Announcements

- Midterm in class on Friday 5/4
  - You can bring one letter-size sheet of notes (can write on both sides)
  - Practice exams available on website
- Game plan:
  - HW3/WQ3: due next Tues 4/17
  - HW4/WQ4: due on 4/24
  - HW5/WQ5: due on 5/1
  - HW6: released on 5/4

Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

**Example 1**

R encodes a graph e.g., connected cities

```
R =
1 2
2 1
2 3
1 4
3 4
4 5
```

**Example 2**

R encodes a graph e.g., connected cities

```
R =
1 2
2 1
2 3
1 4
3 4
4 5
```

**What does it compute?**

Multiple rules for the same IDB means OR

**typeof values**

- `number` (aka `varchar`)
- `symbol`
### Datalog Semantics

**Fixpoint semantics**

- **Step:**
  
  \[ \text{IDB}_0 = \text{empty relations} \]

- **Repeat:**
  
  \[ \text{IDB}_{t+1} = \text{Compute Rules(EDB, IDB)} \]

- **Until:**
  
  \[ \text{IDB}_\infty = \text{IDB}_{t+1} \]

- **Remark:** since rules are monotone:

  \[ \phi = \text{IDB}_0 \cup \text{IDB}_1 \cup \text{IDB}_\infty \cup \ldots \]

  It follows that a datalog program w/o functions (+, *, ...) always terminates. (Why?)
Three Equivalent Programs

R encodes a graph
e.g., connected cities

\[
\begin{array}{ccc}
1 & 2 & 3 \\
2 & 3 & 6 \\
2 & 3 & 4 \\
3 & 4 & 5 \\
4 & 5 & 1 \\
\end{array}
\]

Three Equivalent Programs

\[
\begin{align*}
& \text{T}(x,y) : - R(x,y). \\
& \text{T}(x,y) : - R(x,z), \ T(z,y). \\
& \text{T}(x,y) : - R(x,y). \\
& \text{T}(x,y) : - \ T(x,z), \ R(z,y). \\
& \text{T}(x,y) : - R(x,y). \\
& \text{T}(x,y) : - \ T(x,z), \ T(z,y). \\
\end{align*}
\]

Right linear

Left linear

Non-linear

Aggregates

\[
\begin{align*}
& \text{SELECT min(id) as minId} \\
& \text{FROM Actor as a} \\
& \text{WHERE a.name = 'John'} \\
& \text{Q(maxId) : maxId = max x : \{ a.id | a.name = 'John' \}} \\
\end{align*}
\]

Aggregates in Souffle:
- count
- min
- max
- sum

Assign variable to
the value of the aggregate

Aggregates

\[
\begin{align*}
& \text{SELECT min(id) as minId} \\
& \text{FROM Actor as a} \\
& \text{WHERE a.name = 'John'} \\
& \text{Q(y,c) : c = count \{ a.id | a.name = 'John' \}} \\
\end{align*}
\]

Can't use variable that are not
aggregated in the outer /head atoms

Counting

\[
\begin{align*}
& \text{Q(c) : c = count \{ a.id | a.name = 'John' \}} \\
\end{align*}
\]

Meaning (in SQL)

SELECT count(*) as c \\
FROM Actor as a \\
WHERE a.name = 'John'

Grouping

\[
\begin{align*}
& \text{Q(y,c) : Movie(_,_,y), c = count \{ Movie(_,_,y) \}} \\
\end{align*}
\]

Meaning (in SQL)

SELECT m.year, count(*) \\
FROM Movie as m \\
GROUP BY m.year

More Features

- Aggregates
- Grouping
- Negation
For each person, compute the total number of descendants

\[
\text{ParentChild}(p, c)
\]

For each person, compute the total number of descendants

\[
\text{ParentChild}(x, y) := \text{ParentChild}(x, y).
\]

\[
\text{ParentChild}(x, z) := \text{ParentChild}(x, y), \text{ParentChild}(y, z).
\]

// For each person, count the number of descendants
\[
\text{T}(p, c) := \text{D}(p, _), c := \text{count : } \{ \text{D}(p, y) \}.
\]

// Find the number of descendants of Alice
Example

For each person, compute the total number of descendants

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p_), c = count : { D(p,y) }.

// Find the number of descendants of Alice
Q(d) :- T(p,d), p = "Alice".

Negation: use "!"

Find all descendants of Alice, who are not descendants of Bob

// for each person, compute his/her descendants
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).

// Compute the answer: notice the negation
Q(x) :- D("Alice",x), !D("Bob",x).

Safe Datalog Rules

Here are unsafe datalog rules. What’s "unsafe" about them?

U1(x,y) :- ParentChild("Alice",x), y != "Bob"
U2(x) :- ParentChild("Alice",x), !ParentChild(x,y)

Safe Datalog Rules

Here are unsafe datalog rules. What’s "unsafe" about them?

U1(x,y) :- ParentChild("Alice",x), y != "Bob"
U2(x) :- ParentChild("Alice",x), !ParentChild(x,y)

A datalog rule is safe if every variable appears in some positive relational atom
Stratified Datalog

- Recursion does not cope well with aggregates or negation
- Example: what does this mean?
  \[
  A() :- !B().
  B() :- !A().
  \]
- A datalog program is stratified if it can be partitioned into strata
  - Only IDB predicates defined in strata 1, 2, ..., n may appear under ! or agg in stratum n+1.
- Many Datalog DBMSs (including souffle) accepts only stratified Datalog.

Many Datalog DBMSs (including souffle) accepts only stratified Datalog.

If we don’t use aggregates or negation, then the Datalog program is already stratified

If we do use aggregates or negation, it is usually quite natural to write the program in a stratified way