Relačné a logické bázy dát

Ján Šturc

Jar, 2013

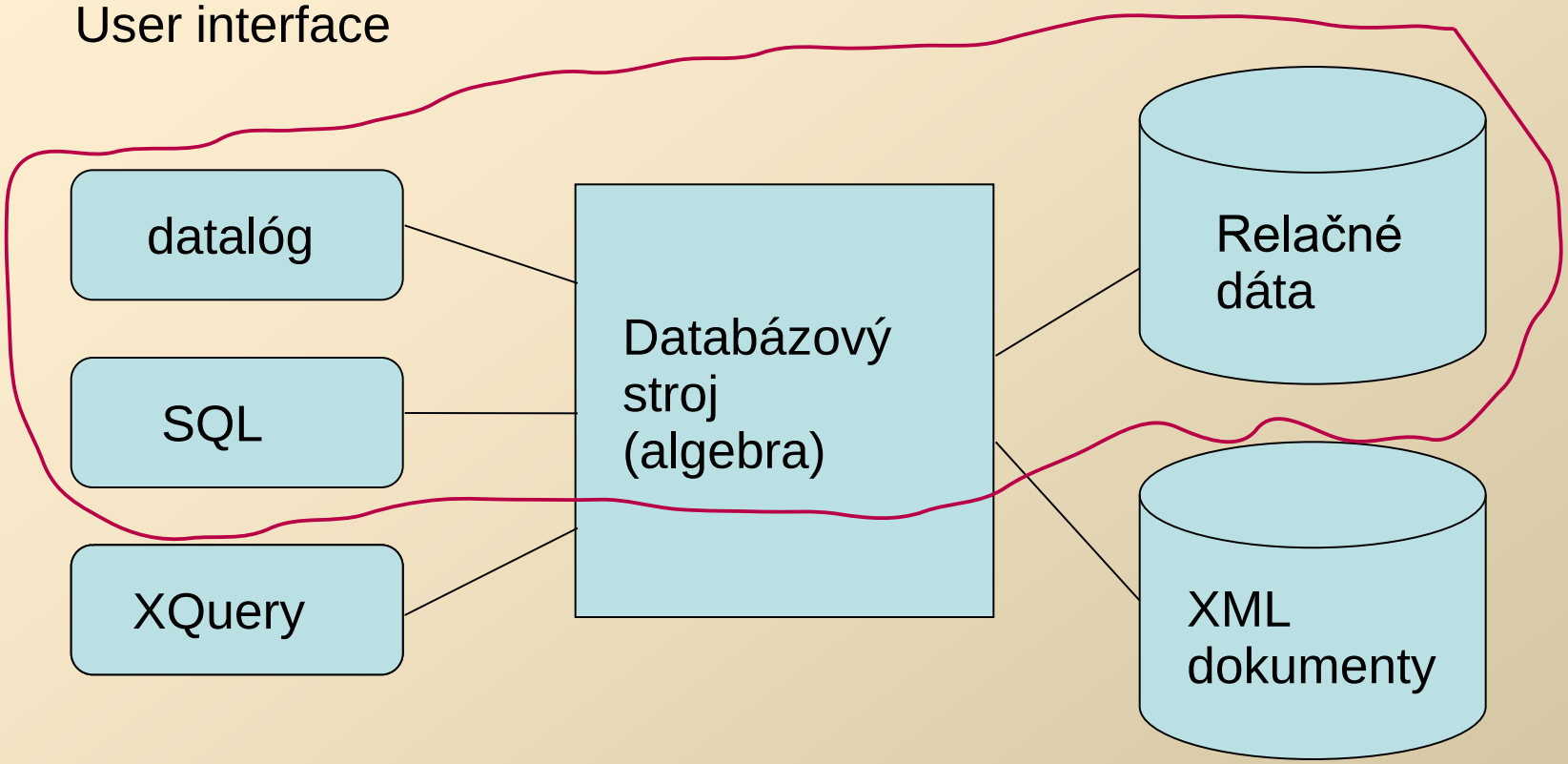
Upresnenie obsahu

- Názov prednášky je historický
- Presnejšie ide o implementáciu a optimalizáciu databázových systémov so zameraním na
  - *relačné a*
  - *deduktívne bázy dát*
- Existujú aj iné databázy – "navigačné"
  - sieťové, objektové, xml, textové
  - Sú zatiaľ sú len výzvou pre výskum, nemajú teóriu. Možno je to len únik programátorov pred matematikou.

Na úrovni matematickej abstrakcie

- **Je to skoro jedno**
  - GRAF G = $<N, E \subseteq N \times N>$ binárna relácia
  - STROM špeciálny prípad grafu
- Graf implementujeme buď ako binárnu reláciu, alebo strom s odkazmi.
- Nejde o podstatu, ale o to ako sa nám hodí o tom hovoriť.

Štruktúra moderného db systému



OOQL

Objektový prístup k DB prináša málo. OOQL je prakticky SQL. (Tabuľky nie sú 1.NF, sú štruktúrované.)
Vhodný je pri navrhovaní DB a pre niektoré špeciálne typy DB. Napr.: Geografické databázy.

Načo sa zameriava „súčasný“ databázový výskum?

1. Integrácia informácii.
2. Olap a dátové kocky.
3. Spracovanie prúdov.
4. Semištruktúrované dáta a XML.
5. Partnerské a grid databázy.
6. Dolovanie informácii z dát (Datamining).
7. Distribuované databázy (už aspoň 30 rokov). Problém je, že ich vlastne nikto nechce.
8. Cloud computing?

Úlohy

- Rozšírenia datalógu, funkčné symboly (pure prolog), sémantika a modely
- Preklad datalógu a SQL do relačnej algebry
- Mapovanie XML dokumentov do relačne-logickej schémy (shredding)
- **Optimalizácie**
  - **Na úrovni datalógu** (uživateľského jazyka)
  - **Na úrovni algebry** (medzijazyka), že je to algebra je trochu abstrakcia.
  - **Na fyzickej úrovni** (počet diskových prenosov)

Funkčné symboly

- Logická interpretácia: predikáty vracajú logické hodnoty (true, false), funkcie vracajú ľubovolné hodnoty.
- Náhrada funkcie predikátom:
  $f(x_0, \ldots, x_{n-1}) = y \Leftrightarrow p(x_0, \ldots, x_{n-1}, y)$ (Graf funkcie)
  Platí funkčná závislosť $x_0, \ldots, x_{n-1} \rightarrow y$.
- Funkčné symboly budeme používať len na štruktúrovanie a reprezentáciu dát.
- Prológ, CL všetky dáta aj prirodzené čísla vytvára pomocou funkčných symbolov.

Príklad: aritmetika

nat(0).
nat(s(x)) ← nat(x).
add(x,0,x).
add(x,s(y),s(z)) ← add(x,y,z).
le(x,y) ← add(x,z,y).
mult(x,0,0).
mult(x,s(y),z) ← mult(x,y,v),add(v,x,z).
mult(x,y,z) ← mult(y,x,z).

Vysvetlenie k príkladu

Je to datalógovská inplementácia unárnej aritmetiky. Prirodzené čísla sú reprezentované termami
0, s(0), s(s(0)), ...

Predikát nat hovorí *je prirodzené číslo*. Predikáty add(x,y,z) ≡ z=x+y, mult(x,y,z) ≡ z=x×y a le(x,y) ≡ x ≤ y.

V štandardných situáciach takéto definície nebudeme používať, ale spoľahneme sa na zabudované aritmetické predikáty, ktoré sú efektívnejšie. Ak je to potrebné, dá sa však v datalógu takto programovať.

Cvičenie (skôr z teórie vypočitateľnosti ako z databáz):
Naprogramujte všetky elementárne matematické operácie (+, -,×, /) a funkcie (sqrt, sin, cos, log, exp). Vymyslite reprezentáciu, celých a racionálnych čísieľ.

Termy ako dátové štruktúry

Fragment XML dokumentu:

```
<adr>
    <miesto>
        <ulica> Dlhá </ulica>
        <cis> 123 </cis>
    </miesto>
    <obec> Pezinok
        <psc> 86100 </psc>
    </obec>
</adr>
```

a(m(u(Dlhá),c(123)),o(Pezinok,p(86100)))

Iný príklad binárne stromy

- Predikáty:
  - strom(x) platí, keď x je binárny strom
  - label(z) platí, keď z je meno vrcholu
- Funkčné symboly
  - null prázdny strom
  - node(A,l,r) strom s koreňom A, l je ľavý a r pravý podstrom
- Pravidlá
  - strom(null).
  - strom(node(A,x,y)) ← label(A),strom(x),strom(y).

Výpočet datalógových programov s funkčnými symbolmi

- Nevznikajú problémy. Naivná a seminaivná evaluácia funguje.
- Najmenší pevný bod existuje.Nemusí však byť konečný. Napr.: nat(x), strom(x) a pod.
- Herbrandovo univerzum ≡ Množina všetkých termov takých, že každý z nich sa môže vyskytnúť pri nejakom výpočte daného datalógového programu.

Termy a substitúcie

Definícia (term):

1. Premenná je term.
2. Nech $t_0, ..., t_{n-1}$ sú termy a f je n-árny funkčný symbol, potom aj $f(t_0, ..., t_{n-1})$ je term.

Definícia (substitúcia):

Substitúciou nazývame funkciu $\sigma: V \rightarrow T$, kde V je množina premenných a T množina termov.

Substitúcie zapisujeme: $\sigma = [\{x_i \mapsto t_i\}_{i=1}^{n}]$

Aplikácia substitúcie:

Substitúciu σ na term t aplikujeme tak, že výsledok je term s=tσ, ktorý vznikne nahradením premenných vyskytujúcich sa v t aj σ príslušnými termami. Nahradenie sa vykoná naraz pre všetky premenné.

Usporiadanie a ekvivalencia na množine termov

Definícia: Hovoríme, že term t je aspoň tak všeobecný ako term s, píšeme $t \sqsupseteq s$, ak existuje substitúcia σ taká, že $s=t\sigma$.

Definícia: Hovoríme, že termy t a s sú ekvivalentné, píšeme $t \cong s$, ak $t \sqsupseteq s$ a $s \sqsupseteq t$.

Lema: Dva termy sú ekvivaletné, $t \cong s$ práve vtedy, keď sa odlišujú len pomenovanín premenných.

Niekedy sa táto štruktúra nazýva „term algebra“. Nie je to algebra, je to niečo ako zväz. Nie je to úplný zväz maximá sú premenné minimá uzavreté (ground) termy. Presnejšie zväz dostaneme po faktorizácii podľa ekvivalencie.

Substitúcie

Definícia: Hovoríme, že substitúcia σ je zložením substitúcii ρ a τ, píšeme $\sigma = \rho \circ \tau$, ak pre každý term t platí $t\sigma = (t\,\tau)\rho$.

Hoci substitúcie sú funkcie, skladanie substitúcií nie je skladanie funkcií.

Definícia: Hovoríme, že substitúcia σ je aspoň tak všeobecná ako substitúcia τ, píšeme $\sigma \sqsupseteq \tau$, ak existuje substitúcia ρ taká, že $\sigma = \rho \circ \tau$.

Definícia: Dve substitúcie σ a τ sú ekvivalentné, píšeme $\sigma \simeq \tau$, ak $\sigma \sqsupseteq \tau$ a $\tau \sqsupseteq \sigma$.

Lema: Dve permutácie premenných (nie nutne tých istých) sú ekvivalentné substitúcie.

Pozn.: Prázdna (všade nedefinovaná) substitúcia, rôzne permutácie premenných sú ekvivalentné substitúcie.

Substitúcia, ktora nie je ekvivalentná identickej substitúcii sa nazýva špecializácia.

Term matching

Úloha: Daný je term t a vzor term s. Úlohou je nájsť substitúciu ρ takú, že tρ = s. (Rieší otázku, či term s vznikol špecializáciou termu t)

Riešenie: Začneme prázdnou (všade nedefinovanou) substitúciou ρ aplikujeme rekurzívnu procedúru:

procedure match(u,v): **boolean**
if u is a variable **then** { **if** ρ(u) is undefined **then** { ρ(u):= v; **return**(true)}
else if ρ(u) = v **then return**(true)
else return(false) }
else if u = f(u_0, … u_{k-1}) and v = f(v_0, … v_{k-1}) **then**
{ i:= 0; **while** i < k and match(u_i,v_i) **do** i:= i+1;
if i = k **return**(true) **else return**(false) }
else return(false)
end;

Ak procedúra vráti true, v ρ bude hľadaná substitúcia. Ak vráti false, riešenie neexistuje.

Unifikácia – riešenie rovníc v term algebre

1. Úloha: Daná je dvojica termov t a s. Úlohou je nájsť dvojicu najvšeobecnejších substitúcií τ a σ takých, že $t\tau = s\sigma$.
2. Obvykle sa v literatúre unifikáciou nazýva riešenie úlohy: Daná je dvojica termov t a s najdite najvšeobecnejšiu substitúciu σ takú, že $t\sigma = s\sigma$. Táto úloha je klasickou matematickou úlohou. Vo volnej algebre termov riešte rovnicu: $t = s$.
3. Matematici riešia rovnice aj v algebrách charaterizovaných nejakým systémom rovnosti, vtedy hovoríme o unifikácii *mod E*, kde *E* je množina rovností. (UI paramodulácia)
4. Úloha 1 sa dá modifikovať nasledovne: Nájdite dvojicu najvšeobecnejších substitúcií ι a σ, takých, že $t\,\iota\,\sigma = s\sigma$. Túto úlohu nazývame slabou unifikáciou.

Tvrdenia

T1: Z riešenia úlohy 4 dostaneme riešenie úlohy 1 tak, že $\tau = \iota \circ \sigma$.

T2: Najvšeobecnejšie riešenie úlohy 4 dostaneme tak, že ι je premenovanie premenných v t tak, aby boli rôzne od premenných vyskytujúcich sa v s. Označíme $t' = t\,\iota$.
A riešime úlohu 2: $t'\sigma = s\sigma$

T3: Riešenie úlohy 1 podľa T1 a T2 je najvšeobecnejšie riešenie úlohy 1

Algoritmy unifikácie

Existujú dva prístupy:

- Manipulácia s množinou rovníc
  - Herbrand (1930)
  - Marteli Montanari (1982)
- Manipulácia so stromami resp. dagmi termov.
  - Robinson (1965)
  - Corbin Bidoit (1983)
  - Ružička Prívara (1989)

Vybrali sme len niekoľko reprezentatov z oboch tried algoritmov. V tejto prednáške uvedieme len dva ostatné doporučujeme na samostatné štúdium.

Herbrandova metóda

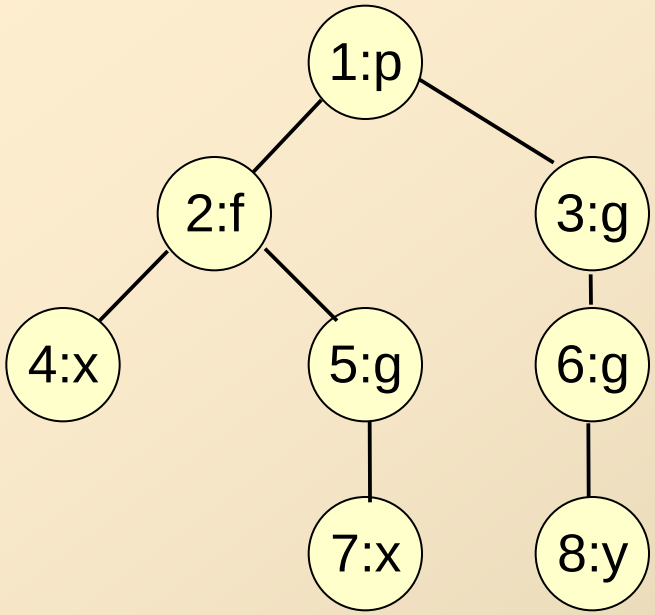
Herbrandova metóda predpokladá, že je daná množina E rovníc medzi termami.
Nech σ je prázdna substitúcia.
Algoritmus:
while E ≠ Ø **do**
{ select an equation e ∈ E; E:= E – {e};
if e is x = t or e is t = x **then**
{ **if** x occurs in t **then** return(solution does not exist)
else { σ:=σ∘[x↦t]; E:=E[x↦t] }
else if e is of the form $f(s_0, \ldots, s_{n-1}) = f(t_0, \ldots, t_{n-1})$ **then**
$E:=E \cup \{s_0 = t_0, \ldots, s_{n-1} = t_{n-1}\}$
else return(solution does not exist)
};

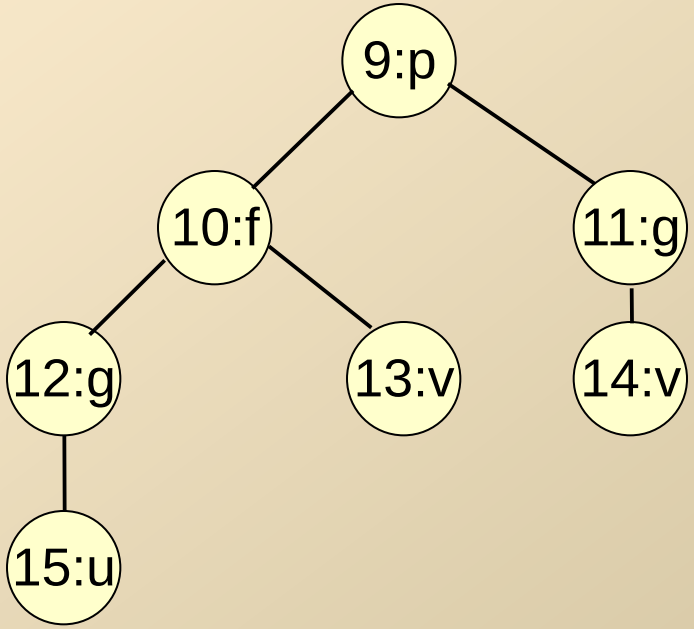
Algoritmus riešenia rovníc

Predpokladáme, že termy sú reprezentované stromami alebo dagmi.
Definujeme ekvivalenciu medzi vrcholmi nasledovne:

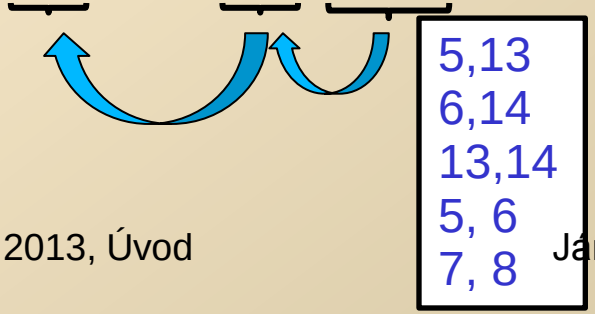
1. Korene oboch termov sú ekvivalentné.
2. Ak sú dva uzly ekvivalentné musia byť ekvivalentné aj dvojice ich synov v poradí.
3. Listy označené rovnakou premennou alebo tou istou konštatou sú ekvivalentné.
4. Z takto vypočítaných dvojíc ekvivalentných uzlov (hrán) vypočítame symetrický, reflexívny a tranzitívny uzáver. Množiny ekvivalentných uzlov.
5. K triedám ekvivalencie priradíme substitúcie:
   - Ak nejaká trieda obsahuje uzly s rôznym funkčnými symbolmi, riešenie neexistuje
   - Ak obsahuje iba premenné, vyberieme jednu, na ktorú ostatné premenujeme.
   - Ak obsahuje premenné aj uzly s funkčným symbolom, nahradíme všetky premenné podtermom prísluchajúcemu jednému uzlu s funkčným symbolom (je jedno ktorému).
6. Preveríme na zacyklenie (occur check).

Príklad 1





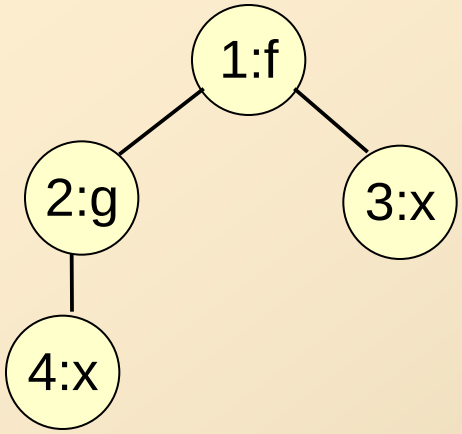
Triedy ekvivalencie:
{1,9}, {2,10}, {3,11},
{4,7,8,12}, {5,6,13,14}, {15}

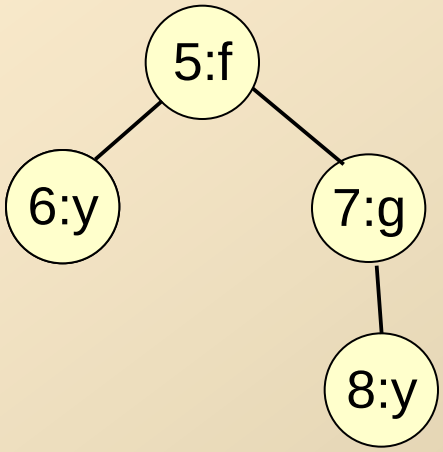


Konštrukcia substitúcie (mgu):
y↦g(u), x↦g(u), v↦g(x).

Explicitné riešenie:
x = g(u), y = g(u), v = g(g(u)).

Príklad 2





Triedy ekvivalencie:
{1,5}, {2,6,8}, {3,4,7}

Konštrukcia mgu:
$y \mapsto g(x)$, $x \mapsto g(y)$.
Cyklus.

Riešenie neexistuje.
x = g(g(...g(u)...)), y = g(g(...g(u)...)) ?

Asymptotická zložitosť riešenia

- Traverzovaním oboch výrazov vytvoríme triedy ekvivalencie také, že každý uzol je ekvivaletný iba sám so sebou. *O(n).*
- Nasleduje synchronné traverzovanie:
  - Find uzol n_1
  - Find uzol n_2
  - Union n_1 n_2

A(m, n) = **if** *m=0* **then** *n+1*
else if *n=0* **then** *A(m-1, 1)*
else *A(m-1, A(m, n-1))*;
D(n) = A(n,n). *α(D(n)) = n.*

- Zložitosť union find úlohy je *O(nα(n))*, kde *α* je inverzná Ackermannova funkcia.
- Kontrola zacyklenia je zložitosti *O(n)* napr. topologickým triedením.

Efektívna implementácia

Nie je potrebné implementovať program ako, sme ho prezentovali kvôli ľahšej analýze. Nepotrebujeme iniciálnu fázu. Môžeme začať priamo synchronným traverzovaním. K tomu musíme viesť oddelene triedy ekvivalencie uzlov a triedy ekvivalencie premenných.
Ak operácia Find nie je úspešná založíme novú triedu ekvivalencie. O triede ekvivalencie, okrem zoznamu uzlov vedieme aj funkčný symbol a premennú.
Ak sa funkčný symbol pri opätovnom vyhladaní nezhoduje, môžeme hneď skončiť. Ak trieda ekvivalencie obsahuje už nejakú premennú a pri opätovnom vyhladaní máme inú premennú, urobíme tieto premenné ekvivaletné.
Occur check sa modifikuje: Či niektorá premenná výslednej substitúcie nie je ekvivalentná premennej na ľavej strane.

Varovanie

Každý pokus o explicitné vyjadrenie výsledku môže viesť k exponenciálnej zložitosti výpočtu.

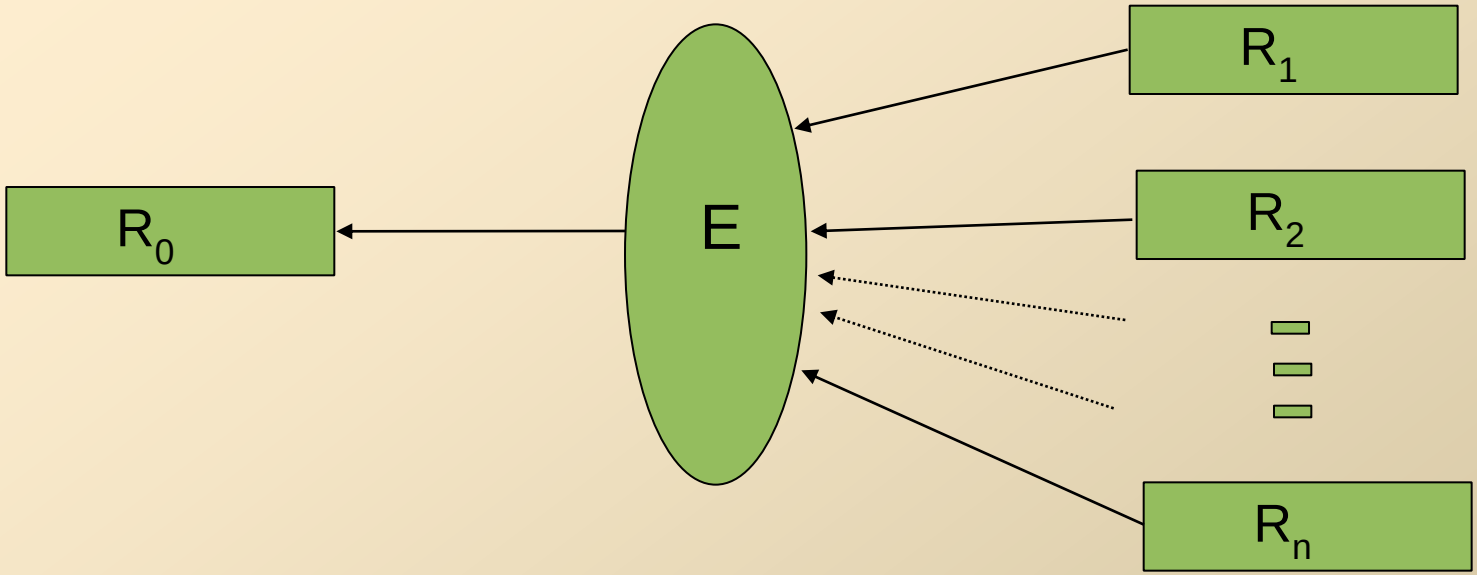
Príklad:

$f(x_0, x_1, x_2, \ldots, x_{n-1}) = f(g(x_1, x_1), g(x_2, x_2), \ldots, g(x_n, x_n))$

$x_0 \mapsto g(x_1, x_1)$
$x_1 \mapsto g(x_2, x_2)$
$x_2 \mapsto g(x_3, x_3)$
...
$x_{n-1} \mapsto g(x_n, x_n)$

Postupným spätným explicitným vyjadrením:
$x_{n-1} = g(x_n, x_n)$
$x_{n-2} = g(g(x_n, x_n), g(x_n, x_n))$
$x_{n-3} = g(g(g(x_n, x_n), g(x_n, x_n)), g(g(x_n, x_n), g(x_n, x_n))$
...
Pre dĺžku l_{i-1} i-1. výrazu platí: $l_{i-1} = 2 \times l_i + 4$
1, 6, 16, 36, 76, 156, 316, ...

Výpočet datalógových programov – rekapitulácia



Štruktúra výrazu E:
- Projekcia
- Zjednotenie
- Bezpečné výrazy

Táto schéma výpočtu má v prípade datalógu s funkčnými symbolmi malý problém, relačná algebra pracuje s tabuľkami a premennými atribútmi. V jazyku vystupujú podciele a argumenty.

Výpočet rekurzívnych programov bez negácie

Systém rovníc: $\vec{P} = \vec{E}(\vec{P}, \vec{R})$, kde $\vec{P}$ sú intencionálne a $\vec{R}$ extenzionálne predikáty.

Riešenie: naivnou iteráciou. Začnememe $\vec{P_0} = \vec{0}$. Postupne počítame postupne P_1, P_2, … , podľa vzorca:

$$\vec{P}_i = \vec{E}(\vec{P}_{i-1}, \vec{R}),$$

pokiaľ $\vec{P}_i \neq \vec{P}_{i+1}$.

Podľa Tarského vety o pevnom bode tento proces vždy skončí (konverguje). Stačí overiť, že prirodzené spojenie, zjednotenie, projekcia a premenovanie sú „neklesajúce" operácie. Vypočítané riešenie je najmenší pevný bod uvedeného systému rovníc. Z vlastnosti implikácií plynie, že každé riešenie uvedeného systému rovníc musí obsahovať vety vypočítané našim algoritmom.

Konverzia argumentov na premenné Q = atov(p, P)

Nech P je relácia pre podcieľ $p(t_1, \ldots, t_k)$. Nech $X_1, ..., X_n$ sú premenné vyskytujúce sa v tomto podcieli. Definujeme reláciu Q so schémou $X_1...X_n$ nasledovne:

Q := Ø;

for each tuple $<s_1, ..., s_k> \in P$ **do**

  if match($p(t_1, \ldots, t_k)$, $p(s_1, ..., s_k)$) **then**

Q := Q ∪ $<\rho(X_1), ..., \rho(X_n)>$;

Na túto operáciu sa môžeme dívať ako na novú operáciu relačnej algebry, zvrhlú selekcio-projekciu, alebo druhorádovú funkciu podobnú agregácii.

Po transformácií podcieľov operáciou **atov** môžeme výrazy vyhodnotiť operáciami relačnej algebry. Výsledok musíme konvertovať späť na argumenty.

Konverzia premenných na argumenty S = vtoa(s, R)

Nech $R(X_1, ..., X_n)$ je výsledok výpočtu výrazu E pre telo bezpečného pravidla. Nech $s(t_1, \ldots, t_m)$ je predikát pre hlavu pravidla obsahujúci premenné $X_1, ..., X_n$. Definujeme reláciu S pre hlavu pravidla nasledovne:

S := Ø;
for each tuple μ **do**
{ σ := Ø;
for i := 1 **to** n **do** $\sigma := \sigma \cup [X_i \mapsto \mu[X_i]]$;
$S := S \cup \langle t_1, \ldots, t_m \rangle \sigma$
}

Výpočet pravidla

Nech r: $p \leftarrow q_1, \ldots, q_k$ je bezpečné pravidlo a nech $R_1, \ldots, R_k$ sú relácie pre podciele $q_1, \ldots, q_k$, potom reláciu R_0 pre hlavu p vypočítame nasledovne:

1. Pre každý podcieľ q_i vypočítame reláciu $Q_i = atov(q_i, R_i)$.
2. Vypočítame $P = Q_1 \bowtie \ldots \bowtie Q_k$ reláciu pre telo.
3. Výsledok $R_0 = vtoa(p, P)$.

Poznámky: Operácia **atov** je obvykle zbytočná (identická) pre zabudovaný predikát. Join so zabudovaným predikátom je obvykle selekcia. Za istých okolnosti ho môžeme uplatniť aj inak (vstupná množina – pseudoklúč).
Výpočet môžeme optimalizovať a premenné, ktoré sa nevyskytujú vo výsledku môžeme odprojektovať hneď ako ich nepotrebujeme pre join.

Naívna evaluácia

- Predošlý algoritmus rieší jediné pravidlo. Ak máme viac pravidiel s tou istou hlavou, vypočítame každé vzlášť a výsledok zjednotíme.
- Ak graf závislosti predikátov je acyklicky jeho topologickým utriedením dostaneme poradie výpočtu pravidiel (podľa predikátu v hlave).
- Ak graf závislosti predikátov nie je acyklicky. Rozdelíme predikáty na intenzionále a a extenzionálne (dané). Všetky intenzionálnym predikátom priradíme prázdnu množinu a iterujeme podľa Tarského vety o pevnom bode. (T.j. v každej iterácií vypočítame nové všetky nové intenzionálne predikáty za pomoci starých.
- Je tu asi ešte značný priestor pre optimalizáciu: relaxácia na miesto iterácie, separácia cyklov.

Seminaívna evaluácia (diferenčná schéma)

- Pre každý intenzionálny predikát p_i, definujeme jeho diferenciu Δp_i, tejto priradíme reláciu ΔP_i.
- Pre každú nerekurzívnu množinu pravidiel s hlavou p_i tvaru $p_i \leftarrow q_1, \ldots, q_k$, kde q_j sú extenzionálne predikáty, inicializujeme ΔP_i výsledkom výpočtu tejto množiny pravidiel.
- Pre každé pravidlo s intenzionálnym predikátom v hlave vytvor množinu diferenčných pravidiel ako množinu všetkých takých pravidiel, že hlava je Δp_i a v tele aspoň pred jedným intenzionálnym predikátom je Δ.
- Opakujeme nasledujúci cyklus vypočítame $\Delta P_i'$ pomocou existujúcich ΔP_i. Položíme $\Delta P_i = \Delta P_i' - P$ a $P = P \cup \Delta P_i$. Cyklus opakujeme pokiaľ aspoň jedno ΔP_i je neprázdne.