaluacije programa za određeni upit.

Primjer 2.6.1. *Neka je dan sljedeći Datalog program, skup činjenica i upit.*

$$\left.\begin{array}{l} brid(1,2). \\ brid(1,3). \\ brid(2,4). \\ brid(3,4). \\ brid(4,5). \\ brid(5,6). \\ brid(5,7). \end{array}\right\} \text{činjenice}$$

$$put(X,Y) :- brid(X,Y). \tag{2.6a}$$
$$put(X,Y) :- brid(X,Z),\ put(Z,Y). \tag{2.6b}$$



Slika 2.10: Usmjereni graf

Pretpostavimo da nas zanimaju svi čvorovi u koje se može doći iz čvora 2.

$$Q \equiv ?put(2,X).$$

Stabla dokaza dubine 0

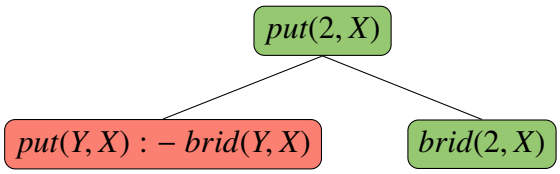
Koje činjenice su osnovne instance od upita Q? U našem primjeru to nije slučaj ni za jednu činjenicu.

Stabla traženja dubine 1

Trebamo naći sva pravila $\boldsymbol{R} \equiv B :- A_1, \ldots, A_k$ *tako da postoji zamjena varijable tako da B odgovora upitu Q. Tada kažemo još i da su Q i B ujedinjeni (eng. unifiable).*

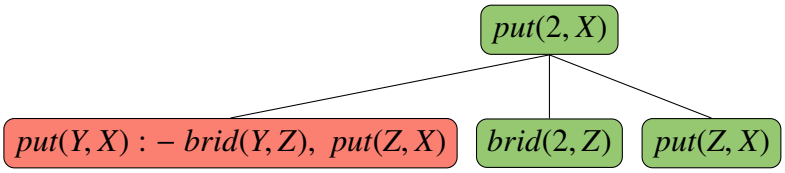
*Za svako pravilo **R** konstruiramo stablo traženja koje kao korijen ima upit Q. Sa korijenom Q spojimo čvor koji sadrži pravilo R i k podciljnih čvorova koji reprezentiraju $A_1, \ldots, A_k$ u svojoj ujedinjenoj formi.*

Pravilo: $put(X, Y) :- brid(X, Y).$



Slika 2.11: Stablo traženja T_1

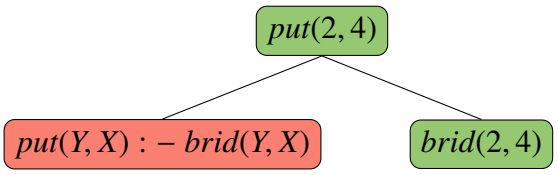
Pravilo: $put(X, Y) :- brid(X, Z),\ put(Z, Y).$



Slika 2.12: Stablo traženja za T_2

- *Da bi iz zadanog stabla traženja (2.11 ili 2.12) generirali stabla dokaza moramo naći zamjenu varijable ρ takvu da za svaki ciljni čvor sa klauzulom C vrijedi $\rho(C) \in$ **P** $\cup$ EDB, gdje je **P** Datalog program zadan skupom pravila. Primjenom zamjene varijable ρ na cijelo stablo traženja dobivamo stablo dokaza sa rezultatom upita u korijenu.*

Zamjena varijable za stablo 2.11 je $\rho := \{X = 4\}$.

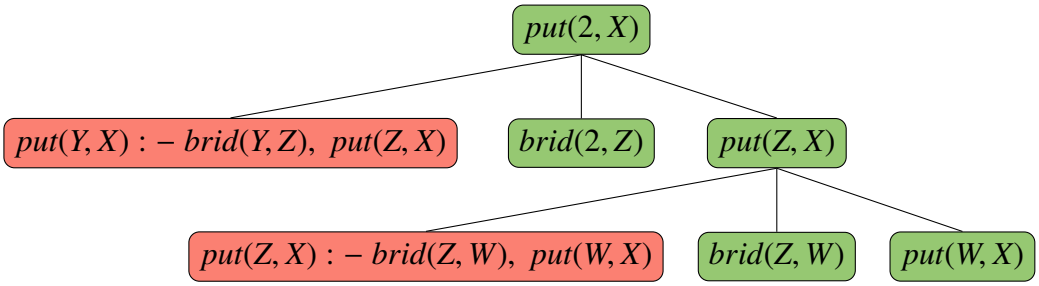


Slika 2.13: Stablo dokaza sa zamjenom varijable $\rho := \{X = 4\}$

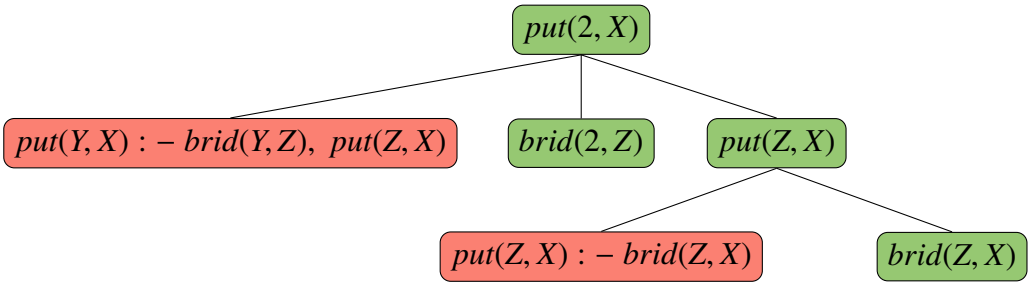
Stablo 2.12 nema zamijene varijable.

- *Za svaki* $n > 1$ *proširujemo sva postojeća stabla traženja dubine* $n-1$ *tretirajući svaki podciljni čvor kao ciljni čvor, odnosno dodajemo nove čvorove koji sadrže pravila i podciljne čvorove.*

Proširujemo stablo traženja 2.12. Tada dobivamo dva stabla traženja 2.14 i 2.15.

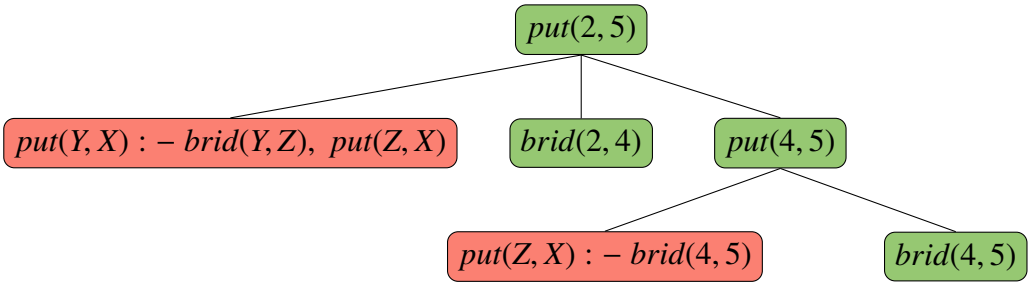


Slika 2.14: Stablo traženja $T_{2,2}$



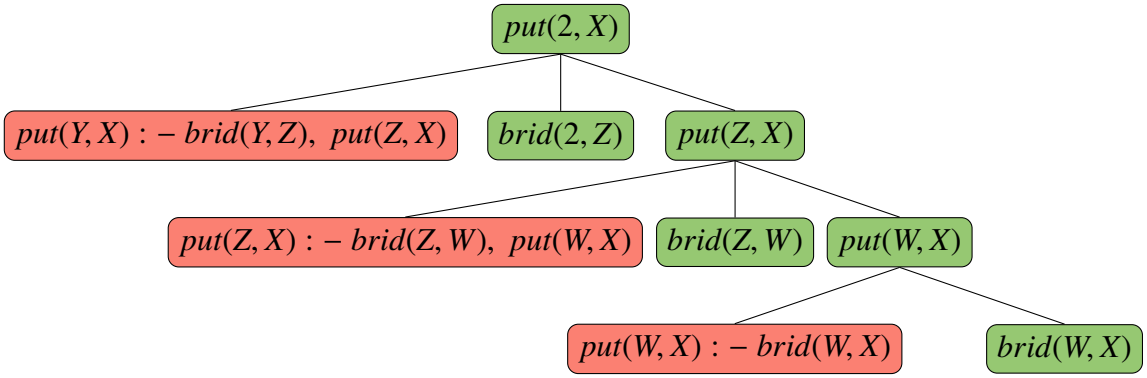
Slika 2.15: Stablo traženja $T_{2,1}$

Stablo 2.14 nema zamijene varijable, dok za stablo 2.15 postoji zamjena varijable $\rho := \{Z = 4,\ X = 5\}$. *Tada odgovarajuće stablo dokaza izgleda kao na slici 2.16*



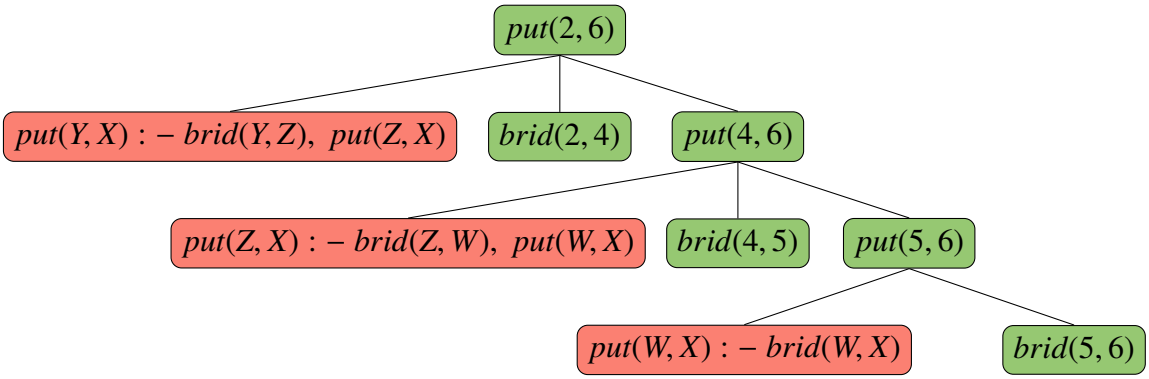
Slika 2.16: Stablo dokaza sa zamjenom varijable $\rho := \{Z = 4,\ X = 5\}$

Proširimo li još jednom stablo traženja 2.14 korištenjem pravila 2.6a dobivamo stablo traženja 2.17

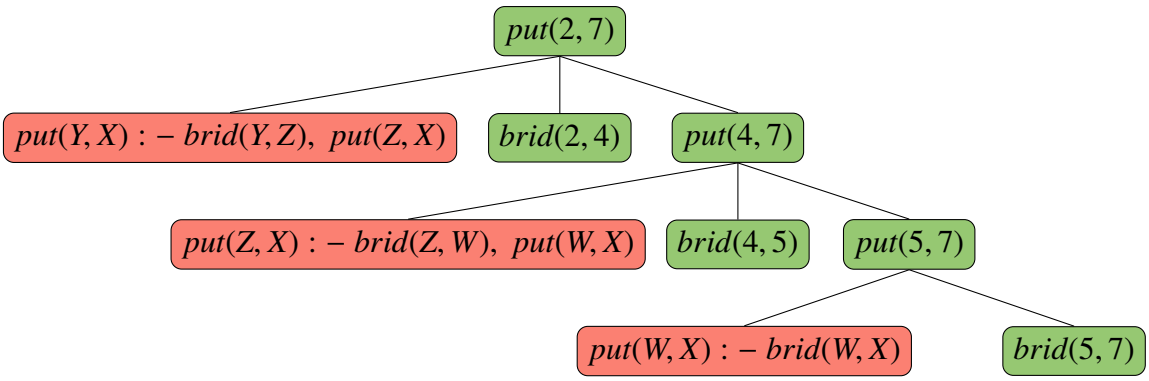


Slika 2.17: Stablo traženja $T_{2,2,1}$

Za stablo traženja 2.17 postoje dvije zamjene varijable $\rho_1 := \{Z = 4,\ W = 5,\ X = 6\}$ *i* $\rho_2 := \{Z = 4,\ W = 5,\ X = 7\}$*, što rezultira odgovarajućim stablima dokaza 2.18 i 2.19*



Slika 2.18: Stablo dokaza sa zamjenom varijable $\rho_1 := \{Z = 4,\ W = 5,\ X = 6\}$

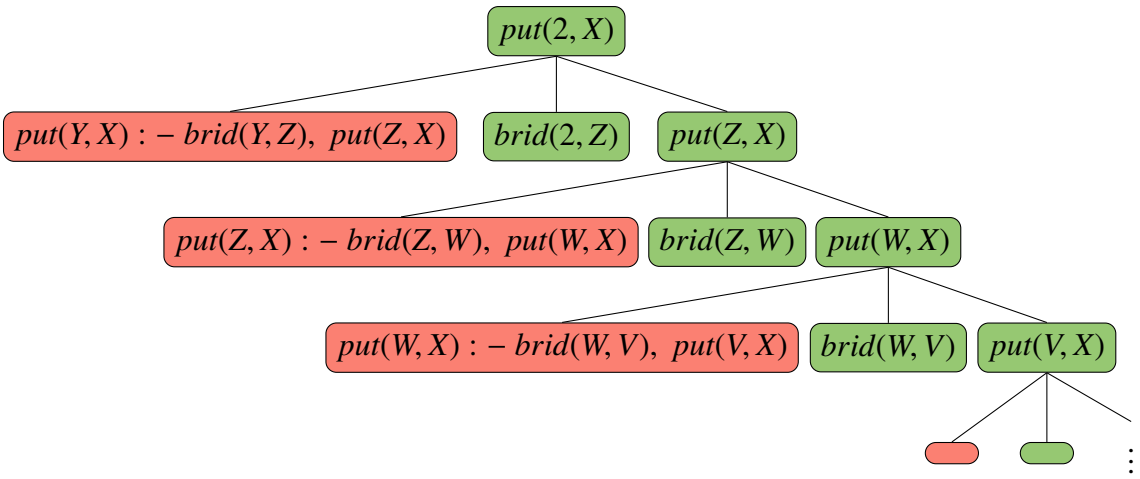


Slika 2.19: Stablo dokaza sa zamjenom varijable $\rho_2 := \{Z = 4,\ W = 5,\ X = 7\}$

Time smo dobili sve čvorove koje možemo posjetiti iz čvora 2*, to su čvorovi* 4,5,6 *i* 7.

Napomena 2.6.2. *Primjenom strategije koja odražava ideju ulančavanja unazad ne generiramo nužno sva stabla dokaza za određeni upit. Moguća su samo stabla dokaza koja imaju osnovne činjenice u svim čvorovima koji su listovi stabla. Takva stabla nazivaju se* ***potpuna stabla dokaza****. No za svako stablo dokaza koje odgovara određenom upitu postoji odgovarajuće potpuno stablo dokaza.*

Također, možemo uočiti da ulančavanjem unazad stabla dokaza mogu dosegnuti proizvoljnu dubinu. Pretpostavimo da u primjeru 2.6.1 stalno iteriramo pravilo 2.6b. Tada bismo dobili stablo traženja koje izgleda kao na slici 2.20



Slika 2.20: Stablo traženja

Pitanje koje se sada nameće jest kada trebamo stati sa generiranjem stabala, jer a-priori ne znamo koja je dubina rekurzije. Znamo da se dubina rekurzije očito povećava sa brojem čvorova. No ipak znamo da je broj smislenih kombinacija činjenica iz ekstenzijske baze podataka i predikata u programu ***P*** *ograničen jer:*

- *Baza podataka i Datalog program su konačni.*
- *Možemo zamijeniti samo bilo koji konstantski simbol iz neke činjenice u bilo kojem predikatnom simbolu na bilo kojoj poziciji varijable.*

Teorem 2.6.3. *Ulančavanje unazad ostaje potpuno ako je dubina stabla traženja ograničena na*

$$\#predikata \cdot \#konstanti^{\,max(args)}$$

gdje je:

$\#predikata$ *– broj korištenih predikatnih simbola*

$\#konstanti$ *– broj korištenih konstantskih simbola*

$max(args)$ *– maksimalni broj argumenata, (mjesnost) svih predkiatnih simbola.*

Dokaz teorema može se pronaći u [7].

Koristeći rezultat teorema 2.6.3 možemo zaustaviti proces ulančavanja unazad nakon zadnje smislene produkcije.

Napomena 2.6.4. *Mnogi algoritmi ulančavanja unazad oslanjaju se na konceptima stabala traženja i stabala dokaza. No strategija generiranja stabala može varirati. U primjeru 2.6.1 stabla traženja su generirana jedno po jedno prema njihovoj dubini. Ovakav pristup sličan je pretraživanju po širini. Alternativno moguć je i pristup pretraživanja po dubini.*

Top-down algoritam koji smo prezentirali u primjeru 2.6.1 je izuzetno naivan. On za određeni upit generira sva moguća stabla traženja i stabla dokaza sve do dubine u najgorem mogućem slučaju. Stoga su performanse takvog algoritma daleko od optimalnih.

S područja logike možemo posuditi koncept **rezolucije**, poznate tehnike koja je originalno bila korištena za automatizirane dokazivače teorema u kontekstu općenitih klauzula (dakle ne samo Hornovih klauzula). Specifična tehnika rezolucije korištena u logičkom programiranju se naziva **SLD rezolucija** (eng. Linear Resolution with Selection Function for Definite Clauses). SLD rezolucija odnosi se na klasu algoritama. Neki popularniji algoritmi u toj klasi su oni generalni algoritmi korišteni u jezicima poput Prologa ili Lispa. Mi ćemo proučiti pojednostavljeni algoritam SLD rezolucije pogodan za Datalog.

2.7 Bottom-up evaluacija

S bottom-up strategijom evaluacije sreli smo se u odjeljku 2.3, gdje smo za evaluaciju imali naivni algoritam iteracije fiksne točke, kretali smo od činjenica iz ekstenzijske baze podataka, zatim smo generirali sve moguće činjenice, pa smo onda odbacivali one koje nisu odgovarale upitu. Dakle, Datalog programe možemo jednostavno evaluirati bottom-up strategijom, no naivni algoritam za evaluaciju (odnosno algoritam fiksne točke) je vrlo ne efikasan. U ovom odjeljku proučiti ćemo bolji algoritam za evaluaciju tzv. semi-naivni algoritam za linearni Datalog program **P**, odnosno algoritam delta-iteracije. Kasnije ćemo ukratko objasniti i ideju kako bismo semi-naivni algoritam proveli i nad nelinearnim Datalog programom. Najprije uvodimo nekoliko definicija kako bismo preciznije definirali linearnost Datalog programa.

Definicija 2.7.1. *Neka su p i q dva predikata intenzijskog djela baze podataka nekog Datalog programa **P**. Kažemo da su p i q **međusobno rekurzivni** ako su oba odgovarajuća čvora tih predikata u grafu predikatnih ovisnosti sadržana u istom ciklusu.*

Definicija 2.7.2. *Neka je dano pravilo $R \equiv B :- A_1, \ldots, A_n$. Kažemo da je pravilo R **linearno u odnosu na** A_i ako se A_i pojavljuje najviše jednom u tijelu pravila R*

Definicija 2.7.3. *Neka je dan Datalog program **P** koji sadrži pravilo $R \equiv B :- A_1, \ldots, A_n$. Kažemo da je pravilo R **linearno u programu P** ako postoji najviše jedan A_i u tijelu pravila (moguće i sam B) koji je međusobno rekurzivan sa B u programu **P**.*

Definicija 2.7.4. *Datalog program* $\boldsymbol{P}$ *je* ***linearan*** *ako:*

i) su sva pravila programa $\boldsymbol{P}$ *linearna u* $\boldsymbol{P}$*;*

ii) graf predikatne ovisnosti od $\boldsymbol{P}$ *sadrži najviše jedan krug (petlju).*

Sjetimo se elementarne produkcije $T_{\mathbf{P}}$, odnosno glavnog operatora za iteraciju fiksne točke. Naivna iteracija fiksne točke bila je $I_{n+1} \coloneqq T_{\mathbf{P}}(I_n)$. Također je vrijedilo $I_n \subseteq I_{n+1}$, odnosno svaki rezultat bio je podskup sljedećeg rezultata. Mogli bi reći da je operator elementarne produkcije čuvao znanje. Ideja je sada da upravo to iskoristimo. Želimo izbjeći ponovno računanje već znanih činjenica na način da osiguramo da nam je barem jedna od činjenica u tijelu pravila nova. Naime, nove činjenice uvijek uključuju nove činjenice zadnjeg koraka iteracije, inače bi već bile izračunate prije.

Algoritmi za semi-naivnu linearnu evaluaciju Datalog programa nazivaju se **delta-iteracije**. U svakom koraku iteracije računamo samo razliku između uzastopnih rezultata:

$$\begin{aligned}
&\Delta I_i \coloneqq I_i \setminus I_{i-1} \\
&\Delta I_1 \coloneqq I_1 \setminus I_0 = T_{\mathbf{P}}(\emptyset) \\
&\Delta I_{i+1} \coloneqq I_{i+1} \setminus I_i = T_{\mathbf{P}}(I_i) \setminus I_i = T_{\mathbf{P}}(I_{i-1} \cup \Delta I_i) \setminus I_i
\end{aligned}$$

Uočimo da vrijedi:

$$\begin{aligned}
I_{i-1} \cup \Delta I_i &= I_i, \\
I_{i-1} \cap \Delta I_i &= \emptyset
\end{aligned}$$

Dakle, važno je efikasno izračunati $\Delta I_{i+1} \coloneqq T_{\mathbf{P}}(I_{i-1} \cup \Delta I_i) \setminus I_i$. Specijalno operator elementarne produkcije $T_{\mathbf{P}}$ je često ne efikasan jer primjenjuje sva pravila na ekstenzijskoj bazi podataka. Zato da bi osigurali efikasnost definiramo **pomoćnu funkciju**.

Definicija 2.7.5. *Pomoćna funkcija elementarne produkcije* $T_{\boldsymbol{P}}$ *je preslikavanje* $\boldsymbol{aux_P} : 2^{B_P} \times 2^{B_P} \to 2^{B_P}$*, tako da vrijedi* $T_{\boldsymbol{P}}(I_{i-1} \cup \Delta I_i) \setminus I_i = \boldsymbol{aux_P}(I_{i-1},\ \Delta I_i) \setminus I_i$.

Pomoćne funkcije možemo izabrati na inteligentan način uzimajući u obzir samo rekurzivne dijelove. Klasična metoda izvođenja pomoćnih funkcija je **simbolička diferencija**.

Operator simboličke diferencije dF možemo primijeniti na odgovarajućim izrazima E relacijske algebre Datalog programa **P**:

- $dF(E) := \Delta R$, ako je E IDB relacija R.
- $dF(E) := \emptyset$, ako je E EDB relacija R.
- $dF(\sigma_\Theta(E)) = \sigma_\Theta(dF(E))$.
- $dF(\pi_\Theta(E)) = \pi_\Theta(dF(E))$.
- $dF(E_1 \cup E_2) = dF(E_1) \cup dF(E_2)$.
- $dF(E_1 \times E_2) = E_1 \times dF(E_2) \cup dF(E_1) \times E_2 \cup dF(E_1) \times dF(E_2)$.
- $dF(E_1 \bowtie_\Theta E_2) = E_1 \bowtie_\Theta dF(E_2) \cup dF(E_1) \bowtie_\Theta E_2 \cup dF(E_1) \bowtie_\Theta dF(E_2)$.

Kada smo pronašli prikladnu pomoćnu funkciju tada **delta iteracije** provodimo na sljedeći način:

Inicijalizacija:

- $I_0 := \emptyset$
- $\Delta I_1 := T_{\mathbf{P}}(\emptyset)$

Iteriramo sve dok $\Delta I_{i+1} = \emptyset$

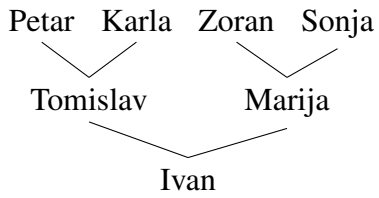
- $I_i := I_{i-1} \cup \Delta I_i$
- $\Delta I_{i+1} := \mathbf{aux}_{\mathbf{P}}(I_{i-1},\ \Delta I_i) \setminus I_i$

Za stratificirane $Datalog^{f,neg}$ programe, iteraciju primjenjujemo na svakom straumu. Sljedećim primjerom ilustrirati ćemo algoritam delta iteracije.

Primjer 2.7.6. *Neka je dan skup činjenica (EDB) i Datalog program* ***P****.*

$$\left.\begin{array}{l} roditelj(Tomislav, Ivan). \\ roditelj(Marija, Ivan). \\ roditelj(Petar, Tomislav). \\ roditelj(Karla, Tomislav). \\ roditelj(Zoran, Marija). \\ roditelj(Sonja, Marija). \end{array}\right\} \textit{činjenice}$$

$$predak(X,Y) :- roditelj(X,Y). \tag{2.7a}$$
$$predak(X,Y) :- roditelj(X,Z),\ predak(Z,Y). \tag{2.7b}$$



Slika 2.21: Obiteljsko stablo

Pomoćnu funkciju izračunat ćemo koristeći simboličke diferencije na odgovarajućim izrazima relacijske algebre. Program prvo transformiramo u relacijsku algebru.

- $predak(X,Y) :- roditelj(X,Y).$

  $\longmapsto \boldsymbol{F} := \pi_{\#1,\#2}(\sigma_{istina}(roditelj)) = roditelj$

- $predak(X,Y) :- roditelj(X,Z),\ predak(Z,Y).$

  $\longmapsto \boldsymbol{F} := \pi_{\#1,\#2}(roditelj \bowtie_{\#2=\#1} predak)$

Ujedinimo sva povezana pravila:

- $eval(predak) := roditelj \cup \pi_{\#1,\#2}(roditelj \bowtie_{\#2=\#1} predak)$

Računamo pomoćnu funkciju $\boldsymbol{aux}_{predak}$ *koristeći simboličke diferencije:*

$$\begin{aligned}
&\boldsymbol{aux}_{predak}(predak,\ \Delta predak) := dF\Big(roditelj \cup \pi_{\#1,\#2}(roditelj \bowtie_{\#2=\#1} predak)\Big)\\
&= dF(roditelj) \cup \pi_{\#1,\#2}(dF(roditelj \bowtie_{\#2=\#1} predak))\\
&= \emptyset \cup \pi_{\#1,\#2}(dF(roditelj) \bowtie_{\#2=\#1} predak \cup roditelj \bowtie_{\#2=\#1} dF(predak)\\
&\quad \cup dF(roditelj) \bowtie_{\#2=\#1} dF(predak))\\
&= \pi_{\#1,\#2}(\emptyset \cup roditelj \bowtie_{\#2=\#1} dF(predak) \cup \emptyset)\\
&= \pi_{\#1,\#2}(roditelj \bowtie_{\#2=\#1} dF(predak))
\end{aligned}$$

Sada kada smo našli pogodnu pomoćnu funkciju radimo delta iteracije.

$$
\begin{aligned}
predak_0 &:= \emptyset \\
\Delta predak_1 &:= T_{\boldsymbol{P}}(\emptyset) = \{predak(Tomislav, Ivan),\ predak(Marija, Ivan), \\
&\qquad predak(Petar, Tomislav),\ predak(Karla, Tomislav), \\
&\qquad predak(Zoran, Marija),\ predak(Sonja, Marija)\} \\
predak_1 &:= predak_0 \cup \Delta predak_1 = \Delta predak_1 \\
\Delta predak_2 &:= \boldsymbol{aux}_{predak}(predak_0, \Delta predak_1) \setminus predak_1 \\
&:= \pi_{\#1,\#2}(roditelj \bowtie_{\#2=\#1} \Delta(predak_1)) \setminus predak_1 \\
&= \{predak(Petar, Ivan),\ predak(Karla, Ivan), \\
&\quad predak(Zoran, Ivan),\ predak(Sonja, Ivan)\} \\
predak_2 &:= predak_1 \cup \Delta predak_2 \\
&= \{predak(Tomislav, Ivan),\ predak(Marija, Ivan), \\
&\quad predak(Petar, Tomislav),\ predak(Karla, Tomislav), \\
&\quad predak(Zoran, Marija),\ predak(Sonja, Marija), \\
&\quad predak(Petar, Ivan),\ predak(Karla, Ivan), \\
&\quad predak(Zoran, Ivan),\ predak(Sonja, Ivan)\} \\
\Delta predak_3 &:= \boldsymbol{aux}_{predak}(predak_1, \Delta predak_2) \setminus predak_2 \\
&:= \pi_{\#1,\#2}(roditelj \bowtie_{\#2=\#1} \Delta(predak_2)) \setminus predak_2 = \emptyset
\end{aligned}
$$

Dakle najmanja fiksna točka je $predak_2 \cup roditelj$

Napomena 2.7.7. *U ovom odjeljku smo proučavali i ilustrirali semi-naivnu metodu evaluacje (delta-iteracije) za linearni Datalog. Isto smo mogli raditi i za nelinearni Datalog program. Samo u tom slučaju imali bismo operator diferencije višeg reda, čime bi izrazi postali vrlo glomazni. Ukoliko imamo nelinearni Datalog program, možemo također transformirati svako nelinearno pravilo u skup pravila u kojem svako pravilo postaje linearno u odnosu na svaku od varijabli originalnog pravila, tada možemo koristiti varijantu semi naivne metode evaluacije. Zbog jednostavnosti promotrimo nelinearno pravilo* $C \equiv B :- A_1, \ldots, A_n, B^1, \ldots, B^m$*, gdje imamo m ponavljanja predikata glave B u tijelu pravila C. Pravilo C je tada ekvivalentno sljedećem programu.*

$$
\begin{aligned}
&B :- A_1, \ldots, A_n, B, R_1. \\
&R_1 :- B, R_2. \\
&R_2 :- B, R_3. \\
&\vdots \\
&R_{m-1} :- B.
\end{aligned}
$$

Primjenom ove konstrukcije na sva rekurzivna nelinearna pravila dobivamo program čija su sva pravila linearna u odnosu na sve varijable. Takav sustav možemo riješiti varijantom semi-naivnog algoritma opisanog u [7] koji je nešto manje efikasan od klasičnog semi-naivnog algoritma opisanog u ovom odjeljku ali i dalje nudi uštede u računanju evaluacije.

2.8 Guranje selekcije

Transformacija Dataloga u relacijsku algebru otvara vrata još jednoj vrsti optimizacije. Ukoliko upit uključuje operacije spajanja relacije (join) ili Kartezijev produkt tada guranjem svih selekcija sve do ulaznih relacija možemo izbjeći velike međurezultate. Takva vrsta optimizacije naziva se **guranje selekcije** (eng. push-selection). No sada imamo novi operator u našem planu upita, **iteraciju najmanje fiksne točke** (u oznaci LFP).

Sljedeće pravilo govori nam kada možemo zamijeniti iteraciju najmanje fiksne točke i selekciju.

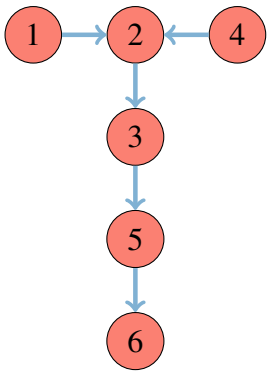
- Neka je p predikat u linearnom i rekurzivnom Datalog programu. Pretpostavimo da imamo upit oblika $?p(\ldots, c, \ldots)$ koji veže neku varijablu X na i-toj poziciji sa konstantom c. Selekcija $\sigma_{\#i=c}$ i iteracija najmanje fiksne točke (LFP) mogu se sigurno zamijeniti ako se X pojavljuje u svim literalima s predikatom p na i-toj poziciji.

Primjer 2.8.1. *Ilustrirajmo guranje selekcije na sljedećem Datalog programu i skupu činjenica.*

$$\left.\begin{array}{l} brid(1,2). \\ brid(4,2). \\ brid(2,3). \\ brid(3,5). \\ brid(5,6). \end{array}\right\} \textit{činjenice}$$

$$put(X,Y) :- brid(X,Y). \tag{2.8a}$$

$$put(X,Y) :- brid(X,Z),\ put(Z,Y). \tag{2.8b}$$



Slika 2.22: Usmjereni graf

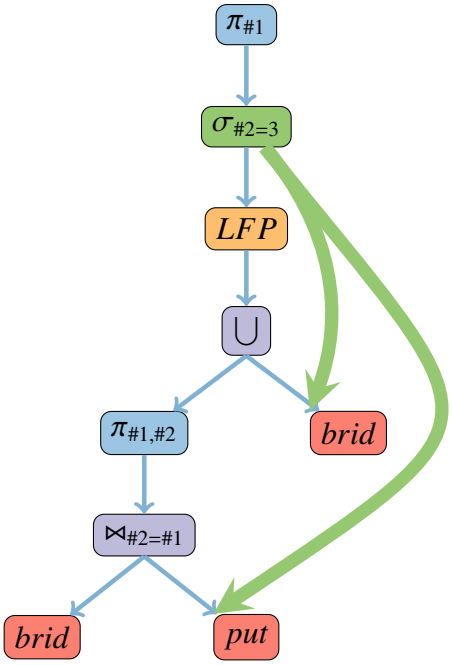
Ukoliko Datalog program prebacimo u relacijsku algebru dobivamo izraz $brid \cup \pi_{\#1,\#2}(brid \bowtie_{\#2=\#1} put)$ *koji koristimo u svakoj iteraciji. Neka nam je zadan upit* $?put(X,3)$*, odnosno zanima nas iz kojih sve čvorova postoji put do čvora 3. Evaluaciju fiksne točke tada možemo prikazati izrazom*

$$\pi_{\#1}\sigma_{\#2=3}\Big(LFP(brid \cup \pi_{\#1,\#2}(brid \bowtie_{\#2=\#1} put))\Big) \tag{2.9}$$

gdje bismo prvo izračunali fiksnu točku izraza $brid \cup \pi_{\#1,\#2}(brid \bowtie_{\#2=\#1} put)$ *a potom na kraju selekcijom* $\sigma_{\#2=3}$ *odbacili bismo sve putove koji ne završavaju u čvoru 3. Ovakav način evaluacije pa zatim selekcije je vrlo ne efikasan. Upit* $\pi_{\#1}\sigma_{\#2=3}$ *u izrazu 2.9 veže drugi argument predikata put sa konstantom* 3.

$$put(X,Y) :- brid(X,Y).$$
$$put(X,Y) :- brid(X,Z),\ put(Z,Y).$$

Stoga selekciju možemo gurnuti sve do relacija brid i put (slike 2.23 i 2.24), tako da je zamjena selekcije i iteracije najmanje fiksne točke (LFP) ***sigurna****.*



Slika 2.23: Grafički prikaz guranja selekcije u izrazu 2.9

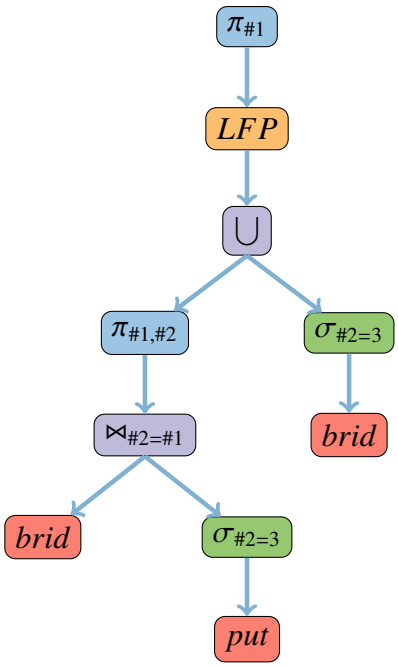
Da bismo dali odgovor na zadani upit uzimamo u obzir samo činjenice i pravila koja imaju korektan drugi argument.

$$\left.\begin{array}{l} brid(2,3). \end{array}\right\} \quad \text{činjenica}$$

$$\left.\begin{array}{l} put(2,3). \end{array}\right\} \quad \text{pravilo 2.8a}$$

$$\left.\begin{array}{l} put(1,3). \\ put(4,3). \end{array}\right\} \quad \text{pravilo 2.8b}$$

Što nam daje kao rezultat skup $\{1,\ 2,\ 4\}$.



Slika 2.24: Grafički prikaz gurnute selekcije sve do relacija *brid* i *put*

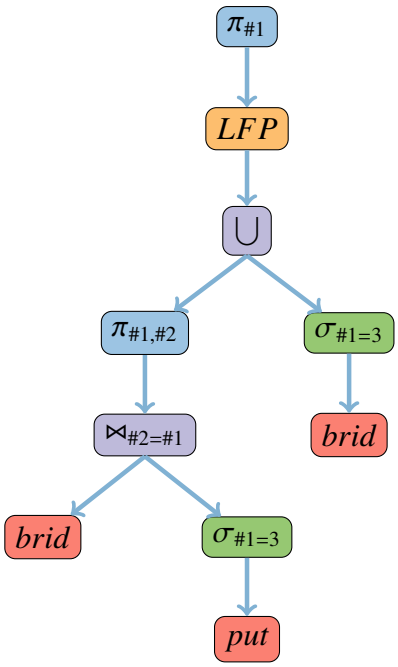
Primjer 2.8.2. *Ovim primjerom ilustrirati ćemo kada guranje selekcije nije moguće. Neka nam je dan isti Datalog program i skup činjenica kao u primjeru 2.8.1. No upit neka je drugačiji. Neka nas sada zanima do kojih sve čvorova postoji put iz čvora* 3*, odnosno postavljamo upit* $?put(3, Y)$*. Evaluaciju fiksne točke tada možemo prikazati izrazom*

$$\pi_{\#1}\sigma_{\#1=3}\Big(LFP(brid \cup \pi_{\#1,\#2}(brid \bowtie_{\#2=\#1} put))\Big) \tag{2.10}$$

gdje bismo prvo izračunali fiksnu točku izraza $brid \cup \pi_{\#1,\#2}(brid \bowtie_{\#2=\#1} put)$ *a potom na kraju selekcijom* $\sigma_{\#1=3}$ *odbacili bismo sve putove koji ne počinju čvoru 3. No razlog zašto ovdje nije moguće izvršiti sigurnu zamjenu selekcije i najmanje fiksne točke (LFP) leži u sljedećoj činjenici. Upit* $\pi_{\#1}\sigma_{\#1=3}$ *u izrazu 2.10 veže prvi argument puta sa konstantom* 3*.*

$$put(X, Y) :- brid(X, Y). \tag{2.11a}$$
$$put(X, Y) :- brid(X, Z),\ put(Z, Y). \tag{2.11b}$$

I sada kada bismo gurnuli selekciju sve do relacija brid i put kao što je na slici 2.25, zbog pravila 2.11b biti će narušena sigurnost zamjene. Jer selekcija $\sigma_{\#1=3}$ *i najmanja fiksna točka (LFP) mogu se sigurno zamijeniti samo ako se X pojavljuje na prvoj poziciji u* ***svim literalima*** *koji sadrže predikat put, što nije slučaj kod pravila 2.11b.*



Slika 2.25: Grafički prikaz gurnute selekcije sve do relacija *brid* i *put*

Kada bi pokušali izvršiti zamjenu selekcije i najmanje fiksne točke (LFP) kao na slici 2.25 imali bismo:

$brid(3,5).$	*činjenica*
$put(3,5).$	*pravilo* 2.11*a*
$\emptyset$	*pravilo* 2.11*b*

Što kao rezultat daje {5}*, što je očito pogrešno, odnosno nepotpuno, jer znamo da iz čvora* 3*, postoji put do čvora* 5 *i do čvora* 6*.*

2.9 Magični skupovi

U ovom odjeljku opisati ćemo jednu od metoda prerade programa. Osnovna ideja prerade programa je transformacija programa **P** u neki semantički ekvivalentni program $\mathbf{P}'$ koji tada možemo brže evaluirati koristeći istu evaluacijsku tehniku.

Pametan način prerade programa funkcionirao bi na sljedeći način:

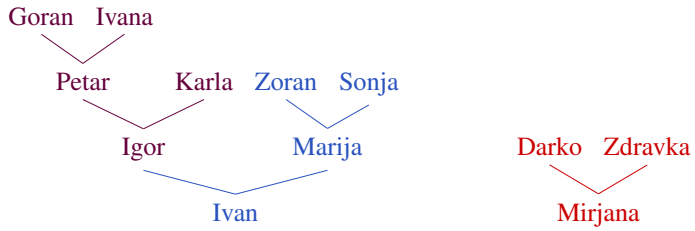
Za zadani upit i program **P**:

$$predak(X,Y) :- roditelj(X,Y).$$
$$predak(X,Y) :- predak(X,Z),\ roditelj(Z,Y).$$
$$predak(Igor,\ Y)$$

Ekvivalentni program $\mathbf{P}'$ za isti upit izgleda ovako:

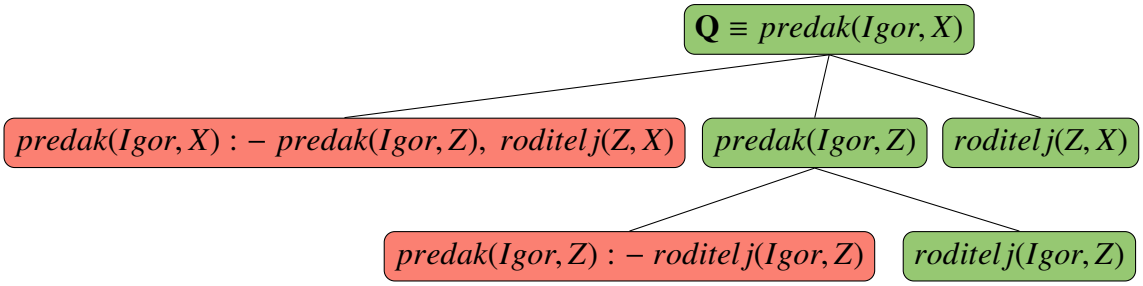
$$predak(Igor,\ Y) :- roditelj(Igor,\ Y).$$
$$predak(Igor,\ Y) :- predak(Igor,\ Z),\ roditelj(Z,Y).$$
$$?predak(Igor,\ Y)$$

Jednostavna transformacija preskočiti će dedukciju mnogih (u ovom slučaju svih) nepotrebnih činjenica. Ovo je zapravo ništa drugo nego guranje selekcije koju smo obradili u odjeljku 2.8 koristeći relacijsku algebru. Ovakva transformacija je prilično direktna i jednostavna. No postoje i ne intuitivne ali efektivne metode prerade programa, **magični skupovi**. Magični skupovi su metoda prerade programa koja iskorištava sintaktičku formu upita. Osnovna ideja jest uhvatiti neke od vezanih uzoraka top-down strategije te ih uklopiti u metodu prerade programa. Ako postoji podcilj sa vezanim argumentom, riješavanje tog podcilja može dovesti do novih instanci drugih argumenata u originalnom pravilu. Samo potencialno korisne dedukcije bi trebale biti provedene. Pogledajmo stablo nasljeđivanja 2.26. Ako postavimo upit $?predak(Igor,\ Y)$ (tko su Igorovi predci?) tada ima smisla gledati samo podstablo označeno ljubičastom bojom na slici 2.26. Dio stabla označen plavom bojom, te pogotovo stablo označeno crvenom bojom su suvišni i njih ne želimo uzimati u obzir.



Slika 2.26: Obiteljska stabla

Tipično top-down stablo traženja za cilj $predak(Igor, X)$ sa ograničenjem mogućih zamjena varijabli izgleda kao na slici 2.27. Sada je cilj inkorporirati takvo ograničenje u metodama prerade programa.



Slika 2.27: Stablo traženja

Kod metoda prerade programa, propagiranje vezanja je kompliciranije nego korištenje top-down pristupa. Strategija magičnog skupa bazirana je na dograđivanju pravila sa dodatnim ograničenjima (prikupljenih u magičnom predikatu). To se olakšava "**ukrašvanjem**" (eng. adorning) predikata. Koristi se predaja informacija u prolazu da bi se propagirala informacija vezanja. Da bi mogli izvršiti transformaciju magičnih skupova, potrebne su nam neke pomoćne definicije i razmatranja.

- Svaki **upit** (cilj) možemo gledati kao **pravilo**, te ga kao takvog dodajemo u program. Primjerice, upit $?predak(Igor, X)$ pretvaramo u pravilo $q(X) :- predak(Igor, X)$.
- Argumenti predikata mogu se **razlikovati**. Argumenti koje razlikujemo imaju doseg ograničen ili sa konstantama unutar istog predikata ili sa varijablama koje su već i same ograničene. Ili preciznije, argument razlikujemo ako:
  - je konstanta;
  - ili ako je vezan "ukrasom";
  - ili ako se pojavljuje u činjenici ekstenzijske baze podataka koja ima argument koji razlikujemo.
- Razlikujemo pojavljivanja predikata ako razlikujemo sve njegove argumente. U slučaju činjenice ekstenzijske baze podataka razlikujemo ili sve ili nijedan od argumenata. Pojavljivanja predikata su tada ukrašena (pribilježena) kako bi mogli izraziti koje argumente razlikujemo. Ukrašavanja se dodaju predikatu, primjerice $p^{fb}(X, Y)$ označava da je varijabla X slobodna (free), a varijabla Y vezana (bound). Takav predikat razlikujemo od predikata $p^{bb}(X, Y)$ gdje su obje varijable X i Y vezane.
- Za svaki argument postoje dva moguća ukrasa:

  b – označava vezane argumente, odnosno varijable koju razlikujemo.

  f – označava slobodne argumente, odnosno varijable koje ne razlikujemo.

- Za predikat sa n argumenata postoji 2^n mogućih ukrašenih pojavljivanja. Primjerice, $p^{ff}(X,Y)$, $p^{fb}(X,Y)$, $p^{bf}(X,Y)$ i $p^{bb}(X,Y)$. Ta ukrašena pojavljivanja predikata se tretiraju kao različiti predikati od kojih je svaki definiran svojim skupom pravila.
- Ideja metode magičnog skupa je da magični skup sadrži sve moguće interesantne konstantske vrijednosti. Magični skup računamo **rekurzivno** koristeći **magična pravila**.
- Svako pojavljivanje ukrašenog predikata ima svoja pravila koja ga definiraju. U tim pravilima atributi su ograničeni prema obrascu ukrasa na magični skup.

Sada preostaju sljedeći problemi. Kako se računa magični skup? I kako su pravila za pojavljivanja ukrašenih predikata definirana? Prije nego što riješimo te probleme, moramo saznati koja su ukrašena pojavljivanja potrebna. Odnosno, moramo naći dostupni ukrašeni sustav. Inkorporiramo upit kao pravilo i zamijenimo sve predikate njihovim odgovarajućim ukrasima.

Za primjer:

$$?predak(Igor, X)$$
$$predak(X,Y) :- roditelj(X,Y).$$
$$predak(X,Y) :- predak(X,Z),\ roditelj(Z,Y).$$

Inkorporiramo upit kao pravilo:

$$q(X) :- predak(Igor, X)$$
$$predak(X,Y) :- roditelj(X,Y).$$
$$predak(X,Y) :- predak(X,Z),\ roditelj(Z,Y).$$

Pojavljivanja ukrašenih predikata:

$$R_0 \equiv q^f(X) :- predak^{bf}(Igor, X)$$
$$R_1 \equiv predak^{bf}(X,Y) :- roditelj(X,Y).$$
$$R_2 \equiv predak^{bf}(X,Y) :- predak^{bf}(X,Z),\ roditelj(Z,Y).$$

Da bi definirali magični skup, stvaramo **magična pravila**.

- Za svako pojavljivanje ukrašenog predikata u pravilu predikata intenzijske baze podataka stvaramo magično pravilo koje odgovara desnoj strani tog pravila.

– Pojavljivanja predikata zamjenjuju se magičnim predikatima, vezani argumenti se koriste u glavi pravila, slobodne argumente ispuštamo.

– Magični predikati u glavi su pribilježeni svojim podrijetlom (pravilo i predikat), dok su oni na desnoj strani pribilježeni samo predikatom.

$$q^f(X) :- predak^{bf}(Igor, X) \longmapsto magični\_R_0\_predak^{bf}(Igor)$$

$$predak^{bf}(X, Y) :- predak^{bf}(X, Z),\ roditelj(Z, Y)$$
$$\longmapsto magični\_R_2\_predak^{bf}(X) :- magični\_predak^{bf}(X)$$

Dakle, za jednu pojavu ukrašenog predikata dobivamo više magičnih predikata.

- Ovisno o pravilu stvaranja primjerice $magični\_R_0\_predak^{bf}$ i $magični\_R_2\_predak^{bf}$, oba koriste $magični\_predak^{bf}$.

- Sada su nam potrebna komplementarna pravila koja spajaju magične predikate. Ukrašeni magični predikat slijedi iz specijalnog magičnog predikata pravila sa istim ukrasom.

$$magični\_predak^{bf}(X) :- magični\_R_0\_predak^{bf}(X)$$

$$magični\_predak^{bf}(X) :- magični\_R_2\_predak^{bf}(X)$$

Na kraju imamo kompletnu definiciju magničnih predikata sa različitim ukrasima. U našem primjeru imamo samo **bf** ukras.

$$magični\_R_0\_predak^{bf}(Igor)$$

$$magični\_R_2\_predak^{bf}(X) :- magični\_predak^{bf}(X)$$

$$magični\_predak^{bf}(X) :- magični\_R_0\_predak^{bf}(X)$$

$$magični\_predak^{bf}(X) :- magični\_R_2\_predak^{bf}(X)$$

Magični $magični\_predak^{bf}$ skup sadrži sve moguće korisne konstante koje treba uzeti u obzir kod evaluacije podcilja *predak* sa vezanim prvim argumentom u trenutnom programu. Kako su svi magični skupovi definirani, originalna pravila dostupnog ukrašenog sustava moraju se ograničiti obzirom na odgovarajuće magične skupove. Svako pravilo koje koristi ukrašeni predikat intenzijske baze podataka u svom tijelu, nadopunimo dodatnim literalom koji sadrži odgovarajući magični skup.

$$predak^{bf}(X,Y) :- predak^{bf}(X,Z),\ roditelj(Z,Y)$$

$$\longmapsto predak^{bf}(X,Y) :- magični\_predak^{bf}(X),\ predak^{bf}(X,Z),\ roditelj(Z,Y)$$

Na kraju dobivamo prerađeni program:

$$magični\_R_0\_predak^{bf}(Igor).$$

$$magični\_R_2\_predak^{bf}(X) :- magični\_predak^{bf}(X).$$

$$magični\_predak^{bf}(X) :- magični\_R_0\_predak^{bf}(X).$$

$$magični\_predak^{bf}(X) :- magični\_R_2\_predak^{bf}(X).$$

$$predak^{bf}(X,Y) :- roditelj(X,Y).$$

$$predak^{bf}(X,Y) :- magični\_predak^{bf}(X),\ predak^{bf}(X,Z),\ roditelj(Z,Y).$$

$$q^{f}(X) :- predak^{bf}(Igor,X)$$

U ovom konkretnom primjeru moguće su još neke optimizacije. Tako nije potrebno razdvojiti pojavljivanja $magični\_R_0\_predak^{bf}$ i $magični\_R_2\_predak^{bf}$, jer među njima nema ovisnosti. Stoga ih možemo objediniti i preimenovati. Također imamo samo jedan ukras **bf**, pa ga možemo ispustiti. Tako dobivamo sljedeći semantički ekvivalentan i kraći program:

$$magični(Igor).$$

$$q(X) :- predak(Igor,X).$$

$$predak(X,Y) :- roditelj(X,Y).$$

$$predak(X,Y) :- magični(X),\ predak(X,Z),\ roditelj(Z,Y).$$

Napomena 2.9.1. *Formalni algoritam prerade programa metodom magičnih skupova dan je u [7].*

Poglavlje 3

Studijski primjer

Ovo poglavlje odnosi se na praktični dio diplomskog rada. U prvom odjeljku opisujemo jezik *LogiQL* (*Logic Query Language*) u kojem možemo realizirati deduktivnu bazu podataka. Tu ćemo navesti neka osnovna svojstva *LogiQL* jezika koja će nam biti potrebna za realizaciju deduktivne baze podataka.

U drugom odjeljku u prethodno opisanom jeziku realiziramo samu deduktivnu bazu podataka za određenu domenu problema. Tu ćemo vidjeti mnoge prednosti takve baze podataka postavljanjem raznih rekurzivnih upita kakvi se ne mogu izraziti standardnim *S QL*-om koji slijedi *S QL*2 standard.

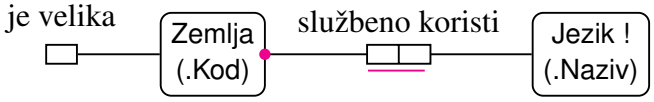
3.1 Logic Query Language – LogiQL

Među mnogim *Datalog* sustavima i sustavima baziranim na *Datalogu*, *Logic Query Language* (*LogiQL*) je dobar primjer suvremenog sustava s odličnim rezultatima u industrijskoj poslovnoj optimizaciji, a posebno u prediktivnoj i kontrolnoj analizi koje uključuju velike skupove podataka. *LogiQL* proširuje standardni *Datalog* na više načina. On tako podržava razdvajanje programa u nezavisne blokove (module), sadrži razne ugrađene funkcije (agregatne, aritmetičke, funkcije za manipulaciju stringova, konverzije tipova itd.), prepoznaje razne tipove podataka (string, int, float, decimal, datetime itd.), podržava kreaciju vlastitih tipova entiteta, omogućuje transakcije i mnoge druge napredne značajke.

Dok zadržava mnoge tradicionalne značajke jezika *Datalog*, *LogiQL* pruža i dodatnu sintaksu kako bi se napravila razlika između raznih ograničenja koji čuvaju integritet baze podataka i samih pravila izvoda, te pojednostavnila formulacija raznih aspekata (primjerice definicija funkcijskih predikata). Komercijalnu verziju *LogiQL – a* može se nabaviti preko kompanije *LogicBlox* koja je i razvila sami sustav. Dostupna je također i besplatna ”cloud-

based" verzija *LogiQL – a* kojoj možemo pristupiti putem web preglednika (Chrome ili Firefox) koju ćemo koristiti za potrebe ovog diplomskog rada.

U ovom odjeljku cilj je da se na jednostavnim primjerima usvoje neke važne ideje i koncepti jezika *LogiQL*. No odakle uopće početi? Krenimo od nečeg što nam je već poznato. Slika 3.1 prikazuje jednostavan model podataka u Object Role Modeling (ORM) [3] notaciji. Tu vidimo da su zemlje entiteti koje identificirmo njihovim kodom i da su jezici entiteti koje identificiramo njihovim nazivom. Unarni činjenični tip "Zemlja je velika" koristimo da bismo zabilježili koja od zemalja ima veliku površinu, dok binarni činjenični tip "Zemlja službeno koristi Jezik" bilježi za svaku zemlju jezik koji koristi u službenim dokumentima (to može i ne mora biti službeni jezik neke zemlje, primjerice Australia i SAD nemaju službeni jezik ali oboje koriste engleski jezik u službenim dokumentima). Dijagram na slici 3.1 specificira sve činjenice u terminima uloga koje igraju objekti. Svi činjenični tipovi u ORM dijagramu su skupovno orijentirani, tako da njihove pripadne tablice činjenica neće nikada prikazivati duplikate. Logički predikat je uređeni skup uloga u jednom činjeničnom tipu, pa tako kombinacija imena objektnih tipova ("Zemlja" i "Jezik") sa nazivima predikata ("je velika" i "službeno koristi") daje nam naziv činjeničnog tipa. Crta ispod uloga činjeničnog tipa "Zemlja službeno koristi Jezik" označuje da zemlja može koristiti više jezika u službenim dokumentima, i da jezik može biti korišten u više zemlja. Točka na dijagramu označava da svaka zemlja službeno koristi neki jezik, dok znak "!" koji se nalazi kraj imena objekta "Jezik" naglašava da je "Jezik" nezavisan. Primjerice, može postojati neki drevni jezik koji se više službeno nigdje ne koristi.



Slika 3.1: Object Role Modeling dijagram

U relacijskoj shemi 3.1 za isti problem možemo vidjeti tri relacijske tablice *Zemlja* i *KoristiSluzbenoJezik* i *Jezik* sa podcrtanim primarnim ključevima.

$$\begin{aligned} &\textbf{Zemlja} \ (\ \underline{kodZemlje},\ jeVelika\ ) \\ &\textbf{KoristiSluzbenoJezik} \ (\ \underline{kodZemlje,\ nazivJezika}\ ) \\ &\textbf{Jezik} \ (\ \underline{nazivJezika}\ ) \end{aligned} \tag{3.1}$$

U praksi većina entiteta (npr. *Zemlja* ili *Jezik*) mogu se identificirati na jednostavan način povezujući ih sa jednom vrijednošću (npr. naziv jezika ili kod zemlje). Način na koji se vrijednost referira na svoj pripadni entitet naziva se **referentni način** ili kraće *refmode*.

U ORM dijagramu takve jednostavne sheme referenci su obično prikazane u kompaktnoj formi uključujući referentni način (Kod, Naziv) u zagrade odmah ispod imena entiteta kao na slici 3.1. Točka prije refmod-a indicira da je to popularan refmod. Na slici 3.2 crte koje označavaju ograničenje u svrhu osiguravanja jedinstvenosti nad svakom ulogom činjeničnog tipa indiciraju da su to jedan naprema jedan relacije (npr. svaka zemlja ima najviše jedan kod, i svaki kod je kod najviše jedne zemlje). Velike točke na referentnim ulogama indiciraju da su te uloge obavezne (tj. svaka zemlja ima kod i svaki jezik ima naziv). Stoga predikati koji preslikavaju *Zemlju* u *KodZemlje* te *Jezik* u *NazivJezika* su injekcje. Označavanjem ograničenja koje osigurava jednistvenost nad ulogama od *KodZemlje* i *NazivJezika* sa duplim crtama indicira da te injekcije daju preferiranu referentnu shemu za *Zemlju* i *Jezik*.

Slika 3.2: Referentne sheme za Zemlju i Jezik

U *LogiQL – u* entitet *Zemlja* zajedno sa svojom referentnom shemom može se deklarirati na sljedeći način:

$$Zemlja(z),\ imaKodZemlje(z : kz) \text{ -> } string(kz). \tag{3.2}$$

U formuli 3.2 *Zemlja* je unarni predikat za entitet *Zemlja*, *string* je unarni predikat za tip podataka string, te *imaKodZemlje* je binanri predikat za referentnu vezu koja povezuje zemlje sa njihovim kodovima (ili preciznije vrijednosti podataka koji reprezentiraju te kodove zemalja). Simbol desne strelice "->" predstavlja materijalnu implikaciju. Dvotočka ":" u binarnom predikatu *imaKodZemlje* razlikuje *imaKodZemlje* kao *refmode* (i stoga injektivan) predikat, pa više nema potrebe za pisanjem daljnjeg koda kako bi se primijenila ograničenja na obavezno članstvo i jedinstvenost na tom predikatu. Formule u *LogiQL – u* uvijek završavaju točkom. Formula 3.2 tretira se kao skračenica za sljedeće četiri logičke formule:

$$\forall z\ (\ Zemlja\ z \rightarrow Entitet\ z\ ) \tag{3.3a}$$
$$\forall z, kz\ [\ z\ imaKodZemlje\ kz\ \rightarrow\ (Zemlja\ z\ \wedge\ string\ kz)\ ] \tag{3.3b}$$
$$\forall z\ [\ Zemlja\ z\ \rightarrow\ \exists^{1} kz\ (\ z\ imaKodZemlje\ kz\ )\ ] \tag{3.3c}$$
$$\forall kz\ \exists^{0\ldots 1} z\ (\ z\ imaKodZemlje\ kz\ ) \tag{3.3d}$$

Formula 3.3a deklarira da je *Zemlja* entitet, tj za svaki individualni z, ako je z *Zemlja*, tada je z entitet. Formula 3.3b je ograničenje koje deklarira tipove argumenata predikata *imaKodZemlje*. Formula 3.3c predstavlja ograničenje u svrhu očuvanja integriteta kako bi se osiguralo da svaka zemlja ima točno jedan (i najviše jedan) kod. Formula 3.3d ograničava da se svaki kod zemlje refeira na najviše jednu zemlju. Da bi izbjegli pisanje kvantifiktora "∀", *LogiQL* formule pretpostavljaju da su varijable koje se pojavljuju sa obje strane znaka "->" implicitno univerzalno kvantificirane. Analogno entitet *Jezik* možemo deklarirati zajedno sa svojom referentnom shemom na sljedeći način:

$$Jezik(l),\ imaNazivJezika(l : nl) \text{ -> } string(nl). \tag{3.4}$$

Činjenične tipove "Zemlja je velika" i "Zemlja službeno koristi Jezik" možemo definirati na sljedeći način:

$$\begin{aligned} &jeVelika(z) \text{ -> } Zemlja(z). \\ &sluzbenoKoristi(z,\ l) \text{ -> } Zemlja(z),\ Jezik(l). \end{aligned} \tag{3.5}$$

Formule 3.5 odgovaraju sljedećim logičkim formulama:

$$\begin{aligned} &\forall z\ (\ z\ jeVelika\ \rightarrow\ Zemlja\ z\ ) \\ &\forall z,\ l\ [\ z\ sluzbenoKoristi\ l\ \rightarrow\ (\ Zemlja\ z\ \wedge\ Jezik\ l\ )\ ] \end{aligned} \tag{3.6}$$

To znači da predikat *jeVelika* možemo primijeniti samo na zemlje, dok predikat *sluzbenoKoristi* možemo primijeniti samo na parove (*zemlja*, *jezik*). Obavezno članstvo, da svaka zemlja službeno koristi neki jezik možemo u *LogiQL – u* napisati na sljedeći način:

$$Zemlja(z) \text{ -> } sluzbenoKoristi(z, \_). \tag{3.7}$$

U formuli 3.7 znak "_" predstavlja anonimnu varijablu koja označava da na tom mjestu može biti neka vrijednost jezika. Formula 3.7 odgovara sljedećoj logičkoj formuli:

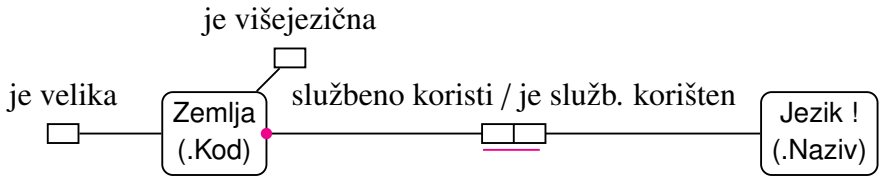
$$\forall z\ (\ Zemlja\ z\ \rightarrow\ \exists l\ z\ sluzbenoKoristi\ l\ ) \tag{3.8}$$

Kako bismo istražili još neka dodatna svojstva *LogiQL* jezika proširimo ORM dijagram na slici 3.1 na način prikazan slikom 3.3. Primijetimo prvo da ORM dijagram na slici 3.3 sada dodatno prikazuje i inverzni predikat za činjenični tip "Zemlja službeno koristi Jezik". Inverzni predikat možemo deklarirati koristeći pravilo 3.9, gdje simbol lijeve strelice "<-" označuje inverznu materijalnu implikaciju u logici, te ga čitamo "ako".

$$jeSluzbenoKoristenOd(l, z) \text{ <- } sluzbenoKoristi(z, l). \tag{3.9}$$

Formula 3.9 odgovara sljedećoj logičkoj formuli:

$$\forall l,\ z\ (\ l\ jeSluzbenoKoristenOd\ z \leftarrow\ z\ sluzbenoKoristi\ l\ ) \tag{3.10}$$



Slika 3.3: Object Role Modeling dijagram

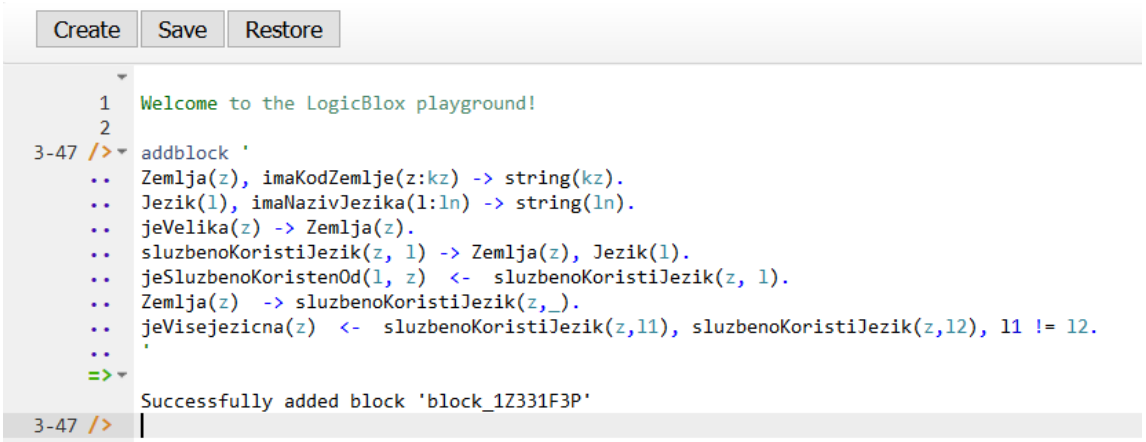
Sljedeća stvar koju možemo primijetiti na ORM dijagramu na slici 3.3 jest da imamo i predikat koji nam govori da neke zemlje mogu biti višejezične. Takav predikat bit će izveden koristeći sljedeće pravilo:

$$jeVisejezicna(z) \text{ <- } sluzbenoKoristi(z, l1),\ sluzbenoKoristi(z, l2),\ l1! = l2. \tag{3.11}$$

Formula 3.11 odgovara sljedećoj logičkoj formuli gdje možemo primijetiti da su varijable koje se pojavljuju samo u tijelu pravila implicitno egzistencijalno kvantificirane:

$$\forall z\ [\ z\ jeVisejezicna\ \leftarrow\ \exists l_1, l_2\ (z\ sluzbenoKoristi\ l_1\ \wedge\ z\ sluzbenoKoristi\ l_2\ \wedge\ l_1 \neq l_2)\ ]$$

U nastavku ćemo vidjeti kako možemo realizirati do sada konstruiranu bazu podataka u online verziji *LogiQL − a*, te kako možemo dodavati činjenice i postavljati upite nad bazom. Za unos sheme koristimo besplatni alat **REPL** (Read-Eval-Print-Loop) dostupan na web adresi [5]. Otvaranjem REPL-a u prozoru web pretraživača (Chrome ili Firefox) automatski se stvara novi radni prostor gdje možemo dodati blok naredbom *addblock* u kojem unutar jednostrukih navodnika možemo pohraniti shemu baze podataka kao na slici 3.4.



Slika 3.4: LogiQL REPL, pohrana sheme baze podataka

Sljedeći korak je dodavanje činjenica, odnosno punjenje baze podataka. Dodavanje činjenica vrši se tzv. **delta pravilima** oblika +č*injenica*. Znak "+" znači dodavanje činjenice, dok za ažuriranje/dodavanje i brisanje koristimo znakove "^" i "−". Spomenuta delta pravila pišemo unutar jednostrukih navodnika naredbom *exec*. Entitete dodajemo zajedno sa njihovim *refmode* predikatima, dok ostale činjenice dodajemo pravilima kao na slici 3.5.

```
3-47 /> exec '
     ..  +Zemlja(z), +imaKodZemlje(z,"AU").
     ..  +Zemlja(z), +imaKodZemlje(z,"CA").
     ..  +Zemlja(z), +imaKodZemlje(z,"FR").
     ..  +Zemlja(z), +imaKodZemlje(z,"US").
     ..  +Zemlja(z), +imaKodZemlje(z,"VA").
     ..
     ..  +Jezik(l), +imaNazivJezika(l,"Engleski").
     ..  +Jezik(l), +imaNazivJezika(l,"Francuski").
     ..  +Jezik(l), +imaNazivJezika(l,"Talijanski").
     ..  +Jezik(l), +imaNazivJezika(l,"Akadijski").
     ..
     ..  +jeVelika(z) <- imaKodZemlje(z, "AU").
     ..  +jeVelika(z) <- imaKodZemlje(z, "CA").
     ..  +jeVelika(z) <- imaKodZemlje(z, "FR").
     ..  +jeVelika(z) <- imaKodZemlje(z, "US").
     ..
     ..  +sluzbenoKoristiJezik(z,l) <- imaKodZemlje(z,"AU"), imaNazivJezika(l,"Engleski").
     ..  +sluzbenoKoristiJezik(z,l) <- imaKodZemlje(z,"CA"), imaNazivJezika(l,"Engleski").
     ..  +sluzbenoKoristiJezik(z,l) <- imaKodZemlje(z,"CA"), imaNazivJezika(l,"Francuski").
     ..  +sluzbenoKoristiJezik(z,l) <- imaKodZemlje(z,"US"), imaNazivJezika(l,"Engleski").
     ..  +sluzbenoKoristiJezik(z,l) <- imaKodZemlje(z,"FR"), imaNazivJezika(l,"Francuski").
     ..  +sluzbenoKoristiJezik(z,l) <- imaKodZemlje(z,"VA"), imaNazivJezika(l,"Talijanski").
     ..  '
3-47 />
```

Slika 3.5: LogiQL REPL, pohrana činjenica

Naredbom *print* možemo provjeriti sadržaj predikata kao na slici 3.6

```
3-58 /> print Jezik
     =>
```

10000000000	Akadijski
10000000002	Francuski
10000000003	Engleski
10000000013	Talijanski

```
3-58 /> print jeVelika
     =>
```

10000000004	CA
10000000005	AU
10000000006	US
10000000007	FR

Slika 3.6: LogiQL REPL, ispis sadržaja predikata

Brojevi prikazani na slici 3.6 su interni identifikatori koje REPL alat automatski generira. Upiti se postavaljaju kao pravila unutar jednostrukih navodnika naredbom *query*. Tako primjerice ukoliko bi nas zanimalo koje su to velike zemlje koje koriste službeno francuski jezik postavili bismo upit kao na slici 3.7

```
3-58 /> query '_(z)<-jeVelika(z),sluzbenoKoristiJezik(z,l),imaNazivJezika(l,"Francuski").'
     =>
```

10000000004	CA
10000000007	FR

Slika 3.7: LogiQL REPL, primjer upita

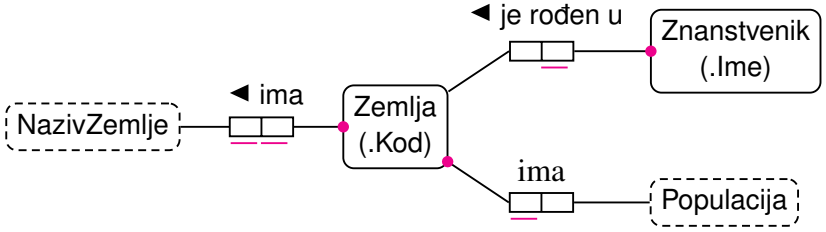
Sa slike 3.7 možemo vidjeti da upit pišemo kao pravilo koje u svojoj glavi ima anonimni predikat $\_(l)$ kako bi prikazao rezultat u tablici. Primijetimo kako primjerice korištenjem iste varijable z u više uvjeta dobivamo ekvivalent relacijskoj operaciji *join* na vrlo koncizan način. Za kraj pogledajmo upit koji se oslanja na pretpostavku zatvorenog svijeta. Pretpostavimo da želimo prikazati zemlje koje nisu velike. Tada bismo postavili upit kao na slici 3.8. Primijetimo da ovdje koristimo negaciju koja se izražava znakom "!". Neuspjeh da se pronađe Vatikan (VA) u listi velikih zemalja shvaća se kao da Vatikan nije velika zemlja (negacija kao neuspjeh) pa je stoga Vatikan vraćen kao rezultat.

```
3-58 /> query '_(z)<-Zemlja(z),!jeVelika(z).'
     =>
```

10000000001	VA

Slika 3.8: LogiQL REPL, primjer upita

Sljedeći malo složeniji primjer naglašava potrebu za jednom vrlo važnom vrstom predikata podržanom u *LogiQL − u*, to su *funkcijski predikati*.



Slika 3.9: Object Role Modeling dijagram

ORM dijagram na slici 3.9 uključuje tri binarna činjenična tipa.
Jednostavno ograničenje u svrhu osiguravanja jedinstvenosti na svakoj ulozi činjeničnog tipa "Zemlja ima NazivZemlje" (dvije crte jedna do druge) ukazuje da je to jedan naprema jedan odnos. Odnosno, svaka zemlja ima najviše jedno ime i svaki naziv zemlje se odnosi na najviše jednu zemlju. Ograničenje u svrhu osiguravanja jedinstvenosti na ulogu znanstvenika činjeničnog tipa "Znanstvenik je rođen u Zemlji" ukazuje da je svaki znanstvenik rođen u najviše jednoj zemlji, dok nedostatak ograničenja u svrhu osiguravanja jedinstvenosti na drugoj ulozi ukazuje na to da neke zemlje mogu biti rodno mjesto više od jednog znanstvenika. Stoga činjenični tip "Znanstvenik je rođen u Zemlji" ima mnogo naprema jedan odnos. Slično, činjenični tip "Zemlja ima Populaciju" također ima mnogo naprema jedan odnos. Strelice pored imena predikata označavaju smjer čitanja, dok točke na tri činjenična tipa označuju obavezna članstva. Odnosno, svaka zemlja ima naziv i populaciju, te je svaki znanstvenik rođen u nekoj zemlji.

$$\begin{aligned} &\textbf{Zemlja}\ (\ \underline{\underline{kodZemlje}},\ \underline{nazivZemlje},\ populacija\ ) \\ &\textbf{Znanstvenik}\ (\ \underline{imeZnanstvenika},\ kodZemljeRođenja\ ) \end{aligned} \tag{3.12}$$

Relacijska shema 3.12 prikazuje ograničenje u svrhu osiguravanja jedinstvenosti za *Zemlju*, gdje je dvostrukim crtama podcrtan primarni ključ te je jednostrukom crtom podcrtan sekundarni ključ. U *LogiQL − u* referentna shema za *Zemlju* i *Znanstvenika* može se deklarirati na isti način kao i primjeru ranije:

$$Zemlja(z),\ imaKodZemlje(z : kz) \text{ -> } string(kz).$$
$$Znanstvenik(zn),\ imaImeZnanstvenika(zn : izn) \text{ -> } string(izn).$$

Činjenični tipovi "Zemlja ima NazivZemlje", "Znanstvenik je rođen u Zemlji" i "Zemlja ima Populaciju" su funkcijski činjenični tipovi. Jer primjerice, naziv zemlje i populacija oboje ovise funkcijski o zemlji, dok je zemlja rođenja funkcija od znanstvenika. U

LogiQL – u funkcijski činjenični tipovi se mogu deklarirati korištenjem uglatih zagrada. Argumenti koji funkcijski određuju završni argument nalaze se u uglatim zagradama, iza kojih slijedi operator "=", te završni argument. Činjenične tipove možemo deklarirati na sljedeći način:

$$\begin{aligned} &zemljaRodjenjaOd[s] = z \text{ -> } Znanstvenik(s),\ Zemlja(z). \\ &nazivZemljeOd[z] = nz \text{ -> } Zemlja(z),\ string(nz). \\ &populacijaOd[z] = n \text{ -> } Zemlja(z),\ int(n). \end{aligned} \tag{3.13}$$

Činjenične predikate smo mogli alternativno deklarirati koristeći i obične zagrade koje koristimo za deklaraciju ne funkcijskih činjeničnih tipova, no notacija uglatih zagrada je učinkovitija budući da ona deklarira ne samo tipove argumenata već i funkcijsku prirodu predikata. Kada bismo koristili predikate sa običnim zagradama tada bismo morali deklarirati funkcijsku prirodu predikata u posebnom ograničenju. Primjerice predikat "$zemljaRodjenjaOd[s] = z$ -> $Znanstvenik(s)$, $Zemlja(z)$." ekvivalentan je kombinaciji formula 3.14.

$$\begin{aligned} &jeRodjenU(s,z) \text{ -> } Znanstvenik(s),\ Zemlja(z). \\ &jeRodjenU(s,z1),\ jeRodjenU(s,z2) \text{ -> } z1 = z2. \end{aligned} \tag{3.14}$$

Ako se radi o jedan naprama jedan predikatu, tada notacija uglatih zagrada bilježi samo jedan od njegovih ograničenja u svrhu osiguravanja jedinstvenosti. Drugo ograničenje u svrhu osiguravanja jedinstvenosti moramo deklarirati zasebno. Primjerice da bi osigurali da se svaki naziv zemlje primjenjuje na samo jednu zemlju morali bismo dodati sljedeće ograničenje:

$$nazivZemljeOd[z1] = nz,\ nazivZemljeOd[z2] = nz \text{ -> } z1 = z2.$$

Tri ograničenja koja osiguravaju obavezna članstva dana su formulama 3.15.

$$\begin{aligned} &Zemlja(z) \text{ -> } nazivZemljeOd[z] = \_. \\ &Zemlja(z) \text{ -> } populacijaOd[z] = \_. \\ &Znanstvenik(s) \text{ -> } zemljaRodjenjaOd[s] = \_. \end{aligned} \tag{3.15}$$

```
6-29 /> addblock '
     .. Zemlja(z), imaKodZemlje(z:kz) -> string(kz).
     .. Znanstvenik(s), imaImeZnanstvenika(s:sn) -> string(sn).
     .. zemljaRodjenjaOd[s] = z -> Znanstvenik(s), Zemlja(z).
     .. nazivZemljeOd[z] = nz -> Zemlja(z), string(nz).
     .. populacijaOd[z] = n -> Zemlja(z), int(n).
     .. nazivZemljeOd[z1] = nz, nazivZemljeOd[z2] = nz -> z1 = z2.
     .. Zemlja(z) -> nazivZemljeOd[z] = _.
     .. Zemlja(z) -> populacijaOd[z] = _.
     .. Znanstvenik(s) -> zemljaRodjenjaOd[s] = _.
     .. '
     => 
     Successfully added block 'block_1Z331FIK'
6-29 /> exec '
     .. +Zemlja(z), +imaKodZemlje(z,"AU").
     .. +Zemlja(z), +imaKodZemlje(z,"DE").
     .. +Zemlja(z), +imaKodZemlje(z,"IT").
     .. 
     .. +Znanstvenik(s), +imaImeZnanstvenika(s,"Albert Einstein").
     .. +Znanstvenik(s), +imaImeZnanstvenika(s,"Fibonacci").
     .. +Znanstvenik(s), +imaImeZnanstvenika(s,"Gottfried Leibniz").
     .. 
     .. +zemljaRodjenjaOd[s]=z <- imaImeZnanstvenika(s,"Albert Einstein"), imaKodZemlje(z,"DE").
     .. +zemljaRodjenjaOd[s]=z <- imaImeZnanstvenika(s,"Gottfried Leibniz"), imaKodZemlje(z,"DE").
     .. +zemljaRodjenjaOd[s]=z <- imaImeZnanstvenika(s,"Fibonacci"), imaKodZemlje(z,"IT").
     .. 
     .. +nazivZemljeOd[z] = "Australija" <- imaKodZemlje(z,"AU").
     .. +nazivZemljeOd[z] = "Njemacka" <- imaKodZemlje(z,"DE").
     .. +nazivZemljeOd[z] = "Italija" <- imaKodZemlje(z,"IT").
     .. 
     .. +populacijaOd[z] = 23583600 <- imaKodZemlje(z,"AU").
     .. +populacijaOd[z] = 80716000 <- imaKodZemlje(z,"DE").
     .. +populacijaOd[z] = 60762320 <- imaKodZemlje(z,"IT").
     .. '
```

Slika 3.10: LogiQL REPL, pohrana sheme i činjenica

Pohranjivanje sheme baze podataka i dodavanje činjenica u bazu prikazano na slici 3.10. Nakon što smo pohranili shemu baze podataka i dodali činjenice u bazu možemo postavljati razne upite, primjerice ukoliko bismo željeli prikazati sve znanstvenike, te imena i populacije zemlja gdje su rođeni pri čemu je populacija tih zemlja veća od 70 000 000 tada bismo postavili upit kao na slici 3.11

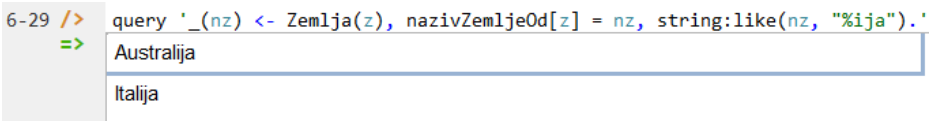
```
query '_(s, nz, p) <- zemljaRodjenjaOd[s]=z, nazivZemljeOd[z]=nz, populacijaOd[z]=p, populacijaOd[z]>70000000.'
```

10000000005	Albert Einstein	Njemacka	80716000
10000000007	Gottfried Leibniz	Njemacka	80716000

Slika 3.11: LogiQL REPL, pohrana sheme i činjenica

LogiQL podržava i razne funkcije za manipulaciju i podudaranje stringova. Tako primjerice funkcija $string : like(str, uzorak)$, gdje je *str* string, a *uzorak* koji stavljamo u navodnike je uzorak koji želimo prepoznati u stringu i koji može sadržavati znakove ”%”

za niz nula ili više znakova i "_" za točno jedan bilo koji znak. Primjerice ako bi nas zanimalo koje su to zemlje koje u svom nazivu završavaju sa *ija* tada bismo postavili upit kao na slici 3.12.



Slika 3.12: LogiQL REPL, pohrana sheme i činjenica

U ovom odjeljku dan je uvod u *LogiQL* te su objašnjeni tek neki osnovni koncepti koji će biti važni za razumijevanje sljedećeg odjeljka. Sljedeći odjeljak obrađuje složeniji primjer baze podataka, gdje će biti predstavljeni neki složeniji koncepti, kao što su podtipovi, rekurzivna pravila, razne agregacije i složeniji upiti. Za detaljniji uvod, primjere i razne druge funkcionalnosti jezika *LogiQL* vidi [6] i [4].

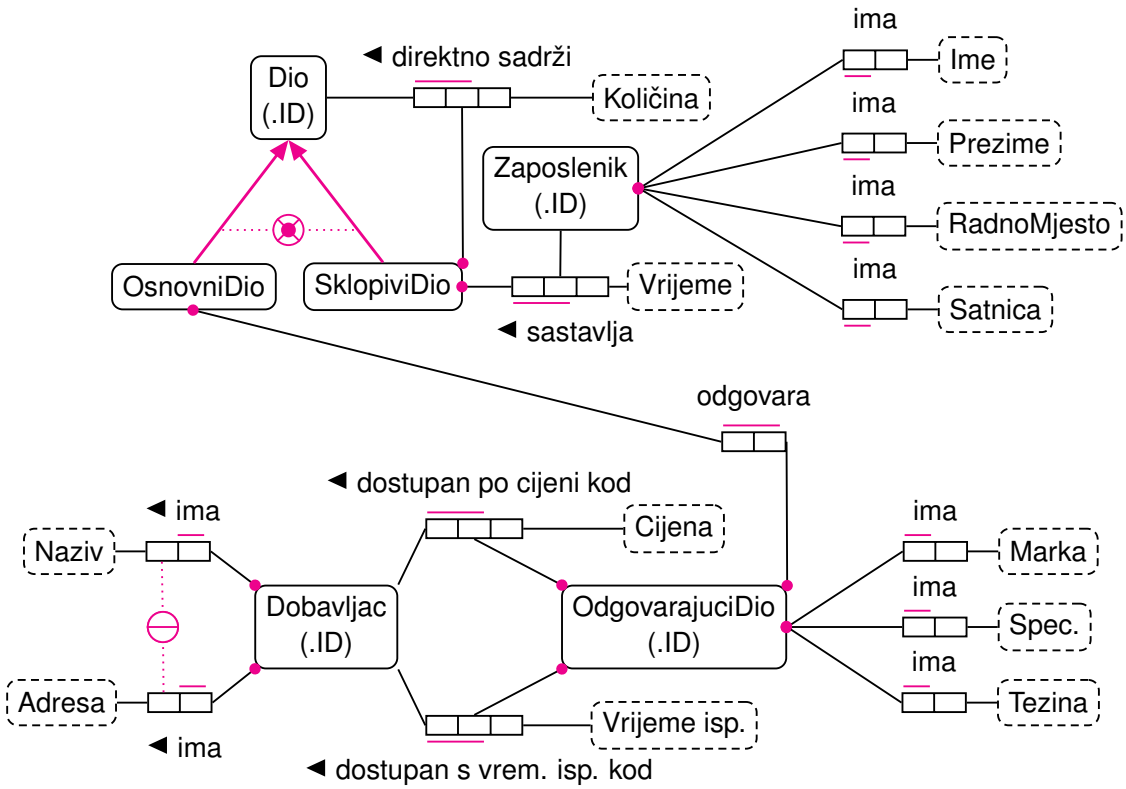
3.2 Konstrukcijska sastavnica

U ovom odjeljku realiziramo bazu podataka za konstrukcijsku sastavnicu materijala (eng. Bill of materials) koja će predstavljati jedan vrlo dobar primjer na kojem ćemo vidjeti potrebu za deduktivnom bazom podataka i njene prednosti nad relacijskom bazom podataka.

Konstrukcijska sastavnica ima znatnu primjenu u raznim prerađivačkim industrijama. Riječ je o bazi podataka koja sadrži sklopive dijelove koji se mogu sastojati djelomično od drugih sklopivih poddijelova koji se ponovo mogu sastojati od sklopivih poddijelova itd. sve dok se cijeli sastav ne razgradi u elementarne dijelove (koji se moraju nabaviti, primjerice od nekog dobavljača). Iz same formulacije problema možemo uočiti da će se javljati potreba za upitima koji koriste rekurzije. Primjerice, može nas zanimati koliko je elementarnih dijelova potrebno za neki sklopivi dio, ili u koje sve sklopive dijelove direktno ili indirektno ulazi neki sklopivi dio.

U današnjici postoje razni softverski paketi za konstrukcijske sastavnice sa raznim rekurzivim strukturama kojima manipulira i upravlja aplikacijski kod pisan u nekom programskom jeziku treće generacije (C, C++, C#, Java...). Prednost deduktivne baze podataka je ta da ona može preuzeti manipulaciju i upravljanje rekurzijama na sebe te time rasteretiti aplikaciju i komunikacijski kanal sa samom aplikacijom.

U nastavku realiziramo deduktivnu bazu podataka (odnosno konstrukcijsku sastavnicu) za proizvodnju bicikala. Kako bismo mogli definirati shemu deduktivne baze podataka i bolje razumjeti potrebne funkcionalnosti takve baze proučimo prvo Object Role Modeling (ORM) dijagram na slici 3.13. Na njemu najprije možemo uočiti entitet *Dio* koji je reprezentiran svojim identifikacijskim brojem (*ID*-om), te dva entiteta *OsnovniDio* i *SklopiviDio* koji predstavljaju podtipove entiteta *Dio* što je označeno strelicama. Znak zaokružene prekrižene točke koji se nalazi između strelica sugerira da podtipovi tvore particiju. Dakle očito ćemo dijelove moći podijeliti na dijelove koje sklapamo (*SklopiviDio*) i neke bazične dijelove koje ne možemo sklapati, već ih nabavljamo (*OsnovniDio*).



Slika 3.13: Object Role Modeling dijagram za konstrukcijsku sastavnicu

Podtipove i njihove particije možemo prikazati u *LogiQL − u* sljedećim formulama:

$$Dio(p),\ dio\_ID(p : ID) \text{ -> } string(ID). \tag{3.16a}$$
$$OsnovniDio(p) \text{ -> } Dio(p). \tag{3.16b}$$
$$SklopiviDio(p) \text{ -> } Dio(p). \tag{3.16c}$$
$$lang : isEntity[`OsnovniDio] = true. \tag{3.16d}$$
$$lang : isEntity[`SklopiviDio] = true. \tag{3.16e}$$
$$Dio(p) \text{ -> } OsnovniDio(p)\ ;\ SklopiviDio(p). \tag{3.16f}$$
$$SklopiviDio(p) \text{ -> } !OsnovniDio(p). \tag{3.16g}$$

Formula 3.16a deklarira *Dio* kao entitet zajedno sa njegovim *refmode* predikatom. Formule 3.16b i 3.16c deklariraju *OsnovniDio* i *SklopiviDio* kao podtipove dijela. Odnosno svaki *OsnovniDio* je *Dio* i svaki *SklopiviDio* je *Dio*. Formule 3.16d i 3.16e. deklariraju *OsnovniDio* i *SklopiviDio* kao entitete. Formule 3.16f i 3.16g odnose se na deklaraciju particije podtipova, odnosno *Dio* može biti *OsnovniDio* ili *SklopiviDio* i ništa drugo, te *SklopiviDio* ne može biti *OsnovniDio*.

Ternarni činjenični tip "SklopiviDio direktno sadrži Dio u određenoj Količini" koristimo da bismo zabilježili koje sve dijelove i u kojim količinama direktno sadrži neki SklopiviDio. To svakako nisu svi dijelovi koji ulaze u sastav nekog dijela već samo oni dijelovi (sklopivi ili osnovni) koje neki *SklopiviDio* direktno sadrži. Dijelove koji indirektno ulaze u sastav nekog sklopivog dijela nećemo pohranjivati već ćemo ih izvoditi rekurzivnim pravilima kao što ćemo vidjeti kasnije. Također ćemo pretpostaviti restrikciju da niti jedan *Dio* neće direktno ili indirektno sadržavati samog sebe. Činjenični tip "SklopiviDio direktno sadrži Dio u određenoj Količini" zajedno sa obaveznim članstvom sklopivog dijela u *LogiQL − u* definiramo na sljedeći način:

$$direktnoSadrzi[p1, p2] = kol \text{ -> } SklopiviDio(p1),\ Dio(p2),\ int(kol). \tag{3.17a}$$
$$SklopiviDio(s) \text{ -> } direktnoSadrzi[s, \_] = \_. \tag{3.17b}$$

Formula 3.17a definira činjenični tip "SklopiviDio direktno sadrži Dio u određenoj Količini" kao funkcijski predikat sa dva funkcijska argumenta (ključa) $p1$ i $p2$ koji funkcijski određuju završni argument *kol* koji predstavlja količinu dijela $p2$ sadržanog u dijelu $p1$. Formula 3.17b odnosi se na obavezno članstvo entiteta *SklopiviDio* u funkcijskom predikatu *direktnoSadrzi*.

Entitet *Zaposlenik* koji reprezentiramo njegovim *ID*-om nastupa sa obaveznim članstvom u četiri binarna činjenična tipa "Zaposlenik ima Ime", "Zaposlenik ima Prezime", "Zaposlenik ima RadnoMjesto" te "Zaposlenik ima Satnicu". Zaposlenik također

nastupa bez obaveznog članstva u ternarnom predikatu ”Zaposlenik sastavlja SklopiviDio u određenom Vremenu”. Također možemo vidjeti i da jedan zaposlenik može sastavljati više sklopivih dijelova te da više zaposlenika mogu sastavljati jedan sklopivi dio. Spomenute činjenične tipove i entitet *Zaposlenik* u *LogiQL − u* prikazujemo sljedećim formulama:

$$Zaposlenik(e),\ zaposlenik\_ID(e : ID) \text{ -> } string(ID). \tag{3.18a}$$

$$zaposlenik\_ime[e] = ime \text{ -> } Zaposlenik(e),\ string(ime). \tag{3.18b}$$

$$zaposlenik\_prezime[e] = prezime \text{ -> } Zaposlenik(e),\ string(prezime). \tag{3.18c}$$

$$zaposlenik\_nazivRadnogMjesta[e] = RadnoMj \text{ -> } Zaposlenik(e),\ string(radnoMj). \tag{3.18d}$$

$$zaposlenik\_satnica[e] = satnica \text{ -> } Zaposlenik(e),\ decimal(satnica). \tag{3.18e}$$

$$Zaposlenik(e) \text{ -> } zaposlenik\_ime[e] = \_. \tag{3.18f}$$

$$Zaposlenik(e) \text{ -> } zaposlenik\_prezime[e] = \_. \tag{3.18g}$$

$$Zaposlenik(e) \text{ -> } zaposlenik\_nazivRadnogMjesta[e] = \_. \tag{3.18h}$$

$$Zaposlenik(e) \text{ -> } zaposlenik\_satnica[e] = \_. \tag{3.18i}$$

$$vrijemeSastavljanja[e, p] = sati \text{ -> } Zaposlenik(e),\ SklopiviDio(p),\ decimal(sati). \tag{3.18j}$$

$$SklopiviDio(s) \text{ -> } vrijemeSastavljanja[\_, s] = \_. \tag{3.18k}$$

Formula 3.18a deklarira *Zaposlenika* kao entitet zajedno sa njegovim *refmode* predikatom. Činjenični tipovi ”Zaposlenik ima Ime”, ”Zaposlenik ima Prezime”, ”Zaposlenik ima RadnoMjesto” te ”Zaposlenik ima Satnicu” definirani su redom formulama 3.18b, 3.18c, 3.18d i 3.18e. dok je obavezno članstvo *Zaposlenika* definirano formulama 3.18f, 3.18g, 3.18h i 3.18i. Formula 3.18j definira činjenični tip ”Zaposlenik sastavlja SklopiviDio u određenom Vremenu”. Obavezno članstvo sklopivog dijela definirano je formulom 3.18k.

Sljedeća dva entiteta koja možemo uočiti na slici 3.13 su *OdgovarajuciDio* i *Dobavljac*. Entitet *OdgovarajuciDio* predstavlja dijelove raznih težina, marki i specifikacija koji odgovaraju osnovnim dijelovima i dostupni su kod jednog ili više dobavljača po raznim cijenama i vremenima dostave. Primjerice, *OsnovniDio* bio bi sjedalo za određeni tip bicikla, kojem bi odgovarala sjedala raznih marki, težina i specifikacija dostupna kod dobavljača. Jednom osnovnom dijelu može očito odgovarati više odgovarajućih dijelova, te jedan *OdgovarajuciDio* može odgovarati više osnovnih dijelova (primjerice može postojati neki univerzalni odgovarajući dio koji odgovara raznim osnovnim dijelovima). Binarni činjenični tip ”Osnovnom dijelu odgovara OdgovarajuciDio” i entitet *OdgovarajuciDio* koji još nastupa i u tri funkcijska predikata ”OdgovarajuciDio ima Marku”, ”OdgovarajuciDio ima Specifikaciju” i ”OdgovarajuciDio ima Težinu” te obavezna članstva entiteta

OsnovniDio i *Odgovara juciDio* možemo u *LogiQL − u* prikazati sljedećim formulama:

Odgovara juciDio(*od*), *odgDio_ID*(*od* : *odID*) -> *string*(*odID*).
odgovara juciDio_marka[*od*] = *marka* -> *Odgovara juciDio*(*od*), *string*(*marka*).
odgovara juciDio_spec[*od*] = *spec* -> *Odgovara juciDio*(*od*), *string*(*spec*).
odgovara juciDio_tezina[*od*] = *tezina* -> *Odgovara juciDio*(*od*), *int*(*tezina*).
Odgovara juciDio(*od*) -> *odgovara juciDio_marka*[*od*] = _.
Odgovara juciDio(*od*) -> *odgovara juciDio_spec*[*od*] = _.
Odgovara juciDio(*od*) -> *odgovara juciDio_tezina*[*od*] = _.
odgovara juciDioZaOsnovniDio(*odgDio*, *osnDio*) -> *Odgovara juciDio*(*odgDio*),
OsnovniDio(*osnDio*).
OsnovniDio(*o*) -> *odgovara juciDioZaOsnovniDio*(_, *o*).
Odgovara juciDio(*od*) -> *odgovara juciDioZaOsnovniDio*(*od*, _).

Entitet *Dobavljac* sudjeluje u dva binarna činjenična tipa ”Dobavljač ima Naziv” i ”Dobavljač ima Adresu”, gdje zaokruženi znak jedinstvenosti između tih činjeničnih tipova označava ograničenje u svrhu ostvarivanja jedinstvenosti (za svaki naziv i adresu najviše jedan dobavljač može imati taj naziv i tu adresu). Entitet *Dobavljac* također nastupa zajedno sa entitetom *Odgovara juciDio* u ternarnim činjeničnim predikatima ”Odgovarajući dio je dostupan po cijeni kod Dobavljača” i ”Odgovarajući dio je dostupan s vremenom

isporuke kod Dobavljača". U *LogiQL – u* to možemo prikazati sljedećim formulama:

$$Dobavljac(d),\ dobavljac\_ID(d : ID) \text{ -> } string(ID). \tag{3.19a}$$

$$dobavljac\_naziv[d] = naziv \text{ -> } Dobavljac(d),\ string(naziv). \tag{3.19b}$$

$$dobavljac\_addresa[d] = adresa \text{ -> } Dobavljac(d),\ string(adresa). \tag{3.19c}$$

$$Dobavljac(d) \text{ -> } dobavljac\_naziv[d] = \_. \tag{3.19d}$$

$$Dobavljac(d) \text{ -> } dobavljac\_addresa[d] = \_. \tag{3.19e}$$

$$dobavljac\_naziv[d1] = n,\ dobavljac\_naziv[d2] = n,\ dobavljac\_addresa[d1] = a,\ dobavljac\_addresa[d2] = a \text{ -> } d1 = d2. \tag{3.19f}$$

$$odgDioKodDobav\_cijena[doba, odgDio] = cijena \text{ -> } Dobavljac(doba),\ OdgovarajuciDio(odgDio),\ decimal(cijena). \tag{3.19g}$$

$$odgDioKodDobav\_vrIsp[doba, odgDio] = dan \text{ -> } Dobavljac(doba),\ OdgovarajuciDio(odgDio),\ int(dan). \tag{3.19h}$$

$$OdgovarajuciDio(od) \text{ -> } odgDioKodDobav\_cijena[\_, od] = \_. \tag{3.19i}$$

$$OdgovarajuciDio(od) \text{ -> } odgDioKodDobav\_vrIsp[\_, od] = \_. \tag{3.19j}$$

$$odgDioKodDobav\_cijena[d, od] = \_ \text{ -> } odgDioKodDobav\_vrIsp[d, od] = \_. \tag{3.19k}$$

$$odgDioKodDobav\_vrIsp[d, od] = \_ \text{ -> } odgDioKodDobav\_cijena[d, od] = \_. \tag{3.19l}$$

Formula 3.19a deklarira entitet *Dobavljac* zajedno sa njegovm refmode predikatom. Činjenični tipovi "Dobavljač ima Naziv" i "Dobavljač ima Adresu" definirani su redom formulama 3.19b i 3.19c. Obavezna članstva entiteta *Dobavljac* definirana su formulama 3.19d i 3.19e. Ograničenje u svrhu očuvanja integriteta kako bi osigurali da za svaki naziv i adresu najviše jedan dobavljač može imati taj naziv i tu adresu definirano je formulom 3.19f. Ternarni činjenični predikati "Odgovarajući dio je dostupan po cijeni kod Dobavljača" i "Odgovarajući dio je dostupan s vremenom isporuke kod Dobavljača" definirani su redom formulama 3.19g i 3.19h, dok je obavezno članstvo entiteta *OdgovarajuciDio* u navedenim ternarnim predikatima definirano redom formulama 3.19i i 3.19j. Ograničenje da svaki odgovarajući dio kod istog dobavljača mora imati cijenu i vrijeme isporuke definiramo formulama 3.19k i 3.19l.

Time smo završili sa deklaracijom sheme baze podataka za konstrukcijsku sastavnicu, sada možemo raznim rekurzivnim pravilima izvađati neke dodatne činjenice koje nemamo direktno pohranjene u bazi. Primjerice može nas zanimati koji sve poddijelovi i u kojoj količini ulaze direktno i indirektno u sastav nekog dijela. Primjerice, na slici 3.14 možemo vidjeti da se kotač direktno sastoji od zračnice, gume, felge i ventila. No vidimo i da je felga sklopivi dio koji ima poddijelove od kojih su neki također sklopivi.

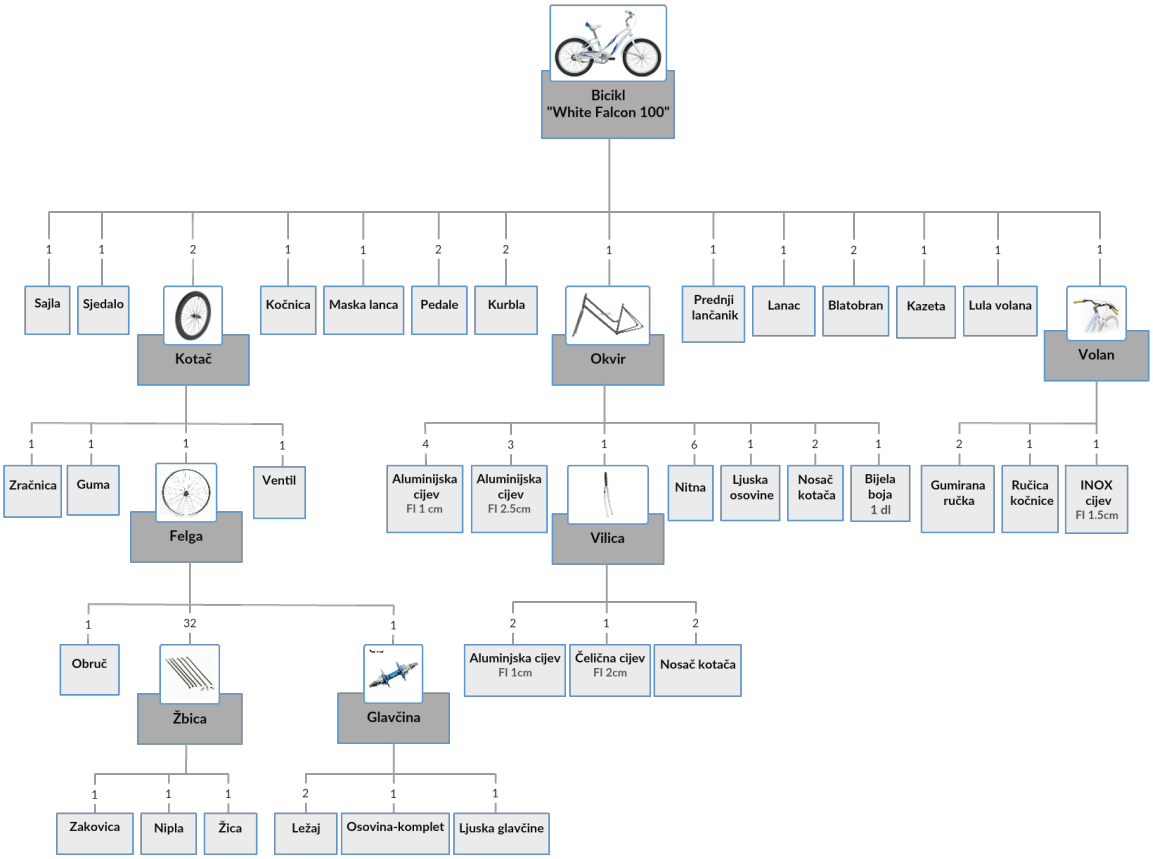
Da bismo dobili sve poddijelove nekog sklopivog dijela u *LogiQL−u* dodajemo sljedeća pravila:

$$sviPoddijelovi[put, dio, poddio] = direktnoSadrzi[dio, poddio] <- \\ put = dio\_ID[dio]. \tag{3.20a}$$

$$sviPoddijelovi[put, dio, poddio] = kvant <- kvant = \\ sviPoddijelovi[put1, dio, poddio1] * direktnoSadrzi[poddio1, poddio], \\ string(put),\ put = put1 + " - " + dio\_ID[poddio1]. \tag{3.20b}$$

$$!sviPoddijelovi[\_, dio, dio] = \_. \tag{3.20c}$$

Pravilo 3.20a definira rekurzivni predikat, te predstavlja bazni slučaj u induktivnoj definiciji rekurzivnog predikata. Pravilo 3.20c predstavlja ograničenje da neki sklopivi dio ne može sadržavati samog sebe. Pravilo 3.20a se obično naziva *pravilo izlaska*. Pravilo 3.20b definira tranzitivno zatvorenje svih poddijelova, to pravilo je rekurzivno jer se glava predikata *sviPoddijelovi* također nalazi i u tijelu pravila. Umnoškom vrijednosti funkcijskih predikata *sviPoddijelovi* i *direktnoSadrzi* dobivamo ukupnu količinu dijelova koje sadrži neki sklopivi dio. String *put* u prvom argumentu funkcijskog predikata *sviPoddijelovi* služi kako bi razlikovali iste dijelove koji u istoj količini dolaze u različitim sklopivim dijelovima nekog proizvoda (sklopivog dijela). U stringu *put*, akumuliraju se putovi od sklopivog dijela do dijela koji pridonosi njegovom sastavu. Kada ne bi bilježili *put* tada bi semi-naivni algoritam maknuo sve duplikate, pa kada bi isti dio bio potreban u istoj količini sa dva drugačija puta prikazala bi se samo jedna torka.



Slika 3.14: Konstrukcijska sastavnica bicikla

Slika 3.15 prikazuje upit te ispis svih dijelova bicikla koristeći pravila 3.20a i 3.20b. Pravilo koje uključuje agregatnu funkciju *min* daje nam najjeftiniju cijenu odgovarajućeg dijela za svaki osnovni dio.

$$\begin{aligned} &najjeftiniji[osnDio] = cijena \text{ <- } agg \text{ << } cijena = min(cijena1) \text{ >>} \\ &\quad odgovarajuciDioZaOsnovniDio(odgDio, osnDio), \\ &\quad odgDioKodDobav\_cijena[\_, odgDio] = cijena1. \end{aligned} \tag{3.21a}$$

Agregatne funkcije (kao što su *total*, *count*, *max* i *min*) pozivaju se koristeći specijalnu *agg* << ... >> sintaksu, pa se pravila koja koriste agregacije nazivaju *agregatna pravila*. Unutar dvostrukih šiljastih zagrada << ... >> varijabli se dodjeljuje rezultat primjene agregatne funkcije na činjenice koje zadovoljavaju filtrirajući uvjet (formulu) koji slijedi iza zatvorenih šiljastih zagrada >>.

```
query '_(s, "White_Falcon_100", p, kol)<- sviPoddijelovi[s,d,pd]=kol, dio_ID[d]="White_Falcon_100", dio_ID[pd]=p.'
```

White_Falcon_100	White_Falcon_100	Blatobran	2
White_Falcon_100	White_Falcon_100	Kazeta_WF	1
White_Falcon_100	White_Falcon_100	Kocnica_WF	1
White_Falcon_100	White_Falcon_100	Kotac_WF	2
White_Falcon_100	White_Falcon_100	Kurbla_WF	2
White_Falcon_100	White_Falcon_100	Lanac_WF	1
White_Falcon_100	White_Falcon_100	Lula_volana...	1
White_Falcon_100	White_Falcon_100	Maska_lanca	1
White_Falcon_100	White_Falcon_100	Okvir_WF	1
White_Falcon_100	White_Falcon_100	Pedale_WF	2
White_Falcon_100	White_Falcon_100	Prednji_lanc...	1
White_Falcon_100	White_Falcon_100	Sajla_WF	1
White_Falcon_100	White_Falcon_100	Sjedalo_WF	1
White_Falcon_100	White_Falcon_100	Volan_WF	1
White_Falcon_100 - Kotac_WF	White_Falcon_100	Felga_WF	2
White_Falcon_100 - Kotac_WF	White_Falcon_100	Guma_WF	2
White_Falcon_100 - Kotac_WF	White_Falcon_100	Ventil_WF	2
White_Falcon_100 - Kotac_WF	White_Falcon_100	Zracnica_WF	2
White_Falcon_100 - Kotac_WF - Felga_WF	White_Falcon_100	Glavcina_WF	2
White_Falcon_100 - Kotac_WF - Felga_WF	White_Falcon_100	Obruc_WF	2
White_Falcon_100 - Kotac_WF - Felga_WF	White_Falcon_100	Zbica_WF	64
White_Falcon_100 - Kotac_WF - Felga_WF - Glavcina_WF	White_Falcon_100	Lezaj_WF	4
White_Falcon_100 - Kotac_WF - Felga_WF - Glavcina_WF	White_Falcon_100	Ljuska_glavc...	2
White_Falcon_100 - Kotac_WF - Felga_WF - Glavcina_WF	White_Falcon_100	Osovina_ko...	2
White_Falcon_100 - Kotac_WF - Felga_WF - Zbica_WF	White_Falcon_100	Nipla_WF	64
White_Falcon_100 - Kotac_WF - Felga_WF - Zbica_WF	White_Falcon_100	Zakovica_WF	64
White_Falcon_100 - Kotac_WF - Felga_WF - Zbica_WF	White_Falcon_100	Zica_WF	64
White_Falcon_100 - Okvir_WF	White_Falcon_100	Alum_cijev_...	4
White_Falcon_100 - Okvir_WF	White_Falcon_100	Alum_cijev_...	3
White_Falcon_100 - Okvir_WF	White_Falcon_100	Bijela_boja_...	1
White_Falcon_100 - Okvir_WF	White_Falcon_100	Ljuska_osovi...	1
White_Falcon_100 - Okvir_WF	White_Falcon_100	Nitna_WF	6
White_Falcon_100 - Okvir_WF	White_Falcon_100	Nosac_kota...	2
White_Falcon_100 - Okvir_WF	White_Falcon_100	Vilica_WF	1
White_Falcon_100 - Okvir_WF - Vilica_WF	White_Falcon_100	Alum_cijev_...	2
White_Falcon_100 - Okvir_WF - Vilica_WF	White_Falcon_100	Celicna_cije...	1
White_Falcon_100 - Okvir_WF - Vilica_WF	White_Falcon_100	Nosac_kota...	2
White_Falcon_100 - Volan_WF	White_Falcon_100	Gumirana_r...	2
White_Falcon_100 - Volan_WF	White_Falcon_100	Inox_cijev_1...	1
White_Falcon_100 - Volan_WF	White_Falcon_100	Rucica_koc...	1

Slika 3.15: Rezultat upita za sve poddijelove bicikla

Koristeći pravila 3.20a, 3.20b i agregatno pravilo 3.21a definiramo pravilo koje će nam dati cijene najjeftinijih odgovarajućih dijelova za osnovne dijelove koji ulaze u sastav nekog

sklopivog dijela te dobavljače kod kojih te najjeftinije dijelove možemo nabaviti.

$$\begin{aligned} najjeftinijiOsnDijeloviNekogDijela[dio, dob, osnDio] = cijena <- \\ sviPoddijelovi[\_, dio, osnDio] = \_, cijena = najjeftiniji[osnDio], \\ odgovarajuciDioZaOsnovniDio(odgDio, osnDio), \\ cijena = odgovarajuciDioKodDobavljaca\_cijena[dob, odgDio]. \end{aligned} \tag{3.22a}$$

```
query '_("White_Falcon_100",od,dbv,c)<-najjeftinijiOsnDijeloviNekogDijela[d,dob,osnDio]=cij,
dio_ID[d]="White_Falcon_100",dobavljac_ID[dob]=dbv ,dio_ID[osnDio]=od, c=decimal:string:convert[cij].'
```

White_Falcon_100	Alum_cijev_10_WF	Dynamic-113	120.00000
White_Falcon_100	Alum_cijev_25_WF	Dynamic-113	180.00000
White_Falcon_100	Bijela_boja_1dl_WF	Dynamic-113	30.00000
White_Falcon_100	Blatobran	Bike_Sport-118	70.00000
White_Falcon_100	Celicna_cijev_20_WF	Dynamic-113	100.00000
White_Falcon_100	Guma_WF	Ciklo-116	100.00000
White_Falcon_100	Gumirana_rucka_WF	DSG-115	15.00000
White_Falcon_100	Inox_cijev_15_WF	Dynamic-113	250.00000
White_Falcon_100	Kazeta_WF	BIZ-117	50.00000
White_Falcon_100	Kocnica_WF	DSG-115	69.50000
White_Falcon_100	Kurbla_WF	Huni-119	60.00000
White_Falcon_100	Lanac_WF	Bike_Sport-118	100.00000
White_Falcon_100	Lezaj_WF	Dynamic-113	15.00000
White_Falcon_100	Ljuska_glavcine_WF	Dynamic-113	15.00000
White_Falcon_100	Ljuska_osovine_WF	Dynamic-113	40.00000
White_Falcon_100	Lula_volana_WF	BIZ-117	150.00000
White_Falcon_100	Maska_lanca	Ciklo-116	50.00000
White_Falcon_100	Nipla_WF	Dynamic-113	3.00000
White_Falcon_100	Nitna_WF	Dynamic-113	5.00000
White_Falcon_100	Nosac_kotaca_prednji_WF	Rog_Joma-120	15.00000
White_Falcon_100	Nosac_kotaca_zadnji_WF	Rog_Joma-120	20.00000
White_Falcon_100	Obruc_WF	DSG-115	70.00000
White_Falcon_100	Osovina_komplet_WF	Dynamic-113	5.00000
White_Falcon_100	Pedale_WF	Rog_Joma-120	59.00000
White_Falcon_100	Prednji_lancanik_WF	Huni-119	169.00000
White_Falcon_100	Rucica_kocnice_WF	DSG-115	40.00000
White_Falcon_100	Sajla_WF	MR_Big_Sport-...	30.00000
White_Falcon_100	Sjedalo_WF	MR_Big_Sport-...	100.00000
White_Falcon_100	Ventil_WF	BIZ-117	10.00000
White_Falcon_100	Zakovica_WF	Dynamic-113	2.00000
White_Falcon_100	Zica_WF	Dynamic-113	3.00000
White_Falcon_100	Zracnica_WF	MR_Big_Sport-112	39.00000

Slika 3.16: Najjeftinije cijene odgovarajućih dijelova za osnovne dijelove bicikla

Sada koristeći pravila 3.22a, 3.20a, i 3.20b, možemo izvesti sljedeća pravila:

$$\begin{aligned} &najjeftinijiOsnCijeneSastavaKumul[put, dio, osnDio] = cijena <- \\ &\quad jedinicnaCijena = najjeftinijiOsnDijeloviNekogDijela[dio, \_, osnDio], \\ &\quad kvant = sviPoddijelovi[put, dio, osnDio], \\ &\quad kvantDecimal = int : decimal : convert[kvant], \\ &\quad cijena = jedinicnaCijena * kvantDecimal. \end{aligned} \tag{3.23a}$$

$$\begin{aligned} &najjeftOsnCijeneSastavaKumulGrupirano[dio, osnDio] = cijena <- \\ &\quad agg << cijena = total(c) >> \\ &\quad najjeftinijiOsnCijeneSastavaKumul[\_, dio, osnDio] = c. \end{aligned} \tag{3.23b}$$

Pravilom 3.23a dobivamo najjeftinije kumulativne cijene odgovarajućih dijelova za osnovne dijelove koji ulaze u sastav nekog sklopivog dijela. Primjerice na slici 3.14 u sastav bicikla ulaze ukupno 64 žbice, pa će cijena jedne žbice biti pomnožena sa 64 kako bismo dobili ukupnu cijenu. *LogiQL* podržava razne operacije konverzija ugrađenih tipova. Da bismo mogli pomnožiti količinu osnovnih dijelova koji ulaze u sastav tipa *int* sa cijenama koje su tipa *decimal*, moramo napraviti konverziju količine u tip *decimal* sa $int : decimal : convert[kvant]$. Neki osnovni dio može pridonijeti sklopivom dijelu preko više sastava, pa ukoliko ne želimo imati ponavljanja takvih dijelova moramo ih grupirati i zbrojiti im cijene koristeći pravilo 3.23b. Primjerice na slici 3.14 aluminijska cijev FI 1cm pridonosi sastavu bicikla preko dva različita sklopiva dijela (vilice i okvira), stoga će ih pravilo 3.23a prikazati zasebno, a pravilo 3.23b će ih grupirati i zbrojiti ima cijene. U pravilu 3.23b agregatna funkcija *total* predstavlja sumu. Slika 3.17 prikazuje upit te ispis najjeftinijih kumulativnih cijena odgovarajućih dijelova za osnovne dijelove bicikla.

Sljedećim pravilima možemo dobiti cijene sastavljanja pojedinih sklopivih dijelova, gdje množenjem satnica zaposlenika i vremenima (pravilo 3.24a) koja su potrebna da bi se pojedini sklopivi dio mogao sastaviti dobivamo upravo informaciju o cijenama sastavljanja svakog sklopivog dijela kojeg sklapa pojedini zaposlenik. Moguće je da više zaposlenika radi na jednom dijelu, stoga možemo zbrojiti cijene takvih dijelova te prikazati cijene sastavljanja pojedinih dijelova pravilom 3.24b.

$$\begin{aligned} &cijenaSastavljanja[zap, dio] = sklopiviDio\_vrijemeSastavljanja[zap, dio] * \\ &\quad zaposlenik\_satnica[zap]. \end{aligned} \tag{3.24a}$$

$$\begin{aligned} &cijenaSastavljanjaGrupirano[dio] = cijena <- agg << cijena = total(c) >> \\ &\quad c = cijenaSastavljanja[\_, dio]. \end{aligned} \tag{3.24b}$$

```
query '_("White_Falcon_100",od,c)<-najjeftOsnCijeneSastavaKumulGrupirano[d,osnDio]=cij,
dio_ID[d]="White_Falcon_100" ,dio_ID[osnDio]=od, c=decimal:string:convert[cij].'
```

White_Falcon_100	Alum_cijev_10_WF	720.00000
White_Falcon_100	Alum_cijev_25_WF	540.00000
White_Falcon_100	Bijela_boja_1dl_WF	30.00000
White_Falcon_100	Blatobran	140.00000
White_Falcon_100	Celicna_cijev_20_WF	100.00000
White_Falcon_100	Guma_WF	200.00000
White_Falcon_100	Gumirana_rucka_WF	30.00000
White_Falcon_100	Inox_cijev_15_WF	250.00000
White_Falcon_100	Kazeta_WF	50.00000
White_Falcon_100	Kocnica_WF	69.50000
White_Falcon_100	Kurbla_WF	120.00000
White_Falcon_100	Lanac_WF	100.00000
White_Falcon_100	Lezaj_WF	60.00000
White_Falcon_100	Ljuska_glavcine_WF	30.00000
White_Falcon_100	Ljuska_osovine_WF	40.00000
White_Falcon_100	Lula_volana_WF	150.00000
White_Falcon_100	Maska_lanca	50.00000
White_Falcon_100	Nipla_WF	192.00000
White_Falcon_100	Nitna_WF	30.00000
White_Falcon_100	Nosac_kotaca_prednji_WF	30.00000
White_Falcon_100	Nosac_kotaca_zadnji_WF	40.00000
White_Falcon_100	Obruc_WF	140.00000
White_Falcon_100	Osovina_komplet_WF	10.00000
White_Falcon_100	Pedale_WF	118.00000
White_Falcon_100	Prednji_lancanik_WF	169.00000
White_Falcon_100	Rucica_kocnice_WF	40.00000
White_Falcon_100	Sajla_WF	30.00000
White_Falcon_100	Sjedalo_WF	100.00000
White_Falcon_100	Ventil_WF	20.00000
White_Falcon_100	Zakovica_WF	128.00000
White_Falcon_100	Zica_WF	192.00000
White_Falcon_100	Zracnica_WF	78.00000

Slika 3.17: Najjeftinije kumulativne cijene odg. dijelova za osnovne dijelove bicikla

Ukupnu cijenu sastava svih sklopivih dijelova zajedno sa ukupnim cijenama najjeftinijih odgovarajućih dijelova za osnovne dijelove koji ulaze u sastav nekog dijela možemo

izvesti sljedećim pravilima:

$$\begin{aligned} &cijenaSastavljanjaSaNajjeftDijelovima[put, dio, dijelovi] = cijene <- \\ &\quad kvant = sviPoddijelovi[put, dio, dijelovi], SklopiviDio(dijelovi), \\ &\quad cijena = cijenaSastavljanjaGrupirano[dijelovi], \\ &\quad kvant1 = int : decimal : convert[kvant], cijene = kvant1 * cijena. \end{aligned} \tag{3.25a}$$

$$\begin{aligned} &cijenaSastavljanjaSaNajjeftDijelovima[put, dio, dijelovi] = cijene <- \\ &\quad najjeftinijiOsnCijeneSastavaKumul[put, dio, dijelovi] = cijene. \end{aligned} \tag{3.25b}$$

Ukoliko bismo željeli prikazati samo ukupne cijene sastavljanja svih sklopivih dijelova za bicikl, bez osnovnih dijelova tada bismo mogli postaviti upit kao na slici 3.18.

```
query '_(put,"White_Falcon_100",sklop_d,c)<-cijenaSastavljanjaSaNajjeftDijelovima[put, dio, d]=cijene,
SklopiviDio(d), dio_ID[dio]="White_Falcon_100", dio_ID[d]=sklop_d, c=decimal:string:convert[cijene].'
```

White_Falcon_100	White_Falcon_100	Kotac_WF	90.00000
White_Falcon_100	White_Falcon_100	Okvir_WF	130.00000
White_Falcon_100	White_Falcon_100	Volan_WF	35.00000
White_Falcon_100 - Kotac_WF	White_Falcon_100	Felga_WF	70.00000
White_Falcon_100 - Kotac_WF - Felga_WF	White_Falcon_100	Glavcina_WF	17.00000
White_Falcon_100 - Kotac_WF - Felga_WF	White_Falcon_100	Zbica_WF	430.08000
White_Falcon_100 - Okvir_WF	White_Falcon_100	Vilica_WF	45.00000

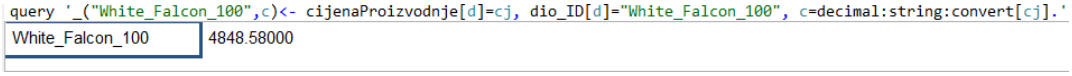
Slika 3.18: Ukupne cijene sastavljanja svih sklopivih dijelova za bicikl

Ukoliko bi nas zanimalo kolika bi bila ukupna cijena proizvodnje bicikla tada bismo mogli pomoću prethodna dva pravila 3.25a i 3.25b definirati sljedeća dva pravila:

$$\begin{aligned} &grupCijenaSastavljanjaSaNajjeftDijelovima[dio] = cijena <- \\ &\quad agg << cijena = total(cijene) >> \\ &\quad cijene = cijenaSastavljanjaSaNajjeftDijelovima[\_, dio, \_]. \end{aligned} \tag{3.26a}$$

$$\begin{aligned} &cijenaProizvodnje[dio] = cijena <- \\ &\quad c1 = grupCijenaSastavljanjaSaNajjeftDijelovima[dio], \\ &\quad c2 = cijenaSastavljanjaGrupirano[dio], cijena = c1 + c2. \end{aligned} \tag{3.26b}$$

Pravilom 3.26a zbrajamo sve ukupne cijene, te pravilom 3.26b da bi dobili cijenu proizvodnje nekog sklopivog dijela moramo dodati još i samu cijenu sastavljanja tog sklopivog dijela. Slika 3.19 prikazuje cijenu proizvodnje bicikla sa najjeftinijim odgovarajućim dijelovima za osnovne dijelove.



Slika 3.19: Cijena proizvodnje bicikla sa najjeftinijim odgovarajućim dijelovima

Zaključak

Prava snaga deduktivnih baza podataka leži u činjenici da one mogu dati (deducirati) puno više podataka nego što je u njima pohranjeno. Tako nešto je gotovo nemoguće implementirati u klasičnoj relacijskoj bazi podataka što deduktivnu bazu podataka čini puno moćnijom od klasične relacijske baza podataka. Deduktivne baze podataka sve do današnjice nisu doživjele veliki komercijalni uspjeh, odnosno nisu našle neki široki spektar primjene kao što su to našle relacijske baze podataka. Razlog tome je vjerojatno taj što se deduktivne baze podataka koriste za stvaranje velikih baza znanja što je uglavnom izvan dosega stvarnih potreba većine današnjih aplikacija.

No, ono što možemo primijetiti u zadnjih nekoliko godina je trend porasta količine znanja koji je eksponencijalan. Potrebno je pohranjivati sve veće količine podataka, stoga su koncepte deduktivnih baza podataka počeli koristiti i neki sustavi za upravljanje relacijskim bazama podataka (DB2, Oracle, MSSQL, MySQL itd.) omogućujući postavljenje upita koji koriste linearne rekurzije (rekurzivni SQL). Neki od njih (primjerice IBM-ov DB2) koriste čak i magične skupove kao metodu optimizacije rekurzivnih upita. Također razvoj u raznim drugim granama kao što su rudarenje podataka (eng. data mining), integracija podataka, mreže računala, analiza programa, sigurnost podataka te računanje u oblaku (eng. cloud computing) polako ali sigurno proširuje spektar primjene deduktivnih baza podataka. U tu svrhu je i nastao trenutno jedan od najpopularnijih komercijalnih sustava za deduktivne baze podataka (LogicBlox sustav) koji koristi deklarativni jezik LogiQL kao proširenje standardnog Datalog jezika.

U ovom diplomskom radu opisani su osnovni teorijski i praktični koncepti deduktivnih baza podataka, stoga cilj daljnjeg istraživanja mogao bi biti proširenje koncepata opisanih ovim radom na primjerice temporalne deduktivne baze podataka koje specificiraju temporalne veze među podacima, gdje vrlo važnu ulogu igraju transakcije. S druge strane kombinacijom deduktivnih baza podataka sa značajkama objektno orijentiranih sustava mogao bi se dobiti moćniji sustav koji bi mogao stvoriti bližu vezu između deduktivnih baza podataka i programskih jezika.

Bibliografija

[1] K. R. Apt, H. A. Blair i A. Walker, *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988, str. 89–148, ISBN 0-934613-40-0, http://dl.acm.org/citation.cfm?id=61352.61354.

[2] Ralf Hinze (auth.) Armin B. Cremers, Ulrike Griefahn, *Deduktive Datenbanken: Eine Einführung aus der Sicht der logischen Programmierung*, Artificial Intelligence / Künstliche Intelligenz, Vieweg+Teubner Verlag, 1994, ISBN 978-3-528-04700-9,978-3-663-09572-9.

[3] T. A. Halpin, *Object-role modeling fundamentals : a practical guide to data modeling with ORM*, Technics Publications, Basking Ridge, NJ, 2015, ISBN 9781634620741.

[4] LogicBlox, *LogicBlox 4 Reference Manual*, https://developer.logicblox.com/content/docs4/core-reference/html/index.html, [Zadnji put pristupljeno 10. lipnja 2016.].

[5] LogicBlox, *LogiQL REPL (Read-Eval-Print-Loop)*, https://repl.logicblox.com/?, [Zadnji put pristupljeno 13. lipnja 2016.].

[6] T A Halpin; Spencer Rugaber, *LogiQL: A Query Language for Smart Databases*, Emerging Directions in Database Systems and Applications, CRC Press, 2014, ISBN 978-1-4822-4493-9,1482244934,978-1-4822-4494-6.

[7] Letizia Tanca (auth.) Stefano Ceri, Georg Gottlob, *Logic Programming and Databases*, Surveys in Computer Science, Springer-Verlag Berlin Heidelberg, 1990, ISBN 978-3-642-83954-2,978-3-642-83952-8.

Sažetak

Ovaj diplomski rad sastoji se od tri poglavlja koja se logički mogu podijeliti u dva dijela. U prvom dijelu (1. i 2. poglavlje) iznijete su teorijske osnove i svojstva deduktivnih baza podataka te jezika Datalog. Razumijevanje pojmova i rezultata opisanih u prvom dijelu ključno je za realizaciju deduktivne baze podataka opisanu u drugom dijelu diplomskog rada.

Kako su deduktivne baze podataka nastale kao hibrid logičkog programiranja i relacijskih baza podataka, jasno je da će logika prvog reda igrati ključnu ulogu pri formalizaciji, razvoju i realizaciji deduktivnih baza podataka. Time se bavi prvo poglavlje gdje su iznijeti neki važni pojmovi i rezultati logike prvog reda. Tako najprije krećemo od same sintakse logike prvog reda gdje opisujemo kako izgledaju sintaktički ispravne formule nekog jezika logike prvog reda. Nakon što imamo sintaktički ispravne izjave želimo ih interpretirati sa semantičke točke gledišta. Na kraju prvog poglavlja uvodimo i neke ključne pojmove (Herbrandov svemir, Herbrandova baza, Herbrandova interpretacija) kako bi pripremili put za drugo poglavlje u kojem definiramo deduktivni sustav za evaluaciju Datalog programa.

Drugo poglavlje započinje proučavanjem sintakse Dataloga gdje definiramo kako izgledaju činjenice i pravila kojima pohranjujemo i izvodimo nove znanje. Zatim definiramo kako izgleda korektno napisan Datalog program uvodeći pojam stratifikacije. Nakon što naučimo pisati korektne Datalog programe bavimo se semantikom Dataloga. Tu ćemo vidjeti kako evaluirati Datalog programe koristeći iteraciju fiksne točke kao deduktivni sustav, pri čemu uz dane aksiome koje pruža sam Datalog program kao jedino pravilo dedukcije koristimo tzv. elementarnu produkciju. Zatim želimo omogućiti implementaciju Datalog koncepata unutar relacijske algebre preslikavanjem programa pisanih u Datalogu u relacijsku algebru. U drugom dijelu drugog poglavlja cilj nam je poboljšati samu evaluacju Datalog programa, stoga proučavamo razne metode evaluacije i optimizacije.

U trećem poglavlju u softveru koji prepoznaje Datalog jezik (Logic Query Langauge) realiziramo jednu konkretnu deduktivnu bazu podataka. Cilj je opisati razne mogućnosti softvera postavljanjem rekurzivnih upita nad deduktivnom bazom podataka.

Summary

This thesis consists of three chapters that can be logically divided into two parts. The first part (1. and 2. chapter) sets out the theoretical basis and properties of deductive databases and Datalog language. Understanding the concepts and results described in the first part is crucial for the realization of deductive database described in the second part of the thesis.

Deductive databases were created as a hybrid of logic programming and relational databases, so it is clear that the first-order logic will play a key role in the formalization, development and implementation of deductive databases. First chapter presents some important concepts and results of first-order logic. First we start from the syntax of first-order logic, where we define how do syntactically correct formula of first-order logic language look like. Once we have syntactically correct statements we want to interpret them from the semantic point of view. At the end of the first chapter we introduce some key concepts (Herbrand universe, Herbrand base, Herbrand interpretation) to prepare the way for the second chapter where we define a deductive system for evaluating Datalog programs.

The second chapter begins by studying the syntax of Datalog language where we define how do facts and rules by which we store data and derive new knowledge look like. Then we define how does correctly written Datalog program look like by introducing the concept of stratification. Once we learn how to write Datalog programs correctly we deal with the semantics of Datalog. Here we'll see how to evaluate Datalog programs using fixed-point iteration as a deductive system, where with the given axioms provided by Datalog program the only deduction rule will be the so-called elementary production. Then we will enable the implementation of Datalog concepts within the relational algebra by mapping a Datalog program into relational algebra. In the second part of the second chapter we aim to improve evaluation of a Datalog program, by studying various evaluation and optimization methods.

In the third chapter by using the software that recognizes Datalog language (Logic Query Langauge) we aim to build a concrete deductive database. The goal is to describe the various capabilities of the software by writeing recursive queries over the deductive database.

Životopis

Goran Brajdić rođen je 04.10.1982. u Zagrebu, gdje je pohađao Osnovnu i Srednju školu. Od malena pokazuje interes za tehniku i programiranje te u rujnu upisuje VII. gimnaziju u Zagrebu. Maturirao je u svibnju 2001. godine s temom "Tehnologije i princip rada čvrstog diska".U listopadu 2006. upisuje preddiplomski sveučilišni studij *Matematika* – PMF Matematički odsjek u Zagrebu koji završava u srpnju 2009. s akademskim nazivom Sveučilišni prvostupnik matematike. U jesen 2013. upisuje diplomski sveučilišni studij *Računarstvo i matematika* – PMF Matematički odsjek u Zagrebu. Diplomski studij završava u ljetu 2016. na temu "Deduktivne baze podataka" s akademskim nazivom Magistar računarstva i matematike.

Član je i voditelj programerske sekcije Kluba mladih tehničara Grada Zagreba, u sklopu kojeg sudjeluje i osvaja nagrade na raznim natjecanjima. Volonterski obavlja i posao voditelja tečajeva osnova rada na računalu i MS Office alata za umirovljenike. Aktivno se služi engleskim i pasivno njemačkim jezikom. U slobodno vrijeme bavi se sportom (biciklizam, teretana, trčanje).