CS422 Principles of Database Systems

Introduction to Datalog

Chengyu Sun
California State University, Los Angeles

Example

- Find tuples where
  - A > 2, or
  - B < 2

R

A	B
1	1
2	1
3	2
3	3
1	4

Relational Algebra vs. Datalog

- Relation Algebra

$SELECT_{A>2 \text{ OR } B < 2}$ (R)

- Datalog

Ans(a,b) ← R(a,b), a > 2
Ans(a,b) ← R(a,b), b < 2

Datalog Program and Rules

- A datalog program (query) consists of one or more rules

1st rule → Ans(a,b) ← R(a,b), a > 2
2nd rule → Ans(a,b) ← R(a,b), b < 2

Rules

- A rules consists of
  - Head (consequent)
  - ←
  - Body (antecedent)

head → [Ans(a,b)] ← [R(a,b), a > 2] ← body

Atoms

- A rule head consists of a single atom
- A rule body is the *AND* of one or more atoms
- An atom evaluates to either true of false

atom → [Ans(a,b)] ← [R(a,b),] [a > 2] ← atoms

More about Atoms

- An atom is also called a subgoal
- There're two types of atoms
  - Relational atoms
  - Arithmetic atoms
- And one more thing, *atoms* usually refer to *relational atoms*

Ans(a,b) ← R(a,b), a > 2

Relational atom Arithmetic atom

Predicates

- A relation atom consists of a predicate and the arguments of the predicate

Predicate *Ans* with *arguments* a and b

Ans(a,b) ← R(a,b), a > 2

Predicate *R* with *arguments* a and b

More about Predicates

- Two types of predicates
  - Extensional predicates (EDB) – stored relations
  - Intensional predicates (IDB) – computed relations
- No EDB in rule heads

EDB

IDB Ans(a,b) ← R(a,b), a > 2

Arguments and Variables

- Arguments can be either variables or constants
- If a variable appears in the head, it's called a distinguished variable; otherwise it's a non-distinguished variable

Ans(a,b) ← R(a,b), a > 2

a,b are both distinguished variables

Examples of Unsafe Rules

- S1(x) ← R(y,z)
- S2(x) ← R(y,z), x < 10
- S3(x) ← NOT R(x,y)

R

A	B
1	1
2	1
3	2
3	3
1	4

Safe Rules

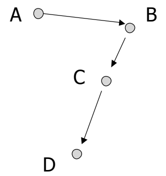
- A rules is safe if
  - each distinguished variable
  - each variable in a arithmetic subgoal
  - each variable in a negated subgoal

  also appears in a *non-negated, relational subgoal*

From Relation Algebra to Datalog

- Relation algebra
  - Intersection
  - Union
  - Difference
  - Project
  - Selection
  - Product/Join
- Datalog
  - ??

Need for Recursion …



Edges

P1	P2
A	B
B	C
C	D

… Need for Recursion

- Compute a relation *Paths(x,y)* – (x,y) is a tuple in *Paths* if there exists a path from x to y
  - Self-join is not enough
  - In fact, it's cannot be done in relational algebra

Recursive Datalog Solution

- Paths(x,y) ← Edges(x,y)
- Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Round 1

Edges

P1	P2
A	B
B	C
C	D

Paths

x	y

Paths(x,y) ← Edges(x,y)
Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Round 1

Edges

P1	P2
A	B
B	C
C	D

Paths

x	y
A	B
B	C
C	D

Paths(x,y) ← Edges(x,y)
Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Round 2

Edges

P1	P2
A	B
B	C
C	D

Paths

x	y
A	B
B	C
C	D

Paths(x,y) ← Edges(x,y)
Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Round 2

Edges

P1	P2
A	B
B	C
C	D

Paths

x	y
A	B
B	C
C	D
A	C
B	D

Paths(x,y) ← Edges(x,y)
Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Round 3

Edges

P1	P2
A	B
B	C
C	D

Paths

x	y
A	B
B	C
C	D
A	C
B	D

Paths(x,y) ← Edges(x,y)
Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Round 3

Edges

P1	P2
A	B
B	C
C	D

Paths

x	y
A	B
B	C
C	D
A	C
B	D
A	D

Paths(x,y) ← Edges(x,y)
Paths(x,y) ← Paths(x,z),Edges(z,y)

Evaluation of Recursive Rules – Done

- No more tuples can be added to Paths – we have reaches a fixed-point.

Definition of Recursion

- Form a *dependency graph* whose nodes = IDB predicates.
- Arc *X* -> *Y* if and only if there is a rule with *X* in the head and *Y* in the body.
- Cycle = recursion; no cycle = no recursion.

More Complex Recursive Examples

- *Cousins* in Ullman's notes

Recursion + Negation = Trouble

- Example 1

R: { 1 }

P(x) ← R(x), NOT Q(x)
Q(x) ← R(x), NOT P(x)

- Example 2

P(x) ← R(x), NOT P(x)

More Use of the Dependency Graph

- Form a *dependency graph* whose nodes = IDB predicates.
- Arc *X* -> *Y* if and only if there is a rule with *X* in the head and *Y* in the body.
- Cycle = recursion; no cycle = no recursion.

- Label Arc X → Y negative with a - sign

Stratified Recursion

- If a cycle in the dependency graph has no negative arc, it's a *stratified recursion.*
- We restrict ourselves to only recursions that are *stratified*.

Un-stratified recursion:

Exercises

- 10.1.1, 10.1.2
- 10.2.1, 10.2.2, 10.2.5
- 10.3.2