Query languages 2 (NDBI006)
Expressive power - Part 1

J. Pokorný

MFF UK

Content of the course

1. Three semantics of domain relational calculus (DRC). Definite and safe formulas of DRC. Proof of the equivalence of the relational algebra (RA) and DRC restricted to definite formulas.
2. Proofs of equivalence of query languages.
3. Transitive closure of a binary relation. Impossibility to express it in relational algebra.
4. Composition of RA expressions, the least fixpoint approach, minimal fixpoint.
5. Datalog language – its three possible semantics. Evaluation of a Datalog program without recursion.
6. Evaluation of a Datalog program with recursion - naïve evaluation, method of differences.
7. Datalog with negation. Stratified Datalog
8. Expressive power of Datalog. A relationship of Datalog to other relational languages.
9. Logical problems of information systems.

Content of the course

10. Recursive SQL.
11. Graph Databases
12. Tableau query as visual query interface for e-shops, conjunctive query containment and homomorphism theorem.
13. Tableau query with inequality for e-shops
14. Tableau query and algorithmic complexity of query inclusion.
15. Formal framework for transferability of querying models
16. Datalog with recursion and functional symbols.
17. Datalog with recursion and functional symbols - completeness.

DRC semantics (1)

Assumptions: query expressions $\{x_1,...,x_k \mid A(x_1,...,x_k)\}$, *A* is a DRC formula,

database **R***, *dom* is domain for **R**; *actual domain* of formula *A*, *adom*(*A*), is a set of values from relations in *A* and constants in *A*.

Three problems:

- potential possibility of infinite answer (in the case of infinite *dom*)
- situation, when TRUE-assignment of free variables is not from **R***.
- how to implement evaluation of a quantification (in the case of infinite *dom*) in a finite time.

DRC semantics (2)

3 semantics of DRC, solving the problems:

(i) unlimited interpretation with restricted output

(ii) limited interpretation

(iii) domain-independent queries

Notation: result of a query Q evaluation in the unlimited interpretation as $Q^{dom}[\mathbf{R}^*]$.

Then:

- for (i) the result is defined as $Q^{dom}[\mathbf{R}^*] \cap adom^k$, where k is řád resulted relation.
- for (ii) variables ranges over *adom*, i.e. $Q^{adom}[\mathbf{R}^*]$.

DRC semantics (3)

Q.: $\{x \mid \neg R(A{:}x)\}$

$\Rightarrow$The answer depends on *dom*(*A*).

$\Rightarrow$ Query expression defines different queries for each different domain.

Remark: A query, returning $\emptyset$, can be domain dependent in the case the quantified variable ranges over an infinite set, e.g.

Q.: $\{x \mid \forall\, y\, \mathrm{R}(x,y)\}$

Df.: We say that a query expression is *domain-independent* (*definite*) if the answer to it does not depend on *dom*.

Query language is *domain-independent*, if each its expression is domain-independent. The result of *Q* is equal to $Q^{\mathrm{dom}}[\mathbf{R}^*] = Q^{\mathrm{adom}}[\mathbf{R}^*]$.

BOOK(ISBN,TITLE, AUTHOR)
COPY(C_Number, ISBN)
LOAN(C_Number, Cust_Number, DueDate)

DRC semantics (7)

$\Rightarrow$ evaluation of a domain independent expression in unlimited interpretation returns the same result as in restricted interpretation.

Ex.: $\neg$ BOOK(TITLE:'Introduction to DBS', AUTHOR:a)

IS NOT definite.

$\exists$cn (COPY(cn,i) $\wedge$ LOAN(cn,b,dd))

IS definite

$\exists$cn (COPY(cn,d) $\vee$ LOAN(cn,b,dd))

IS NOT definite, if variables are untyped or of too "wide" types

Theorem (Di Paola 1969): Definiteness of A is not decidable.

$\Rightarrow$ The language of domain-independent expressions is not decidable.

Remark: Relational algebra is a domain-independent language.

DRC semantics (5)

Notation of DRC:

- in unlimited interpretation with restricted output DRC^{rout},
- in limited interpretation DRC^{lim}
- domain independent expressions DRC^{ind}.

Statement: $DRC^{rout} \cong DRC^{lim} \cong DRC^{ind}$. Moreover,

(i) if *Q* is a DRC expression, then there is a domain independent expression Q', which after evaluation returns the same result as Q in unlimited interpretation with restricted output.

(ii) if *Q* is a DRC expression, then there is a domain independent expression Q', which after evaluation returns the same result as Q in limited interpretation.

DRC semantics (6)

Proof (sketch): trivially DRC^{rout} and DRC^{lim} are at least so powerful as DRC^{ind}, i.e. DRC^{ind} < DRC^{lim} and DRC^{rout} < DRC^{lim}

- We show a power of DRC^{lim}

If $Q \in$ DRC^{ind}, then it returns $Q^{dom}[\mathbf{R}^*]$, přičemž $Q^{dom}[\mathbf{R}^*] = Q^{adom}[\mathbf{R}^*]$.

Let $Q \in$ DRC. Then it is possible to construct Q‘ so that all free and bound variables in the formula of query *Q*‘ are restricted to the active domain. Then $D'^{adom}[\mathbf{R}^*] = D^{adom}[\mathbf{R}^*]$. Expression *Q*‘ is however domain independent, so DRC^{lim} < DRC^{ind}. We also demonstrated the (ii) part of the statement. Thus DRC^{lim} $\cong$ DRC^{ind}.

- It holds, that DRC^{rout} is more powerful than DRC^{lim}. A proof of (i) is technically more complicated (see [Hull and Su 94]).

Safe formulas in DRC

Df.: A safe DRC formula, *A*, is a DRC formula, which is definite and syntactically characterizable.

1. $\forall$, $\Rightarrow$ are eliminated
2. if *A* contains a disjunction, then is it is a subformula

   $\varphi_1(x_1,\ldots,x_s) \vee \varphi_2(x_1,\ldots,x_s)$,

   i.e. φ_i contain the same free variables,
3. if A contains a conjunction (maximal), e.g.,

   $\varphi \equiv \varphi_1 \wedge \ldots \wedge \varphi_r$, $r \geq 1$, then each free variable in φ is limited, i.e., at least one of the following conditions holds:

   - A variable is free in φ_i , which is neither arithmetic comparison and nor negation,
   - there is $\varphi_i \equiv x=a$, where a is a constant,
   - there is $\varphi_i \equiv x=y$, where y is limited.

Safe formulas in DRC

4. $\neg$ can be used only in conjunctions of type 3.

Remarks:

- Any safe formula is definite.
- There are definite formulas which are not safe.

Ex.:

$x=y$ IS NOT safe

$x=y \vee R(x,y)$ IS NOT safe

$x=y \wedge R(x,y)$ IS safe

$R(x,y,z) \wedge \neg (P(x,y) \vee Q(y,z))$ IS NOT safe, is definite.

$R(x,y,z) \wedge \neg P(x,y) \wedge \neg Q(y,z)$ IS safe!

Equivalence of relational languages

4 approaches:

- domain relational calculus (DRC)
- tuple relational calculus (NRC)
- relational algebra (A_R)
- DATALOG

We prove: DRC $\cong A_R$

Lemma: Let φ be a Boolean expression created by using $\neg$, $\wedge$, $\vee$ and simple selections X θ Y or X θ k, where $\theta \in \{\leq, \geq, \neq, <, >, =\}$, k is constant and X, Y are attribute names. Then for E(φ), where E $\in$ A_R, there is a relational expression E', whose each selection is simple and E(φ) $\cong$ E'.

Proof: 1. each $\neg$ is propagated to a simple selection and θ is replaced by its negation.

Equivalence of relational languages

2. by induction on the number of operators $\wedge$, $\vee$.

for $\emptyset$ of operators - trivial

$E(\varphi) \equiv E(\varphi_1 \wedge \varphi_2)$ and E contains at most selections, which are simple. Then $E(\varphi) \equiv E(\varphi_1)(\varphi_2)$.

$E(\varphi) \equiv E(\varphi_1 \vee \varphi_2)$ and E contains at most selections, which are simple. Then $E(\varphi) \equiv E(\varphi_1) \cup E(\varphi_2)$.

Ex.: $E \equiv R(\neg (A_1 = A_2 \wedge (A_1 < A_3 \vee A_2 \leq A_3)))$

then $\varphi \equiv A_1 \neq A_2 \vee (A_1 \geq A_3 \wedge A_2 > A_3)$

and $E' \equiv R(A_1 \neq A_2) \cup R(A_1 \geq A_3)(A_2 > A_3)$

From relational algebra to DRC

Theorem: Each query expressible in A_R is expressible in DRC.

Proof: by induction on the number of operators in relational expression E.

1. $\emptyset$ operators in E.

   $E \equiv R \rightarrow \{x_1,\ldots,x_k \mid R(x_1,\ldots,x_k)\}$

   $E \equiv$ const. relation $\rightarrow \{x_1,\ldots,x_k \mid x_1 = a_1 \wedge \ldots \wedge x_k = a_k \vee$
   $x_1 = b_1 \wedge \ldots \wedge x_k = b_k \vee \ldots\}$

2. $E \equiv E_1 \cup E_2$ by the induction hypothesis there are formulas e_1 and e_2 with free variables $x_1,\ldots,x_k$

   $\rightarrow \{x_1,\ldots,x_k \mid e_1(x_1,\ldots,x_k) \vee e_2(x_1,\ldots,x_k)\}$

From relational algebra to DRC

3. $E \equiv E_1 - E_2$

$$\rightarrow \{x_1,\ldots,x_k \mid e_1(x_1,\ldots,x_k) \wedge \neg e_2(x_1,\ldots,x_k)\}$$

4. $E \equiv E_1[i_1,\ldots,i_k]$

$$\rightarrow \{x_{i1},\ldots,x_{ik} \mid \exists x_{j1},\ldots,x_{j(n-k)}\, e_1(x_1,\ldots,x_n)\}$$

5. $E \equiv E_1 \times E_2$

$$\rightarrow \{x_1,\ldots,x_m\, x_{m+1},\ldots,x_{m+n} \mid e_1(x_1,\ldots,x_m) \wedge e_2(x_{m+1},\ldots,x_{m+n})\}$$

6. $E \equiv E_1\,(\varphi)$

$$\rightarrow \{x_1,\ldots,x_k \mid e_1(x_1,\ldots,x_k) \wedge x_A\, \theta\, x_B)\}, \quad \text{if } \varphi \equiv A\, \theta\, B$$

or $\quad x_A\, \theta\, a \quad$ if $\varphi \equiv A\, \theta\, a$

By lemma, it is enough, when φ denotes a simple selection.

Semantic definition of definite formulas

Sufficient conditions for definite formulas A:

1. components of TRUE-assignment of A are from *adom*(A).
2. if $A' \equiv \exists y\ \varphi(y)$, then if for a y_0

   $\varphi(y_0) \Leftrightarrow$ TRUE, then $y_0 \in$ adom(φ).
3. if $A' \equiv \forall y\ \varphi(y)$, then if for a y_0

   $\varphi(y_0) \Leftrightarrow$ FALSE, then $y_0 \in$ adom(φ).

Remark: 2. and 3. holds for any allowable values of free variables in φ (except y).

Remark: explanation of condition 3.

$\forall y\ \varphi(y) \Leftrightarrow \neg\exists y\ \neg\varphi(y)$

$\Rightarrow$ if for a y_0 $\neg\varphi(y_0) \Leftrightarrow$ TRUE, then by 2., $y_0 \in$ adom($\neg\varphi$).

Semantic definition of definite formulas

Since adom($\neg\varphi$) = adom(φ), then

$\varphi(y_0) \Leftrightarrow$ FALSE $\Rightarrow y_0 \in$ adom(φ).

Statement: Elimination of $\forall$ and $\wedge$ from a definite formula leads to a definite formula as well.

From DRC to relational algebra

Statement: Each query expressible by a definite expression of DRC is expressible in A_R.

Proof: by induction on the number of operators in A of the definite expression $\{x_1,\ldots,x_k \mid A(x_1,\ldots,x_k)\}$ (+)

- We express *adom*(*A*) as expression A_R. We denote it as E .
- W alter *A*, that it contains only $\exists$, $\vee$, $\neg$.
- The proof will be done for *adom*(A)$^k \cap \{x_1,\ldots,x_k \mid A'(x_1,\ldots,x_k)\}$. When A' $\equiv$ A and A is definite, $\cap$ leads to expression (+).

By induction:

1. $\varnothing$ of operators in A'. Then A' is atomic formula.

$x_1 \;\theta\; x_2 \rightarrow (E \times E)(1 \;\theta\; 2)$

$x_1 \;\theta\; a \;\rightarrow\; E(1 \;\theta\; a)$

$R(x_1,\ldots,x_m) \rightarrow R(\ldots \wedge i_1 \;\theta\; i_2 \wedge \ldots)[\ldots, i_1,\ldots]$, when, e.g., $x_{i1} = x_{i2}$

From DRC to relational algebra

2. A' has at least one operator and the induction hypothesis holds for all subformulas from A' with less operators than A'.
   - $A'(u_1,\ldots,u_m) \equiv A^1(u_1,\ldots,u_n) \vee A^2(u_1,\ldots,u_p)$. Then for expressions $adom(A)^m \cap \{\underline{u}|A^i(\underline{u})\}$ there are relational expressions E_i. Transformation leads to $\cup$.

     Ex.: $A'(u_1,u_2,u_3,u_4) \equiv A^1(u_1,u_3,u_4) \vee A^2(u_2,u_4)$

     $\rightarrow (E_1\times E)\ [1,4,2,3] \cup (E_2\times E\times E)\ [3,1,4,2]$
   - $A'(u_1,\ldots,u_m) \equiv \neg A^1(u_1,\ldots,u_m)$. Then for expression

     $adom(A)^m \cap \{\underline{u}|A^1(\underline{u})\}$ there is a relational expression E_1. Transformation leads to -, i.e. $E^m - E_1$
   - $A'(u_1,\ldots,u_m) \equiv \exists u_{m+1}A^1(u_1,\ldots,u_m, u_{m+1})$. Then for expression $adom(A)^{m+1} \cap \{\underline{u}|A^1(\underline{u})\}$ there is a relational expression E_1. Transformation leads to [], i.e. $E_1[1,2,\ldots,m]$.

If $A' \equiv A$, then the answer is not changed.

From DRC to relational algebra

Ex.: $\{w,x \mid R(w,x) \wedge \forall y(\neg S(w,y) \wedge \neg S(x,y))\}$ is a definite expression.

Justification: $dom(\neg S(w,y) \wedge \neg S(x,y)) = dom(S)$

Let $y_0 \notin dom(S)$. Then $\neg S(w,y_0) \wedge \neg S(x,y_0) \Leftrightarrow TRUE$.

So, the condition 3 from sufficient conditions is fulfilled

Eliminating $\wedge$ and $\forall$, we obtain the definite expression:

$\{w,x \mid \neg (\neg R(w,x) \vee \exists y(S(w,y) \vee S(x,y))\}$

Transformation:

$S(w,y) \vee S(x,y) \rightarrow (S \times E)[1,3,2] \cup (S \times E)[3,1,2]$

$\exists y$ (-"-) $\rightarrow$ (-"-) [1, 2] we denote as E'

Remark: E' can be optimized as $(S \times E)[1,3] \cup (S \times E)[3,1]$

$\neg R(w,x) \rightarrow E^2 - R$

$\neg R(w,x) \vee \exists y(S(w,y) \vee S(x,y) \rightarrow (E^2 - R) \cup E'$

$\neg$ (-"-) $\rightarrow E^2 - ((E^2 - R) \cup E')$

From DRC to relational algebra

Problem: the result leads to a non-effective evaluation

Optimization:

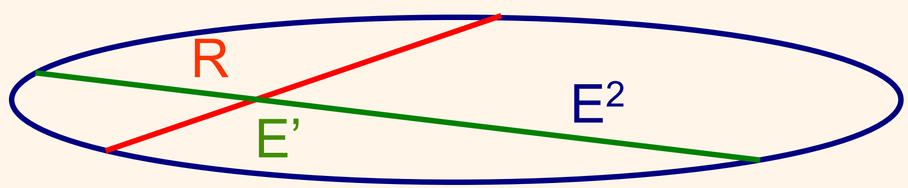
Let $\underline{X}$ denote the complement of X w.r.t. E.

It holds: $\underline{X \cup Y} = \underline{X} \cap \underline{Y}$

$\Rightarrow E^2 - ((E^2 - R) \cup E') = (E^2 - (E^2 - R)) \cap (E^2 - E')$ $=$

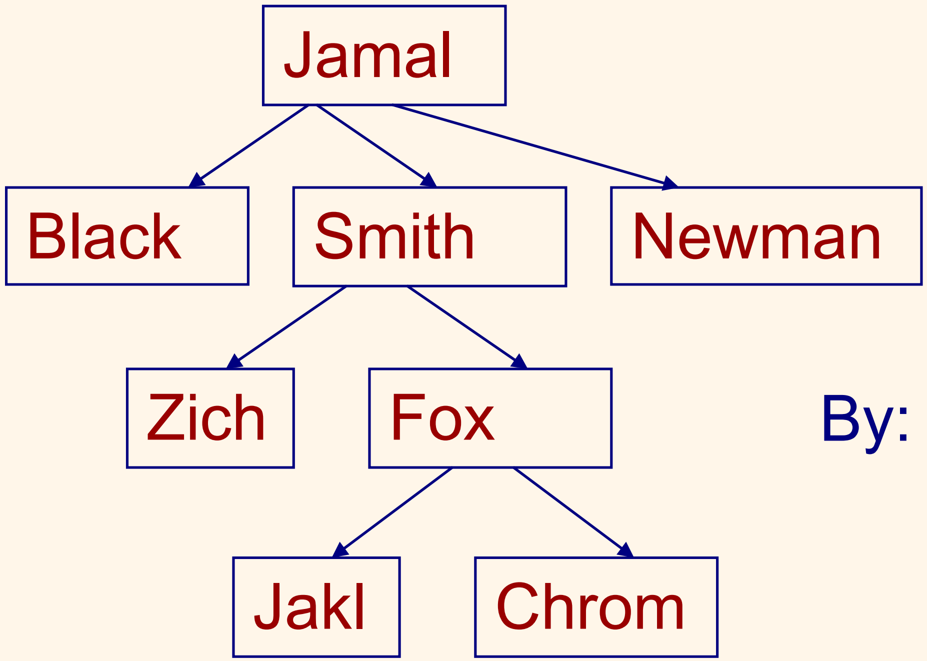
$R \cap \underline{E'} = R - E'$

Visualization:



Expressive power of DRC (A_R)

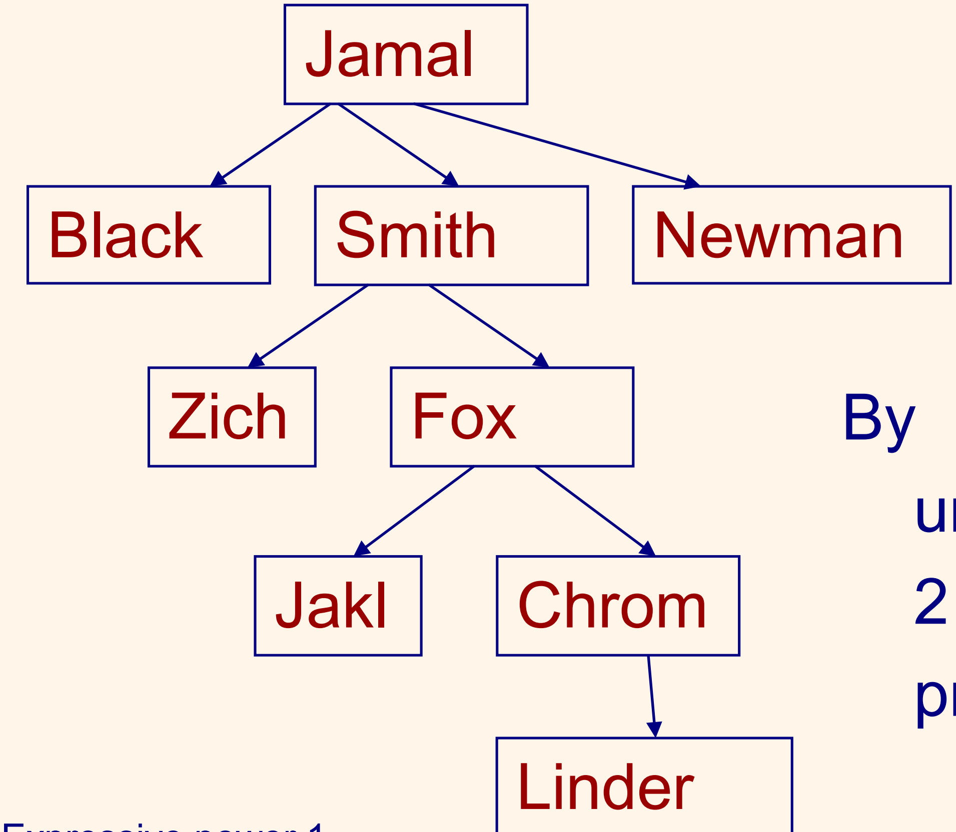
Q.: Find all subordinates of Smith.



By: union
1 join
projection

Expressive power of DRC (A_R)

Q.: Find all subordinates of Smith.



By

union

2 joins

projection

Query transitive closure (0)

Notions:

Df.: Binary relation R is transitive, if for each $(a,b)\in R$ and $(b,c)\in R$ also $(a,c)\in R$.

Df.: Transitive closure of the relation R, R^+, is least transitive relation containing R.

Database notions: relation schema R, relation R^*

Ex.: SUP-SUB(Superior, Subordinate) reflects transitive relationships on a conceptual level.

SUP-SUB* contains only direct relationships, e.g. (Jamal, Smith), (Fox, Chrom), …

Goal: calculate transitive closure of the relation SUP-SUB*

Assumption: We will consider relations, which are transitive on a conceptual level.

Query transitive closure (1)

Statement: Let R be a binary relation schema. Then there is no expression A_R, calculating for each relation R* its transitive closure R^+.

Proof:

1. Consider $\Sigma_s = \{a_1, a_2, \ldots, a_s\}$, $s \geq 1$, as a set of constants, for which no ordering exists, and

$$R_s = \{a_1a_2,\ a_2a_3, \ldots, a_{s-1}a_s\}$$

Remark: $R_s \Leftrightarrow$ graph $a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_s$, i.e., transitivity is defined by connectivity in a directed graph.

Remark: if an ordering $<$ is defined on Σ_s, then

$R_s^+ \cong (R_s[1] \times R_s[2])(1 < 2)$

2. We show, that for arbitrary expression E(R) there is *s* such, that $E(R_s) \neq R_s^+$.

Query transitive closure (2)

3. Lemma: Let E be a relational algebra expression. Then for sufficiently big *s*

   $E(R_s) \cong \{b_1,\ldots,b_k \mid \Gamma(b_1,\ldots,b_k)\}$,

where $k \geq 1$ and Γ is a formula in a disjunctive normal form.

Atomic formulas in Γ have a special form:

   $b_i = a_j$, $b_i \neq a_j$,

   $b_i = b_j + c$ or $b_i \neq b_j + c$, where c is (not necessarily positive) constant, where $b_j + c$ is abbreviation for “such a_m, for which $b_j = a_{m-c}$“

Domain of interpretation for assignments to variables b_j is Σ_s.

Remark: $b_i = b_j + c \Leftrightarrow b_i$ is behind b_j in the distance c nodes.

Query transitive closure (3)

4. Proof by contradiction.

There is E such, that $E(R) = R^+$ and any relation R, i.e. also $E(R_s) = R_s^+$ for sufficiently big *s*

- by lemma, $R_s^+ \cong \{b_1,b_2 \mid \Gamma(b_1,b_2)\}$

There are two cases:

(a) each clause z Γ contains an atom of form

$b_1 = a_i$, $b_2 = a_i$ or $b_1 = b_2 + c$ ($\Leftrightarrow b_2 = b_1 - c$)

Let $b_1b_2 = a_m a_{m+d}$,

where $m >$ arbitrary i and $d >$ arbitrary c

Query transitive closure (4)

$\Rightarrow$ $b_1 = a_m$ and $b_2 = a_{m+d}$ do not meet any clause from Γ.

$\Rightarrow$ contradiction ($a_m a_{m+d} \notin R_s^+$)

(b) in Γ there are clauses with atoms containing only $\neq$.

Let $b_1 b_2 = a_{m+d} a_m$, where neither $b_i \neq a_m$ nor $b_i \neq a_{m+d}$ is contained in Γ, and $d > 0$ is greater than arbitrary c in $b_1 \neq b_2 + c$ or $b_2 \neq b_1 + c$ in Γ (see construction of Γ)

$\Rightarrow$ $a_{m+d}\, a_m \in E(R_s)$ for sufficient s, but $\notin R_s^+$ $\Rightarrow$ contradiction

Thus: for arbitrary expression E, always there is *s* for which

$$E(R_s) \neq R_s^+$$

Query transitive closure (5)

5. Proof of lemma – by induction on the number of operators in E

I. $\emptyset$ of operators $\Rightarrow$ E $\equiv R_s$ or E is a constant relation

$\Rightarrow E \equiv \{b_1,b_2 \mid b_2 = b_1 + 1\}$ and

$E \equiv \{b_1 \mid b_1 = c_1 \vee b_1 = c_2 \vee \ldots \vee b_1 = c_m\}$,

respectively

II. a) $E \equiv E_1 \cup E_2$, E_1-E_2, $E_1 \times E_2$

$E_1 \cong \{b_1,\ldots,b_k \mid \Gamma_1(b_1,\ldots,b_k)\}$

$E_2 \cong \{b_1,\ldots,b_m \mid \Gamma_2(b_1,\ldots,b_m)\}$

$\Rightarrow$ for $\cup$ and - k=m and therefore

Query transitive closure (6)

$E \cong \{b_1,\ldots,b_k \mid \Gamma_1(b_1,\ldots,b_k) \vee \Gamma_2(b_1,\ldots,b_k)\}$,

$E \cong \{b_1,\ldots,b_k \mid \Gamma_1(b_1,\ldots,b_k) \wedge \neg\, \Gamma_2(b_1,\ldots,b_k)\}$, respectively.

$\Rightarrow$ for $\times$

$E \cong \{b_1,\ldots,b_k\ b_{k+1},\ldots,b_{k+m} \mid \Gamma_1(b_1,\ldots,b_k) \wedge \Gamma_2(\ b_{k+1},\ldots,b_{k+m})\}$

!! Then a transformation to DNF follows.

b) $E \equiv E_1(\varphi)$ a φ contains either $=$ or $\neq$

$\Rightarrow E \cong \{b_1,\ldots,b_k \mid \Gamma_1\ (b_1,\ldots,b_k) \wedge \varphi(b_1,\ldots,b_k)\}$

Query transitive closure (7)

c) $E \equiv E_1[S]$

We will consider a projection removing one attribute

$\Rightarrow$ It is about a sequence of permutations of variables and elimination of the last component.

The elimination of b_k leads to

$\{b_1,\ldots,b_{k-1} \mid \exists\, b_k\, \Gamma(b_1,\ldots,b_k)\}$, where Γ is in DNF

$\Rightarrow$ by a)

$\cup_{i=1..m}\{b_1,\ldots,b_{k-1} \mid \exists b_k\, \Gamma_i(b_1,\ldots,b_k)\}$

$\Rightarrow$ we will eliminate $\exists$ from one conjunct

- in Γ_i there are not $b_k=a_i$, $b_i=b_k+c$, and $b_k=b_i+c$

$\Rightarrow$ $\{b_1,\ldots,b_{k-1} \mid \underline{\Gamma}_i(b_1,\ldots,b_{k-1})\}$

where $\underline{\Gamma}_i$ does not contain $b_k \neq a_i$, $b_i \neq b_k+c$, or $b_k \neq b_i+c$

Query transitive closure (8)

- in Γ_i there is either $b_k=a_i$ or $b_i=b_k+c$ or $b_k=b_i+c$

  $\Rightarrow$ substitutions for b_k will take place.

  The results are adjusted to TRUE

  or FALSE

  or $b_t=b_j+g$

  and the following inequalities are added:

  $b_i \neq a_j$ for $s\text{-}c < j \leq s$,

  $b_i \neq a_j$ for $1 < j \leq c$, respectively

Transitive closure functionally

Df.: A composition R ° S of binary relations R, S defined on domain D is a binary relation

$$\{a,b \mid \exists c \in D, (a,c) \in R^* \wedge (c,b) \in S^* \}$$

Let f be a function assigning to a binary relation R a binary relation R´ (both relations are defined on D).

Df.: Let R be relational variable and f(R) relational expression. Then the least fixpoint (LFP) of the equation

$$R = f(R) \qquad (1)$$

is a relation R* such, that:

- $R^* = f(R^*)$ */fixpoint/*
- $S^* = f(S^*) \Rightarrow R^* \subseteq S^*$ */minimality/*

Df.: f is monotonic if for each two relations R^*_1 and R^*_2

$$R^*_1 \subseteq R^*_2 \Rightarrow f(R^*_1) \subseteq f(R^*_2)$$

Transitive closure functionally

Statement: f is monotonic if and only if

$$f(R_1 \cup R_2) \supseteq f(R_1) \cup f(R_2)$$

Df.: f is additive if and only if

$$f(R_1 \cup R_2) = f(R_1) \cup f(R_2)$$

Statement: Additive function is monotonic.

Theorem (Tarski): If f is monotonic, then the LFP of equation (1) exists.

LFP construction: For a finite relation R, we obtain LFP by repeating application of f.

Initialize R by $\emptyset$, then $f^{i-1}(\emptyset) \subseteq f^{i}(\emptyset)$.

Then there is $n_0 \geq 1$ such that

$$\emptyset \subset f(\emptyset) \subset f^1(\emptyset) \subset \ldots \subset f^{n0}(\emptyset) = f^{n0+1}(\emptyset)$$

Relation $f^{n0}(\emptyset)$ is the LFP of the equation (1).

Transitive closure functionally

Proof: By induction on *i,* it is shown, that relation $f^{n0}(\emptyset)$ is contained in each fixpoint of equation (1).

Statement: The transitive closure of a binary relation R* defined on D is the LFP of the equation

$$S = S \circ R^* \cup R^*$$

where S is a relational variable (binary, defined on D).

Proof: $f(S) = S \circ R^* \cup R^*$

$\Rightarrow$ $f^n(\emptyset) = \cup_{i=1..n} R^* \circ R^* \circ \ldots \circ R^*$

which leads to the transitive closure

$$\cup_{i = 1..\infty} R^* \circ R^* \circ \ldots \circ R^*$$

Transitive closure functionally

Ex.: Consider the relation schema

FLIGHTS(FROM, TO, DEPARTURE, ARRIVAL)

Task: to express CONNECTIONS with transfers

Solution: CONNECTIONS* is given as the LFP of equation

CONNECTIONS = FLIGHTS $\cup$ (FLIGHTS $\times$ CONNECTIONS) $(2=5 \wedge 4< 7)[1, 6, 3, 8]$

Statement: Each relational algebra expression not containing difference is additive in all its variables.

Transitive closure functionally

Remarks:

- Non-monotonic expression can have a LFP,
- Not every expression involving the difference operator fails to be monotone.

Df.: A minimální fixpoint (MFP) of equation (1) is such fixpoint R*, that there is no other fixpoint, which is a proper subset of R*.

$\Rightarrow$ $\exists$ LFP, then it is the only one MFP.

If there is more MFPs, then they are mutually non-comparable and no LFP exists.

Databases intensionally

Ex.: Consider predicates

F(x,y) x is a father of y

M(x) x is a man

S(x,y) x is a sibling of y

B(x,y) x is a brother of y

Extensional database (EDB):

F(James, Paul) (1)

F(James, Jerry) (2)

F(Jerry, Veronika) (3)

Databases intensionally

Intensional database (IDB):

M(x):- F(x,y) (4)

S(y,w) :- F(x,y), F(x,w) (5)

B(x,y) :- S(x,y),M(x) (6)

Queries:

Q_1: Has Paul a brother?

Q_2: Find all (x,y), where x is a brother of y.

Q_3: Find all (x,y), where x is a sibling of y.

Remark: EDB + IDB create a logical program (LP)

Solution of LP by the resolution method

EDB as a set of facts

IDB as a set Horn clauses:

$F(x,y) \Rightarrow M(x)$

$F(x,y) \wedge F(x,w) \Rightarrow S(y,w)$

$S(x,y) \wedge M(x) \Rightarrow B(x,y)$

Assumption: Formulas in IDB are universally quantified, e.g.,

$\forall x \forall y \forall w \, (F(x,y) \wedge F(x,w) \Rightarrow S(y,w))$

Reformulation of Q_1 : $\exists z \, B(z,Paul)$

Solution of LP by the resolution method

Resolution method:

- Uses a proof by contradiction
- inference is equivalent to deriving an empty clause (NIL); in other cases it is not possible to say, whether clauses is derivable

Principle: $A_1 \vee \ldots \vee A_i \vee B_1$ $\qquad$ $C_1 \vee \ldots \vee C_j \vee \neg B_2$

- Unification: by substitutions we try to achieve to do B_1 and B_2 complementary.
- Deriving a resolvent: If after unification the input has a form $\underline{A}_1 \vee \ldots \vee \underline{A}_i \vee B$ and $\underline{C}_1 \vee \ldots \vee \underline{C}_j \vee \neg B$, then it is possible to derive $\underline{A}_1 \vee \ldots \vee \underline{A}_i \vee \underline{C}_1 \vee \ldots \vee \underline{C}_j$

Solution of LP by the resolution method

Statement: A resolvent is (un)satisfiable, if input clauses were (un)satisfiable.

The procedure goal: to derive NIL

A is a logical conseqence of W

Justification: $W=\{A_1,...,A_m\}$, then $W \models A$ if and only if

$A_1 \wedge ... \wedge A_m \wedge \neg A$ is unsatisfiable

By the Gödel theorem, unsatisfiability is partially decidable, i.e. there is a procedure P such that for each formula φ the following holds:

if φ is unsatisfiable, then $P(\varphi)$ terminates and announces it,

if φ is satisfiable, then $P(\varphi)$ either terminates and announces it, or fails to terminate.

Solution of LP by the resolution method

Ex.: We add to EDB and IDB $\neg$B(z,Paul) (7)

and run the resolution method:

(8) S(Jerry,w) :- F(James,w)	from (2),(3)
(9) S(Jerry,Paul)	from (8),(1)
(10) M(Jerry)	from (3),(4)
(11) B(Jerry,y) :- S(Jerry,y)	from (10),(6)
(12) B(Jerry,Paul)	from (11),(9)
(13) NIL	from (12),(7)

Terminology and constraints

- terms: variables or constants
- facts are atomic formulas containing only constants
- rules are Horn clauses

  L_0:- $L_1,\ldots,L_n$

  where L_i are atomic (positive) formulas
- atomic formulas or negations of atomic formulas are called literals.
- positive and negative literals
- facts are called basic literals

Terminology and constraints

- structure of rules:

  L_0 head of a rule

  $L_1, \ldots, L_n$ body of a rule

Remark: Facts and literals are also Horn clauses.

DATALOG - syntax and semantics (1)

1. Datalog program is a collection of facts and rules.
2. Three kinds of predicate symbols:
   - $R_i \in R$
   - S_i ... virtual relations
   - built-in predicates $\leq, \geq, \neq, <, >, =$

   R_i and S_i are called ordinary.

   Remark: $\neq$ will not conceived as a negation (we will compare only bound variables)
3. Semantics of logic programs can be built by at least in three different ways:
   - proof theoretic,
   - model theoretic,
   - with fixpoints.

DATALOG - syntax and semantics (2)

- proof theoretic approach

Method: interpretation of rules as axioms usable to a proof, i.e. we make substitutions in body of rules and derive new facts from heads of rules. In the case of Datalog, it is possible to obtain *just all* derivable facts.

- model theoretic approach

Method: to predicate symbols we associate relations (a logical model) which satisfy the rules.

Ex.: Consider a logical program LP

IDB: P(x) :- Q(x)
Q(x) :- R(x),

i.e. Q and P denote virtual relations.

DATALOG - syntax and semantics (3)

- Let: R(1) Q(1) P(1)
  Q(2) P(2) M_1
  P(3)

Relations P*, Q*, R* make a model M_1 of the logical program LP.

- Let: R(1) (and other facts have value FALSE). Then relations P*, Q*, R* are not a model of the LP.
- Let: R(1) Q(1) P(1) M_2

then relations P*, Q*, R* make a model M_2 of the LP.

Let EDB: R(1), i.e. relational DB is given as

$R^* = \{(1)\}$.

then M_1 and M_2 are with the given DB consistent.

DATALOG - syntax and semantics (4)

- M_2 is even a minimal model, i.e. when we change anything there, we destroy consistency.
- M_1 does not make a minimal model.

Remark: with both semantics we obtain the same result.

Disadvantages of both approaches: non-effective algorithms in the case, when EDB is given by database relations.

DATALOG - dependency graph (1)

- with fixpoints

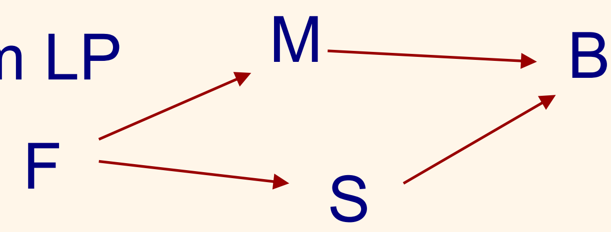
Method: evaluating algorithm+relational DB machine

Df.: a dependency graph of a logical program LP

nodes: predicates from R and IDB

edges: (U,V) is an edge, if there is a rule

V :- … U ...



Ex.: extension of the original example

M(x):- F(x,y)

S'(y,w) :- F(x,y), F(x,w), $y \neq w$

B(x,y) :- S'(x,y), M(x)

C(x,y) :- F(x_1,x), F(x_2,y), S'(x_1,x_2)

C(x,y) :- F(x_1,x), F(x_2,y), C(x_1,x_2)

DATALOG - dependency graph (2)

R(x,y) :- S‘(x,y)

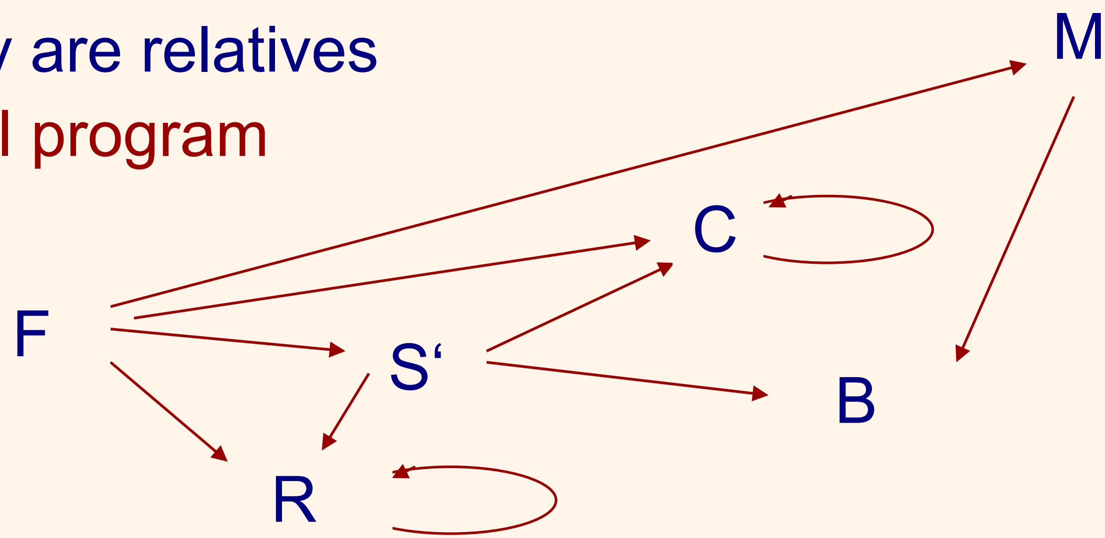
R(x,y) :- R(x,z), F(z,y)

R(x,y) :- R(z,y), F(z,x)

where C(x,y) … x is a cousin of y, i.e. their fathers are brothers

R(x,y) … x and y are relatives

recursive datalogical program



DATALOG - dependency graph (3)

R, C … recursive predicates

Df.: A logical program is recursive if there is a cycle in its dependency graph.

DATALOG - safe rules

Df.: safe rule

A variable x occurring in a rule is limited, if it occurs in the body of literal L of the same rule, where:

- L is given by an ordinary predicate, or
- L is of form x = a or a = x, or
- L is of form x=y or y=x and y is limited.

A rule is safe, if all its variables are limited.

Ex.: safety of rules

IS_GREATER_THAN(x,y) :- $x > y$

FRIENDS(x,y) :- M(x)

S'(y,w) :- F(x,y), F(x,w), $y \neq w$

Non-recursive DATALOG

- Its dependency graph is acyclic.
- There is a topological ordering of nodes such, that $R_i \rightarrow R_j$ implies $i < j$.

Remark: ordering is not given unambiguously

Ex.: ordering F - M - S - B

Non-recursive DATALOG

Principle of the algorithm (for one virtual relation):

(1) $U(x_1,\ldots,x_k)$:- $V_1(x_{i1},\ldots,x_{ik}),\ldots, V_s(x_{j1},\ldots,x_{js})$

transform to joins and selection

(2) for U it is performed

apply a projection on the result

(3) Steps (1), (2) are performed for all rules with U in their heads and for partial results

apply a union

Remark: Due to the acyclicity and topological ordering, the steps (1), (2) can be always applied for a rule.

Non-recursive DATALOG

Convention: variable x $\rightarrow$ attribute X

Rule rewriting:

- C(x,y) :- F(x_1,x), F(x_2,y), S'(x_1,x_2)

  1. step:

  AUX(X1,X,X2,Y) = F(X1,X) * F(X2,Y) * S'(X1,X2)

  2. step:

  C(X,Y) = AUX[X,Y]

- for S'

  S'(Y,W) = (F(X,Y) * F(X,W)) (Y$\neq$ W)[Y,W]

Non-recursive DATALOG

Other possibilities:

- V(x,y) :- P(a,x), R(x,x,z), U(y,z)

  1. and 2. step:

  V(.,.) = (P(1=a)[2] * R(1=2)[1,3] * U)[.,.]

Problem: In the rule head, constants, the same variables, and different orders of variables can occur.

A request on a rectification, i.e., a transformation of rules in such way, that heads with the same predicate symbol have a tuple of the same variables.

Non-recursive DATALOG

Ex.: P(a,x,y) :- R(x,y)

P(x,y,x) :- R(y,x)

We introduce u, v, w and do the substitutions:

P(u,v,w) :- R(x,y), u = a, in = x, w = y

P(u,v,w) :- R(y,x), u = x, in = y, w = x

$\Rightarrow$ P(u,v,w) :- R(v,w), u = a,

P(u,v,w) :- R(v,u), w = u

Lemma:

(1) If the rule is safe, then after rectification too.

(2) The original and rectified rule are equivalent, i.e., after its evaluation we obtain the same relation.

Non-recursive DATALOG

Statement: The evaluated program provides for each predicate from IDB a set of facts, which forms

1. the set of just those facts, provable from EDB by application of rules from IDB.
2. a minimal model for EDB + IDB .

Proof: by induction in the rules ordering.