Datalog Terminology

Head - Body - Atom/Subgoal/Relational predicate

Base Relations (EDB) vs Derived Relations (IDB)

- Negation + Aggregate

Wildcard

\[
\text{Helper}(a,b) :- \text{Base1}(a,b,\_)
\]

\[
\text{NonAns}(j) :- \text{Base2}(j,k), !\text{Base3}(k)
\]

\[
\text{Ans}(x) :- \text{Helper}(x,y), !\text{NonAns}(y)
\]
Query Safety

Need a positive relational atom of every variable

What’s wrong with this query?

Find all of Alice’s children without children:
\[ U(x) \; :- \; \text{ParentChild(“Alice”,x), !ParentChild(x,y)} \]
Query Safety

U(x) :- ParentChild("Alice",x), !ParentChild(x,y)
It is domain dependent! Unsafe!

Double negation to the rescue. Why does this work?
NonAns(x) :- ParentChild("Alice",x), ParentChild(x,y)
# All of Alice’s children with children
U(x) :- ParentChild("Alice",x), !NonAns(x)
# All of Alice’s children without children (safe!)

But we can do better...
Query Safety

But we can do better...

\[
\text{hasChild}(x) \,:= \text{ParentChild}(x,\_)
\]

# People with children

\[
\text{U}(x) \,:= \text{ParentChild}(\text{“Alice”},x), \neg \text{hasChild}(x)
\]

# All of Alice’s children without children (safe!)
Datalog with Recursion

Able to write complicated queries in a few lines

Graph analysis

Done with query once output does not change.

VERY similar idea to context-free grammars (CSE 311)
Stratified Datalog

Recursion might not work well with negation

E.g.
A(x) :- Table(x), !B(x)
B(x) :- Table(x), !A(x)

Solution: Don’t negate or aggregate on an IDB predicate until it is defined
Stratified Datalog Query
Stratified Datalog

Only IDB predicates defined in strata 1, 2, ..., n may appear under ! or agg in stratum n+1

```
D(x,y) <- ParentChild(x,y).
D(x,z) <- D(x,y), ParentChild(y,z).
N[x] = m  <-  agg<<m = count()>> D(x,y).
Q(d) <- N["Alice"] = d.
```

```
D(x,y) <- ParentChild(x,y).
D(x,z) <- D(x,y), ParentChild(y,z).
Q(x)  <-  D("Alice",x), !D("Bob",x).
```

```
A()  <-  !B().
B()  <-  !A().
```

Non-stratified