Univerzitet u Beogradu
Matematički fakultet

Jedan pristup zasnivanju objektno-orijentisanog upitnog jezika i njegove formalne semantike

Magistarski rad

Goran Šuković

Odbrana: 8.01.98
Komisija: Gordana Pavlović (mentor)
Marjana Zakrzević
Žarko Mijajlović (predsednik komisije)

Pitanja: 1. Da li postoji implementacija baziran na F (3F) logici
2. Da li postoji metod (ili je moguć) iz relacionih baza podataka u F sistem.
3. Kako se rešili sa kvantorima
4. Vrednost istina
5. Semantika (istina, implementacija, pozitivna klasa

Beograd, 1997. godine

SADRŽAJ

0. Uvod

Dekadu koja je za nama u računarskom svijetu, po mnogo čemu je obilježio objektno-orijentisani pristup, posebno u oblasti baza podataka i programskih jezika. Iako je sam termin "objektno-orijentisani" definisan prilično neformalno, mnogi koncepti kao što su identitet objekata, složeni objekti, metodi, učaurenje i naslijedivanje postali su najvidljiviji aspekt takvog pristupa. Jedan od razloga za veliku popularnost ovog pristupa je činjenica da je na taj način moguće riješiti problem nekompatibilnosti postojećih programskih jezika za pisanje aplikacija i jezika baza podataka, takozvani "impedance mismatch" problem. Istovremeno sa razvojem objektno-orijentisanog pristupa, deduktivni pristup je dostigao široku popularnost. Pošto se logika može koristiti i kao formalizam za izračunavanja i kao jezik za definisanje podataka, pristalice deduktivnog pristupa smatraju da se na taj način takode rješava problem nekompatibilnosti. Medutim, oba pristupa pate od odredenih nedostataka. Objektno-orijentisanom pristupu je nedostajala formalna semantika, koja je tradicionalno važna u jezicima baza podataka. S druge strane, deduktivne baze podataka nemaju mogućnost apstrakcije podataka i najčešće koriste modele podataka koji nisu dovoljno fleksibilni.

Uprkos navedenim nedostacima i nepostojanju konsenzusa o "objektno-orijentisanom modelu podataka", proteklih nekoliko godina razvijeno je više sistema za upravljanje podacima koji se deklarišu kao, manje ili više, objektno-orijentisani. Najpoznatiji medu njima su Gemstone, O2, Orion, Starburst, UniSQL, Postgres , itd. Paralelno sa razvojem sistema, pojavili su se i deklarativni upitni jezici, najčešće kao nadgradnja nekog od postojećih jezika tipa SQL, kao što su OQL, CQL++, ESQL2 i XSQL([5], [7], [11], [12]). Niti jedan od ovih jezika u potpunosti ne obuhvata sve aspekte koje bi objektno-orijentisani model trebalo da posjeduje. Pored toga, sam koncept deklarativnog upitnog jezika u kojem je moguće otkrivanje unutrašne strukture objekta znači narušavanje učaurenja kao jednog od osnovnih aspekata objektno-orijentisanog pristupa.

Pojavom deklarativnih upitnih jezika, optimizacija upita je zauzela važno mjesto u istraživanjima vezanim za baze podataka. Deklarativnost jezika znači da korisnik upitom naznači koje podatke želi, dok sistem sam generiše najefikasniji plan izvršavanja upita na osnovu raspoloživih statističkih podataka, postojanja indeksa i klastera. Prvi korak u optimizaciji najčešće predstavlja transformisanje upita u neku grafičku reprezentaciju u obliku grafa. Zbog složenije strukture samog objektno-orijentisanog modela, proces optimizacije u objektno-orijentisanim sistemima složeniji je nego kod relacionih sistema. Činjenica da domen atributa ili metoda može biti proizvoljna klasa, postojanje metoda u upitima i hijerarhija klasa su samo neki od novih aspekata o kojima sistem mora da vodi računa prilikom optimizacije upita.

Problematika koju tretira ovaj rad odnosi se na problem zasnivanja objektno-orijentisanog upitnog jezika i njegove formalne semantike. Takode se razmatra i jedan način grafičke reprezentacije upita, kao prvi korak u algebarskoj optimizaciji. Detaljno je razradena logika (SF-Logika) koja na deklarativan način opisuje glavne strukturne karakteristike objektno-orijentisanog pristupa. Ova logika predstavlja jednu varijantu F-Logike (engl. - "frame logic") čije su osnovne osobine izložene u radovima M. Kiefer-a, G. Lausen-a i J. Wu-a ([8], [9], [10]). Dobijena SF-Logika služi kao podloga za

definisanje semantike objektnog upitnog jezika QLO (engl. - "query language for objects") zasnovanog na jeziku XSQL. Osnovni aspekti jezika XSQL obradeni su u radovima Won Kim-a, M. Kiefer-a i Y. Sagiv-a ([7]). Upitni jezik QLO zasnovan je na pojmu složenog izraza (engl. - "path expression") definisanom u XSQL-u. Pored nekih sintaksnih razlika u odnosu na XSQL, QLO ima mogućnost imenovanja pojedinih objekata (slično upitnom jeziku za O2) i rekurzivne upite. Poseban osvrt dat je na koncept učaurenja u deklarativnim upitnim jezicima. Tradicionalno, u složenom izrazu, pored metoda, ravnopravno mogu da učestvuju i atributi, što dovodi do otkrivanja unutrašnje strukture objekata. Uvodenjem ograničenja da u složenim izrazima mogu učestvovati samo metodi koji pripadaju interfejsu objekta postiže se očuvanje koncepta učaurenja.

Struktura rada je sledeća: u prvoj glavi dat je prikaz jednog objektno-orijentisanog modela podataka. Prikazane su sintaksa i semantika SF-logike kao i formalna teorija SF-Logike. U drugoj glavi, formalno je definisan pojam složenog izraza i njegova semantika. Takode je definisana gramatika upitnog jezika QLO, zasnovanog na XSQL i data je njegova formalna semantika, kao i način prevođenja upita QLO-a u odgovarajuće programe SF-Logike. Osnovne karakteristike jezika QLO i SF-Logike prikazane su kroz veći broj primjera. U glavi tri prikazan je program *SFLogika*, napisan primjenom programskog paketa *Microsoft Visual C++ 4.0*, koji ilustruje prevodenje upita QLO jezika u odgovarajuće programe SF-Logike. Glava četiri sadrži opis grafičke reprezentacije upita, kao jednog od mogućih početnih koraka u procesu optimizacije. U petoj glavi dat je zaključak i navedeni neki mogući pravci daljeg istraživanja. U prilogu je dat opis osnovnih funkcija programa *SFLogika*. Na kraju je prikazan opširan spisak referenci po abecednom redu.

1. Prikaz modela podataka

U ovom poglavlju razmotrićemo ukratko osnovne koncepte modela podataka na kome se zasniva upitni jezik. Dat je neformalan opis osnovnih pojmova, dok je formalna definicija modela podataka zasnovana na jednoj varijanti F-logike ([11]).

1.1. Osnovni pojmovi modela podataka

Objekti i njihova jednakost Objekat je apstraktan ili konkretan entitet u realnom svijetu. Svaki objekat identifikujemo pomoću identifikatora objekta (OID-a - "object identifier") koji može biti fizički ili logički. Logički identifikator je u stvari samo sintaksni termin u upitnom jeziku. Svaki logički OID jedinstveno odreduje objekat ali jedan objekat može imati više logičkih identifikatora. Fizički identifikator objekta je pojam vezan za implementaciju sistema i predstavlja, u stvari, pokazivač na objekat. Za razliku od fizičkih identifikatora, logički identifikatori mogu da nose i odredenu semantičku informaciju. Tako na primjer, '13' je logički identifikator apstraktnog objekta koji pripada klasi prirodnih (ili cijelih) brojeva i koji ima sve osobine prirodnog broja trinaest. Slično, "Institut za fiziku" je logički identifikator objekta koji ima sva uobičajena svojstva niza karaktera 'I', 'n', 's', itd. U tekstu koji slijedi koristićemo pojam objekat ili jednakost objekata u smislu logičkih identifikatora.

Atributi Objekat je opisan preko svojih osobina to jest atributa. Svaki objekat u našem modelu podataka predstavljen je torkom. Svaka pozicija u torki je vrijednost jednog atributa. Atribut može biti skalaran, i tada je njegova vrijednost jedan OID, ili može biti skupovni (engl. - "set-valued" ili "multivalued") kada je njegova vrijednost (uredeni) skup OID - a. Objekti koji su skupovi opisani su kao torke koje imaju samo jedan skupovni atribut. Skalarni atributi označavaju se jednom strelicom ($\rightarrow$) dok se skupovni atributi označavaju sa dvije strelice ($\rightarrow\rightarrow$).

PRIMJER 1.1.1. Objekat Rečenica (koji predstavlja skup riječi) može biti opisan na sledeći način : Klasa Rečenica (tip :(Sadržaj $\rightarrow\rightarrow$ Riječ)) .
Navedena definicija ima sledeće značenje : svaki objekat klase Rečenica je torka koja na prvoj (i jedinoj) poziciji sadrži skup riječi koje čine tu rečenicu. $\Diamond$

Atribut može biti definisan ili nedefinisan za neki objekat. Ako je definisan, tada atribut ima vrijednost. Nedefinisanost atributa je analogija sa NULL vrijednostima u relacionoj bazi podataka. Takode postoji mogućnost da neki atribut nije primjenljiv na neki objekat (tj. atribut je upotrebljen u okviru objekta na koji se ne može primjeniti) čime se postiže efekat greške u tipovima (engl. - "type error").

U zavisnosti od svoje pozicije u upitu, svaki OID može imati ulogu objekta ili atributa. Na taj način je omogućeno postavljanje upita koji otkrivaju strukturu baze podatka bez poznavanja sistemskih klasa koje reprezentuju šemu baze.

Klase Klase predstavljaju jedan način grupisanja objekata tj. klasa organizuje objekte u skupove povezanih entiteta. Klase su takođe objekti pa mogu imati atribute i nad njima se mogu postavljati upiti kao i nad ostalim objektima. Objekte koji nisu klase nazivaćemo individualnim objektima. Pretpostavićemo da su skup imena atributa (metoda), skup imena klasa i skup individualnih objekata međusobno disjunktni.

Uvođenjem pojma klasa uvode se i neke specijalne relacije između individualnih i klasnih objekata kao i između samih klasnih objekata. Relacija *instanca klase* (engl. - "instance of") je definisana između individualnog objekta i klase i kazuje nam da li individualni objekat pripada klasi (tj. klasu tretiramo kao skup instanci). Relacija *podklasa klase* (engl. - "subclass of") definisana je između dvije klase. Ako je klasa C podklasa klase K (ili K je nadklasa klase C), tada sve instance klase C pripadaju i klasi K. Relacija *podklasa klase* je aciklična pa definiše hijerarhiju klasa. Klasu K nazivamo baznom (osnovnom) klasom dok klasu C nazivamo izvedenom klasom.

Definicija 1.1.1 Graf nasljeđivanja je konačan orijentisan slabo povezan graf G=(X, R) , gdje je :

a) skup tjemena X je skup imena klasa;
b) skup grana R sadrži orijentisanu granu (uređeni par) (C, K) ako je klasa C podklasa klase K.

Kako je relacija podklasa aciklična, to je i graf nasljeđivanja acikličan.

Prilikom definisanja klase možemo da navedemo i listu njenih nadklasa i/ili podklasa kao i klase preklapanja i/ili disjunktne klase. Na taj način definišemo relacije *disjunktnost* i/ili *preklapanje* između dvije klase.

Pored navedenih relacija možemo definisati i relacije koje predstavljaju složene (kompozitne ili kompleksne) objekte (relacija *dio od* - "part-of") kao i relaciju *verzija od* (engl. - "version-of"). Ove relacije mogu biti definisane kao dio modela (kao što je na primjer, urađeno u ORION-u [11]) ili se njihova implementacija prepušta programeru.

Domen atributa je klasa, pa pored relacije *nadklasa/podklasa*, postoji i relacija *atribut/domen* između klase K i klase C koja je domen atributa A klase K. Relacija *atribut/domen* definiše poseban graf koji se naziva *kompozicionim grafom* ([11]). Za razliku od grafa nasljeđivanja, kompozicioni graf ne mora da bude acikličan tj. domen atributa A klase K može da bude sama klasa K ili neka od njenih direktnih ili indirektnih podklasa ili nadklasa.

Definicija 1.1.2 Kompozicioni graf je konačan orijentisan označen graf G=(X,R) , gdje je :

a) skup tjemena X je skup imena klasa;
b) skup grana R sadrži orijentisanu granu (uređeni par) l=(C, K) ako je u klasi C definisan atribut A čiji je domen klasa K; ako je atribut A skalaran, tada je grana l označena labelom A; ako je atribut A skupovni, tada je grana l označena labelom A^*.

Metodi Metod je uređeni par čija je prva komponenta ime metoda a druga komponenta je parcijalna funkcija (implementacija metoda). Implementacija metoda opisana je signaturom metoda (koja ujedno definiše i tip metoda), jezikom na kojem je

implementacija izvršena (neki od viših programskih jezika ili sam upitni jezik) i samim kodom (tekstom) metoda. Signatura metoda ima oblik

$$Meth: Arg_1, \cdots, Arg_n \to R \text{ - skalarni metod (*) ili}$$

$$Meth: Arg_1, \cdots, Arg_n \to\to R \text{ - skupovni metod (**)}$$

gdje je *Meth* ime metoda, $Arg_1, \cdots, Arg_n$ imena klasa parametara (argumenata) metoda i R ime klase rezultata. Atribute smatramo nularnim metodima pa su i atributi obuhvaćeni navedenom definicijom. Zbog estetskih razloga, umjesto Attr :→ R (ili Attr : →→ R) pišemo Attr→R (tj. Attr→→R). Značenje gore navedenih signatura je sledeće : ako su metodu *Meth* kao argumenti predati objekti koji su redom instance klasa $Arg_1, \cdots, Arg_n$, tada bi rezultat izvršavanja metoda trebalo da bude instanca (ili skup instanci, ako je metod skupovni) klase R. Striktno govoreći, postoji i 0-ti argument, jer se poziv metoda ostvaruje preko neke instance klase K u kojoj je definisan metod *Meth*. Kako se signatura metoda navodi u definiciji klase, to se 0-ti argument ne uključuje u signaturu.

Metodi mogu biti operatorski, atributni, programski i generički. Svi metodi klase čine sumedu (engl. - "interface") te klase.

Tipovi Klase su osnovni način klasifikacije objekata. Tipovi predstavljaju način klasifikacije objekata po njihovoj strukturi, dok objekte grupišemo u klase na osnovu nekih semantičkih kriterijuma. Instance iste klase dijele zajednička strukturna svojstva, pa grupisanje u klase implicira klasifikaciju po tipovima. Tip klase odreden je tipom njenih metoda i atributa. Tip metoda odreden je njegovom signaturom. Tip metoda definišemo na sledeći način : neka je metod *Meth* klase K opisan signaturom (*) (ili signaturom (**)). Tip metoda *Meth* je $K, Arg_1, \cdots, Arg_n \triangleright R$, gdje je K ime klase u kojoj je definisan metod *Meth*, $Arg_1, \cdots, Arg_n$ imena klasa kojima pripadaju argumenti metoda, R je klasa kojoj pripada rezultat, a $\triangleright$ označava $\Rightarrow$ ili $\Rightarrow\Rightarrow$. Metod može imati više signatura (tj. više tipova) i u tom slučaju kažemo da ima polimorfni tip. Preciznije, jednom imenu metoda možemo pridružiti više različitih implementacija (tj. funkcija) gdje svaka funkcija ima različitu signaturu. Pošto je dozvoljeno preklapanje klasa, moguće je da metod za istu vrstu argumenta kao rezultat daje instance (ili skupove instanci) različitih klasa. Tip takvog metoda zapisujemo na sledeći način:

$$Meth: K, Arg_1, \cdots, Arg_n \triangleright \{R_1, R_2\}.$$

PRIMJER 1.1.2. Neka je ZaposleniStudenti metod klase Odsjek koji vraća spisak studenata koji su u radnom odnosu na datom odsjeku (tj. signatura ovog metoda u klasi Odsjek je oblika *ZaposleniStudenti: Semestar →→ Student*); isti ti studenti moraju biti instance klase Zaposleni, koja je klasa preklapanja sa klasom Student. Tada signaturu metoda *ZaposleniStudent* možemo zapisati na sledeći način:

$$ZaposleniStudenti: Semestar \to\to \{Student, Zaposleni\}$$

dok je njegov tip

$$ZaposleniStudenti: Semestar \Rightarrow\Rightarrow \{Student, Zaposleni\}$$ ◊

Nasleđivanje Klasa može da naslijedi metode i atribute iz jedne ili više svojih nadklasa. Ako je K podklasa klase C, tada klasa K nasleduje sve metode i atribute

definisane u klasi C (kako one definisane u samoj klasi C tako i one koje je klasa C naslijedila od svojih nadklasa). Ovakva vrsta nasledivanja naziva se nasledivanjem ponašanja (engl. - behavioral inheritance). Na primjer, ako je neki od argumenata metoda *Meth* instanca klase C, tad se umjesto tog argumenta može uzeti instanca klase K. Drugim riječima, svaka instanca klase K može da se ponaša kao instanca klase C tj. može se upotrebiti na svakom mjestu na kojem je predviđena upotreba instanci klase C. Postoji i druga vrsta nasledivanja, takozvano strukturno nasledivanje (engl. - structural inheritance) : ako je klasa K podklasa klase C, tada objekti klase K nasleduju opštu strukturu objekata iz klase C. Strukturno nasledivanje je detaljnije obradeno u poglavlju Tipovi i nasledivanje.

1.2. Tipovi i nasledivanje

Nasledivanje ponašanja znači da se nasleduju definicije metoda, koje u izvedenoj klasi mogu biti zamijenjene novim definicijama (engl. - "overriding"). Ako je K klasa izvedena iz klase C i ako je *Meth* metod definisan u klasi C, tada K nasleduje definiciju (i signaturu) metoda *Meth*. Medutim, ako u klasi K redefinišemo metod *Meth* (metodu *Meth* pridružimo novu parcijalnu funkciju tj. implementaciju), tada nova definicija "prepisuje" naslijedenu definiciju tj. svaka upotreba metoda *Meth* preko instanci klase K poziva definiciju iz klase K a ne onu iz klase C. Pošto atribute tretiramo kao nularne metode to su i atributi obuhvaćeni navedenim razmatranjem.

Strukturno nasledivanje znači da se tipovi metoda (ali ne i kôd metoda) uvijek nasleduju ali se nikada ne "prepisuju". Podklasa K klase C nasleduje sve signature koje postoje u klasi C a može imati i nove signature (koje su rezultat novih deklaracija za metod *Meth* u klasi K).

PRIMJER 1.2.1. U klasi Zaposleni možemo definisati metod Ocjena sa signaturom *Ocjena : Projekat* $\rightarrow$ *Integer* , koji svakom zaposlenom dodjeljuje ocjenu učešća na projektu; u klasi Student postoji metod *Ocjena : Predmet* $\rightarrow$ *Integer*, koji za svakog studenta odreduje ocjenu koju je dobio na ispitu iz predmeta. Metodu *Ocjena* možemo pridružiti dva tipa : *Zaposleni, Projekat* $\Rightarrow$ *Isplata* (iz klase Zaposleni) i *Student, Predmet* $\Rightarrow$ *Integer* (iz klase Student). U klasi StudentskoOsoblje, koja je izvedena iz klasa Zaposleni i Student, metod *Ocjena* kao rezultat daje objekat klase Isplata ako mu je argument iz klase Projekat; ako je argument instanca klase Predmet, rezultat će biti ocjena (tj. prirodan broj izmedu 5 i 10). Klasa StudentskoOsoblje naslijedila je signature metoda *Ocjena* i iz klase Zaposleni i iz klase Student. $\Diamond$

U opštem slučaju, hijerarhija klasa predstavljena je usmjerenim acikličnim grafom (DAG) tj. klasa K može imati dvije ili više neuporedivih nadklasa (klasa koje medusobno nisu u relaciji *podklasa* ili *nadklasa*). Skup signatura metoda *Meth* u klasi K sadrži sve signature naslijedene iz svih nadklasa klase K. Pitanje nasleđivanja ponašanja je znatno složenije. Pretpostavimo da su C_1 i C_2 medusobno neuporedive nadklase klase K i da je metod *Meth* definisan i u C_1 i u C_2. U tom slučaju nije jasno koju od

definicija metoda *Meth* nasleđuje klasa K. U našem slučaju, konflikt se rješava eksplicitno : kao dio definicije podklase naznači se koja se definicija nasleđuje (opcija INHERITS u definiciji atributa/metoda). Ako nije specificirana opcija INHERITS, tada se metod nasleđuje iz prve nadklase iz liste nadklasa (SUPERCLASSES) klase K. Drugi način rješavanja ovog problema je preimenovanje metoda u izvedenoj klasi pomoću opcije RENAME. Bez obzira koju je definiciju metoda *Meth* naslijedila klasa K (čak i kada je *Meth* redefinisan u klasi K), zbog strukturnog nasljedivanja, klasa K nasleđuje sve signature koje metod *Meth* ima u C_1 i u C_2.

Neka su $T_0, T_1, \cdots, T_n \triangleright RT$ (2) i $A_0, A_1, \cdots, A_n \triangleright RA$ (3) proizvoljni tipovi metoda, gdje $\triangleright$ označava $\Rightarrow$ ili $\Rightarrow\Rightarrow$.

Definicija 1.2.1. Tip (2) je nadtip (supertip) tipa (3) (ili tip (3) je podtip tipa (2)) ako za svako i, i=1,2,...,n, T_i je podklasa klase A_i, ako je *RT* nadklasa klase *RA* i (2) i (3) su ili oba skalarna ili oba skupovna (tj. koriste iste vrste strelica).

U definiciji 1.2.1., nadtip označava nadskup tj. skup funkcija opisanih sa (2) je nadskup skupa funkcija opisanih sa (3). Pojam nadklase i podklase ne tretira se striktno (tj. klasa K je podklasa/nadklasa klase K).

Definicija 1.2.2. Ako je metodu *Meth* pridružen tip (3), tada *Meth* posjeduje tip (2) ako je (2) nadtip tipa (3).

Skup svih tipova koje posjeduje neki metod zatvoren je u odnosu na relaciju nadtip pa se na ovaj način ostvaruje efekat strukturnog nasljedivanja ([2], [11]).

PRIMJER 1.2.2. Posmatrajmo metod *Ocjena* definisan u primjeru 1.2.1. Ovom metodu pridružen je tip *Zaposleni, Projekat* $\Rightarrow$ *Integer* (neka je to tip T_1). Klasa *StudentskoOsoblje* koja je podklasa klase Zaposleni, nasleđuje ovaj metod. U klasi *StudentskoOsoblje*, ovom metodu pridružen je novi tip (neka je to T_2) *StudentskoOsoblje, Projekat* $\Rightarrow$ *Integer*. Tada je T_1 podtip tipa T_2 i metod *Ocjena* posjeduje tipove T_1 i T_2 (na osnovu definicije 1.2.2.).

1.2.1. Sistemske klase

Konceptualna šema relacione baze podataka implementira se preko sistemskih relacija (tabela). To su relacija koja opisuje sve relacije u bazi, relacija koja opisuje sve atribute svih relacija baze, itd. Pored toga, u sistemskim tabelama čuvaju se informacije o pristupu bazi, privilegije pojedinih korisnika, informacije o kreiranim pogledima, statistički podaci koji se koriste za optimizaciju upita, itd.

Primjer 1.3.1. Sistemski katalog DB2 sistema sadrži više od 30 tabela (relacija), koje opisuju osnovne tabele u bazi, definisane poglede i indekse, korisnike i njihova prava dostupa, planove izvršavanja upita, itd. Tabela SYSTABLES ima polja NAME (naziv tabele), CREATOR (ko je kreirao tabelu), COLCOUNT (broj kolona u tabeli), itd.

Tabela SYSCOLUMNS sadrži po jedan zapis za svaku kolonu svake tabele : ime kolone (NAME), ime tabele (TBNAME), tip (COLTYPE) , itd. ◊

Prirodno je da šema objektno-orijentisane baze podataka bude opisana pomoću sistemskih klasa. Sistemske klase, pored toga što sadrže informacije o svim klasama u bazi i atributima i metodima definisanim za svaku klasu, sadrže i informaciju o agregaciji (izmedu klase i atributa) odnosno generalizaciji (odnos klasa/podklasa). Sistemske klase takode treba da sadrže i statističke podatke neophodne za optimizaciju upita, kao što su broj instanci klase, broj instanci svih njenih podklasa, i slično. Neke od sistemskih klasa su :

- SysKlasa
- SysAtribut
- SysMetod
- SysIndeks
- Objekat

Sistemska klasa SysKlasa opisana je atributima : *ImeKlase, AtributiKlase, Nadklase, Podklase, KlasePreklapannja, DisjunktneKlase, MetodiKlase. AtributiKlase* je skup svih atributa definisanih u datoj klasi ili naslijedenih od nadklasa. Atribut *MetodiKlase* predstavlja skup metoda definisanih u datoj klasi ili naslijeđenih od nadklasa. Atributi *AtributiKlase* i *MetodiKlase* klase SysKlasa sadrže vrijednosti za atribute i metode definisane u datoj klasi kao i za sve atribute i metode naslijedene iz nadklasa. Ovakva tehnika poznata je pod nazivom "izravnavanje" ili "peglanje" ("flattening") hijerahije klasa. Ako se hijerarhija klasa realizuje tako da svaka klasa predstavlja samo atribute/metode definisane samo za nju i ako je u upitu referenciran atribut/metod koji je naslijeđen iz nadklase, tada se mora pretraživati hijerarhija da bi se odredilo iz koje je nadklase referencirani atribut/metod. Primjenom tehnike "peglanja", referencirani atribut/metod se pronalazi u instanci klase SysKlasa, tj. ne pretražuje se hijerarhija klasa, čime se postiže ubrzanje pristupa šemi i procesa obrade upita. Nedostaci ove tehnike su što se mora izvršiti pretraživanje klase SysKlasa u slučaju kada se briše atribut ili metod klase jer brisanje atributa ili metoda treba izvršiti i u svim podklasama.

Klasa SysAtribut (SysMetod) sadrži po jednu instancu za svaki atribut (metod) definisan u nekoj klasi. Klasa SysAtribut ima atribute *ImeAtributa, Domen* i *Multiple. Domen* je skup klasa kojima pripadaju vrijednosti atributa. Vrijednost atributa *Multiple* može biti TRUE ili FALSE (*Multiple*=TRUE ako je atribut skupovni, inače je *Multiple*=FALSE).

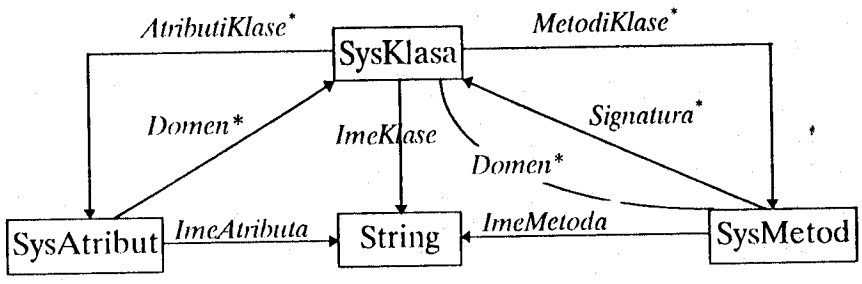
Klasa SysMetod ima atribute *ImeMetoda, Jezik, Signatura, Domen* i *KodMetoda*. *Signatura* opisuje tip argumenata i tip rezultata metoda (tj. klase kojima pripadaju argumenti i rezultat metoda). *KodMetoda* je string koji je ili ime datoteke koja sadrži kod programa ili string koji predstavlja upit jezika. *Jezik* je ili neki od viših programskih jezika kao što su C, C++, Basic ili upitni jezik.

Klasa SysIndeks sadrži po jednu instancu za svaki indeks kreiran u bazi podataka. Atributi klase SysIndeks su : *ImeIndeksa*, *ImeKlase* i *Staza*. Atribut *Staza* opisuje niz atributa koji polazi iz klase *ImeKlase* i nad kojima je napravljen indeks čije je

ime *ImeIndeksa*. Instancama ove klase pristupamo prilikom generisanja plana izvršavanja upita.

Klasa Objekat je korijen hijerarhije klasa i kao svoje instance ima sve objekte baza podataka, bez obzira da li su individualni, atributski ili klasni. Ova klasa je nadklasa svake klase, iako se ne navodi u listi nadklasa prilikom definisanja klase.

Dio kompozicionog grafa sistemskih klasa prikazan je na slici 1.3.1., gdje simbol * označava skupovni atribut.



Slika 1.3.1.

1.3. Pregled F-Logike

F-Logika predstavlja jedan mogući formalni okvir za najveći dio strukturnih karakteristika objektno-orijentisanih i takozvanih "frame-based" programskih jezika. I samo slovo F u nazivu F-Logika potiče od engleske riječi "frame". Te karakteristike obuhvataju identitet objekata, složene objekte, metode, učaurenje i naslijedivanje. Odnos F-Logike i objektno-orijentisanog pristupa identičan je odnosu predikatskog računa prvog reda i relacionog modela podataka ([8]). Sintaksa F-Logike je višeg reda, što izmedu ostalog daje mogućnost korisniku da na uniforman način dobija informacije o konceptualnoj šemi i same podatke iz baze podataka. Osnovne osobine F-Logike prikazaćemo kroz sledeći primjer.

PRIMJER 1.3.1

1. Definicija klase *Zaposleni*

   $Zaposleni[Ime \Rightarrow String; Plata \Rightarrow Double; Djeca \Rightarrow\Rightarrow Child(Zaposleni); Stan \Rightarrow Adresa;$
   $Sprema \Rightarrow String; Sprema * \rightarrow 'SSS'; Radi\_Na\_Projektu \Rightarrow\Rightarrow Projekat;$
   $Zajednicki\_Rad\#Zaposleni \Rightarrow\Rightarrow Projekat; ProsjecnaPlata \rightarrow 1250]$

2. Definicija klase *Programer*

   $Programer : Zaposleni$

   $Programer[ProgramskiJezici \Rightarrow\Rightarrow String; Sprema * \rightarrow 'VSS']$

3. Definicija odnosa klasa/podklasa - $SistemProgramer : Programer$

4. Definicija klase *SistemProgramer*

   $SistemProgramer[OperativniSistem \Rightarrow\Rightarrow String]$

5. Definisanje pripadnosti objekta klasi - $marko : Zaposleni$

6. Definicija vrijednosti atributa i metoda objekta *marko*

$$marko[Ime \rightarrow 'Marko\ Rudan'; Radi\_na\_Projektu \rightarrow\!\!\rightarrow \{ekon, direkcija\}; Plata \rightarrow 1450$$
$$Stan \rightarrow adr[Grad \rightarrow 'Beograd'; Ulica \rightarrow 'Mose\ Pijade\ 25']; Djeca \rightarrow\!\!\rightarrow \{mira, ana\}]$$

7. Definisanje pripadnosti objekta klasi - $janko : Programer$
8. Definicija vrijednosti atributa i metoda objekta *janko*

$$janko[Ime \rightarrow 'Janko\ Peric'; ProgramskiJezici \rightarrow \{'C', 'C++'\}; Sprema \rightarrow 'VSS']$$

9. Definisanje pripadnosti objekta klasi - $mira : Child(marko)$ $ana : Child(marko)$
10. Definisanje metoda *Zajednicki_rad*

$$X[Zajednicki\_rad\#Y \rightarrow\!\!\rightarrow Z] \leftarrow X : Zaposleni \wedge X[Radi\_Na\_Projektu \rightarrow\!\!\rightarrow Z]$$
$$\wedge Y : Zaposleni \wedge Y[Radi\_Na\_Projektu \rightarrow\!\!\rightarrow Z]$$

11. Upit - $?- janko[Zajednicki\_rad\#marko \rightarrow\!\!\rightarrow X]$

Definisanje klase prikazano je u primjerima 1, 2 i 3. *Zaposleni*, *Programer* i *SistemPrograme*r su konstruktori objekata (u ovom slučaju imena klasa). Na isti način, *marko*, *janko*, *mira* i *ana* su identifikatori individualnih objekata, a *Ime*, *Stan*, *Radi_Na_Projektu*, *Zajednicki_rad* su identifikatori metoda (atribute tretiramo kao nularne metode). Klasa je opisana atributima i metodima, koji mogu biti skalarni ($\Rightarrow$) ili skupovni ($\Rightarrow\Rightarrow$). Metod sa n argumenata tretiramo kao parcijalnu funkciju oblika $Objekti \times Objekti'' \rightarrow Objekti$ ili $Objekti \times Objekti'' \rightarrow P(Objekti)$ (u zavisnosti od toga da li je metod skupovni ili skalarni). Metodi jednog objekta su učaureni (engl. - "encapsulated") u objektu i čine interfejs tog objekta. Strelice oblika $\Rightarrow\Rightarrow$ i $\Rightarrow$ označavaju da je metod definisan; $\rightarrow\!\!\rightarrow$ i $\rightarrow$ označavaju vrijednost metoda i mogu biti upotrijebljene jedino ako je metod definisan. Strelice oblika $*\rightarrow\!\!\rightarrow$ i $*\rightarrow$ označavaju nasledne osobine, to jest vrijednosti koje nasleđuju svi objekti klase kao i sve podklase; $\rightarrow\!\!\rightarrow$ i $\rightarrow$ označavaju osobine koje važe samo za navedeni objekat i koje se ne nasleduju. Tako primjer 1 označava da za svaki objekat klase Zaposleni, vrijednosti atributa *Ime*, *Plata* i *Stan* moraju pripadati klasama *String*, *Float* i *Adresa*, respektivno.

U trenutnoj sinataksi, ":" označava pripadnost objekta klasi, dok "::" označava podklasu. Tako u primjerima 2 i 3, klasa *Programer* je podklasa klase *Zaposleni*, a klasa *SistemPrograme*r je podklasa klase *Programer*, dok je u primjerima 5, 7 i 9 naznačeno da objekti *marko*, *janko* i *mira* pripadaju redom klasama *Zaposleni*, *Programer* i *Child(janko)*. U primjeru 6 definisane su vrijednosti metoda objekta čiji je identifikator *marko*: vrijednost atributa *Ime* je string *'Marko Rudan'*, atributa *Plata* je realan broj 1450 (objekat klase Float) a vrijednost atributa Stan je objekat *adr* klase Adresa. Vrijednost atributa *Radi_Na_Projektu* je skup objekata $\{ekon, direkcija\}$ koji pripadaju klasi *Projekat*. U primjeru 8 ponovljen je isti postupak za objekat *janko*. Atribut *Ime* nije direktno definisan za klasu *Programer* već je naslijeden od nadklase *Zaposleni*. Takode, atribut *Sprema* je u klasi *Programer* naznačen kao osobina koja se nasleđuje ($*\rightarrow$), pa kako *janko* pripada klasi *Programer*, imamo da je vrijednost atributa *Sprema* objekta *janko* string *'VSS'* - naslijedena iz klase *Programer* ($janko[\ldots; Sprema \rightarrow 'VSS']$). Objekat *janko* je individualni objekat, pa je atribut *Sprema* označen kao osobina koja se

ne nasljeduje (strelica je $\rightarrow$ a ne $*\rightarrow$). Medutim, u klasi *SistemProgramer* koja je podklasa klase *Programer*, atribut Sprema je osobina koju nasleduju, bilo podklase klase, bilo objekti klase. Za razliku od atributa *Sprema*, atribut *ProsjecnaPlata* klase Zaposleni je osobina koja se ne nasleduje. Individualni objekti ne nasleduju ovu osobinu jer je ona vezana za samu klasu a ne za pojedine objekte te klase (analogija statičkoj promjenljivoj u C++). Takode, podklase ne nasleduju ovu osobinu, jer će se vrijednost prosječne plate za podklasu *Programer* najvjerovatnije razlikovati od date vrijednosti za klasu *Zaposleni*.

U primjeru 1, atribut *Djeca* klase Zaposleni definisan je na sledeći način : $Djeca \Rightarrow Child(Zaposleni)$, da bi vrijednost atributa *Djeca* objekta *marko* bio skup $\{mira, ana\}$. Objeki *mira* i *ana* pripadaju klasi *Child*(*marko*). Vrijednost atributa *Djeca* objekta *marko* bio bi skup objekata koji bi pripadali klasi *Child*(*marko*), kako je i naznačeno u definiciji klase *Zaposleni*. *Child* je unarni konstruktor objekata, pa je *Child(marko)* identifikator klase.

Primjer 10 prikazuje metod *Zajednicki_rad* klase *Zaposleni*. Metod ima jedan argument (koji mora pripadati klasi *Zaposleni*) a kao rezultat vraća skup objekata klase *Projekat* . Upotreba ovog metoda prikazana je u upitu iz primjera 11. Rezultat upita su svi projekti na kojima osoba čiji je identifikator *janko* radi zajedno sa osobom čiji je identifikator *marko*.

1.4. SF-Logika

U ovom poglavlju formalno se definiše SF-Logika kao jedna varijanta F-Logike. Naziv SF-Logika je skraćenica od "sorted F-Logic"([8]). Za razliku od F-Logike, skup konstruktora objekata u SF-Logici je podijeljen na tri medusobno disjunktna podskupa koji predstavljaju klase, metode i individualne objekte ([8], [9], [10]). Na taj način se postiže veća sintaksna jasnoća, iako je izražajna moć ovih logika jednaka. Ideja o podjeli skupa konstruktora na medusobno disjunktne podskupove potiče od samih autora F-Logike, kao jedna od njenih mogućih nadgradnji. Zbog jasno razgraničenih klasnih, metodskih i individualnih konstruktora, nema potrebe za uvodenjem naslednih i nenaslednih osobina koje postoje kod F-Logike, čime se umanjuje složenost sintakse i semantike SF-Logike.

1.4.1. Matematički pojmovi

Neka su A i B proizvoljni skupovi. Sa *Total(A, B)* označavamo skup svih totalnih funkcija iz A u B a sa *Partial(A, B)* skup svih parcijalnih funkcija iz A u B. Partitivni skup skupa A označavamo sa *P(A)*. Ako je $\{S_i : i \in N\}$ familija skupova, tada $\prod_{i=1}^{\infty} S_i$ označava Dekartov proizvod skupova iz date familije, tj. skup svih beskonačnih torki $(s_1,\ldots,s_n,\ldots)$, gdje je $s_i \in S_i, i \in N$.

Ako je $\leq$ relacija parcijalnog uredenja na skupu A, tada relaciju parcijalnog uredenja na skupu A^n definišemo na sledeći način : za svaka dva elementa $u=(u_1,\ldots,u_n), v=(v_1,\ldots,v_n)$ skupa A^n, $u \leq v$ ako je $u_i \leq v_i, i=1,\ldots,n$. Na isti način definišemo relaciju $\in$ za uredene n-torke : $u \in V$, gdje je $u=(u_1,\ldots,u_n), u_1,\ldots,u_n \in A$, $V=(A_1,\ldots,A_n), A_1,\ldots,A_n \subseteq A$ ako je $u_i \in A_i$, $i=1,\ldots,n$.

Neka su $\leq_A$ i $\leq_B$ parcijalna uredenja skupova A i B. Označimo sa AM podskup skupa *Partial(A,B)* takav da[1] : $f \in AM$ ako za svako a i b iz skupa A, ako je $f(a)$ definisano i ako je $a \leq_A b$, tada je i $f(b)$ definisano i važi da je $f(b) \leq_B f(a)$. Neka UP označava podskup skupa *P(A)* takav da : $X \in UP$ ako za svako $a \in X$ i svako $b \in A$ za koje je $a \leq_A b$, važi da $b \in A$.[2]

1.4.2 SF-logika na primjerima

Prikazaćemo fragment baze podatka o kompanijama i njihovim radnicima zapisane formulama SF-Logike. Kompletna šema ove baze prikazana je u poglavlju 2.4. Na nekoliko karakterističnih primjera, ilustrovaćemo osnovne sintaksne osobine SF-Logike. Uporedujući date formule sa formulama F-Logike mogu se uvidjeti osnovne sličnosti i razlike ovih logika.

PRIMJER 1.4.2.1.

[1] AM - skraćenica za anti monotona

[2] UP - skraćenica za zatvorenost naviše (engl. - "upward closedness")

Definicija klasa

1. definicija klase *Osoba*

$$Osoba[Ime \Rightarrow String; Godine \Rightarrow Integer; Stan \Rightarrow Adresa;$$
$$Vozila\_U\_Vlasnistvu \Rightarrow\Rightarrow Vozilo; NovaOsoba\#String, Integer, Adresa \Rightarrow Osoba];$$

2. definicija klase *Zaposleni*

$Zaposleni :: Osoba$ - klasa *Zaposleni* je podklasa klase *Osoba* ;

$$Zaposleni[Kvalifikacije \Rightarrow\Rightarrow String; Plata \Rightarrow Double; Clanovi\_porodice \Rightarrow\Rightarrow Osoba;$$
$$Zavisni \Rightarrow\Rightarrow Osoba; Radi\_Na\_Projektu \Rightarrow\Rightarrow Projekat;$$
$$Zajednicki\_Rad\#Zaposleni \Rightarrow\Rightarrow Projekat;$$
$$NoviZaposleni\#String, Double \Rightarrow Zaposleni];$$

3. definicija klase *Kompanija*

$$Kompanija[Ime \Rightarrow String; Sjediste \Rightarrow Adresa; Odsjeci \Rightarrow\Rightarrow Sektor; Penzioneri \Rightarrow\Rightarrow Osoba;$$
$$Predsjednik \Rightarrow Zaposleni; Sef\_Odsjeka\#String, String \Rightarrow Zaposleni];$$

Definisanje individualnih objekata

4. $janko : Zaposleni$ - osoba čiji je identifikator *janko* pripada klasi *Zaposleni*
5. podaci o janku

$$janko[Ime \rightarrow "Janko\ Rudan"; Stan \rightarrow adr[Ulica \rightarrow "Beogradska\ 23"; Grad \rightarrow "Nis"];$$
$$Godine \rightarrow 29; Vozila\_U\_Vlasnistvu \rightarrow\rightarrow \{zast101, corsa\}; plata \rightarrow 1460;$$
$$Radi\_Na\_Projektu \rightarrow\rightarrow \{javni\_radovi, ekonomski\_fakultet\}];$$

6. $corsa : Vozilo$
7. $zast101 : Vozilo$;
8. $zast101[Model \rightarrow "scala\ 55"; Proizvodjac \rightarrow zastava; Tezina \rightarrow 983{,}25; boja \rightarrow "plava"]$;
9. $corsa[Model \rightarrow "corsa\ 1.4i"; Proizvodjac \rightarrow opel\_ge; boja \rightarrow "siva"]$;
10. $opel\_ge : Kompanija$;
11. $opel\_ge[Ime \rightarrow "Opel\ Germany"; Predsjednik \rightarrow bob[Ime \rightarrow "Bob\ Rice"; Godine \rightarrow 48]]$
12. *marko : zaposleni*

$$marko[Ime \rightarrow "Marko\ Ban"; Stan \rightarrow adr1[Ulica \rightarrow "Marka\ Miljanova\ 23"; Grad \rightarrow "Nis"];$$
$$plata \rightarrow 1460; Clanovi\_Porodice \rightarrow\rightarrow \{janko, mira, dijana\}$$
$$Radi\_Na\_Projektu \rightarrow\rightarrow \{javni\_radovi, ekonomski\_fakultet, vodovod\}];$$

Definisanje atributa/metoda

13. Definisanje metoda *Zajednicki_Rad*

$$X[Zajednicki\_rad\#Y \rightarrow\rightarrow Z] \leftarrow X : Zaposleni \wedge X[Radi\_Na\_Projektu \rightarrow\rightarrow Z]$$
$$\wedge Y : Zaposleni \wedge Y[Radi\_Na\_Projektu \rightarrow\rightarrow Z]$$

14.Definisanje metoda *Sef_odsjeka*

$$X[Sef\_Odsjeka\#Y,Z \rightarrow D] \leftarrow X:Kompanija \wedge Y:String \wedge Z:String$$
$$\wedge\ X[Odsjeci \rightarrow M[Ime \rightarrow Y; Upravnik \rightarrow D; Lokacija \rightarrow S]] \wedge S[Drzava \rightarrow Z]$$

Upiti

15. ?- $X:Zaposleni \wedge X[Ime \rightarrow "Janko\ Rudan"; Godine \rightarrow G; Adresa \rightarrow A]$;

16. ?- $janko[Zajednicki\_rad\#marko \rightarrow\rightarrow X]$;

17. ?- $janko[Zajednicki\_rad\#X \rightarrow\rightarrow javni\_radovi]$;

18. ?- $X:Kompanija \wedge X[Sjediste \rightarrow Y; Sef\_Odsjeka\#"Marketing","Yugoslavia" \rightarrow Z]$;

U prva tri primjera prikazani su tipovi klasa, to jest signature njihovih metoda i atributa (atribute tretiramo kao nularne metode). Na primjer, *Sef_Odsjeka* je skalarni metod klase *Kompanija* koji kao argumente ima dva objekta (oba iz klase *String*), a kao rezultat daje objekat iz klase *Zaposleni*. Slično tvrđenje važi i za metod *Zajednicki_Rad*, koji je skupovni (što je označeno sa $\Rightarrow\Rightarrow$), pa kao rezultat vraća skup objekata klase *Projekat*. Atribut *Plata* je skalarni (nularni skalarni metod), dok je atribut *Clanovi_porodice* skupovni, jer je njegova vrijednost skup objekata klase *Osoba*.

U primjerima 4-11 definisane su vrijednosti metoda i atributa više individualnih objekata i njihova pripadnost pojedinim klasama. Tako na primjer, izraz u primjeru 4 *janko : Zaposleni*, označava da objekat *janko* pripada klasi Zaposleni. Primjer 5 određuje osobine objekta *janko*: njegovo ime (string 'Janko Rudan'), njegovu adresu (objekat čiji je identifikator *adr* koji pripada klasi *Adresa*), starost (29 - objekat klase *Integer*), visinu njegove plate (1460 - objekta klase *Float*), vozila koja posjeduje (dva vozila čiji su identifikatori *corsa* i *zast101* - oba pripadaju klasi *Vozilo*) i projekte na kojima trenutno radi (projekti sa nazivima *ekonomski_fakultet* i *javni_radovi* iz klase *Projekti*). Na sličan način su definisani i objekti *corsa* i *zast101* (iz klase Vozilo), *marko* (iz klase *Zaposleni*) i *opel_ge*, koji pripada klasi *Kompanija*. Skalarni atributi/metodi označeni su sa $\rightarrow$, dok su skupovni atributi označeni sa $\rightarrow\rightarrow$.

Kako je objekat *janko* instanca klase *Zaposleni*, to je moguće da atribut Vozila_U_Vlanistvu ima vrijednost $\{corsa, zast101\}$, jer je u definiciji klase *Zaposleni* naglašeno da je atribut Vozila_U_Vlanistvu skupovni i da elementi tog skupa moraju pripadati klasi Vozilo ($Osoba[\ldots; Vozila\_U\_Vlasnistvu \Rightarrow\Rightarrow Vozilo; \ldots]$).

U primjeru 14, metod *Sef_odsjeka*, za kompaniju *X*, prikazuje šefa odsjeka *Y* lociranog u državi *Z*. Metod *Zajednicki_rad* (primjer 13) daje sve projekte na kojima zajedno sa osobom *X* radi i osoba *Y*. Značenje izraza $Radi\_Na\_Projektu \rightarrow\rightarrow Z$ je da promjenljiva *Z* dobija vrijednosti elemenata skupa *X.Radi_Na_Projektu*, a ne da je Z jednaka samom skupu *X.Radi_Na_Projektu*.

Prvi upit (primjer 15) daje sve zaposlene čije je ime "Janko Rudan" i pri tome se daju vrijednosti atributa *Godine* i *Adresa*. Rezultat upita je ranije definisan objekat : $janko[Ime \rightarrow "Janko\ Rudan"; Stan \rightarrow adr[Ulica \rightarrow "Beogradska\ 23"; Grad \rightarrow "Nis"]$;

$Godine \rightarrow 29]$

Drugi upit prikazuje sve projekte na kojima zajedno rade zaposleni *marko* i *janko,* dok treći upit određuje sve zaposlene koji rade zajedno sa jankom na projektu *javni_radovi.* U četvrtom upitu (primjer 18), za svaku kompaniju prikazuje se njeno sjedište i šef odsjeka marketinga za Jugoslaviju.◊

1.4.3. Sintaksa SF logike

Alfabet SF-logike sastoji se od:

- međusobno disjunktnih skupova konstruktora (funkcijskih ili operacijskih slova) : F_1 (konstruktori individualnih objekata), F_2 (konstruktori klasnih objekata) i F_3 (konstruktori atribut/metod-objekata)
- međusobno disjunktnih skupova promjenljivih : Var_1, Var_2 i Var_3 to jest skupa individualnih promjenljivih, skupa klasnih promjenljivih i skupa atributskih/metodskih promjenljivih, respektivno;
- skupa relacijskih simbola (slova) *Pred*;
- logičkih veznika i simbola : $\wedge, \vee, \neg, \leftarrow, \forall, \exists$
- pomoćnih simbola : $\{,\},(,),[,]$, $\rightarrow, \Rightarrow, \rightarrow\rightarrow, \Rightarrow\Rightarrow$, #, itd.

Svakom od elemenata skupova F_1, F_2, F_3 odgovara po jedan broj iz skupa $N \cup \{0\}$, koji predstavlja arnost funkcijskog slova. Na isti način, svakom elementu *pr* skupa *Pred* odgovara jedan prirodan broj *n* - arnost relacije *pr.* Elementi skupova F_1 , F_2 i F_3 arnosti 0 predstavljaju konstante. Skupovi $F_1, F_2, F_3, Var_1, Var_2, Var_3$ i *Pred* su najviše prebrojivi. Najčešći je slučaj da su skupovi funkcijskih i relacijskih simbola konačni, dok su skupovi promjenljivih beskonačni. Uvedimo oznake : $F = F_1 \cup F_2 \cup F_3$ i $Var = Var_1 \cup Var_2 \cup Var_3$.

Definicija 1.4.3.1. Skupovi Ω_1, Ω_2 i Ω_3 su najmanji skupovi takvi da :

a) Svaki element x skupa Var_1 pripada skupu Ω_1. Svaki element x skupa Var_2 pripada skupu Ω_2. Svaki element x skupa Var_3 pripada skupu Ω_3.

b) Ako je $f \in F_1$ simbol arnosti $n, n \geq 0$ i ako su $t_1, t_2, \ldots, t_n$ elementi skupa $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$, tada je $f(t_1, t_2, \ldots, t_n)$ element skupa Ω_1.

c) Ako je $f \in F_2$ simbol arnosti $n, n \geq 0$ i ako su $t_1, t_2, \ldots, t_n$ elementi skupa $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$, tada je $f(t_1, t_2, \ldots, t_n)$ element skupa Ω_2.

d) Ako je $f \in F_3$ simbol arnosti $n, n \geq 0$ i ako su $t_1, t_2, \ldots, t_n$ elementi skupa $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$, tada je $f(t_1, t_2, \ldots, t_n)$ element skupa Ω_3.

Elemente skupa Ω_1 nazivamo individualnim termima. Elemente skupa Ω_2 nazivamo klasnim termima, dok elemente skupa Ω_3 nazivamo metodskim termima. Skup svih individualnih terma u kojima se ne pojavljuju promjenljive označavamo sa $U(\Omega_1)$ a svaki elemenat tog skupa nazivamo osnovnim indivudualnim termom. Na konceptualnom nivou, elementi skupa $U(\Omega_1)$ predstavljaju logičke identifikatore individualnih objekata. Objekti predstavljeni "složenim" id-termima, kao što je na

primjer term *adresa(marko, "Podgorica", 81000)*, se pojavljuju u slučaju kada se kompleksni objekat konstruiše od drugih objekata (kakvi su, u primjeru, objekti *marko*, *"Podgorica"* i *81000*). Na isti način se definišu osnovni klasni i metodski termi i skupovi $U(\Omega_2)$ i $U(\Omega_3)$. Neka je $U(\Omega)=U(\Omega_1)\cup U(\Omega_2)\cup U(\Omega_3)$.

Formule u SF-logici se definišu slično formulama u predikatskoj logici.

Definicija 1.4.3.2. (SF-Formula) Skup SF_FOR je najmanji skup takav da :

a) Ako je $pr \in Pred$ predikatsko slovo arnosti n i ako su $t_1, t_2, \ldots, t_n$ termi (to jest $t_1, t_2, \ldots, t_n \in F$), tada je $pr(t_1, t_2, \ldots, t_n)$ elemenat skupa SF_FOR.

b) Ako je $o \in \Omega_1$ i $C \in \Omega_2$, tada je $o{:}C$ elemenat skupa SF_FOR.

c) Ako su $C_1, C_2 \in \Omega_2$, tada je $C_1{::}C_2$ elemenat skupa SF_FOR.

d) Ako je $o \in \Omega_1$, tada je $o[ListaMetodskihIzraza]$ elemenat skupa SF_FOR, gdje je *ListaMetodskihIzraza* lista skalarnih ili skupovnih metodskih izraza medusobno razdvojenih znakom ' ; ' . Ako je $m \in \Omega_3$ (ime metoda) i $o_1, o_2, \ldots, o_n, r, r_1, \ldots, r_k \in \Omega_1, k, n \geq 0$, tada je $m\#o_1, o_2, \ldots, o_n \to r$ skalarni metodski izraz, a $m\#o_1, o_2, \ldots, o_n \to\to \{r_1, \ldots, r_k\}$ je skupovni metodski izraz.

e) Ako je $C \in \Omega_2$, tada je $C[ListaSignaturaMetoda]$ elemenat skupa SF_FOR, gdje je *ListaSignaturaMetoda* lista signatura skalarnih ili skupovnih metoda medusobno razdvojenih znakom ' ; '. Ako je $m \in \Omega_3$(ime metoda) i ako su $C, C_1, C_2, \ldots, C_n, R_1, R_2, \ldots, R_k \in \Omega_2, n, k \geq 0$ (imena klasa), tada je izraz $m\#C_1, C_2, \ldots, C_n \Rightarrow \{R_1, R_2, \ldots, R_k\}$ signatura skalarnog metoda, a izraz $m\#C_1, C_2, \ldots, C_n \Rightarrow\Rightarrow \{R_1, R_2, \ldots, R_k\}$ je signatura skupovnog metoda..

f) Ako su φ i ψ elementi skupa SF_FOR , tada su i $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \leftarrow \psi$ elementi skupa SF_FOR.

g) Ako je φ elemenat skupa SF_FOR i X promjenljiva (tj. $X \in Var$), tada su $(\forall X)\varphi$ i $(\exists X)\varphi$ elementi skupa SF_FOR.

Elemente skup SF_FOR nazivamo SF-formulama ili formulama.

NAPOMENE:

- $\leftarrow$ označava implikaciju : $\varphi \leftarrow \psi$ je drugi način zapisivanja za $\varphi \vee \neg\psi$;
- ako se unutar { } pojavljuje samo jedan term, tada umjesto $P[\ldots \to \{R\}]$ pišemo $P[\ldots \to R]$;
- ako je metod m nularni, tj. ako je m atribut , tada umjesto $P[m\#\to\ldots]$ pišemo samo $P[m \to \ldots]$;
- zbog kompaktnosti zapisa , često se različite formule mogu kombinovati u jednu; na primjer, izraz $X[A \to a; f \to d[m(x,y) \to\to \{b,c\}; A \to t]]$ je kraći način zapisivanja za formulu $X[A \to a; f \to d] \wedge d[m(x,y) \to\to \{b,c\}; A \to t]$; takode, izraz

$X\left[A \Rightarrow K{:}V\left[f \Rightarrow\Rightarrow \{R, Z{:}C\}\right]\right]$ je skraćeni način zapisivanja formule $X[A \Rightarrow K] \wedge K{:}V \wedge V\left[f \Rightarrow\Rightarrow \{R, Z\}\right] \wedge Z{:}C$;

- formule iz tačaka *a), b)* i *c)* definicije 1.4.3.2 su atomske formule, jer se ne mogu razložiti na prostije formule; formule nastale primjenom *f)* i *g)* predstavljaju izraze signature;
- formule iz tačaka *a), b), c), d)* i *e)* definicije 1.4.3.2 nazivaju se elementarne (molekularne) formule ([8]);
- ako su svi funkcijski simboli iz F arnosti nula, tada su jedini termi imena objekta (individualnih, klasnih ili metodskih);
- neformalno, formula $o{:}C$ iskazuje tvrdenje da objekat o pripada klasi C, a formula $C_1{::}C_2$ tvrdenje da je klasa C_1 podklasa klase C_2; izrazi signature definišu tip metoda ili atributa klase to jest, metodski izraz $m\#o_1, o_2, \ldots, o_n \to r$ predstavlja poziv metoda m sa argumentima $o_1, o_2, \ldots, o_n$, koji daje rezultat r;

Definicija 1.4.3.3. Konstitutivni atom elementarne formule φ je :

- ako je $\varphi = o{:}C$ ili $\varphi = C_1{::}C_2$ ili $\varphi = pr(t_1, \ldots, t_n), pr \in Pred$, tada je φ konstitutivni atom;
- ako $\triangleright$ označava ili $\Rightarrow$ ili $\Rightarrow\Rightarrow$ i ako je $\varphi = C\left[m\#C_1, C_2, \ldots, C_n \triangleright \{R_1, \ldots, R_k\}\right]$; tada su $C\left[m\#C_1, C_2, \ldots, C_n \triangleright \{\ \}\right]$ i $C\left[m\#C_1, C_2, \ldots, C_n \triangleright \{R_i\}\right], i = 1, \ldots, k$ konstitutivni atomi;
- ako je $\varphi = o\left[m\#o_1, o_2, \ldots, o_n \to r\right]$, tada je $o\left[m\#o_1, o_2, \ldots, o_n \to r\right]$ konstitutivni atom;
- ako je $\varphi = o\left[m\#o_1, o_2, \ldots, o_n \to\to \{r_1, \ldots, r_k\}\right]$, tada je svaki od izraza $o\left[m\#o_1, o_2, \ldots, o_n \to\to \{\ \}\right]$ i $o\left[m\#o_1, o_2, \ldots, o_n \to\to \{r_i\}\right], \quad i = 1, \ldots, k$ konstitutivni atom;
- ako je $\varphi = o[ListaMetodskihIzraza]$ (ili $\varphi = o[ListaIzrazaSignature]$), tada su konstitutuvni atomi svi konstitutivni atomi svih metodskih izraza (izraza signature) koji se pojavljuju u *ListaMetodskihIzraza* (*ListaIzrazaSignature*).

1.4.4. Semantika SF-Logike

SF-struktura je torka $I = (D, \in, \leq, IPred, IF_1, IF_2, IF_3, IDS, IDM, ITS, ITM)$. Skup $D = D_1 \cup D_2 \cup D_3$ nazivamo domenom strukture I ; skupovi D_1 (skup individualnih objekata), D_2 (skup klasnih objekata) i D_3 (skup metodskih objekata) su međusobno disjunktni. Skup D predstavlja skup svih objekata u realnom svijetu. Svaki term identifikujemo sa nekim objektom iz skupa D. Relacija $\in$ je binarna relacija skupa D, tačnije $\in \subseteq D_1 \times D_2$, a $\leq$ je relacija parcijalnog uređenja na skupu D_2. $\in$ i $\leq$ su povezane na sledeći način : ako $a \in C$ i $C \leq K$, tada $a \in K$. Preslikavanje *IPred* pridružuje svakom n-arnom predikatu p iz skupa *Pred* jednu n-arnu relaciju skupa D, to jest, $IPred: Pred \to \bigcup_{i=1}^{\infty} P(D^i)$. Kao i u predikatskoj logici, svakom funkcijskom slovu arnosti n skupa F pridružujemo jednu totalnu funkciju čiji je domen D^n a kodomen skup D. Formalno, $IF_k : F_k \to \bigcup_{i=1}^{\infty} Total(D_k^i, D_k), k = 1,2,3$.

Preslikavanja IF_1, IF_2 i IF_3 svakom termu pridružuju jedan element skupa D . Međutim, termi mogu da predstavljaju i metode. Metod je funkcija koje se poziva u okviru nekog objekta (objekat domaćin - engl. "host object"), koja može da ima nula ili više argumenata i kao rezultat vraća jedan objekat (ako je metod skalaran) ili skup objekata (ako je metod skupovni). To znači da bi svakom metodskom termu trebalo pridružiti jednu parcijalnu funkciju. Kako je dozvoljeno postojanje promjenljivih čija je oblast važenja skup metoda, pridruživanje funkcije ne vršimo metodskom termu već elementima skupa D. Pored toga, metod može biti pozvan sa različitim brojem argumenata (polimorfizam arnosti - engl. "arity polymorphism"), pa za svaku moguću arnost metoda m moramo dodijeliti odgovarajuću funkciju. Formalno se to postiže preslikavanjem *IDS* koje interpretira skalarni metodski izraz (definicija 2 - tačka *d)*) i koje je definisano na sledeći način : $IDS: D_3 \to \bigcup_{i=0}^{\infty} Partial(D_1^{i+1}, D_1)$. Poziv skupovnog metoda se realizuje funkcijom *IDM* koja interpretira skupovni metodski izraz i definisana je sa : $IDM: D_3 \to \bigcup_{i=0}^{\infty} Partial(D_1^{i+1}, P(D_1))$. Preslikavanja *IDS* i *IDM* svakom elementu skupa D_3 pridružuju beskonačnu torku funkcija parametrizovanih po arnosti (tj. skupom prirodnih brojeva). Za svaki prirodan broj k, $IDS^{(k)}(m)$ je funkcija koja ima k argumenata, to jest metodu m pridružena je jedna k-arna funkcija (u stvari, funkcija $IDS^{(k)}(m)$ je $(k+1)$-arna, jer je prvi argument te funkcije objekat-domaćin). $IDS^{(k)}(m)(o, o_1, \ldots, o_k)$ je poziv k-arne funkcije koja je definisana na objektu o (host - objekat) i čiji su argumenti $o_1, \ldots, o_k$. U objektno-orijentisanim sistemima, izraz $IDS^{(k)}(m)(o, o_1, \ldots, o_k)$ interpretiramo kao zahtjev objektu o da pozove skalarni metod m čiji su argumenti $o_1, \ldots, o_k$.

Pošto metode interpretiramo kao funkcije, tip skalarnog metoda i tip skupovnog metoda (definicija 1.4.4.2 - tačke *f)* i *g)*) interpretiramo kao funkcionalne tipove. Za specifikaciju funkcionalnog tipa potrebno je odrediti tipove argumenata funkcije i tip

rezultata. Kao i kod poziva metoda, moramo voditi računa o polimorfizmu. Funkcionalni tipovi opisani su preslikavanjima $ITS: D_3 \to \bigcup_{i=0}^{\infty} AM(D_2^{i+1}, UP(D_2))$ (za skalarne metode), i $ITM: D_3 \to \bigcup_{i=0}^{\infty} AM(D_2^{i+1}, UP(D_2))$ (za skupovne metode). Značenje izraza $ITS^{(n)}(m)(C, C_1, \ldots, C_n)$ je tip funkcije $IDS^{(n)}(m)(o, o_1, \ldots, o_n)$ (analogno i za skupovne metode). Ako je $ITS^{(n)}(m)(C, C_1, \ldots, C_n)$ definisan, tada je domen te funkcije skup (n+1)-torki klasa $(C, C_1, \ldots, C_n)$ takvih da metod $IDS^{(n)}(m)$ može imati argumente $(o, o_1, \ldots, o_n)$, ako je $(o, o_1, \ldots, o_n) \in (C, C_1, \ldots, C_n)$. Za svaku torku klasa $(C, C_1, \ldots, C_n)$, ako je definisan $ITS^{(n)}(m)(C, C_1, \ldots, C_n)$, tada on predstavlja tip funkcije $v = IDS^{(n)}(m)(o, o_1, \ldots, o_n)$, za sve argumente $(o, o_1, \ldots, o_n) \in (C, C_1, \ldots, C_n)$ (to jest, ako je definisano $v = IDS^{(n)}(m)(o, o_1, \ldots, o_n)$, tada za svaku klasu R iz $ITS^{(n)}(m)(C, C_1, \ldots, C_n)$, važi $v \in R$). Na isti način, ako je $ITM^{(n)}(m)(C, C_1, \ldots, C_n)$ definisan, tada je domen te funkcije skup (n+1)-torki klasa $(C, C_1, \ldots, C_n)$ takvih da metod $IDM^{(n)}(m)$ može imati argumente $(o, o_1, \ldots, o_n)$ samo ako je $(o, o_1, \ldots, o_n) \in (C, C_1, \ldots, C_n)$. Za svaku torku klasa $(C, C_1, \ldots, C_n)$, ako je definisan $ITM^{(n)}(m)(C, C_1, \ldots, C_n)$, tada on predstavlja tip funkcije $IDM^{(n)}(m)(o, o_1, \ldots, o_n)$, za svaku torku individualnih objekta $(o, o_1, \ldots, o_n)$ za koju je $(o, o_1, \ldots, o_n) \in (C, C_1, \ldots, C_n)$ (to jest, postoji $IDM^{(n)}(m)(o, o_1, \ldots, o_n)$ i za svako v iz $IDM^{(n)}(m)(o, o_1, \ldots, o_n)$ i svako R iz $ITM^{(n)}(m)(C, C_1, \ldots, C_n)$, važi $v \in R$).

Ako je $v = IDS^{(n)}(m)(o, o_1, \ldots, o_n)$ definisan i ako je njegov tip klasa R, tada v mora pripadati svakoj klasi G koja je nadklasa klase R $(R \leq G)$. Dakle skup $ITS^{(n)}(m)(C, C_1, \ldots, C_n)$ mora pripadati $UP(D_2)$. Ako je definisan $IDS^{(n)}(m)$ na objektima tipa $(C, C_1, \ldots, C_n)$ (to jest $(o, o_1, \ldots, o_n) \in (C, C_1, \ldots, C_n)$), tada $IDS^{(n)}(m)$ mora biti definisan i na objektima tipa $(K, K_1, \ldots, K_n)$ $((a, a_1, \ldots, a_n) \in (K, K_1, \ldots, K_n))$, gdje je $(K, K_1, \ldots, K_n) \leq (C, C_1, \ldots, C_n)$ i pri tome $IDS^{(n)}(m)(a, a_1, \ldots, a_n)$ mora biti istog tipa kao i $IDS^{(n)}(m)(o, o_1, \ldots, o_n)$ (a to je $ITS^{(n)}(m)(C, C_1, \ldots, C_n)$). Dakle, mora da važi $ITS^{(n)}(m)(C, C_1, \ldots, C_n) \subseteq ITS^{(n)}(m)(K, K_1, \ldots, K_n)$, to jest antimonotonost.

Dodjeljivanje vrijednosti promjenljivim vrši se funkcijom $v: V \to D$, pri čemu važi $v(Var_k) \subseteq D_k$, $k = 1,2,3$. Vrijednost terma dobija se kao i u predikatskoj logici : ako je f individualni term arnosti 0, tada je $v(f) = IF_1(f)$; ako je f individualni term arnosti n, tada je $v(f(t_1, t_2, \ldots, t_n)) = IF_1(f)(v(t_1), v(t_2), \ldots, v(t_n))$. Analogno se definišu vrijednosti klasnog i metodskog terma.

<u>Definicija 1.4.4.1.</u> (Pomoćne definicije) Neka je φ proizvoljna formula.

a) Skup podformula formule φ je najmanji skup X takav da $\varphi \in X$ i :

- ako je $\varphi = \psi \wedge \phi$, tada $\psi \in X$ i $\phi \in X$;
- ako je $\varphi = \psi \vee \phi$, tada $\psi \in X$ i $\phi \in X$;

- ako je $\varphi = \neg\psi$, tada $\psi \in X$;
- ako je $\varphi = \psi \leftarrow \phi$, tada $\psi \in X$ i $\phi \in X$;
- ako je $\varphi = (\forall y)\psi$ ili $\varphi = (\exists y)\psi$, tada $\psi \in X$;

b) Skup slobodnih promjenljivih SP je :

- ako je t term, tada skup $SP(t)$ čine sve promjenljive terma t;
- ako je φ formula oblika $\varphi = p(t_1,\ldots,t_n), p \in Pred$, tada je $SP(\varphi) = SP(t_1)\cup\ldots\cup SP(t_n)$;
- ako je φ formula oblika $\varphi = o:C$, tada je $SP(\varphi) = SP(o)\cup SP(C)$;
- ako je φ formula oblika $\varphi = C_1 : C_2$, tada je $SP(\varphi) = SP(C_1)\cup SP(C_2)$;
- ako je φ formula oblika $\varphi = o[m\#o_1, o_2, \ldots, o_n \to r]$, tada je $SP(\varphi) = SP(o)\cup SP(m)\cup SP(o_1)\cup\ldots\cup SP(o_n)\cup SP(r)$;
- ako je φ formula oblika $\varphi = o[m\#o_1, o_2, \ldots, o_n \to\to r]$, tada je $SP(\varphi) = SP(o)\cup SP(m)\cup SP(o_1)\cup\ldots\cup SP(o_n)\cup SP(r)$;
- ako je φ oblika $\varphi = C[m\#C_1, C_2, \ldots, C_n \Rightarrow \{R_1, R_2, \ldots, R_k\}]$, tada je $SP(\varphi) = SP(C)\cup SP(m)\cup SP(C_1)\cup\ldots\cup SP(C_n)\cup SP(R_1)\cup\ldots\cup SP(R_k)$;
- ako je φ oblika $\varphi = C[m\#C_1, C_2, \ldots, C_n \Rightarrow\Rightarrow \{R_1, R_2, \ldots, R_k\}]$, tada je $SP(\varphi) = SP(C)\cup SP(m)\cup SP(C_1)\cup\ldots\cup SP(C_n)\cup SP(R_1)\cup\ldots\cup SP(R_k)$;
- ako je φ formula oblika $\varphi = \neg\psi$,tada je $SP(\varphi) = SP(\psi)$;
- ako je φ formula oblika $\varphi = \psi \wedge \phi$ ili $\varphi = \psi \vee \phi$ ili $\varphi = \psi \leftarrow \phi$ tada je $SP(\varphi) = SP(\psi)\cup SP(\phi)$;
- ako je φ formula oblika $\varphi = (\forall y)\psi$ ili $\varphi = (\exists y)\psi$, tada je $SP(\varphi) = SP(\psi)\setminus\{y\}$.

c) term t je slobodan za promjenljivu x u formuli φ ako ne postoji promjenljiva y takva da je : $y \in SP(t)$ i $(\forall y)\psi$ (ili $(\exists y)\psi$) je podformula formule φ i $x \in SP((\forall y)\psi)$ (ili $x \in SP((\exists y)\psi)$).

d) (Supstitucija) $(\ )_c^p : Var\cup F\cup SF\_For \to Var\cup F\cup SF\_For$, $p \in Var$

- ako je $x \in Var$: $(x)_c^p = \begin{cases} c, \text{ako je } x = p \\ x, \text{ako je } x \neq p \end{cases}$;
- ako je $f \in F$ term arnosti n : $(f(t_1,\ldots,t_n))_c^p = f((t_1)_c^p,\ldots,(t_n)_c^p)$;
- ako je φ formula oblika $\varphi = pr(t_1,\ldots,t_n), pr \in Pred$ i ako je c slobodno za promjenljivu p u formuli φ, tada je $(pr(t_1,\ldots,t_n))_c^p = pr((t_1)_c^p,\ldots,(t_n)_c^p)$;
- ako je φ formula oblika $\varphi = o:C$ i c je slobodno za promjenljivu p u formuli φ , tada je $(\varphi)_c^p = (o)_c^p : (C)_c^p$;

- ako je φ formula oblika $\varphi = C_1 : C_2$ i c je slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (C_1)_c^p : (C_2)_c^p$;
- ako je φ formula oblika $\varphi = o[m\#o_1, o_2, \ldots, o_n \to r]$ i ako je c slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (o)_c^p[(m)_c^p \#(o_1)_c^p, \ldots, (o_n)_c^p \to (r)_c^p]$;
- ako je φ formula oblika $\varphi = o[m\#o_1, o_2, \ldots, o_n \to\to r]$ i ako je c slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (o)_c^p[(m)_c^p \#(o_1)_c^p, \ldots, (o_n)_c^p \to\to (r)_c^p]$;
- ako je φ oblika $\varphi = C[m\#C_1, C_2, \ldots, C_n \Rightarrow \{R_1, R_2, \ldots, R_k\}]$ i ako je c slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (C)_c^p[(m)_c^p \#(C_1)_c^p, \ldots, (C_n)_c^p \Rightarrow \{(R_1)_c^p, \ldots, (R_k)_c^p\}]$;
- ako je φ oblika $\varphi = C[m\#C_1, C_2, \ldots, C_n \Rightarrow\Rightarrow \{R_1, R_2, \ldots, R_k\}]$ i ako je c slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (C)_c^p[(m)_c^p \#(C_1)_c^p, \ldots, (C_n)_c^p \Rightarrow\Rightarrow \{(R_1)_c^p, \ldots, (R_k)_c^p\}]$;
- ako je φ formula oblika $\varphi = \neg\psi$ i ako je c slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (\psi)_c^p$;
- ako je φ formula oblika $\varphi = \psi \lhd \phi$, gdje $\lhd$ označava $\wedge, \vee$ ili $\leftarrow$ i c je slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (\psi)_c^p \lhd (\phi)_c^p$;
- ako je φ formula oblika $\varphi = (\forall y)\psi$ (ili $\varphi = (\exists y)\psi$) i ako je c slobodno za promjenljivu p u formuli φ, tada je $(\varphi)_c^p = (\forall y)(\psi)_c^p$ (ili $(\varphi)_c^p = (\exists y)(\psi)_c^p$).

<u>Definicija 1.4.4.2.</u> (Valuacija) Valuacija val_v u odnosu na dodjelu vrijednosti promjenljivim v, je funkcija $val_v : SF\_FOR \to \{0,1\}$ koja zadovoljava sledeće uslove :

- ako je $\varphi = o : C$, tada je $val_v(\varphi) = 1$, ako je $v(o) \in v(C)$; $val_v(\varphi) = 0$, inače;
- ako je $\varphi = C_1 : C_2$, tada je $val_v(\varphi) = 1$, ako je $v(C_1) \in v(C_2)$; $val_v(\varphi) = 0$, inače;
- ako je $\varphi = pr(t_1, \ldots, t_n), pr \in Pred$, tada je $val_v(\varphi) = 1$, ako je $(v(t_1), \ldots, v(t_n)) \in Ipred(pr)$; $val_v(\varphi) = 0$, inače;
- ako je $\varphi = o[m\#o_1, o_2, \ldots, o_n \to r]$, tada je : $val_v(\varphi) = 1$ ako je definisano $IDS^{(n)}(v(m))(v(o), v(o_1), \ldots, v(o_n))$ i jednako $v(r)$; $val_v(\varphi) = 0$, inače;
- ako je $\varphi = o[m\#o_1, o_2, \ldots, o_n \to\to \{r_1, r_2, \ldots, r_k\}]$, tada je : $val_v(\varphi) = 1$ ako je definisan $IDM^{(n)}(v(m))(v(o), v(o_1), \ldots, v(o_n))$ i sadrži skup $\{v(r_1), \ldots, v(r_k),\}$; $val_v(\varphi) = 0$, inače;

- ako je $\varphi = C\left[m\#C_1, C_2, \ldots, C_n \Rightarrow \{R_1, \ldots, R_k\}\right]$, tada je : $val_v(\varphi) = 1$ ako je definisan skup $IDS^{(n)}(v(m))(v(o), v(o_1), \ldots, v(o_n))$ i ako sadrži skup $\{v(R_1), \ldots, v(R_k),\}$; $val_v(\varphi) = 0$, inače;
- ako je $\varphi = C\left[m\#C_1, C_2, \ldots, C_n \Rightarrow\Rightarrow \{R_1, \ldots, R_k\}\right]$, tada je : $val_v(\varphi) = 1$ ako je definisan skup $IDS^{(n)}(v(m))(v(o), v(o_1), \ldots, v(o_n))$ i ako sadrži skup $\{v(R_1), \ldots, v(R_k),\}$; $val_v(\varphi) = 0$, inače;
- ako je $\varphi = \psi \wedge \phi$, tada je $val_v(\varphi) = \min(val_v(\psi), val_v(\phi))$;
- ako je $\varphi = \psi \vee \phi$, tada je $val_v(\varphi) = \max(val_v(\psi), val_v(\phi))$
- ako je $\varphi = \neg\psi$, tada je $val_v(\varphi) = 1 - val_v(\psi)$;
- ako je $\varphi = \psi \leftarrow \phi$, tada je $val_v(\varphi) = \max(1 - val_v(\psi), val_v(\phi))$;
- ako je $\varphi = (\forall x)\psi$, tada je $val_v(\varphi) = \min\{val_v(\psi_x^a) : a \in F, a \text{ arnosti } 0\}$;
- ako je $\varphi = (\exists x)\psi$, tada je $val_v(\varphi) = \max\{val_v(\psi_x^a) : a \in F, a \text{ arnosti } 0\}$;

Definicija 1.4.4.3. Formula φ je tačna u strukturi I pri dodjeli vrijednosti v, u oznaci $I|=_v \varphi$, ako je $val_v(\varphi) = 1$. Ako je φ zatvorena formula, tada pišemo $I|= \varphi$ i kažemo da je I model formule φ.

1.4.5. Osobine SF-Struktura

Sledeća tvrdenja važe na skupu $U(\Omega)$:

1. Za svako p, $I|= p = p$;
2. Ako je $I|= p = q$, tada je $I|= q = p$;
3. Ako je $I|= p = q$ i $I|= q = r$, tada $I|= p = r$;
   Osobine 1., 2. i 3. dokazuju da je = relacija ekvivalencije na $U(\Omega)$.
4. Ako je $I|= (p = q) \wedge S$, tada je $I|= (S)_q^p$;
5. $I|= p::p$; - refleksivnost ::;
6. Ako je $I|= p::q$ i $I|= q::r$, tada je $I|= p::r$ - tranzitivnost ::;
7. Ako je $I|= p::q$ i $I|= q::p$, tada je $I|= p = q$ - acikličnost ::;
8. Ako je $I|= p:q$ i $I|= q::r$, tada je $I|= p:r$ - inkluzija podklasa;
9. Ako $\triangleright$ označava ili $\Rightarrow$ ili $\Rightarrow\Rightarrow$ i ako S označava ili $\{\}$ ili $\{R_1, \ldots, R_k\}, Ri \in U(\Omega), i = 1, \ldots, k$ i ako je $I|=C\left[m\#C_1, C_2, \ldots, C_n \triangleright S\right]$ i $I|= K::C$, tada je $I|=K\left[m\#C_1, C_2, \ldots, C_n \triangleright S\right]$ - strukturno nasledivanje (podklasa nasleduje signature metoda od nadklase);

10. Ako $\triangleright$ označava ili $\Rightarrow$ ili $\Rightarrow\Rightarrow$ i ako S označava ili $\{\}$ ili $\{R_1,\ldots,R_k\}, Ri \in U(\Omega), i=1,\ldots,k$ i ako je $I\models C[m\#C_1,C_2,\ldots,C_n \triangleright S]$ i $I\models C_1::K_1, i=1,\ldots,n$, tada je $I\models K[m\#K_1,K_2,\ldots,K_n \triangleright S]$;

11. Ako $\triangleright$ označava ili $\Rightarrow$ ili $\Rightarrow\Rightarrow$ i ako $S \in U(\Omega)$ i ako je $I\models C[m\#C_1,C_2,\ldots,C_n \triangleright S]$ i $I\models S::G$, tada je $I\models C[m\#C_1,C_2,\ldots,C_n \triangleright G]$;

12. Ako je $I\models o[m\#o_1,o_2,\ldots,o_n \to r]$ i $I\models o[m\#o_1,o_2,\ldots,o_n \to g]$, tada je $I\models r=g$;

13. Ako je $t \in U(\Omega)$, tada $I\models t[\,]$;

14. Ako je φ elementarna formula, tada je $I\models\varphi$ ako za svaki konstitutivni atom v formule φ važi $I\models v$.

Sve navedene osobine se veoma jednostavno dokazuju.

1.4.6. Formalna teorija SF-logike

Kao i u predikatskoj logici, moguće je definisati formalnu teoriju SF-Logike. Broj pravila izvodenja je znatno veći nego kod predikatske logike, što je direktna posledica znatno složenije strukture formula SF-Logike.

Skolemove funkcije i preneksna normalna forma Formula φ ima preneksni normalni oblik ako postoji formula ψ bez kvantifikatora takva da $\varphi = Q_1 x_1 \ldots Q_n x_n \psi$, gdje $Q_i, i = 1,\ldots,n$ označava jedan od kvantifikatora $\forall$ ili $\exists$ ([1]). Slično predikatskoj logici, svaka SF-formula se može prevesti u njoj ekvivalentnu formulu u preneksnom normalnom obliku. U SF-logici takode važe De-Morganovi zakoni, pa se svaka formula može transformisati u konjuktivnu ili disjunktivnu normalnu formu. Kako su termi u SF-logici identični termima u predikatskoj logici, svaka formula u preneksnom normalnom obliku može se svesti na ekvivalentnu Skolemovom standardnu formu.

PRIMJER 1.4.6.1.

1. Data je SF-formula $\forall X \exists Y f(X,Y)\left[a \to\to \{b,c\}; m(X) \to g(X,Y)\right]$, koja je u preneksnom normalnom obliku. Skolemova standarna forma date formule je : $\forall X\ f(X,h(Y))\left[a \to\to \{b,c\}; m(X) \to g(X,h(Y))\right]$, gdje je h novi funkcionalni simbol.
2. Skolemova standarna forma formule $\exists X \forall Y \forall Z \exists U \forall V \exists W P(X,Y,Z,U,V,W)$ ima oblik $\forall Y \forall Z \forall V\ P(a,Y,Z,f(Y,Z),V,g(Y,Z,V))$, gdje je a term arnosti 0 a f i g su funkcijska slova arnosti 2 i 3 respektivno.
3. Data je formula $\varphi = \forall X \exists Y \exists Z\ ((\neg p(X,Y) \wedge q(X,Z)) \vee r(X,Y,Z))$, koja je u preneksnom normalnom obliku jer je oblika $\varphi = \forall X \exists Y \exists Z\ \phi$ i ϕ ne sadrži kvantifikatore. Formula ϕ naziva se matricom formule φ. Transformacija u Skolemovu standardnu formu se izvodi u dva koraka: u prvom koraku, primjenom De-Morganovih zakona dobijamo konjuktivnu normalnu formu formule (matrice) ϕ: $\forall X \exists Y \exists Z\ ((\neg p(X,Y) \vee r(X,Y,Z)) \wedge (q(X,Z) \vee r(X,Y,Z)))$. Drugi korak daje traženi oblik: $\forall X\ ((\neg p(X,f(x)) \vee r(X,f(x),g(x))) \wedge (q(X,g(x)) \vee r(X,f(x),g(x))))$. ◊

Disjunkt definišemo kao disjunkciju literala, pa je svaka od formula φ u prethodnim primjerima prikazana kao konjunkcija disjunkta. Formula φ je odredena skupom svojih disjunkta. Lako se dokazuje da važi sledeće tvrdenje : ako je S skup koji predstavlja skup disjunkta Skolemove standardne forme formule φ, tada je φ protivrečna ako i samo ako je skup S protivrečan [17]. U nastavku ovog poglavlja smatraćemo da je formula odredena skupom svojih disjunkta.

Erbranove strukture Neka je L jezik SF-logike, sa skupom funkcijskih slova F. Elemente skupa $U(\Omega)$ nazivamo osnovnim termima. Sve elementarne formule u kojima se ne pojavljuju imena promjenljivih nazivamo osnovnim elementarnim formulama i one čine Erbranovu bazu za jezik L, u oznaci $HB(L)$.

Definicija 1.4.6.1. Skup $H \subseteq HB(L)$ je Erbranova struktura (skraćeno E-struktura) ako je H zatvoren u odnosu na $\models$.

Napomene:

1. Uslov $\models$ - zatvorenosti je neophodan, jer se iz skupa osnovnih elementarnih formula mogu izvesti nove netrivijalne formule; na primjer $\{A::B, B::C\} \models A::C$;
2. Sve osobine SF-struktura se mogu primjeniti i na E-strukture, zamjenjujući $I \models \varphi$ sa $\varphi \in H$; na primjer, ako je $I \models p::q$ i $I \models q::p$, tada je $I \models p = q$ postaje : ako je $p::q \in H$ i $q::p \in H$, tada je $p = q \in H$; slično važi i za ostale osobine 1-14.

Definicija 1.4.6.2. Neka je H E-struktura. Tada je :

- osnovna elementarna formula φ je tačna u strukturi H (u oznaci $H \models \varphi$) ako je $\varphi \in H$;
- osnovna formula $\neg\varphi$ je tačna u strukturi H ako $\varphi \notin H$;
- osnovni disjunkt $L_1 \vee \ldots \vee L_n$ je tačan u H, ako je bar jedan od $L_i, i = 1,\ldots,n$ tačan u H;
- disjunkt C je tačan u H ako su svi osnovni disjunkti koji se dobijaju iz disjunkta C tačni u strukturi H.

Postoji jednostavan način konstrukcije E-struktura iz SF-struktura i obrnuto. Neka je S proizvoljan skup disjunkta i neka je I SF-struktura za S; tada je E-struktura za S skup svih osnovnih elementarnih SF-formula koje su zadovoljene u strukturi I. Ako je H Erbranova struktura za skup disjunkta S, tada se SF-struktura za S dobija na nešto složeniji način. Naime, postoji mogućnost da dva različita elementa iz H označavaju isti objekat.

PRIMJER 1.4.6.2. Ako je *zoran* brat osobe *goran*, tada je *brat*(*goran*)=*zoran*, iako *brat*(*goran*) i *zoran* označavaju različite elemente skupa H.

Rješenje ovog problema je standardno: relacija jednakosti je relacija ekvivalencije, pa se za domen SF-strukture I uzima skup klasa ekvivalencije (faktor skup $H/=$). Označimo sa $[t]$ klasu ekvivalencije elementa $t \in H$. Tada je SF-struktura $I = (D, \in, \leq, IPred, IF_1, IF_2, IF_3, IDS, IDM, ITS, ITM)$ gdje je :

- domen D strukture I je skup $H/=$;
- $[o] \in [C]$, ako je $o:C \in H$;
- $[C_1] \leq [C_2]$, ako je $C_1::C_2 \in H$;
- $Ipred(pr) = \{([t_1],,[t_n]) \mid pr(t_1,\ldots,t_n) \in H\}, pr \in Pred$;
- $IF_1(c) = [c]$, ako je c funkcionalni simbol arnosti 0; slično za IF_2 i IF_3;
- $IF_1(f)(([t_1],\ldots,[t_n])) = [f(t_1,\ldots,t_n)]$, ako je f funkcionalni simbol arnosti n; slično za IF_2 i IF_3;
- $IDS^{(n)}([m])([o],[o_1],\ldots,[o_n]) = \begin{cases} [s], \text{ako je } o[m\#o_1,\ldots,o_n \to s] \in H \\ \text{nedefinisano, u suprotnom} \end{cases}$, $n \in N$;

- $IDM^{(n)}([m])([o],[o_1],\ldots,[o_k]) = \begin{cases} \{[s] \mid o[m\#o_1,\ldots,o_k \rightarrow\rightarrow s] \in H\}, \\ \quad \text{ako } o[m\#o_1,\ldots,o_k \rightarrow\rightarrow \{\ \}] \in H, n \in N; \\ \text{nedefinisano, u suprotnom} \end{cases}$

- $ITS^{(n)}([m])([o],[o_1],\ldots,[o_k]) = \begin{cases} \{[s] \mid o[m\#o_1,\ldots,o_k \Rightarrow s] \in H\}, \\ \quad \text{ako } o[m\#o_1,\ldots,o_k \Rightarrow \{\ \}] \in H, n \in N \\ \text{nedefinisano, u suprotnom} \end{cases}$

- $ITM^{(n)}([m])([o],[o_1],\ldots,[o_k]) = \begin{cases} \{[s] \mid o[m\#o_1,\ldots,o_k \Rightarrow\Rightarrow s] \in H\}, \\ \quad \text{ako } o[m\#o_1,\ldots,o_k \Rightarrow\Rightarrow \{\ \}] \in H, n \in N \\ \text{nedefinisano, u suprotnom} \end{cases}$

Kao i u slučaju predikatske logike, važi Erbranova teorema: skup disjunkta S nije zadovoljiv ako i samo ako postoji konačan skup S' osnovnih instanci skupa S koji nije zadovoljiv ([17]). Ova teorema predstavlja osnovu za formalnu teoriju SF-logike.

Aksiome i pravila izvođenja Formalna teorija SF-logike sastoji se od jedne aksiome i četrnaest pravila izvođenja. Veći broj pravila izvođenja u odnosu na klasičnu predikatsku logiku potiče od složenije strukture SF-formula. Da bi se mogla definisati pravila izvođenja, moramo definisati unifikaciju i supstituciju. Formalna teorija SF-Logike izvedena je iz formalne teorije F-Logike.

Neka je dat jezik L sa skupom terma Ω. Supstitucija je preslikavanje $\sigma: Var \rightarrow \Omega$ koje je identitet svuda osim na nekom konačnom skupu $A \subseteq Var$. Skup A nazivamo domenom supstitucije σ, u oznaci $dom(\sigma)$. Ako je $A = \{p_1,\ldots,p_n\} \subseteq Var$, tada supstituciju σ zapisujemo u sledećem obliku : $\sigma = \{p_1 / t_1,\cdots,p_n / t_n\}$. Supstituciju iz definicije 1.4.4.2.(d) možemo predstaviti sa $\sigma = \{p / t\}$. Na isti način kao i u definiciji 1.4.4.2.(d), supstitucija se proširuje na terme i formule. Kažemo da je supstitucija osnovna, ako je $\sigma(X) \in U(\Omega)$, za svako $X \in dom(\sigma)$. Za datu supstituciju σ i formulu φ, $\sigma(\varphi)$ je instanca formule φ. $\sigma(\varphi)$ je osnovna instanca ako ne sadrži promjenljive.

Neka T i P označavaju terme ili atomske formule. Supstitucija σ je unifikator za T i P ako je $\sigma(T) = \sigma(P)$. Unifikator σ je maksimalan, u oznaci $mgu(\sigma)$ ako za svaki drugi unifikator α za T i P postoji supstitucija β, takva da je $\alpha = \beta \circ \sigma$.

Unifikacija ostalih elementarnih formula je nešto složenija, jer se umjesto zahtjeva da formule postanu identične poslije unifikacije, zahtijeva se da jedna formula bude podformula druge formule ([8]).

Definicija 1.4.6.3. Neka su L i L' elementarne formule oblika $L = S[\ldots], L' = S[\ldots]$. L je elementarna podformula formule L', u oznaci $L \prec L'$, ako je svaki konstitutivni atom formule L istovremeno i konstitutivni atom formule L'. Supstitucija σ je unifikator iz L u L' ako je $\sigma(L) \prec \sigma(L')$.

PRIMJER 1.4.6.3.

1. Neka je $L = S[a\#b \to c], L' = S[x\#y \to d; f\#a,b \to m]$, tada je $\sigma = \{a/x, b/y, c/d\}$ unifikator iz L u L'. Unifikator iz L' u L ne postoji, jer se konstitutuvni atom $f\#a,b \to m$ formule L' ne može prevesti u konstitutuvni atom formule L;
2. Neka je $L = S[a\#b \to c], L' = S[x\#y \to d; f\#e \to g]$; postoje dva unifikatora iz iz L u L' : $\sigma_1 = \{a/x, b/y, c/d\}$ i $\sigma_2 = \{a/f, b/e, c/g\}$. Unifikator iz L' u L je $\sigma_3 = \{x/a, y/b, d/c, f/a, e/b, g/c\}$;
3. Neka je $L = S[a \to\to \{X\}], L' = S[a \to\to \{b,c\}]$; postoje dva unifikatora iz L u L' koje možemo nazvati maksimalnim: $\sigma_1 = \{X/a\}$ i $\sigma_2 = \{X/b\}$. ◊

Neka su L i L' elementarne formule oblika $L = S[\ldots], L' = S[\ldots]$ i neka je UN(L,L') skup svih unifikatora iz L u L'. Na skupu UN(L,L') uvedimo relaciju $\lhd$ na sledeći način: $\alpha \lhd \beta$ ako postoji supstitucija γ takva da je $\beta = \gamma \circ \alpha$. Unifikator $\alpha \in UN(L, L')$ je maksimalan ako za svaki unifikator $\beta \in UN(L, L')$, ako je $\beta \lhd \alpha$, tada je $\alpha \lhd \beta$. Skup $\Sigma \subseteq UN(L, L')$ maksimalnih unifikatora je kompletan ako za svaki unifikator $\alpha \in UN(L, L')$ postoji unifikator $\beta \in \Sigma$ takav da je $\beta \lhd \alpha$.

Pojam unifikatora se na prirodan način proširuje na konačan broj terma ili formula. Na primjer, σ je unifikator elementarnih formula T_1,..., T_n u T ako je $\sigma(T_i) \prec \sigma(T), i = 1, \cdots, n$. Slično, pojam unifikatora se proširuje na uredene torke $(P_1, \cdots, P_n)$ i $(Q_1, \cdots, Q_n)$ na sledeći način : σ je unifikator datih torki ako je $\sigma(P_i) = \sigma(Q_i), i = 1, \cdots, n$. Prije primjene pravila izvodenja, svi disjunkti koji se pojavljuju ne smiju imati zajedničkih promjenljivih, što se postiže preimenovanjem ([14]). Simboli C i C' označavaju disjunkte, L i L' označavaju literale a P, Q , R , S, T označavaju terme. Maksimalni unifikator iz L u L' označavano sa $mgu(L, L')$. Ako jedno pojavljivanje terma T u literalu L zamijeni termom S, tada rezultat zamjene označavamo sa $L[T/S]$. Ako su L i L' literali oblika $L = S[\ldots], L' = S[\ldots]$, tada *merge(L,L')* označava literal čiji skup konstitutivnih atoma čini unija skupova konstitutivnih atoma L i L' i u kome se poziv svakog skupovnog metoda pojavljuje najviše jednom (poziv metoda se sastoji od imena metoda, imena argumenata i tipa poziva - skalarni ili skupovni).

PRIMJER 1.4.6.4.

Dati su literali - elementarne formule L i L' : $A[N \to X, M\#D \to\to H, M \to\to \{P,Q\}]$ i $A[N \to Y, M\#Z \to\to G, M\#D \to\to F]$. Spajanje ova dva literala može se obaviti na dva načina:

$$A[M \to X, M\#D \to\to H, M \to\to \{P,Q\}, N \to Y, M\#D \to\to F, M\#Z \to\to G] \text{ ili}$$

$$A[N \to X, M\#D \to\to \{F,H\}, M \to\to \{P,Q\}, N \to Y, M\#Z \to\to G] \quad (*)$$

Pozivi metoda $M\#D \to\to$, $M\#Z \to\to$ i $M \to\to$ su medusobno različiti. U prvom slučaju, poziv metoda $M\#D \to\to$ se pojavljuje dva puta, pa je rezultat operacije *merge* literal označen sa (*). ◊

Jedina aksioma je : $X::X$. Svako od pravila izvodenja predstavljeno je u obliku $\frac{A_1,\cdots,A_n}{B}$ (iz premisa $A_1,\cdots,A_n$ izvodimo zaključak B). Pravila izvodenja su :

1. $$\frac{\neg L\vee C, L'\vee C', \sigma = mgu(L,L')}{\sigma(C\vee C')}$$

2. $$\frac{L\vee L'\vee C, \sigma = mgu(L,L')}{\sigma(L\vee C)} \qquad \frac{\neg L\vee\neg L'\vee C, \sigma = mgu(L,L')}{\sigma(\neg L\vee C)}$$

3. $$\frac{L[T]\vee C, (T' = T'')\vee C', \sigma = mgu(T,T')}{\sigma(L[T/T'']\vee C\vee C')}$$

4. $$\frac{(P::Q)\vee C, (Q'::P')\vee C', \sigma = mgu((P,Q),(P',Q'))}{\sigma((P=Q)\vee C\vee C')}$$

5. $$\frac{(P::Q)\vee C, (Q'::R)\vee C', \sigma = mgu(Q,Q')}{\sigma((P::R)\vee C\vee C')}$$

6. $$\frac{(P:Q)\vee C, (Q'::R)\vee C', \sigma = mgu(Q,Q')}{\sigma((P:R)\vee C\vee C')}$$

7. $$\frac{P[M\#T_1,\cdots,T_n\Rightarrow R]\vee C, (Q::S)\vee C', \sigma = mgu(P,S)}{\sigma(S[M\#T_1,\cdots,T_n\Rightarrow R]\vee C\vee C')}$$

8. $$\frac{P[M\#T_1,\cdots,T_n\Rightarrow\Rightarrow\{R_1,\cdots,R_k\}]\vee C, (Q::S)\vee C', \sigma = mgu(P,S)}{\sigma(S[M\#T_1,\cdots,T_n\Rightarrow\Rightarrow\{R_1,\cdots,R_k\}]\vee C\vee C')}$$

9. $$\frac{P[M\#T_1,\cdots T_i,\cdots,T_n\Rightarrow R]\vee C, (Q::S)\vee C', \sigma = mgu(T_i,S)}{\sigma(P[M\#T_1,\cdots,Q,\cdots,T_n\Rightarrow R]\vee C\vee C')}$$

10. $$\frac{P[M\#T_1,\cdots T_i,\cdots,T_n\Rightarrow\Rightarrow\{R_1,\cdots,R_k\}]\vee C, (Q::S)\vee C', \sigma = mgu(T_i,S)}{\sigma(P[M\#T_1,\cdots,Q,\cdots,T_n\Rightarrow\Rightarrow\{R_1,\cdots,R_k\}]\vee C\vee C')}$$

11. $$\frac{P[M\#T_1,\cdots,T_n\Rightarrow R]\vee C, (Q::S)\vee C', \sigma = mgu(R,Q)}{\sigma(P[M\#T_1,\cdots,T_n\Rightarrow S]\vee C\vee C')}$$

12. $$\frac{P[M\#T_1,\cdots,T_n\Rightarrow\Rightarrow\{R_1,\cdots,R_i,\cdots,R_k\}]\vee C, (Q::S)\vee C', \sigma = mgu(R_i,Q)}{\sigma(P[M\#T_1,\cdots,T_n\Rightarrow\Rightarrow\{R_1,\cdots,S,\cdots,R_k\}]\vee C\vee C')}$$

13. $$\frac{P[M\#T_1,\cdots,T_n\to R]\vee C, P'[M'\#T_1',\cdots,T_n'\to R']\vee C', \sigma = mgu(\vec{T},\vec{T'})}{\sigma((R=R')\vee C\vee C')}$$, gdje je

$\vec{T} = (P, M, T_1,\cdots,T_n)$ i $\vec{T'} = (P', M', T_1',\cdots,T_n')$

14. $$\frac{P[\ldots]\vee C, Q[\ldots]\vee C', \sigma = mgu(P,Q), L = merge(\sigma(P[\ldots]),\sigma(Q[\ldots]))}{L\vee\sigma(C\vee C')}$$

15. $\dfrac{\neg P[\,] \vee C}{C}$

PRIMJER 1.4.6.5.

Neka su *a, b, c, d, e* i *g* imena klasa, *met*, *atr1* i *atr2* imena atributa i metoda, *o, r* i *f* konstruktori individualnih objekata arnosti 0, 0 i 1 respektivno i *p* predikatsko slovo arnosti 1. Takode, neka su X, Z, O i S promjenljive. Polazni skup disjunkta čine formule 1-7 , dok je proces izvodenja prikazan u tačkama 8-13 :

1. $a :: b$
2. $r : a$
3. $p(r)$
4. $c[m\#b \Rightarrow \{e,g\}; atr1 \Rightarrow\Rightarrow d]$
5. $o[atr2 \rightarrow r]$
6. $o[atr2 \rightarrow f(S)] \vee \neg p(X) \vee \neg O[M\#X \Rightarrow S]$
7. $\neg p(f(Z))$
8. $o[atr2 \rightarrow f(S)] \vee \neg O[M\#a \Rightarrow S]$ - iz 3. i 6. , na osnovu pravila izvodenja 1, gdje je $\sigma = \{X / r\}$. Formalno : $\dfrac{p(r), \neg p(X) \vee r[atr2 \rightarrow f(S)] \vee \neg O[M\#X \Rightarrow S], \sigma = \{X / r\}}{r[atr2 \rightarrow f(S)] \vee \neg O[M\#r \Rightarrow S]}$.
9. $c[m\#a \Rightarrow \{e,g\}]$ - iz 1. i 4. , na osnovu pravila 10. Formalno , $\dfrac{c[m\#b \Rightarrow \{e,g\}], a :: b}{c[m\#a \Rightarrow \{e,g\}]}$.
10. $o[atr2 \rightarrow f(e)]$ - iz 8. i 9., na osnovu pravila 1, gdje je $\sigma = \{O / c, M / m, S / e\}$. Dato σ je unifikator atoma $O[M\#a \Rightarrow S]$ u atom $c[m\#a \Rightarrow \{e,g\}; atr1 \Rightarrow\Rightarrow d]$ (primjer nesimetričnog unifikatora).
11. $r = f(e)$ - iz 5. i 10., na osnovu pravila 13.
12. $p(f(e))$ - iz 3. i 11.
13. $\perp$ - iz 6. i 12., na osnovu pravila 1, gdje je $\sigma = \{Z / e\}$. ◊

2. Upitni jezik

Uprkos činjenici da objektno-orijentisani model podataka prirodno vodi ka navigacionom modelu izračunavanja upita, postoji više upitnih jezika kojim se mogu zadavati upiti na deklarativan način, kao što su XSQL ([7]), ESQL2 (extended SQL2), OQL (Object Query Language) i drugi. Jezik QLO (engl. - "query language for objects") definisan u ovom poglavlju predstavlja kombinaciju XSQL-a, SQL-a za Microsoft SQL Server i nekih karakteristika iz upitnog jezika za O_2 ([4])koje nisu prisutne u ovim jezicima (rekurzija, specificiranje fizičkog grupisanja objekata putem klastera i dodjeljivanje imena objektima).

2.1. Složeni izrazi

Ideja složenog izraza ("path-expression") postoji već duži niz godina i imala je različite implementacije. Pojam složenog izraza izložen u ovom tekstu oslanja se na definiciju iz jezika XSQL [2], sa nekim manjim izmjenama u sintaksi. Osnovna ideja u XSQL vezana za složene izraze je uvođenje selektora koji specificiraju podatke ili djelove šeme, čime se postiže jednostavnije formiranje upita. Složeni izrazi tretiraju atribute i metode na uniforman način i dozvoljavaju upotrebu promjenljivih čiji domeni nisu samo individualni objekti već i imena klasa, metoda i atributa. Takođe, na vrlo jednostavan način se postiže "peglanje" ugnježđenih struktura pa nije potrebno uvoditi operatore tipa *Unnest* ili *Collapse* koji su potrebni kod ugnježđenih relacija ili nekih objektnih sistema (Jasmin [5]).

Složeni izraz opisuje staze duž kompozicionog grafa i možemo ga smatrati kompozicijom metoda. Primjeri složenih izraza zasnovani su na šemi koja je data u poglavlju 2.4.

PRIMJER 2.1.1. Složeni izraz *sony.Sjediste.Ulica* opisuje stazu koja počinje u objektu *sony* klase Kompanija, prolazi kroz sjedište te kompanije (što je objekat klase Adresa) i završava u ulici sjedišta kompanije. U ovom slučaju, *sony* je selektor a Sjediste i Ulica su atributski izrazi. ◊

U opštem slučaju, složeni izraz može da ima znatno komplikovaniji oblik jer uključuje i pozive metoda, selektore koji mogu biti promjenljive, itd. Formalnije, složeni izraz je oblika $sel_0.MetIzraz_1[sel_1]\cdots.MetIzraz_n[sel_n]$, gdje je $n \geq 0$ i svaki od selektora, osim prvog, je opcionalan . Selektor može biti ili OID ili individualna promjenljiva (promjenljiva čiji je domen skup OID-a individualnih objekata) ili ID-funkcija (funkcija koja definiše OID). Svaki od izraza $MetIzraz_i$, $i = 1,\cdots,n$ je ili ime atributa ili poziv metoda ili promjenljiva čiji je domen skup imena atributa. Opšti oblik izraza $MetIzraz_i$, $i = 1,\cdots,n$ ima sledeći oblik :

- $meth(a_1,\ldots,a_n)$, gdje je *meth* ime metoda a $a_1,\ldots,a_n$ argumenti metoda;
- *attr* , gdje je *attr* ime atributa;

- *&ime_promjenljive* , gdje *ime_promjenljive* označava promjenljivu čiji je domen skup imena atributa/metoda.

PRIMJER 2.1.2. Neki primjeri složenih izraza :

(a) X.Stan.Grad
(b) Petar.Clanovi_Porodice.Godine
(c) X.Vozila_U_Vlasnistvu[Y].Boja[Z]
(d) X.BrojClanovaPorodice('Marko Markovic')
(e) X.&Y.Grad
(f) Kompanija.Predsjednik[P].BrojClanovaPorodice(P.Ime)[W]

U primjeru 2.1.2.(a), X je promjenljiva koja označava objekat klase Osoba (tj. domen promjenljive X je klasa Osoba), dok su Stan i Grad atributi klasa Osoba i Adresa, respektivno. U primjeru 2.1.2.(b), Petar je OID nekog objekta klase Osoba (ili klase Zaposleni), dok je Clanovi_Porodice skupovni atribut klase Osoba. BrojClanovaPorodice je metod definisan u klasi Osoba čiji je argument string a rezultat cio broj. Primjeri 2.1.2.(d) i 2.1.2.(f) ilustruju upotrebu metoda u složenim izrazima. Argument metoda u primjeru 2.1.2.(d) je literal (znakovna konstanta) dok je u primjeru 2.1.2.(f) argument metoda drugi složeni izraz (tj. izraz P.Ime). U primjeru 2.1.2.(e), početni selektor je promjenljiva čiji je domen klasa Osoba, dok znak '&' ispred promjenljive Y označava da je domen promjenljive Y skup imena atributa/metoda. ◊

Definicija 2.1.1. Niz objekata $o_0, o_1, \cdots, o_n$ $(n \geq 0)$ iz baze podataka zadovoljava složeni izraz $sel_0.MetIzraz_1[sel_1]. \cdots .MetIzraz_n[sel_n]$ (1) ako je :

- $o_0 = sel_0$
- za svako i = 1, 2, ... , n , ako postoji sel_i tada je $o_i = sel_i$
- za svako i = 1, 2, ... , n , važi :
  - ♦ ako je $MetIzraz_i = meth_i$ tada je $meth_i$ definisan na argumentima $a_{i,1}, a_{i,2}, \cdots, a_{i,k_i}$ u okviru objekta o_{i-1} (tj. $meth_i$ je definisan u klasi C kojoj pripada objekat o_{i-1} i argumenti metoda su $a_{i,1}, a_{i,2}, \cdots, a_{i,k_i}$); ako je $meth_i$ skalaran tada je o_i rezultat poziva metoda sa navedenim argumentima tj. $o_i = meth_i(a_{i,1}, a_{i,2}, \cdots, a_{i,k_i})$; ako je metod skupovni, tada objekat o_i pripada skupu koji je rezultat poziva metoda sa navedenim argumentima tj. $o_i \in meth_i(a_{i,1}, a_{i,2}, \cdots, a_{i,k_i})$;
  - ♦ ako je $MetIzraz_i = attr_i$, tada je $attr_i$ ime atributa koji je definisan u klasi C kojoj pripada objekat o_{i-1}; ako je $attr_i$ skalaran, tada je o_i vrijednost atributa $attr_i$ objekta o_{i-1}; ako je $attr_i$ skupovni, tada o_i pripada skupu vrijednosti atributa $attr_i$ objekta o_{i-1};
  - ♦ ako je $MetIzraz_i$ = &X, tada promjenljiva X uzima vrijednosti imena atributa i metoda klase C kojoj pripada objekat o_{i-1}.

Definicija 2.1.2. Vrijednost složenog izraza (1) je skup svih objekata o_n takvih da niz objekata $o_0, o_1, o_2, \cdots, o_n$ zadovoljava složeni izraz (1).

Ako su svi metodi (ili atributi) u složenom izrazu skalarni, tada je vrijednost izraza najviše jedan OID; ako je bar jedan od metoda (ili atributa) skupovni, vrijednost izraza je skup OID-a (koji može biti i prazan ili jednoelementan).

PRIMJER 2.1.3. Pretpostavimo da klasa Osoba ima samo tri instance čiji su identifikatori o_1, o_2 i o_3. Neka je vrijednost atributa Ime objekta o_1 string 'Petar Popović', vrijednost atributa Stan je objekat a_1 klase Adresa, a vrijednost atributa Grad objekta a_1 je string 'Podgorica'. Drugim riječima, osoba sa imenom 'Petar Popović' stanuje u Podgorici (vrijednosti ostalih atributa nisu nam, za sada, bitne). Na isti način, neka je :

o_2.Ime = 'Marko Marković', o_2.Stan = a_2, a_2.Grad = 'Beograd'

o_3.Ime = 'Janko Janković', o_3.Stan = a_3, a_3.Grad = 'Budva'

Fiksirajmo vrijednost promjenljive X u složenom izrazu X.Stan.Grad iz primjera 5.(a), na primjer X = o_1. Dati složeni izraz zadovoljava sledeći niz objekata (o_1,a_1,'Podgorica') , dok je vrijednost izraza objekat (string) 'Podgorica'. Slično, ako je X=o_2 , tada niz objekata (o_2, a_2, 'Beograd') zadovoljava dati izraz, dok je vrijednost izraza objekat 'Beograd'. Na isti način se dobija da je vrijednost datog složenog izraza 'Budva', u slučaju kada je X = o_3.

Pretpostavimo da je objektu o_1 na neki način dodijeljeno ime Petar i da pored navedenih vrijednosti atributa za o_1 važi i sledeće :

o_1.Clanovi_porodice = {o_4,o_5}

o_4.Godine = 12 , o_5.Godine = 35

o_1.Vozila_U_Vlasnistvu = {v_1,v_2,v_3}

v_1.Model = 'Zastava 128' , v_1.Boja = 'Crvena'

v_2.Model = 'Zastava 101' , v_2.Boja = 'Plava'

v_3.Model = 'Peugeot 405' , v_3.Boja = 'Bijela'

Identifikatori v_1, v_2 i v_3 su logički identifikatori objekata koji pripadaju klasi Vozilo, a o_4 i o_5 su identifikatori objekata klase Osoba. Složeni izraz Petar.Clanovi_Porodice.Godine zadovoljavaju dva niza objekata: (o_1, o_4, 12) i (o_1, o_5, 35) a vrijednost tog izraza je skup objekata {12, 35}. Složeni izraz X.Vozila_U_Vlasnistvu[Y].Boja[Z] iz primjera 5.(c), za fiksiranu vrijednost promjenljive X = o_1 , zadovoljavaju sledeća tri niza objekata : (o_1, v_1, 'Crvena'), (o_1, v_2 , 'Plava') i (o_1, v_3 , 'Bijela') . Vrijednost ovog složenog izraza (za fiksirano X = o_1) je skup objekata {'Crvena','Plava','Bijela'}. Za svaki niz objekata koji zadovoljava dati složeni izraz, promjenljive Y i Z dobijaju vrijednosti kako je naznačeno u definiciji 1. Tako Y uzima vrijednosti iz skupa {v_1, v_2, v_3} a selektor Z iz skupa {'Crvena','Bijela','Plava'} i to na sledeći način : ako je Y = v_1, tada je Z = 'Crvena'; ako je Y = v_2, tada je Z = 'Plava'; ako je Y = v_3, tada je Z = 'Bijela'. ◊

Složeni izrazi mogu biti upoređivani koristeći standardne operatore poređenja (<, >, <=, <>, itd.) ili skupovne operatore (IN, SUBSET, CONTAIN). Kako u opštem slučaju složeni izraz predstavlja skup, moguće su i standardne skupovne operacije (UNION, INTERSECT, DIFFERENCE) nad složenim izrazima.

2.2. Pregled upitnog jezika QLO

Upitni jezik QLO sastoji se iz dva dijela : jezika za definisanje šeme (kreiranje i brisanje klasa, dodavanje i brisanje nadklasa/podklasa, dodavanje i brisanje atributa i metoda, itd.) i jezika za manipulisanje objektima (selekcija objekata iz jedne ili više klasa, dodavanje i brisanje individualnih objekata, mijenjanje vrijednosti atributa, itd.). Naredbe manipulativnog dijela jezika su :

- naredba za selekciju objekata iz jedne ili više klasa (SELECT-naredba)
- naredba za promjenu jedne ili više vrijednosti atributa objekata neke klase (UPDATE-naredba)
- naredba za brisanje instanci klase (DELETE-naredba)
- naredba za kreiranje instance (NEW-naredba)
- naredba za dodjeljivanje imena (NAME-naredba)

Svaka od navedenih naredbi (osim SELECT-naredbe u nekim slučajevima) kao rezultat daje ili vrši promjene na instancama klasa koje je definisao korisnik tj. pomoću ovih naredbi nije moguće mijenjati konceptualnu šemu baze podataka. Sistemske klase se u manipulativnim naredbama koriste samo za provjeru sintaksne korektnosti upita kao i za provjeru tipova atributa i metoda korišćenih u upitu.

Osnovna naredba manipulativnog dijela jezika je SELECT-naredba, koja ima sledeći oblik:

SELECT *ListaIzraza*
FROM *ListaKlasa*
OID OF *ListaPromjenljivih*
WHERE *Predikat*

FROM-blok, OID OF- blok i WHERE-blok u naredbi su opcionalni. Značenje ove naredbe je sledeće : iz liste klasa navedene u FROM-bloku izaberi one instance koje zadovoljavaju *Predikat* naveden u WHERE-bloku (ako on postoji) i prikaži vrijednosti onih atributa ili metoda tih instanci koji su navedeni u *ListiIzraza*. Svaki od izraza u *ListiIzraza* predstavlja jedan složeni izraz. Ako postoji OID OF-blok, tada se kreira novi objekat čiji se OID dobija kao funkcija OID-a navedenih u *ListiPromjenljivih. Predikat* je logički izraz koji se sastoji od složenih izraza, operatora poredenja i skupovnih operatora. Pored manipulisanja korisničkim klasama, pomoću SELECT-naredbe možemo da otkrivamo (ali ne i da mijenjamo) strukturu baze podataka (tj. konceptualnu šemu), koristeći klasne i atributske promjenljive.

Osnovni oblik naredbe za promjenu vrijednosti atributa (UPDATE-naredbe) ima sledeći oblik :

UPDATE *ImeKlase*
SET *ListaDodjeljivanja*
WHERE *Predikat*

Kao i kod SELECT-naredbe, WHERE-blok je opcionalan. Značenje ove naredbe je sledeće : za sve instance klase *ImeKlase* koje zadovoljavaju *Predikat,* atributi koji su

navedeni u *ListiDodjeljivanja* dobijaju navedene vrijednosti. Ako ne postoji *Predikat* , tada se promjena primjenjuje na sve instance klase.

Opšti oblik DELETE-naredbe ima sledeći oblik :

DELETE FROM *ImeKlase*
WHERE *Predikat*

Kao i ranije, WHERE - blok je opcionalan. Ovom naredbom se vrši brisanje svih instanci iz *ImeKlase* koje zadovoljavaju navedeni *Predikat.* Ako predikat ne postoji, brišu se sve instance date klase.

Naredbom za dodjeljivanje imena (NAME-naredbom) pojedinačnom objektu pridružuje se ime, koje je u stvari logički identifikator i može biti upotrebljeno kao referenca u nekom drugom upitu. Opšti oblik ove naredbe je :

NAME ImeObjekta = NEW ImeKlase

Deklarativni dio jezika omogućava kreiranje i brisanje klasa, dodavanje i odbacivanje atributa i metoda klase kao i njenih podklasa, nadklasa i klasa preklapanja tj. kreiranje i mijenjanje konceptualne šeme baze podataka. Pored toga je omogućeno kreiranje i brisanje indeksa i specifikacija načina fizičkog smještanja podataka preko klastera. Naredbe za kreiranje indeksa i klastera daju mogućnost administratoru baze podataka (ili programeru) da utiče na fizičku organizaciju baze. Naredbe deklarativnog dijela jezika su :

- CREATE naredba - kreiranje klase, indeksa, klastera;
- DROP naredba - brisanje klase, atributa, metoda, nadklase, podklase, itd;
- ADD naredba - suprotno DROP naredbi : dodavanje atributa ili metoda, dodavanje nadklase ili podklase, itd.

2.3. Gramatika upitnog jezika

Sintaksa upitnog jezika definisana je sledećom kontekstno-slobodnom gramatikom. Zbog preglednosti, pravila gramatike su grupisana.

Definicija naredbe

Naredba ::= NaredbaManipulacije | NaredbaDefinisanjaSheme
NaredbaManipulacije ::= SelectNaredba | UpdateNaredba | DeleteNaredba
| NameNaredba
NaredbaDefinisanjaSheme ::= CreateNaredba | AddNaredba | DropNaredba

Definicija Select naredbe

SelectNaredba ::= SELECT SelectLista FromOpc WhereOpc
SelectLista ::= SelectBlok | SelectLista, SelectBlok
SelectBlok ::= SelectIzraz AliasOpc
AliasOpc ::= AS alias | ε

SelectIzraz ::= PathIzraz | {PathIzraz}
PathIzraz ::= GlavniSelektor AtributskiIzrazOpc | Func(PathIzraz)
GlavniSelektor ::= ID | ImeKlase
AtributskiIzrazOpc ::= AtributskiIzrazOpc . MetodIliAtribut | ε
Func ::= COUNT | SUM | MAX | MIN | AVG
Alias ::= ID

Definicija From dijela

FromOpc ::= FROM FromLista OidFunkcijaOpc | ε
FromLista ::= FromBlok | FromLista, FromBlok
FromBlok ::= ImeKlase PomocnoImeOpc
PomocnoImeOpc ::= ID | ε
ImeKlase ::= IME_KLASE
OidFunkcijaOpc ::= OID OF OidLista | ε
OidLista ::= ID | OidLista, ID

Definicija logičkog dijela (WHERE - blok) upita

WhereOpc ::= WHERE WhereUslov | ε
WhereUslov ::= Term | WhereUslov OR Term
Term ::= Faktor | Term AND Faktor
Faktor ::= NotOpc Prim
Prim ::= Predikat | (WhereUslov)
NotOpc ::= NOT | ε
Predikat ::= Poredjenje | ClanPredikat | PodskupPredikat | NadskupPredikat
ClanPredikat ::= Izraz NotOpc IN Izraz
PodskupPredikat ::= Izraz NotOpc SUBSET Izraz
NadskupPredikat ::= Izraz NotOpc CONTAINS Izraz
Poredjenje ::= KvantOpc Izraz OperatorPoredjenja KvantOpc Izraz
KvantOpc ::= FORALL | EXISTS | ε
OperatorPoredjenja ::= < | <= | <> | >= | >

Definicija složenog izraza

Izraz ::= AritmetickiIzraz
AritmetickiIzraz ::= AritmetickiIzraz AddOp AritmetickiTerm
| AritmetickiTerm
AritmetickiTerm ::= AritmetickiTerm MultOp AritmetickiFaktor
| AritmetickiFaktor
AritmetickiFaktor ::= (AritmetickiIzraz) | ImpJoinIzraz
AddOp ::= + | -
MultOp ::= * | /
ImpJoinIzraz ::= PocetniSelektor ListaMetodSelektorOpc
| Func(ImpJoinIzraz) | (SelectNaredba)
PocetniSelektor ::= Selektor | KlasnaPromjenljiva | OidFunkcija(ListaArgumenata)

```
KlasnaPromjenljiva ::= $ID
ListaMetodSelektorOpc ::= ListaMetodSelektorOpc MetodskiBlok | ε
MetodskiBlok ::= Izbor SelektorOpc
Izbor ::= MetodIliAtribut | PrefiksOpc ID | *(NizAtributa)
SelektorOpc ::= [Selektor] | ε
MetodIliAtribut ::= ImeAtributa | PrefiksOpc ID
                  | ImeMetoda(ListaArgumenata) | *(NizAtributa)
ImeAtributa ::= IME_ATRIBUTA
ImeMetoda ::= IME_METODA
PrefiksOpc ::= $ | * | & | ε
NizAtributa ::= MetodIliAtribut | MetodIliAtribut . ImeAtributa
Argument ::= Oid | PathIzraz
ListaArgumenata ::= Argument OstaliArgumenti | ε
OstaliArgumenti :: = OstaliArgumenti , Argument | ε
Selektor ::= Oid | ID | OidFunkcija(ListaArgumenata)
Oid ::= BROJ | LITERAL | {ListaBrojevaIliLiterala}
ListaBrojevaIliLiterala ::= ListaBrojeva | ListaLiterala
ListaBrojeva ::= BROJ | ListaBrojeva , BROJ
ListaLiterala ::= LITERAL | ListaLiterala , LITERAL
OidFunkcija ::= ImePogleda
```

Definicija Delete naredbe

```
DeleteNaredba ::= DELETE FROM ImeKlase PomocnoImeOpc
                  WhereOpc
                      | DELETE ImenovaniObjekat
ImenovaniObjekat ::= ID
```

Definicija Update naredbe

```
UpdateNaredba ::= UPDATE CLASS ImeKlase
                  SET ListaDodjela WhereOpc
                      | UPDATE ImenovaniObjekat SET ListaDodjela
ListaDodjela ::= ImeAtributa = Izraz
                 | ListaDodjela , ImeAtributa = Izraz
```

Definicija Create naredbe

```
CreateNaredba ::= CREATE CLASS ImeKlase (AtributiOpc NadklaseOpc
                  PodklaseOpc DisjunktneKlaseOpc KlasePreklapanjaOpc
                      OpisMetodaOpc PreimenovanjeOpc )
                  | CREATE VIEW ImePogleda AS (SelectNaredba)
                  | CREATE INDEX ImeIndeksa ON CLASS ImeKlase
                    AS ListaAtributa
                  | CREATE CLUSTER ON ImeKlase AS OpisClustera
```

ListaAtributa ::= ImeAtributa | ListaAtributa . ImeAtributa
ImePogleda ::= ID
ImeIndeksa ::= ID
NadklaseOpc ::= SUPERCLASSES ListaKlasa | ε
PodklaseOpc ::= SUBCLASSES ListaKlasa | ε
DisjunktneKlaseOpc ::= DISJOINT CLASS ListaKlasa | ε
KlasePreklapanjaOpc ::= OVERLAP CLASS ListaKlasa | ε
ListaKlasa ::= ImeKlase | ListaKlasa , ImeKlase
OpisClustera ::= (ListaGrana)
ListaGrana ::= Grana | ListaGrana , Grana
Grana ::= ImeAtributa | ImeAtributa (ListaGrana)

Definicija Name naredbe

NameNaredba ::= NAME ImenovaniObjekat AS NEW ImeKlase
ImenovaniObjekat ::= ID

Definicija atributa klase

AtributiOpc ::= ATTRIBUTES : (OpisAtributa) | ε
OpisAtributa ::= DefinicijaAtributa | OpisAtributa ; DefinicijaAtributa
DefinicijaAtributa ::= ImeAtributa TipAtributa DefinicijaAgregata
NasledjujeOpc SlozeniObjektOpc DefaultOpc
TipAtributa ::= $\rightarrow$ | $\rightarrow\rightarrow$
DefinicijaAgregata ::= ImeKlase | (OpisAtributa)
NasledjujeOpc ::= INHERITS ImeKlase | ε
SlozeniObjekatOpc ::= COMPLEX EksluzivnostOpc ZavisnostOpc | ε
EksluzivnostOpc ::= SHARED | ε
ZavisnostOpc ::= DEPENDENT | ε
DefaultOpc ::= DEFAULT Oid | ε

Definicija metoda klase

OpisMetodaOpc ::= METHODS ListaMetoda | ε
ListaMetoda ::= SignaturaMetoda | SignaturaMetoda ; ListaMetoda
SignaturaMetoda ::= ImeMetoda(ListaArgumentKlasa) TipMetoda Rezultat
TipMetoda ::= TipAtributa
Rezultat ::= ImeKlase | { ListaKlasa }
ListaKlasa ::= ImeKlase | ListaKlasa , ImeKlase
ArgImeKlase ::= ImeKlase | { ImeKlase }
ListaArgumentKlasa ::= ArgImeKlase | ListaArgumentKlasa , ArgImeKlase

Definicija Add naredbe

AddNaredba ::= ADD MetaObjekat TO CLASS ImeKlase
AddNaredba ::= ADD METHOD SignaturaMetoda
TO CLASS ImeKlase DefinicijaNovogMetoda
MetaObjekat ::= ATTRIBUTE DefinicijaAtributa
| DodatniObjekat ImeKlase
DodatniObjekat ::= SUPERCLASS | SUBCLASS | OVERLAP CLASS
| DISJOINT CLASS
DefinicijaNovogMetoda ::= ImeMetoda(ListaParametara)=Vrijednost
Implementacija
ListaParametara ::= ImeArgumenta | ListaParametara , ImeArgumenta
Implementacija ::= CODE AS Program
Program ::= (Naredba)
| FILE PutDoKoda LANGUAGE ImeProgramskogJezika

Definicija Drop naredbe

DropNaredba ::= Drop NepotrebniObjekat FROM CLASS ImeKlase
NepotrebniObjekat ::= ATTRIBUTE ImeAtributa
| METHOD ImeMetoda
| CLASS ImeKlase
| DodatniObjekat ImeKlase

Definicija preimenovanja atributa i metoda

PreimenovanjeOpc ::= RENAME SpecifikacijaAtributa AS ImenovaniObjekat
SpecifikacijaAtributa ::= ImeAtributa | ImeKlase . ImeAtributa | ImeMetoda
| ImeKlase . ImeMetoda

Napomena : Terminalni simboli gramatike predstavljeni su velikim slovima. Simbol '|' ima ulogu znaka disjunkcije tj. pravilo gramatike oblika A ::= B | C je zamjena za dva pravila : A ::= B i A ::= C. Simbol ε označava praznu riječ (riječ dužine nula).

2.4. Primjer konceptualne šeme baze podataka

Dat je primjer šeme jedne baze podataka. Klase su opisane na neformalan način (bez upotrebe naredbe CREATE CLASS). Svi primjeri upita iz poglavlja 2.4. odnose se na ovu bazu podataka.

klasa Osoba (Ime → String ;
Godine → Integer ;
Stan → Adresa ;
Vozila_U_Vlasnistvu →→ Vozilo))

klasa Zaposleni ((Kvalifikacija →→ String ;

```
                    Plata → Double;
                    Clanovi_porodice →→ Osoba )
                    Zavisni →→ Osoba
                    Radi_Na_Projektu →→ Projekat)
                    Metodi : Zajednicki_Rad(Zaposleni) →→ Projekat ;
                    Nadklase : Osoba )

klasa Adresa (    (Ulica → String ;
                    Grad → String;
                    Drzava → String )
                    Telefon → String)
              )

klasa Vozilo (    ( Model → String ;
                    Proizvodjac → Kompanija;
                    Tezina → Double;
                    Boja → String))

klasa Biciklo (    ( Velicina → Integer )
                    Nadklase : Vozilo )

klasa Auto ( (Motor → MotorVozila;
              Karoserija → AutoKaroserija)
                Nadklase : Vozilo)

klasa Motocikl (    (Motor → MotorVozila)
                    Nadklase : Vozilo )

klasa MotorVozila ( (Masina → KlipniMotor ;
                    Prenos → String)

klasa KlipniMotor ( (Snaga → Integer ;
                    Kubikaza → Integer;
                    Broj_Cilindara → Integer )

klasa DvocilindricniMotor ( (Smjesa → String)
                            Nadklase : KlipniMotor )

klasa CetvorocilindricniMotor ( (Kompresija → String)
                                Podklase : TurboMotor, Dizelmotor ;
                                Nadklase : KlipniMotor )

klasa TurboMotor ( (Ubrizgavanje → String)
                    Nadklase : CetvorocilindricniMotor )

klasa DizelMotor (  (Filter → String)
```

Nadklase : CetvorocilindricniMotor)

klasa Karoserija ((Sasija → String ;
Unutrasnjost → String;
Broj_Vrata → Integer))

klasa Kompanija ((Ime → String ;
Sjediste → Adresa;
Odsjeci →→ Sektor;
Predsjednik → Osoba;
Penzioneri →→ Osoba))

klasa Sektor ((Ime → String ;
Lokacija → Adresa;
Funkcija → String;
Upravnik → Zaposleni;
Radnici →→ Zaposleni))

klasa Projekat ((NazivProjekta → String ;
Opis → String;
OdgovornoLice → Zaposleni;
Budget → Double;
AnagazovaniRadnici →→ Zaposleni))

2.5. Primjeri upita

1. SELECT X FROM Osoba X
   Rezultat upita su identifikatori svih instanci klase Osoba.

2. SELECT X FROM Zaposleni
   Rezultat su identifikatori svih insatnci klase Zaposleni.

3. SELECT X FROM Osoba X WHERE X.Stan.Grad = 'Podgorica'
   Identifikatori svih osoba koje stanuju u Podgorici.

4. SELECT X FROM Osoba X WHERE X.Stan.Grad['Podgorica']
   Rezultat je isti kao i upitu 3.

5. SELECT X.Ime FROM Osoba X
   WHERE X.Stan.Grad = 'Podgorica' AND X.Godine < 26
   Rezultat upita su imena svih osoba koje stanuju u Podgorici i koje su mlade od 26 godina.

6. SELECT Y FROM Zaposleni X WHERE X.Stan[Y].Grad ['Podgorica']
   Rezultat ovog upita su OID-i adresa svih zaposlenih osoba koje stanuju u Podgorici; promjenljiva Y označava OID neke instance klase Adresa jer je domen atributa Stan klasa Adresa. Atribut Stan nije definisan u klasi Zaposleni već je naslijeden od klase Osoba.

7. SELECT Z FROM Zaposleni X , Auto Y
   WHERE X.Vozila_U_Vlasnistvu[Y].Motor.Masina[Z]
   Rezultat upita je skup OID-a mašina koje pripadaju automobilima čiji su vlasnici zaposleni; promjenljiva Y služi da ograniči pretragu na klasu Auto, koja je podklasa klase Vozilo.

8. SELECT &Y FROM Osoba X WHERE X.&Y.Grad['Podgorica']
   Rezultat ovog upita je skup svih atributa takvih da za neki objekat x klase Osoba složeni izraz x.y.Grad['Podgorica'] je tačan (u ovom slučaju, to je atribut Stan klase Osoba); znak & ispred promjenljive Y označava da je to promjenljiva atributskog tipa (tj. njen domen je skup imena atributa). Ovaj upit ilustruje mogućnost jezika da na prirodan način vrši upite nad šemom baze podataka, bez poznavanja sistemskih klasa.

9. SELECT X FROM Auto Y
   WHERE Y.Proizvodjac[X] AND
   X.Predsjednik.Vozila_U_Vlasnistvu.Boja CONTAINS {'plava' , 'crna'}
   AND X.Predsjednik.Godine < 30

Rezultat upita su sve auto-kompanije čiji je predsjednik mlađi od 30 godina i posjeduje bar dva vozila od kojih je jedno plave a drugo crne boje. Pošto je atribut Vozila_U_Vlasnistvu klase Osoba skupovni, to je vrijednost složenog izraza X.Predsjednik.Vozila_U_Vlasnistvu.Boja skup OID-a (u našem slučaju skup literala) pa primjenjujemo skupovni operator CONTAINS.

10. SELECT X FROM Zaposleni X
    WHERE X.Stan.Grad = FORALL X.Clanovi_Porodice.Stan.Grad
    Rezultat upita su OID-i svih zaposlenih takvih da svi članovi porodice dotičnog zaposlenog stanuju u istom gradu u kome stanuje on sam. Kao i u prethodnom upitu, atribut Clanovi_Porodice klase Osoba je skupovni atribut pa je vrijednost složenog izraza X.Clanovi_Porodice.Stan.Grad skup gradova u kojima stanuju članovi porodice zaposlenog. Zbog toga je moguće primjeniti univerzalni kvantifikator FORALL.

11. SELECT X FROM Zaposleni X
    WHERE X.Stan.Grad = FORALL X.Clanovi_Porodice.Stan.Grad
    AND COUNT(X.Clanovi_Porodice)>4 AND X.Plata<400
    Rezultat ovog upita su svi zaposleni koji zaraduju manje od 400 dinara i imaju više od 4 člana porodice koji stanuju u istom gradu u kom stanuje i zaposleni.

12. SELECT X.Ime , W.Plata FROM Kompanija X
    WHERE X.Odsjeci.Radnici[W]
    U svim do sada navedenim upitima, u Select-dijelu upita nalazila se samo jedna promjenljiva ili ime klase. Ovaj upit ilustruje opštiji slučaj kada se u Select-dijelu nalazi više promjenljivih ili izraza. Rezultat upita je skup uredenih parova takvih da je prva komponenta uredenog para ime kompanije a druga komponenta je plata nekog radnika koji je zaposlen u nekom od odsjeka te kompanije.

13. SELECT X, Y FROM Zaposleni X , Zaposleni Y
    WHERE FORALL Y.Clanovi_Porodice.Godine <
    FORALL X.Clanovi_Porodice.Godine
    Rezultat upita su svi parovi zaposlenih takvi da su svi članovi porodice prvog zaposlenog stariji od svakog člana porodice drugog zaposlenog.

14. SELECT X , Y FROM Kompanija X
    WHERE X.Ime = EXISTS X.Odsjeci.Radnici[Y].Ime
    Ovaj upit demonstrira mogućnost eksplicitnog spajanja. Rezultat upita su parovi (Kompanija-objekat, Zaposleni-objekat) takvi da zaposleni ima isto ime kao i kompanija u kojoj radi.

15. SELECT W.Plata AS Zarada
    FROM Kompanija X
    OID OF X, W
    WHERE X.Odsjeci.Radnici[W]

Do sada navedeni primjeri upita kao rezultat su davali ili skup individualnih objekata ili skup uređenih parova (torki). Ovaj upit kreira nove objekte pri čemu se OID kreiranog objekta dobija kao funkcija nekih drugih objekata (tj. njihovih identifikatora). Rezultujući objekat ima jedan atribut čije je ime dato iza ključne riječi AS i čiji se OID dobija kao funkcija koja zavisi od OID-a navedenih u OID OF klauzuli upita (OID-a Kompanije i OID-a Radnika, u našem slučaju). Iako su rezultat upita samo plate zaposlenih u kompaniji, OID rezultujućeg objekta zavisi i od X i od W. U slučaju kada svaki zaposleni radi u samo jednoj kompaniji, tada možemo pisati :

```
SELECT W.Plata AS Zarada
FROM Kompanija X
OID OF W
WHERE X.Odsjeci.Radnici[W]
```

16. SELECT Y.Ime AS ImeKompanije, Y.Odsjeci.Radnici AS Osoblje
FROM Kompanija Y
OID OF Y
Vrijednost atributa ImeKompanije je ime kompanije onog objekta klase Kompanija koji je pridružen promjenljivoj Y a vrijednost atributa Osoblje je skup svih radnika zaposlenih u toj kompaniji. Rezultat upita je skup uređenih parova (ime-kompanije, skup-zaposlenih) .

17. SELECT Y.Ime AS ImeKompanije, {W} AS Izdrzavani
FROM Kompanija Y
OID OF Y
WHERE Y.Penzioneri[W] OR Y.Odsjeci.Radnici.Zavisni[W]
Za proizvoljno dodjeljivanje objekta y promjenljivoj Y, određuje se skup svih objekata w takvih da kada promjenljiva W dobije vrijednost w , logički izraz u Where-dijelu upita postaje tačan. Vrijednost atributa Izdrzavani je skup svih takvih objekata w, kao što je naznačeno u Select-dijelu upita uvođenjem velikih zagrada ({ W}). Iz prethodnog i ovog primjera se vidi da klauzula OID OF ima ulogu sličnu GROUP BY klauzuli u relacionom SQL-u. Takođe, kako su atributi Odsjeci, Radnici i Zavisni skupovni, implicitno se vrši poravnavanje ("flattening"), tako da izraz Y.Odsjeci.Radnici.Zavisni[W] predstavlja skup osoba a ne skup skupova.

18. CREATE VIEW PlateKompanije AS
(SELECT X.Ime AS Ime Kompanije, Y.Ime AS ImeSektora,
W.Plata AS Zarada
FROM Kompanija X
OID OF X, W
WHERE X.Odsjeci[Y].Radnici[W])
Za svakog radnika w , pogled sadrži objekat koji se sastoji od od imena kompanije, imena odsjeka u kome radi w i plate radnika w. Identifikator novog objekta dobija se kao funkcija identifikatora objekata klasa Kompanija i Zaposleni.

19. SELECT X.Proizvodjac.Ime
FROM Auto X , Zaposleni Y
WHERE X.Proizvodjac[Z] and PlateKompanije(Z, Y).Zarada > 1000
Kako je ranije naznačeno, početni selektor može biti OID-funkcija. PlateKompanije je OID-funkcija pridružena pogledu definisanom u upitu 18. Argumenti funkcije su objekat klase Kompanija i objekat klase Zaposleni.

20. CREATE CLASS Osoba (
ATTRIBUTES : (Ime → String ;
Godine → Integer ;
Stan → Adresa ;
Vozila_U_Vlasnistvu →→ Vozilo);
SUBCLASSES : Zaposleni)

Kreira se klasa Osoba koja ima četiri atributa (Ime, Godine, Stan i Vozila_U_Vlasnistvu) i jednu podklasu (Zaposleni).

21. ADD ATTRIBUTE DatumRodjenja → Date TO CLASS Osoba
Dodavanje atributa DatumRodjenja klasi Osoba. Atribut je skalaran a njegov domen je primitivna klasa Date. Ako ne postoji konflikt imena, atribut DatumRodjenja postaje novi atribut i za sve podklase klase Osoba.

22. DROP ATTRIBUTE DatumRodjenja FROM CLASS Osoba
Izbacivanje atributa DatumRodjenja iz klaseOsoba. Svi metodi u klasi Osoba i svim njenim podklasama koji su imali reference na ovaj atribut označeni su kao neupotrebljivi. Atribut se takode briše i iz svih podklasa klase Osoba.

23. ADD METHOD DirektorskaPlata(x→String)→Double
TO CLASS Kompanija
Dodavanje metoda DirektorskaPlata klasi Kompanija. Jedini argument metoda je string koji je ime direktora nekog odsjeka kompanije. Kao rezultat, metod daje visinu plate direktora. Izvršavanjem ovog upita mijenja se struktura klase Kompanija (dodaje se signatura metoda DirektorskaPlata) tj. mijenja se šema baze podataka. Tijelo metoda piše se odvojeno od signature i najčešće se to radi ili u nekom višem programskom jeziku (C, C++, Basic) ili u samom upitnom jeziku. Na primjer, tijelo metoda DirektorskaPlata može da ima sledeći oblik:

SELECT W FROM Kompanija Y
WHERE Y.Odsjeci.Upravnik[Z].Ime = #1 AND Z.Plata[W]

#1 označava prvi argument metoda.

24. SELECT X.Ime , Y , W FROM Kompanija X
WHERE X.Sektor.Ime[Y] AND X.DirektorskaPlata(Y)[W]
Rezultat upita je skup trojki takvih da je prva komponenta ime kompanije, druga komponenta ime odsjeka i treća komponenta je plata upravnika odsjeka. Prikazana je upotreba metoda DirektorskaPlata koji je definisan upitom 23.

25. UPDATE CLASS Osoba
SET Godine = 26 , Vozila_U_Vlasnistvu = {'Scala 101' , 'Jugo 45'}
WHERE Ime = 'Marko Petrovic'
Promjena vrijednosti atributa Godina i Vozila_U_Vlasnistvu objekta klase Osoba čije je ime 'Marko Petrovic' .

26. NAME *add1* = NEW Adresa
UPDATE *add1*
SET Ulica = 'Bratstva Jedinstva 19', Grad = 'Beograd'

NAME *add2* = NEW Adresa
UPDATE *add2*
SET Ulica = 'Mitra Bakica 74', Grad = 'Podgorica'

Imena *add1* i *add2* dodijeljena su instancama klase Adresa.

27. NAME PX = NEW Osoba
UPDATE PX
SET Ime = 'Marko Jankovic' , Adresa = add1
Kreira se novi objekat klase Osoba, tom objektu se dodjeljuje ime PX, zatim se vrijednosti atributa Ime i Adresa postavljaju na 'Marko Jankovic' i *add1*.

28. UPDATE PX
SET Godine = 23 , Adresa = *add2*, Ime = 'John McEnroe'
Vrijednosti atributa Godine, Adresa i Ime objekta PX definisanog u upitu 27 postavljene su na nove vrijednosti. Promijenjena je vrijednost atributa Ime Vrijednost atributa Adresa postavljena je na objekat *add2* koji je definisan u upitu 26. Isti efekat mogao se postići pomoću ugnježdenih upita na sledeći način :

```
UPDATE PX
SET Godine = 23 , Ime = 'John McEnroe',
Adresa = (SELECT X FROM Adresa X
            OID OF X
            WHERE X.Ulica = 'Mitra Bakica 74'
        AND X.Grad = 'Podgorica')
```

29. DELETE FROM Osobe X WHERE X.Stan.Grad = 'Podgorica'
Brisanje svih osoba koje su nastanjene u Podgorici.

30. DROP CLASS Osoba
Brisanje klase Osoba.

31. DROP METHOD DirektorskaPlata FROM CLASS Kompanija
Brisanje metoda DirektorskaPlata iz klase Kompanija. Metod se takode briše i iz svih podklasa klase Kompanija koje su naslijedile ovaj metod .

32. SELECT X.Ime , Y.Ime , W FROM Kompanija X
WHERE X.Sektor[Y] AND Y.Ime[Z] AND X.DirektorskaPlata(Z)[W]
Ovaj upit daje isti rezultat kao i upit 24. ali su argumenti u pozivu metoda promjenljive a ne složeni izrazi. Uvodenjem novog složenog izraza u Where-uslov i njegovom konjunkcijom sa postojećim predikatom, uvijek je moguće postići da su argumenti metoda ili OID-funkcije ili promjenljive a ne složeni izrazi.

33. ADD ATTRIBUTE Nadredjeni → Zaposleni TO CLASS Zaposleni
Dodavanje atributa *Nadredjeni* čiji je domen klasa Zaposleni klasi Zaposleni. Ovaj atribut nam je potreban zbog rekurzivnih upita (primjer 34)

34. CREATE VIEW PPSefovi AS
(SELECT X.*(Nadredjeni) FROM Zaposleni X
WHERE X.Ime = 'Petar Popović')

SELECT X FROM PPSefovi X
WHERE X.Godine<30
Primjer definisanja rekurzivnog pogleda (i upita). Prvo se definiše pogled koji za radnika čije je ime 'Petar Popović' određuje OID-e svih njemu nadredenih radnika. Simbol * označava da je upit rekurzivan i da se izraz u zagradi iza * ponavlja dok je to moguće. Kada se odredi instanca klase Zaposleni koja predstavlja nadređenog radnika Petru Popovicu, postupak se ponavlja za tu instancu (tj. određuje se nadredeni od neposrednog nadredenog), zatim za novodobijenu instancu i tako dalje. Dakle, navedeni upit predstavlja tranzitivno zatvorenje klase Zaposleni u odnosu na atribut Nadredjeni. U opštem slučaju, izraz oblika $X.*(Att_1.Att_2.\cdots.Att_n)$ označava sledeće izraze :

$X.Att_1.Att_2.\cdots.Att_n$ ili

$X.Att_1.Att_2.\cdots.Att_n.Att_1.Att_2.\cdots.Att_n$ ili ...

$X.\underbrace{Att_1.Att_2.\cdots Att_n}_{1}.\underbrace{Att_1.Att_2.\cdots Att_n}_{2}.\cdots.\underbrace{Att_1.Att_2.\cdots Att_n}_{k}$.

Pošto domen atributa Att_n nije primitivna klasa i kako je broj objekata u bazi konačan, ovakva definicija rekurzije nam garantuje da se uvijek može dobiti odgovor. Drugi upit koristi definisani rekurzivni pogled i daje sve nadredene 'Petra Popovića' koji su mladi od 30 godina.

35. CREATE CLASS ŽenaZaposleni (SUPERCLASSES : Zaposleni)
Kreira se podklasa klase Zaposleni, koja se koristi u sledećem primjeru.

36. SELECT Y FROM Zaposleni X , ŽeneZaposleni Y
WHERE X.*(Nadredjeni)[Y] AND X.Ime = 'Petar Popović'
Rezultat upita su OID-i svih nadredenih ženskih osoba Petra Popovića.

37. SELECT X FROM Vozila X
WHERE X.Boja = 'Plava' AND
X IN X.Proizvodjac.Predsjednik.Vozila_U_Vlasnistvu

Ovo je primjer cikličnog upita. Rezultat upita je skup svih vozila plave boje koje vozi predsjednik kompanije koja ih je proizvela.

38. SELECT Model, 1000 * Tezina
FROM Vozilo
Rezultat upita su uredeni parovi (*model, težina*), gdje je težina prikazana u gramima. Aritmetički izraz je upotrijebljen kao složeni izraz.

39. UPDATE Zaposleni
SET Plata = 1.1 * Plata
WHERE Plata < 450
Još jedan primjer upotrebe aritmetičkog izraza kao složenog izraza.

40. CREATE INDEX Ind_gradovi ON CLASS Zaposleni AS Adresa.Grad
Ovom naredbom kreira se takozvani "path"-indeks ([4], [11]) na klasi Zaposleni.

2.6. Upitni jezik i princip učaurenja

Jedna od osnovnih osobina objektno-orijentisanog pristupa je učaurenje (engl. - "encapsulation"). Svaki objekat je učauren to jest njegova unutrašnja struktura nije vidljiva korisniku tog objekta; umjesto toga, korisnik zna da je objekat sposoban za izvodenje odredenih funkcija, koje su predstavljene metodima ([3]). Na primjer, unutrašnja struktura objekata klase *Zaposleni* predstavljena je atributima *Ime, Adresa, Vozila_U_Vlasnistvu* (koji su naslijedeni iz klase *Osoba*), *Kvalifikacija, Plata, Clanovi_porodice* i *Radi_Na_Projektu*. Metodi za objekat klase *Zaposleni* mogu biti *Zajednicki_rad, Povecaj_platu, Dodaj_clana_porodice* i slično.

Prednost koncepta učaurenja je da promjena unutrašnje strukture objekta ne uslovljava promjenu aplikativnog koda koji koristi taj objekat. Naravno, promjena strukture objekta mora se odraziti na kod koji implementira metode. Na taj način se ostvaruje jedan oblik nezavisnost podataka.

Koristeći terminologiju programskog jezika C++ ([16]), atribute tretiramo kao privatne članove klase (engl. - "private members"), dok metodi čine javni (engl.- "public") dio klase. Ovi metodi su tada jedini način na koji možemo pristupiti datom objektu. Kôd koji implementira ove metode može da vidi unutrašnju strukturu objekta, to jest metodi, i samo oni, mogu da naruše učaurenje.

PRIMJER 2.6.1

Data je klasa *Segment,* koja predstavlja duži u ravni. Duž je opisana početnom i krajnjom tačkom, to jest pomoću dva objekta klase *Tacka*. Jedan od metoda klase *Segment* može biti *Duzina_duzi*. Rezultat poziva metoda *Duzina_duzi* je realan broj (to jest, objekat klase *float*) koji predstavlja dužinu date duži. U programskom jeziku C++, klase *Segment* i *Tacka* i metod *Duzina_duzi* možemo opisati na sledeći način :

```
class Tacka
 { private :
       float apscisa;
       float ordinata;

   public:
       float GetApscisa();
       float GetOrdinata();
       void SetApscisa();
       void SetOrdinata();
       ...
 }
```

```
class Segment
 { private :
       Tacka  pocetna_tacka;
       Tacka  krajnja_tacka;
   public :
       float Duzina_duzi();

 }
```

```
float Segment :: Duzina_duzi()
 {
       float p, q;
```

```
    p = pocetna_tacka.GetApscisa() - krajnja_tacka.GetApscisa() ;
    q = pocetna_tacka.GetOrdinata() - krajnja_tacka.GetOrdinata() ;
    return sqrt(p*p + q*q);
}
```

Kako je *Duzina_duzi* jedini metod klase *Segment*, jedini podatak koji možemo dobiti o duži je njena dužina. Da bi dobili ostale podatke o duži, kao što su sredina duži, početna ili krajnja tačka, moramo definisati odgovarajuće metode. Najčešće su to metodi koji vraćaju i postavljaju vrijednost odgovarajućeg atributa, kao što je to slučaj sa klasom *Tacka*. Ako se promijeni struktura klase *Tacka* (na primjer, umjesto Dekartovih koordinata, tačku predstavljamo preko polarnih koordinata - [17]) i ako se kôd metoda *GetApscisa* i *GetOrdinata* izmijeni u skladu sa promjenama strukture, tada nije potrebno vršiti nikakve izmjene u kôdu metoda *Duzina_duzi*. Slično, ako se promijeni unutrašnja struktura objekata klase *Segment*, samo se mora promijeniti kôd metoda *Duzina_duzi*. Ako izaberemo da duž predstavljamo preko središta duži, dužine duži i nagiba (ugla koji duž zaklapa sa pozitivnim dijelom apscisne ose), tada se klasa *Segment* i kôd metoda *Duzina_duzi* mijenjaju na sledeći način :

```
class Segment
{ private :
      Tacka  srediste;
      float  duzina;
      float nagib;
  public :
      float  Duzina_duzi();
      ...
}

float Segment :: Duzina_duzi()
{
      return  duzina;
}
```

Aplikativni kôd koji je koristio objekte klase *Segment* (to jest, metod *Duzina_duzi*) ne mora se mijenjati. Kôd metoda *Duzina_duzi* direktno pristupa "private" članu *duzina* klase *Segment*, to jest metodi klase jedini mogu narušiti učaurenje. ◊

Ako je jedini način pristupa objektima preko ranije definisanih metoda, tada se složeni izrazi u upitima ("ad hoc queries" - [3]) moraju kreirati na poseban način da se ne bi narušilo učaurenje. Jedna mogućnost je da se složeni izrazi u upitima sastoje samo od poziva metoda. U tom slučaju, za svaki atribut A mora se definisati metod *GetA*, gdje je rezultat poziva metoda *GetA* vrijednost atributa A. Metod *GetA* može biti skalaran ili

skupovni, u zavisnosti od atributa A. Postavljanje vrijednosti atributa ostvaruje se preko metoda *SetA*; rezultat poziva metoda *SetA* je postavljanje vrijednosti atributa A na željenu vrijednost. Tada se sve UPDATE-naredbe mogu vršiti preko metoda oblika *SetA*. Metodi *GetA* i *SetA* su takozvani atributski metodi; jedini način na koji možemo pristupiti vrijednosti atributa A ili promijeniti njegovu vrijednost je preko ovih metoda. Ako je signatura atributa A oblika $A \triangleright C$, gdje $\triangleright$ je ili $\rightarrow\!\!\!\rightarrow$ ili $\rightarrow$, tada su signature metoda *GetA* i *SetA* date sa $GetA() \triangleright C$ i $SetA(C) \triangleright NIL$, gdje je *NIL* oznaka da metod ne vraća vrijednost. U slučaju kada za neki atribut A nisu definisani *GetA* i *SetA*, tada vrijednost atributa A ostaje u potpunosti van domašaja korisnika, iako je možda ta vrijednost upotrebljena u nekom od metoda.

PRIMJER 2.6.2.

Posmatrajmo klasu *Osoba* iz poglavlja 2.4. Atribut Godine predstavlja broj godina osobe. Za svaki objekat te klase, tako definisan atribut se mora mijenjati na datum rođendana date osobe. Ako uvedemo atribut *Datum_rodjenja*, tada se atribut *Godine* može implementirati preko datuma rođenja, bez ranije navedenih nedostatatka. Ako poštujemo učaurenje, tada umjesto atributa *Godine* uvedimo metod *Godine()*. Dodajmo i odgovarajuće atributske metode : *GetIme*, *SetIme*, *GetStan*, *SetStan*, *GetVozila_U_Vlasnistvu* i *SetVozila_U_Vlasnistvu*. Tada klasa *Osoba* ima sledeći oblik :

klasa *Osoba* (Atributi : (*Ime* $\rightarrow$ *String* ;
Datum_rodjenja $\rightarrow$ *Datum* ;
Stan $\rightarrow$ *Adresa* ;
Vozila_U_Vlasnistvu $\rightarrow\!\!\!\rightarrow$ *Vozilo*)
Metodi : (*GetIme()* $\rightarrow$ *String;*
SetIme(String) $\rightarrow$ *Nil;*
GetStan() $\rightarrow$ *Adresa;*
SetStan(Adresa) $\rightarrow$ *Nil;;*
GetVozila_U_Vlasnistvu() $\rightarrow\!\!\!\rightarrow$ *Vozilo;*
SetVozila_U_Vlasnistvu({Vozilo}) $\rightarrow$ *Nil;*
Godine() $\rightarrow$ *Integer*
BrojClanovaPorodice(String) $\rightarrow$ *Integer*))

Metodi *GetIme*, *SetIme*, *GetStan*, *SetStan*, *GetVozila_U_Vlasnistvu* i *SetVozila_U_Vlasnistvu* su atributski metodi. Rezultat poziva metoda *Godine* za objekat klase *Osoba* je broj godina date osobe, koji dobijamo kada od tekuće godine oduzmemo godinu atributa *Datum_rodjenja* (*Datum* je predefinisana klasa, kao *Float* ili *Integer*). Vrijednost atributa *Datum_rodjenja* je na ovaj način sakrivena od korisnika, iako je ta vrijednost upotrebljena u kodu koji implementira metod *Godine*. ◊

Ako je jedini način pristupa vrijednostima atributa preko unaprijed definisanih metoda, tada se pojam složenog izraza iz poglavlja 2.1. mora redefinisati. Formalno, složeni izraz je i dalje oblika $sel_0.MetIzraz_1[sel_1].\cdots.MetIzraz_n[sel_n]$, gdje je $n \geq 0$ i svaki od selektora, osim prvog, je opcionalan . Selektor može biti ili OID ili individualna promjenljiva (promjenljiva čiji je domen skup OID-a individualnih objekata) ili ID-funkcija (funkcija koja definiše OID). Svaki od izraza $MetIzraz_i$, $i = 1,\cdots,n$ je ili poziv

metoda ili promjenljiva čiji je domen skup imena metoda. Opšti oblik izraza $MetIzraz_i$, $i = 1,\cdots,n$ ima sledeći oblik :

- $meth(a_1,\ldots,a_n)$, gdje je *meth* ime metoda a $a_1,\ldots,a_n$ argumenti metoda.
- *&ime_promjenljive* , gdje *ime_promjenljive* označava promjenljivu čiji je domen skup imena metoda.

Definicije 2.1.1. (kada je zadovoljen složeni izraz) i 2.1.2 (vrijednost složenog izraza) ostaju nepromijenjene.

PRIMJER 2.6.3.

Pretpostavimo da su za sve atribute klasa iz poglavlja 2.4. definisani odgovarajući atributski metodi (slično primjeru 2.6.2.). Tada složeni izrazi imaju sledeći oblik (ovo je preradeni primjer 2.1.2) :

(a) *X.GetStan().GetGrad()*
(b) *Petar.GetClanovi_Porodice().Godine()*
(c) *X.GetVozila_U_Vlasnistvu*()[Y].*GetBoja*()[Z]
(d) *X.BrojClanovaPorodice*('Marko Markovic')
(e) *X.&Y.GetGrad()*
(f) *Kompanija.GetPredsjednik*()[P].*BrojClanovaPorodice*(*P.GetIme*())[W]

U primjeru (a), *X* je promjenljiva koja označava objekat klase Osoba (tj. domen promjenljive *X* je klasa *Osoba*), dok su *GetStan* i *GetGrad* atributski metodi klasa *Osoba* i *Adresa*, respektivno. U primjeru (b), *Petar* je OID nekog objekta klase *Osoba* (ili klase *Zaposleni*), dok je *GetClanovi_Porodice* skupovni atributski metod klase *Osoba* (ili *Zaposleni*). *BrojClanovaPorodice* je metod definisan u klasi *Osoba* čiji je argument string a rezultat cio broj. Primjeri (d) i (f) ilustruju upotrebu neatributskih metoda u složenim izrazima. Argument metoda u primjeru (d) je literal (znakovna konstanta) dok je u primjeru (f) argument metoda drugi složeni izraz (to jest izraz *P.GetIme*()). U primjeru (e), početni selektor je promjenljiva čiji je domen klasa *Osoba*, dok znak '&' ispred promjenljive Y označava da je domen promjenljive Y skup imena metoda. U datom primjeru, vrijednost promjenljive Y je metod *GetStan* . ◊

Svaki od upita iz primjera 2.4.1. sada se može transformisati tako da se složeni izrazi sastoje samo od poziva metoda ili promjenljivih. Upiti koji otkrivaju strukturu baze podataka sada prikazuju samo metode klase a ne atribute, tako da unutrašnja struktura objekta ostaje sakrivena.

PRIMJER 2.6.3.

(a) SELECT X FROM Osoba X WHERE X.GetStan().GetGrad() = 'Podgorica'

(b) SELECT X.GetIme() FROM Osoba X
WHERE X.GetStan().GetGrad() = 'Podgorica' AND X.Godine() < 26

(c) SELECT Y FROM Zaposleni X
WHERE X.GetStan()[Y].GetGrad()['Podgorica']

(d) SELECT Z FROM Zaposleni X , Auto Y
WHERE X.GetVozila_U_Vlasnistvu()[Y].GetMotor().GetMasina()[Z]

(e) SELECT &Y FROM Osoba X WHERE X.&Y.GetGrad()['Podgorica']
Rezultat ovog upita je skup svih metoda takvih da za neki objekat *x* klase *Osoba* složeni izraz *x.y.GetGrad*['*Podgorica*'] je tačan (u ovom slučaju, to je atributski metod *GetStan* klase Osoba); znak & ispred promjenljive Y označava da je to promjenljiva metodskog tipa (tj. njen domen je skup imena metoda). Ovaj upit ilustruje mogućnost jezika da na prirodan način vrši upite nad šemom baze podataka, bez poznavanja sistemskih klasa. Jedine informacije o strukturi objekta koje dobijamo odnose se na imena metoda definisanih za dati objekat.

(f) SELECT W.GetPlata() AS Zarada
FROM Kompanija X
OID OF X, W
WHERE X.GetOdsjeci().GetRadnici()[W]

Rezultujući objekat ima jedan atribut čije je ime dato iza ključne riječi AS (u ovom slučaju *Zarada*).

(g) CREATE CLASS *Osoba* (
ATTRIBUTES : (*Ime* → *String* ;
Godine → *Integer* ;
Stan → *Adresa* ;
Vozila_U_Vlasnistvu →→ *Vozilo*);
METHODS : (*GetIme()* → *String;*
SetIme(String) → *Nil;*
GetStan() → *Adresa;*
SetStan(Adresa) → *Nil;*
GetVozila_U_Vlasnistvu() →→ *Vozilo;*
SetVozila_U_Vlasnistvu({Vozilo}) → *Nil;*
Godine() → *Integer*
BrojClanovaPorodice(String) → *Integer*))

(h) UPDATE CLASS Osoba X
SET X.SetGodine(26) , X.SetVozila_U_Vlasnistvu ({'Scala 101' , 'Jugo 45'})
WHERE X.GetIme() = 'Marko Petrovic'
Promjena vrijednosti atributa *Godina* i *Vozila_U_Vlasnistvu* objekta klase *Osoba* čije je ime 'Marko Petrovic' .

(i) NAME *add1* = NEW Adresa
UPDATE *add1*
SET *add1.SetUlica*('Bratstva Jedinstva 19'), *add1.SetGrad*('Beograd');

(j) NAME PX = NEW Osoba
UPDATE PX

```
    SET PX.SetIme('Marko Jankovic') , PX.SetAdresa(add1)

(k) DELETE FROM Osobe X WHERE X.GetStan().GetGrad() = 'Podgorica'

(l) UPDATE Zaposleni X
    SET X.SetPlata(1.1*Y)
    WHERE X.GetPlata()[Y]< 450
(m) CREATE VIEW PPSefovi AS
    ( SELECT X.*(GetNadredjeni()) FROM Zaposleni X
    WHERE X.GetIme() = 'Petar Popović' )

    SELECT X FROM PPSefovi X
    WHERE X.Godine() <30
```

Primjer definisanja rekurzivnog pogleda (i upita).

Svi primjeri predstavljaju modifikovane upite iz poglavlja 2.5. ◊

Posebno moramo razmotriti slučaj kada se u složenom izrazu pojavljuje poziv metoda oblika *SetAtribut* (primjer 2.6.3(h)). Tip rezultata svih metoda oblika *SetAtribut* je *NIL*, pa jedini način upotrebe ovog tipa metoda u složenim izrazima je u UPDATE-naredbi. U svim ostalim slučajevima, poziv metoda *SetAtribut* u složenom izrazu dovodi do greške.

Drugi pristup problemu učaurenja je postojanje dva različita môda upitnog jezika, kako je naznačeno u ([3]). Prvi môd je takozvani programerski môd, u kojem se objektima pristupa jedino preko metoda. U programerskom môdu, striktno se poštuje princip učaurenja. U tom slučaju, naredbe upitnog jezika su sastavni dio nekog programskog jezika, kao što je CO_2 ili BasicO_2 ([3]). Drugi môd je interaktivni, "ad hoc" môd, u kojem upitni jezik može narušavati učaurenje.

2.7. Semantika upita

Za proizvoljni upit P manipulativnog dijela jezika, svaka promjenljiva zamjenjuje se objektnim identifikatorom (OID-om) vodeći računa da identifikatori odgovaraju vrsti promjenljive (tj. da li je u pitanju klasna promjenljiva, ime metoda/atributa ili individualna promjenljiva). Način izračunavanja izraza u WHERE-bloku upita je sledeći : složeni izraz je tačan ako njegova vrijednost nije prazan skup; poređenje dva složena izraza je tačno ako vrijednosti ta dva izraza zadovoljavaju odgovarajuću relaciju (na primjer <, =, FORALL ... >, itd.). Logički operatori (*and, or* i *not*) izračunavaju se na uobičajen način. Ako je WHERE-blok tačan, tada se izračunavaju složeni izrazi u SELECT-bloku upita. Rezultat upita je skup torki čije su komponente vrijednosti izraza iz SELECT-bloka upita. Na primjeru upita iz primjera 2.5.9, navedeni postupak izgleda ovako : za svaki objekat y iz klase Auto određuje se izraz y.Proizvodjac i taj objekat označavamo promjenljivom x. Za svaki takav objekat x (koji mora pripadati klasi Kompanija) izračunavaju se složeni izrazi x.Predsjednik.Godine i 20 (koji je identifikator, pa jeste složeni izraz) kao i složeni izrazi x.Predsjednik.Vozila_U_Vlasnistvu.Boja i {'Plava', 'Crna'}, i provjerava se da li zadovoljavaju navedene relacije (u našem slučaju < i sadrži tj. CONTAINS). Ako izračunati izrazi zadovoljavaju navedene relacije, onda objekat x dodajemo rezultatu. Pretpostavili smo da se logički izrazi izračunavaju s lijeva u desno.

Modifikacije šeme objektno-orijentisane baze podataka vrši se pomoću naredbi deklarativnog dijela upitnog jezika tj. preko naredbi CREATE, DROP i ADD. Svaka od ovih naredbi vrši dodavanje, brisanje ili mijenjanje jedne ili više instanci neke od sistemskih klasa. Izmjene šeme možemo klasifikovati u dvije grupe : izmjene strukture klase i izmjene u relacijama koje postoje između klasa (relacije podklasa/nadklasa, disjunktnost klasa, klase preklapanja). Izmjene sadržaja klase se odnose na dodavanje i brisanje atributa/metoda klase i promjenu imena klase. Drugu vrstu promjena čine sledeće operacije:

- dodavanje i brisanje klase ;
- uklanjanje klase K iz liste nadklasa (podklasa, klasa preklapanja, disjunktnih klasa) klase C ;
- dodavanje klase K listi nadklasa (podklasa, klasa preklapanja, disjunktnih klasa) klase C .

Neke od navedenih promjena direktno su vezane sa promjenama strukture klase: na primjer, brisanjem klase, brišu se i svi atributu i metodi te klase.

Naredba CREATE CLASS kreira novu instancu klase SysKlasa; za svaki atribut i metod koji se navede u ATTRIBUTES i METHODS dijelu naredbe kreira se po jedna instanca u klasama SysAtribut i SysMetod.

PRIMJER 2.7.1. Prikazaćemo šta se dešava sa sistemskim klasama na upitu iz primjera 2.5.20. :

```
CREATE CLASS Osoba (
        ATTRIBUTES : ( Ime → String ;
                       Godine → Integer ;
                       Stan → Adresa ;
                       Vozila_U_Vlasnistvu →→ Vozilo );
```

SUBCLASSES : Zaposleni ;)

Prvo se kreira instanca k_1 sistemske klase *SysKlasa* takva da je k_1. *ImeKlase* = 'Osoba' i k_1.*Podklase* = {'Zaposleni'} (k_1 je OID te instance). Zatim se u klasi *SysAtribut* kreiraju instance (na primjer, neka su to a_1, a_2, a_3 i a_4) koje odgovaraju redom atributima Ime, Godine, Stan i Vozila_U_ Vlasnistvu. Za atribut Ime imamo a_1.ImeAtributa = 'Ime', a_1.DomenAtributa = String i a_1.Multiple = False (atribut nije skupovni). Slično važi i za ostale atribute. Sada postavljamo da je k_1.*AtributiKlase* = $\{a_1, a_2, a_3, a_4\}$. U slučaju da je navedena neka nadklasa klase Osoba, tada bi skupu k_1.*AtributiKlase* dodali sve atribute te nadklase. ◊

Naredbom ADD može se vršiti dodavanje novog atributa, metoda, podklase, nadklase, klase preklapanja ili disjunktne klase. Klasi koja je specificirana u naredbi dodaje se navedeni objekat; postupak se ponavlja za sve podklase date klase.

PRIMJER 2.7.2. Prikazaćemo kakav je efekat izvršavanja ADD naredbe iz primjera 2.5.21. :

ADD ATTRIBUTE DatumRodjenja → Date TO CLASS Osoba

Neka je k instanca klase *SysKlasa* koja predstavlja klasu Osoba. Za svaku podklasu klase Osoba (u ovom slučaju to je samo klasa Zaposleni) i za samu klasu Osoba, provjerava se da li postoji u toj klasi atribut sa imenom DatumRodjenja; ako postoji, tada se ova naredba ne može izvršiti i daje se poruka o grešci. U suprotnom, kreira se instanca a klase *SysAtribut* takva da je:

a.*DomenAtributa* = 'Date'
a.*ImeAtributa* = 'DatumRodjenja'
a.*Multiple* = False

Skupu atributa klase Osoba dodaje se a kao poslednji elemenat skupa. Za svaku podklasu klase Osoba, skupu atributa podklase dodaje se atribut a. ◊

Naredbom DROP *Atribut/Metod* FROM CLASS *ImeKlase*, iz klase *ImeKlase* i svih njenih podklasa briše se navedeni atribut/metod. U klasi *SysKlasa*, iz skupa vrijednosti atributa *AtributiKlase* instance koja odgovara datoj klasi, briše se OID instance koja odgovara atributu/metodu koji se uklanja; postupak se ponavlja za svaku podklasu klase *ImeKlase*. Na kraju se iz klase *SysAtribut* (*SysMetod*) brišu instance koje odgovaraju datom atributu/metodu.

Primjena naredbe DROP za brisanje klase je proces koji se izvodi u četiri koraka: prvo se navedena klasa ukloni iz liste podklasa svake od njenih nadklasa; drugo, isti postupak se sprovede i za podklase date klase; treće, obrišu se atributi i metodi te klase a zatim se ukloni i sama klasa iz hijerahije klasa. Sličan postupak važi i za brisanje podklase, nadklase, klase preklapanja ili disjunktne klase.

2.8. Semantika upita u SF-Logici

SF-program je niz SF-formula oblika $A \leftarrow B$, gdje je A elementarna SF-formula, a B je konjunkcija elementarnih formula. Uobičajeno je da se formula oblika $A \leftarrow B$ naziva pravilom (engl. - "rule"). Formula A se naziva glavom a B tijelom pravila $A \leftarrow B$.

PRIMJER 2.8.1. U formulama 1-12 iz primjera 1.4.2.1, glava je čitava formula, dok je tijelo prazno. U formuli 13 iz istog primjera, formula $X[Zajednicki\_rad\#Y \rightarrow\rightarrow Z]$ je glava, dok je formula

$X:Zaposleni \wedge X[Radi\_Na\_Projektu \rightarrow\rightarrow Z] \wedge Y:Zaposleni \wedge Y[Radi\_Na\_Projektu \rightarrow\rightarrow Z]$

tijelo pravila. Slično je i sa primjerom 14. ◊

Programom se specificira šta koji metod treba da radi, definišu se signature metoda i objekti se organizuju u hijerarhiju, pa svaki SF-program možemo podijeliti na tri medusobno disjunktna dijela. Prvi dio čini definicija hijerahije baza podataka, to jest sva pravila čija je glava elementarna formula oblika $o:C$ ili $C_1::C_2$. Drugi dio je definisanje signatura metoda i čine ga pravila čija je glava neatomska elementarna formula u kojoj se pojavljuju samo izrazi signature. Treći dio je definisanje objekata (to jest, vrijednosti njihovih atributa i metoda) i sačinjavaju ga pravila čija je glava elementarna SF-formula u kojoj nema izraza signature. Primjer SF-programa čine formule definisane u PRIMJERU 1.4.2.1. Dio programa koji definiše hijerarhiju čine formule 4, 6, 7, 10 i prva od formula iz 2. Definisanje signatura metoda se vrši u primjerima 1, 2 i 3, dok se definisanje objekata obavlja formulama 5, 8, 9, 11, 12, 13 i 14.

Upit u SF-logici je izraz oblika ?- Q , gdje je Q elementarna formula. Na ovaj način se ne gubi opštost, jer se upit može transformisati u navedeni oblik dodavanjem odgovarajućeg pravila.

PRIMJER 2.8.2.

Upit ?- $X:Zaposleni \wedge X[Ime \rightarrow "Janko\ Rudan"; Godine \rightarrow G; Adresa \rightarrow A]$ nije oblika ?- Q , pa uvodimo novo pravilo :

$$obj(X)[Godine \rightarrow G; Adresa \rightarrow A] \leftarrow$$
$$X:Zaposleni \wedge X[Ime \rightarrow "Janko\ Rudan"; Godine \rightarrow G; Adresa \rightarrow A]$$

a upit postaje ?- $obj(X)[Godine \rightarrow G; Adresa \rightarrow A]$. ◊

Rezultat upita Q u odnosu na SF-program P je najmanji skup $\Re$ elementarnih formula takav da:

- svaka instanca q formule Q pripada kanonskom modelu programa P ;
- $\Re$ je zatvoren u odnosu na $\models$.

2.9. Prevodenje QLO-upita u SF-programe

U SF-Logici, vrijednosti atributa ili metoda mogu da budu skupovi objekata, koji se u upitima mogu uporedivati (poglavlje 2.5, upiti 9, 10, 11 i 13). Takode je moguće vršiti provjeru da li odredeni objekat pripada nekom skupu objekata. Zbog jednostavnosti i kraćeg zapisivanja formula SF-Logike, uvešćemo relacije *subset*, *contains* i *equal* koje prikazuju odnose izmedu dva skupovna atributa ili metoda, kao i relaciju *member* koja prikazuje odnos skalarnog i skupovnog atributa. Definisanje ovih relacija u SF-Logici slično je uporedivanju relacija u klasičnom logičkom programiranju.

Ako su dati objekti $o_1[Attr_1 \rightarrow\rightarrow ...]$, $o_2[Attr_2 \rightarrow\rightarrow ...]$ i o_3, tada $member(o_3, o_1, Attr_1)$ označava da je objekat o_3 element skupa koji je skup vrijednosti

atributa $Attr_1$ objekta o_1. Slično, $subset(o_1, Attr_1, o_2, Attr_2)$ (ili $equal(o_1, Attr_1, o_2, Attr_2)$, $contains(o_1, Attr_1, o_2, Attr_2)$) označava da je skup vrijednosti atributa $Attr_1$ objekta o_1 podskup skupa vrijednosti atributa $Attr_2$ objekta o_2 (to jest, jednak je tom skupu - *equal* ili ga sadrži - *contains*). Ako je R proizvoljna binarna relacija, tada tranzitivno zatvorenje relacije R označavamo sa TR. Tranzitivno zatvorenje je potrebno zbog rekurzivnih upita (poglavlje 2.5, upit 34). Ove relacije mogu se definisati na sledeći način ([8],[9],[10]):

$$member(o_3, o_1, Attr_1) \leftarrow o_1[Attr_1 \rightarrow\rightarrow o_3]$$

$$notsubset(o_1, Attr_1, o_2, Attr_2) \leftarrow o_1[Attr_1 \rightarrow\rightarrow X] \wedge \neg o_2[Attr_2 \rightarrow\rightarrow X]$$
$$subset(o_1, Attr_1, o_2, Attr_2) \leftarrow \neg notsubset(o_1, Attr_1, o_2, Attr_2)$$

$$contains(o_1, Attr_1, o_2, Attr_2) \leftarrow subset(o_2, Attr_2, o_1, Attr_1)$$

$$notequal(o_1, Attr_1, o_2, Attr_2) \leftarrow notsubset(o_1, Attr_1, o_2, Attr_2)$$
$$notequal(o_1, Attr_1, o_2, Attr_2) \leftarrow notsubset(o_2, Attr_2, o_1, Attr_1)$$
$$equal(o_1, Attr_1, o_2, Attr_2) \leftarrow \neg notequal(o_1, Attr_1, o_2, Attr_2)$$

$$TR(x, y) \leftarrow R(x, y)$$
$$TR(x, y) \leftarrow R(x, z) \wedge TR(z, y)$$

PRIMJER 2.9.1.

U primjerima koji slijede koristimo konvenciju da velika slova označavaju promjenljive. Sve promjenljive koje se pojavljuju u upitima su implicitno univerzalno kvantifikovane.

1. SELECT X FROM Osoba X
   ?- $X : Osoba$

2. SELECT X FROM Zaposleni
   ?- $X : Zaposleni$

3. SELECT X FROM Osoba X WHERE X.Stan.Grad = 'Podgorica'
   ?- $X : Osoba \wedge X[Stan \rightarrow S[Grad \rightarrow 'Podgorica']]$

4. SELECT X FROM Osoba X WHERE X.Stan.Grad['Podgorica']
   ?- $X : Osoba \wedge X[Stan \rightarrow S[Grad \rightarrow 'Podgorica']]$

5. SELECT X.Ime FROM Osoba X

WHERE X.Stan.Grad = 'Podgorica' AND X.Godine < 26

$$o(X)[Ime \to I] \leftarrow X{:}Osoba \wedge X[Ime \to I; Stan \to S[Grad \to 'Podgorica'], Godine \to G] \wedge (G < 26)$$

$$?- o(X)[Ime \to I];$$

Izraz $g<26$ bi formalno trebao biti zapisan kao $p(g,26)$, gdje je p relacijsko slovo koje označava relaciju <. Prvo je dato pravilo kojim se formira novi objekat (koji zavisi od objekta X, što je označeno sa $o(X)$), a zatim upit kojim se određuju svi novodobijeni objekti o.

6. SELECT Y FROM Zaposleni X OID OF X
WHERE X.Stan[Y].Grad ['Podgorica']

$$obj(X)[attr \to Y] \leftarrow X{:}Zaposleni \wedge X[Stan \to Y[Grad \to 'Podgorica']]$$

$$?- obj(X)[attr \to Y]$$

7. SELECT Z FROM Zaposleni X, Auto Y
WHERE X.Vozila_U_Vlasnistvu[Y].Motor.Masina[Z]

$$obj(X)[attr \to Z] \leftarrow X{:}Zaposleni \wedge Y{:}Auto \wedge X[Vozila\_U\_Vlasnistvu \to\to Y[Motor \to M[Masina \to Z]]]$$

$$?- obj(X)[attr \to Z]$$

8. SELECT &Y FROM Osoba X WHERE X.&Y.Grad['Podgorica']

$$obj[attr \to Y] \leftarrow X{:}Osoba \wedge X[Y \to G[Grad \to 'Podgorica']]$$

$$?- obj[attr \to Y]$$

9. SELECT X FROM Auto Y OID OF Y
WHERE Y.Proizvodjac[X] AND
X.Predsjednik.Vozila_U_Vlasnistvu.Boja CONTAINS {'plava', 'crna'}
AND X.Predsjednik.Godine < 30;

$$obj_1[attr_1 \to\to \{'plava', 'crna'\}]$$

$$obj_2(X)[attr_2 \to\to B] \leftarrow X{:}Kompanija \wedge X[Predsjednik \to P[Vozila\_U\_Vlasnistvu \to\to V[Boja \to B]]]$$

$$res(Y)[Attr \to X] \leftarrow Y{:}Auto \wedge Y[Proizvodjac \to X] \wedge contains(obj_2(X), attr_2, obj_1, attr_1) \wedge X[Predsjednik \to F[Godine \to G]] \wedge (G < 30)$$

$$?- res(Z)[Attr \to X]$$

10. SELECT X FROM Zaposleni X
WHERE X.Stan.Grad = FORALL X.Clanovi_Porodice.Stan.Grad

$$rel(X)[attr \to X] \leftarrow X{:}Zaposleni \wedge X[Stan \to A[Grad \to G]] \wedge (\forall G)(X[Clanovi\_Porodice \to\to Y[Stan \to B[Grad \to F]]] \wedge G = F)$$

$$?- rel(X)[attr \to X]$$

11. SELECT X.Ime , W.Plata FROM Kompanija X
WHERE X.Odsjeci.Radnici[W]

$$ob(X)[Ime \to I; Plata \to W] \leftarrow X{:}Kompanija \wedge X[Odsjeci \to\to O[Radnici \to\to R[Plata \to W]]]$$

$$?- ob(X)[Ime \to I; Plata \to W]$$

12. SELECT X, Y FROM Zaposleni X , Zaposleni Y
WHERE FORALL Y.Clanovi_Porodice.Godine <
FORALL X.Clanovi_Porodice.Godine

$$obj[attr_1 \to X; attr_2 \to Y] \leftarrow (X{:}Zaposleni \wedge Y{:}Zaposleni \wedge (\forall G)(X[Clanovi\_Porodice \to\to O[Stan \to B[Grad \to G]]] \wedge (\forall H)(Y[Clanovi\_Porodice \to\to U[Stan \to C[Grad \to G]]]) \wedge (G < H)))$$

$$?- obj[attr_1 \to X; attr_2 \to Y]$$

13. SELECT X , Y FROM Kompanija X
WHERE X.Ime = EXISTS X.Odsjeci.Radnici[Y].Ime

$$r[attr_1 \to X; attr_2 \to Y] \leftarrow X{:}Kompanija \wedge X[Ime \to I] \wedge (\exists G)(X[Odsjeci \to\to O[Radnici \to\to Y[Ime \to G]]] \wedge G = I)$$

$$?- r[attr_1 \to X; attr_2 \to Y]$$

14. SELECT W.Plata AS Zarada
FROM Kompanija X
OID OF X, W
WHERE X.Odsjeci.Radnici[W]

$$Z(X,W)[Zarada \to P] \leftarrow X{:}Kompanija \wedge X[Odsjeci \to\to O[Radnici \to\to W[Plata \to P]]]$$

$$?- Z(X,W)[Zarada \to P]$$

15. SELECT Y.Ime AS ImeKompanije, Y.Odsjeci.Radnici AS Osoblje
FROM Kompanija Y
OID OF Y

$$Z(Y)[ImeKompanije \to I; Osoblje \to\to W] \leftarrow Y{:}Kompanija \wedge Y[Odsjeci \to\to O[Radnici \to\to W]; Ime \to I]$$

$$?- Z(Y)[ImeKompanije \to I; Osoblje \to\to W]$$

16. SELECT Y.Ime AS ImeKompanije, {W} AS Izdrzavani
FROM Kompanija Y
OID OF Y
WHERE Y.Penzioneri[W] OR Y.Odsjeci.Radnici.Zavisni[W]

$$Z(Y)\left[ImeKompanije \rightarrow I; Izdrzavani \rightarrow\rightarrow \{W\}\right] \leftarrow Y: Kompanija \wedge$$
$$(Y\left[Odsjeci \rightarrow\rightarrow O\left[Radnici \rightarrow\rightarrow R\left[Zavisni \rightarrow\rightarrow W\right]\right]; Ime \rightarrow I\right] \vee$$
$$Y\left[Odsjeci \rightarrow\rightarrow U\left[Radnici \rightarrow\rightarrow P\left[Zavisni \rightarrow\rightarrow W\right]\right]; Ime \rightarrow I\right])$$
$$?- Z(Y)\left[ImeKompanije \rightarrow I; Izdrzavani \rightarrow\rightarrow \{W\}\right]$$

17. CREATE VIEW Isplate AS
(SELECT X.Ime AS Ime Kompanije, Y.Ime AS ImeSektora,
W.Plata AS Zarada
FROM Kompanija X
OID OF X, W
WHERE X.Odsjeci[Y].Radnici[W])

$$PlateKomp\left[ImeKompanije \Rightarrow String; ImeSektora \Rightarrow String; Zarada \Rightarrow Double\right]$$
$$Isplate(X, W): PlateKomp\left[ImeKompanije \rightarrow I; ImeSektora \rightarrow S; Zarada \rightarrow P\right] \leftarrow$$
$$X: Kompanija \wedge X\left[Odsjeci \rightarrow\rightarrow Y\left[Ime \rightarrow S; Radnici \rightarrow\rightarrow W\left[Plata \rightarrow P\right]\right]; Ime \rightarrow I\right]$$

18. SELECT X.Proizvodjac.Ime
FROM Auto X , Zaposleni Y
WHERE X.Proizvodjac[Z] and PlateKompanije(Z, Y).Zarada > 1000
$$obj(X)[attr \rightarrow I] \leftarrow X: Auto \wedge X\left[Odsjeci \rightarrow Z; Proizvodjac \rightarrow P[Ime \rightarrow I]\right]$$
$$\wedge Y: Zaposleni \wedge PlateKompanije(Z, Y)[Zarada \rightarrow W] \wedge (W > 100)$$
$$?- obj(X)[attr \rightarrow I]$$

19. CREATE CLASS Osoba (
ATTRIBUTES : (Ime → String ;
Godine → Integer ;
Stan → Adresa ;
Vozila_U_Vlasnistvu →→ Vozilo);
SUBCLASSES : Zaposleni)
$$Osoba[Ime \Rightarrow String; Godine \Rightarrow String; Stan \Rightarrow Adresa;$$
$$Vozila\_U\_Vlasnistvu \Rightarrow\Rightarrow Vozilo]$$
$$Zaposleni : Osoba$$

20. ADD ATTRIBUTE DatumRodjenja → Date TO CLASS Osoba
$$Osoba[DatumRodjenja \Rightarrow Date];$$

21. ADD METHOD DirektorskaPlata(String) → Double
TO CLASS Kompanija
DirektorskaPlata(X → String) → Double
(SELECT W FROM Kompanija Y
WHERE Y.Odsjeci.Upravnik[Z].Ime[X] AND Z.Plata[W])

$$Kompanija[DirektorskaPlata(String) \Rightarrow Double];$$

$$Y\left[DirektorskaPlata(X) \rightarrow W\right] \leftarrow Y{:}\,Kompanija \wedge$$
$$Y\left[Odsjeci \rightarrow\!\!\rightarrow O\left[Upravnik \rightarrow Z[Ime \rightarrow X]\right]\right] \wedge Z[Plata \rightarrow W]$$

22. SELECT X.Ime , Y , W FROM Kompanija X
WHERE X.Odsjeci.Ime[Y] AND X.DirektorskaPlata(Y)[W]

$$obj(X)\left[attr_1 \rightarrow I, attr_2 \rightarrow Y, attr_3 \rightarrow W\right] \leftarrow X{:}\,Kompanija \wedge$$
$$X\left[Odsjeci \rightarrow\!\!\rightarrow O[Ime \rightarrow Y]; DirektorskaPlata(Y) \rightarrow W; Ime \rightarrow I\right]$$
$$?- obj(X)\left[attr_1 \rightarrow I, attr_2 \rightarrow Y, attr_3 \rightarrow W\right]$$

23. NAME *add1* = NEW Adresa
UPDATE *add1*
SET Ulica = 'Bratstva Jedinstva 19', Grad = 'Beograd'

NAME *add2* = NEW Adresa
UPDATE *add2*
SET Ulica = 'Mitra Bakica 74', Grad = 'Podgorica'

$$add1{:}\,Adresa \wedge add1\left[Ulica \rightarrow' Bratstva\ Jedinstva\ 19'; Grad \rightarrow' Beograd'\right]$$
$$add2{:}\,Adresa \wedge add2\left[Ulica \rightarrow' Mitra\ Bakica\ 74'; Grad \rightarrow' Podgorica'\right]$$

24. NAME PX = NEW Osoba
UPDATE PX SET Ime = 'Marko Zec', Adresa = *add1*

$$X{:}\,Osoba \wedge X\left[Ime \rightarrow' Marko\ Zec'; Adresa \rightarrow add1)\right]$$

26. ADD ATTRIBUTE Nadredjeni → Zaposleni TO CLASS Zaposleni

$$Zaposleni\left[Nadredjeni \Rightarrow Zaposleni\right]$$

27. CREATE VIEW PPSefovi AS
(SELECT Y FROM Zaposleni X
WHERE X.*(Nadredjeni)[Y] AND X.Ime = 'Petar Popović')

SELECT X FROM PPSefovi X
WHERE X.Godine<30

$$PPsefovi\left[Nadredjeni \Rightarrow Zaposleni\right]$$
$$tr(X,Y) \leftarrow X{:}\,Zaposleni \wedge X\left[Nadredjeni \rightarrow Y\right]$$
$$tr(X,Y) \leftarrow X{:}\,Zaposleni \wedge X\left[Nadredjeni \rightarrow Z\right] \wedge tr(Z,Y)$$
$$res(Y){:}\,PPsefovi\left[Nadredjeni \rightarrow Y\right] \leftarrow X{:}\,Zaposleni \wedge X\left[Ime \rightarrow' Petar\ Popovic'\right]$$
$$\wedge tr(X,Y)$$
$$?- res(Y)\left[Nadredjeni \rightarrow Y\right]$$

28. CREATE CLASS ŽenaZaposleni (SUPERCLASSES : Zaposleni)

$ZenaZaposleni :: Zaposleni$

29. SELECT Y FROM Zaposleni X , ŽeneZaposleni Y
WHERE X.*(Nadredjeni)[Y] AND X.Ime = 'Petar Popović'

$$tr(X,Y) \leftarrow X:Zaposleni \wedge Y:ZenaZaposleni \wedge X[Nadredjeni \rightarrow Y]$$

$$tr(X,Y) \leftarrow X:Zaposleni \wedge Z:ZenaZaposleni \wedge X[Nadredjeni \rightarrow Z] \wedge tr(Z,Y)$$

$$res(Y)[Attr \rightarrow Y] \leftarrow X:Zaposleni \wedge X[Ime \rightarrow 'Petar\ Popovic'] \wedge tr(X,Y)$$

$$?- res(Y)[Attr \rightarrow Y]$$

30. SELECT X FROM Vozila X
WHERE X.Boja = 'Plava' AND
X IN X.Proizvodjac.Predsjednik.Vozila_U_Vlasnistvu

$$obj(X)[attr \rightarrow\rightarrow Y] \leftarrow X:Vozila \wedge X[Proizvodjac \rightarrow P[Predsjednik \rightarrow O[Vozila\_U\_Vlasnistvu \rightarrow\rightarrow Y]]]$$

$$res(X)[attr_1 \rightarrow X] \leftarrow X:Vozila \wedge X[Boja \rightarrow 'plava'] \wedge member(X, obj(X), attr)$$

$$?- res(X)[attr_1 \rightarrow X]$$

31. SELECT X.Model, 1000 * X.Tezina
FROM Vozilo X

$$res(X)[attr_1 \rightarrow M; attr_2 \rightarrow 1000 * T] \leftarrow X:Vozilo \wedge X[Model \rightarrow M; Tezina \rightarrow T]$$

$$?- res(X)[attr_1 \rightarrow A; attr_2 \rightarrow B]$$

Proces prevodenja upita jezika QLO u programe SF-Logike je pravolinijski. Osnovna ideja prevodenja je da se pojedinim djelovima QLO-upita pridruže formule SF-Logike ([18]) i da se te formule povežu odgovarajućim logičkim veznicima. U SELECT-naredbi, svaki složeni izraz upita prevodi se na jednu formulu SF-Logike, vodeći računa da li su definisani selektori ili ne. Iz FROM-bloka upita definiše se pripadnost pripadnost promjenljivih klasama, dok OID OF dio upita definiše argumente konstruktora novog objekta. Promjenljive koje se dobiju prevodenjem složenih izraza mogu se uporedivati primjenom standardnih operatora poredenja, ako su u pitanju skalarne vrijednosti ili primjenom skupovnih relacija definisanih na početku ovog poglavlja, u skladu sa WHERE-blokom polaznog upita. U slučaju kada se vrši uporedivanje skupovnih atributa, prvo se kreiraju novi objekti koji predstavljaju skupove objekata, pa se oni uporeduju primjenom relacija *equal*, *subset* i *contains*. Tako su u upitu 9 prvo kreirani objekti $obj_1[attr_1 \rightarrow\rightarrow \{'plava','crna'\}]$ i $obj_2(X)[attr_2 \rightarrow\rightarrow B]$, koji su kasnije uporedeni ($contains(obj_2(X), attr_2, obj_1, attr_1)$). ◊

3. Implementacija

U ovom poglavlju opisane su osnovne karakteristike programa *SFLogika* koji vrši prevodenje upita jezika QLO u odgovarajuće programe SF-Logike. Program simulira rad objektno-orijentisanog sistema za upravljanje bazama podataka, u dijelu koji je vezan za parsiranje ulaznog upita. Umjesto procesiranja upita, to jest njegovog izvršavanja, program transformiše polazni upit u niz klauzula i upita SF-Logike. Program *SFLogika* napisan je za Windows 95 radno okruženje, primjenom programskog paketa Microsoft Visual C++ 4.0. Od mogućih modela programa, odabran je tip aplikacije zasnovan na dijalozima (engl. - "dialog based"), pomoću kojih korisnik unosi tražene podatke.

3.1. Leksičke konvencije

Razmatramo pet leksičkih klasa: identifikatore, ključne riječi, konstante, operatori i ostali separatori. Blanko, tabulatori i oznake novog reda (jednom rječju bjeline) se zanemaruju, osim kad razdvajaju simbole. Razmak je ponekad potreban da razdvoji susjedne identifikatore, ključne riječi ili konstante. Identifikator je niz slova i cifara; prvi znak mora biti slovo. Velika i mala slova se ne razlikuju. Svakom identifikatoru, u zavisnosti od sintaksne pozicije u upitu, odgovara jedan od sledećih tokena: ID, IME_KLASE, IME_METODA ili IME_ATRIBUTA.

Ključne riječi jezika QLO su:

SELECT	AS	COUNT	SUM
MAX	MIN	AVG	FROM
OID	OF	WHERE	AND
OR	NOT	IN	SUBSET
CONTAIN	FORALL	EXIST	DELETE
UPDATE	CREATE	ADD	RENAME
SET	CLASS	VIEW	INDEX
SUPRECLASS	SUBCLASS	OVERALP	DISJOINT
NAME	NEW	INHERITS	COMPLEX
SHARED	DEPENDENT	DEFAULT	TO
CODE	LANGUAGE	FILE	

Svakoj ključnoj riječi odgovara po jedan token. Na primjer, ključnoj riječi SELECT odgovara token SELECT, dok ključnoj riječi DELETE odgovara token BRISANJE.

Konstante mogu biti brojne ili znakovne. Brojne konstante se dalje dijele na cjelobrojne i razlomljene. Cjelobrojna konstanta je niz cifara. Razlomljena konstanta sastoji se od decimalnog dijela (koji je cjelobrojna konstanta), decimalne tačke i razlomljenog dijela (koji je niz cifara). Znakovna konstanta je niz od jednog ili više znakova koji se nalaze izmedu jednostrukih navodnika (na primjer, 'Marko Rudan'). Ako se unutar navodnika nalazi znak ', \ ili znak " , tada se oni označavaju dodavanjem

obrnute kose crte \ , kao u programskom jeziku C (to jest \' , \" ili \\ )[16]. Brojnoj konstanti odgovara token BROJ, a znakovnoj konstanti token LITERAL.

Operatori su : $, #, &, +, -, *, /, <, >, = , <>, <=, >= i ->. Prva tri operatora su unarna. Operator '*' može biti unarni (u slučaju kada označava tranzitivno zatvorenje) ili binarni (kada označava množenje). Operatori + , - i / označavaju standardne aritmetičke operacije. Operator -> označava tip metoda ili atributa. Ostali operatori su uobičajeni relacioni operatori.

Separatori su sve vrste zagrada, zarez - ',' , tačka-zarez - ';' i dvije tačke - ':'. Svakom od operatora i separatora odgovara po jedan token iz skupa *Tokens*. Svi tokeni prikazani su na slici 3.1.

```
enum Tokens {COMMA = ',' , DOT = '.' , DDOT = ':' , SCOMMA = ';' ,
             DOLAR = '$' , HASH = '#' , ADRESS = '&' ,
             PLUS = '+' , MINUS = '-' , MULT = '*' , DIV = '/' ,
             LT = '<' , GT = '>' , EQ = '=' ,
             LMZ = '(' , DMZ = ')' , LSZ = '[' , DSZ = ']' , LVZ = '{' , DVZ = '}' ,
             LE , NE , GE , ARROW ,
             ID , LITERAL , BROJ , ATTRIBUTE , METHOD , CLASS ,
             IME_KLASE , IME_METODA , IME_ATRIBUTA ,
             SELECT , FROM , WHERE , OID , OF , AS ,
             COUNT , MIN , MAX , AVG , SUM,
             BRISANJE , IZMJENA , KREIRANJE , ADD , NAME , RENAME ,
             VIEW , INDEKS , NADKLASA , PODKLASA , DISJOINT , OVERLAP ,
             ATTRIBUTES , METHODS , INHERITS , SET , TO ,
             COMPLEX , SHARED , DEPENDENT , DEFAULT ,
             CODE , LANGUAGE , DATOTEKA ,
             KONJ, DISJ, NEGA, FORALL, EXIST, ELEM , SADRZI , PODSKUP,
             ON, DROP, NEW , END_STRING };
```

Slika 3.1.

3.2. Sintaksna analiza

Gramatika iz poglavlja 2.4. transformisana je uklanjanjem lijeve rekurzije i lijevim faktorisanjem. Konstruisan je rekurzivni prediktivni parser za tako dobijenu gramatiku. Prvo su kreirane klase *CEntry* i *CSistemskaTabela*, koje su pomoćne klase za klasu Parsing koja, u stvari, realizuje parser. Objekat klase *CEntry* predstavlja jedan zapis u tabeli simbola. Klasa *CSistemskaTabela* je realizovana kao dvostruko povezana lista objekata tipa *CEntry*. Izgled klasa *CEntry* i *CSistemskaTabela* prikazan je na slici 3.2. Član klase *m_tip* je vrijednost odgovarajućeg tokena, dok se sama trenutna leksema čuva u članu *m_leksema*. Ako je tekući token IME_METODA ili IME_ATRIBUTA, tada je vrijednost člana *m_klasa* ime klase kojoj pripada metod/atribut, a *m_domen* sadrži klasu kojoj pripada rezultat poziva metoda/atributa. *m_Signatura* opisuje signaturu metoda i služi za provjeru tipa. *m_multi* dobija vrijednost "->", ako je metod skalarni, ili "->->", ako je metod skupovni. Ako je tekući token IME_KLASE, tada članovi *m_nadklase* i *m_klasa* sadrže spisak nadklasa date klase i ime klase, respektivno. U slučaju kada je tekući token ID, *m_klasa* je ime klase čiji objekti mogu biti vrijednost promjenljive.

```
class CEntry : public CObject
{
        // Construction
        public:
                CEntry();

        // Attributes
        public:
                int m_tip ;
                CString m_leksema ;
                CString m_multi;
                CString m_domen;
                CString m_klasa;
                CString m_Signatura;
                CString m_nadklase;
        public:
                virtual ~CEntry();
};
class CSistemskaTabela : public CObList
{
        public:
                void Install(CEntry *pEntry);
                CEntry* GetHeadEntry() ;
                CEntry* GetNextEntry(POSITION);
                CEntry* Trazi(CString arg);
};
```

Slika 3.2 .

Leksički analizator realizovan je kao funkcija članica klase *Parsing* - funkcija *GetNextToken()*[17]. Pored funkcija koje realizuju pravila gramatike, klasa *Parsing* sadrži i funkcije za upis elementa u tabelu simbola i njenu inicijalizaciju (funkcije *AddEntry(CEntry * pEntry)* i *PopuniTabeluSimbola()*), kao i funkciju *Match(int)*, koja vrši provjeru da li se tekući simbol poklapa sa simbolom koji sintaksno odgovara pravilu gramatike. Izgled klase *Parsing* prikazan je na slici 3.3.

```
class Parsing : public CObject
{
        // Konstruktor i destrukutor

        public:
                Parsing();
                ~Parsing();

        // Članovi klase
```

Slika 3.3.

```
CString m_upit;
int tekuci_simbol;
CString tekuca_leksema;

CString flogic;
CString head;
CString hier;
CString pom;
CString path;
CString pom_lista;
CString poredi;
CString last_var;
CString greska;
CString poredi_skupove;
CString skup1;
CString skup2;

int tekuci_broj;
int brojac;

BOOL m_NadklasaFlag;
BOOL m_Create;

CSistemskaTabela tabela; // Sistemska tabela

// Metodi klase

int Parsing::GetNextToken();
void Parsing::Match( int );
void Parsing::Retract();
void Parsing::PopuniTabeluSimbola();
void Parsing::AddEntry(CEntry *pEntry);

void Parsing::Naredba();
void Parsing::SelectNaredba();
void Parsing::SelectLista();
void Parsing::SelectListaOstatak();
void Parsing::SelectBlok();
void Parsing::AliasOpc();
void Parsing::SelectIzraz();

void Parsing::PathIzraz();
void Parsing::GlavniSelektor();
void Parsing::AtributskiIzrazOpc();
void Parsing::ListaAtrIzrazaOpc();
void Parsing::Func();
void Parsing::Alias();
```

Slika 3.3. (nastavak)

```
void Parsing::FromOpc();
oid Parsing::FromLista();
void Parsing::FromListaOstatak();
void Parsing::FromBlok();
void Parsing::PomocnoImeOpc();
void Parsing::OidFunkcijaOpc();
void Parsing::OidLista();
void Parsing::OidListaOstatak();

void Parsing::WhereOpc();
void Parsing::WhereUslov();
void Parsing::WhereUslovPrim();
void Parsing::Term();
void Parsing::TermPrim();
void Parsing::Faktor();
void Parsing::Prim();
void Parsing::NotOpc();
void Parsing::Predikat();
void Parsing::Poredjenje();
void Parsing::KvantOpc();
void Parsing::OstatakPoredjenja();
void Parsing::Izraz();
void Parsing::IzrazPrim();
void Parsing::AritmetickiTerm();
void Parsing::AritmetickiTermPrim();
void Parsing::AritmetickiFaktor();
void Parsing::AddOp();
void Parsing::MultOp();
void Parsing::ImeMetoda();

void Parsing::ImpJoinIzraz();
void Parsing::PocetniSelektor();
void Parsing::KlasnaPromjenljiva();
void Parsing::ListaMetodSelektorOpc();
void Parsing::ListaMetodSelektorOpcOstatak();
void Parsing::MetodskiBlok();
void Parsing::Izbor();
void Parsing::SelektorOpc();
void Parsing::MetMetoda();
void Parsing::MidOstatak();
void Parsing::MetodIliAtribut();
void Parsing::ImeAtributa();
void Parsing::PrefiksOpc();
void Parsing::NizAtributa();
void Parsing::Argument();
void Parsing::ListaArgumenata();
void Parsing::OstaliArgumenti();
```

Slika 3.3. (nastavak)

```
void Parsing::Selektor();
void Parsing::Oid();
void Parsing::ListaBrojevaIliLiterala();
void Parsing::ListaBrojeva();
void Parsing::ListaBrojevaOstatak();
void Parsing::ListaLiterala();
void Parsing::ListaLiteralaOstatak();
void Parsing::OidFunkcija();

void Parsing::DeleteNaredba();
void Parsing::DeleteNaredbaOstatak();
void Parsing::ImenovaniObjekat();

void Parsing::UpdateNaredba();
void Parsing::UpdateNaredbaOstatak();
void Parsing::ListaDodjela();
void Parsing::ListaDodjelaOstatak();

void Parsing::CreateNaredba();
void Parsing::CreateNaredbaOstatak();
void Parsing::ListaAtributa();
void Parsing::ListaAtributaOstatak();
void Parsing::NadklaseOpc();
void Parsing::PodklaseOpc();
void Parsing::DisjunktneKlaseOpc();
void Parsing::KlasePreklapanjaOpc();

void Parsing::NameNaredba();
void Parsing::OpisAtributa();
void Parsing::OpisAtributaOstatak();
void Parsing::DefinicijaAtributa();
void Parsing::TipAtributa();
void Parsing::DefinicijaAgregata();
void Parsing::NasledjujeOpc();
void Parsing::SlozeniObjekatOpc();
void Parsing::EkskluzivnostOpc();
void Parsing::ZavisnostOpc();
void Parsing::DefaultOpc();
void Parsing::OpisMetodaOpc();
void Parsing::ListaMetoda();
void Parsing::ListaMetodaOstatak();
void Parsing::SignaturaMetoda();
void Parsing::Rezultat();
void Parsing::TipMetoda();
void Parsing::ListaKlasa();
void Parsing::ListaKlasaOstatak();
```

Slika 3.3.(nastavak)

```
        void Parsing::AddNaredba();
        void Parsing::AddNaredbaOstatak();
        void Parsing::DodatniObjekat();
        void Parsing::MetaObjekat();
        void Parsing::AtributiOpc();
        void Parsing::DefinicijaNovogMetoda();
        void Parsing::ImeArgumenta();
        void Parsing::ListaParametara();
        void Parsing::ListaParametaraOstatak();
        void Parsing::Implementacija();
        void Parsing::Program();

        void Parsing::DropNaredba();
        void Parsing::NepotrebniObjekat();
        void Parsing::PreimenovanjeOpc();
        void Parsing::SpecifikacijaAtributa();
        void Parsing::SpecifikacijaAtributaOstatak();
};
```

Slika 3.3.(nastavak)

m_upit je član klase koji sadrži upit koji se prevodi. Članovi *tekuci_simbol* i *tekuca_leksema* čuvaju vrijednosti tokena koji se trenutno nalazi u ulaznom stringu i vrijednost njegove lekseme, respektivno. Članovi klase *m_Create* i *m_NadklasaFlag*, označavaju da li je ulazni string CREATE-naredba i da li se u upitu definišu nadklase date klase (definicija podklasa, klasa preklapanja i disjunktnih klasa nije programski realizovana). Svi ostali članovi klase definišu po jedan dio izlaznog stringa, dok string *flogic* predstavlja krajnji rezultat. Tako *head* označava glavu odgovarajuće SF-formule, dok string *poredi* sadrži sva poredjenja povezana logičkim veznicima && (AND) i || (OR) (na primjer, *X1<>12 && member(X2, o(X), attr1)*). String *hier* definiše pripadnost objekta klasi ili odnos klasa/podklasa (na primjer, *X:Osoba && Zaposleni::Osoba*); *pom_lista* služi za generisanje listi (lista argumenata, lista klasa i slično), a *pom* sadrži djelove SF-formula koji odgovaraju složenim izrazima jezika QLO.

PRIMJER 3.1.1. U jednom trenutku prevodenja upita :

```
SELECT X FROM Osoba X OID OF X
WHERE X.Metod1(12, 'as', 13) <> 14 AND X.Ime = 'Marko Rudan'
```

vrijednosti članova objekta *my* klase *Parsing* su :

my.head = "o(X)" ; *my.hier* = "X : Osoba" ; *my.poredi* = "*X1 <>14 && X2 = 'Marko Rudan'* " , *my.pom_lista* = "*12, 'as', 13*". String *pom* dobija dvije vrijednosti : prvo vrijednost "X[Metod1#12,'as',13->X1]" koja odgovara složenom izrazu X.Metod1(12, 'as', 13), a zatim vrijednost "X[Ime->X2]", koja odgovara složenom izrazu X.Ime . Tada je *my.path* = "X[Metod1#12,'as',13->X1] && X[Ime->X2]". Rezultat obrade datog upita smješta se u string *flogic*, tako da na kraju imamo:

my.flogic = "o(x)<-X:Osoba && X[Metod1#12,'as',13->X1] && X[Ime->X2] && X1 <>14 && X2 = 'Marko Rudan' ".

Inicijalizacija tabele simbola ostvaruje se unošenjem ključnih riječi , kao i podataka o metodima i klasama.

```
class CMetod : public CObject
{
        public:
                CMetod();
                DECLARE_SERIAL(CMetod)
        // Attributes
        public:
                CString m_ImeMetoda;
                CString m_Signatura;
                CString m_ImeKlase;
                CString m_Rezultat;
                CString m_Multi;
        public:
                virtual void Serialize(CArchive &ar);
                virtual ~CMetod();
};

class CListaMetoda : public CObList
{
        // Construction
        public:
                DECLARE_SERIAL(CListaMetoda)

        // Attributes
        public:
                void AddTailMetod (CMetod *pMetod);
                CMetod * GetHeadMetod () ;
                CMetod * GetNextMetod(POSITION);
                CMetod * CListaMetoda::Trazi(CString);
};
```

Slika 3.4.

Na slikama 3.4 i 3.5 prikazane su klase *CListaMetoda* i *CListaKlasa* koje u programu simuliraju sistemske klase objektno-orijentisanog sistema za upravljanje bazama podataka. Ove klase su realizovane kao dvostruko povezane liste, koje sadrže pokazivače na objekte klasa *CMetod* i *CKlasa,* respektivno. Takođe, ove klase su napravljene tako da omogućavaju serijalizaciju, to jest snimanje svog trenutnog stanja u datoteku na disku (makro *DECLARE_SERIAL(CListaMetoda)*). Funkcije članice su u oba slučaja iste: funkcije koje vrše pretraživanje kroz listu i umetanje objekta na kraj liste. Kako su klase izvedene iz MFC klase *CObjectList*, to one nasleđuju funkcionalnost te klase. Značenje pojedinih članova klasa *CMetod* i *CKlasa* već je objašnjeno u dijelu koji se odnosi na tabelu simbola.

```
class CKlasa : public CObject
{
        public:
                CKlasa();
                DECLARE_SERIAL(CKlasa)
                CString m_ImeKlase;
                CString m_Nadklase;
                virtual ~CKlasa();
                virtual void Serialize(CArchive &ar);
};

class CListaKlasa : public CObList
{
        // Construction
        public:
                DECLARE_SERIAL(CListaKlasa)

        // Attributes
        public:
                void AddTailKlasa (CKlasa *pKlasa);
                CKlasa * GetHeadKlasa () ;
                CKlasa * GetNextKlasa(POSITION);
                CKlasa* CListaKlasa::Trazi(CString);
};
```

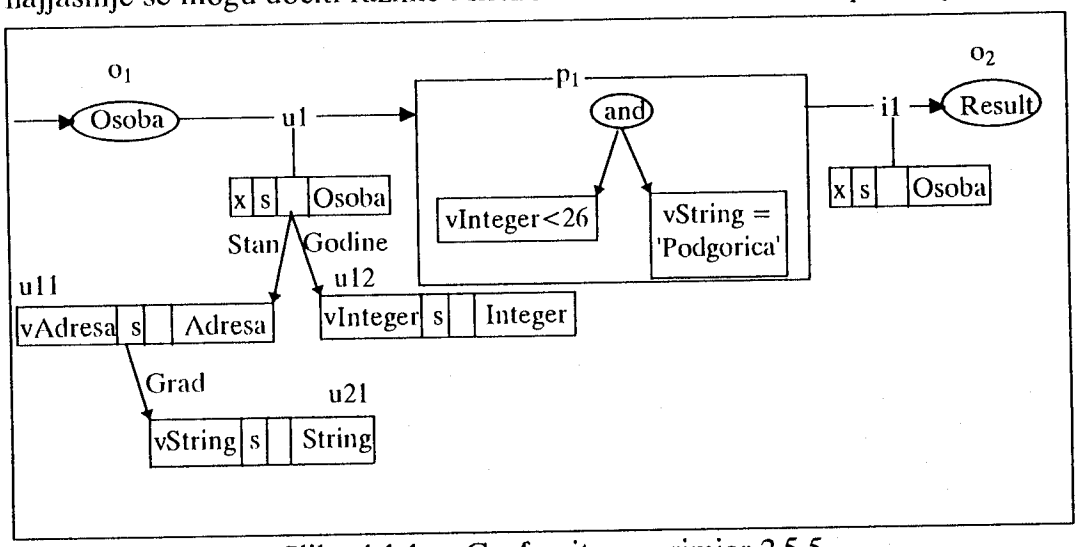
Slika 3.5

4. Strategija izvršavanja upita

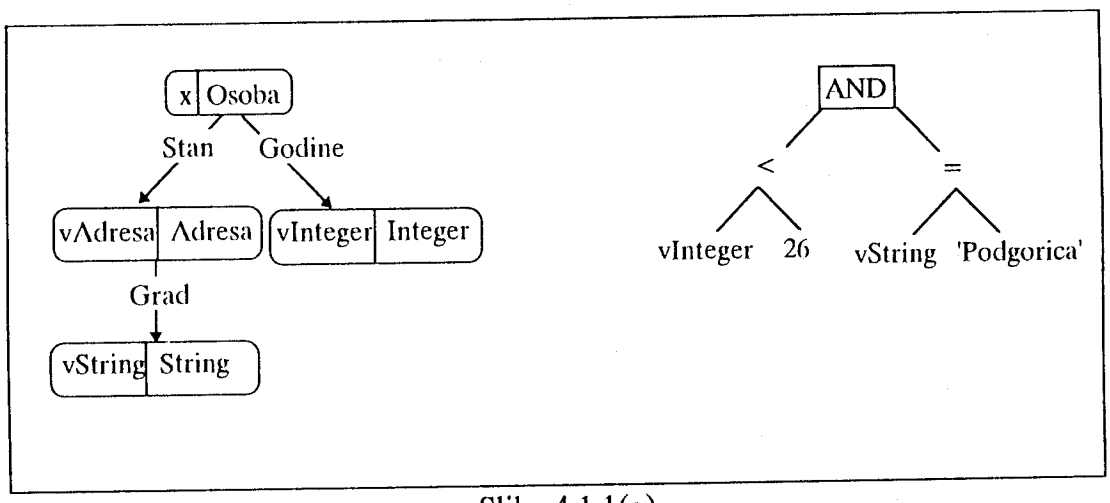
Razvojem deklarativnih upitnih jezika za objektno-orijentisane baze podataka, pitanje optimizacije dobilo je značajan prioritet u istraživanjima. Osnova procesa optimizacije je pronalaženje plana izvršavanja upita koji minimizuje funkciju cijene. Proces optimizacije najčešće uključuje dva povezana nivoa - logički i fizički ([2]) . Logički nivo koristi semantičke osobine jezika u cilju nalaženja upita datog jezika (ili nekog medujezika) koji je ekvivalentan polaznom upitu (engl. - "query rewriting"). Fizički nivo koristi neki model cijene (engl. - "cost based model"), koji je baziran na sistemskim informacijama, za određivanje najboljeg algoritma (ili algoritama) za izvršavanje izabranog izraza. Poseban aspekt optimizacije predstavlja izbor najboljeg plana izvršavanja, jer se primjenom navedenih tehnika može dobiti više najboljih planova izvršavanja. U ovom poglavlju predložena je jedna algebarska formalizacija u obliku grafa upita. Graf upita je prvi korak u transformaciji upita sa termina konceptualnog nivoa na termine fizičkog nivoa.

4.1. Graf upita

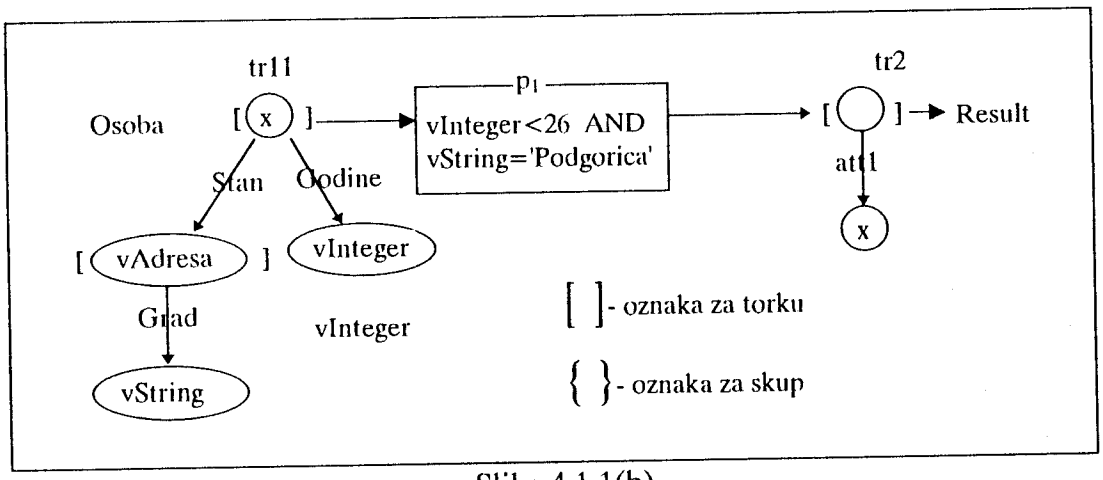
Svaki upit može biti predstavljen grafom upita. Graf upita je kombinacija grafičke reprezentacije objektne algebre iz [2] i SystemGraph-a koji se koristi za reprezentaciju upita u deduktivnim bazama podataka [12]. Za razliku od grafičke reprezentacije u [2], gdje se za svaki upit generišu najmanje dva grafa (jedan koji predstavlja strukturu predikata i jedan ili više grafova koji predstavljaju strukturu ulaza i izlaza), u našoj reprezentaciji koristi se samo jedan povezan graf. Takođe je uveden pojam iteratora, pojavljuju se metodi u složenim izrazima a skupovni atributi se tretiraju način različit od SystemGraph-a. Na slici 4.1.1 prikazan je graf upita za primjer 2.5.5. Na slici 4.1.1(a) prikazana je grafička reprezentacija navedenog upita, u skladu sa pravilima definisanim u [2]. Slika 4.1.1(b) predstavlja SystemGraph interpretaciju datog upita. Upoređujući ove tri slike, najjasnije se mogu uočiti razlike i sličnosti između ove tri interpretacije.



Slika 4.1.1 - Graf upita za primjer 2.5.5.



Slika 4.1.1(a)

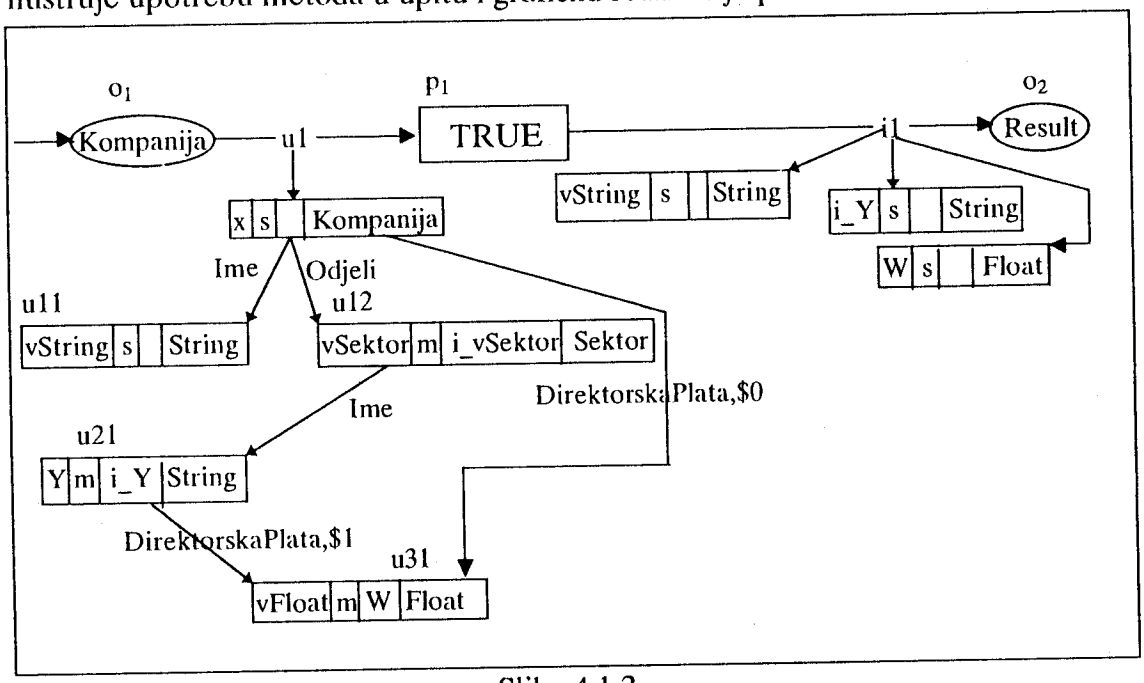


Slika 4.1.1(b)

Graf upita ima tri vrste čvorova : osnovne, parametarske i predikatske. Osnovni čvorovi grafa su imena klasa konceptualne šeme ili imena klasa koje čuvaju privremene rezultate. Na slici 4.1.1, osnovni čvorovi su o_1 i o_2. Osnovni čvor koji je odredište izlazne grane predikatskog čvora naziva se izlazni čvor (u našem slučaju to je o_2). Predikatski čvor (čvor p_1) ima jednu ili više ulaznih grana koje polaze iz osnovnih čvorova, jednu izlaznu granu i logički izraz. Predikatski čvor je označen stablom koje predstavlja strukturu predikata. Parametarski čvor sadrži vrijednost atributa i tip argumenta nekog metoda koji se pojavljuje u upitu (na slici 4.1.1. ne postoji parametarski čvor). Izlazni čvor označen je stablom koje predstavlja tip rezultata. Izlazno stablo je dubine 1, jer se sva potrebna izračunavanja izvršavaju u strukturnim grafovima. Svaka od ulaznih grana predikatskog čvora označena je grafom koji predstavlja objekte koji su potrebni u predikatu ili izlaznoj grani predikata i koji nazivamo strukturnim grafom. Strukturni graf je u stvari labela ulazne grane predikatskog čvora i određen je konceptualnom šemom baze podataka tj. strukturom klasa koje se pojavljuju u upitu (na slici 4.1.1., graf u1 određen je strukturom klase Osoba). Svaki čvor strukturnog grafa ima sledeću strukturu:

ime_promjenljive	*tip_promjenljive*	*iterator*	*ime_klase*

Čvor strukturnog grafa predstavlja skup objekata S koji su instance klase *ime_klase*. Prvo polje *ime_promjenljive* je promjenljiva koja za vrijednosti uzima elemente skupa S; *tip_promjenljive* je *s* ("scalar" - ako je atribut skalaran) ili *m* ("multivalued" - ako je atribut skupovni). Ako je *tip_promjenljive m*, tada polje *iterator* može da sadrži promjenljivu koja služi za iteraciju kroz skup vrijednosti skupovnog atributa. Ovo polje se uglavnom koristi za one predikate u kojima se pojavljuju egzistencijalni ili univerzalni kvantifikatori (FORALL i EXISTS). Ako je polje *tip_promjenljive s*, tada je polje *iterator* NIL. Grane strukturnog grafa označene su imenima atributa klasa iz šeme ili imenima metoda i oznakom broja argumenta. Na slici 4.1.2. prikazan je graf upita za primjer 2.5.24, koji ilustruje upotrebu metoda u upitu i grafičku realizaciju poziva metoda.



Slika 4.1.2.

Primjetimo da strukturni graf postaje stablo u slučaju kada u složenom izrazu ne postoji selektor koji je metod (tj. svi selektori su atributi ili individualni objekti). Ako je *meth* metod koji se pojavljuje kao selektor u nekom složenom izrazu i ako je tom metodu pridružen tip $T_0, T_1, \cdots, T_n \to R$ (*ili* $T_0, T_1, \cdots, T_n \to\to R$), tada je grana koja vodi od čvora čija je klasa T_0 do čvora čija je klasa R označena sa *meth,$0*. Za svaki argument metoda *meth* postoji grana od čvora koji predstavlja taj argument (čija je klasa T_i) do čvora koji predstavlja rezultat izvršavanja metoda (čija je klasa R), označena sa *meth,$i* . Ako je T_i primitivna klasa, vrijednost atributa se nalazi u parametarskom čvoru, koji ima sledeću strukturu:

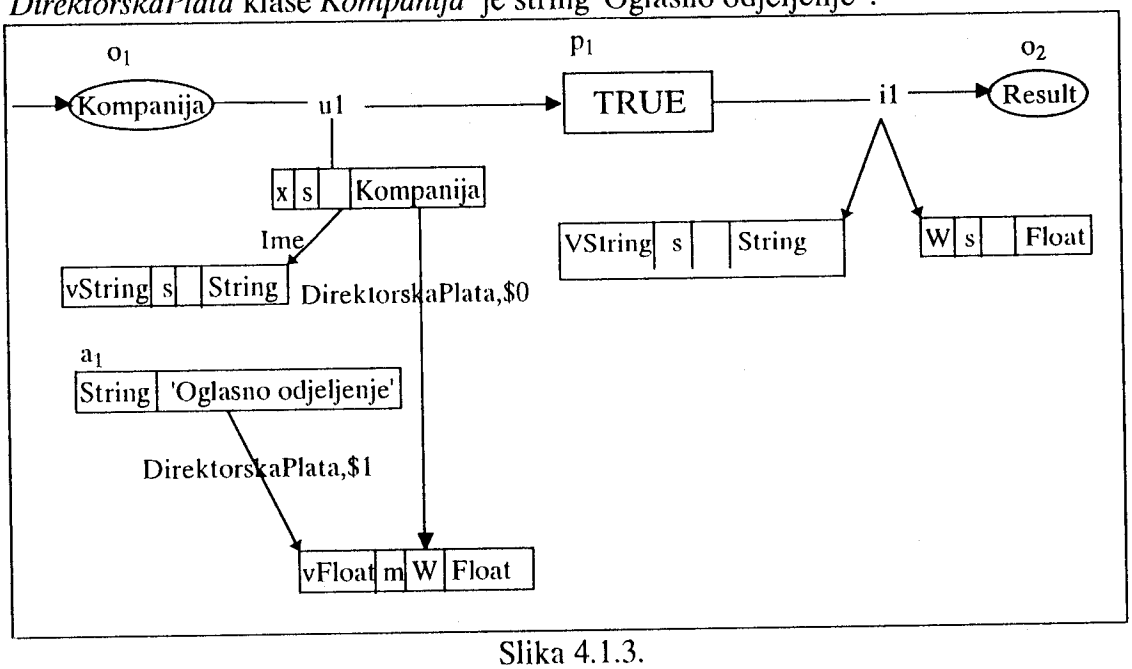
tip_argumenta	*vrijednost_argumenta*

Polje *tip_argumenta* označava primitivnu klasu (na primjer : String, Float, Integer i slično) dok polje *vrijednost_promjenljive* sadrži vrijednost argumenta

metoda. Iz jednog parametarskog čvora može polaziti više grana. Na slici 4.1.3. prikazan je graf upita za sledeću naredbu upitnog jezika :

```
SELECT X.Ime, W
FROM Kompanija X
WHERE X.DirektorskaPlata('Oglasno odjeljenje')[W]
```

Ovaj upit je jednostavnija verzija upita iz primjera 2.5.24.; rezultat upita su imena kompanija i plate direktora oglasnih odjeljenja. Argument metoda *DirektorskaPlata* klase *Kompanija* je string 'Oglasno odjeljenje' .



Slika 4.1.3.

Čvor grafa označen sa a_1 je parametarski čvor koji odgovara stringu 'Oglasno odjeljenje' (tj. instanci 'Oglasno odjeljenje' primitivne klase *String*).

Graf upita G je skup $\{(IzlazniCvor_i, p_i) \mid i = 1, \cdots, n\}$, gdje je $IzlazniCvor_i$ ime i-tog izlaznog čvora, p_i je oznaka predikatskog čvora a n je broj predikatskih čvorova. Svaki od predikatskih čvorova grafa upita G možemo predstaviti na sledeći način: Pred(SkupUlaznihGrana, Predikat, IzlaznoStablo). Svaku ulaznu granu možemo predstaviti uređenim parom *(O, sg)*, gdje je *O* ime osnovnog čvora, a *sg* je strukturni graf koji je pridružen toj grani. Strukturni graf opisujemo kao uredenu četvorku *ImeCvora=(Klasa, ImePromjenljive, ImeIteratora, IzlazneGrane)*. Polje *IzlazneGrane* uređene četvorke je skup oblika $\{(Naslednik_i, Attr, Broj) \mid i = 1, \cdots, k\}$, gdje je k broj izlaznih grana čvora *sg*, $Naslednik_i$ je oznaka i-tog naslednika (tj. završnog čvora i-te izlazne grane) a *Attr* oznaka grane koja vodi od čvora *ImeCvora* do čvora $Naslednik_i$. Polje *Broj* je nedefinisano (NIL) ako je *Attr* ime atributa; ako je *Attr* ime metoda i ako je dati čvor j-ti argument metoda *Attr*, tada polje *Broj* ima vrijednost j (j=0, 1, 2, ..., mk ; mk - broj argumenata metoda *Attr*). Svaki od podgrafova strukturnog grafa *sg* predstavljen je sličnom uređenom četvorkom. Za one podgrafove koji nemaju izlaznih grana, polje *SkupNaslednika* je ∅ (prazan skup). Polja *Klasa,*

ImePromjenljive i *ImeIteratora* preuzimaju se direktno iz odgovarajućeg čvora grafa upita.

Na primjer, graf G1 sa slike 4.1.1. je predstavljen skupom $\{(Result, p_1)\}$, jer postoji samo jedan predikatski čvor. Predikatski čvor p_1 opisan je skupom ulaznih grana $\{(Osoba, u1)\}$, predikatom (gr = 'Podgorica' and g<26) i izlaznim stablom i1, tj. predikatski čvor p_1 je :

Pred($\{(Osoba, u1)\}$; (gr = 'Podgorica' and g<26); i1).

Strukturni graf u1 kojim je označena jedina ulazna grana predikatskog čvora p_1 opisan je četvorkom $u1 = (Osoba, x, Nil, \{(u11, Stan, Nil), (u12, Godine, Nil)\})$; podgraf u11 grafa u1 opisan je četvorkom $u11 = (Adresa, st, Nil, \{(u21, Grad, Nil)\})$ u12 sa: $u12 = (Integer, g, Nil, \varnothing)$ dok je $u21 = (String, gr, Nil, \varnothing)$. Na isti način, u grafu G2 sa slike 4.1.2. imamo da je :

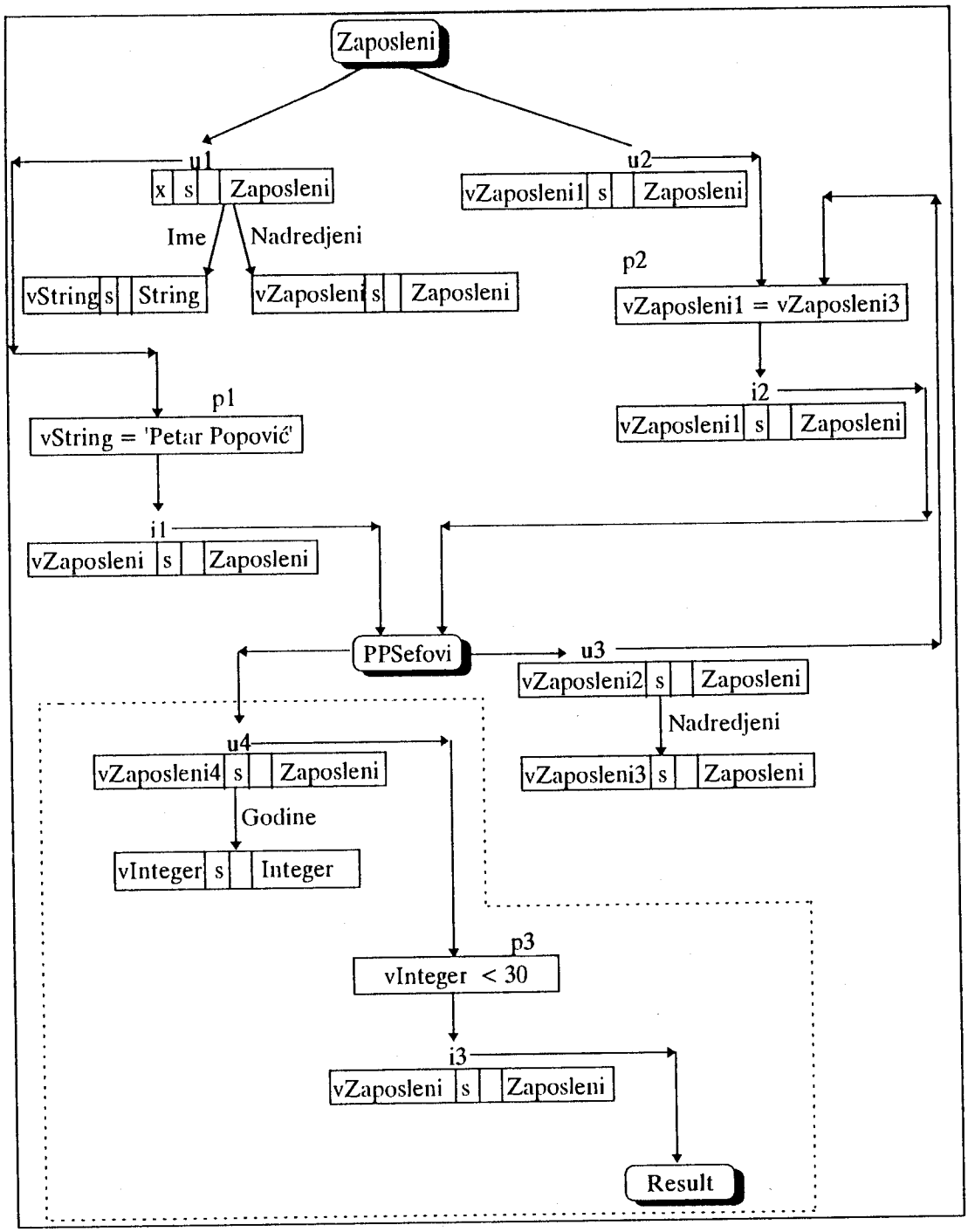
$u1 = (Kompanija, x, Nil, \{(u11, Ime, Nil), (u12, Odjeli, Nil), (u31, DirektorskaPlata, 0)\})$

$u21 = (String, im, i\_im, \{(u31, DirektorskaPlata, 1)\})$.

Predstavljeni su samo čvorovi za u1 i u21, dok se četvorke za ostale čvorove dobijaju na isti način.

4.2. Graf upita za rekurzivne upite

Prevodenje rekurzivnih upita u odgovarajući graf je nešto složenije nego kod nerekurzivnih upita. Za svaki rekurzivni upit definišu se dva predikatska čvora: baza rekurzije (polazni uslov koji objekat treba da zadovoljava) i rekurzivni korak (uslov povezivanja novodobijenih objekata sa već izračunatim). Na slici 4.2.1 prikazan je graf upita za upit iz primjera 2.5.34 . Baza je predikatski čvor p1 dok je rekurzivni korak predstavljen predikatskim čvorom p2. Baza je polazni uslov iz Where-bloka datog upita (u našem slučaju X.Ime = 'Petar Popović'), dok se rekurzivni korak definiše na osnovu atributa koji su uključeni u izraz $*(Att_1.Att_2.\cdots.Att_n)$. Primjetimo da u grafu sa slike 4.2.1 postoji grana koja polazi iz osnovnog (izlaznog) čvora PPSefovi i vodi ka predikatskom čvoru p2. Ova grana je grafički prikaz izraza *(Nadredjeni). Na taj način ostvaruje se rekurzivni korak jer se instance iz čvora PPSefovi spajaju sa instancama klase (čvora) Zaposleni u skladu sa predikatom vZaposleni1= vZaposleni2 (tj. vrijednost atributa *Nadredjeni* nekog objekta iz čvora PPSefovi treba da je jednaka OID-u nekog objekta čvora Zaposleni). U opštem slučaju, grana koja vodi od izlaznog čvora R do nekog od predikatskih čvorova koji imaju granu koja vodi u R, predstavlja grafički prikaz izraza $*(Att_1.Att_2.\cdots.Att_n)$. Dio grafa koji nije uokviren predstavlja definiciju rekurzivnog pogleda PPSefovi. Dio grafa uokviren isprekidanim linijama predstavlja selekciju datu u primjeru 2.5.34. (tj. X.Godine<30).



Slika 4.2.1

4.3. Prevođenje upita u graf

Prevođenje upita u odgovarajući graf upita izvodi se na jednostavan način. Osnovni čvorovi grafa dobijaju se iz FROM-bloka upita na sledeći način : svakoj promjenljivoj ili imenu klase iz FROM-bloka koja je početni selektor nekog složenog izraza u WHERE-bloku upita a kojoj već nije pridružen čvor, pridružujemo po jedan osnovni čvor. Svaka od grana koja polazi iz osnovnog čvora označena je strukturnim grafom koji se konstruiše na osnovu strukture

klase osnovnog čvora (atributa i metoda klase) i strukture ulaznog predikata. Neka je K klasa za koju je konstruisan osnovni čvor O. Korijen strukturnog grafa je čvor u1 takav da je :

$$u1.tip\_promjenljive = s$$
$$u1.iterator = \text{nil}$$
$$u1.ime\_klase = K$$

Ime promjenljive je ili alias zadat u FROM-bloku upita (na primjer, ako je FROM-blok oblika "FROM Osoba X" , ime promjenljive je X) ili se automatski generiše dodavanjem prefiksa *v* imenu klase. Ako postoji više čvorova sa istim imenom promjenljive, razlikovaćemo ih dodavanjem sufiksa 1, 2, itd. Za svaki izraz oblika K.A ili X.A (koji se nalazi ili u SELECT-bloku ili u WHERE-bloku neke SELECT naredbe), gdje je A ime atributa klase K, konstruišemo čvor u2 takav da :

$$u2.tip\_promjenljive = \begin{cases} s, & \text{ako je A skalaran,} \\ m, & \text{ako je A skupovni} \end{cases}$$
$$u2.ime\_klase = C$$

Čvorovi u1 i u2 spojeni su granom označenom sa A, usmjerenom od u1 ka u2. Ako postoji selektor koji je promjenljiva (na primjer, K.A[W] ili X.A[W]), tada je u2.*ime_promjenljive* = W, ako je atribut sklaran; ako je atribut skupovni, tada je u2.*iterator* = W. Ako već nije definisano, ime iteratora generiše se dodavanjem prefiksa *i_* na ime promjenljive; na isti način, ako ime promjenljive nije definisano, generišemo ga dodavanjem prefiksa v imenu klase. Kompozicija više atributa realizuje se na isti način. Na primjer, za složeni izraz X.A.B, pored ranije konstruisanih čvorova u1 i u2, konstruišemo i treći čvor u3 koji je spojen sa u2 granom označenom sa B, gdje je B neki atribut klase C. Ako je promjenljiva pridružena čvoru u2 skupovnog tipa (tj. *u2.tip_promjenljive* = *m*), tada je *u3.tip_promjenljive* = *m* , bez obzira da li je B skalaran ili skupovni atribut. Postupak se ponavlja za svaki složeni izraz u WHERE-bloku ili SELECT-bloku upita.

Neka je K klasa za koju je konstruisan osnovni čvor i neka je korijen strukturnog grafa u1; neka je *Meth* neki metod klase K, sa argumentima $Arg_1, \cdots, Arg_n$ koji kao rezultat daje instancu (ili skup instanci) klase C. Za složeni izraz K.Meth($Arg_1, \cdots, Arg_n$) konstruišemo čvor u4 takav da je :

$$u4.tip\_promjenljive = \begin{cases} s, & \text{ako je } Meth \text{ skalaran,} \\ m, & \text{ako je } Meth \text{ skupovni} \end{cases}$$
$$u4.ime\_klase = C$$

Polja *ime_promjenljive* i *iterator* čvora u4 generišu se na isti način kao i za čvorove u1, u2 i u3. Svaki od argumenata može biti broj, literal, skup brojeva, skup literala, imenovani objekat ili drugi složeni izraz. Ako je argument Arg_i složeni izraz, tada postoji čvor strukturnog grafa, naprimjer ui1, koji označava vrijednost

tog izraza; od čvora ui1 do čvora u4 postoji grana označena sa *Meth,$i*. Takode, od čvora u1 do čvora u4 vodi grana označena sa *Meth,$0*. Ako Arg_i nije složeni izraz , tada za Arg_i konstruišemo parametarski čvor *pi1* takav da :

pi1.tip_argumenta = TipArgumenta

pi1.vrijednost_argumenta = Arg_i

Od čvora pi1 do čvora u4 vodi grana označena sa *Meth,$i*. Primjetimo da se imenovani objekti tretiraju na isti način kao i primitivni objekti (brojevi, literali ili skupovi brojeva ili literala). Medutim, kako je u definiciji signature metoda dozvoljeno da tip argumenta može biti samo klasa, to će argument koji je imenovani objekat čiji tip nije klasa (na primjer, dobijaju se projekcijom iz neke klase) dovesti do greške u tipu.

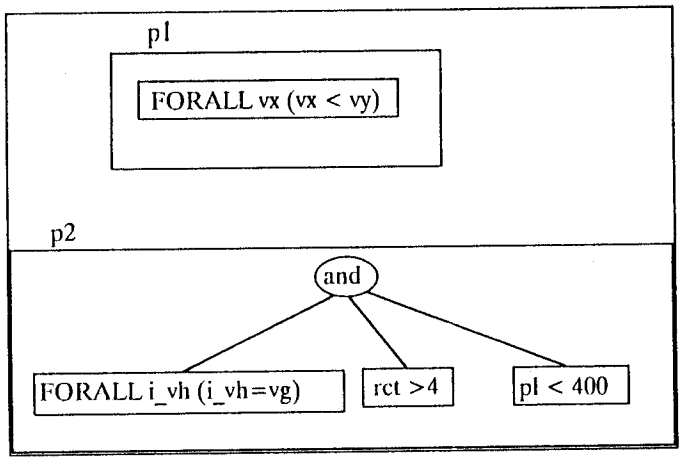
Polje *iterator* čvora strukturnog grafa ne mora uvijek biti definisano, čak i kada je tip promjenljive skupovni (tj. *m*). To polje definišemo u sledećim slučajevima :

- ako u složenom izrazu $sel_0.MetIzraz_1[sel_1]\cdots.MetIzraz_n[sel_n]$ postoji bar jedan skupovni atribut/metod (na primjer, $MetIzraz_i$) i postoji bar jedan od selektora $sel_j, j \geq i$ (svi selektori, osim prvog, su opcionalni);
- ako u složenom izrazu $sel_0.MetIzraz_1.\cdots.MetIzraz_n$ postoji bar jedan skupovni atribut/metod (na primjer, $MetIzraz_i$) i ispred tog izraza se nalazi kvantifikator (EXISTS ili FORALL).

U svim ostalim slučajevima polje *iterator* možemo ostaviti nedefinisanim (NIL).

Izlazni čvor grafa (*Result*) označen je nekim stablom i1. Broj čvorova stabla i1 jednak je broju složenih izraza u SELECT-bloku upita (svakom izrazu u SELECT-bloku odgovara po jedan čvor stabla). Svaki čvor izlaznog stabla označen je nekom promjenljivom koja je već definisana u nekom čvoru nekog od ulaznih strukturnih grafova. Polje *iterator* izlaznog stabla nije definisano. Ako je promjenljiva vx tipa *s* (tj. skalarna), tada čvor izlaznog stabla može imati tip *m* (skupovni tip), ako je u SELECT-bloku specificirano da je vrijednost rezultujućeg atributa skup OID-a (kao u upitu iz primjera 2.5.17). U svim ostalim slučajevima, tip promjenljive u izlaznom stablu poklapa se sa ranije dodijeljenim tipom te promjenljive. Ako su u SELECT-bloku upita navedena nova imena atributa za rezultujuće objekte, tada su grane izlaznog stabla označene novim imenima atributa (primjeri 2.5.15 i 2.5.16). Ako to nije slučaj, grane izlaznog stabla ostaju neoznačene.

Predikatski čvor konstruiše sa na uobičajeni način : svakom logičkom vezniku (AND, OR, NOT) odgovara po jedan unutrašnji čvor stabla. Iz čvorova stabla označenih sa AND ili OR polaze dvije ili više grana, dok čvor NOT može da ima samo jednu izlaznu granu. Listovi stabla su relacije poredenja izmedu promjenljivih definisanih u nekom od strukturnih grafova ili izmedu promjenljivih i OID-a. Predikatski čvorovi za upite iz primjera 2.5.10 (p1) i primjera 2.5.11 (p2) prikazani su na slici 4.3.1. Promjenljive *vx* i *vy* definisane su u nekom od strukturnih grafova koji označavaju neku od ulaznih grana čvora p1. Isto važi i za promjenljive *vg, i_vh, rct* i *pl* iz predikarskog čvora p2.



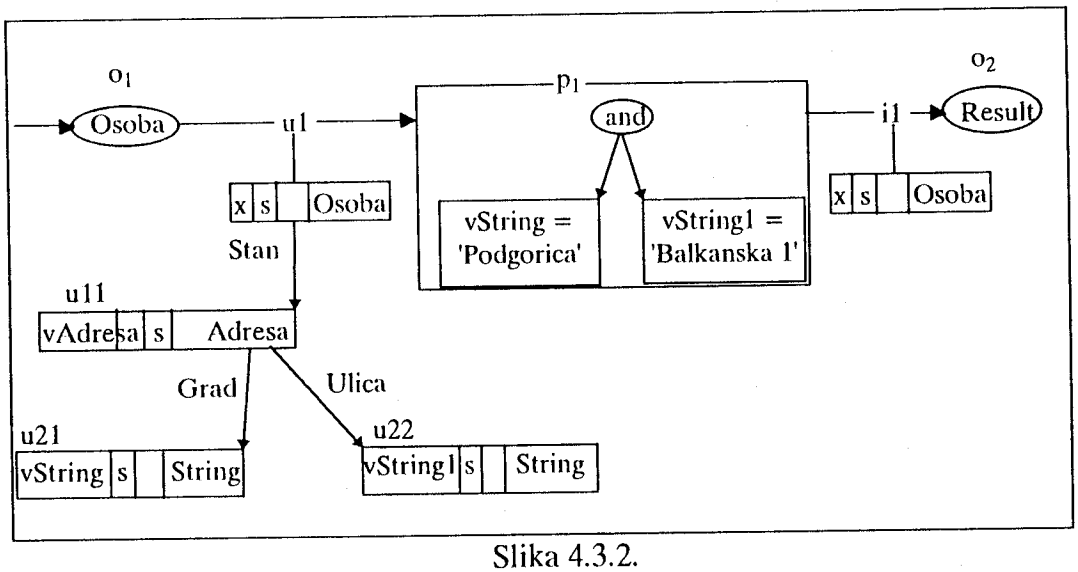
Slika 4.3.1

Zajednički prefiksi složenih izraza predstavljeni su samo jednom stazom u strukturnom grafu pa se na taj način postiže faktorizacija zajedničkih podizraza.

PRIMJER 4.3.1 Posmatrajmo sledeći upit jezika QLO:

```
SELECT X FROM Osoba X
WHERE X.Stan.Grad = 'Podgorica'
        AND X.Stan.Ulica = 'Balkanska 1'
```

Graf upita za ovaj upit prikazan je na slici 4.3.2.



Slika 4.3.2.

Zajednički prefiks složenih izraza X.Stan.Grad i X.Stan.Ulica pojavljuje se u grafu sa slike 4.3.2. kao jedna zajednička grana. ◊

5. Zaključak

Primjena objektno-orijentisanih koncepata je postala važna tema istraživanja u mnogim oblastima računarstva, kao što su programski jezici, baze podataka, reprezentacija znanja, pa čak i arhitektura računara. Na žalost, postoji veliki stepen zabune oko toga šta znači pojam "objektno-orijentisan", a posebno šta su to "objektno-orijentisane" baze podataka. Konfuziju je izazvala činjenica da su se objektno-orijentisani principi razvijali u tri različite oblasti računarstava: prvo u programskim jezicima, zatim u vještačkoj inteligenciji i na kraju u bazama podatka. Uprkos svemu, razvijeno je više različitih modela podataka koji se deklarišu kao objektni ili objektno-orijentisani. Paralelno sa razvojem tih modela, projektovani su i deklarativni upitni jezici koji su, manje ili više uspješno, pokušali da obuhvate sve aspekte pojedinih modela. Kod deklarativnih upitnih jezika tradicionalno važnu ulogu ima proces optimizacije upita. Optimizacija upita u objektno-orijentisanim sistemima baza podataka složenija je nego kod relacionih sistema, što je posledica kompleksnije strukture objektno-orijentisanog modela.

U ovom radu predstavljen je jedan mogući pristup projektovanju deklarativnog upitnog jezika za objektno-orijentisane baze podataka i formalnom zasnivanju njegove semantike. Razvijena je logika koja služi kao podloga za definisanje formalne semantike upita. Prikazan je i način kreiranja jedne grafičke reprezentacije upita, koja može biti početni korak u procesu algebarske optimizacije upita.

Rad se sastoji od uvoda, četiri poglavlja, zaključka i priloga. Na kraju rada dat je opširan spisak referenci.

U prvoj glavi dat je opis jednog modela podataka, koji je osnova na kojoj definišemo upitni jezik. Iz F-Logike izvedena je nova SF-Logika, koja je posebno prilagođena bazama podataka. Sintaksa i semantika F-Logike obuhvataju neke koncepte objektno-orijentisanog pristupa koje nisu karakteristične za baze podataka. Postojanje tih koncepata posledica je činjenice da F-Logika daje formalni okvir ne samo za baze podataka, već i za objektno-orijentisane programske jezike i "frame-based" jezike za vještačku inteligenciju. Eliminisanjem tih koncepata i pojednostavljenjem sintakse (to jest, uvođenjem posebnih konstruktora za individualne, klasne i metodske objekte), dobijena je SF-Logika. Razlika između semantika F-Logike i SF-Logike je posledica navedenih izmjena u sintaksi. Neki pojmovi u semantici F-Logike koji su bili neformalno definisani, u SF-Logici su precizno definisani. Definisana je i formalna teorija SF-Logike, koja predstavlja modifikovanu formalnu teoriju F-Logike.

Druga glava definiše jezik QLO i njegovu vezu sa SF-Logikom. Izložena je formalna definicija složenog izraza, koji je modifikovana varijanta složenog izraza iz jezika XSQL. Definisana je i gramatika koja predstavlja sintaksu jezika QLO, dok su osnovne karakteristike jezika opisane većim brojem primjera. Prikazana je i formalna semantika jezika, kao i način prevođenja upita jezika QLO u odgovarajuće programe SF-Logike. Posebno poglavlje posvećeno je učaurenju i složenima izrazima koji poštuju taj koncept.

Treća glava opisuje program koji prevodi upite jezika QLO u odgovarajuće programe SF-Logike. Program je napisan za Windows radno okruženje, primjenom

programskog paketa *Visual C++*. Dat je pregled klasa koje su kreirane za realizaciju programa, kao i opis i namjena pojedinih članova i funkcija-članica tih klasa. Opisane su i neke glavne karakteristike programa.

U četvrtoj glavi prikazana je grafička reprezentacija upita, kao prvi korak u procesu algebarske optimizacije. Graf upita predstavlja kombinaciju QueryGraph-a za deduktivne baze podataka i grafičke reprezentacije upita u sistemu O2. Data je formalna definicija grafa kao i način kreiranja grafa iz upita. Ovaj graf je osnova za dalju transformaciju upita.

U prilogu je dat dio koda koji implementira najvažnije funkcije programa, dok je za ostale funkcije dat krtak opis. Program je napisan u programskom jeziku C++. Na priloženoj disketi nalazi se kompletan kod programa, izvršna verzija programa i datoteka "klasa.dat", kao i kratak vodič za rad sa programom.

Formalno definisanje semantike upitnog jezika za objektno-orijentisane baze podataka i dalje predstavlja otvoreno polje za istraživanja. Jedan od mogućih pravaca istraživanja može biti definisanje pojma tipa složenog izraza i čitavog upita ([2], [11]) i razmatranje kada je upit "tipovski" korektan. Ova razmatranja mogu biti početni korak u procesu optimizacije, jer ako upit nije "tipovski" korektan, tada nije potrebno izvršavati upit, jer sigurno dolazi do greške prilkom izvršavanja (engl. - "run-time error").

U F-Logici i SF-Logici strukturno nasleđivanje je ugrađeno u semantiku. Nasleđivanje ponašanja je mnogo složenije, zahvaljujući mogućnosti redefinicije metoda (engl. - "overriding") i višestrukom nasleđivanju. Nasleđivanje ponašanja može dovesti do nemonotonog (engl. - "non-monotonic") ponašanja. Neka je P program i neka su ϕ, φ formule logike L; ako iz $P \models \phi$ slijedi $P \wedge \varphi \models \phi$, tada kažemo da je logika L monotona. Za razliku od klasične predikatske logike, SF-Logika i F-Logika nisu monotone. Način prevazilaženja nemonotonosti SF-Logike je jedna od mogućnosti za dalje istraživanje.

Posebno atraktivnu oblast istraživanja predstavlja optimizacija upita. Graf upita definisan u poglavlju 4 može biti na različite načine transformisan u neku novu reprezentaciju koja će u obzir uzeti i način fizičkog smještanja objekata na disku, postojanje indeksa i klastere ([2], [12]). Takođe je moguće definisati novu reprezentaciju koja će se umjesto entiteta na konceptualnom nivou (imenima klasa) baviti fizičkim entitetima - skupovima instanci klase ili podskupom tog skupa ([12]). Jednu mogućnost predstavlja i transformisanje grafa upita u programe SF-Logike (ili neke druge logike), kao i obrnut proces - pretvaranje programa u graf upita.

Prilog 1 - kôd programa

```
// realizacija klase Parsing , datoteka KlasaParsing.CPP

Parsing::Parsing()
{
}

Parsing::~Parsing()
{
        while( !tabela.IsEmpty( ) )
        {
                delete tabela.RemoveHead( );
        }
}

int Parsing::GetNextToken()
{
        int i , pocetak ;
        CString upit1  ;
        char c ;
        CEntry *TableEntry;

        upit1 = m_upit + '\0';

        i = 0;
        while ( 1 )
        {
                while ( upit1[i] == ' ' || upit1[i] == '\t' || upit1[i] == '\n' ) i++ ;
                if (isalpha(upit1[i]) )
                {
                        tekuca_leksima = "";
                        while (isalnum(upit1[i]))
                        {
                                tekuca_leksima = tekuca_leksima + upit1[i];
                                i++;
                        } // end while isalpha ...

                        m_upit = m_upit.Mid(i) ;

                        CString Lexima = tekuca_leksima;

                        Lexima.MakeUpper();
                        TableEntry = tabela.Trazi(Lexima);
                        if (TableEntry == NULL)
                        {
                                CEntry *NewEntry = (CEntry *) new CEntry;
                                NewEntry->m_leksima = Lexima;
                                NewEntry->m_tip = ID;
                                tabela.Install(NewEntry);
                                return ID;
```

```
        }
        else
                return TableEntry->m_tip;
} // end if isalpha ...
else
{
        if (isdigit(upit1[i]) )
        {
                pocetak = i;
                while (isdigit(upit1[i])) i++;
                if (upit1[i]  == '.')
                {
                        i++;
                        while (isdigit(upit1[i])) i++;
                }
                tekuca_leksima = upit1.Mid(pocetak, i - pocetak);
                m_upit = m_upit.Mid(i);
                tekuci_simbol = BROJ ;
                return BROJ ;
        } // end if isalpha ...
        switch ( upit1[i] )
        {
                case '-' :
                        i++;
                        if (upit1[i]=='>')
                        {
                                tekuca_leksima = "->";
                                i++;
                                m_upit = m_upit.Mid(i) ;
                                return ARROW ;
                        }
                        else
                        {
                                tekuca_leksima = upit1[i-1];
                                m_upit = m_upit.Mid(i) ;
                                return MINUS ;
                        }
                        break;
                 case '<' :
                        i++;
                        if (upit1[i] == '>')
                        {
                                tekuca_leksima = "<>";
                                i++;
                                m_upit = m_upit.Mid(i) ;
                                return NE ;
                        }
                        else if (upit1[i] == '=')
                                {
                                        tekuca_leksima = "<=";
                                        i++;
                                        m_upit = m_upit.Mid(i) ;
```

```
                                        return LE ;
                                }
                                else
                                {
                                        tekuca_leksima = upit1[i-1];
                                        m_upit = m_upit.Mid(i) ;
                                        return LT ;
                                }
                        break;
                case '>' :
                        i++;
                        if (upit1[i]=='=')
                        {
                                tekuca_leksima = ">=";
                                i++;
                                m_upit = m_upit.Mid(i) ;
                                return GE ;
                        }
                        else
                        {
                                tekuca_leksima = upit1[i-1];
                                m_upit = m_upit.Mid(i) ;
                                return GT ;
                        }
                case '\'' :
                        tekuca_leksima = upit1[i] ;
                        i++ ;
                        while ( upit1[i] != '\'' )
                        {
                                tekuca_leksima = tekuca_leksima + upit1[i];
                                i++;
                        } // end while
                        tekuca_leksima += upit1[i];
                        i++ ;
                        m_upit = m_upit.Mid(i) ;
                        return LITERAL ;
                default :
                        tekuca_leksima = upit1[i];
                        c = upit1[i];
                        if (c == '\0') return END_STRING;
                        i++;
                        m_upit = m_upit.Mid(i) ;
                        return c ;
                } // end switch
        } // end else isalpha ...
} // end while i < ...
}

void Parsing::Match( int arg )
{
        if ( arg == tekuci_simbol )
        {
```

```
                tekuci_simbol = GetNextToken();
        }
        else
        {
                AfxMessageBox("Leksicka greska!");
                greska = "Greska u leksickoj analizi!";
        }
}

void Parsing::Naredba()
{
        tekuci_simbol = GetNextToken();
        path = "";
        flogic = "";
        hier = "";
        poredi = "";
        brojac = 0;
        greska = "";
        m_Create = FALSE;
        switch(tekuci_simbol)
        {
        case SELECT :
                head = "o(" ;
                SelectNaredba();
                break;
        case IZMJENA :
                UpdateNaredba();
                break;
        case ADD:
                AddNaredba();
                break;
        case KREIRANJE:
                head = "" ;
                CreateNaredba();
                break;
        case NAME :
                NameNaredba();
                break;
        case DROP:
                DropNaredba();
                break;
        case BRISANJE:
                DeleteNaredba();
                break;
        default:
                break;
        }
}
void Parsing::SelectNaredba()
{
        if (greska != "")
                return;
```

```
        Match( SELECT );
        SelectLista();
        FromOpc();
        WhereOpc();
        if (skup1 != "")
                flogic += skup1 + '\n';
        if (skup2 != "")
                flogic += skup2 + '\n';
        flogic += head + hier;
        if (path != "")
                flogic += "  &&  " + path;
        if (poredi != "")
                flogic += "  &&  " + poredi;
}

void Parsing::SelectLista()
{
        if (greska != "")
                return;
        SelectBlok();
        SelectListaOstatak();
}
```

```
// probniDlg.cpp : implementation file

#include "stdafx.h"
#include "probni.h"
#include "probniDlg.h"
#include "KlasaParsing.h"
#include "NewMetod.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMetod

IMPLEMENT_SERIAL( CMetod, CObject, 1)
CMetod::CMetod()
{
}

CMetod::~CMetod()
{
}

void CMetod::Serialize(CArchive &ar)
{
        CObject::Serialize(ar);
        if (ar.IsStoring())
        {
                ar<<m_ImeMetoda<<m_Signatura<<m_ImeKlase<<m_Multi<<m_Rezultat ;
        }
        else
        {
                ar>>m_ImeMetoda>>m_Signatura>>m_ImeKlase>>m_Multi>>m_Rezultat ;
        }
}

// CListaMetoda

IMPLEMENT_SERIAL( CListaMetoda , CObList , 1 )

void CListaMetoda::AddTailMetod( CMetod *pMetod)
{
        AddTail(pMetod);
}

CMetod* CListaMetoda::GetHeadMetod()
{
        return (CMetod *)(GetHead());
}
```

```
CMetod* CListaMetoda::GetNextMetod(POSITION pos)
{
        return (CMetod *)(GetNext(pos));
}

CMetod* CListaMetoda::Trazi(CString arg)
{
        // Sekvencijalna pretraga po imenu klase
}

CMetod* CListaMetoda::TraziMetod(CString arg)
{
        // Sekvencijalna pretraga liste metoda, po imenu metoda
}

// Klasa CKlasa

IMPLEMENT_SERIAL( CKlasa , CObject , 1)

CKlasa::CKlasa()
{
}

CKlasa::~CKlasa()
{
}

void CKlasa::Serialize(CArchive &ar)
{
        // Serijalizacija liste metoda, analogno listi metoda
}

//////////////////////////////////////////////////////////
// Klasa CListaKlasa

IMPLEMENT_SERIAL( CListaKlasa, CObList, 1 )

void CListaKlasa::AddTailKlasa( CKlasa *pKlasa)
{
        AddTail(pKlasa);
}

CKlasa* CListaKlasa::GetHeadKlasa()
{
        return (CKlasa *)(GetHead());
}

CKlasa* CListaKlasa::GetNextKlasa(POSITION pos)
{
        return (CKlasa *)(GetNext(pos));
}
```

```
CKlasa* CListaKlasa::Trazi(CString arg)
{
	// Sekvencijalno trazenje kroz listu klasa, na osnov uimena klase.
}

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	// Automatski generisan "About box".
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CProbniDlg dialog

CProbniDlg::CProbniDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CProbniDlg::IDD, pParent)
{
	// Kreiranje "dialog box"-a , koje automatski obavlja Class Wizard
}

void CProbniDlg::DoDataExchange(CDataExchange* pDX)
{
	// Automatska razmjena podataka, generise je Class Wizard
}

/////////////////////////////////////////////////////////////////////////////
// CProbniDlg message handlers

BOOL CProbniDlg::OnInitDialog()
{
	// Inicijalizacija resursa potrebnih za "dialog box", koju automatski radi Class Wizard
}

void CProbniDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	// Realizacija sistemskih komandi
}

void CProbniDlg::OnPaint()
{
	// Ako "dialog box" ima dugme za minimiziranje, tada se poziva ova funkcija
}

HCURSOR CProbniDlg::OnQueryDragIcon()
{
```

```
        // Sistemski kursor , kada korisnik odvlaci ikonu
        return (HCURSOR) m_hIcon;
}

void CProbniDlg::OnButtonpars()
{       // Parsiranje ulaznog stringa
        Parsing my;

        UpdateData(TRUE);
        m_upit.MakeUpper();
        my.m_upit = m_upit;
        my.PopuniTabeluSimbola();

        // Dodavanje klasa tabeli simbola
        POSITION pos = lstKlasa.GetHeadPosition();
        while (pos != NULL)
        {
                CString leksima ;
                CKlasa *kl =(CKlasa *)lstKlasa.GetNext(pos);
                CEntry *entry = (CEntry *) new CEntry;
                leksima = kl->m_ImeKlase;
                leksima.MakeUpper();
                entry->m_leksima = leksima;
                entry->m_tip = IME_KLASE;
                my.AddEntry(entry);
        }

        // Dodavanje metoda tabeli simbola
        POSITION pos1 = lstMetoda.GetHeadPosition();
        while (pos1 != NULL)
        {
                CMetod *met = (CMetod *)lstMetoda.GetNext(pos1);
                CEntry *entry = (CEntry *) new CEntry;
                entry->m_leksima = met->m_ImeMetoda;
                if (met->m_Signatura == "ATRIBUT")
                        entry->m_tip = IME_ATRIBUTA;
                else
                        entry->m_tip = IME_METODA;
                entry->m_multi = met->m_Multi;
                entry->m_klasa = met->m_ImeKlase;
                entry->m_domen = met->m_Rezultat;
                entry->m_Signatura = met->m_Signatura;
                my.AddEntry(entry);
        }

        // Parsiranje ulaznog stringa
        my.Naredba();
        m_result = my.flogic; // smjestanje rezultata
        UpdateData(FALSE);

// Ako nema greske u Create-naredbi, dodati klasu i njene metode listama
        if (my.m_Create && my.greska == "")
```

```
    {
        POSITION pos2 = my.tabela.GetHeadPosition();
        while (pos2!= NULL)
        {
            CEntry *entry = (CEntry *)my.tabela.GetNext(pos2);
            switch (entry->m_tip)
            {
            case IME_METODA:
            case IME_ATRIBUTA:
                if (lstMetoda.TraziMetod(entry->m_leksima) == NULL)
                {
                    CMetod *met = (CMetod *)new CMetod;
                    met->m_ImeMetoda = entry->m_leksima;
                    met->m_ImeKlase = entry->m_klasa;
                    met->m_Rezultat = entry->m_domen;
                    met->m_Multi = entry->m_multi;
                    met->m_Signatura = entry->m_Signatura;
                    lstMetoda.AddTail(met);
                }
                break;
            case IME_KLASE:
                if (lstKlasa.Trazi(entry->m_leksima) == NULL)
                {
                    CKlasa *kl = (CKlasa *)new CKlasa;
                    kl->m_ImeKlase = entry->m_leksima;
                    kl->m_Nadklase = entry->m_nadklase;
                    lstKlasa.AddTail(kl);
                }
                break ;

            default :
                break;
            } // end switch
        } // end ehile
    }
}

void CProbniDlg::OnOK()
{
    // Dodavanje novog metoda/atributa.
    // Podaci o metodu dobijaju se iz posebnog "dialog box"-a (klasa CNewMetod).
    // header - "NewMetod.h" ; implementacija - "NewMetod.cpp".
}

void CProbniDlg::OnDblclkList1()
{
    // Prikaz metoda i atributa klase.
    // Dvostrukim klikom na elemenat liste klasa, prikazuju se metodi i atributi te klase.
    // Ako je flag m_Ucurenje = TRUE, tada se prikazuju samo metodi klase.
}

void CProbniDlg::OnSnimi()
```

```
{
        //  Snimanje trenutnog stanja sistemskih klasa.
        // Stanje se smjesta u datoteku "Klasa.dat" .
}

void CProbniDlg::OnCancel()
{
        // Kraj rada sa programom.
        // Snimanje trenutnog stanja, na zahtjev korisnika.
        // Oslobadja se memorija koju su drzale lista klasa i lista metoda.

}

void CProbniDlg::OnUcitaj()
{
        // Ucitavanja snimljenog stanja sistemskih klasa.
        // Podaci se ucitavaju iz datoteke "Klasa.dat".
        // Brise se postrojeci sadrzaj liste klasa i liste metoda.
        // Popunjavaju se liste metoda i klasa i prikazuje se sadrzaj liste klasa.
}

void CProbniDlg::OnBrisiMetod()
{
        // Brisanje metoda/atributa klase.
        // Metod se brise iz liste metoda.
}

void CProbniDlg::OnNovaKlasa()
{
        // Kreiranje nove klase
        // Podaci o novoj klasi dobijaju se iz posebnog "dialog box"-a.
        // "Dialog box" za kreiranje nove klase opisan je u "NewMetod.h" i "NewMetod.cpp"
}

void CProbniDlg::OnBrisiKlasu()
{

        // Brisanje klase
        // Klasa se brise iz liste klasa.
        // Brisu se i svi metodi i atributi klase i iz nje izvedenih klasa.
        // Ime klase se uklanja iz svake  od lista nadklasa.
} // end function

void CProbniDlg::OnEncapsulate()
{
        if (m_Ucauri.GetCheck() == 1)
                m_Encapsulation = TRUE;
        else
                m_Encapsulation = FALSE;
}
```

Reference

1. Chang, C. L. , Lee, R. C. T., - "Symbolic Logic and Mechanical Theorem Proving", Academic Press, NY, 1973
2. Cluet, S. , Delobel, C. - "General Framework for the Optimization of Object-Oriented Queries", ACM SIGMOD Conference on Data Management, San Diego, CA, 1992
3. Date, C. J. - "An Introduction to Database Systems", 6th edition, Addison-Wesley, 1995
4. Deux , O. et al. - "The Story of O2", IEEE Transactions On Knowlwdge and Data Engineering, Vol. 2, No. 1, pp. 91-108, 1990
5. Ishikawa, H. , Suzuki, F. , Kozakura, F. - "The Model, Language and Implementation of an Object-Oriented Multimedia Knowledge Base Management System", ACM Transactions on Database Systems, Vol. 18, No.I, 1993
6. Jović, B. - "Geometrijski tipovi podataka u objektno-orijentisanim bazama podataka", Magistarski rad, Matematički fakulet, Beograd, 1995
7. Kiefer, M. , Kim , W. , Sagiv, Y. - "Quering Object-Oriented Databases", ACM SIGMOD Conference on Data Management, San Diego, CA, 1992
8. Kiefer, M. , Lausen , W. , Wu, J. - "Logical Foundations of Object-Oriented and Frame-Based Languages", Technical Report 93/06, Department of Computer Science, SUNY, NY,1993
9. Kiefer, M. , Warren, D. , Chen , W. - "HILogic : A Foundation for Higher-Order Logic Programming", Journal of Logic Programming, 15(3):187-230, 1993
10. Kiefer, M. , Wu, J. - "A First-Order Theory of Types and Polymorphism in Logic Programming", Technical Report 90/23, Department of Computer Science, SUNY, NY, 1993
11. Kim, W. - "An Introduction to Object-Oriented Databases", McGrawHill, 1989
12. Lanzelote, R. , Valduriez, P. , Zait, M. - "Optimization of Object-Oriented Queries Using Cost-Controlled Strategies", ACM SIGMOD Conference on Data Management, San Diego, CA, 1992
13. Pavlović - Lažetić, G. - "Integrisano okruženje za obradu srpskog jezika"
14. Pavlović - Lažetić, G. , Nenadić , G. - "An Object Model for Natural Language Document" , Matematički vesnik, 47(1995):61-84, 1995
15. Ornstein, J. , Hardhvala, S. , Margulies, B. , Sakahara, D. - "Query Processing in ObjectStore Database System", ACM SIGMOD Conference on Data Management, San Diego, CA, 1992
16. Stroustrup, B. - "Programski jezik C++", "Mikro knjiga", Beograd, 1991
17. Ullman, J., Aho, Z., Hopcroft, D. - "Compiler Design - Principles, Techniques and Tools", McGraw-Hill, 1986
18. Vujošević, S., - "Matematička logika", CID Podgorica, 1996