CSE 544
Relational Calculus

Lecture #2

January 11th, 2011

Announcements

- HW1 is posted
- First paper review due next week
- Friday: project groups are due

Announcements

- Guest lecture on Tuesday, 1/18, by Bill Howe
  SQLShare: Smart Services for Ad Hoc Databases

- Need to makeup **one** lecture this/next week. When ?
  - Friday 1/14, 12:30-2pm ?
  - Wednesday, 1/19, morning (9…12) ?
  - Friday, 1/21, late morning (11...1pm) ?

Today's Agenda

Relational calculus:

- Domain Relational Calculus
- Non-recursive datalog (a reasonable abstraction of SQL)
- Relational algebra

They are equivalent and why we care

Domain Relational Calculus

Given:

- A vocabulary: $R_1, \ldots, R_k$
- An arity, $ar(R_i)$, for each $i=1,\ldots,k$
- An infinite supply of variables $x_1, x_2, x_3, \ldots$
- Constants: $c_1, c_2, c_3, \ldots$

Domain Relational Calculus

Terms t and Formulas φ are:

$$
\begin{aligned}
t &::= x \;\mid\; a \quad \text{/* variable or constant */} \\
\varphi &::= R(t_1, \ldots, t_k) \;\mid\; t_i = t_j \\
&\quad \mid\; \varphi \wedge \varphi' \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi' \\
&\quad \mid\; \forall x.\varphi \;\mid\; \exists x.\varphi
\end{aligned}
$$

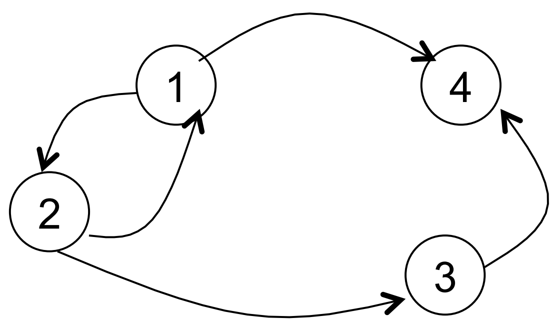
A query Q is any formula φ, usually written as:

$$\boxed{Q = \{\, \mathbf{x} \mid \varphi \,\}}$$

where $\mathbf{x} = (x_1, x_2, \ldots)$ are the free variables in φ.

Example: Querying a Graph

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4

What do these queries return ?

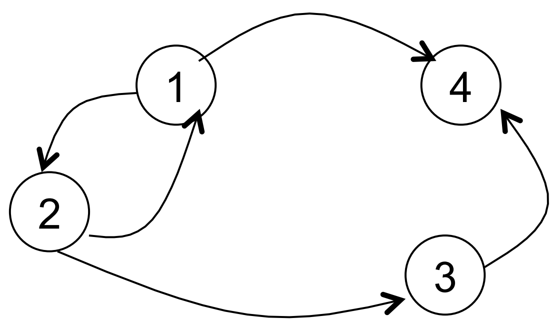
$\{x \mid \exists y.R(x,y)\}$

$\{x \mid \exists y.\exists z.\exists u.(R(x,y) \land R(y,z) \land R(z,u))\}$

$\{(x,y) \mid \forall z.(R(x,z) \rightarrow R(y,z))\}$

Example: Querying a Graph

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4

What do these queries return ?

$\{x \mid \exists y.R(x,y)\}$

Nodes that have at least one child: {1,2,3}

$\{x \mid \exists y.\exists z.\exists u.(R(x,y) \wedge R(y,z) \wedge R(z,u))$

Nodes that have a great-grand-child: {1,2}

$\{(x,y) \mid \forall z.(R(x,z) \rightarrow R(y,z)\}$

Every child of x is a child of y:
{(1,1),(2,2),(3,1),(3,3),(4,1), (4,2), (4,3)}

Semantics of a Formula

- A *database instance* (or a *model*) is
  $\mathbf{D} = (D, R_1^D, \ldots, R_k^D)$
  - D = a set, called domain, or universe
  - $R_i^D \subseteq D \times D \times \ldots \times D$, ($ar(R_i)$ times)
    $i = 1,\ldots,k$
- A *valuation* is a function $s : \{x_1, x_2, \ldots\} \rightarrow D$
- The semantics of a *formula* φ says when
  $\mathbf{D} \models \varphi[s]$ holds

The domain D is not part of the database,
yet we need it to define the semantics. Where ?

Semantics of $\mathbf{D} \vDash \varphi[s]$

$\mathbf{D} \vDash (R(t_1, ..., t_n))\ [s]$	If $(s(t_1), ..., s(t_n)) \in R^D$
$\mathbf{D} \vDash (t = t')\ [s]$	If $s(t) = s(t')$
$\mathbf{D} \vDash (\varphi \wedge \varphi')\ [s]$	If $\mathbf{D} \vDash \varphi[s]$ and $\mathbf{D} \vDash (\varphi')\ [s]$
$\mathbf{D} \vDash (\varphi \vee \varphi')\ [s]$	If $\mathbf{D} \vDash \varphi[s]$ or $\mathbf{D} \vDash \varphi'[s]$
$\mathbf{D} \vDash (\neg\varphi')\ [s]$	If it is not the case $\mathbf{D} \vDash \varphi[s]$

Semantics of $\mathbf{D} \models \varphi[s]$

Notation: $s_{a/x}$ = the valuation s modified to map x to a

$\boxed{\mathbf{D} \models (\forall x.\varphi)\ [s]}$

If for every constant a in D,
$\mathbf{D} \models \varphi[s_{a/x}]$

$\boxed{\mathbf{D} \models (\exists x.\varphi)\ [s]}$

If for some constant a in D,
$\mathbf{D} \models \varphi[s_{a/x}]$

Semantics of $\mathbf{D} \vDash \varphi[s]$

Notation: $s_{a/x}$ = the valuation s modified to map x to a

$\boxed{\mathbf{D} \vDash (\forall x.\varphi)\ [s]}$

If for every constant a in D,
$\mathbf{D} \vDash \varphi[s_{a/x}]$

$\boxed{\mathbf{D} \vDash (\exists x.\varphi)\ [s]}$

If for some constant a in D,
$\mathbf{D} \vDash \varphi[s_{a/x}]$

Note: here we need the domain D

Semantics of a Query

The semantics of a *query* $Q = \{ \mathbf{x} \mid \varphi \}$ is

$$\boxed{Q(\mathbf{D}) = \{ s(\mathbf{x}) \mid \text{for all } s \text{ such that } \mathbf{D} \models \varphi[s] \}}$$

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x: \quad \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x: \quad \exists y. \exists z. \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x: \quad \exists y.\ \mathrm{Frequents}(x, y) \wedge \forall z.(\mathrm{Serves}(y,z) \Rightarrow \mathrm{Likes}(x,z))$$

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x: \quad \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x:\ \forall y.\text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

What does this query compute ?

$$x:\ \forall y.\text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find drinkers that frequent <u>only</u> bars that serves <u>only</u> beer they like.

Summary of Relational Calculus

- Same as First Order Logic
- See book for the two variants:
  - Domain relational calculus (what we discussed)
  - Tuple relational calculus
- This is a powerful, concise language

Datalog Rule

Head

Body

$$\boxed{\underbrace{Q(\mathbf{x})}_{\text{Head}} \quad :- \quad \underbrace{L_1, \quad L_2, \quad \ldots, \quad L_k}_{\text{Body}}}$$

L_i = a *literal*, or *atom*, or *subgoal;* can be one of:

- $R(x_1, x_2, \ldots)$ = relational atom
- $u = v$, or $u \neq v$, where u, v = variables or constants

S = a new symbol

$\mathbf{x} = (x_1, x_2, \ldots)$ are *head variables*

Example: $\boxed{Q(x) :- Likes(x,y),Likes(x,z),y\neq z}$

Semantics of Datalog Rule

$Q(\mathbf{x}) \text{ :- } L_1, L_2, \ldots, L_k$ $=$ $Q = \{\mathbf{x} \mid \exists y_1.\exists y_2 \ldots L_1 \wedge L_2 \wedge \ldots \wedge L_k\}$

$y_1, y_2, \ldots$ are all non-head variables

The semantics is:

$Q(\mathbf{D}) \text{ :- } \{ s(x) \mid s = \text{valuation s.t. } D \vDash L_1[s], \ldots, D \vDash L_k[s] \}$

Note: a datalog rule is also called a *conjunctive query*

Non-Recursive Datalog

$S_1(\mathbf{x}_1)$:- $body_1$
$S_2(\mathbf{x}_2)$:- $body_2$
$S_3(\mathbf{x}_3)$:- $body_3$
. . .

Each symbol S_k may appear in $body_{k+1}$, $body_{k+2}$, …, but may not appear in $body_1$,…,$body_k$.

Example:

What does Q compute ?

```
A(x,y) :- R(x,z),R(z,y)
B(x,y) :- A(x,z),A(z,y)
C(x,y) :- B(x,z),B(z,y)
Q(x,y) :- C(x,z),C(z,y)
```

Non-Recursive Datalog

$S_1(\mathbf{x}_1)$:- $body_1$
$S_2(\mathbf{x}_2)$:- $body_2$
$S_3(\mathbf{x}_3)$:- $body_3$
. . .

Each symbol S_k may appear in $body_{k+1}$, $body_{k+2}$, …, but may not appear in $body_1$,…,$body_k$.

Example:

```
A(x,y) :- R(x,z),R(z,y)
B(x,y) :- A(x,z),A(z,y)
C(x,y) :- B(x,z),B(z,y)
Q(x,y) :- C(x,z),C(z,y)
```

What does Q compute ?

A: all pairs (x,y) connected by a path of length 16.

Comparison

Is non-recursive datalog equivalent
to the Relational Calculus ?

Comparison

Is non-recursive datalog equivalent
to the Relational Calculus ?

What about writing this query in non-recursive datalog ?

$$Q = \{x \mid \exists y.\ \text{Frequents}(x,y) \wedge \text{Serves}(y,\text{'Bud'}) \vee \exists z.\ \text{Frequents}(x,z) \wedge \text{Serves}(z,\text{'Miller'})\}$$

Comparison

Is non-recursive datalog equivalent to the Relational Calculus ?

What about writing this query in non-recursive datalog ?

$$Q = \{x \mid \exists y.\ \text{Frequents}(x,y) \wedge \text{Serves}(y,\text{'Bud'}) \vee \exists z.\ \text{Frequents}(x,z) \wedge \text{Serves}(z,\text{'Miller'})\}$$

```
Q(x) :- Frequents(x,y), Serves(y,'Bud')
Q(x) :- Frequents(x,z), Serves(z,'Miller')
```

Comparison

Is non-recursive datalog equivalent to the Relational Calculus ?

Now what about writing this query in non-recursive datalog ?

$$\boxed{Q = \{x \mid \exists y.\ \text{Frequents}(x,y) \wedge \neg \exists z.\text{Likes}(x,z) \wedge \text{Serves}(y,z)}$$

Comparison

Is non-recursive datalog equivalent
to the Relational Calculus ?

Now what about writing this query in non-recursive datalog ?

$$\boxed{Q = \{x \mid \exists y.\ \text{Frequents}(x,y) \wedge \neg \exists z.\text{Likes}(x,z) \wedge \text{Serves}(y,z)}$$

Impossible ! Proof on next slides.

Monotone Queries

- Given two database instances
  $\mathbf{D} = (D, R_1, \ldots, R_k)$ and $\mathbf{D'} = (D', R'_1, \ldots, R'_k)$
  we write $\mathbf{D} \subseteq \mathbf{D}'$ if $D \subseteq D'$, $R_1 \subseteq R'_1, \ldots, R_k \subseteq R'_k$.

- In other words, **D'** is obtained by adding tuples to **D**.

- We say that Q is ***monotone***
  if whenever $\mathbf{D} \subseteq \mathbf{D'}$, $Q(\mathbf{D}) \subseteq Q(\mathbf{D'})$.

Monotone Queries

Fact. Every datalog rule is monotone.

Why ?

Fact. Every datalog program is monotone.

Why ?

Fact. This query is not monotone:
$Q = \{x \mid \exists y.\ Likes(x,y) \wedge \neg \exists z. Frequents(x,z) \wedge Serves(z,y)$

Why ?

Monotone Queries

Fact. Every datalog rule is monotone.

Why ?

Recall the semantics: $Q(\mathbf{D}) = \{ s(x) \mid \mathbf{D} \models L_1[s],\ldots,\mathbf{D} \models L_k[s] \}$

If $\mathbf{D} \models L_i[s]$ then $\mathbf{D'} \models L_i[s]$ hence $Q(\mathbf{D}) \subseteq Q(\mathbf{D'})$.

Fact. Every datalog program is monotone.

Why ?

Fact. This query is not monotone:
$Q = \{x \mid \exists y.\ Likes(x,y) \wedge \neg\exists z.Frequents(x,z) \wedge Serves(z,y)$

Why ?

Monotone Queries

Fact. Every datalog rule is monotone. Why ?

Recall the semantics: $Q(\mathbf{D}) = \{ s(x) \mid \mathbf{D} \vDash L_1[s],\ldots,\mathbf{D} \vDash L_k[s] \}$
If $\mathbf{D} \vDash L_i[s]$ then $\mathbf{D'} \vDash L_i[s]$ hence $Q(\mathbf{D}) \subseteq Q(\mathbf{D'})$.

Fact. Every datalog program is monotone. Why ?

By induction on the number of rules

Fact. This query is not monotone:
$Q = \{x \mid \exists y.\ \text{Likes}(x,y) \wedge \neg \exists z.\text{Frequents}(x,z) \wedge \text{Serves}(z,y)$

Why ?

Monotone Queries

Fact. Every datalog rule is monotone. Why ?

Recall the semantics: $Q(\mathbf{D}) = \{\ s(x) \mid \mathbf{D} \vDash L_1[s],\ldots,\mathbf{D} \vDash L_k[s]\ \}$
If $\mathbf{D} \vDash L_i[s]$ then $\mathbf{D'} \vDash L_i[s]$ hence $Q(\mathbf{D}) \subseteq Q(\mathbf{D'})$.

Fact. Every datalog program is monotone. Why ?

By induction on the number of rules

Fact. This query is not monotone:
$Q = \{x \mid \exists y.\ Likes(x,y) \wedge \neg\exists z. Frequents(x,z) \wedge Serves(z,y)$ Why ?

By adding a tuple to Frequents we may remove some answer

Monotone Queries

Recall the relational calculus:

$t ::= x \mid a$ /* variable or constant */

$\varphi ::= R(t_1, \ldots, t_k) \mid t_i = t_j$

$\quad\mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \neg\varphi'$

$\quad\mid \forall x.\varphi \mid \exists x.\varphi$

What fragment of the relational calculus is monotone ?

Monotone Queries

Recall the relational calculus:

$$t ::= x \mid a \quad \text{/* variable or constant */}$$

$$\varphi ::= R(t_1, \ldots, t_k) \mid t_i = t_j \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \neg \varphi' \mid \forall x.\varphi \mid \exists x.\varphi$$

What fragment of the relational calculus is monotone ?

$$t ::= x \mid a \quad \text{/* variable or constant */}$$

$$\varphi ::= R(t_1, \ldots, t_k) \mid t_i = t_j \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \exists x.\varphi$$

Non-Recursive Datalog$^{\neg}$

$$Q(\mathbf{x}) \ :- \ L_1, \ L_2, \ \ldots, \ L_k$$

L_i = may also be $\neg R(x1, x2, \ldots)$

- $R(x1, x2, \ldots)$ = relational atom
- $u = v$, or $u \neq v$, where u, v = variables or constants

Non-Recursive Datalog$^{\neg}$

Example:

$$Q = \{x \mid \exists y.\ \text{Likes}(x,y) \wedge \neg \exists z.\text{Frequents}(x,z) \wedge \text{Serves}(z,y)$$

Non-Recursive Datalog$^{\neg}$

Example:

$$Q = \{x \mid \exists y.\ Likes(x,y) \wedge \neg \exists z.Frequents(x,z) \wedge Serves(z,y)\}$$

```
FS(x,y) :- Frequents(x,z),Serves(z,y)
Q(x)   :- Likes(x,y), ¬FS(x,y)
```

Rel. Calculus → non-rec. Datalog$^{\neg}$

Theorem. Every query in the Relational Calculus can be translated to non-recursive Datalog$^{\neg}$

Rel. Calculus ➔ non-rec. Datalog$^\neg$

Proof

First remove ∀ by using De Morgan: $\forall x.\varphi = \neg\exists x.\neg\varphi$

Then, inductively, for each subformula φ
create a datalog rule F(x):-body
that computes $Q = \{ \mathbf{x} \mid \varphi \}$

Rel. Calculus ➔ non-rec. Datalog$^{\neg}$

$\boxed{R(t_1, \ldots, t_k)}$ ⇨ $\boxed{F(\mathbf{x}) \text{ :- } R(t_1, \ldots, t_k)}$

Rel. Calculus ➔ non-rec. Datalog$^{\neg}$

$F_1(\mathbf{x_1})$:- … /* for φ_1 */

$F_2(\mathbf{x_2})$:- … /* for φ_2 */

$\boxed{\varphi_1 \wedge \varphi_2}$ ⇨ $\boxed{\text{???}}$

Rel. Calculus ➔ non-rec. Datalog$^{\neg}$

$F_1(\mathbf{x_1})$:- … /* for φ_1 */

$F_2(\mathbf{x_2})$:- … /* for φ_2 */

$\boxed{\varphi_1 \wedge \varphi_2}$ $\Rightarrow$ $\boxed{F(\mathbf{x}) \text{ :- } F_1(\mathbf{x_1}), F_2(\mathbf{x_2})}$

Rel. Calculus ➔ non-rec. Datalog$^\neg$

$F_1(\mathbf{x_1})$:- … /* for φ_1 */

$F_2(\mathbf{x_2})$:- … /* for φ_2 */

$\boxed{\varphi_1 \vee \varphi_2}$ $\Rightarrow$ $\boxed{\textbf{? ? ?}}$

Rel. Calculus ➔ non-rec. Datalog$^\neg$

$\boxed{\varphi_1 \vee \varphi_2}$ ⇨

$F_1(\mathbf{x_1})$:- … /* for φ_1 */

$F_2(\mathbf{x_2})$:- … /* for φ_2 */

$F(\mathbf{x})$:- $F_1(\mathbf{x_1})$

$F(\mathbf{x})$:- $F_2(\mathbf{x_2})$

Rel. Calculus ➔ non-rec. Datalog$^\neg$

$F_1(\mathbf{x})$:- … /* for φ_1 */

$\boxed{\exists x.\varphi_1}$ ⇨ $\boxed{???}$

Rel. Calculus ➔ non-rec. Datalog$^{\neg}$

$F_1(\mathbf{x})$:- … /* for φ_1 */

$\boxed{\exists x.\varphi_1}$ ⇨ $\boxed{F(\mathbf{y}) \text{ :- } F_1(\mathbf{x})}$

where $\mathbf{y} = \mathbf{x} - \{x\}$

Rel. Calculus → non-rec. Datalog$^{\neg}$

$F_1(\mathbf{x})$:- … /* for φ_1 */

$\boxed{\neg\varphi_1}$ ⇒ $\boxed{F(\mathbf{x}) \text{ :- } \neg F_1(\mathbf{x})}$

End of Proof

Unsafe Queries

$\boxed{Q(x) \text{ :- } x=y}$

What’s wrong ?

$\boxed{Q(x) \text{ :- } \neg R(x,y)}$

Unsafe Queries

Definition. A datalog$^\neg$ rule S :- L_1, …, L_k is called *safe*, if every variable x appears in at least one positive relational atom.

Examples

Q(x) :- T(x), S(y), x=y

Q(x) :- T(x), S(y), ¬R(x,y)

We require that all rules of a non-recursive datalog$^\neg$ program to be safe.

Unsafe Queries

But what about Relational Calculus ?

What does it mean for a query Q to be safe ?

Are these queries safe ?

$Q = \{x \mid \forall y.\ \text{Frequents}(x,y)\}$

$Q = \{(x,y) \mid \text{Faculty}(x) \vee \text{Student}(y)\}$

Unsafe Queries

Definition. A query Q is *<u>domain independent</u>*,
if for any two database instances **D**, **D'**,
such that they have the same relations R1, …,Rk
but possibly different domains D, D', Q(**D**) = Q(**D'**).

A safe datalog$^{\neg}$ rule is domain independent (obvious).
Does the converse hold ?

Note: domain independent queries are also called *<u>safe</u>* queries, somewhat confusingly.

Unsafe Queries

Explain why these queries
are not domain independent:

$$\boxed{Q = \{x \mid \forall y.\ \text{Frequents}(x,y)\}}$$

$$\boxed{Q = \{(x,y) \mid \text{Faculty}(x) \lor \text{Student}(y)\}}$$

Unsafe Queries

Theorem. The problem: *"given a relational query Q decide whether Q is safe"* is undecidable.

Bummer !...

Why doesn't the following work ?
Translate Q to a datalog$^{\neg}$ program,
then check that each rule is safe.

Is this a decision procedure
for domain independence ?

Active Domain Semantics

Definition. The *active domain* ADom of a database instance **D** = (D, R_1^D, …, R_k^D), is the set of all constants in R_1^D, …, R_k^D

The active domain can be computed by a boring relational query:

```
ADom(x) :- Frequents(x,y)
ADom(y) :- Frequents(x,y)
ADom(x) :- Likes(x,y)
ADom(y) :- Likes(x,y)
ADom(x) :- Serves(x,y)
ADom(y) :- Serves(x,y)
```

Active Domain Semantics

Make the following two changes to standard semantics:

$\boxed{\mathbf{D} \vDash (\forall x.\varphi)\ [s]}$ If for every constant a in ADom, $\mathbf{D} \vDash \varphi[s_{a/x}]$

$\boxed{\mathbf{D} \vDash (\exists x.\varphi)\ [s]}$ If for some constant a in ADom, $\mathbf{D} \vDash \varphi[s_{a/x}]$

Active Domain Semantics

Theorem. If Q is domain-independent, then its standard semantics coincides with the active domain semantics

Why ?

Active Domain Semantics

Theorem. If Q is domain-independent, then its standard semantics coincides with the active domain semantics

Why ?

Let **D** = (D, R_1^D, …, R_k^D), be a database instance.
Denote **D'** = (ADom, R_1^D, …, R_k^D)

Since Q is d.i. we have Q(**D**) = Q(**D'**)

Q(**D'**) is the same as the active-domain semantics on **D**

Rel. Calculus → non-rec. Datalog$^{\neg}$

If Q is *domain independent*,
then we want to translate it into
a *safe* non-rec. datalog$^{\neg}$ program

$\neg\varphi_1$ ⇒ $F(\mathbf{x})$:- $\neg F_1(\mathbf{x})$

This rule is unsafe !
How do we make it safe ?

Rel. Calculus $\rightarrow$ non-rec. Datalog$^{\neg}$

If Q is *domain independent*,
then we want to translate it into
a *safe* non-rec. datalog$^{\neg}$ program

$\boxed{\neg\varphi_1}$ $\Rightarrow$

$$\boxed{F(\mathbf{x}) \text{ :- } Adom(x_1), Adom(x_2), \ldots, Adom(x_k), \neg F_1(\mathbf{x})}$$

Where $\mathbf{x} = (x_1, x_2, \ldots x_k)$

Discussion

- Non-recursive datalog$^\neg$ is a simple, friendly, very popular notation for queries
- Naturally compositional: sequence of rules
- Exposes a hierarchy of queries
  - Single rule datalog = conjunctive queries
  - Non-rec. datalog program = monotone-RC
  - Non-rec. datalog$^\neg$ program = RC
- No quantifiers ! Because all are $\exists$

Why We Care

SQL ***is*** non-recursive datalog$^{\neg}$!

Q(**x**) :- R(..), S(..), T(..) → SELECT **x** FROM R, S, T WHERE …

Q1(**x**) :- …
Q2(**y**) :- …
Q(**z**) :- Q1(**x**) , Q2(**y**) → Subqueries in the FROM clause

Q(**x**) :- R(..), S(..), ¬T(..) → Subquery in the WHERE clause

Why We Care

- When we use a database system we need to write some complex queries
  - *The Masochists*: Find drinkers that frequent only bars that server no beer that they like
- Write it first in Relational Calculus
  - Easy exercise
- Translate it to Non-recursive datalog$^\neg$
  - This is subtle, but we have seen it in detail
- Translate Non-recursive datalog$^\neg$ to SQL
  - Easy exercise

Relational Algebra

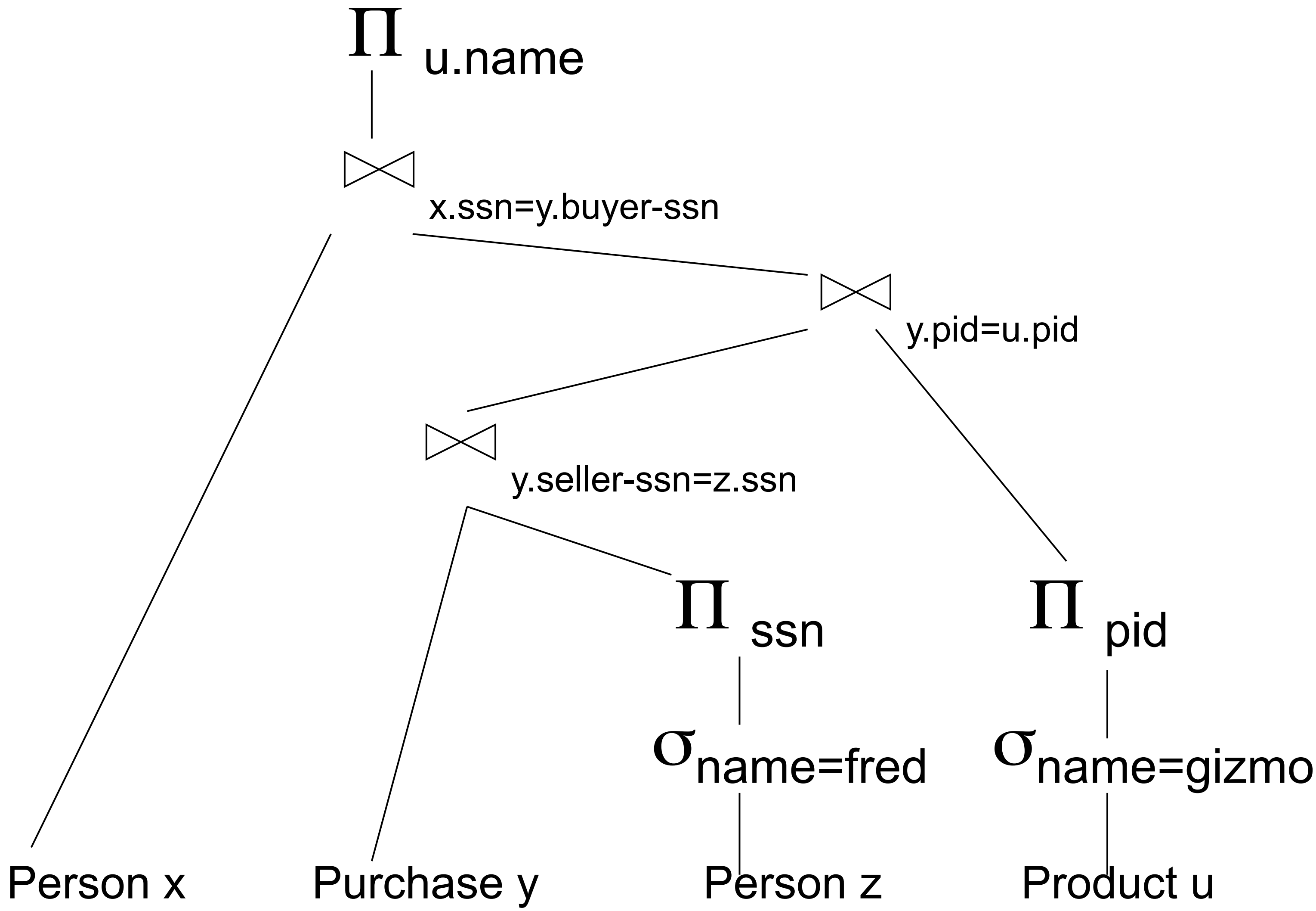
The Basic Five operators:

- Union: $\cup$
- Difference: -
- Selection: σ
- Projection: Π
- Join: $\bowtie$

Plus: cartesian product $\times$, renaming ρ

We will study RA in detail when we discuss Query Execution

Complex RA Expressions



Equivalence Theorem

Theorem. Every domain independent query in RC can be translated to safe non-recursive Datalog$^{\neg}$

Done that.

Theorem. Every safe non-recursive Datalog$^{\neg}$ can be translated to Relational Algebra.

Easy exercise – to do at home

Theorem. Relational Algebra expressions can be translated to a domain independent RC query

Easy exercise – to do at home

Discussion

- What is the monotone fragment of RA ?
  –

- What are the safe queries in RA ?
  –

- Where do we use RA (applications) ?
  –
  –

Discussion

- What is the monotone fragment of RA ?
  - No difference: $\cup$, σ, Π, $\bowtie$
- What are the safe queries in RA ?
  - All RA queries are safe
- Where do we use RA (applications) ?
  - Translating SQL (WHAT ➔ HOW)
  - Some new query languages (Pig-Latin)