Datalóg – logika ako dátový model

- Syntax datalógu je podobná syntaxy jazyku logického programovania – prológu.
- Základný princíp definuje sa najprv pohľad – program. Potom dotaz – na čiastočnú zhodu na niektorý predikát definovaného pohľadu.

Príklad:

šéf(Vedúci, Zamestnanec) ← vedie (Vedúci, Zamestnanec).

šéf(Vedúci, Zamestnanec) ← šéf(Vedúci, Z), vedie(Z, Zamestnanec).

? – šéf(X, peter).

Nebudem dodržovať prológovske konvencie.

Mená relácii a konštanty budú začínať veľkými písmenami. Premenné malými.

- Premenné
- Databázové – extenzionálne predikáty
- Definované – intencionálne predikáty
- Zabudované predikáty (=, <,>, ...)

Príklad použitia klauzálnej logiky

Databázová relácia *rodič(x, y)*

súrodenci(x, y) ← *rodič(z, x), rodič(z, y), x ≠ y.*
bratranci(x, y) ← *rodič(u, x), rodič(v, y), súrodenci(u, v).*
bratranci(x, y) ← *rodič(u, x), rodič(v, y), bratranci(u, v).*
príbuzní(x,y) ← *súrodenci(x, y).*
príbuzní(x,y) ← *príbuzní(x, z), rodič(z, y).*
príbuzní(x,y) ← *príbuzní(z, y), rodič(z, x).*

Implementácia aritmetiky zabudovaných funkcií

p(x, z) ← *q(x, u, v), z=(u+v)/ 5* . alebo presnejšie
p(x, z) ← *q(x, u, v), add(u, v, y), div(y, 5, z).*

V zabudovaných funkciách predikátoch treba „strážiť“ vstupné množiny.

Terminológia – klauzuly a formuly

Fakt = (pozitívna) ***atomická formula***

- $p(a_1, a_2, \ldots, a_n)$
- $p(X_1, X_2, \ldots, X_n)$

Literál = ***atomická formula*** alebo ***negovaná atomická formula*** ($p, \neg p$)

Klauzula = ***logický súčet*** litearálov (literály pospájané logickou spojkou $\vee$ (***or***)).

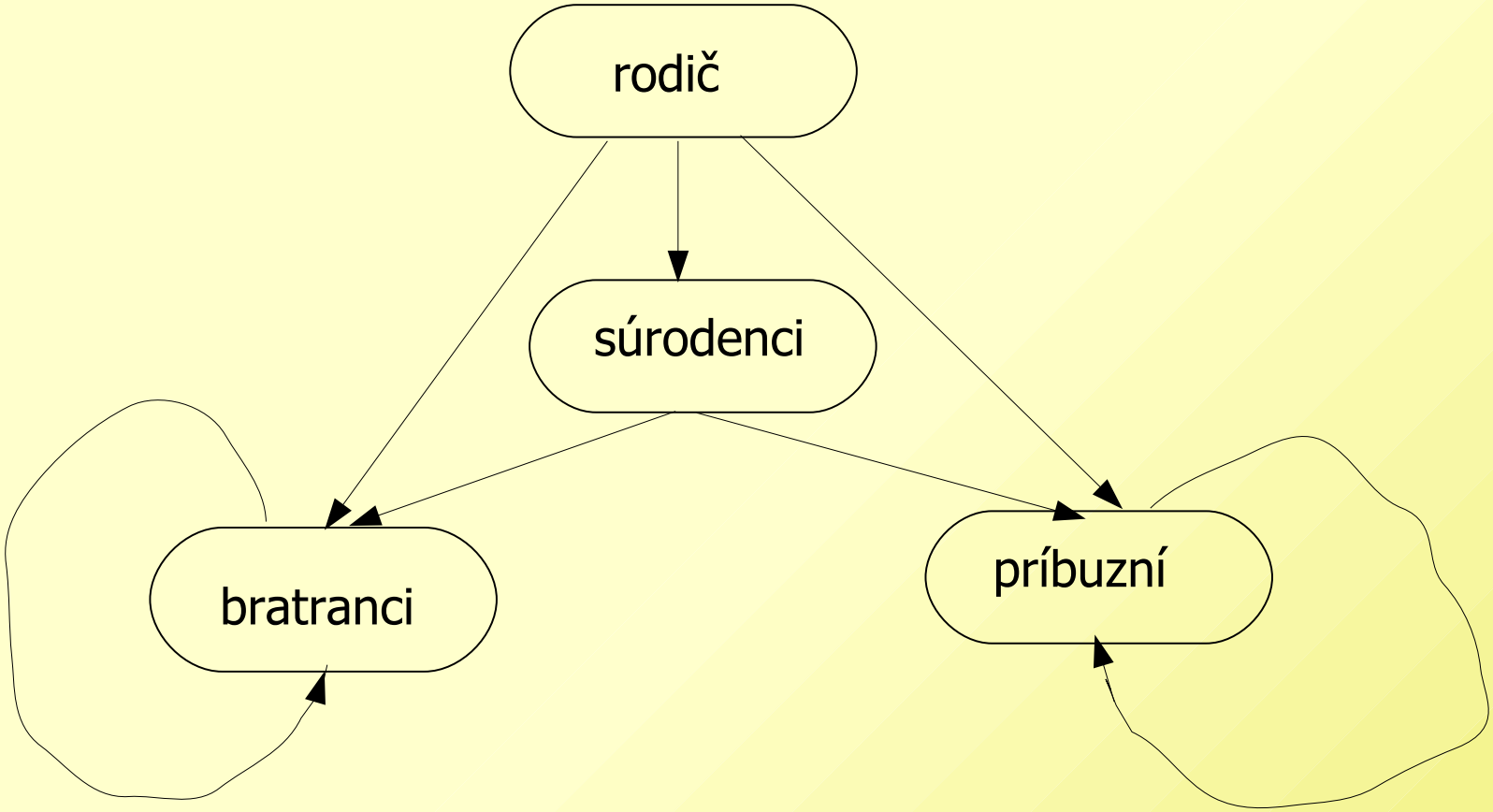
Hornova klauzula = klauzula obsahujúca ***najviac jednu*** (nenegovanú) ***atomickú formulu***.

Hornove klauzuly:

- len 1 atomická formula – fakt
- zjednotenie negovaných atomických formúl – podmienka
- práve jedna nenegovaná atomická formula

$p_0 \vee \neg p_1 \vee \ldots \vee \neg p_k \equiv p_1 \wedge \ldots \wedge p_k \Rightarrow p_0$ $\qquad$ $p_0 \leftarrow p_1, \ldots, p_k$

Graf závislosti predikátov



Uzly sú predikáty. Hrana z uzla p_1 do uzla p_2, ak predikát p_2 je definovaný pomocou predikátu p_1. Existuje pravidlo s hlavou p_1 ktorého telo obsahuje p_2.

Bezpečné pravidlá – bezpečné premenné

- Premenná je bezpečná ak sa vyskytuje v obyčajnom (nenegovanom, nezabudovanom predikáte) tela pravidla alebo v predikáte x = a.
- Ak v predikáte x = y jedna premenná je bezpečná, potom aj druhá je bezpečná.

Pravidlo je bezpečné, ak všetky jeho premenné sú bezpečné.
Dôsledok: Každá premenná v hlave sa musí vyskytovať aj v tele pravidla.

Pravidlo je **rekurzívne**, keď predikát hlavy sa vyskytuje aj v tele pravidla. Program je rekurzívny, keď graf závislosti predikátov obsahuje cyklus.

Výpočet nerekurzívnych programov

každé pravidlo prerobíme na relačný výraz: náhradou čiarok za prirodzené spojenia resp. rozdiely.

Problémom je, keď viac pravidiel má ten istý predikát v hlave.
V takomto prípade musíme vytvoriť „rektifikované“ pravidlá:

Predikáty v hlave obsahujú tie isté premenné a iba premenné.

Premenovanie, zavedenie premennej

$p(x, a) \leftarrow \ldots$.

$p(x, y) \leftarrow \ldots, y = a$.

Výsledný výraz pre predikát p je zjednotenie výrazov, pre telá rektifikovaných pravidiel s hlavou p.

Poradie výpočtu predikátov je dané topologickým utriedením grafu závislosti predikátov.
Rektifikáciu a úpravu na algebraické výrazy môžeme urobiť aj pre rekurzívne programy – systém rovníc v relačnej algebre.

Výpočet rekurzívnych programov bez negácie

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

Riešenie: naivnou iteráciou. Začnememe $\vec{P}_0 = \emptyset$. Postupne počítame postupne $\vec{P}_1$, $\vec{P}_2$, … , podľa vzorca:

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

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.

Negácia – stratifikované programy

Rozdiel, agregácia nie sú neklesajúce operácie. Vo všeobecnosti programy s nimi nemusia mať pevný bod.

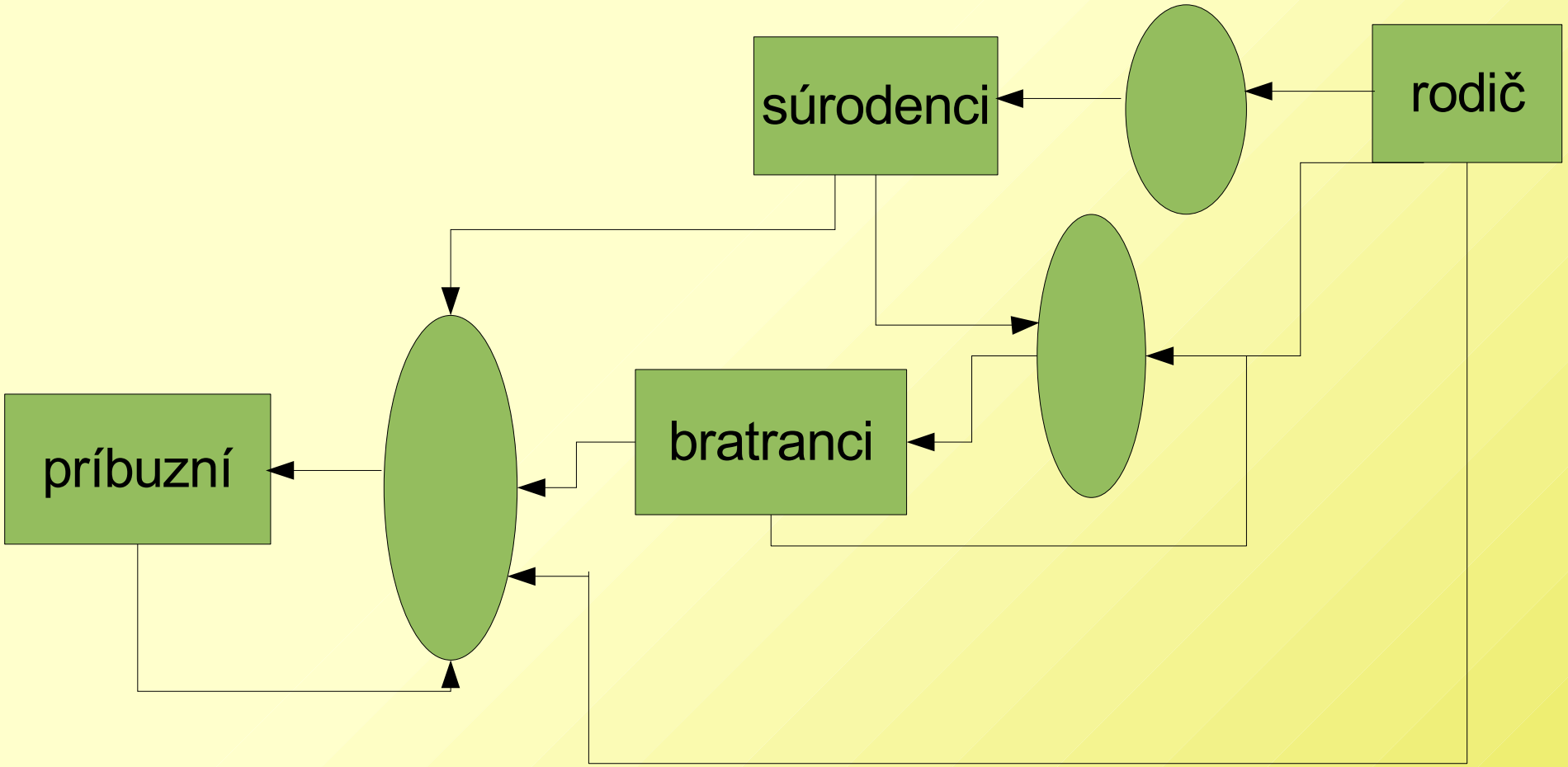
Stratifikované programy: Ak definícia predikátu ***p*** obsahuje negáciu predikátu ***q***, potom v grafe závislosti predikátov neexistuje cesta od ***p*** ku ***q***.

Podmienky stratikácie:

- Každému predikátu je priradené celé číslo stratum (vrstva)
- Ak pravidlo $p \leftarrow \ldots q \ldots$. Potom $\mathcal{S}(p) \geq \mathcal{S}(q)$.
- Ak pravidlo $p \leftarrow \ldots \neg q \ldots$. Potom $\mathcal{S}(p) > \mathcal{S}(q)$.

Postup stratifikácie: Na začiatku, priradíme všetkým predikátom stratum 1 a spočítame predikáty - *n*. Prezeráme postupne pravidlá. Ak je porušená niektorá s podmienok stratifikácie, zvýšime stratum predikátu hlavy na najmenšie prirodzené číslo, ktoré ju splňuje. Ak stratum väčšie ako *n,* program sa nedá stratifikovať. Opakuj zakiaľ sa niečo mení.

Stroj na výpočet datalógovských programov



SQL-99 „with statement“

WITH R_1 **AS** <dotaz>, R_2 **AS** <dotaz>, ..., R_n **AS** <dotaz>
<dotaz obsahujúci R_1, R_2, ..., R_n a eventuálne iné relácie>

Idea:

(1) Vypočítať R_1, R_2, ..., R_n do pracovných (temporary) relácií

(2) Vyhodnotiť dotaz obsahujúci R_1, ..., R_n a ostatné relácie

(3) Zrušiť R_1, R_2, ..., R_n

- Príkaz slúži na optimalizáciu opakujúcich sa vložených poddotazov
- Nahradzuje možnosť písať dotazy za from
- Môže byť rekurzívny, v tom prípade sa počíta metódou pevného bodu.

Rekurzia v SQL-99

Príklad: Najdite predkov Márie z relácie Rodičia(rodič, dieťa)

```
WITH RECURSIVE Predkovia(predok, potomok) AS
((SELECT rodič as predok, dieťa as potomok FROM Rodičia)
UNION
(SELECT Predkovia.predok, Rodičia.dieťa as potomok
FROM Predkovia, Rodičia
WHERE Predkovia.potomok = Rodičia.rodič ))

SELECT predok FROM Predkovia WHERE potomok = "Mária"
```

Poznámka k príkazu **with**

- Nerekurzívny with má malý význam. Väčšina implementácii SQL dovoluje dotazy v zozname with napísať v select príkaze za from.
- Aj create view sa dá použiť.
- Možno zlepšuje čitateľnosť.

WITH R_1 **AS** <dotaz1>, R_2 **AS** <dotaz2>, ..., R_n **AS** <dotazn>
<dotaz obsahujúci R_1, R_2, ..., R_n a eventuálne iné relácie>;

sa preloží ako

SELECT ...
FROM <dotaz1> R_1, <dotaz2> R_2 ..., <dotazn> R_n
. . .

Manipulácie s dátami

- Vkladanie ($\cup =$ skratka za $P := P \cup \_$)
  - hodnôt
  - výsledku dotazu
- Vynechanie ($- =$ skratka za $P := P - \_$)
- Modifikácia $(:=)$

Napodiv algebru stačí rozšíriť o príkaz priradenia.

Pozor: Na rozdiel od SQL na pravej strane priradenia pri modifikácii musí byť zjednotenie modifikovanej a nemodifikovanej časti.

Použijeme tú istú notáciu aj pre datalóg a kalkul.

Vloženie vety

Všeobecný tvar vloženia:

INSERT INTO R(A_1,…., A_n) VALUES (v_1,…., v_n)

R(A_1,…., A_n) $\cup =$ (A_1 = v_1, ... , A_n= v_n)

Vloženie nového nákupu do databázy:

INSERT INTO Kontrakt(predávajúci, kupujúci, výrobok, sklad)
VALUES ('Jožo', 'Fero', 'rádiobudík', 'Tesko-2')

Môžeme vynechať mená atribútov, ak uvedieme všetky v správnom poradí.

Ak neuvedieme všetky atribúty, budú namiesto neuvedených atribútov hodnoty NULL.

Vloženie výsledku dotazu

```
INSERT   INTO   PRODUKT(názov)
    SELECT  DISTINCT  výrobok
    FROM     Kontrakt
    WHERE  výrobok  NOT IN
                (SELECT  názov
                  FROM     Produkt)
```

Dotaz nahradzuje rezervované slovo VALUES.
Dotaz nasleduje za príkazom vloženia .

Produkt (názov, cena, kategória, výrobca) $\cup =$
Kontrakt (_, _, _, názov),$\neg$Produkt (názov, _, _, _),
cena=NULL, kategória=NULL, výrobca=NULL

Produkt (názov, cena, kategória, výrobca)
Kontrakt (predávajúci, kupujúci, sklad, výrobok)

Vynechanie

```
DELETE    FROM    Kontrakt
WHERE     predávajúci = 'Jožo'   AND
          výrobok  = 'holiaci strojček'
```

Syntax podmienky je ako vo formule selekt.

V SQL neexistuje žiadny spôsob odstrániť len jeden výskyt n-tice, ktorá sa v tabuľke vyskytuje viackrát.

```
Kontrakt (predávajúci,  kupujúci,  sklad,  výrobok) – =
        Kontrakt ('Jožo',  kupujúci,  sklad,  'holiaci strojček')
```

Modifikácia (zmena)

```
UPDATE Produkt
SET    cena = cena*(1 – 0.1)
WHERE  Produkt.meno  IN
                (SELECT výrobok
                  FROM    Akcie
                  WHERE  dátum = today);
```

Produkt (názov, cena, kategória, výrobca) : =
Produkt (názov, cena, kategória, výrobca),Akcie(názov,'today') ∪
Produkt (názov, cena, kategória, výrobca),¬Akcie(názov,'today')

Akcie(výrobok, dátum)

Významné tvrdenia o jazykoch na manipuláciu s dátami.

- Kalkul, algebra, datalóg a SQL sú ekvivalentné
  - dôkaz sa urobí prekladom SQL→algebra→kalkul→datalog→SQL
  - indukciou cez štrukúru formúl
- Sú to úplné programovacie jazyky
  - dôkaz simuláciou Turingovho stroja v datalógu alebo SQL
  - Hint: Tri relácie Páska(CisloPolicka, Symbol), TuringovStroj (štvorice/pätice), Stav(stav)

SQL a programovanie

- V sql zväčša neprogramujeme kompletné aplikácie a projekty.
- Rozšírenia PL/SQL (oracle), PgSQL (postgres), 4GL, … .
- Vloženie SQL do hostiteľského jazyka (embeded SQL), C, C++, Pascal, java, … .
  - Statické
  - Dynamické

Pojem kurzoru

- „Problém impedancie“: programovací jazyk pracuje s individuálnými premennými, SQL s množinami.
- Riešenie kurzor – tabuľka v ktorej sa pohybujeme po riadkoch

  as / is

  update

  - declare <názov> [...] cursor for <select > for read only;
  - open <názov>, close <názov>
  - fetch [next | prior | first |last] into < premenné>
  - premenná hostiteľkého jazyka :meno
  - indikátory a stavové premenné srbd

Príklad použitia kurzoru

```
EXEC SQL DECLARE customer_cur CURSOR FOR
SELECT * from customers
WHERE cno >= :cno
FOR READ ONLY;
cno = 1234;
EXEC SQL SET TRANSACTION READ ONLY;
EXEC SQL OPEN customer_cur;
EXEC SQL FETCH customer_cur INTO :cno, :phone;
while (sqlca.sqlcode == 0) {
printf("%d\t%s\n", cno, phone);
EXEC SQL FETCH customer_cur INTO :cno, :phone;
}
EXEC SQL CLOSE customer_cur;
EXEC SQL COMMIT;
```