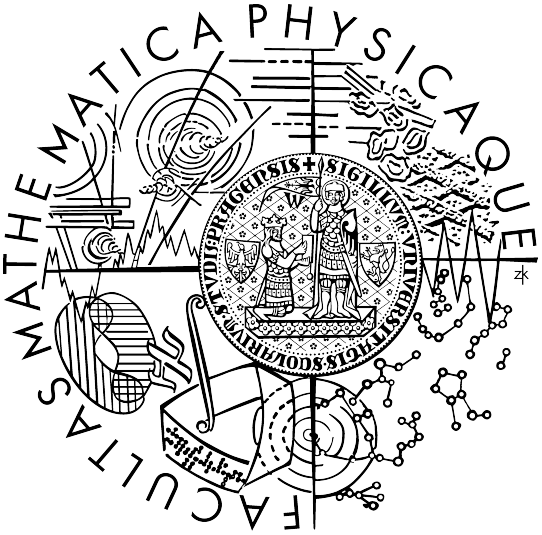
UNIVERZITA KARLOVA V PRAZE
MATEMATICKO-FYZIKÁLNÍ FAKULTA

DIPLOMOVÁ PRÁCE



Martin Janušek

Dynamický Datalog

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Antonín Kosík

Studijní program: INFORMATIKA

Rád by som poďakoval svojmu diplomovému vedúcemu, RNDr. Antonínovi Kosíkovi, za odbornú pomoc a cenné pripomienky pri zostavovaní diplomovej práce.

Prehlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa 8.8 2006 Martin Janušek

Obsah

Názov práce: Dynamický Datalog

Autor: Martin Janušek

Katedra : Katedra softwarového inženýrství

Vedúci diplomovej práce: RNDr. Antonín Kosík

e-mail vedúceho: kosik@ksint.ms.mff.cuni.cz

Abstrakt: V tejto práci skúmame možnosti rozšírenia Datalogu o možnosť aktualizácie. Zaoberáme sa tiež možnosťou optimalizácie dotazovania pomocou zexplicitňovania (materializácie) intezionálnych pravidiel. Na základe nášho výskumu predstavujeme návrh nového jazyka D-Datalogu. Náš návrh rozširuje databázu Datalogu o integritné obmedzenia a množinu intenzionálnych faktov (materializáciu intenzionálnej databáze). Zavádzame možnosť aktualizovať extenzionálnu databázu i množinu intenzionálnych faktov, kedy užívateľ môže explicitne určiť, ktoré intenzionálne dôsledky sa majú nachádzať v materializácií. Aktualizácie sa neprevádzajú ihneď po vyhodnotení. Namiesto toho sú uchovávané do okamihu ukončenia vyhodnotenia dotazu a až potom sú premietnuté do databázy (sémantika aktualizácií s odložením). Tento prístup sa približuje sémantike relačných databáz. Použiteľnosť nášho návrhu demonštrujeme prototypovou implementáciou v jazyku Java.

Kľúčové slová: Datalog aktualizácia materializácia

Title: Dynamic Datalog

Author: Martin Janušek

Department: Department of Software Engineering

Supervisor: RNDr. Antonín Kosík

Supervisor's e-mail : kosik@ksint.ms.mff.cuni.cz

Abstract: In this study we examine possibilities of integration extensional updates into Datalog. Our research involves also possibility of query evaluation optimization by making intensional rules explicit (materialization). As a result od our approach we present new query language D-Datalog. Our proposal extends Datalog database by set of constraints and set of intensional facts (materialization of intensional database). We introduce possibility of extensional and intensional updates. By intensional updates we mean updates of set of intensional facts so user can directly state, which consequences of intensional clauses should be in materialization. Updates are not executed as soon as they are evaluated. Instead, they are collected and then applied to the database when the query evaluation is completed (non-immediate update semantics). This approach is similar to semantics of relational databases. Prototype implementation of D-Datalog in Java is part of our solution.

Keywords: Datalog updates materialization

1 Úvod

Dynamický aspekt databáz je jednou so základných požiadaviek na súčasné databázové systémy. Ak chceme o deduktívnej databáze uvažovať ako o reálne použiteľnej aplikácií, jej formálne základy musia byť rozšírené o dynamické aspekty súčasných databázových systémov, ako je prevádzanie základných aktualizácií extenzionálnej databáze, definícia a údržba materializovaných pohľadov a iné. Existuje niekoľko návrhov jazykov ([1],[2],[3],[4]), ktoré spájajú logické programovanie a schopnosť aktualizácií. Všeobecne sú všetky tieto návrhy založené na zavedení špeciálneho atómu, ktorý definuje aktualizácie v pravidlách, avšak líšia sa v použití rozličných postupov na riešenie aspektov jazyka ako napr. techniky vyhodnocovania, sémantiky, pozície aktualizačného atómu (v hlave alebo v tele pravidla), spôsobu vyhodnocovania konfliktov, či schopnosti modelovať transakcie.

Základná definícia jazyka Datalogu nedefinuje možnosť prevádzať aktualizácie extenzionálnej databázy. Všeobecne je zavedný syntaktický prístup definície aktualizácií pomocou aktualizačných atómov v prefixnom tvare, no takmer v každom návrhu je odlišná sémantika.

Techniky materializácie umožňujú významne zrýchliť vyhodnocovanie dotazov tým, že zexplicitňujú implicitné informácie. Čas potrebný k vyhodnoteniu sa teda stáva primárne závislý na rýchlosti aktualizácie. Vhodný návrh, ktorý kombinuje aktualizácie a techniky materializácie v Datalogu, môže byť základom pre implementáciu reálne použiteľnej aplikácie.

Cieľom práce je skúmať možnosti rozšírenia syntaxe a sémantiky Datalogu tak, aby bolo možné aktualizovať extenzionálnu (prípadne intenzionálnu) časť deduktívnej databázy, a možnosť optimalizácie dotazovania pomocou materializácie (zexplicitňovania) intenzionálnych pravidiel so zameraním na praktickú použiteľnosť návrhu. Súčasťou skúmania je i konkrétny návrh rozšírenia Datalogu o tieto dynamické aspekty a jeho prototypová implementácia.

Kedže existuje viacero alternatívnych definícií Datalogu a každá z nich definuje

sémantiku do určitej úrovne v závislosti na požadovanej výpočtovej sile (s negáciou, s rekurziou), potrebujeme si zvoliť úroveň z ktorej budeme vychádzať pri definícií Datalogu s aktualizáciou. Podrobnejšiu definíciu syntaxe a sémantiky z ktorej budeme vychádzať popisujeme v kapitole 2. V tretej kapitole skúmame možnosti rozšírenia syntaxe a sémantiky Datalogu, hľadáme výhody a nevýhody našich návrhov a prípadné riešenia vzniknutých problémov. V kapitole 4 si definujeme syntax Dynamického Datalogu (D-Datalog) a na základe zvolenej sémantiky z predchádzajúcej kapitoly navrhujeme model algoritmu výpočtu programov s aktualizáciami nad materializovanou databázou v tomto jazyku. V piatej kapitole predstavíme prototypovú implementáciu nášho riešenia a v záverečnej kapitole zhodnotíme dosiahnuté ciele, porovnáme s existujúcimi návrhmi a zhrnieme výsledky a prínosy.

2 Datalog

V tejto časti formálne definujeme Datalog, jeho syntax a sémantiku, ktorú neskoršie rozširujeme o dynamické aspekty. Definujeme tiež sémantiku pre rôzne skupiny databáz (definitné, stratifikovateľné, lokálne stratifikovateľné) z hladiska konštrukcie ich preferovaného modelu.

Logika 1. rádu poskytuje základ pre relačné dotazovacie jazyky. Tento základ vychádza zo zavedenia relačného dátového modelu a nástrojov na prístup k tomuto modelu: relačného kalkulu a relačnej algebry. Relačný kalkul poskytuje model pre deklaratívne dotazovacie jazyky založený na logike, kým relačná algebra poskytuje jeho operačný ekvivalent.

Napriek tomu, že je možné transformovať výrazy relačného kalkulu do relačnej algebry a naopak, vyjadrovacia sila relačného kalkulu (a teda aj relačnej algebry) je obmedzená a nedokáže vyjadriť niektoré dôležité dotazy (napr. tranzitívny uzáver). Táto situácia viedla k vzniku mocnejších (z pohľadu vyjadrovacej sily) dotazovacích jazykov rozširujúcich relačný kalkul ([5]). Jedným z takýchto jazykov je práve deduktívny dotazovací jazyk Datalog a našim cieľom je vytoriť jeho rozšírenie obohatené o dynamické aspekty.

2.1 Syntax

Pri formálnej definícií syntaxe Datalogu vychádzame z [5] a [6]. Logické databázy reprezentujú vedomosti o svete aplikácie popísaním vzťahov medzi objektami, či vlastnosťami objektov. Tieto vzťahy a vlastnosti sú zapísané vo vhodnom logickom jazyku. Syntax jazyka Datalogu vychádza z jazyka 1.rádu.

Definícia 1 - Jazyk 1.rádu

Jazyk 1.rádu Δ je množina symbolov pozostávajúca z nasledujúcich disjunktných množín

Con_Δ je množina konštant jazyka Δ

- Var_Δ je množina premenných jazyka Δ
- Fun_Δ je množina funkčných symbolov jazyka Δ
- $Pred_\Delta$ je množina predikátových symbolov jazyka Δ

kde uvažujeme, že množiny Fun_Δ a $Pred_\Delta$ sú reprezentované nasledovne

$$Fun_\Delta = \bigcup_{i=1}^{\infty} Fun_\Lambda^i \quad a \quad Pred_\Delta = \bigcup_{i=1}^{\infty} Pred_\Lambda^i ,$$

Fun_Λ^i (resp. $Pred_\Lambda^i$) sú disjunktné množiny a prirodzené číslo *i* určuje počet argumentov – *aritu* zodpovedajúceho funkčného či predikátového symbolu. Jazyk 1. rádu obsahuje tiež logické spojky ($\neg, \wedge, \vee, \rightarrow$), kvantifikátory ($\forall, \exists$) a pomocné symboly (zátvorky a pod.).

Zo symbolov jazyka sa podľa istých pravidiel tvoria 2 typy výrazov,

- *termy* – popisujú objekty sveta aplikácie
- *formule* – vyjadrujú rôzne tvrdenia o objektoch

Definícia 2 - Množina termov

Nech Δ je jazyk 1.rádu, potom *množina termov jazyka Δ* je induktívne definovaná nasledovne,

- Akákoľvek konštanta $c \in Con_\Lambda$ je term
- Akákoľvek premenná $v \in Var_\Lambda$ je term
- Ak $f \in Fun_\Lambda^n$ je funkčný symbol arity *n* a $t_1, \ldots, t_n$ sú termy, potom $f(t_1, \ldots, t_n)$ je term.

Množina premenných, ktoré sa objavujú v terme *t* sa označuje *var(t)*. Term *t* sa označuje *ground*, práve vtedy ak *var(t) = Ø*.

Na vyjadrenie tvrdenia o nejakom objekte aplikačného sveta sa v logike používajú *formule*.

Definícia 3 - Formule 1.rádu

Formule 1.rádu alebo skrátene *formule* jazyka 1. rádu Δ sú induktívne definované nasledovne

- Ak $f \in Pred_\Lambda^n$ je predikátový symbol arity *n* a $t_1, \ldots, t_n$ sú termy, potom $P(t_1, \ldots, t_n)$ je formula, tiež nazývaná *atomická formula* alebo *atóm*.
- Ak φ, ψ sú formule, potom sú formule aj $\neg\varphi, (\psi \wedge \varphi), (\psi \vee \varphi), (\psi \rightarrow \varphi)$
- Ak φ je formula a *v* premenná, potom $\forall v \varphi$ *a* $\exists v \varphi$ sú formule

Jazyk Datalogu má určité obmedzenia oproti jazyku 1.rádu, vzhľadom na požiadavok konečnosti výpočtu.

Definícia 4 - Jazyk Datalogu

Jazyk Datalogu Δ je jazyk 1. rádu bez funkčných symbolov.

Dôsledky definície jazyka Datalogu sú

- Akýkoľvek term v jazyku Datalogu je buď premenná alebo konštanta.
- Každý ground term v takom jazyku je konštanta.

Pre jazyk Datalogu uvažujeme rozdelenie množiny predikátových symbolov $Pred_\Delta$ na dve disjunktné množiny $Pred_\Lambda = \{Pred^I_\Lambda \cup Pred^E_\Lambda\}$, kde $Pred^I_\Lambda$ je množina intenzionálnych a $Pred^E_\Lambda$ je množina extenzionálnych predikátových symbolov.

Definícia 5 - Literály,cieľa klauzule

Literál je buď atóm (A) alebo jeho negácia ($\neg A$). Literály, ktoré sú atómy, sa nazývajú *pozitívne* a literály, ktoré sú negácie atómov, sa nazývajú *negatívne*. *Cieľ* je formula v tvare $L_1, \ldots, L_n$, kde $L_i, 0 \le i \le n$ je literál. (*Normálna*) *klauzula* je akákoľvek formula v tvare $L_1 \wedge \ldots \wedge L_n \rightarrow A$, kde $L_i, 0 \le i \le n$ je literál a A je extenzionálny alebo intenzionálny atóm. Atóm A sa nazýva *hlavička* a konjunkcia $L_1 \wedge \ldots \wedge L_n$ sa nazýva *telo* klauzule. Klauzula s prázdnym telom ($n=0$) sa nazýva *fakt*. Klauzula, ktorá neobsahuje negatívne literály sa nazýva *definitná*.

Konvencia

Podľa zaužívanej typografickej konvencie budeme v ďalšom texte pri znázorňovaní klauzúl používať namiesto formy $L_1 \wedge \ldots \wedge L_n \rightarrow A$ jej identický zápis $A :- L_1, \ldots, L_n$. Pre fakty budeme vynechávať implikačný symbol a budeme používať zápis A.

Definícia 6 - Databáza Datalogu

(Definitná) Databáza Datalogu je akákoľvek konečná množina (definitných) klauzulí.

Databáza Datalogu sa skladá z 2 častí, z intenzionálnej (IDB), ktorá obsahuje klauzule s neprázdnym telom tzv. *pravidlá* a extenzionálnej (EDB), ktorá obsahuje fakty. Predikátové symboly v hlavičkách klauzúl z IDB (resp. EDB) tvoria $Pred^{I}_{\Lambda}$ (resp. $Pred^{E}_{\Lambda}$).

Ako z vyššie uvedeného vyplýva, pre databázu Datalogu platí $DB = \{EDB \cup IDB\}$ a $EDB \cap IDB = \emptyset$.

Definícia 7 - Definícia predikátu

Hovoríme, že klauzula $P(t_1, \ldots, t_m) :- L_1, \ldots, L_n$ *definuje predikát P. Definícia predikátu P* v databáze Datalogu *D* je množina všetkých klauzulí *D*, ktoré definujú *P*.

Definícia 8 - Program Datalogu

Program Datalogu P je akákoľvek Databáza Datalogu t.ž. pre každú klauzulu $C \in P$ je predikátový symbol v hlave *C* intenzionálny. Program D-Datalogu sa nazýva *pozitívny*, ak všetky klauzuly v *P* sú definitné.

Okrem obmedzení na jazyk definujme obmedzenie na klauzuly, ktoré môže byť formulované použitím notácie *bezpečných premenných klauzuly* a *obmedzenia oboru hodnôt*.

Definícia 9 - Množina bezpečných premenných

Nech $C = (A :- L_1, \ldots, L_n)$ je normálna klauzula. Množina *bezpečných premenných C* je definovaná nasledovne:

- Ak L_i je extenzionálny alebo intenzionálny literál, potom každá premenná v L_i je bezpečná
- Ak L_i je rovnica $s \simeq t$ a všetky premenné v *s* sú bezpečné, tak potom každá premenná v *t* je bezpečná
- Ak L_i je rovnica $s \simeq t$ a všetky premenné v *t* sú bezpečné, tak potom každá premenná v *s* je bezpečná

Definícia 10 - Obmedzený obor hodnôt

Klauzula *C* má *obmedzený obor hodnôt,* ak každá jej premenná je bezpečná.

Evidentne, ak *C* neobsahuje žiadny vstavaný predikát, potom má *C* obmedzený obor hodnôt práve vtedy a len vtedy, ak sa každá premenná v jej tele vyskytuje v pozitívnom literále.

V Datalogu sú parametre výrokov (formúl) definované pomocou premenných a objekty sú reprezentované pomocou termov. Potrebujeme teda mechanizmus na substituovanie premenných za termy.

Definícia 11 - Substitúcie

Nech Λ je Jazyk Datalogu potom zobrazenie z množiny premenných do množiny termov $\theta: v \to t, v \in Var\lambda, t \in Var_\lambda \cup Con_\lambda$ nazývame *substitúcia*. Existuje zavedené značenie $\{t_1/v_1, \ldots, t_n/v_n\}$ pre substitúciu θ definovanú nasledovne

$$\theta(u) = \begin{array}{ll} \cdots & t_i \, ak \, u = v_i \\ \ddots & u \, ak \, u \notin \{v_1, \ldots, v_n\} \end{array}$$

Aplikácia substitúcie na term t sa značí $t\theta$. Vzťah medzi substitučným a substituovateľným termom je definovaný pojmom inštancia termu.

Definícia 12 - Inštancie termu

Term t' je *inštancia* termu t, označujeme $t \leq t'$, ak existuje substitúcia θ t.ž. $t\theta = t'$. Term t' je *ground inštancia* termu t, ak t' je inštancia termu t a t' je ground.

So zavedenou substitúciou termu môžeme aplikovať substitúciu na formule, kedy substituujeme všetky ich parametre (premenné).

Definícia 13 - Aplikácia substitúcie

Aplikáciou substitúcie θ na formulu φ dostaneme novú formulu označovanú $\varphi\theta$ definovanú induktívne

- Ak je φ atomická formula $P(t_1, \ldots, t_n)$ potom $\varphi\theta = P(t_1\theta, \ldots, t_n\theta)$
- Ak má φ formu $\psi \, o \, \varphi, o \in \{\wedge, \vee, \to\}$ potom $\varphi\theta = \psi\theta \, o \, \varphi\theta$, kde $o \in \{\wedge, \vee, \to\}$
- Ak je φ má formu $\neg\psi$ potom $\varphi\theta = \neg(\psi\theta)$

Podobne ako pre termy, zavádzame i pre formule pojem inštancia.

Definícia 14 - Inštancie formule

Formula φ' je *inštancia* formule φ, označujeme $\varphi \leq \varphi'$, ak existuje substitúcia θ t.ž. $\varphi\theta = \varphi'$. Formula φ' je *ground inštancia* formule bez kvantifikátorov φ, ak φ' je inštancia formule φ a φ' je ground.

2.2 Sémantika

Deduktívne databázy sú založené na formalizme logiky 1.rádu. Podľa sémantiky logiky 1.rádu existujú dva hlavné smery definície sémantiky deduktívnych databáz:

- teória 1.rádu (dôkazne-teoretický prístup)
- model teórie 1. rádu (modelovo-teoretický prístup)

Dôkazne-teoretický prístup asociuje s databázou teóriu 1. rádu T. Dotaz je považovaný za pravdivý v databáze, ak je logickým dôsledkom T. Množina klauzúl databázy je množina formúl 1.rádu, a preto môže byť sama považovaná za teóriu 1.rádu. Avšak teória T asociovaná s databázou v dôkazne-teoretickom prístupe nie je samotná databáza. Obyčajne obsahuje všetky klauzuly databázy a dodatočné formuly, vyjadrujúce princípy na ktorých je založená sémantika deduktívnych databáz.

Modelovo-teoretický prístup asociuje s databázou triedu interpretácií (preferované modely databázy). Podľa tohto prístupu je cieľ logickým dôsledkom databáze práve vtedy a je pravdivý vo všetkých preferovaných modeloch databáze. Modelovo-teoretický prístup je viac flexibilnejší ako dôkazne-teoretický, hlavne pri zavedení negácie. ([5],[7])

Existujú tri hlavné predpoklady sémantiky deduktívnych databáz ([5])

- predpoklad unikátneho mena – každý objekt musí mať unikátne pomenovanie
- predpoklad uzavretej domény – každý objekt danej interpretácie musí mať pomenovanie
- predpoklad uzavretého sveta – negatívna informácia je dôsledkom databáze, ak pozitívna informácia nie je dôsledkom databáze

Definícia 15 - Interpretácia jazyku

Nech *L* je jazyk Datalogu, potom *interpretácia L* pozostáva z nasledujúceho:

- Neprázdna množina prvkov U nazývaná *doména intepretácie*
- Pre každú konštantu v L existuje priradenie prvku z U
- Pre každú n-árnu funkciu v L existuje priradenie mapovania $U^n \to U$. (Z definície jazyka Datalogu však žiadne funkcie neobsahuje)
- Pre každý n-árny predikát q v L existuje priradenie mapovania $U^n \to \{True, False\}$

Pre definitné databázy postačí uvažovať o Herbrandových interpretáciach. Herbrandová interpretácia (HI) je interpretácia, ktorej doména sú všetky ground termy a konštanty sú reprezentované sami sebou.

Definícia 16 - Herbrandov model

Nech L je jazyk Datalogu, DB je množina klauzúl a I je interpretácia L. Ak je I Herbrandová interpretácia a zároveň je I model DB, potom sa I nazýva *Herbrandov model DB*.

Herbrandové univerzum pre L , označované U_L, je definované ako množina všetkých termov (v jazyku obsahujúcom funkčné symboly, obsahuje i rekurzívne konštruované termy, kedy použijeme argumenty funkcií ako konštanty v L alebo prvky v U_L).

Herbrandova báza B_L je definovaná ako množina atómov, ktoré môžu byť vytvorené priradením prvkov U_L do argumentov predikátov. Ak je dané priradenie konštánt, potom Herbrandova interpretácia je definovaná priradením relácie Q arity n každému n-árnemu predikátu q, kde $q(a_1, \ldots, a_n)$ je pravdivé ak $(a_1, \ldots, a_n) \in Q$, $a_1, \ldots, a_n \in U_L$. Alternatívne, Herbrandova interpretácia L je podmnožinou Herbrandovej bázy L.

Herbrandovo univerzum U_P a Herbrandova báza B_P sú pre program P , definované ako U_L a B_L jazyka L, ktorý obsahuje konštanty, funkcie a predikáty, ktoré sa objavujú v P.

V prípade, že by sme povolili v Jazyku Datalogu funkčné symboly bez obmedzenia, nemohli by sme zaručiť konečnosť výpočtu. Na jednoduchom príklade si ukážeme program P v jazyku s funkčnými symbolmi, pre ktorý dostaneme nekonečný Herbrandov priestor a počet Herbrandových interpretácií. Následne dostaneme počet strát po stratifikácií rovný prvému ordinálu ω, t.j. počet krokov na zostrojenie minimálneho modelu P je nekonečný.

Príklad 1 - „Nekonečný“ program

Program P:

$$p(f(x)) \quad :- \quad q(x)$$
$$q(a) \quad :- \quad p(x)$$

Herbrandovo univerzum U_P

$$U_P = \{a, f(a), f(f(a)), \ldots, f^n(a), \ldots\}$$

Herbrandova báza B_P

$$B_P = \{p(f^n(a)) \mid n \geq 0\} \cup \{q(f^m(a)) \mid m \geq 0\}$$

Nech *r* je pravidlo programu P. Potom *ground(r)* označuje množinu ground (t.j. počet premenných v každom terme je 0) inštancií *r* (napr. všetky pravidlá získané dosadením hodnôt z Herbrandového univerza U_P do premenných v *r*).

Príklad 2 - Model databáze

Nech DB je databáza v jazyku L. Potom Herbrandová interpretácia J obsahujúca všetky ground atómy L (Herbrandová báza) je model DB.

Tento model nám nevyjadruje taký význam databázy, aký by sme očakávali (platí v ňom všetko). Podľa predpokladu uzavretého sveta, ak nie je fakt explicitným dôsledkom databázy, potom je dôsledkom databázy jeho negácia. Tento predpoklad môžeme preformulovať nasledovne: zo všetkých možných významov databázy musíme vybrať také, kde je minimalizovaná množina pozitívnych dôsledkov. Pre Herbrandové interpretácie to znamená voliť minimálne Herbrandové modely ako preferované modely databázy.

Definícia 17 - Minimálny a najmenší Herbrandov model

Herbrandov model J množiny formúl D sa nazýva *minimálny Herbrandov model D,* ak platí, že $\forall J', J' \text{ je Herbrandov model } D : J' \not\subset J$

Herbrandov model J množiny formúl D sa nazýva *najmenší Herbrandov model D,* ak platí, že $\forall J', J' \text{ je Herbrandov model } D : J \subseteq J'$

Pre každú množinu definitných klauzúl platí, že má najmenší Herbrandov model.([5])

2.2.1 Definitné databázy

Interpretácia klauzúl ako deduktívnych pravidiel je základom pre sémantiku deduktívnych databáz. Táto sémantika sa nazýva sémantika pevného bodu.

Definícia 18 - Pevný bod

Nech S je množina a $f: S \to S$. Potom $a \in S$ je pevný bod f, ak $f(a) = a$.

V sémantike pevného bodu sa používajú deduktívne pravidlá zodpovedajúce klauzulám na získanie bezprostredného dôsledku.

Definícia 19 - Operátor bezprostredného dôsledku

Nech D je databáza a I je interpretácia P. Potom je *operátor bezprostredného dôsledku* zobrazenie $T_D: I \to J$, kde P je program s aktualizáciou a I, J sú interpretácie.

$$T_D(I) = \{ A \in B_D \mid \exists r: A :- A_1, \ldots, A_n \in ground(D), \\ \forall i \in \{1, \ldots, n\} \text{ ak } A_i \text{ je pozitívny tak } A_i \in I, \\ \text{ak } A_i \text{ je negatívny tak } A_i \notin I \}$$

Pre pevný bod operátora bezprostredného dôsledku T_D platí nasledujúce([5],[6]):

- T_D je Herbrandov model D
- Ak je D definitná, potom T_D je najmenší Herbrandov model

2.2.2 Stratifikovateľné databázy

Vzhľadom na to, že pri použití negácie môže nastať situácia, kedy existuje niekoľko minimálnych modelov a žiaden pevný bod, existuje pre normálne databázy sémantika perfektného modelu, ktorý je preferovaný model pre očakávaný význam databáze.

Pre databázy umožňujúce rozdelenie predikátov do úrovní (strata) existuje spôsob určovania perfektného modelu.

Definícia 20 - Mapovanie úrovní

Mapovanie úrovní pre jazyk Datalogu Λ je akákoľvek funkcia f z množiny predikátov $Pred_\Lambda$ do množiny pozitívnych celých čísel. Pre predikátový symbol $P \in Pred_\Lambda$ je číslo $f(P)$ nazývané *úroveň predikátu P pod f*

Definícia 21 - Statifikovaná databáza

Nech D_Λ je databáza Datalogu. Ak existuje mapovanie úrovní f nad D_Λ t.ž. pre všetky $C=(A:-L_1,...,L_n)\in D_\Lambda$ platí

- Pre všetky $L_i, 1\le i\le n$, L_i *je pozitívne* je $f(A)\ge f(L_i)$
- Pre všetky $L_j, 1\le j\le n$, L_j *je negatívne* je $f(A)>f(L_j)$

potom sa D_Λ nazýva *stratifikovaná* a f je *stratifikácia* D_Λ

Perfektný model stratifikovateľných databáz sa konštruuje hierarchicky pomocou stratifikácie databázy. Nech D je stratifikovateľná databáza a $D_1\cup...\cup D_n$ je jej stratifikácia. Uvažujme interpretácie $J_1,...,J_n$ definované pre $\forall J_k, 1\le k\le n$ nasledovne:

- J_k sú v rovnakom jazyku
- J_k obsahuje všetky predikáty z nižších úrovní ako vstavané (t.j. majú pevne definované zobrazenie v Herbrandovej interpretácií)

  Konštrukcia indukciou:
  - Kedže D_1 je definitná , tak má najmenší Herbrandov model. Nech je tento model J_1
  - Máme zkonštruovaný model J_k a chceme model J_{k+1}. Považujme všetky predikáty (a ich negácie) v nižších úrovniach $(<k+1)$ za vstavané. Potom je D_{k+1} definitná a J_{k+1} je jej najmenší model

Pre $J=J_1\cup...\cup J_n$ platí nasledujúce ([5],[6]):

- J je minimálny Herbrandov model D
- J je perfektný Herbrandov model D, ktorý sa zhoduje s najmenším Herbrandovým modelom
- J je pevný bod T_D

2.2.3 Lokálne stratifikovateľné databázy

Pre obecnejší typ databáz určujeme perfektný model na základe závislostného usporiadania ground atómov.

Definícia 22 - Graf závislosti

Graf závislosti normálnej databázy D v jazyku L je orientovaný graf, ktorého množina vrcholov obsahuje vrchol pre každý atóm patriaci do Herbrandovej bázy (B_L) a hrany sú definované nasledovne:

- $A \rightarrow B$ ak existuje klauzula $C \in ground(D)$, taká že A sa vyskytuje v tele C a B je hlavička C
- $A \stackrel{\neg}{\rightarrow} B$ ak sa A vyskytuje negatívne v tele C

Atóm A je preferovaný pred B ($A \ll B$), ak existuje cesta z A do B, ktorá obsahuje aspoň jednu negatívnu hranu. Definíciu rozšírime na Herbrandové interpretácie J, K nasledovne: $J \neq K$, $\forall A \in J \setminus K$, $\exists B \in K \setminus J : B \ll A$ potom $J \ll_H K$

Databázy, ktoré môžeme dobre usporiadať na základe ich grafu závislosti nazývame lokálne stratifikované. Formálne ich definujeme nasledovne.

Definícia 23 - Lokálne stratifikovaná databáza

Databáza D v jazyku L je *lokálne stratifikovaná* práve vtedy, ak je relácia $\ll$ na B_L dobre usporiadaná (t.j, ak každá neprázdna podmnožina B_L má najmenší prvok vzhľadom k usporiadaniu B_L; najmenší prvok je prvok $\{p \mid p, r \in B_L \wedge \forall r(p \ll r \vee r = p)\}$.

Alternatívna definícia lokálnej stratifikovateľnosti vychádza z ohodnotenia atómov v ground databáze. Pre lokálne stratifikovateľné databázy platí nasledovné ([5]):

- Databáza D je lokálne stratifikovaná práve vtedy a len vtedy ak existuje mapovanie $l : B_P \rightarrow \mathbb{N}$ t.ž. pre každú klauzulu $C = (A :- L_1, ..., L_n) \in ground(D)$ platí
  - $l(A) \geq l(L_i), \quad 1 \leq i \leq n$
  - $l(A) > l(L_i), L_i \, je \, negatívne, \quad 1 \leq i \leq n$
- Herbrandov model je perfektný ak je minimálny v zmysle $\ll_H$ usporiadania
- Každá lokálne stratifikovatelná datbáza má unikátny perfektný model, ktorý je minimálny a zároveň je aj pevný bod T_D

3 Syntax a sémantika aktualizácie

Definícia Datalogu nešpecifikuje princíp, akým by sa mali prevádzať aktualizácie. Našou snahou je skúmať vlastnosti aktualizácií Datalogu z „praktickej“ stránky jazyka (t.j. pribížiť ho požiadavkám na súčasné databázové modely) a pri zachovaní všeobecnej funkcionality čo najmenej obmedziť jeho výrokovú silu. Zároveň analyzujeme nedostatky našich návrhov a ukazujeme prípadné riešenia, či smery, ktoré by mohli viest k riešeniu vzniknutých problémov.

Pri zavádzaní aktualizácií do Datalogu je potrebné definovať syntax a sémantiku vzhľadom na rôzne aspekty jazyka. V tejto časti predstavíme základnú syntax a sémantiku aktualizácií a definujeme sémantiku možností aktualizácie pri konflikte dát (silná vs. slabá sémantika). Ďalším aspektom, ktorý skúmame, je forma sémantiky týkajúca sa momentu fyzickej aktualizácie. Jedna atomizuje výpočet z hľadiska aktualizácie a tým sa približuje sémantike relačných databázových systémov (sémantika s odložením), kým druhá dynamicky aktualizujú databázu už počas behu výpočtu (okamžitá sémantika).

3.1 Základná syntax aktualizácií

Pri rozširovaní syntaxe Datalogu o aktualizácie vychádzame z intuitívneho a v literatúre všeobecne zavedeného značenia ([1],[2],[3],[4],[8]).

Definícia 24 - Jazyk Datalogu s aktualizáciami

Jazyk Datalogu s aktualizáciami Δ je jazyk Datalogu rozšírený o množinu *aktualizačných* symbolov $\alpha = \{+, -\}$, ktoré sa používajú v prefixnom tvare spolu s predikátovými symbolmi.

Definícia 25 - Aktualizačný atóm

Nech je $p(A_1, \ldots, A_n)$ relácia databázy v jazyku Datalogu s aktualizácio a $\gamma = \{+, -\}$ je množina aktualizačných symbolov. Potom atóm $\gamma\, p(t_1, \ldots, t_n)$ je *aktualizačný atóm*.

Aktualizačný atóm sa môže vyskytovať buď v tele alebo v hlavičke klauzúl. Prvý prípad používame v intenzionálnych pravidlách databázy, kde je aktualizácia súčasťou výpočtu, druhá forma pravidiel sa používa v tzv. integritných obmedzeniach, ktoré kontrolujú konzistentnosť databázy po výpočte.

3.2 Základná sémantika aktualizácií

Aktualizácia extenzionálnej databázy je definovaná intuitívne ako rozšírenie či zúženie množiny extenzionálnych atómov, nad ktorými sa prevádzajú dotazy. Ak je *p* základná relácia databázy Datalogu, potom aktualizačný atóm pridá resp. odstráni fakt $p(t_1, ..., t_n)$ z extenzionálnej databázy.

Existujú snahy o zovšeobecnenie aktualizácií i na intenzionálne relácie ([2]), čo však prináša nové problémy, napr. prevedenie aktualizácie intenzionálneho pravidla na aktualizáciu (či dokonca viac aktualizácií) extenzionálnych faktov, kedy môže nastať situácia, že existuje viac ako jedna možnost takéhoto prevedenia pre danú aktualizáciu, a teda aktualizácia nie je deterministická.

Príklad 3 - Konflikt pri aktualizácií intenzionálnej relácie

$$
\begin{aligned}
&EDB: \\
&\quad spoj(a, b). \\
&IDB: \\
&\quad cesta(X, Y) :- spoj(X, Y). \\
&\quad cesta(X, Y) :- spoj(X, Z), cesta(Z, Y). \\
&Aktualizácia\ intenzionálnej\ relácie: \\
&\quad +cesta(a, c)
\end{aligned}
$$

Daná aktualizácia intenzionálnej relácie sa môže previesť buď na extenzionálny fakt $spoj(a, c)$ alebo na $spoj(b, c)$.

3.3 Silná a slabá sémantika

Podľa podmienok za akých je možné danú aktualizáciu prevádzať je možné rozdeliť aktualizačnú sémantiku na silnú a slabú sémantiku aktualizácií.

Silná sémantika (SiS) je sémantika obsahujúca silné aktualizácie t.j. aktualizácie, ktoré umožnia vloženie resp. odstránenie atómu z databázy iba ak daný atóm neexistuje resp. existuje v databáze. Naproti tomu slabá sémantika (SlaS) obsahuje slabé aktualizácie t.j. na ich prevedenie nie je nutné overovať žiadne podmienky.

Príklad 4 - Slabá a silná aktualizácia

EDB	*Slabá sémantika*	*Silná sémantika*
$p(a), p(b)$	$+p(a), -q(b)$	~~$+p(a), -q(b)$~~
$q(a)$	$+p(c), -p(b)$	$+p(c), -p(b)$

Definícia 26 - Štruktúra pre silnú sémantiku aktualizácií

Štruktúra $\Re^s$ pre silné aktualizácie sa skladá z

- EDB
- priradenie pre každý aktualizačný atóm $\pm p(\bar{t})$, kde $\bar{t}$ je ground term t.ž.

$$+p(\bar{t}) = True \Leftrightarrow p(\bar{t}) \notin EDB$$
$$-p(\bar{t}) = True \Leftrightarrow p(\bar{t}) \in EDB$$

Definícia 27 - Štruktúra pre slabú sémantiku aktualizácií

Štruktúra $\Re^w$ pre slabé aktualizácie[1] sa skladá z priradenia pre každý aktualizačný atóm $\oplus p(\bar{t}), \ominus p(\bar{t})$, kde $\bar{t}$ je ground term t.ž.

$$\delta p(\bar{t}) = \begin{array}{ll} \cdots & False \quad ak \ \exists(\gamma p(\bar{t}) \rightarrow True) \in \Re^w : \gamma \neq \delta \\ \ddots & True \quad inak \end{array},$$

kde $\delta, \gamma \in \{\oplus, \ominus\}$.

Zobrazenie θ z premenných do termov pre aktualizačné atómy nazývame $\Re$ - ohodnotenie.

Definícia 28

Aktualizačné atómy U_1, U_2 sú riešiteľné práve vtedy, ak existuje ohodnotenie θ t.ž. $\Re \models (U_1, U_2)\theta$. Potom sa θ nazýva $\Re$ - riešenie U_1, U_2.

Vyššie uvedená definícia $\Re$ - riešenia platí pre štruktúru slabej i silnej aktualizácie. Pre silné aktualizácie kontroluje, či nie sú 2 aktualizácie komplementárne (aktualizácie sú komplementárne; aktualizácie $+p(X), -p(Y)$ sa môžu stať komplementárne pre inštancie a/X, a/Y), a či môže byť vkladaná či odstraňovaná informácia aktualizovaná podľa silnej sémantiky. Pre slabé aktualizácie kontroluje iba, či nie sú 2 aktualizácie komplementárne.

1 Na zvýraznenie rozdielu medzi silnou a slabou sémantikou budeme používať pre silnú znaky $+, -$ a pre slabú $\oplus, \ominus$

Všeobecne sa viac používa silná sémantika, pretože napriek určitému oslabeniu vyjadrovacej sily jazyka z pohľadu prevádzania aktualizácií, udržuje konzistenciu databázy a predchádza vzniku problémov spojených s komplementárnymi aktualizáciami.

Príklad 5 - Konflikt pri použití slabej aktualizácie

Uvažujme administratívnu databázu univerzity. Táto databáza obsahuje informácie o študentoch a skúškach, ktoré zložili. Je prepojená s knižnicou, kde eviduje knihy a ich pôžičky, a zároveň posiela požiadavky študentom na vrátenie požičaných kníh:

- EDB:

$$\begin{array}{ll} student(frantisek). & skuska(fyzika). \\ student(maria). & skuska(anglictina). \\ kniha(othello, anglictina). & pozicka(quanta, frantisek). \\ kniha(principia, fyzika). & pozicka(principia, frantisek). \\ kniha(quanta, fyzika). & \end{array}$$

Intenzionálna časť databázy predstavuje pravidlá na aktualizáciu extenzionálnej databázy. V prípade, že študent zloží skúšku, pridá sa fakt *vykonana(S,E)* (P1), ak opustí školu, odstráni sa záznam o ňom z databázy (P2). Knižnica odmietne požičať knihu študentovi, ktorý ešte nevrátil inú požičanú knihu a na ktorej vrátenie existuje požiadavka (P3), alebo ak je požadovaná kniha požičaná (P4). Po vrátení knihy sa záznam o pôžičke odstráni (P5). Výnimočne môže knižnica predĺžiť pôžičku pomocou P6.

- IDB:

$$\begin{array}{lll} zlozi(S,E) & :- & student(S), skuska(E), +vykonana(S,E). \quad [P1] \\ skonci(S,E) & :- & student(S), -student(S). \quad [P2] \\ zamietnipozicku(B,S) & :- & poziadavka(X,S), kniha(X,E). \quad [P3] \\ zamietnipozicku(B,S) & :- & pozicka(B,X), student(S). \quad [P4] \\ vratil(B,S) & :- & pozicka(B,S), -pozicka(B,S). \quad [P5] \\ predlz(B) & :- & pozicka(B,S), -poziadavka(B,S). \quad [P6] \end{array}$$

Integritné obmedzenia (IO) zaručujú konzistentný stav databázy po aktualizácii. Ak študent odchádza, je potrebné odstrániť všetky informácie o jeho vykonaných skúškach (IO 1), a ak má požičanú nejakú knihu, tak mu poslať požiadavku na jej vrátenie (IO 2). V prípade, že je kniha vrátená a existuje požiadavka na jej vrátenie, je potrebné túto požiadavku odstrániť z DB (IO 3). Špeciálne pravidlo knižnice zároveň určuje, že ak má študent po vykonaní skúšky požičanú nejakú knihu, musí ju vrátiť a preto sa vytvorí požiadavka na jej vrátenie (IO 4).

- IO:

$$-student(S), vykonana(S,E) \quad -: -vykonana(S,E). \quad [IO1]$$
$$-student(S), pozicka(B,S) \quad -: +poziadavka(B,S). \quad [IO2]$$
$$-pozicka(B,S), poziadavka(B,S) \quad -: -poziadavka(B,S). \quad [IO3]$$
$$+vykonana(S,E), pozicka(B,S), kniha(B,E) \quad -: +poziadavka(B,S). \quad [IO4]$$

V prípade, že František zloží skúšku z Fyziky, povolí mu knižnica prostredníctvom jeho profesora predĺženie pôžičky na knihu Quanta. V databázi to môžeme zaznamenať pomocou vyhodnotenia transakcie

$$zlozi(Frantisek, Fyzika), predlz(Quanta).$$

Pravidlo P1 spôsobí vloženie *vykonana(Frantisek,Fyzika)* do databázy a zároveň spustí IO4, ktorého podmienky spĺňajú *pozicka(Quanta, Frantisek)*, *kniha(Quanta, Fyzika)* a *pozicka(Principia, Frantisek)*, *kniha(Principia, Fyzika)*, takže by sa malo taktiež previesť vloženie *poziadavka(Principia,Frantisek)* a *poziadavka(Quanta,Frantisek)*. Avšak pravidlo P6, na základe *predlz(Quanta)* v druhej časti transakcie, by malo požiadať o odstránenie faktu *poziadavka(Quanta,Frantisek),* z čoho vzniká konflikt.

3.3.1 Okamžitá a odložená sémantika

Vzhľadom na princíp deduktívnych databázových systémov, kde existujú dva základné typy relácií (extenzionálne – relácie sú uložené priamo v databáze; intenzionálne – relácie sa odvodzujú za behu programu a v databáze pre ne existuje iba odvodzovacie pravidlo), môžeme pozerať na výpočet dotazu z pohľadu aktualizácie extenzionálnej časti, buď ako na postupnosť atomických operácií alebo môžeme považovať celý výpočet za jednu operáciu.

3.3.2 Okamžitá sémantika

Existujúce prístupy na zavedenie aktualizácií do deklaratívnych dotazovacích jazykov zavádzajú určitú formu kontroly. Táto forma kontroly je úzko spojená z klasickou sémantikou aktualizácií tzv. sémantikou okamžitej aktualizácie (SOKA). Pri použití SOKA sa výpočet považuje za postupnosť atomických operácií a databáza sa aktualizuje ihneď po vyhodnotení aktualizačného atómu.

Pre vytvorenie sémantiky okamžitej aktualizácie (SOKA) vychádzame zo sémantiky konštrukcie pevného bodu. Zavedieme, podobne ako sémantika konštrukcie pevného bodu, operátor bezprostredného dôsledku (OBED) pre aktualizácie, ktorý však na rozdiel od pôvodného, berie ohľad aj na eventuálnu zmenu extenzionálnej databázy.

Definícia 29 - OBED pre program s aktualizáciou v okamžitej sémantike

Nech P je program obsahujúci aktualizácie a I je interpretácia P. Potom je *operátor bezprostredného dôsledku pre program s aktualizáciou v okamžitej sémantike* T_1 zobrazenie $T_1 : I \to J$, kde P je program s aktualizáciou a I, J sú interpretácie.

$$T_1(I) = \{ A \in B_P \mid \exists r : A :- C_1, \ldots, C_m, A_1, \ldots, A_n \in ground(P),$$
$$\forall i \in \{1, \ldots, n\}\ ak\ A_i\ je\ pozitívny\ tak\ A_i \in I,$$
$$ak\ A_i\ je\ negatívny\ tak\ A_i \notin I,$$
$$\forall C_j, C_k, \Re \models (C_j, C_k)\theta, 1 \le j, k \le m \}$$

V prípade, že sa po aplikácií T_1 nezmení extenzionálna databáza, funkčnosť je rovnaká ako pri všeobecnom operátore bezprostredného dôsledku. Rozdiel nastáva v prípade, že sa vyhodnotí pravidlo, ktoré obsahuje aktualizačný atóm. Potom je doména výslednej interpretácie rozšírená (resp. zúžená) o pridané (odstránené) ground termy.

Na ukázanie, že vyššie uvedený operátor je sémanticky korektný potrebujeme ukázať, že jeden krok (t.j. jedna jeho aplikácia) nenaruší sémantiku programu. Predpokladajme, že pôvodný program P je lokálne stratifikovaný. Každý lokálne stratifikovaný program má unikátny preferovaný model, ktorý je zároveň aj jeho minimálnym modelom a pevným bodom. Ak dokážeme, že po aplikácií nášho operátora bezprostredného dôsledku (t.j. po aktualizácií modelu) sa vlastnosť „je lokálne stratifikovaný“ nezmení, máme v každom kroku sémanticky korektný program.

Veta 1 – O zachovaní lokálnej stratifikovateľnosti

Nech D je lokálne stratifikovaná databáza, G je závislostný graf D, p je extenzionálny predikátový symbol D a $D_A = \{D \cup p(\bar{X}/\bar{t})\}$ (resp. $D_A = \{D \setminus p(\bar{X}/\bar{t})\}$). Potom je D_A lokálne stratifikovaná.

Dôkaz

- pridanie extenzionálneho faktu $p(\bar{t})$
  - ak existuje pravidlo intenzionálnej databázy $C = (A :- L_1, \ldots, L_n)$ t.ž. $L_i = p(\bar{X}/\bar{t}), 1 \le i \le n$, potom $p(\bar{t}) \in B_P$ a teda D_A je lokálne stratifikovaná
  - inak priradíme $p(\bar{t})$ v mapovaní l ľubovolnú hodnotu čo neporuší podmienky lokálnej stratifikovateľnosti
- odstránenie extenzionálneho faktu $p(\bar{t})$

- analogicky

□

3.3.3 Determinizmus pri výbere pravidiel

Pri výpočte môže nastať prípad, kedy je dôležité poradie vyhodnocovaných pravidiel. Z princípu konštrukcie T_1 nemôže poradie vyhodnocovaných pravidiel zmeniť pravdivostné ohodnotenie literálu v klauzule a to z dôvodu, že vstupná intepretácia v jednom kroku T_1 sa nezmení eventuálnymi aktualizáciami pre rôzne klauzuly. Takýmito aktualizáciami sa však môže meniť program (resp. jeho extenzionálna časť), a tak ovplyvňovať $\Re$ ohodnotenie aktualizačných atómov. Problém nastáva v prípade používania silnej sémantiky aktualizácií, kedy $\Re$ ohodnotenie berie ohľad na to, či aktualizovaný atóm už existuje resp. neexistuje v databáze. V prípade, že by v rámci jedného kroku vznikli komplementárne aktualizácie (nie však v rámci jednej klauzuly, to neumožňuje $\Re$ ohodnotenie pre silnú i slabú sémantiku), závisí na poradí vyhodnocovaných klauzúl, a teda celý výpočet je nedeterministický.

Príklad 6 - Nedeterminizmus pri výbere pravidiel

Program P:
$p(a).$
$q(a).$
$r(X) :- +q(X), p(X).$
$s(X) :- q(X).$
$t(X) :- -q(X), p(X).$

$U_P = \{a\}$
$B_P = \{p(a), q(a), r(a), s(a), t(a)\}$
$ground(P) = \{p(a)., q(a)., r(a) :- +q(a), p(a). \ ①,\ s(a) :- q(a). \ ②,\ t(a) :- -q(a), p(a). \ ③\}$

$$T_1(\emptyset) = \{p(a), q(a)\}$$
$$\cdots$$
$$T_1(T_1(\emptyset))$$
$$\vdots \qquad \vdots$$
$$①<③ \qquad ③<①$$
$$\vdots \qquad \vdots$$
$$\{p(a), s(a), t(a)\} \qquad \{p(a), s(a), t(a), r(a)\}$$

V príklade sú naznačené 2 kroky (aplikácie T_1). V prvom kroku sa vytvorí interpretácia pozostávajúca iba z faktov programu P, kým program sa nezmení. V druhom kroku sa aplikuje operátor bezprostredného dôsledku na výsledok prvého kroku. Tu sa prejavuje nedeterminizmus výpočtu podľa zvolenia poradia vyhodnocovania aktualizácií v klauzulách.

V prípade, že klauzula $r(a) :- +q(a), p(a).$ (označená ako 1) bude vyhodnotená pred klauzulou $t(a) :- -q(a), p(a).$ (označená ako 3), bude vyhodnotená ako neúspech,

pretože podľa silnej sémantiky aktualizácie je $\Re$ ohodnotenie aktualizačného atómu na vloženie už existujúceho faktu *false* a tak výsledná interpretácia nebude obsahovať hlavu klauzule 1 ($r(a)$).

Naopak v prípade, že bude klauzula 3 vyhodnotená pre klauzulou 1, klauzula 3 bude úspešná (program obsahuje fakt $q(a)$ a preto ho môže odstrániť) a následne bude úspešná aj klauzula 1 (program už neobsahuje fakt $q(a)$ a preto ho môžem pridať). Výsledná interpretácia bude obsahovať ako $t(a)$, tak aj $r(a)$.

Riešením je úprava konštrukcie dôsledkov databáze pomocou operátora bezprostredného dôsledku. Na odstránenie problému s komplementárnymi aktualizáciami potrebujeme zafixovať program (resp. jeho extenzionálnu časť), aby $\Re$ ohodnotenie aktualizačných atómov nebolo prevádzané na programe, ktorý je zároveň aktualizovaný už vyhodnotenými aktualizačnými atómami.

Definícia 30 - deterministický OBED pre program s aktualizáciou v okamžitej sémantike

Nech P je program obsahujúci aktualizácie a I je interpretácia P. Potom je *deterministický operátor bezprostredného dôsledku pre program s aktualizáciou v okamžitej sémantike* T_2 zobrazenie $T_2 : <P, I> \rightarrow <P', J>$, kde P, P' je program s aktualizáciou a I,J sú interpretácie.

$$T_2(P,I) = \{ <(P \cup P^{Add}) \setminus P^{Del}, A \in B_P> \mid \exists r : A :- C_1, \ldots, C_m, A_1, \ldots, A_n \in ground(P),$$
$$\forall i \in \{1, \ldots, n\}\ ak\ A_i\ je\ pozitívny\ tak\ A_i \in I,$$
$$ak\ A_i\ je\ negatívny\ tak\ A_i \notin J,$$
$$\forall C_j, C_k, \Re \mid = (C_j, C_k)\theta, 1 \le j, k \le m$$
$$\forall l \in \{1, \ldots, m\}\ ak\ C_l = +D\ tak\ D \in P^{Add},$$
$$ak\ C_l = -D\ tak\ D \in P^{Del} \}$$

Vyššie uvedená konštrukcia T_2 vychádza z T_1 , nenarúša korektnosť sémantiky a zachováva jeho kontinuálnosť. Zároveň rieši problém eventuálneho nedeterminizmu výberu poradia klauzúl pre vyhodnotenie tým, že neumožní komplementárne aktualizácie i v rôznych klauzulách z dôvodu vyhodnocovania $\Re$ ohodnotenia na rovnakom programe pre všetky z nich.

Na druhej strane sa trochu zoslabí pojem „okamžitej“ aktualizácie nakoľko aktualizácia neprebehne ihneď po vyhodnotení klauzule, ale až po vyhodnotení všetkých klauzúl v rámci jedného kroku T_2 .

3.3.3.1 Aktualizácia databázy pri neúspešnom vyhodnotení cieľa

Ďalším problémom pri vyhodnocovaní programu s aktualizáciami je prípad, kedy sa pri výpočte cieľa nenájde riešenie, ale zaktualizuje sa databáza pri snahe o výpočet.

Príklad 7 - aktualizácia DB pri neúspešnom vyhodnotení cieľa

$$
\begin{array}{ll}
\textit{Program } P: & U_P = \{a, b\} \\
\quad p(a). & B_P = \{p(a), q(a), r(a), \\
\quad q(b). & \qquad p(b), q(b), r(b)\} \\
\quad r(X) :- +q(X), p(X). & \textit{ground}(P) = \{p(a)., q(b)., \\
\textit{Cieľ} & \qquad\qquad r(a) :- +q(a), p(a). \\
\quad ?\, r(b). & \qquad\qquad r(b) :- +q(b), p(b).
\end{array}
$$

$$
\begin{aligned}
&T_2^0(P, \emptyset) = \{P, \{p(a), q(b)\}\} \\
&T_2^1(P, \emptyset) = \{P \cup \{q(a)\}, \{p(a), q(b), r(a)\}\} \\
&T_2^2(P, \emptyset) = \{P \cup \{q(a)\}, \{p(a), q(b), r(a), q(a)\}\} \\
&T_2^3(P, \emptyset) = T_2^2(P, \emptyset) \rightarrow \textit{fail}
\end{aligned}
$$

Po štvrtej aplikácií T_2 sa už nezmení ani P ani I a teda sme dosiahli pevného bodu a hľadaný cieľ sme nenašli, ale databáza je zmenená.

Ku podobnému problému dochádza v prípade, že cieľ dotazu obsahuje premennú, pre ktorú sa hľadajú všetky substitúcie, ale od určitého kroku sa žiadne dalšie substitúcie nenájdu. Napriek tomu môžu prebehnúť ďalšie aktualizácie, čo z hľadiska očakávanej sémantiky nie je správne. Táto situácia môže nastať iba v prípade, že náš algoritmus výpočtu zisťuje v každom kroku, či našiel riešenie (substitúciu).

Príklad 8 - aktualizácia DB pri nájdení všetkých výsledkov

$$
\begin{array}{ll}
\textit{Program } P: & \\
\quad p(a). & U_P = \{a\} \\
\quad r(X) :- +q(X), p(X). & B_P = \{p(a), q(a), r(a), s(a), t(a)\} \\
\quad s(X) :- +t(X), r(X). & \textit{ground}(P) = \{p(a)., q(a)., \\
\textit{Cieľ} & \qquad\qquad r(a) :- +q(a), p(a). \\
\quad ?\, r(Y). & \qquad\qquad s(a) :- +t(a), p(a).
\end{array}
$$

$$T_2^0(P,\emptyset)=\{P,\{p(a)\}\}$$
$$T_2^1(P,\emptyset)=\{P\cup\{q(a)\},\{p(a),r(a)\}\}\rightarrow a/Y$$
$$T_2^2(P,\emptyset)=\{P\cup\{q(a)\},\{p(a),r(a),q(a)\}\}$$
$$T_2^3(P,\emptyset)=\{P\cup\{q(a),t(a)\},\{p(a),r(a),q(a),s(a)\}\}$$
$$T_2^4(P,\emptyset)=\{P\cup\{q(a),t(a)\},\{p(a),r(a),q(a),s(a),t(a)\}\}$$
$$T_2^5(P,\emptyset)=T_2^4(P,\emptyset)\rightarrow\square$$

V druhom kroku sa nájde substitúcia a/Y , ale pokračuje sa ďalej a v štvrtom kroku sa zmení databáza (pridá sa fakt $t(a)$) , no žiadna ďalšia substitúcia sa nenájde.

Jedným z riešení je definovať pre každú aktualizáciu jej opačnú aktualizáciu, čo je jednoduché vzhľadom k našej definícií aktualizácií. Problém nastáva v prípade, keď používame slabú aktualizačnú sémantiku a v postupnosti prevedených aktualizácií sa vyskytnú dve rovnaké aktualizácie. Ak je jedna z týchto aktualizácií vo vetve výpočtu, ktorá sa vyhodnotí ako zbytočná, odstraňujú sa jej dôsledky opačnou aktualizáciou, ktorá však zároveň anuluje aj zmenu prevedenú druhou totožnou aktualizáciou.

Iným riešením je pre každý krok (príp. skupinu krokov, pre ktoré sa databáza nemení) vyhodnotenia udržiavať vlastnú inštanciu databázy a v prípade zistenia zbytočného vyhodnotenia určitej vetvy výpočtu, aktuálnu databázu zahodiť a vrátiť sa k inštancií náležiacej ku poslednému platnému kroku (napr. poslednej substitúcií pre cieľ). Tento spôsob je však značne náročný na pamäťové zdroje a preto je možné nad ním uvažovať iba pre relatívne malé databázy, či už z pohľadu dát alebo veľkosti stromu výpočtu. Alternatívou k tomuto riešeniu je zavedenie určitej formy transakcií, no to znova narušuje pôvodný zámer definovať sémantiku „okamžitých“ aktualizácií.

3.3.3.2 Nekonečnosť výpočtu

Pri výpočte programu pomocou operátora bezprostredného dôsledku môže nastať prípad, kedy sa dostaneme pri kontinuálnej aplikácií do stavu program-interpretácia, v ktorom sme sa už nachádzali. Dosiahneme tým stav, kedy sa už žiadnu novú informáciu nedozvieme, no zároveň sme nenašli pevný bod, a preto nemôžeme ukončiť výpočet.

Príklad 9 - Nekonečnosť výpočtu

Program P:
$p(a).$
$r(X):-+q(X),p(X).$
$s(X):-r(X).$
$t(X):--q(X),s(X).$

$U_P=\{a\}$
$B_P=\{p(a),q(a),r(a),s(a),t(a)\}$
$ground(P)=\{p(a).,$
$r(a):-+q(a),p(a).$
$s(a):-r(a).$
$t(a):--q(a),s(a).$

$$T_2^0(P,\emptyset)=\{P,\{p(a)\}\}$$
$$T_2^1(P,\emptyset)=\{P\cup\{q(a)\},\{p(a),r(a)\}\}$$
$$T_2^2(P,\emptyset)=\{P\cup\{q(a)\},\{p(a),q(a),s(a)\}\}$$
$$T_2^3(P,\emptyset)=\{P,\{p(a),t(a),q(a)\}\}$$
$$T_2^4(P,\emptyset)=\{P\cup\{q(a)\},\{p(a),r(a)\}\}=T_2^1(P,\emptyset)$$

Po štvrtom kroku sa dostaneme do stavu identického so stavom po prvom kroku a v prípade, že výpočet pokračuje (napr. hľadáme všetky substitúcie pre nejaký cieľ alebo dotaz nenašiel žiadne výsledky), dostane sa do nekonečného cyklu, čo porušuje požadovanú vlastnosť Datalogu – konečnosť výpočtu.

Riešením je definovať stavy dvojice program-interpretácia ako vrcholy orientovaného grafu a aplikácie operátora bezprostredného dôsledku ako jeho hrany a pri vzniku cyklu ukončíme výpočet. Pri pridaní hrany môžu nastať nasledujúce prípady

- pridanie hrany nespôsobí vzniknutie cyklu – pokračujeme vo výpočte
- pridanie hrany sposobí vzniknutie cyklu
  - cyklus obsahuje jednu hranu – dostaneme pevný bod
  - cyklus obsahuje viac hrán – operátor vygeneroval všetky dôsledky a neexistuje pevný bod

Vzhľadom na konečnosť B_P existuje konečný počet vrcholov a teda môžeme pridať iba konečný počet hrán, čo zaručuje konečnosť výpočtu.

3.3.4 Sémantika s odložením

Iná forma sémantiky je sémantika aktualizácie s odložením (SASO). Pri použití SASO sa výpočet považuje za jednu atomickú operáciu a tak sa fyzická aktualizácia prevádza až po skončení behu programu, prípadne ak vyhodnotenie programu prebieha v konzistentných krokoch vzhľadom k databázi, po jednom alebo viacerých takýchto krokoch (určitá forma transakcií).

Prvý dôležitý aspekt SASO je jej priblíženie k sémantike aktualizácie používanej v jazykoch relačných databázových systémoch, konkrétne SQL. V SQL je možné aktualizovať prvky relácie, ktoré určíme pomocou obmedzovacej podmienky v dotaze. Obmedzovacia podmienka je boolovská kombinácia predikátov, ktorá určí prvky pre aktualizáciu. Aktualizačný dotaz sa skladá z výberu prvkov, ktoré budú zmenené. Dôležité je, že obmedzovacia podmienka sa aplikuje na prvky pred aplikovaním akejkoľvek zmeny, takže jej vyhodnotenie je nezávislé na aktualizáci v rámci jedného dotazu.

Ďalšou vlastnosťou SASO je, že zaručuje pre prevedenú množinu aktualizácií jej nezávislosť na poradí vyhodnocovania aktualizačných atómov, takže SASO zaistí zachovanie deklaratívneho prístupu, umožní vynechať rôzne formy kontroly, charakteristické pre IUS a dáva priestor pre statickú analýzu pravidiel, ktorá môže detekovať pravidlá prevádzajúce nekonzistné aktualizácie.

Oproti IUS umožňuje NUS definovať jednoduché transakcie, kedže prevádza fyzické aktualizácie až po vyhodnotení dotazu, a to buď všetky alebo žiadnu.

Formálnejšia definícia konštrukcie dôsledkov databáze v NUS vychádza podobne ako v IUS z konštrukcie podľa operátoru bezprostredného dôsledku (OBED). Rozdiel oproti bežnej definícií OBEDu je v potrebe pamätať si stav množiny tzv. kandidátov na aktualizáciu, t.j. množiny aktualizačných atómov, ktoré budú po úspešnom behu výpočtu aplikované na databázu.

Definícia 31 - OBED pre program s aktualizáciou v odloženej sémantike

Nech P je program obsahujúci aktualizácie a I je interpretácia P. Potom je *operátor bezprostredného dôsledku pre program s aktualizáciou v odloženej sémantike* T_3 zobrazenie $T_3 : <I, M> \rightarrow <J, N>$, kde P je program s aktualizáciou a I, J sú interpretácie a M,N sú množiny kandidátov na aktualizácie..

$$T_3(I, M) = \{ <A \in B_P, N> | \exists r : A :- C_1, \ldots, C_m, A_1, \ldots, A_n \in ground(P),$$
$$\forall i \in \{1, \ldots, n\} \text{ ak } A_i \text{ je pozitívny tak } A_i \in I,$$
$$\text{ak } A_i \text{ je negatívny tak } A_i \notin J,$$
$$N = M \cup \{C_1, \ldots, C_m\}$$
$$\forall C_j, C_k \in N : \Re \models (C_j, C_k)\theta \}$$

Na rozdiel od operátorov definovaných pre SOKA musíme kontrolovať $\Re$ ohodnotenie pre celú množinu aktualizačných atómov vždy, keď ju rozširujeme, a teda v priebehu celého behu program nám nesmú vzniknúť komplementárne aktualizácie.

Na druhej strane je pomerne jednoduché ukázať, že problémy, ktoré vznikali pri SOKA nám pri použití SASO nevznikajú.

Nakoľko sa extenzionálna časť databázy mení až na konci behu výpočtu, nemôže nastať problém pri determinizme výberu pravidiel v jednotlivých krokoch programu. Nemôže ani nastať prípad, kedy by pridanie aktualizačného atómu do množiny kandidátov na aktualizácie v jednom kroku spôsobilo, že jej prípadná komplementárna aktualizácia už nebude pridaná (a teda závisí na poradí, v ktorom ich pridávame). Dôvodom je princíp

konštrukcie množiny kandidátov na aktualizácie, kedy sa táto množina rozširuje v rámci jedného kroku tak, že všetky aktualizácie pridané v jednom kroku sa pridávajú naraz a nesmú byť komplementárne ani už z existujúcou množinou kandidátov na aktualizácie a ani medzi sebou.

Riešenie problému aktualizácie databázy pri neúspešnom vyhodnotení cieľa vychádza z „transakčnosti“ SASO, kedy sa databáza aktualizuje až na konci výpočtu. Ak podmienime túto aktualizáciu úspešným behom programu, nemôže nastať prípad, že sa nám zmení databáza pri neúspešnom výpočte.

Vzhľadom na to, že operátor bezprostredného dôsledku pre program s aktualizáciou v odloženej sémantike je monotónny a počet krokov je konečný, výsledkom je pevný bod a teda nemôže nastať prípad, kedy je výpočet nekonečný.

4 Jazyk D-Datalog

V tejto časti si definujeme syntax a sémantiku nového jazyka Dynamický Datalog (D-Datalog), ktorý rozširuje Datalog a zavádza možnosť aktualizovať extenzionálnu a materializovať intenzionálnu databázu. Jazyk D-Datalog pracuje so stratifikovateľnými databázami.

4.1 Syntax D-Datalogu

Syntax D-Datalogu vychádza zo zavedenej syntaxe Datalogu s aktualizáciami. Na rozdiel od zvyčajných definícií databázy Datalogu, rozširuje túto definíciu o štruktúry spojené s aktualizáciami, materializáciou a integritnými obmedzeniami.

Definícia 32 - Jazyk D-Datalogu

Jazyk D-Datalogu Δ je akýkoľvek jazyk Datalogu s aktualizáciami.

Definícia 33 - Databáza D-Datalogu

Databáza D-Datalogu je databáza Datalogu v jazyku D-Datalogu rozšírená o

- *množinu intenzionálnych faktov (IFDB)*
- *množinu intenzionálnych kandidátov (ICDB)*
- *množinu integritných obmedzení (IDB_{IO})*

Rozšírenie databázy D-Datalogu o nové štruktúry je dôsledkom potreby persistentne uchovávať informácie o jej zmenách a udržovaní jej konzistencie. Definujeme tiež novú formu intenzionálnych pravidiel, vzhľadom na zavedenie aktualizačných atómov.

Definícia 34 - Intenzionálna databáza D-Datalogu

Intenzionálna databáza (IDB) D-Datalogu je množina pravidiel vo forme

$$H \ :- \ U_1, \ldots, U_s, B_1, \ldots, B_t,$$

kde $B_1, \ldots, B_t$ sú základné alebo odvodené atómy (tzv. dotazovacie atómy), H je odvodený atóm a $U_1, \ldots, U_s$ sú aktualizačné atómy.

Poradie atómov v tele pravidla je irrelevantné, a tak v snahe vyhnúť sa koplikovaným definíciam sme zaviedli vyššie uvedený zápis, kde sme spojili aktualizačné a dotazovacie atómy do skupín.

Nové štruktúry databázy D-Datalogu sú definované ako množiny formúl v jazyku D-Datalogu.

Definícia 35 - Množina intenzionálnych kandidátov

Množina intenzionálnych kandidátov (ICDB) databázy D-Datalogu je množina ground atómov v tvare $p^{\gamma}(\bar{X}), \gamma = \{+, -\}$ t.ž. $p(\bar{X}) \in ICDB \Rightarrow p \in Pred^I$

Definícia 36 - Množina intenzionálnych faktov

Množina intenzionálnych faktov (IFDB) databázy D-Datalogu je množina ground atómov t.ž. $p(\bar{t}) \in IFDB \Rightarrow p \in Pred^I$

Definícia 37 - Množina integritných obmedzení

Množina integritných obmedzení (IO) je množina pravidiel vo forme $U :- E_1, \ldots, E_s, B_1, \ldots, B_t$, kde $B_1, \ldots, B_t$ sú základné alebo odvodené atómy (tzv. dotazovacie atómy), U je odvodený aktualizačný atóm a $E_1, \ldots, E_s$ sú aktualizačné atómy reprezentujúce kandidátov na aktualizácie (t.j. množinu aktualizačných atómov, ktoré fyzicky aktualizujú EDB (príp. IFDB) po úspešnom vyhodnotení programu D-Datalogu).

4.2 Sémantika D-Datalogu

Pre D-Datalog sme zvolili sémantiku aktualizácie s odložením (SASO). Jazyk D-Datalogu zavádza pre každú jeho reláciu možnosť jej aktualizácie. Neobvyklá, ale z pohľadu zovšeobecnenia prístupu a dotazovania nad reláciami bez nutnosti zisťovať, či je relácia základná alebo odvodená, je možnosť aktualizovať tiež intenzionálnu časť databázy D-Datalogu. Ak je p odvodená relácia, potom aktualizačný atóm $\pm p(t_1, \ldots, t_n)$ explicitne pridá resp. odstráni fakt $p(t_1, \ldots, t_n)$ z množiny intenzionálnych kandidátov.

Náš prístup definuje ako súčasť databázy D-Datalogu množinu intenzionálnych faktov (IFDB). IFDB je množina ground atómov, ktoré vznikli ako výsledok procesu nazývaný

materializácia pohľadu. Pohľad definujeme ako odvodený vzťah v pojmoch základných vzťahov a materializácia je všeobecný proces vytvárania a uchovávania dát generovaných z týchto pohľadov ([9]). V prípade D-Datalogu môžme definovať pohľad ako pravidlo intenzionálnej databázy. Pohľad umožňuje odvodzovať logické dôsledky z pravidiel intenzionálnej databázy a teda tvorí jej zamýšlaný význam. Jeho materializácia je teda preferovaný model databázy. Zmyslom zavádzania materializácie je snaha o optimalizáciu výkonu dotazovania hlavne pri výskyte rekurzívnych predikátov. Materializácia umožňuje urýchliť vyhodnocovanie dotazov pomocou zexplicitňovania implicitných väzieb.

Problém nastáva v prípade aktualizácie extenzionálnej databázy, IFDB je nutné odvodiť znovu. V prípade, že by sme odvodzovali pohľady znovu z celej databázy, bolo by to veľmi neefektívne. Ideálne je odvodiť iba pohľady závislé na aktualizovaných reláciach. Na to existuje heuristika - inkrementálna údržba materializovaných pohľadov, ktorá sa zaoberá algoritmami (tzv. algoritmy inkrementálnej údržby) na opätovné odvodenie častí pohľadov, ktoré boli závislé na aktualizovaných dátach. Techniky údržby sú kľúčové pre materializácie, kedže ju umožňujú inkrementálne aktualizovať v prípade zmeny a celkovo určujú stupeň optimalizácie.

Problém aktualizácie D-Datalogu okrem iného spočíva v prevedení aktualizácie intenzionálneho pravidla na aktualizáciu (či dokonca viac aktualizácií) extenzionálnych faktov. Aby sme sa vyhli nedeterministickému prevedeniu na extenzionálne fakty, zavádzame do D-Datalogu množinu intenzionálnych kandidátov (ICDB). ICDB obsahuje explicitne definované aktualizácie intenzionálnej databázy a tak má užívateľ priamú možnosť ovplyvniť vyhodnotenie implicitného pravidla bez ohľadu na možnosť či nemožnosť odvodenia alebo eventuálny dôsledok materializácie.

Množina integritných obmedzení (IO) je množina intenzionálnych pravidiel, ktoré určujú konzistentný stav databázy. Po vyhodnotení aktualizačného programu sa vyhodnotia integritné obmedzenia a v prípade, že stav databázy po aktualizáciach prevádzaných programom nie je konzistentný, prevedú sa upravujúce aktualizácie.

Celkovo môžeme výpočet rozdeliť na 2 vetvy. Prvá počíta výsledok dotazu bez aktualizácií , tá druhá výsledok dotazu s aktualizáciami.

4.2.1 Dotaz bez aktualizácií

V prípade, že cieľ C neobsahuje aktualizačné atómy, využívame materializácie ku optimalizácií dotazu. Vzhľadom na to, že máme skonštruovanú materializáciu databázy (t.j. odvodené všetky fakty z pravidiel), prevádzame dotaz iba nad $EDB \cup IFDB$, teda iba nad „extenzionálnou“ časťou databázy. Hľadáme teda substitúciu θ t.ž. $C\theta$ je pravdivá v perfektnom modele DB, v tomto prípade v množine $EDB \cup IFDB$.

4.2.2 Dotaz s aktualizáciami

Ak dotaz obsahuje aktualizácie, môžeme ho zapísať nasledovne $U_1^E, \ldots, U_s^E, U_1^I, \ldots, U_t^I, B_1, \ldots, B_u$, kde $U_1^E, \ldots, U_s^E$ sú aktualizačné atómy extenzionálnych relácií, $U_1^I, \ldots, U_t^I$ sú aktualizačné atómy intenzionálnych relácií a $B_1, \ldots, B_t$ sú dotazovacie atómy. V tomto prípade si výpočet rozdelíme do viacerých krokov.

Uvažujeme rozdelenie dotazu na aktualizačnú a dotazovaciu časť. V prvom kroku prevedieme výpočet nad dotazovaciou časťou obdobne ako pre dotaz bez aktualizácií. Ak existuje riešenie, vytvoríme z dotazovacej časti množinu kandidátov na aktualizácie. V ďalšom kroku na základe integritných obmedzení eventuálne rozšírime túto množinu o ďalšie aktualizácie a v treťom kroku pomocou algoritmu DRed (viz. príloha A) odvodíme materializáciu a množinu aktualizácií eventuálne rozšírime o aktualizácie, ktoré vznikli pri materializácií. V nasledujúcom kroku upravíme ICDB na základe aktualizácií intenzionálnych pravidiel v množine aktualizácií a konečne v poslednom kroku fyzicky zapíšeme zmeny do EDB a upravíme IFDB podľa ICDB.

4.2.2.1 Krok 1 – Vyhodnotenie dotazovacej časti

Tento krok sa prevádza na dotazovacej časti dotazu a je totožný s postupom v prípade dotazu bez aktualizácií. V prípade, že dotaz sa vyhodnotí ako pravdivý (t.j nájde sa nejaká substitúcia prípadne špeciálny typ substitúcie *True*, ak neobsahuje dotaz premenné), postupujeme ďalej ku aktualizácií databázy. V opačnom prípade ukončíme výpočet.

Algoritmus 1 - Krok 1

```
   Vstup: dotaz  D = C1, ..., Cn, B1, ..., Bm
          databáza DB < EDB,IDB,IFDB,ICDB,IO >
  Výstup: množina substitúcií S alebo fail

 Program:
      S = Ø
      for λ in (B1, ..., Bl)λ⊂(DB.ICDB∪DB.EDB) {
```

```
        S = S∪{λ}
    }
    if S = Ø {
       fail
    }
    return S
```

4.2.2.2 Krok 2 – Aplikácia integritných obmedzení

V tomto kroku vytvoríme množinu kandidátov na aktualizácie (KnA) do ktorej vložíme všetky aktualizačné atómy z aktualizačnej časti dotazu. Overíme, že platí $\forall U_j, U_k \in KnA \exists \theta : \Re \models (U_j, U_k)\theta\}$, a teda KnA neobsahuje neplatné aktualizácie. Potom sa vyhodnotia integritné obmedzenia a v prípade, že KnA obsahuje úpravy databázy, ktoré by viedli k nekonzistentnosti, rozšíri sa táto množina o upravujúce aktualizácie.

Algoritmus 2 - Krok 2

```
  Vstup: dotaz  D = C1,...,CnB1,...,Bm
         databáza DB < EDB,IDB,IFDB,ICDB,IO >
  Výstup: množina kandidátov na aktualizácie KnA
         alebo fail

Program:
     if ∃θ:ℜ|=(Cj,Ck)θ,1≤i,j≤n} {
        continue
     } else {
        fail
     }
     C = C1,...,Cm
     KnA = {C1}∪...∪{Cm}
     for E = E1,...,Ek,F1,...,Fl in DB.IO {
        for λ in (F1,...,Fl)λ⊂(DB.ICDB∪DB.EDB) {
           if (∀Ei,1≤i≤k:Eiλ∈C) {
                 KnA = KnA∪{Eλ}
           }
        }
     }
     return KnA
```

Telá integritných obmedzení môžu obsahovať aktualizačné i neaktualizačné atómy. Neaktualizačné atómy $B_1, \ldots, B_t$ sa vyhodnotia podobne ako v kroku 1, t.j. hľadajú sa všetky substitúcie λ t.ž. $\forall B_i, 1 \le i \le t : B_i\lambda \in EDB \cup IDFB$. Aktualizačné atómy sa vyhodnocujú nasledovne. Aktualizačný atóm E_i v tele IO sa považujú za pravdivý, ak $E'_i\lambda \in KnA$.

Hlavičky integritných obmedzení, ktoré sa vyhodnotia ako pravdivé, rozšíria množinu kandidátov na aktualizácie.

4.2.2.3 Krok 3 - Materializácia

V tomto kroku odvodíme množinu intenzionálnych faktov pri zohľadnení stávajúcej databázy a množiny kandidátov na aktualizácie. Zároveň tento výpočet môže rozšíriť KnA. Existujúci program modifikujeme podľa algoritmu DRed (Príloha A) a ako množinu aktualizácií použijeme podmnožinu KnA obsahujúcu aktualizáciu extenzionálnych relácií.

Algoritmus 3 - Krok 3

```
   Vstup: množina kandidátov na aktualizácie KnA
          databáza DB < EDB,IDB,IFDB,ICDB,IO >
  Výstup: materializovaná databáza MDB
          množina kandidátov na aktualizácie KnA
          alebo fail

Program:
     lokDB = DRed(DB,KnA)
     MDB = DB
     <I,N> = fixpoint( T3(Ø,Ø),lokDB )
      KnA = KnA∪{N}
     if not(∃θ:ℜ|=(Ej,Ek)θ,Ei∈KnA,1≤i,j≤|KnA|) {
        fail
     }
     MDB.IFDB = I
     return MDB,Kna
```

Následne vytvoríme pomocou operátora T_3 a sémantiky pevného bodu, preferovaný model databázy. Pri konštrukcií pomocou tohto operátora získame tiež množinu kandidátov na aktualizácie, ktorá vznikla pri výpočte. Rozšírime KnA o túto množinu. Intenzionálna časť preferovaného modelu sa stane IFDB.

4.2.2.4 Krok 4 – Aplikácia aktualizácie do ICDB

V tomto kroku si rozdelíme KnA na 2 disjuktné množiny podľa toho, či aktualizujú intenzionálnu alebo extenzionálnu reláciu.

Algoritmus 4 - Krok 4

```
   Vstup: množina kandidátov na aktualizácie KnA
          materializovaná databáza MDB
          < EDB,IDB,IFDB,ICDB,IO >
  Výstup: materializovaná databáza MDB s upravenou
          množinou ICDB
```

```
Program:
     KnA = {KnA_I ∪ KnA_E}
     for αp(t̄), α={+,−} in KnA_I {
        if ∃p^γ(t̄) ∈ MDB.ICDB, γ={+,−}, γ≠α {
            MDB.ICDB = MDB.ICDB\{p^γ(t̄)}
        }
        MDB.ICDB = MDB.ICDB∪{p^α(t̄)}
     }
     return MDB
```

Prvky z podmnožiny KnA obsahujúcej aktualizácie intenzionálnych relácií sa transformujú na atómy ICDB. V prípade, že ICDB už obsahuje prvok, ktorý reprezentuje komplementárnu aktualizáciu ku práve pridávanej, tento prvok sa odstráni z ICDB.

4.2.2.5 Krok 5 – Aktualizácia EDB

Posledný krok spočíva v prevedení aktualizácií z extenzionálnej časti KnA do EDB a aktualizácie IFDB pomocou kandidátov na intenzionálne aktualizácie ICDB. Výsledkom je kozistentná, aktualizovaná a materializovaná databáza.

Algoritmus 5 - Krok 5

```
  Vstup: množina kandidátov na aktualizácie KnA
         materializovaná databáza MDB
         < EDB,IDB,IFDB,ICDB,IO >
 Výstup: materializovaná databáza MDB s
         aktualizovanými množinami EDB a IFDB

Program:
     KnA={KnA_I ∪ KnA_E} , KnA_E={KnA_E^+ ∪ KnA_E^-}
     for αp(t̄), α={+,−} in KnA_E {
        if α={−} {
            MDB.EDB = MDB.EDB\{p(t̄)}
        } else {
            MDB.EDB = MDB.EDB∪{p(t̄)}
        }
     }
     for r^γ(t̄), γ={+,−} in DB.ICDB {
        if γ={−} {
            MDB.IFDB = MDB.IFDB\{p(t̄)}
        } else {
            MDB.IFDB = MDB.IFDB∪{p(t̄)}
        }
     }
     return MDB
```

5 Popis implementácie D-Datalogu

Súčasťou nášho návrhu je i prototypová implementácia jazyka D-Datalogu. V tejto kapitole popíšeme jej schému a štruktúru.

Implementácia D-Datalogu je jednoduchý interpret dotazov v jazyku D-Datalogu, ktorý demonštruje nami navrhnutý algoritmus vyhodnocovania. Celá aplikácia funguje v textovom režime, kedy užívateľ po nahraní inicializačnej databázy môže cez príkazovú riadku zadávať dotazy.

Prototypová implementácia jazyka D-Datalogu je napísaná v jazyku Java. Pre objektový jazyk Java sme sa rozhodli z dôvodu dobrej prenositeľnosti a možnosti zapúzdrenia do appletov so zámerom prezentácie cez webové rozhranie. Nevýhodou tohto riešenia je nižšia rýchlosť pri vyhodnocovaní zložitejších dotazov alebo pri rozsiahlejšej databáze, ale vzhľadom na očakávané využitie (demonštrácia implementácie algoritmu) nie je táto vlastosť limitujúca.

5.1 Model procesu vyhodnocovania

Proces vyhodnocovania prebieha v niekoľkých fázach. Prvá fáza je načítanie vstupného súboru s databázou a jej prevedenie do objektového modelu aplikácie. V prípade, že vstupná databáza obsahuje extenzionálne fakty, je potrebné vytvoriť materializáciu tejto databázy. Ak tieto dva kroky prebehnú úspešné, aplikácia sa dostane do stavu, kedy očakáva vstup od užívateľa. V prípade neúspechu aplikácia skončí.

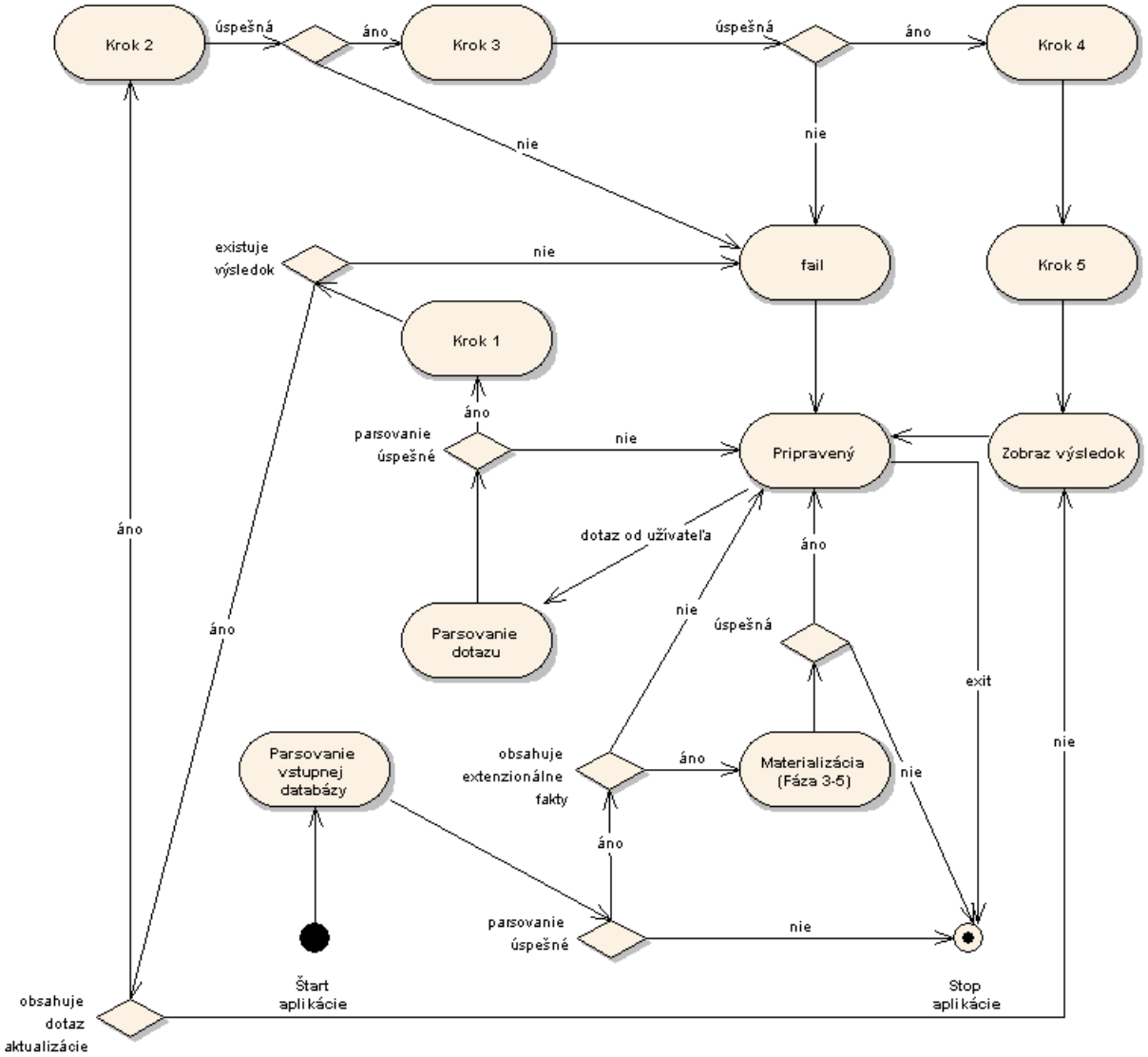
Ďalšou fázou je prijímanie vstupu (dotazov) od užívateľa a jeho vyhodnotenie. Dotaz je postupnosť dotazovacích a aktualizačných (extenzionálnych či intenzionálnych) atómov, oddelená čiarkami a ukončená bodkou. Užívateľom zadaný dotaz sa následne parsuje do objektového modelu aplikácie a posiela sa na spracovanie spolu s databázou do algoritmu vyhodnocovania D-Datalogu. V prípade, že nastane pri parsovaní chyba, dostane sa aplikácia do stavu ***fail***, vypíše príčinu chyby a opäť sa prepne do stavu ***pripravený***.

V nasledujúcej fázi sa dotaz rozdelí na 2 disjunktívne časti dotazovaciu a aktualizačnú (najviac jedna z nich môže byť prázdna). Ak je dotazovacia časť neprázdna, vyhľadávajú sa výsledné substitúcie (Krok 1). Ak sa nenájde žiadna substitúcia, opäť sa dostane aplikácia do stavu ***fail*** a následne do stavu ***pripravený***. V opačnom prípade pokračuje výpočet do stavu

vyhodnocovania integritných obmedzení (Krok 2) a následne do stavu materializácie (Krok 3). Ak aktualizačné atómy vzniknuté vo Fáze 2 a Fáze 3 narúšajú sémantiku silnej aktualizácie, považuje sa to za chybu a jej dôsledok je stav ***fail***.

V záverečnej fázi sa prevedie aktualizácia ICDB (Krok 4) a aktualizácia EDB a IFDB (Krok 5) a aplikácia sa dostane späť do stavu ***pripravený***. V tejto fázi sú dáta už v konzistentnom stave a preto nenastáva prípad prepnutia do stavu ***fail***. Model procesu vyhodnocovanie je schématicky znázornený na obrázku 1.

Obrázok 1 - Model procesu vyhodnocovania



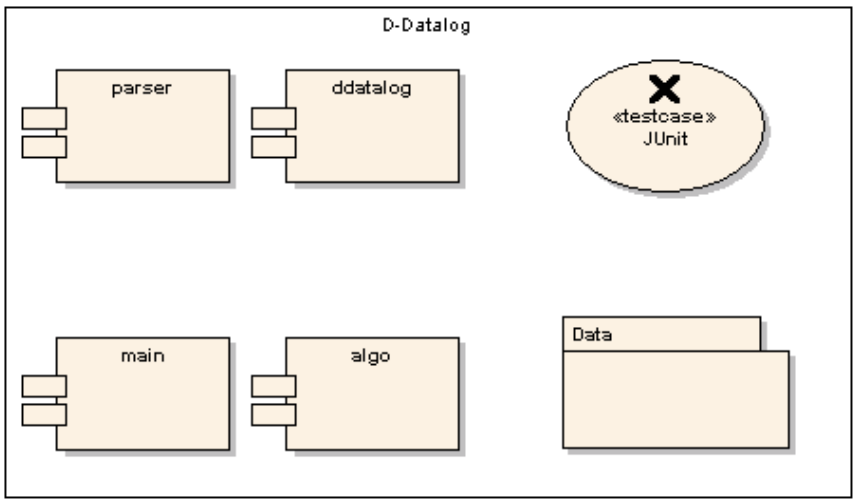
5.2 Schéma aplikácie

Aplikácia pozostáva zo 4 komponentov, balíčka dátových tried a sady JUnit testov.

- **Komponenty**
  - **Parser**

    Tento komponent obsahuje triedy, ktoré slúžia na spracovanie vstupných dát (databáza, dotazy) v textovej forme a prevádza ich do objektového modelu. Na základe definovanej syntaxe dokáže rozpoznať rôzne typy i špeciálnych entít ako aktualizačné či nedefinitné atómy.

    Obrázok 2 - Schéma aplikácie



  - **Algo**

    V komponente algo sú implementované algoritmické postupy používané pri výpočte. Obsahuje implementáciu materializačného algoritmu DRed, implementáciu algoritmu konštrukcie pevného bodu za použitia operátora bezprostredného dôsledku a unifikačné algoritmy.

  - **DDatalog**

    V tomto komponente sú impementácie jednotlivých krokov algoritmu vyhodnocovania jazyku D-Datalogu tak ako sú definované v predchádzajúcej kapitole.

  - **Main**

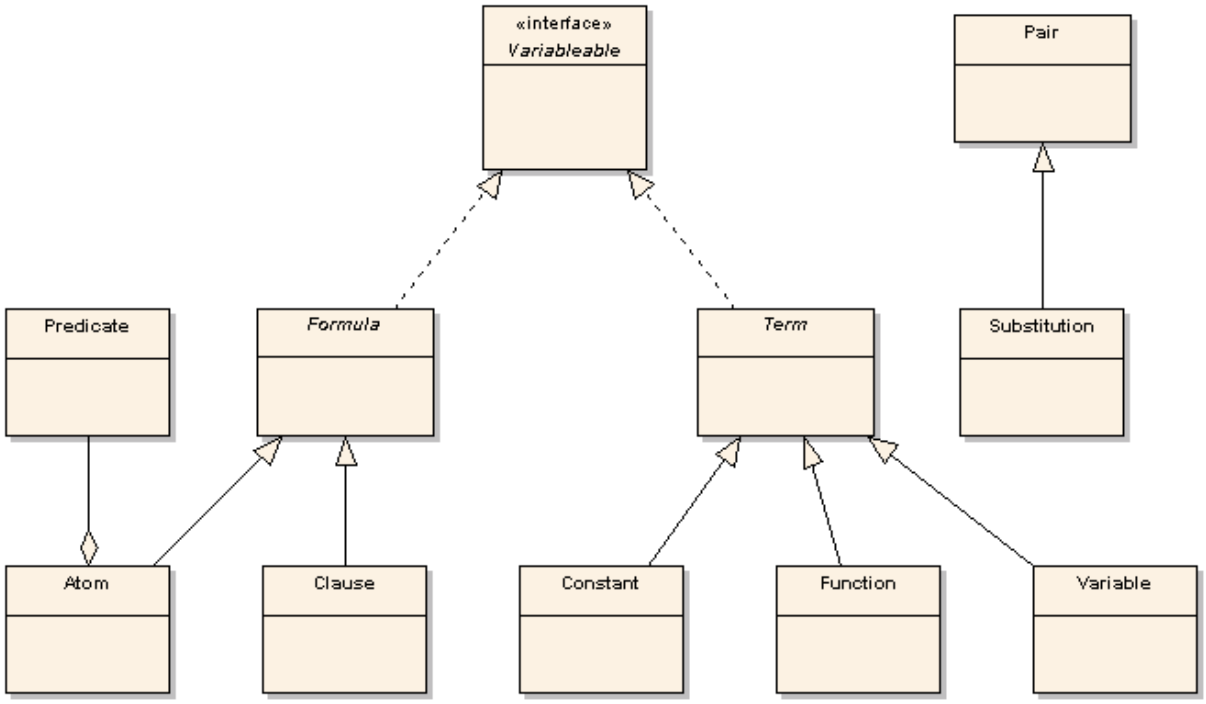
    Komponent main obsahuje riadiace triedy, ktoré slúžia na prijímanie vstupu od užívateľa, prevádzanie príkazov a následné zobrazovanie výsledkov. Tento komponent obsahuje tiež triedy aplikačných výnimiek a utility na klonovanie inštancí tried.

- **Balíčky**
  - **Data**

    Balíček data obsahuje dátové triedy reprezentujúce jednotlivé entity (term, klauzula,...) používané v aplikácií.
- **Testy**
  - **JUnit**

    Funkcionalita komponent je pokrytá unit testami a testovacími zostavami podľa špecifikácie JUnit.
  -

Obrázok 3 - Schéma dátového balíčku



5.3 Popis tried

Aplikácie je rozvrstvená do 5 balíčkov vzhľadom na použitie jednotlivých tried.

5.3.1 Balíček data

Balíček ***data*** obsahuje nasledujúce triedy a rozhrania

- rozhranie **Variableable** (rozširuje *java.io.Serializable*)

  Hlavičky metódy rozhrania Variableable umožňujúce z implementujúcich tried získať množinu premenných.
- abstraktná trieda **Term** (implementuje *Variableable*)

  Abstraktná trieda reprezentujúca spoločného predka prvkov z množiny termov

jazyka 1.rádu. Jej potomci sú kokrétne typy termov: Konštanty, Funkcie a Premenné.

- trieda **Constant** (rozširuje *Term*)
  Trieda reprezentujúca konkrétny term – konštantu.
- trieda **Function** (rozširuje *Term*)
  Trieda reprezentujúca konkrétny term – funkciu.
- trieda **Variable** (rozširuje *Term*)
  Trieda reprezentujúca konkrétny term – premennú.
- abstraktná trieda **Formula** (implementuje *Variableable*)
  Abstraktná trieda reprezentujúca spoločného predka pre formule jazyka 1.rádu. V implementácií D-Datalogu má 2 potomkov: Atóm a Klauzulu.
- trieda **Atom** (rozširuje *Formula*)
  Trieda reprezentujúca atomickú formulu v jazyku D-Datalogu. Skladá sa z množiny termov a príznakov určujúcich, či je atóm definitný, aktualizačný či ground.
- trieda **Clause** (rozširuje *Formula*)
  Trieda reprezentujúca intenzionálnu alebo extenzionálnu klauzulu v jazyku D-Datalogu. Skladá sa z atómu reprezentujúceho hlavičku a kolekcie atómov reprezentujúce telo.
- trieda **Predicate**
  Meta-trieda nesúca informáciu o predikátovom symbole a arite predikátu.

5.3.2 Balíček parser

Balíček ***parser*** obsahuje triedy na spracovanie a konverziu textového vstupu do objektového modelu. Okrem funkčných tried obsahuje tiež triedy JUnit testov.

- trieda **DatalogParser**
  Trieda DatalogParser obsahuje metódy na rozpoznanie jednotlivých entít podľa syntaxe a ich prevedenie do objektov. Vstupný prúd znakov rozdelí postupne na reťazce reprezentujúce riadky, klauzuly, atómy a termy a následne z nich vybuduje strom objektov
- trieda **RuleParser**
  Trieda RuleParser je pomocná trieda na rekurzívne rozparsovanie termov – funkcií v atómoch.

5.3.3 Balíček algo

Balíček ***algo*** obsahuje triedy implementujúce pomocné algoritmy a postupy, ktoré sa využívajú pri výpočte hlavného algoritmu vyhodnocovania. Okrem funkčných tried, obsahuje tiež triedy JUnit testov.

- trieda **AnswerSubstitution**

  Trieda AnswerSubstitution implementuje algoritmus na nájdenie výsledných substitúcií pre danú databázu a dotaz. Tento algoritmus vytvorí novú databázu s extenzionálnou databázou tvorenou EDB a IFDB zo vstupnej databázy. Vytvorená databáza má práve jednu intenzionálnu klauzulu, ktorá má hlavičke atóm obsahujúci všetky premenné v dotaze a ako telo má samotný dotaz. Následne sa táto databáza vyhodnotí a substitúcie intenzionálnej klauzule sú výsledné substitúcie pre vstupný dotaz a databázu.

- trieda **DeleteAndRederive**

  Trieda DeleteAndRederive je implementácia algoritmu uvedeného v Prílohe A. Pre vstupnú databázu a množinu aktualizácií zkonštruuje údržbovú databázu, ktorá sa vyhodnocuje namiesto pôvodnej databázy, a to z dôvodu optimalizácie materializácie pôvodnej databázy pri zmene EDB.

- trieda **ICOperator**

  Trieda ICOperator implementuje funkčnosť operátoru bezprostredného dôsledku pre program s aktualizáciou v odloženej sémantike T_3. Pre vstupnú interpretáciu a množinu kandidátov na aktualizácie vráti zodpovedajúce zobrazenie. Každá inštancia triedy musí mať definovanú databázu (stratifikovanú) pre ktorú následne dokáže (hierarchicky) zkonštruovať pevný bod.

- trieda **Stratification**

  Trieda Stratification implementuje algoritmus, ktorý pre danú databázu určí, či je stratifikovateľná alebo nie.

- trieda **StrongSemantics**

  Trieda implementujúca mechanizmus na kontrolu silnej sémantiky aktualizácií použitú v D-Datalogu.

- trieda **Substitutions**

  Pomocná trieda, ktorá má ako vstup substitúciu a inštanciu triedy implementujúcu Variableable. Výsledok je vstupný objekt na ktorom bola aplikovaná vstupná substitúcia.

5.3.4 Balíček ddatalog

Balíček ***ddatalog*** obsahuje triedy implementujúce kroky algoritmu D-Datalogu a dátový objekt reprezentujúci databázu D-Datalogu.

- trieda **Database**

  Dátová trieda reprezentujúca databázu D-Datalogu (teda pätica EDB, IDB, IFDB, ICDB, IO)

- triedy **Phase1-5**

  Triedy obsahujúce implementáciu algoritmu vyhodnocovania dotazov nad databázou D-Datalogu

5.3.5 Balíček main

Balíček ***main*** obsahuje hlavnú triedu programu, triedy výnimiek a všeobecné pomocné triedy.

- trieda **Main**

  Hlavná trieda aplikácie. Obsahuje spracovanie vstupu od užívateľa a následné predanie kontroly zodpovedajúcim komponentom.

- Triedy rozširujúce aplikačnú výnimku (rozširujú java.lang.Exception)

  - trieda **Fail**

    Reprezentuje stav, kedy algoritmus zaznamenal porušenie silnej sémantiky aktualizácií alebo nenašiel riešenie dotazu.

  - trieda **ParsingException**

    Nastáva v prípade, že sa komponente nepodarilo previesť vstup od užívateľa do objektového modelu aplikácie.

  - trieda **StrongSemanticsViolatedException**

    Nastáva v prípade, že aktualizácia narúša silnú sémantiku aktualizácií.

  - trieda **ComplementaryUpdateException**

    Nastáva v prípade, že množina aktualizácií obsahuje komplementárne aktualizácie.

- trieda **CreateCopyFactory**

  Pomocná trieda vytvárajúca kópie inštancií objektov pomocou serializácie.

6 Záver

V tejto práci sme skúmali možnosť rozšírenia Datalogu o dynamické vlastnosti, konkrétne možnosť aktualizácie a optimalizáciu vyhodnocovania dotazov pomocou zexplicitňovania implicitných pravidiel. Cieľom bolo vytvoriť konkrétny návrh rozšírenia Datalogu o aktualizácie, schopnosť materializácie intenzionálnych pravidiel a ukázať použiteľnosť nášho návrhu na prototypovej implementácií.

Výsledkom je návrh jazyka D-Datalogu, ktorý vychádza zo syntaxe a sémantiky Datalogu a rozširuje ho o možnosť aktualizácie intenzionálnej a extenzionálnej databáze. Ďalšou črtou D-Datalogu je zavedenie množiny intenzionálnych faktov (materializácie) pomocou ktorej optimalizuje dotazy nad databázou. Kombináciou týchto dvoch vlastností vznikol model s predpokladom na rýchle vyhodnocovanie dotazov a zároveň nelimitujúcou možnosťou aktualizácií. Použiteľnosť návrhu je ukázaná v protoypovej implementácií D-Datalogu v jazyku Java, ktorá je súčasťou práce. Vzhľadom na výsledok považujeme vytýčené ciele práce za splnené.

Návrhy a riešenia použité v tejto práci môžu byť základom ďalšieho výskumu v rôznych smeroch a oblastiach. Jedným z nich je možnosť skúmania optimalizácie vyhodnocovania programu s aktualizáciou pomocou metódy tzv. magic sets, t.j snahy upraviť orientáciu bottom-up vyhodnocovania viac na cieľ. Iná oblasť pre výskum je tzv. okamžitá sémantika, jej korektnosť a prípadná implementácia.

Existujúce prístupy zavedenia aktualizácií do deduktívnych databáz, sa venujú tomouto problému viac z teoretického hľadiska. Pri štúdií existujúcich návrhov ([1],[2],[3],[4],[8]) pozorujeme, že väčšina z nich definuje rozšírenia Datalogu o aktualizácie v rovine rozšírenia a nadviazania na jeho teoretický podklad. Tieto riešenia sa zameriavajú buď na vytváranie nových rozšírení so špecifickými črtami ([2],[8]) prípadne definujú a formujú aktualizácie v pojmoch logického programovania (constraint logic programming - CLP) ([1],[3],[4]). Náš návrh, na rozdiel od týchto prístupov, navrhuje riešenie, ktoré iba minimálne mení stávajúcu sémantiku Datalogu a jeho aktualizačný

mechanizmus túto sémantiku zapúzdruje. Hlavným prínosom práce je práve návrh rozšírenia o možnosť aktualizácie databázy spolu s optimalizáciou pomocou materializácie a ukážka jej praktickej použiteľnosti, kedže neexistuje žiaden podobný návrh.

Zoznam citovanej literatúry

[1] Montesi D., Bertino E., Matrelli M. (1997) : *Transactions and Updates in Deductive Databases,*IEEE Transactions on Knowledge and Data Engineering archive, vol. 9, issue 5, 573 – 600.

[2] Torlone R. (1994) : *Update operations in deductive databases with functional dependences*, Acta Informatica archive, vol. 31, issue 6, 573-600.

[3] Fioravanti F., Pettorossi A., Proietti M. (2004) : *Transformation Rules for Locally Stratified Constraint Logic Programs.*

[4] Bertino E.,Catania B.,Gervasi V.,Raaeta A. (1998) : *Active-U-Datalog: Integrating Active Rules in a Logical Update Language*, Lecture Notes in Computer Science, vol.1472, 107 - 134.

[5] Voronkov B. (2002) : Lecture notes of International Master Programme on Computational Logic.

[6] Zaniolo C.,Ceri S.,Faloutsos Ch.,Snodgrass R.T.,Subrahmanian V.S.,Zicari R. (1997) : *Advanced Database Systems*.Morgan Kaufmann Publishers, Inc..San Francisco, California, USA.

[7] Štěpánek P. (2000) : *Predikátová logika,* Praha.

[8] Naqvi S., Krishnamurthy R. (1988) : *Database Updates in Logic Programming*, Symposium on Principles of Database Systems archive, Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 251- 262.

[9] Gupta A., Mumick I.S. (1995) : *Maintenance of Materialized Views: Problems, Techniques, and Applications*, IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing, vol. 18, issue 2, 3-18.

[10] Volz R.,Staab S.,Motik B. (2003) : *Incremental Maintenance of Materialized Ontologies*, In 2nd International Conference on Ontologies and Databases (ODBASE), Sicily, Italy.

Prílohy

Príloha A - Algoritmus DRed (Delete and Rederive)

Algoritmus DRed ([10]) vychádza z princípu výpočtu rozdielov v 3 krokoch :

- Nadhodnotiť dôsledky odstránenia faktov, t.j. vytvoriť nadmnožinu všetkých faktov, ktoré môžu byť eventuálne odstránené
- Opätovné odvodenie faktov, ktoré boli označené na odstránenie v kroku 1, ale existujú pre ne alternatívne odvodenia
- Ak je to možné, následky vloženia extenzionálnych predikátov sú premietnuté do IFDB

Prístup je založený na prepísaní originálneho programu a zamýšlaných aktualizácií do údržbového programu, ktorý je následne vyhodnotený namiesto pôvodného programu. Pôvodné pravidlo Datalogu:

$$p :- r_1, \ldots, r_n.$$

je prepísané do množiny nových predikátov a nových údržbových pravidiel, ktoré vyhodnotia rozdiely potrebné na údržbu IFDB. Pre extenzionálne predikáty sa prevedie nasledujúca úprava. Nech S je množina extenzionálnych predikátov, potom upravený program obsahuje

- intenzionálnu klauzulu $s :- s_{EXT}$ pre každý $s \in S$
- extenzionálny fakt $u_{EXT}(\bar{t})$ pre každý fakt $u(\bar{t})$ pôvodnej extenzionálnej databáze

Zamýšlané aktualizácie sa premietnu do údržbového programu nasledovne. Nech $q(\bar{t})$ je extenzionálny fakt

- ak chceme pridať $q(\bar{t})$ do databázy, údržbový program bude obsahovať fakt $q^{Ins}(\bar{t})$
- ak chceme odstrániť $q(\bar{t})$ z databázy, údržbový program bude obsahovať fakt $q^{Del}(\bar{t})$

Údržba intenzionálneho predikátu *p* je dosiahnutá pomocou 6 dodatočných predikátov:

1. p^{Del} vyhodnotí nadhodnotenú množinu faktov, ktoré by mali byť odstránené z IFDB, tzv. kandidáti na odstránenie. Pre extenzionálne predikáty obsahuje p^{Del} explicitne to, čo má byť odstránené z IFDB.
2. p^{Ins} obsahuje množinu faktov, ktoré budú vložené do IFDB. Pre extenzionálne predikáty obsahuje p^{Ins} explicitne to, čo má byť vložené do IFDB.
3. p^{Red} uchováva fakty, ktoré sú označené na zmazanie, ale majú alternatívne odvodenie.
4. p^{New} popisuje nový stav materializácie po aktualizácií.
5. p^{Plus} vyhodnotí skutočné vloženia faktov do IFDB.
6. p^{Minus} vyhodnotí skutočné odstránenia faktov z IFDB.

Vyhodnotením množiny údržbových klauzúl predikátu p získame množinu implicitných vložení a odstránení faktov odvodených z predikátu p do IFDB, prípadne iných predikátov, ktoré závisia na predikáte p .

Príklad 10 - Aplikácia algoritmu DRed

Uvažujme nasledujúcu extenzionálnu databázu

$spoj(f,e).\quad spoj(e,a).\quad spoj(a,b).$
$spoj(b,c).\quad spoj(c,g).\quad spoj(d,c).$
$spoj(e,d).$

a intenzionálny predikát *cesta*, ktorý predstavuje materializovaný pohľad

$cesta(X,Y) :- spoj(X,Y).\qquad [R_1]$
$cesta(X,Y) :- spoj(X,Z), cesta(Z,Y).\qquad [R_2]$

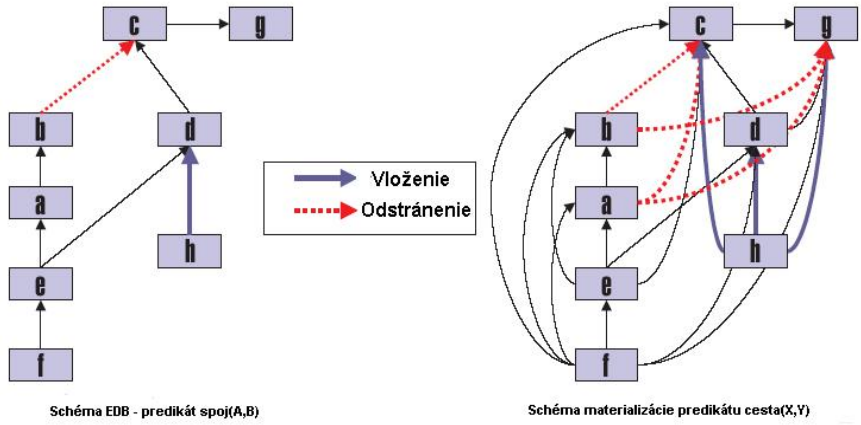
po ktorého vyhodnotení sa vložia nasledujúce fakty do IFDB

$cesta(f,e).\quad cesta(f,a).\quad cesta(f,b).\quad cesta(f,c).\quad cesta(f,g).$
$cesta(f,d).\quad cesta(e,a).\quad cesta(e,b).\quad cesta(e,c).\quad cesta(e,g).$
$cesta(e,d).\quad cesta(a,b).\quad cesta(a,c).\quad cesta(a,g).\quad cesta(b,c).$
$cesta(b,g).\quad cesta(c,g).\quad cesta(d,c).\quad cesta(d,g).$

Ďalej uvažujme efekt nasledujúcich aktualizácií na našu databázu

- odstránenie faktu *spoj(b,c)* z EDB [A_1]
- pridanie faktu *spoj(h,d)* do EDB [A_2]

Obrázok 4 - Schéma príkladu 10



Najskôr vytvoríme extenzionálne fakty a intezionálne pravidlá z pôvodných extenzionálnych faktov. Rozšírime tiež extenzionálnu databázu o informácie o aktualizáciach. Údržbový program pre program z príkladu 10 bude vypadať nasledovne.

Príklad 11 - Upravený program 10 algoritmom DRed

Extenzionálna databáza:

$$spoj_{EXT}(f,e). \quad spoj_{EXT}(e,a). \quad spoj_{EXT}(a,b).$$
$$spoj_{EXT}(b,c). \quad spoj_{EXT}(c,g). \quad spoj_{EXT}(d,c).$$
$$spoj_{EXT}(e,d). \quad spoj^{Del}(b,c). \quad spoj^{Ins}(h,d).$$

Intenzionálna databáza:

$$cesta(X,Y) :- spoj(X,Y). \quad [R_1]$$
$$cesta(X,Y) :- spoj(X,Z), cesta(Z,Y). \quad [R_2]$$
$$spoj(X,Y) :- spoj_{EXT}(X,Y) \quad [R_3]$$

Krok 1 – Množina kandidátov na odstránenie

Prvá množina predikátov údržby materializácie generuje všetkých možných kandidátov na odstránenie pre predikát *p*. Kandidáti na odstránenie sú utvorení z odstránených faktov v tele predikátov. Pre všetky pravidlá a všetky predikáty r_i v tele týchto pravidiel definujeme pravidlo

$$D_i: \quad p^{Del} :- r_1, \ldots, r_{i-1}, r_i^{Del}, r_{i+1}, \ldots, r_n.$$

Ak r_i je extenzionálny predikát, r_i^{Del} obsahuje buď explicitne odstránené fakty v r_i alebo nadmnožinu odvodených faktov, ktoré majú byť odstránené kvôli odstráneniam zapríčineným inými pravidlami.

V programe z príkladu 11 môžeme na základe vyšie uvedenej definície vygenerovať 4 odstraňovacie pravidlá. Transformáciou pravidla *R1* získame jedno a transformáciou pravidla *R2* dve odstraňovacie pravidlá.

$$cesta^{Del}(X,Y) :- spoj^{Del}(X,Y). \quad [R_1 D_1]$$
$$cesta^{Del}(X,Y) :- spoj^{Del}(X,Z), cesta(Z,Y). \quad [R_2 D_1]$$
$$cesta^{Del}(X,Y) :- spoj(X,Z), cesta^{Del}(Z,Y). \quad [R_2 D_2]$$
$$spoj^{Del}(X,Y) :- spoj^{Del}_{EXT}(X,Y). \quad [R_3 D_1]$$

Po vyhodnotení odstraňovacích pravidiel vzhľadom na aktualizáciu *[A1]* dostaneme množinu kandidátov (uvažujeme iba intenzionálny predikát *cesta*) na odstránenie

$$cesta^{Del}(b,c). \quad cesta^{Del}(b,g). \quad cesta^{Del}(a,c).$$
$$cesta^{Del}(a,g). \quad cesta^{Del}(e,c). \quad cesta^{Del}(e,g).$$
$$cesta^{Del}(f,c). \quad cesta^{Del}(f,g).$$

Krok 2 – Alternatívne odvodenie

Ďalším krokom je výpočet množiny faktov, ktoré boli označené na odstránenie, ale existuje pre ne alternatívne odvodenie. Definujeme pre ne nasledujúce pravidlo

$$Z: \quad p^{Red} :- p^{Del}, r_1^{New}, \ldots, r_n^{New}.$$

V programe z príkladu 11 môžeme na základe definície alternatívneho odvodenia vygenerovať 3 pravidlá alternatívneho odvodenia. V definícií používame stav p^{New}, ktorý bude definovaný nižšie.

$$cesta^{Red}(X,Y) :- cesta^{Del}(X,Y), spoj^{New}(X,Y). \quad [R_1 Z]$$
$$cesta^{Red}(X,Y) :- cesta^{Del}(X,Y), spoj^{New}(X,Z), cesta^{New}(Z,Y). \quad [R_2 Z]$$
$$spoj^{Red}(X,Y) :- spoj^{Del}(X,Y), spoj^{New}_{EXT}(X,Z). \quad [R_3 Z]$$

Po vyhodnotení pravidiel alternatívneho odvodenia dostaneme množinu faktov, ktoré majú alternatívne odvodenie

$$cesta^{Red}(e,c). \quad cesta^{Red}(e,g).$$
$$cesta^{Red}(f,c). \quad cesta^{Red}(f,g).$$

Tento stav nastal kvôli faktom *spoj(d,c)* a *spoj(e,d)*, z ktorých boli vygenerované tieto alternatívne odvodenia.

Krok 3 – Vyhodnotenie vložení

Nasledujúci krok prenáša vloženie extenzionálnych faktov do intenzionálnych predikátov. Definujeme v ňom konštrukčné pravidlo, ktoré zlučuje vloženie nových faktov s rozšírením ostatných pravidiel.

$$I_i: \quad p^{Ins} :- r_1^{New}, \ldots, r_{i-1}^{New}, r_i^{Ins}, r_{i+1}^{New}, \ldots, r_n^{New}.$$

Vzhľadom na program z príkladu 11 môžeme na základe definície konštrukčného pravidla odvodiť 3 pravidlá.

$$cesta^{Ins}(X,Y) :- spoj^{Ins}(X,Y). \quad [R_1 I_1]$$
$$cesta^{Ins}(X,Y) :- spoj^{Ins}(X,Z), cesta^{New}(Z,Y). \quad [R_2 I_1]$$
$$cesta^{Ins}(X,Y) :- spoj^{New}(X,Z), cesta^{Ins}(Z,Y). \quad [R_2 I_2]$$
$$spoj^{Ins}(X,Y) :- spoj^{Ins}_{EXT}(X,Z). \quad [R_3 I_2]$$

Vyhodnotenie konštrukčných pravidiel vzhľadom na aktualizáciu $[A_2]$ vytvorí nasledujúcu množinu

$$cesta^{Ins}(h,d). \quad cesta^{Ins}(h,c).$$
$$cesta^{Ins}(h,g).$$

Krok 4 – Stav p^{New}

Nový stav predikátu p po aktualizácií je zachytený v novom predikáte p^{New}, ktorý je definovaný nasledovne

$$N_1: \quad p^{New} :- p, \neg p^{Del.}$$
$$N_2: \quad p^{New} :- p^{Red}.$$
$$N_3: \quad p^{New} :- p^{Ins}.$$

Pravidlo N_1 zaisťuje, že nový stav predikátu p neobsahuje žiadne odstránené informácie. Pravidlo N_2 zaisťuje, že alternatívne odvodenia sú súčasťou nového stavu predikátu a posledný stav N_3 premieta extenzionálne vloženia do nového stavu predikátu.

Krok 5 – Vyhodnotenie skutočných aktualizácií IFDB

Na odvodenie skutočných vložení a odstránení faktov odvodených z predikátu p do IFDB definujeme 2 predikáty p^{Plus}a p^{Minus}. p^{Plus} obsahuje tie fakty, ktoré neboli v IFDB a sú odvodené z vložení do EDB. Naopak p^{Minus} obsahuje fakty, ktoré v IFDB boli, ale algoritmus ich vyhodnotil ako kandidátov na odstránenie a nie sú odvodené ani z vloženia do EDB a ani z alternatívneho odvodenia.

$$Pl: \quad p^{Plus} :- p^{Ins}, \neg p.$$
$$Mi: \quad p^{Minus} :- p^{Del}, \neg p^{Ins}, \neg p^{Red}.$$

Aplikáciou na program z príkladu 11 dostaneme nasledujúce aktualizácie IFDB

$$cesta^{Plus}(h,d). \quad cesta^{Plus}(h,c).$$
$$cesta^{Plus}(h,g).$$
$$\ldots\ldots\ldots$$
$$cesta^{Minus}(a,c). \quad cesta^{Minus}(a,g).$$
$$cesta^{Minus}(b,c). \quad cesta^{Minus}(b,g)$$

čo zodpovedá intuitívnemu výsledku zo schématického náčrtu (Obrázok 4).