Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Kompilátor Datalogu s funkčnými symbolmi do relačnej algebry

Bakalárska práca

2026
Matúš Duchyňa

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Kompilátor Datalogu s funkčnými symbolmi do relačnej algebry

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. Mgr. Tomáš Plachetka, Dr.

Bratislava, 2026
Matúš Duchyňa

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Matúš Duchyňa
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Kompilátor Datalogu s funkčnými symbolmi do relačnej algebry
Compiler of Datalog with function symbols to relational algebra

Anotácia: Definovať rozšírenie Datalogu o funkčné symboly. Skonštruovať vhodnú formálnu gramatiku v syntaxi ANTLR. Implementovať kompilátor do relačnej algebry. Integrovať s kompilátorom relačnej algebry. Overiť funkčnosť na vhodných vstupoch.

Vedúci: doc. Mgr. Tomáš Plachetka, Dr.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
archív

Dátum zadania: 21.10.2025

Dátum schválenia: 27.10.2025 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné vyhlásenie: Čestne vyhlasujem, že celú bakalársku prácu na tému „Kompilátor Datalogu s funkčnými symbolmi do relačnej algebry“, vrátane všetkých jej príloh a obrázkov, som vypracoval/vypracovala samostatne, a to s použitím literatúry uvedenej v priloženom zozname.

Poďakovanie: TODO

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: datalog, relačná algebra, kompilátor, databázový systém

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Klasické moderné databázové systémy poskytujú používateľom na vytváranie dotazov zväčša deklaratívne jazyky, ako napríklad SQL. Výhodou deklaratívneho jazyka je to, že používateľ sa nemusí zaoberať samotným algoritmom, ktorý vypočíta výsledok dotazu. Súčasťou klasických databázových modelov je tzv. *optimalizér*. Tento komponent za pomoci rôznych metadát o databáze vytvorí zo SQL dotazu *vykonávací plán*, t.j. algoritmus výpočtu dotazu. Avšak optimalizér nie vždy vytvorí *optimálny* vykonávací plán. Prinútiť databázový systém používať konkrétny vykonávací plán nie je väčšinou jednoduché. Preto existujú experti, ktorí sa zaoberajú výlučne touto problematikou. Klasické postupy optimalizácie zahŕňajú používanie *hintov*, prepisovanie dotazu na iný ekvivalentný dotaz alebo explicitné nariadenie použitia konkrétneho plánu. Vykonávacie plány však väčšinou používajú veľké množstvo komplikovaných operácií s množstvom parametrov a bez kvalitnej dokumentácie. Kvôli tomu je písanie konkrétneho plánu nepraktické a neefektívne.

Náš databázový systém[1] používa ako dotazovací jazyk relačnú algebru. Konkrétne ide o relačnú algebru s piatimi operátormi a podporou funkčných symbolov. Viac o nej je uvedené v kapitole 2. Relačná algebra nám priamo umožňuje špecifikovať algoritmus materializácie dotazu. Pri jej návrhu sme sa zamerali na jednoduchosť, čo sa však odrazilo na veľkosti dotazov. *Loader*[2] z relačnej algebry do programovacieho jazyka Java a implementáciu jednotlivých funkcií v jazyku Java vytvorili kolegovia Biriukova [1] a Magát [2].

Relačná algebra je síce silným nástrojom na určenie presného algoritmu výpočtu dotazu ale pre bežného používateľa je príliš nízkoúrovňová. Tvorba zložitejších dotazov vyžaduje manuálne skladanie operátorov čo zvyšuje nečitateľnosť kódu a riziko chyby. Tu vzniká priestor pre datalog, deklaratívny dotazovací jazyk ktorý je založený na matematickej logike. Datalog umožňuje efektívne definovať zložité dotazy pomocou pravidiel, ktoré sú často omnoho stručnejšie ako ekvivalentné zápisy v SQL alebo relačnej algebre.

Cieľom mojej bakalárskej práce je pridať datalog s funkčnými symbolmi ako ďalší

[1] Pod *naším* databázovým systémom myslím experimentálny systém, na ktorom pracuje alebo pracovali môj školiteľ a kolegovia pod jeho vedením

[2] Kompilátor

dotazovací jazyk do nášho systému. Konkrétne ide o vytvorenie kompilátora z datalogu do relačnej algebry tak, aby bol použiteľný aj samostatne, bez väzby na konkrétny databázový systém. Dôležité bude spracovanie funkčných symbolov, ktoré sa v štandardnom datalogu (resp. prologu) nevyskytujú. Kompilátor musí byť schopný overiť syntaktickú a sémantickú správnosť datalogového programu a následne vygenerovať čo najoptimálnejší ekvivalentný kód v relačnej algebre.

Podobným softvérom je aj *Soufflé* [3]. Soufflé kompilátor taktiež vykonáva sémantickú kontrolu, ale následne preloží kód datalogu do tzv. *Relation Algebra Machine*. Z *RAM* sa potom vygeneruje vysoko optimalizovaný a paralelizovaný C++ kód. Naše riešenie sa síce nezameriava na rýchlosť preloženého dotazu, ale na jeho jednoduchosť. Náš datalog aj relačná algebra obsahujú iba tie najdôležitejšie komponenty, ale zároveň zanechávajú dotazovaciu silu.

Zásadným rozdielom oproti nášmu riešeniu je, že Soufflé nepodporuje funkčné symboly v podobe, ktorú by sme potrebovali. Rovnako Soufflé nepodporuje ani priame využívanie vstavaných funkcií databázy (*built-in functions*), čo obmedzuje jeho schopnosť spolupracovať s existujúcimi databázovými systémmi. Náš prístup naopak počíta so vstavanou podporou funkcií priamo v relačnej algebre a databáze, čím umožňuje vykonávať komplexné transformácie dát už počas samotného procesu materializácie.

Okrem rozdielu v cieľovej platforme (C++ vs. Java) je dôležitá aj miera abstrakcie vygenerovanej relačnej algebry. Zatiaľ čo RAM je nízkoúrovňová inštrukčná sada navrhnutá pre konkrétny stroj, naša relačná algebra si zachováva matematickú podobu a je bližšia teoretickým definíciám. To umožňuje jednoduchšiu kontrolu správnosti prekladu a prípadnú manuálnu kontrolu vygenerovaného prekladu.

Práca je rozdelená na tri časti. V kapitole 1 bližšie popisujeme datalog ako jazyk logiky a definujeme jeho rozšírenie o funkčné symboly. Pri snahe osamostatniť náš kompilátor od nášho databázového systému sme identifikovali požiadavky na relačnú algebru. Všetky zmeny[3] a ich odôvodnenia sú popísané v kapitole 2. V záverečnej kapitole 3 popisujeme samotný kompilátor, jeho jednotlivé časti vrátane gramatiky nášho datalogu a algoritmu prekladu.

[3]Oproti relačnej algebre definovanej v [1]

Kapitola 1

Datalog

1.1 Datalog ako jazyk logiky

Poznámka 1. $\top$ označuje pravdu (hodnotu *true*). $\bot$ označuje nepravdu (hodnotu *false*). ? označuje hodnotu *unknown*.

Note: Definície 1.1.1. a 1.1.2. sú asi zbytočné a nezapadajú moc do zvyšku definícií.

Definícia 1.1.1. ***n*-árna relácia** r je definovaná ako $r \subseteq D_1 \times D_2 \times \cdots \times D_n$. Jednotlivé prvky relácie r nazývame *tuples*, *záznamy* alebo *usporiadané n-tice*. Jednotlivým zložkám n-tíc hovoríme *atribúty* relácie. Množinám $D_1, D_2, \ldots, D_n$ sa hovorí *domény* (alebo *typy*) atribútov relácie r.

Definícia 1.1.2. ***n*-árny predikát** p prislúchajúci n-árnej relácii $r \subseteq D_1 \times D_2 \times \cdots \times D_n$ je funkcia $p : D_1 \times D_2 \times \cdots \times D_n \to \{\top, \bot\}$ definovaná nasledovne:

$$p(x_1, x_2, \ldots, x_n) = \top \Leftrightarrow (x_1, x_2, \ldots, x_n) \in r$$

$$p(x_1, x_2, \ldots, x_n) = \bot \Leftrightarrow (x_1, x_2, \ldots, x_n) \notin r$$

Definícia 1.1.3. Jazyk logiky pozostáva z

1. predikátových symbolov
2. symbolov pre premenné
3. symbolov pre konštanty (reprezentujú prvky z domén)
4. logických symbolov ($\wedge, \neg$)
5. symbolov všeobecných kvantifikátorov ($\forall$)
6. symbolov pre definície ($\Rightarrow$)
7. čiarok (`,`)
8. a zátvoriek (`(`, `)`).

Definícia 1.1.4. Do **formúl** patria:

1. *atomické formuly* $p(t_1, t_2, \ldots t_n)$, kde p je n-árny predikát a $t_1, t_2, \ldots t_n$ sú konštanty alebo premenné
2. ak A a B sú formuly, tak potom aj $A \wedge B$ a $\neg A$ sú formuly
3. nič iné nie je formula

Definícia 1.1.5. $A \Rightarrow B$ je **implikácia**, ak A je klauzula a B je atomická klauzula.

Definícia 1.1.6. $\forall X \iota$ sa nazýva **kvantifikovaná implikácia**, kde X je premenná a ι je implikácia. Ak je kvantifikovaná každá premenná v kvantifikovanej implikácií, nazývame ju **klauzula**.

Definícia 1.1.7. $K_1 \wedge K_2 \wedge \cdots \wedge K_n$ nazývame **teóriou**, kde $K_1, K_2, \ldots, K_n$ sú klauzuly.

Definícia 1.1.8. Ohodnotenie premenných je funkcia, ktorá priradí každej premenne konštantu. Ohodnotenie o, ktoré premennej X_i priradí konštantu k_i (pre $i \leq n$) označujeme $o(X_1|k_1, \ldots, X_n|k_n)$. Keď vo formule A resp. implikácií ι resp. teórií $K_1 \wedge \cdots \wedge K_n$ nahradíme všetky premenné podľa ohodnotenia o, výslednú formulu označujeme ako $A[o]$ resp. $\iota[o]$ resp. $K_1 \wedge \cdots \wedge K_n[o]$.

Definícia 1.1.9. Interpretácia jazyka je funkcia, ktorá každej formule priradí jednu z hodnôt $\top, \bot, ?$.

Definícia 1.1.10. Nech $\mathcal{I}$ je interpretácia jazyka, φ je formula a o je ohodnotenie premenných. Potom $\mathcal{I} \models_\top \varphi[o]$ resp. $\mathcal{I} \models_\bot \varphi[o]$ resp. $\mathcal{I} \models_? \varphi[o]$ označuje, že formula $\varphi[o]$ je pravdivá resp. neprevdivá resp. *unknown* vzhľadom na interpretáciu jazyka $\mathcal{I}$.

Pravdivosť formuly definujeme nasledovne:

1.
   $\mathcal{I} \models_\top \neg\varphi[o]$ ak $\mathcal{I} \models_\bot \varphi[o]$.
   $\mathcal{I} \models_\bot \neg\varphi[o]$ ak $\mathcal{I} \models_\top \varphi[o]$.
   $\mathcal{I} \models_? \neg\varphi[o]$ ak $\mathcal{I} \models_? \varphi[o]$.
2.
   $\mathcal{I} \models_\top \varphi[o] \wedge \phi[o]$ ak $\mathcal{I} \models_\top \varphi[o]$ a zároveň $\mathcal{I} \models_\top \phi[o]$.
   $\mathcal{I} \models_\bot \varphi[o] \wedge \phi[o]$ ak $\mathcal{I} \models_\bot \varphi[o]$ alebo $\mathcal{I} \models_\bot \phi[o]$.
   $\mathcal{I} \models_? \varphi[o] \wedge \phi[o]$ v ostatných prípadoch.

3.
$\mathcal{I} \models_\top (\varphi \Rightarrow \phi)[o]$ ak platí
   (a) $\mathcal{I} \models_\top \varphi[o]$ a zároveň $\mathcal{I} \models_\top \phi[o]$,
   (b) $\mathcal{I} \models_? \varphi[o]$ a zároveň $\mathcal{I} \models_\top \phi[o]$,
   (c) $\mathcal{I} \models_? \varphi[o]$ a zároveň $\mathcal{I} \models_? \phi[o]$,
   (d) $\mathcal{I} \models_\bot \varphi[o]$.

   $\mathcal{I} \models_\bot (\varphi \Rightarrow \phi)[o]$ ak $\mathcal{I} \models_\top \varphi[o]$ a zároveň $\mathcal{I} \models_\bot \phi[o]$.
   $\mathcal{I} \models_? (\varphi \Rightarrow \phi)[o]$ ak $\mathcal{I} \models_? \varphi[o]$ a zároveň $\mathcal{I} \models_\bot \phi[o]$.
4. Pre kvantifikovaná implikácia $\forall X \iota$ platí:
   $\mathcal{I} \models_\top \forall X \iota$ ak pre *ľubovoľnú* konštantu c platí $\mathcal{I} \models_\top \iota[X|c]$,
   $\mathcal{I} \models_\bot \forall X \iota$ ak pre *niektorú* konštantu c platí $\mathcal{I} \models_\bot \iota[X|c]$,
   inak $\mathcal{I} \models_? \forall X \iota$.

Definícia 1.1.11. Interpretáciu $\mathcal{I}$ teórie $K_1 \wedge \cdots \wedge K_n$ nazývame **model**, ak pre všetky $i \in \{1, \ldots, n\}$ platí $\mathcal{I} \models_\top K_i$.

Definícia 1.1.12. Majme teóriu $K_1 \wedge \cdots \wedge K_n$ a atomickú formulu Q. Potom teóriu $K_1 \wedge \cdots \wedge K_n \wedge Q$ nazývame **teóriu s dotazom** Q.

Definícia 1.1.13. Majme teóriu s dotazom $K_1 \wedge \cdots \wedge K_n \wedge Q$, kde $X_1, \ldots, X_k$ sú premenné v dotaze Q. **Výsledkom dotazu** (vzhľadom na model $\mathcal{I}$ teórie $K_1 \wedge \cdots \wedge K_n$) nazývame množinu usporiadaných k-tic

$$\{(c_1, c_2, \ldots, c_k) \,|\, \mathcal{I} \models_\top Q[X_1|c_1, X_2|c_2, \ldots, X_k|c_k]\}$$

.

Ak dotaz Q neobsahuje žiadnu premennú a platí $\mathcal{I} \models_\top Q$, výsledkom dotazu je $\top$, inak je výsledkom $\bot$.

Poznámka 2. Môžeme si všimnúť, že dotazom sa vieme pýtať ina otázky typu *platí dotaz* $p(K_1, \ldots, K_n)$*?* (kde $K_1, \ldots, K_n$ sú konštanty). Nevieme sa pýtať na opak (*neplatí...*).

Týmto dokážeme odpovedať na niektoré dotazy pre teórie, ktoré nemajú model. Ako príklad môžeme zobrať teóriu

$$\mathcal{T} = (\neg p() \Rightarrow q()) \wedge (\neg q() \Rightarrow p())$$

Jediná validná interpretácia $\mathcal{I}_V$ je $\mathcal{I}_V \models_? p()$ a $\mathcal{I}_V \models_? q()$ (ak by sme zobrali ľubovoľnú inú interpretáciu $\mathcal{I}$, došli by sme k sporu $\mathcal{I} \models_\top p() \wedge \mathcal{I} \not\models_\top p()$ resp. $\mathcal{I} \models_\bot p() \wedge \mathcal{I} \not\models_\bot p()$). Ak by sme sa vedeli pýtať na záporné dotazy, dostali by sme pre teórie s dotazom $\mathcal{T} \wedge p()$ a $\mathcal{T} \wedge \neg p()$ výsledok dotazu $\bot$. S našim prístupom dostaneme pre dotaz „*platí* $p()$“ $\bot$. Z čoho potom musí platiť, že $p()$ je nepravdivé.

1.2 Rozšírenie datalogu o funkčné symboly

Doteraz sme dosádzali do predikátov iba konštanty a premenné. Rozšírime jazyk logiky o tzv. funkčné symboly. Funkčný symbol f (s aritou n) reprezentuje ľubovoľnú funkciu $f : D_1 \times D_2 \times \cdots \times D_n \to D$.

Poznámka 3. Možno doplniť niečo o tom, prečo chceme funkčné symboli.

Definícia 1.2.1. Za **term** považujeme:

- ľubovoľný 0-árny funkčný symbol (ktoré považujeme za *konštanty*),
- ľubovoľnú premennú,
- ak f je n-árny funkčný symbol a $t_1, t_2, \ldots, t_n$ sú termy, potom $f(t_1, t_2, \ldots, t_n)$ je term.

Dôležité je, že term pozostáva z konečného množstva symbolov.

Rozširime definíciu jazyka logiky (1.1.3) o symboli označujúce funkčné symboly. Do predikátov potom nedopĺňame premenné a konštanty, ale termy.

Definícia 1.2.2. Ground term je ľubovoľný term, v ktorom sa nevyskytujú premenné.

V definícii pre ohodnotenie premenných (1.1.8) sa daná funkcia predefinuje na funkciu, ktorá priradí každej premennej nie konštantu, ale *ground term*. Taktiež v definícii interpretácie jazyka (1.1.10) prepíšeme 4. časť na:

> Pre kvantifikovaná implikácia $\forall X \iota$ platí:
> $\mathcal{I} \models_\top \forall X \iota$ ak pre *ľubovoľný* **ground term** c platí $\mathcal{I} \models_\top \iota[X|c]$,
> $\mathcal{I} \models_\bot \forall X \iota$ ak pre *niektorý* **ground term** c platí $\mathcal{I} \models_\bot \iota[X|c]$,
> inak $\mathcal{I} \models_? \forall X \iota$.

Ostatné definície môžu ostať tak, ako boli popísané v predošlej kapitole.

1.3 Syntax rozšíreného datalogu

Zoberme si formálny zápis relatívne jednoduchej teórie s dotazom:

$$[\forall X, Y\,(\text{parent}(X, Y) \Rightarrow \text{ancestor}(X, Y))] \wedge$$
$$[\forall X, Y, Z\,((\text{parent}(X, Z) \wedge \text{ancestor}(Z, Y)) \Rightarrow \text{ancestor}(X, Y))] \wedge$$
$$[\forall X, Y\,(\text{ancestor}(X, Y))]$$

Tento zápis sa rýchlo stáva neprehľadným a dlhým. Preto Datalog používa skrátený zápis.

Klauzula

$$\forall X_1, X_2, \ldots, X_k[(p_1(t_{1,1}, t_{1,2}, \ldots, t_{1,k_1}) \wedge \cdots \wedge p_n(t_{n,1}, t_{n,2}, \ldots, t_{n,k_n})) \Rightarrow q(t_1, t_2, \ldots, t_{k_q})],$$

kde $X_1, X_2, \ldots, X_k$ sú všetky premenné vyskytujúce sa v jednotlivých termoch, sa v skrátenom zápise zapíše ako

$$q(t_1, t_2, \ldots, t_{k_q}) \leftarrow p_1(t_{1,1}, t_{1,2}, \ldots, t_{1,k_1}), p_2(t_{2,1}, t_{2,2}, \ldots, t_{2,k_2}), \ldots, p_n(t_{n,1}, t_{n,2}, \ldots, t_{n,k_n}).$$

Teórii s dotazom $K_1 \wedge K_2 \wedge \cdots \wedge K_n \wedge Q$ zodpovedá $D_1 D_2 \ldots D_n ?Q$, kde D_i je skrátený zápis klauzuly K_i.

Skorej spomenutá teória je reprezentovaná skráteným zápisom:

$$\begin{aligned} &\text{ancestor}(X, Y) \leftarrow \text{parent}(X, Y). \\ &\text{ancestor}(X, Y) \leftarrow \text{parent}(X, Z), \text{ancestor}(Z, Y). \\ &? \, \text{ancestor}(X, Y) \end{aligned}$$

1.4 Well-founded model

Kapitola 2

Relačná algebra

2.1 Definície

Ako bolo spomenuté v úvode, v tejto časti predefinujeme relačnú algebru z [1] aby spĺňala nové požiadavky.

2.2 Rozšírenie algebry o funkčné symboly

Keďže sme rozšírili datalog o funkčné symboly, musíme rozšíriť o funkčné symboly aj relačnú algebru.

Kapitola 3

Kompilátor

3.1 ANTLR

Náš kompilátor používa generátor *parserov* **AN**other **T**ool for **L**anguage **R**ecognition. V tejto časti si priblížime ako *ANTLR* funguje a čo motivovalo zmeny v relačnej algebre.

3.2 Gramatika datalogu pre ANTLR

Program ANTLR potrebuje na vygenerovanie parseru gramatiku parsovaného jazyka.

3.3 Algoritmus prekladu

Po vygenerovaní stromu odvodenia prichádza na rad preklad do relačnej algebry.

Záver

Literatúra

[1] Diana Biriukova. Compiler of relational algebra expressions, 2025.

[2] Tomáš Magát. Todo, 2026.

[3] Soufflé Language Team. Soufflé: Program translation. `https://souffle-lang.github.io/translate`, 2024. [Online; navštívené 7. januára 2026].