Datalog Evaluation

Serge Abiteboul

INRIA

5 mai 2009

Datalog

READ CHAPTER 13

lots of research in the late 80'th

top-down or bottom-up evaluation

direct evaluation vs. compilation into a more efficient program

no product

some influence on logic programming

HERE

1. semi-naive bottom-up evaluation
2. top-down : QSQ
3. bottom-up : Magic

Reverse-Same-Generation

$$rsg(x,y) \leftarrow flat(x,y)$$
$$rsg(x,y) \leftarrow up(x,x1), rsg(y1,x1), down(y1,y)$$

up

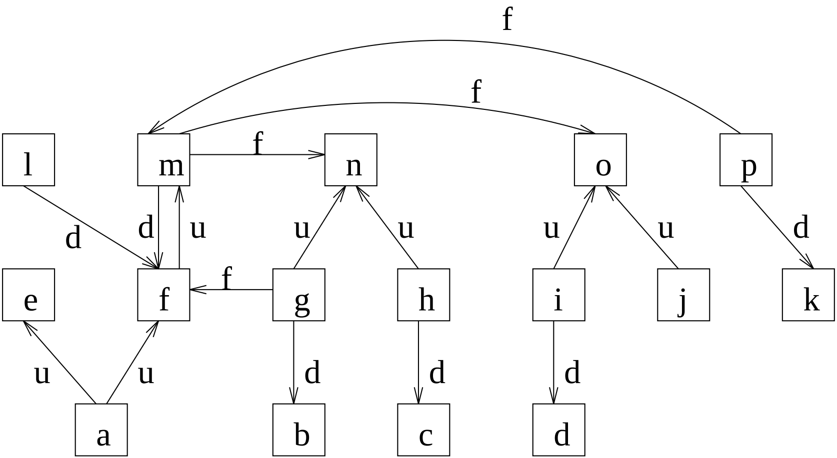
a	e
a	f
f	m
g	n
h	n
i	o
j	o

flat

g	f
m	n
m	o
p	m

down

l	f
m	f
g	b
h	c
i	d
p	k


f
f
l
m
f
n
o
p
d
d
u
u
u
u
d
e
f
f
g
h
i
j
k
u
u
d
d
d
a
b
c
d

Naive algorithm

$$
\begin{aligned}
\text{level } 0 &: \emptyset \\
\text{level } 1 &: \{(g,f),(m,n),(m,o),(p,m)\} \\
\text{level } 2 &: \{\text{level } 1\} \cup \{(a,b),(h,f),(i,f),(j,f),(f,k)\} \\
\text{level } 3 &: \{\text{level } 2\} \cup \{(a,c),(a,d)\} \\
\text{level } 4 &: \{\text{level } 3\}
\end{aligned}
$$

$$
\begin{aligned}
& rsg := \emptyset \\
& repeat \\
& \qquad rsg := rsg \cup flat \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times rsg \times down))) \\
& until \quad fixpoint
\end{aligned}
$$

$$rsg^{i+1} = rsg^{i} \cup flat \cup \pi_{16}(\sigma_{2=4}(\sigma_{3=5}(up \times rsg^{i} \times down)))$$

redundant computation

each layer recomputes all elements of the previous layer

monotonicity

Semi-naive

Focus on the new facts generated at each level

RSG'

$$\begin{array}{lcl} \Delta^1_{rsg}(x,y) & \leftarrow & flat(x,y) \\ \Delta^{i+1}_{rsg}(x,y) & \leftarrow & up(x,x1), \Delta^i_{rsg}(y1,x1), down(y1,y) \end{array}$$

not recursive

not a datalog program

for each input **I**,

$$rsg^{i+1} - rsg^i \subseteq \delta^{i+1}_{rsg} \subseteq rsg^{i+1}$$

RSG(**I**)(rsg) $= \cup_{1 \leq i}(\delta^i_{rsg})$.

less redundancy

An improvement

$\delta_{rsg}^{i+1} \neq rsg^{i+1} - rsg^{i}$

e.g., $(g, f) \in \delta_{rsg}^{2}$, not in $rsg^2 - rsg^1$

use $rsg^{i} - rsg^{i-1}$ instead of place of Δ_{rsg}^{i} in the second "rule" of RSG′

Non linear rules : e.g., ancestor

$$\begin{aligned} anc(x, y) &\leftarrow par(x, y) \\ anc(x, y) &\leftarrow anc(x, z), anc(z, y) \end{aligned}$$

semi-naive evaluation

$$\begin{aligned} \Delta_{anc}^{1}(x, y) &\leftarrow par(x, y) \\ \Delta_{anc}^{i+1}(x, y) &\leftarrow \Delta_{anc}^{i}(x, z), anc^{i}(z, y) \\ \Delta_{anc}^{i+1}(x, y) &\leftarrow anc^{i}(x, z), \Delta_{anc}^{i}(z, y) \end{aligned}$$

still some redundancy : use

$$\begin{aligned} temp^{i+1}(x, y) &\leftarrow \Delta_{anc}^{i}(x, z), anc^{i-1}(z, y) \\ temp^{i+1}(x, y) &\leftarrow anc^{i}(x, z), \Delta_{anc}^{i}(z, y), \end{aligned}$$

Top-down technique : Query-Sub-Query

Program + query

$$\begin{array}{lcl} rsg(x,y) & \leftarrow & flat(x,y) \\ rsg(x,y) & \leftarrow & up(x,x1), rsg(y1,x1), down(y1,y) \\ query(y) & \leftarrow & rsg(a,y) \end{array}$$

Focus on relevant data

avoid deriving unnecessary tuples

A is relevant if there is a proof tree for *query* in which the fact occurs

We will do that (not perfectly) using four ingredients

(a) SLD-resolution but set-at-a-time using relational algebra
(b) start with query constants and "push" constants from goals to subgoals (in the style of to pushing selections into joins)
(c) Use "sideways information passing" to pass constant binding information from one atom to the next in subgoals
(d) Use an efficient global flow-of-control strategy

Technique

adornment and adorned rules

supplementary relations and QSQ templates

the kernel of the technique

global control strategies

Adornment

in $\leftarrow rsg(a, y)$, we have a binding for the first argument of rsg

we denote it rsg^{bf}

later on, we need rsg^{fb}

based on the evaluation of subqueries $(rsg^{fb}, \{\langle e\rangle, \langle f\rangle\})$
in general (R^{γ}, J) where

1. R is a predicate
2. γ an adornment
3. J provides bindings
4. completion for t in J
5. answer of the subquery : set of completions

Sideways information passing

from relational optimization

- evaluation of joins
- P(a,v) join Q(b,w,x) join R(v,w,y)
- evaluate first P(a,v) → obtains some v's
- evaluate then R(v,w,y) restricted by v's → obtains some w's
- finally evaluate Q(b,w,x) restricted by w's

Adorned rule

$R(x, y, z) \leftarrow R_1(x, u, v), R_2(u, w, w, z), R_3(v, w, y, a)$

a subquery involving R^{bfb} with left-to-right evaluation (reorder if necessary)

$R^{bfb}(x, y, z) \leftarrow$
$R_1^{bff}(x, u, v), R_2^{bffb}(u, w, w, z), R_3^{bbfb}(v, w, y, a)$

given head adornment and an ordering of body, algo for adorning

Supplementary relations and QSQ templates

Data structure to store information during evaluation
$n+1$ supplementary relations for a rule with n atoms

$$\begin{array}{lllll} R^{bfb}(x,y,z) \leftarrow & R_1^{bff}(x,u,v), & R_2^{bffb}(u,w,w,z), & R_3^{bbfb}(v,w,y,a) & \\ & \uparrow & \uparrow & \uparrow & \uparrow \\ & sup_0[x,z] & sup_1[x,z,u,v] & sup_2[x,z,v,w] & sup_3[x,y,z] \end{array}$$

variables serve as attribute names in the supplementary relations
QSQ template for the adorned rule : $sup_0, sup_1, sup_2, sup_3$

Kernel of QSQ evaluation

input : program + query

construct adorned rules for the query :

$$
\begin{array}{ll}
(1) & rsg^{bf}(x,y) \leftarrow flat(x,y) \\
(2) & rsg^{fb}(x,y) \leftarrow flat(x,y) \\
(3) & rsg^{bf}(x,y) \leftarrow up(x,x1), rsg^{fb}(y1,x1), down(y1,y) \\
(4) & rsg^{fb}(x,y) \leftarrow down(y1,y), rsg^{bf}(y1,x1), up(x,x1)
\end{array}
$$

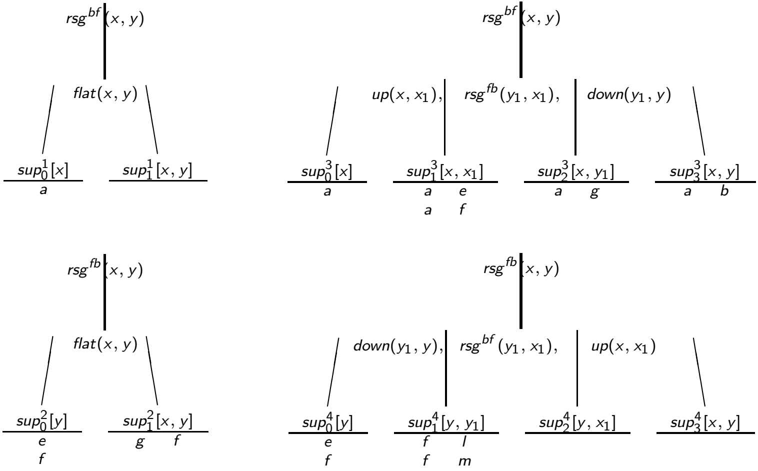
(Note the ordering to pass bindings in (4))

construct QSQ template for each adorned rule i
and the supplementary relation variables sup_j^i

construct also : for each idb R and adornment γ

1. the variable ans_R^{γ} of arity arity(R)
2. the variable $input\_R^{\gamma}$ of arity $bound(R,\gamma)$

initialization : use the query



$input\_rsg^{bf}$
a

$input\_rsg^{fb}$
e
f

ans_rsg^{bf}	
a	b

ans_rsg^{fb}	
g	f

Kinds of steps

A : from $input\_R^\gamma$ to sup_0^i
move new tuples in $input\_R^\gamma$ to 0^{th} supplementary relations

B : from sup_j^i to sup_{j+1}^i
Pass new tuples from one supplementary relation to the next

C : from ans_R^γ to sup_j^i
Use new *idb* tuples (from ans_R^γ) to generate new supplementary relation tuple

D : from sup_n^i to ans_R^γ
Process tuples in the final supplementary relation of a rule

Example

$\langle a \rangle$	*into*	$input\_rsg^{bf}$	init
$\langle a \rangle$		sup_0^1, sup_0^3	A
$\langle a, e \rangle, \langle a, f \rangle$		sup_1^3	B
$\langle e \rangle, \langle f \rangle$		$input\_rsg^{fb}$	B
$\langle g, f \rangle$		ans_rsg^{fb}	B,D and 2nd rule
$\langle a, b \rangle$		ans_rsg^{bf}	C,B,D and 3rd rule

Global control strategies

- basic one : apply steps (A) through (D) until a fixpoint is reached
- QSQR (query-subquery-recursive)
  - in step (B) : pass from supplementary relation sup^i_{j-1} across an *idb* predicate R^γ to supplementary relation sup^i_j
  - introduce new tuples into sup^i_j and into $input\_R^\gamma$
  - perform a recursive call to determine the R^γ-values corresponding to the new tuples added to $input\_R^\gamma$, before considering the new tuples placed into sup^i_j
  - see algorithm in the book

Merci