Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Preklad logických formúl do Datalogu

(Bakalárska práca)

Peter Juhász

Vedúci: Dr. Tomáš Plachetka

Bratislava, 2008

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Chcem sa poďakovať vedúcemu mojej bakalárskej práce Dr. Tomášovi Plachetkovi za cenné rady a pomoc pri písaní tejto práce.

Abstrakt

Názov práce: Preklad logických formúl do Datalogu

Autor: Peter Juhász

Vedúci práce: Dr. Tomáš Plachetka

V tejto práci sa zaoberáme prekladom formúl relačného kalkulu do programov Datalogu. Prvá časť práce obsahuje úvod do problematiky databázových systémov, bližší popis dvoch spomenutých dotazovacích jazykov a popis konkrétneho algoritmu prekladu. Druhá časť sa zaoberá implementáciou tohto algoritmu. Výsledkom tejto práce je Java aplikácia, ktorá názorne po krokoch uskutočňuje preklad logických formúl do Datalogu.

Kľúčové slová: Datalog, relačný kalkul, prekladač.

Obsah

1 Úvod

Databázam a práci s nimi sa študent informatiky nevyhne a ako neskôr ukážeme, s databázami sa stretávame naozaj na každom kroku. Štúdium Databázových systémov je pozitívna zmena v porovnaní s niektorými abstraktnejšími predmetmi a veríme, že študenti vidia jeho praktický význam i využitie, a tak k nemu aj pristupujú. Na prvý pohľad však možno rádového, databáz bližšie neznalého študenta odradí, že sa má zrazu učiť až *štyri* dotazovacie jazyky naraz. Keď si to porovná s iným programovacím jazykom, ktorému je možno venovaná dvojsemestrálna výučba, tak ho na prvom cvičení prechádza smiech a nastupuje trpký výraz.

Táto práca má ukázať, že tento strach nie je na mieste. Tiež má ukázať, že minimálne dva dotazovacie jazyky—relačný kalkul a Datalog—majú k sebe bližšie, ako by niekto mohol čakať. Jadrom tejto práce bude implementácia algoritmu na preklad formúl relačného kalkulu do Datalogu, vďaka ktorej si užívateľ bude môcť samotný preklad formúl nielen teoreticky naštudovať, ale aj v praxi vyskúšať. V prvej časti tejto práce sa budeme venovať úvodu do databázových systémov, vysvetlíme pojmy ako je databáza, dotazovací jazyk a bližšie popíšeme dva dotazovacie jazyky, ktorých sa týka táto práca. V druhej časti sa oboznámime s algoritmom prekladu ľubovoľnej (bezpečnej) formuly relačného kalkulu do Datalogovského programu, ktorý je kľúčovou časťou implementačnej časti tejto práce. Na záver, v tretej časti tejto práce, popíšeme samotnú implementáciu algoritmu, jednotlivé balíky, triedy, kľúčové metódy a reprezentáciu dát.

2 Prehľad problematiky

2.1 Databázy

Čo je databáza? Bežný, informatikou nezaujatý človek si možno ani neuvedomuje, že s databázami sa stretáva na každom kroku. Keď si ráno v potravinách kupuje pečivo, tak predavačka z databázy tovaru vyberie a spočíta vybrané kúpené položky *(select sum(cena) from tovar where nazov in ('rozok','jogurt','mineralka'))*; keď chce zavolať niekomu známemu, tak z telefónneho zoznamu svojho mobilného telefónu, databázy kontaktov, vyberie ten správny kontakt a zavolá mu (select cislo from kontakty where meno='Kamarat' and priezvisko='Fero'); alebo si len chce na internete v databáze Dopravného podniku mesta zistiť, kedy mu ide najbližší autobus domov (select top 1 cas_odchodu from autobusy where autobus='39' and zastavka='Zochova' and smer='Cintorin Slavicie udolie' and cas_odchodu>=convert(varchar, getdate(), 8)); nehovoriac o tom, že celý internet, s ktorým je čím ďalej tým viac ľudí každodenne v kontakte, je jedna veľká databáza. Čo je teda databáza?

Databáza je zdieľaná integrovaná počítačová štruktúra, ktorá zahrňuje:
- koncové—používateľské dáta, t. j. všetko, čo zaujíma používateľa,
- metadáta, čiže dáta o dátach, prostredníctvom ktorých sú dáta integrované a spravované. Alebo iná, vcelku podobná definícia hovorí, že databázový systém je systém, ktorý poskytuje pohodlné, bezpečné, mnohoužívateľské prostredie pre efektívnu manipuláciu s veľkým objemom perzistentných dát. Databázový systém (DBMS) zahŕňa podporu aspoň jedného koncepčného dátového modelu, podporu jazyka vyššej úrovne pre definíciu dát a manipuláciu s nimi, garantovanie bezpečnosti dát, ako odolnosť voči haváriám systému a

odolnosť voči zlým úmyslom užívateľa, súčasný prístup viacerých užívateľov a zachovanie základných ACID (Atomicity, Consistency, Isolation, Durability) požiadaviek pri práci s transakciami. Aj keď štúdium konkrétnych koncepčných modelov, bezpečnosti a vlastností transakcií je bezpochyby veľmi zaujímavé, bližšie sa budeme venovať len spomínaným vyšším dotazovacím jazykom.

2.2 Dotazovacie jazyky

Dotazovací jazyk umožnuje ovládať databázy pomocou príkazov—dotazov. Príkazy je možné rozdeliť na príkazy pre manipuláciu s dátami, príkazy pre definíciu dát a príkazy pre riadenie dát. Každý dotazovací jazyk preto musí spĺnať nasledujúce predpoklady:

1. Musí obsahovať konštrukcie, z ktorých je možné skladať príkazy na definíciu nových dát a jednoznačného popisu ich štruktúry. Táto časť dotazovacieho jazyka sa nazýva jazyk pre definíciu dát (DDL—data definition language)

2. Musí obsahovať konštrukcie na skladanie príkazov pre kladenie dotazov nad množinou dát v databáze, pre vkladanie nových dát, rušenie a zmenu existujúcich dát. Táto časť dotazovacieho jazyka sa nazýva jazyk pre manipuláciu s dátami (DML—data manipulation language)

3. Musí obsahovať konštrukcie pre riadenie prístupových práv jednotlivých užívateľov systému a riadenie transakcií. Táto časť dotazovacieho jazyka sa nazýva jazyk pre riadenie dát (DCL—data control language).

Medzi takéto dotazovacie jazyky, s ktorými sa pre svoju názornosť a zrozumiteľnosť môžeme najčastejšie stretnúť pri hlbšom štúdiu databázových

systémov nielen na akademickej pôde, patria tieto:

- predikátový kalkul
- relačná algebra
- Datalog
- SQL

V súčasnosti najpoužívanejším dotazovacím jazykom v relačných databázach je jazyk SQL. Avšak málokto tuší, že za týmto rozšíreným jazykom sa v skutočnosti v pozadí skrýva iný dotazovací jazyk—relačná algebra—ktorá tvorí akýsi matematický model jazyka SQL, dalo by sa povedať, že SQL je implementáciou relačnej algebry. Zvyšným dvom, možno nie tak známym, dotazovacím jazykom—Datalogu a predikátovému kalkulu—sme sa podrobnejšie venovali v tejto práci.

2.3 Datalog

Na jazyk Datalog se dá dívať rôznymi spôsobmi. Tým, ktorí poznajú jazyk Prolog neunikne, že ide o syntaktickú podmnožinu Prologu s trochu odlišnou interpretáciou pravidiel. Dalo by sa povedať, že ide o Prolog pre databázy, či o logický aparát tvoriaci nadstavbu relačnej databázy. Jazyk Datalog nemá ani jednoznačného tvorcu, vznikol skôr vývojom. Pri zrode pojmu Datalog sa stretávame s menon David Maier, neskôr sa rozvoju Datalogu venoval významný expert na teóriu databáz Jeffrey D. Ullman.

Datalog je jazykom deduktívnych databáz, pri ktorom predpokladáme existenciu základných relácií daných klasickou relačnou databázou. Aby sme mohli používať dáta z relačných databáz k odvodzovaniu pomocou pravidiel,

je nutné vyjadriť prvky relácií ako logické formuly. To je formálne jednoduché. Napríklad fakt, že n-tica (relácia, IS) je prvkom relácie JE_ČASŤOU vieme vyjadriť pomocou atomickej formuly JE_ČASŤOU(relácia, IS). Takéto formuly se nazývajú v terminológii Datalogu tvrdenia. Naozaj, spomínaná formula označuje tvrdenie "relácia je časťou Informačného systému". V Datalogu se zapisujú pravidlá v tvare $L_0 = L_1, ..., L_n$, kde L_i sú atomické formuly. Majú tvar $P(t_1, ..., t_k)$, kde P je predikátový symbol a t_i je buď premenná alebo konštanta. Symbol P je buď meno základnej alebo pomocnej (virtuálnej) databázovej relácie. Pre aplikácie je vhodné, aby medzi predikátmi boli aj binárne porovnávacie predikáty $>$,$<$,$=$,... Ľavá strana pravidla se nazýva hlava, pravá telo. Pomocné relácie sa definujú pomocou jedného alebo viac pravidiel (s rovnakou hlavou), a to tak, že v pravidle je možné použiť aj odkazy na iné pomocné relácie, dokonca, a to je veľmi dôležité, aj na práve definovanú reláciu. Týmto spôsobom získame rekurzívne pravidlá.

Tvrdenia tvoria tzv. extenzionálnu databázu (EDB) fyzicky uloženú v relačnej databáze. Program v Datalogu je daný množinou pravidiel—intenzionálnou databázou (IDB). Program v Datalogu môžeme taktiež chápať ako dotaz, v ktorom požadujeme na základe EDB skonštruovať relácie, vyskytujúce sa v hlavách pravidiel. Terminológia dotazovania, známa z relačných databáz, sa ale musí trochu modifikovať. Relácia v EDB nie je totiž množinou prvkov (či riadkov tabuľky), ale množinou tvrdení tvaru $p(t_1, ..., t_k)$. Ide však o teoretický rozdiel. Z hľadiska užívateľa sú relácie zobrazované opäť pomocou tabuliek [3].

2.4 Relačný kalkul

Relačný kalkul je jeden z dotazovacích jazykov nad relačnou databázou. Pri vyhodnotení dotazu sa použije relácia z databázy a výstupom každého dotazu je tiež relácia (v krajných prípadoch môže byť výsledkom TRUE alebo FALSE, to však nie je typické použitie dotazov). Relačný kalkul je neprocedurálny dotazovací jazyk, čo znamená, že dotaz sa vytvorí popisom vlastností dát, ktoré z databázy chceme dostať. Dotaz však nešpecifikuje spôsob, akým tieto údaje z databázy vybrať. Existujú dva varianty relačného kalkulu: doménový a n-ticový relačný kalkul, ktorým sa táto práca zaoberá. Už z názvu vyplýva, že NRK (n-ticový relačný kalkul) chápe reláciu ako množinu n-tíc. NRK používa v dotazoch premenné, ktoré nadobúdajú hodnoty n-tíc relácie, teda n-tíc danej schémy relácie. Pre každý atribút, ktorý sa v dotaze použije, musí byť jednoznačne určené meno relácie, ktorá ten atribút obsahuje. Množina n-tíc tejto relácie je potom použitá ako množina hodnôt premennej. Dotaz zapísaný v NRK má tento tvar:

$$\{V|f(V)\}$$

kde V je premenná (resp. viac premenných) a *f(V)* je formula NRK obsahujúca premennú V (množinu premenných). Výsledkom dotazu je množina hodnôt voľných premenných (ak je tých voľných premenných n, tak je výsledkom množina všetkých n-tíc), pre ktoré sa po dosadení $V = T$ vyhodnotí formula *f(V)* ako pravdivá. Vyhodnotenie formuly úzko súvisí s predikátovou logikou prvého rádu, ktorá používa rovnaké logické operácie ako sú použité v jazyku NRK. Pre formálnu definíciu syntaxe NRK sa používajú pojmy atóm (atomická formula) a formula. Najprv definujeme atomickú formulu. Nech U, V sú premenné a K nech je konštanta. Ďalej označme *rel* identifikátor relácie a *op* ako jeden operátor z množiny $\{<, >, =, \neq, \leq, \geq\}$. Potom atomická

formula má jeden z tvarov:

1. *rel(V)*
2. *U op V*
3. *U op K, K op U*

Nech *f* a *g* sú formuly NRK, a *f(V)* je formula, ktorá obsahuje premennú *V*. Formulu NRK definujeme rekurzívne a pomocou atomických formúl takto:

1. každá atomická formula je formulou NRK,
2. formuly $\neg f, f \wedge g, f \vee g, f \Rightarrow g, f \Leftrightarrow g$ sú formulami NRK. Kde symbol $\neg$ je negácia, $\wedge$ konjunkcia, $\vee$ disjunkcia, $\Rightarrow$ implikácia a symbol $\Leftrightarrow$ označuje logickú ekvivalenciu,
3. formuly $\exists V(f(V)), \forall V(f(V))$ su formulami NRK.

Symbol $\exists$ je všeobecný, $\forall$ existenčný kvantifikátor. Ak označíme Q ako kvantifikátor, potom formula $QV(f(V))$ znamená, že kvantifikátor Q viaže výskyt premennej V v podformule *f(V)*. Teda premenná V sa v takomto prípade nazýva viazaná. Premenná je vo formule voľná, ak nie je v tejto formule viazaná kvantifikátorom.

2.5 Preklad relačnej formuly do Datalogu

Aj keď sa to na prvý pohľad možno nezdá, štyri spomínané dotazovacie jazyky—Datalog, relačná algebra, relačný kalkul a SQL majú jednu veľmi dôležitú spoločnú vlastnosť—všetky sú to výpočtovo rovnako silné, úplné jazyky (majú silu Turingovho stroja). Toto tvrdenie sa dá dokázať pri skúmaní konkrétnych dvojíc—dá sa ukázať, že Datalog a SQL sú rovnako silné jazyky, SQL a relačná algebra sú rovnako silné jazyky, relačná algebra a Datalog

sú rovnako silné jazyky a na záver, pre nás najzaujímavejšia ekvivalencia—relačný kalkul a Datalog sú rovnako silné jazyky, t.j. ľubovoľný dotaz v Datalogu sa dá vyjadriť ako formula relačného kalkulu a naopak.

Nasledujúci algoritmus, ktorý tvorí základ implementačnej časti tejto práce, umožnuje práve preklad ľubovoľnej formuly relačného kalkulu do programu Datalogu [2]:

1. Prepíšeme formulu do prenexovej formy. Takže všetky kvantifikátory sú vpredu a sú globálne, t.j. vzťahujú sa na celý zvyšok formuly.
2. Telo formuly dáme do disjunktívnej normálnej formy (DNF).
3. Zo všetkých všeobecných kvantifikátorov urobíme existenčné pomocou teorémy $\forall X f \equiv \neg(\exists X \neg f)$ (toto už nie je prenexová forma).
4. Takúto formulu už vieme v Datalogu „mechanicky“ zapísať.

2.5.1 Prenexový tvar

V nasledujúcej časti pomocou definícií a viet matematickej logiky ukážeme, že prvý krok uvedeného algoritmu je realizovateľný pre každú vstupnú formulu, t.j. ku každej formule predikátovej logiky vieme zostrojiť ekvivalentnú formulu v prenexovom tvare [1].

Definícia 2.5.1 *Hovoríme, že formula A je v prenexovom tvare, ak má tvar:*

$$(Q_1 x_1)(Q_2 x_2)...(Q_n x_n)B$$

kde $n \geq 0$ a pre každé $i \in \{1, ..., n\}$ je $Q_i \in \{\forall, \exists\}$, B je otvorená formula a $x_1, ..., x_n$ sú navzájom rôzne premenné. B sa nazýva otvorené jadro formuly A a postupnosť kvantifikátorov, ktoré jej prechádzajú, je prefix.

Každú formulu predikátovej logiky je možné transformovať do prenexového tvaru.

Veta 2.5.1 *Ku každej formule A predikátovej logiky môžeme zostrojiť formulu A' v prenexovom tvare takú, že $\vdash A \leftrightarrow A'$.*

Ideou transformácie formuly do prenexového tvaru [1] je aplikácia prenexových operácií ("vyťahovaním kvantifikátorov pred zátvorku").

Označenie 2.5.1 *Ak Q označuje všeobecný kvantifikátor, tak $\overline{Q}$ označuje existenčný a naopak.*

Prenexné operácie:

1. *podformulu B nahraď nejakým jej variantom B' (premenovanie premenných)*
2. *podformulu $\neg(Qx)B$ nahraď formulou $(\overline{Q}x)\neg B$*
3. *ak sa premenná x nevyskytuje vo formule B, podformulu $B \rightarrow ((Qx)C)$ nahraď formulou $(Qx)(B \rightarrow C)$ (inak premenuj premennú x na premennú, ktorá sa v B nevyskytuje)*
4. *ak sa premenná x nevyskytuje vo formule C, podformulu $((Qx)B) \rightarrow C$ nahraď formulou $(\overline{Q}x)(B \rightarrow C)$ (inak premenuj premennú x na premennú, ktorá sa v B nevyskytuje)*
5. *ak symbol $\bigcirc$ zastupuje symbol $\wedge$ alebo $\vee$, tak podformulu $B \bigcirc (Qx)C$ nahraď formulou $(Qx)(B \bigcirc C)$*

Jadrom dôkazu vyššie uvedenej vety je nasledujúce tvrdenie[1]:

Lema 2.5.1 *Pre ľubovoľné formuly B,C a každú premennú x platí:*

1. $\vdash (\overline{Q}x)\neg B \leftrightarrow \neg(Qx)B$

2. $\vdash (Qx)(B \leftarrow C) \leftrightarrow (B \leftarrow (Qx)C)$, *ak sa* x *nevyskytuje v* B

3. $\vdash (\overline{Q}x)(B \leftarrow C) \leftrightarrow ((Qx)B \leftarrow C)$, *ak sa* x *nevyskytuje v* C

4. $\vdash (Qx)(B \circ C) \leftrightarrow ((Qx)B \circ C)$, $(\circ \in \{\wedge, \vee\})$

Celý dôkaz uvedenej vety je možné nájsť v [1].

2.5.2 DNS—disjunktívny normálny tvar

Obdobne sa dá ukázať, že aj druhý krok uvedeného algoritmu je realizovateľný—k ľubovoľnej formule vieme nájsť jej ekvivalentnú formulu v disjunktívnom normálnom tvare [1].

Veta 2.5.2 *Každá formula* A *výrokovej logiky je ekvivalentná istej formule tvaru*

$$A_1 \wedge A_2 \wedge ... \wedge A_n$$

kde každá z formúl $A_i, i \in \{1, n\}$ *je disjunkciou literálov. Tento tvar nazývame konjunktívny normálny tvar. Formula* A *je taktiež ekvivalentná istej formule tvaru*

$$B_1 \vee B_2 \vee ... \vee B_n$$

kde každá z formúl $A_i, i \in \{1, n\}$ *je konjunkciou literálov. Tento tvar nazývame disjunktívny normálny tvar.*

Dôkaz. matematickou indukciou vzhľadom na štruktúru formuly A

1. báza indukcie: ak A je atomická formula, tak je automaticky v konjunktívnom aj disjunktívnom normálnom tvare

2. indukčný krok:

- nech $A \equiv \neg B$

  Podľa IP existujú pre B formuly B_k (resp. B_d) v KNF (resp. DNF). Keď tieto formuly znegujeme dostaneme formuly ekvivalentné s A v DNF (resp. KNF).

- nech $A \equiv (B \rightarrow C)$

  Platí $\vdash (B \rightarrow C) \leftrightarrow (\neg B \vee C)$. Ďalej podľa IP vieme, že $\vdash B \leftrightarrow B_d$ a $\vdash C \leftrightarrow C_k$. Dosadením $B := B_d$, $C := C_k$ do formuly $\neg B \vee C$ sa B_d po znegovaní zmení na B'_k a využitím distributivity konjunkcií a disjunkcií môžeme formulu upraviť na DNF.

  Ak použijeme zvyšné dve tvrdenia IP a dosadíme $B := B_k$, $C := C_d$ po znegovaní dostaneme priamo formulu $B'_d \vee C_d$ v DNF. ❑

2.5.3 Vytváranie existenčných kvantifikátorov

Tretí krok algoritmu nevyžaduje dodatočné matematické základy, aby sme ukázali jeho realizovateľnosť—každý výskyt všeobecného kvantifikátora v prefixe formuly mechanicky nahradíme príslušným ekvivalentom podľa uvedenej teorémy $\forall X f \equiv \neg \exists X \neg f$.

Príklad:

$$\forall F \forall V \forall C \forall L(objednavky(jozo, V) \wedge dodava(F, V, C, L) \Rightarrow dodava\_nieco\_jozovi(F))$$

$$\Leftrightarrow$$

$$\neg(\exists F \neg(\neg(\exists V \neg(\neg(\exists C \neg(\neg(\exists L \neg((objednavky(jozo, V) \wedge dodava(F, V, C, L) \Rightarrow$$

$$dodava\_nieco\_jozovi(F)))))))))$$

$$\Leftrightarrow$$

$$\neg(\exists F \exists V \exists C \exists L \neg((objednavky(jozo, V) \wedge dodava(F, V, C, L) \Rightarrow dodava\_nieco\_jozovi(F))))$$

2.5.4 Mechanický prepis do Datalogu

V tomto kroku algoritmu je vstupná formula upravená do tvaru, kde všetky kvantifikátory sú existenčné a nachádzajú sa len v prefixe formuly a zároveň samotné telo formuly je v DNF—neobsahuje žiadne implikácie.

Datalog má jeden technický problém, ktorý spočíva v tom, že môžeme negovať len jednoduchý podcieľ (nie konjunkciu). Teda pre ľubovoľný podvýraz, ktorý je negovaný a obsahuje ľubovoľnú konjunkciu atómov, musíme vytvoriť pomocný predikát a pracovať s ním. Aby sme sa vyhli takýmto, podľa môjho názoru zbytočným komplikáciám, skôr ako pristúpime k samotnému mechanickému prepisu formuly do Datalogu, upravíme formulu tak, že rozdistribuujeme (prebubláme) všetky negácie čo najnižšie k samotným predikátom. Takto sa vyhneme tomu, že pri stromovitej predstave reprezentácie formuly budeme mať negáciu na ľubovoľnej inej pozícii ako bezprostredne nad listom. Vďaka tomuto kroku nebudeme musieť pri negovaných konjunkciách atómov v tele formuly vytvárať pomocné predikáty, úplnému vynechaniu vytvárania pomocných predikátov sa však nevyhneme.

Príklad mechanického prepisu:

Počiatočná formula:

$$\forall V \exists T \exists G \neg(((objednavky(jozo, V) \wedge \forall V(dodava(F, T, V, G)))))$$

Upravená formula po troch krokoch algoritmu—pripravená na mechanický prepis:

$\neg(\exists V \neg(\exists T \exists G \neg(\exists V2(objednavky(jozo, V) \wedge dodava(F, V2, T, G)))))$

Mechanický prepis tejto formuly:

$f(F, V, T, G) :- (objednavky(jozo, V) \wedge dodava(F, V2, T, G))$
$g(F, V) :- \neg f(F, V, T, G)$
$h(F) :- \neg g(F, V)$
$answer :- \neg h(F)$

Mechanický prepis takejto upravenej formuly pozostáva z dvoch krokov.

Prvým krokom je spracovanie tela formuly zapísanej v DNF. Ako vieme, v Datalogu sa disjunkcia zapisuje dvoma pravidlami, keď do výsledného predikátu najprv priradíme ľavý a potom pravý operand. Napríklad formulu

$$A \vee B$$

by sme do Datalogovského programu, pričom výsledný predikát sa nazýva answer, prepísali týmto spôsobom:

$$answer := A$$

$$answer := B$$

Druhým krokom mechanického prepisu je práve spomínané vytváranie pomocných predikátov, ktorému sa nevyhneme pri výskyte negácie v prefixe formuly.
Príklad:

$$\exists A \neg(\exists B \exists C \neg(\exists D \exists E(p(X, Y) \wedge q(A, B, C, D, E))))$$

Takúto formulu by sme do Datalogovského programu, pričom výsledný predikát je Answer(X,Y), prepísali nasledovne:

$f(A,B,C,X,Y) := p(X,Y),\ q(A,B,C,D,E)$

$g(A,X,Y) := not\ f(A,B,C,X,Y)$

$answer(X,Y) := not\ g(A,X,Y)$

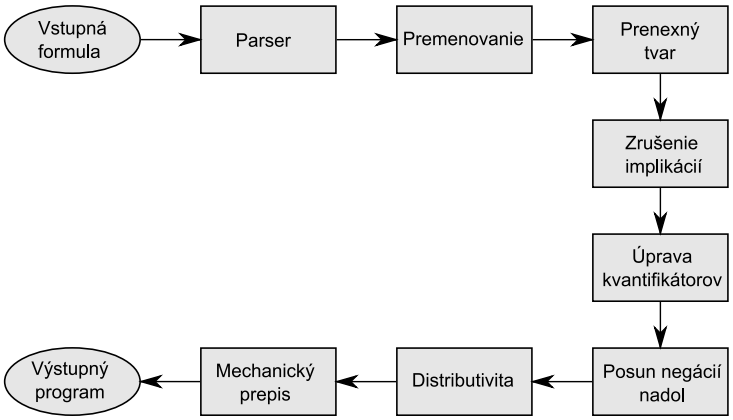
3 Implementácia

V tejto časti popíšeme implementáciu uvedeného algoritmu na preklad logických formúl do Datalogu, ktorá je hlavným cieľom tejto práce. Ako programovací jazyk na realizáciu sme si zvolili Javu 1.6, prostredie na vývoj aplikácie sme si vybrali NetBeans IDE 6.0. Projekt pri spustení vyžaduje dva argumenty. Prvým je názov vstupného súboru, ktorý obsahuje formulu relačného kalkulu zapísanú syntaxou TEXu, druhým je názov výstupného súboru, do ktorého sa uloží preložený datalogovský program v rovnakom formáte.

3.1 Návrh

Implementácia celej tejto práce je vcelku priamočiara a dá sa rozdeliť do 2 častí. Prvou časťou je načítanie a následné spracovanie—rozparsovanie vstupu, druhou časťou spomínaná séria "transformácií" na vstupnej formule podľa krokov uvedeného algoritmu. Transformácie sú množinou konkrétnych algoritmov, ktorých implementácia si nevyžaduje žiadnu zvláštnu analýzu. Otázkou ostáva spôsob reprezentácie vstupnej formuly. Prirodzená je reprezentácia pomocou stromu. Séria transformácií bude následne vďaka referencii na koreň spracovávať príslušný strom, postupne ho transformovať až k výstupu, ktorým bude samotný Datalogovský program.

Triedy samotného projektu sú rozdelené do dvoch balíkov (Packages) a síce—`data` a `transformacie`. V balíku `transformacie` sú implementované všetky kroky algoritmu na preklad formúl a v balíku `data` je všetka ostatná funkcionalita ohľadom prvej časti implementácie—práca so vstupom, formulárom, triedy pre prácu so stromom, repository pomocných funkcií a premenných.



Obrázok 1: Názorná schéma procesu prekladu pri realizovanej implementácii

3.2 Package `data`

Tento package obsahuje nasledovné triedy:

- `Prekladac`
- `Formular`
- `Parser`
- `Uzol`
- `UzolOperacia`
- `UzolOperand`
- `UzolUnarny`
- `Utils`

3.2.1 Trieda Prekladac

Trieda `Prekladac` obahuje metódu `main`, ktorá je spúšťačom celého projektu. Vytvára postupne inštancie objektov `Formular` a `Parser`, ktoré spra-

covávajú vstup a následne inštancie všetkých ostatných tried z balíka `transformacie`.

3.2.2 Trieda `Uzol`

Je to abstraktná trieda, ktorá definuje základné atribúty a metódy pre zdedené triedy. Pri stromovitej štruktúre, v našom prípade sme sa rozhodli pre binárny strom, ktorý úplne postačuje našim požiadavkám, sa v každom uzle stromu uchovávajú referencie na ľavého a pravého syna a samozrejme na otca. Pre prácu s nimi sú vyhradené metódy `get` a `set`, ktoré sú dodefinované pre konkrétne typy uzlov. Trieda Uzol:

```
public abstract class Uzol
{
    public static Formular f;
    public Uzol lavy = null;
    public Uzol pravy = null;
    public Uzol otec = null;
    public int visit;

    public Uzol(Uzol o)
    {
        this.otec = o;
        this.visit = 0;
    }

    public abstract Uzol add(Uzol u);
    public abstract void vypis();
    public abstract void setOtec(Uzol u);
```

```
    public abstract Uzol getOtec();
    public abstract Uzol getLavySyn();
    public abstract Uzol getPravySyn();
    public abstract void setLavySyn(Uzol u);
    public abstract void setPravySyn(Uzol u);
}
```

3.2.3 `UzolOperacia`

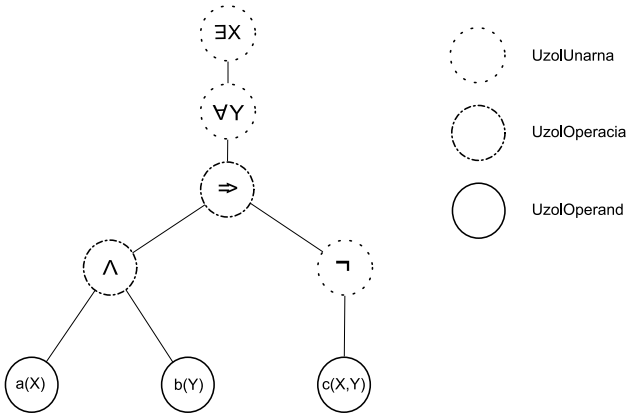
Tento uzol, ako už názov hovorí, slúži na reprezentáciu binárnych operácií v strome. Atribút *character sign* predstavuje znak operácie, môže nadobúdať jeden zo symbolov:

- $\wedge$ - `konjunkcia`
- $\vee$ - `disjunkcia`
- $\Rightarrow$ - `implikácia`

Parametre konštruktora tejto triedy sú referencia na otca a symbol operácie. Synom tohto uzla môže byť ľubovoľný iný uzol. Metóda `vypis` rekurzívne vypíše celý podstrom s vrcholom v danej binárnej operácii.

3.2.4 `UzolOperand`

Táto trieda slúži na reprezentáciu predikátov a aritmetických výrazov v strome. Konštruktor, ktorého vstupnými argumentami sú referencia na otca a reťazec znakov, rozozná, či ide o predikát alebo o aritmetický výraz a na základe toho vstupný reťazec rozparsuje. Inštancie tejto triedy predstavujú listy v strome, táto trieda teda nemá implementovanú metódu `add`. Keďže tento uzol nevytvára koreň žiadnemu podstromu, metóda `vypis` vypíše iba



Obrázok 2: Reprezentácia formuly $\exists X \forall Y((a(X) \wedge b(Y)) \Rightarrow \neg c(X,Y))$ v strome

uložený operand, až na možnosť premenovania premenných, v takom istom tvare, v akom bol daný predikát (aritmetický výraz) zadaný.

3.2.5 `UzolUnarna`

Táto trieda reprezentuje v strome unárnu operáciu—negáciu, ale okrem toho, aj keď názov triedy je `UzolUnarna`, reprezentuje aj dvojicu kvantifikátorov—existenčný a všeobecný. Kvantifikátory majú s unárnou operáciou spoločnú vlastnosť, vďaka ktorej na ich reprezentáciu stačí jedna trieda—na kvantifikátor sa vieme pozerať ako na "unárnu operáciu", keďže podobne ako negácia, má iba jedného syna. Metóda `add` v tejto triede ignoruje premennú `pravy` pravého syna a pracuje iba s premennou `lavy` ľavého syna, do ktorého ukladá syna tohto uzla. Konštruktor tejto triedy, ktorého vstupy sú referencia na otca a reťazec, rozpozná, či sa jedná o negáciu alebo kvantifikátor a na základe toho vstupný reťazec rozparsuje. Metóda `vypis` rekurzívne vypíše a korektne uzátvorkuje celý podstrom s vrcholom v danom uzle triedy `UzolUnarna`.

3.2.6 `Parser`

Očakávaným vstupom programu je formula tvaru

$$\forall vysledne Premenne(lubovolna FormulaRelacneho Kalkulu) \Rightarrow vysledny Predikat(vysledne Premenne)$$

Vstupná formula nie je klasický dotaz, kde výstupom je množina n-tíc všetkých voľných premenných, ktoré po dosadení do dotazu vrátia *TRUE*. Výstupom nášho programu je stále Datalogovský program definujúci nový predikát *vysledny Predikat(vysledne Premenne)*.

Parser rozparsuje uvedenú vstupnú formulu, pričom zaujímavá časť, ktorej sa týka spomínaný algoritmus prekladu, je časť *lubovolnaFormulaRelacnehoKalkulu* (zvyšok—postfix a prefix—sa použije až v poslednej časti pri mechanickom prepise). Nasleduje rekurzívne rozparsovanie tejto časti formuly. V každom kroku algoritmu nájdeme hlavnú operáciu, ktorá sa v danej podformule vyskytuje. Hlavnou operáciou rozumieme takú operáciu, ktorá má na svojej pozícii vo vstupnom reťazci najmenší počet otvorených zátvoriek. Do stromu sa ešte pred vložením konkrétnej nájdenej operácie v prípade výskytu vložia kvantifikátory a negácia, ktoré zodpovedajú danej operácii. Po vložení operácie do stromu sa rekurzívne spracuje ľavá časť podformuly, prvý operand danej operácie a následne pravá časť podformuly, druhý operand operácie.

3.3 Package `transformacie`

Tento package obsahuje nasledovné triedy:

- `Premenovanie`
- `PrenexovyTvar`
- `ZrusenieImplikacii`

- `UpravaKvantifikatorov`
- `PosunNegaciiNadol`
- `Distributivita`
- `MechanickyPrepis`

3.3.1 Trieda Premenovanie

Táto transformácia doteraz nebola spomenutá, ale jedná sa o viac-menej formálnu záležitosť. Ak máme formulu

$$\exists X \forall Y (\forall X (a(X) \wedge b(Y)) \Rightarrow \neg c(X, Y))$$

tak vidíme, že premenná X v predikáte a je v pôsobnosti dvoch kvantifikátorov a zároveň tá istá premenná X sa vyskytuje aj v predikáte C. Keďže sa jedná o dve rozdielne premenné, musíme ich premenovať, aby sa dali odlíšiť. Metóda *runPremenovanie* prehľadáva strom do hĺbky a pamätá si všetky premenné, ktoré boli kvantifikované niekde vyššie. Pri spomínanej formule, pri prehľadávaní stromu zhora nadol, sme najprv narazili na kvantifikátor $\exists X$ a neskôr na $\forall X$. V celom ďalšom podstrome, ktorého koreňom je uzol $\forall X$, musíme premennú X premenovať, keďže je to iná premenná X ako tá, ktorá bola kvantifikovaná $\exists X$ na začiatku formuly.

Ku každej premennej si v dvoch premenných typu `LinkedList` pamätáme aktuálny index, ktorý priradíme premennej pri jej výskyte v strome a maximálny index pre danú premennú, ktorý sa zatiaľ v strome vyskytol. Premenovanie realizujeme postupným pridávaním indexov—prirodzených čísel ($0 \notin \mathbf{N}$) k označeniu premennej.

3.3.2 Trieda `PrenexovyTvar`

Funkcionalita tejto triedy je obsiahnutá v metóde `runPrenexovyTvar`. Táto metóda opäť rekurzívne prehľadáva do hĺbky strom, pričom ju zaujímajú iba uzly typu `UzolOperacia`. Ak na takýto uzol narazí, tak zistí, či jej bezprostredne ľavý alebo pravý syn nie je kvantifikátor—uzol inštancie `UzolUnarny`, ktorého metóda `getSign` vracia $\forall$ alebo $\exists$. Ak áno, tak podľa týchto pravidiel

- $(\forall X f) \wedge g \equiv \forall X (f \wedge g)$
- $(\forall X f) \vee g \equiv \forall X (f \vee g)$
- $(\exists X f) \wedge g \equiv \exists X (f \wedge g)$
- $(\exists X f) \vee g \equiv \exists X (f \vee g)$
- $(\forall X f) \Rightarrow g \equiv \exists X (f \Rightarrow g)$
- $(\exists X f) \Rightarrow g \equiv \forall X (f \Rightarrow g)$
- $f \Rightarrow (\forall X g) \equiv \forall X (f \Rightarrow g)$
- $f \Rightarrow (\exists X g) \equiv \exists X (f \Rightarrow g)$

posunie daný kvantifikátor o úroveň vyššie, zjednodušene povedané—posunie problém nad seba. Keďže daný strom prehľadávame do hĺbky a pri udalosti, ktorú sme pred chvíľou popísali, sa kvantifikátor posunie o úroveň vyššie, v danom prechode stromu už na tento kvantifikátor nenarazíme. Takto by sa nám ho nepodarilo prebublať až do koreňa stromu. Túto metódu budeme teda spúšťať dovtedy, kým sa na strome ešte budú vykonávať nejaké zmeny, inak povedané—kým dve po sebe idúce volania metódy `runPrenexovyTvar`

nebudú mať rovnaký výstup. Takto budeme mať zaručené, že nikde v strome, okrem jeho koreňa, nenájdeme operáciu, ktorej jeden zo synov by bol kvantifikátor.

3.3.3 Trieda `ZrusenieImplikacii`

Táto trieda sa stará o prvú časť druhého kroku algoritmu, ktorým je prepis formuly do DNF. Metóda tejto triedy—`runZrusenieImplikacii`—pozmení vstupný strom tak, že odstráni všetky implikácie, pričom sa nezmení význam stromu a zostane ekvivalentný s pôvodným. Táto vlastnosť je zaručená platnosťou už spomínanej teorémy $f \Rightarrow g \equiv \neg f \vee g$. Algoritmus si opäť všíma iba uzly typu `UzolOperacia`, ak sa jedná o uzol s operáciou $\Rightarrow$ tak operáciu v danom uzle zmení na $\vee$. Následne sa pozrie na ľavého syna, či nie je inštanciou triedy `UzolUnarny`. V tomto kroku je dôležité uvedomiť si, že ak ľavý syn je inštanciou objektu `UzolUnarny`, tak je to negácia. Predchádzajúci krok algoritmu nám totiž zaručil, že všetky kvantifikátory sa už nachádzajú v koreni stromu. Zisťovanie, či ľavý syn nie je negácia, sa môže zdať zbytočné, možno jednoduchšie by sa javilo automatické vloženie negácie ako ľavého syna operácie, kde sme objavili implikáciu. V celej aplikácii sa však snažíme vyhýbať dvojitým negáciám, ktoré sa za správneho chodu programu neobjavia v žiadnej fáze úpravy stromu. Preto, ak ľavý syn uzla `UzolOperacia` je negácia, nevložíme ďalšiu, ale odstránime pôvodnú, čo je vďaka teoréme o dvojitej negácii $\neg\neg f \equiv f$ ekvivaletná operácia.

Tento algoritmus neposúva problém vyššie, tak ako to bolo v predchádzajúcom kroku algoritmu a jediným prechodom stromu dostávame výsledný, ekvivalentný strom bez implikácií.

3.3.4 Trieda `UpravaKvantifikatorov`

V tejto fáze máme formulu upravenú v prenexovom tvare, so všetkými kvantifikátormi na začiatku. Tretí krok algoritmu požaduje zo všetkých všeobecných kvantifikátorov urobiť existenčné. Metóda `runUpravaKvantifikatorov`, ktorá riadi tento proces nemusí prehľadávať celý strom, vykonáva sa dovtedy, kým nenarazí na uzol typu `UzolOperacia`. Je to vďaka vlastnosti, že daný strom je v prenexom tvare a teda nikde nižšie v strome sa už žiaden kvantifikátor nenachádza.

Daná metóda prehľadáva strom od koreňa, pričom ju zaujímajú iba uzly typu `UzolUnarny`, konkrétne uzly reprezentujúce všeobecný kvantifikátor. Ak na neho narazí, aplikuje teorému $\forall X f \equiv \neg(\exists X \neg f)$. Zmení sa typ kvantifikátora metódou `setUnarna(Utils.maly)` zo všeobecného (veľkého) na existenčný (malý). Čo sa týka negácií, ktoré potrebujeme pridať, aby strom zostal ekvivalentný, opäť by sme mohli jednoducho pridať jednu negáciu nad daný uzol—na pozíciu otca a druhú negáciu pod daný uzol—na pozíciu ľavého syna. Aby sme sa však opäť vyhli výskytu dvojitých negácií, algoritmus pred vložením nového uzla negácie do stromu overí, či sa na danom mieste už nevyskytuje negácia. Ak áno, tak ju odstráni, a iba v prípade, že sa tam negácia nevyskytuje, tak vytvorí nový uzol negácie a korektne ho do stromu pripojí.

3.3.5 Trieda `PosunNegaciiNadol`

Metóda `runPosunNegaciiNadol` tejto triedy je len akýmsi zlepšovákom uvedeného algoritmu. Keďže v Datalogu môžeme negovať len jednoduchý podcieľ (nie konjunkciu), aby sme sa vyhli zbytočnému vytváraniu pomocných predikátov, upravíme formulu tak, že rozdistribuujeme (prebubláme) všetky negácie čo najnižšie k samotným predikátom. Takto upravený strom

nebude mať negáciu na ľubovoľnej inej pozícii ako bezprostredne nad listom.

Metóda `runPosunNegaciiNadol` bude rekurzívne prehľadávať vstupný strom do hĺbky, pričom sa bude správať nasledovne

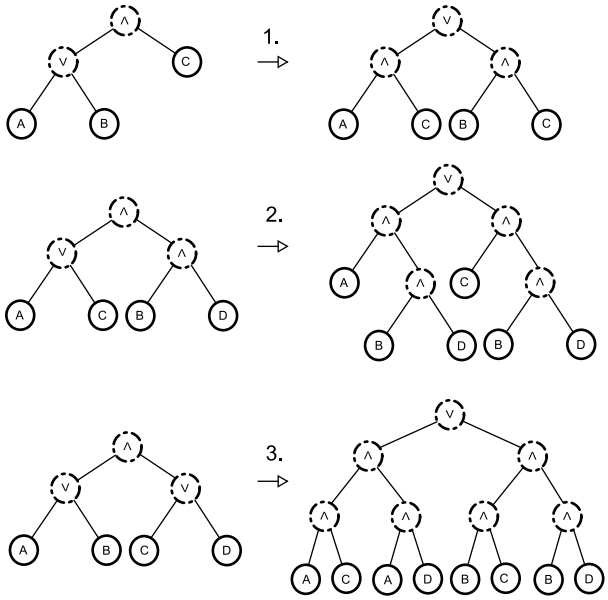
- ak narazí na negáciu pred uzlom `UzolOperacia`, zruší ju a neguje svojich dvoch synov tak, aby sa vyhla dvojitým negáciám,
- ak narazí na negáciu pred uzlom `UzolOperand`, nespraví nič, tento stav vyhovuje,
- ak narazí na iný uzol ako je negácia, rekurzívne sa zavolá na svojich potomkov.

3.3.6 Trieda `Distributivita`

Stromovitá reprezentácia formuly upravenej v DNF vyzerá tak, že ak niektorý z uzlov stromu zodpovedá operácii konjunkcia, tak v podstrome, ktorého koreňom je daný uzol sa nevyskytuje uzol zodpovedajúci operácii disjunkcia. Zjednodušene povedané, že potomkovia tohto uzla sú buď uzly reprezentujúce operáciu konjunkcia alebo sú to listy stromu—uzly typu `UzolOperand`.

Metóda `runDistributivita` tejto triedy nám už sčasti upravenú formulu definitívne upraví do DNF. Strom prehľadáva do hĺbky a pri nájdení uzla reprezentujúceho konjunkciu zistí, či niektorý z jeho synov nie je disjunkcia. Môže nastať 6 prípadov (resp. 3, ak neberieme do úvahy navzájom symetrické prípady), ktoré sú bližšie znázornené na obrázku:

- ľavý syn je disjunkcia, pravý syn je negácia alebo operand (1. časť obrázku),
- ľavý syn je disjunkcia, pravý syn je konjunkcia (2. časť obrázku),



Obrázok 3: Ukážka distributivity

- ľavý aj pravý syn sú disjunkcie (3. časť obrázku).

V každom z týchto prípadov algoritmus pomocou implementovaných pomocných metód `zduplikuj` a `getZduplikovanyUzol`, ktoré slúžia na duplikáciu podstromu, upraví pomocou distributatívnych zákonov daný podstrom presne tak, ako je to znázornené na obrázku. Výsledný strom predstavuje formulu v DNF.

3.3.7 Trieda `MechanickyPrepis`

Posledná transformácia, štvrtý krok algoritmu, je implementovaný v tejto triede `MechanickyPrepis`. Prvým krokom je prepis jadra formuly (všetko okrem prefixu). Keďže táto časť je v DNF, počet pravidiel, ktoré nám vzniknú, bude počet disjunkcií v jadre formuly plus jedna. Každé pravidlo bude jeden z operandov operácie disjunkcia. Ak sa v prefixovej časti formuly nenachádza žiadna negácia, tak, ak za hlavu daných pravidiel použijeme

$vysledny Predikat(vysledne Premenne)$, dostávame finálnu podobu korektného programu v Datalogu, zodpovedajúceho vstupnej logickej formule.

Ak sa v prefixovej časti vyskytujú negácie, musíme použitím pomocných predikátov dotvoriť to, v čom nás Datalog obmedzuje—nemožnosť negovania zložených podcieľov. Počet pomocných predikátov (pomenované sú písmenami *f,g,h,...*) je rovný počtu negácií v prefixe formuly. Algoritmus postupne spracováva podformulu sprava doľava, pričom stále zoberie časť formuly po nasledujúcu negáciu. V každom z týchto krokov vytvára nový pomocný predikát, preto je výsledný počet pomocných predikátov rovný počtu negácií v prefixe. Premenné týchto pomocných predikátov sú všetky *vysledne Premenne* plus všetky premenné neobsiahnuté vo vybranej podformule, teda kvantifikované premenné naľavo od vybranej negácie. Pri spracovaní poslednej časti, keď ako podformulu vezmeme celú formulu, vo výslednom pravidle hlavu pomenujeme $vysledny Predikat(vysledne Premenne)$ a dostávame finálnu verziu korektného Datalogovského programu zodpovedajúceho vstupnej logickej formule.

4 Experimenty

Na konkrétnych príkladoch si ukážeme, ako naimplementovaná aplikácia po jednotlivých krokoch spracúva danú vstupnú formulu, až do výsledného programu v Datalogu.

4.1 Každý človek, ktorý miluje všetky zvieratá, je niekým milovaný

Vstupnou formulou je:

$$\forall X((man(X) \wedge ((\forall Y(animal(Y) \Rightarrow loves(X,Y))) \Rightarrow \exists Y(loves(Y,X)))) \Rightarrow loved\_by\_someone(X))$$

Výstup aplikácie:

POVODNY

$$\forall X((man(X) \wedge ((\forall Y(animal(Y) \Rightarrow loves(X,Y))) \Rightarrow \exists Y(loves(Y,X)))) \Rightarrow loved\_by\_someone(X))$$

PARSER

$$ROOT\ (man(X) \wedge (\forall Y(animal(Y) \Rightarrow loves(X,Y)) \Rightarrow \exists Y loves(Y,X)))$$

PREMENOVANIE

$$ROOT\ (man(X) \wedge (\forall Y(animal(Y) \Rightarrow loves(X,Y)) \Rightarrow \exists Y2 loves(Y2,X)))$$

PRENEXOVY TVAR

$$ROOT\ \exists Y \exists Y2(man(X) \wedge ((animal(Y) \Rightarrow loves(X,Y)) \Rightarrow loves(Y2,X)))$$

ZRUSENIE IMPLIKACII

$$ROOT\ \exists Y \exists Y2(man(X) \wedge (\neg((\neg(animal(Y)) \vee loves(X,Y))) \vee loves(Y2,X)))$$

UPRAVA KVANTIFIKATOROV

$$ROOT\ \exists Y \exists Y2(man(X) \wedge (\neg((\neg(animal(Y)) \vee loves(X,Y))) \vee loves(Y2,X)))$$

POSUN NEGACII NADOL

$$ROOT\ \exists Y \exists Y2(man(X) \wedge ((animal(Y) \wedge \neg(loves(X,Y))) \vee loves(Y2,X)))$$

$DISTRIBUTIVITA$

$ROOT\ \exists Y \exists Y2((man(X) \wedge (animal(Y) \wedge \neg(loves(X,Y)))) \vee (man(X) \wedge loves(Y2,X)))$

$MECHANICKY\ PREPIS$

$loved\_by\_someone(X) : -(man(X) \wedge (animal(Y) \wedge \neg(loves(X,Y))))$

$loved\_by\_someone(X) : -(man(X) \wedge loves(Y2,X))$

4.2 Dvojice krčma a pijan, pre ktoré platí, že pijan v danej krčme nemá rád žiaden alkohol

Vstupnou formulou je:

$$\forall P \forall K(((pijani(P) \wedge krcmy(K)) \wedge \neg(\exists A(capuje(K,A) \wedge lubi(P,A)))) \Rightarrow nelubi\_nic(P,K))$$

Výstup aplikácie:

$POVODNY$

$\forall P \forall K(((pijani(P) \wedge krcmy(K)) \wedge \neg(\exists A(capuje(K,A) \wedge lubi(P,A)))) \Rightarrow nelubi\_nic(P,K))$

$PARSER$

$ROOT((pijani(P) \wedge krcmy(K)) \wedge \neg(\exists A(capuje(K,A) \wedge lubi(P,A))))$

$PREMENOVANIE$

$ROOT((pijani(P) \wedge krcmy(K)) \wedge \neg(\exists A(capuje(K,A) \wedge lubi(P,A))))$

$PRENEXOVY\ TVAR$

$ROOT\forall A((pijani(P) \wedge krcmy(K)) \wedge \neg((capuje(K,A) \wedge lubi(P,A))))$

$ZRUSENIE\ IMPLIKACII$

$ROOT\neg(\forall A\neg(((pijani(P) \wedge krcmy(K)) \wedge \neg((capuje(K,A) \wedge lubi(P,A))))))$

$UPRAVA\ KVANTIFIKATOROV$

$ROOT\neg(\exists A\neg(((pijani(P) \vee krcmy(K)) \wedge \neg((capuje(K,A) \wedge lubi(P,A))))))$

POSUN NEGACII NADOL

$ROOT\neg(\exists A((\neg(pijani(P)) \vee \neg(krcmy(K))) \vee (capuje(K,A) \wedge lubi(P,A))))$

DISTRIBUTIVITA

$ROOT\neg(\exists A((\neg(pijani(P)) \vee \neg(krcmy(K))) \vee (capuje(K,A) \wedge lubi(P,A))))$

MECHANICKY PREPIS

$f(P,K) :- \neg(pijani(P))$

$f(P,K) :- \neg(krcmy(K))$

$f(P,K) :- (capuje(K,A) \wedge lubi(P,A))$

$nelubi\_nic(P,K) :- \neg f(P,K)$

Vygenerovaný program v Datalogu je ekvivalentný vstupnej formule relačného kalkulu, ale nie je bezpečný. Dôvodom je napríklad prvý riadok uvedeného programu, kde sa premenná P vyskytuje v hlave pravidla, ale nevyskytuje sa v žiadnom pozitívnom kontexte. Ako ukázala analýza tohto problému, všeobecným riešením tohto nedostatku by bolo pridanie ďalšej fázy "prebublávanie disjunktov nahor", ktorú by sme zaradili medzi fázu transformácie kvantifikátorov na existenčné a fázu "mechanického prepisu" do Datalogu.

Pri tejto transformácii máme zaručené, že vstupná formula je už v prenexnom tvare a DNF zároveň. Cieľom tejto fázy, ako už názov hovorí, je "prebublanie disjunktov" tak vysoko v stromovej štruktúre formuly, ako to len ide. Daný disjunkt budeme posúvať, prebublávať tak vysoko, aby nebol navyše v pôsobnosti žiadneho kvantifikátora, ktorý neviaže nejakú premennú z daného disjunktu. Prebublávanie sa teda zastaví, ak najbližší kvantifikátor nad disjunktom bude viazať nejakú premennú z disjunktu alebo narazíme na koreň stromu.

Rozpracovaná formula z nášho príkladu v prenexnom tvare a DNF

$$\neg(\exists A((\neg pijani(P) \vee \neg krcmy(K) \vee (capuje(K,A) \wedge lubi(P,A)))))$$

bude po transformácii "prebublávanie disjunktov nahor" vyzerať takto

$$pijani(P) \wedge krcmy(K) \wedge \neg\exists A(capuje(K,A) \wedge lubi(P,A)))))$$

Kedže upravovaná formula bola pred transformáciou v DNF a prebublávame celý disjunkt, tak sa nám posledná disjunkcia v rozpracovanej formule pôsobením negácie zmenila na konjunkciu—prostredná konjunkcia v tomto príklade. Táto vlastnosť je vďaka DNF zaručená po každej takejto transformácii. To nám vyhovuje a výslednú formulu už len mechanicky prepíšeme použitím pomocného predikátu na negovaný zložený podcieľ $(capuje(K,A) \wedge lubi(P,A))$ takto :

$$f(P,K) :- capuje(K,A) \wedge lubi(P,A)$$

$$nelubi\_nic(P,K) :- pijani(P) \wedge krcmy(K) \wedge \neg f(P,K)$$

Tento program v Datalogu je vďaka dodaniu fázy "prebublávanie disjunktov nahor" bezpečný a ekvivalentný pôvodnej formule relačného kalkulu.

4.3 Kto dodáva *pc1* do 4 dní alebo lacnejšie, ako za 15000

Vstupnou formulou je:

$$\forall F(((\exists C \exists L1(dodava(F,pc1,C,L1) \wedge C <= 15000)) \vee (\exists L \exists C2(dodava(F,pc1,C2,L) \wedge L <= 4)))$$
$$\Rightarrow dodava\_pc1\_do4\_alebo\_do15000(F))$$

Výstup aplikácie:

$POVODNY$

$\forall F(((\exists C \exists L1(dodava(F, pc1, C, L1) \wedge C <= 15000)) \vee (\exists L \exists C2(dodava(F, pc1, C2, L) \wedge L <= 4)))$
$\Rightarrow dodava\_pc1\_do4\_alebo\_do15000(F))$

$PARSER$

$ROOT\ (\exists C \exists L1(dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee \exists L \exists C2(dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$PREMENOVANIE$

$ROOT\ (\exists C \exists L1(dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee \exists L \exists C2(dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$PRENEXOVY\ TVAR$

$ROOT\ \exists C \exists L1 \exists L \exists C2((dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee (dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$ZRUSENIE\ IMPLIKACII$

$ROOT\ \exists C \exists L1 \exists L \exists C2((dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee (dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$UPRAVA\ KVANTIFIKATOROV$

$ROOT\ \exists C \exists L1 \exists L \exists C2((dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee (dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$POSUN\ NEGACII\ NADOL$

$ROOT\ \exists C \exists L1 \exists L \exists C2((dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee (dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$DISTRIBUTIVITA$

$ROOT\ \exists C \exists L1 \exists L \exists C2((dodava(F, pc1, C, L1) \wedge (C <= 15000)) \vee (dodava(F, pc1, C2, L) \wedge (L <= 4)))$

$MECHANICKY\ PREPIS$

$dodava\_pc1\_do4\_alebo\_do15000(F) : -(dodava(F, pc1, C, L1) \wedge (C <= 15000))$

$dodava\_pc1\_do4\_alebo\_do15000(F) : -(dodava(F, pc1, C2, L) \wedge (L <= 4))$

4.4 Kto nedodáva niečo z toho, čo objednal *jozo*

Vstupnou formulou je:

$$\forall F(\exists V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg dodava(F, V, C1, L1))) \Rightarrow dodava\_jozovi(F))$$

Výstup aplikácie:

$POVODNY$

$$\forall F(\exists V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg dodava(F, V, C1, L1))) \Rightarrow dodava\_jozovi(F))$$

$PARSER$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$PREMENOVANIE$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$PRENEXOVY\ TVAR$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$ZRUSENIE\ IMPLIKACII$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$UPRAVA\ KVANTIFIKATOROV$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$POSUN\ NEGACII\ NADOL$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$DISTRIBUTIVITA$

$$ROOT\ \forall V \exists C1 \exists L1 \exists M(objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

$MECHANICKY\ PREPIS$

$$dodava\_jozovi(F) :- (objednavky(jozo, V) \wedge (dodavatel(F, M) \wedge \neg(dodava(F, V, C1, L1))))$$

4.5 Datalog a SQL

V tejto časti ešte na záver ukážeme jednoduchosť následného prekladu Datalogovského programu do SQL. Majme formulu relačného kalkulu

$$\forall F((\exists M(dodavatel(F,M) \wedge \forall V objednavky(jozo,V)) \Rightarrow \exists C \exists L dodava(F,V,C,L)) \Rightarrow$$

$$dodava\_vsetko\_jozovi(F))$$

Ekvivalentný program v Datalogu vyzerá nasledovne

$$nedodava\_nieco\_jozovi(F) : -dodavatel(F,M) \wedge objednavky(jozo,V) \wedge \neg dodava(F,V,C,L)$$

$$dodava\_vsetko\_jozovi(F) : -dodavatel(F,M) \wedge \neg nedodava\_nieco\_jozovi(F)$$

Tento Datalogovský program sa dá ľahko preložiť do SQL. Z pomocných predikátov, v našom prípade $nedodava\_nieco\_jozovi(F)$, si vytvoríme *VIEW*

$CREATE\ VIEW\ nedodava\_nieco\_jozovi\ AS$

$SELECT\ D.firma$

$FROM\ dodavatel\ D,\ objednavky\ O$

$WHERE\ O.meno = jozo\ AND\ NOT\ EXISTS\ ($

$SELECT\ *\ FROM\ dodava\ DD$

$WHERE\ DD.firma = D.firma\ AND\ DD.vyrobok = O.vyrobok)$

Druhý riadok Datalogovského programu, ktorý popisuje výsledný predikát $dodava\_vsetko\_jozovi(F)$ prepíšeme použitím pomocného *VIEW* a základnej syntaktickej konštrukcie jazyka SQL nasledovne:

$SELECT\ D.firma$

$FROM\ dodavatel\ D$

$WHERE\ NOT\ EXISTS\ ($

$SELECT\ *\ FROM\ nedodava\_nieco\_jozovi\ NNJ$

$WHERE\ NNJ.firma = D.firma)$

5 Záver

Táto práca mala za cieľ oboznámiť čitateľa s algoritmom prekladu formúl relačného kalkulu do Datalogu [2]. Popísali sme algoritmus, ktorý vstupnú formulu najprv prepíše do prenexového tvaru, telo formuly upraví do disjunktívnej normálnej formy, následne pretransformuje všetky všeobecné kvantifikátory na existenčné a na záver takúto formulu mechanicky prepíše do programu v Datalogu. Taktiež sa nám podarilo realizovať hlavný zámer tejto práce a naprogramovať implementáciu tohto algoritmu do fungujúcej aplikácie.

Aplikácia funguje presne podľa určeného algoritmu, ale v poslednej fáze testovania sme zistili, že tento algoritmus síce generuje ekvivalentné datalogovské programy k zadanej formule relačného kalkulu, ale negeneruje bezpečné datalogovské programy. Riešenie tohto problému popisujeme v časti 4.2. Toto riešenie spočíva v rozšírení algoritmu z časti 2.5 o ďalšiu fázu "prebublávanie disjunktov nahor", ktorá je zaradená medzi fázu transformácie kvantifikátorov na existenčné a fázu "mechanického prepisu" do Datalogu. Navrhnuté riešenie sa pokúsime implementovať do obhajoby tejto práce.

Veríme, že cieľ tejto práce sa nám podarilo naplniť a táto aplikácia bude nápomocná nielen študentom Databázových systémov zbehlých v danej problematike, ktorí si naučenú teóriu budú chcieť overiť aj v praxi, ale aj bežnému čitateľovi, ktorého zaujme daná problematika a v širšom úvode, ktorý je určený práve preňho, i v samotnej implementácii, nájde veľa zaujímavého a poučného.

Ak táto práca aspoň v malej miere napomôže čitateľovi k lepšiemu pochopeniu problematiky prekladu formúl v rámci dvoch spomínaných dotazovacích jazykov, tak si myslíme, že naše úsilie, venované tejto práci, nevyšlo nazmar.

Referencie

[1] Eduard Toman, Vybrané partie z logiky, http://www.dcs.fmph.uniba.sk/texty/logika.pdf, 2005.

[2] Tomáš Plachetka, Úvod do databázových systémov—Dotazovacie jazyky, http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/DB2007/db2007_2.pdf, 2007/2008.

[3] Jaroslav Pokorný, DATALOG, http://archiv.computerworld.cz/ cwarchiv.nsf/clanky/578A5A5A9DCFFA57C12569B000520168?OpenDocument, 1997.