Lecture 27

Tuesday, November 30, 2021 1:49 PM

Datalog

- Becoming comfortable with the language
  Writing programs
- The problem of negation

Stratified negation (Most Datalogs) / Negation-as-failure (Prolog)

Stable model semantics (Answer Set Programming)

⇓

Aggregation

- Demystify the magic / Evaluation algorithms.

Example (Rule / Clause)

Cousin (x y) :- Parent (z, x), Parent(w, y),
Parent (u, z), Parent(u, w).

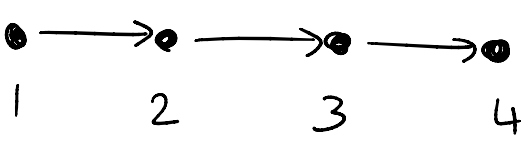
Head | Body

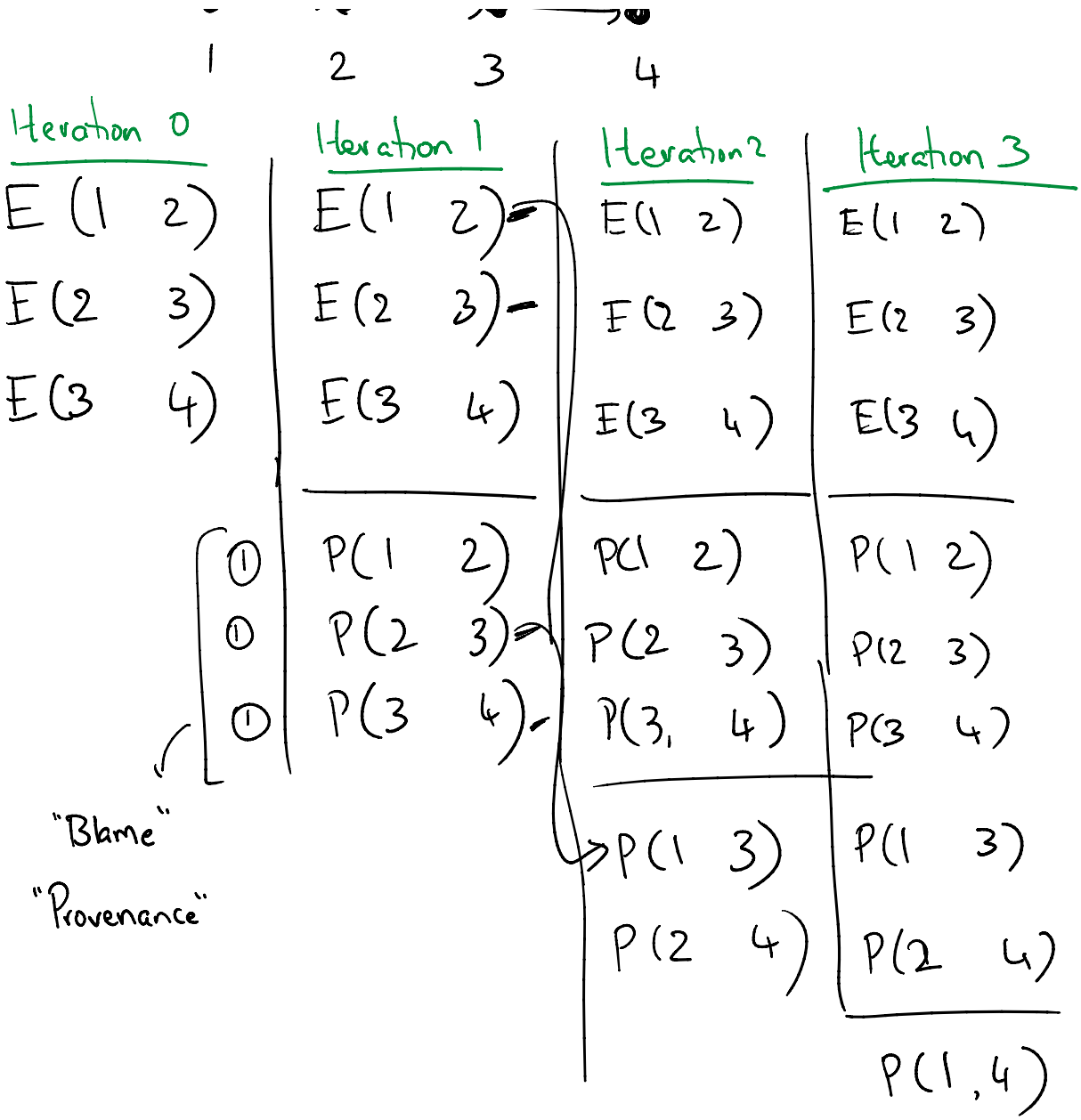
∀ x y z w u, If body
then head

Example

① Path (x, y) :- Edge(x, y)

② Path (x, z) :- Edge(x, y), Path(y, z).





- At this point, the rules don't derive anything new

"Fixpoint" / "Fixed point"

Least fixpoint

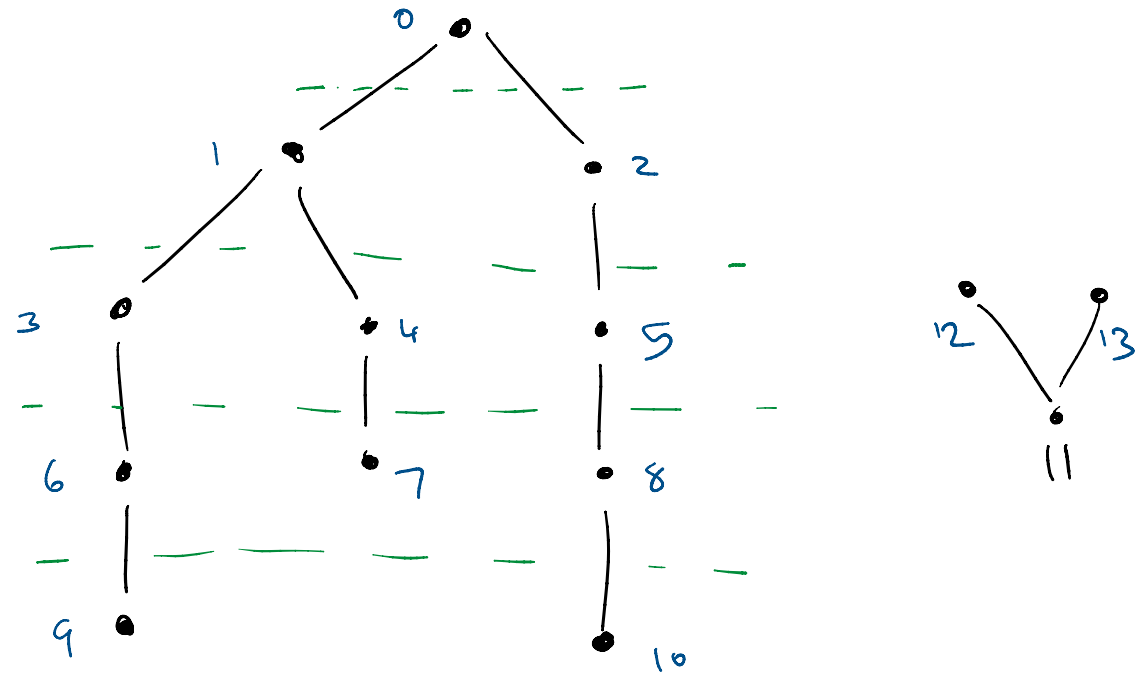
- Programs will terminate

(under some assumptions)

Example : Find all people in the same generation of a family.

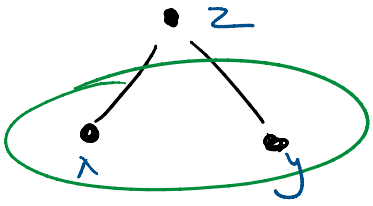
Parent (x y) — "x is a parent of y"

Samegen (x y) — "x & y are in the same generation."

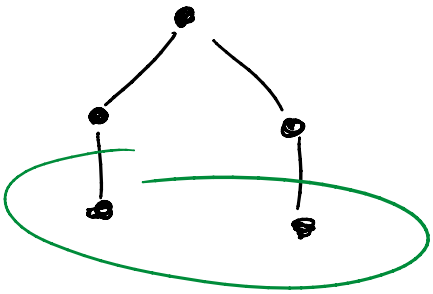


If same parent, then samegen.

sgen(x y) :- parent(z x) parent(z y) (1)

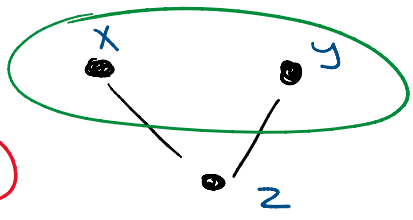


If same grandparent, then samegen

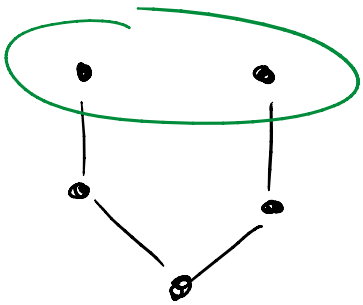


If share offspring, then samegen

sgen(x y) :- parent(x z) parent(y z) (2)



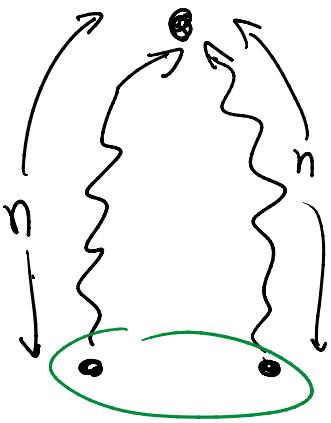
If share grandchild, then samegen



If common ancestor, &

same # of levels to common ancestor

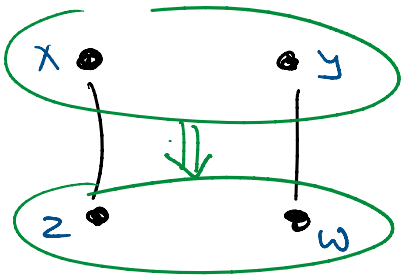
then samegen



If parents in samegen,

then offspring in samegen.

(3)

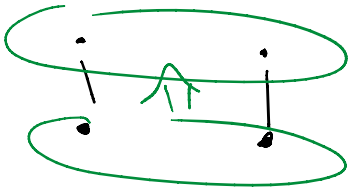


sgen(x y) :- parent(x z) parent(y w) sgen(z w)

If children in samegen,

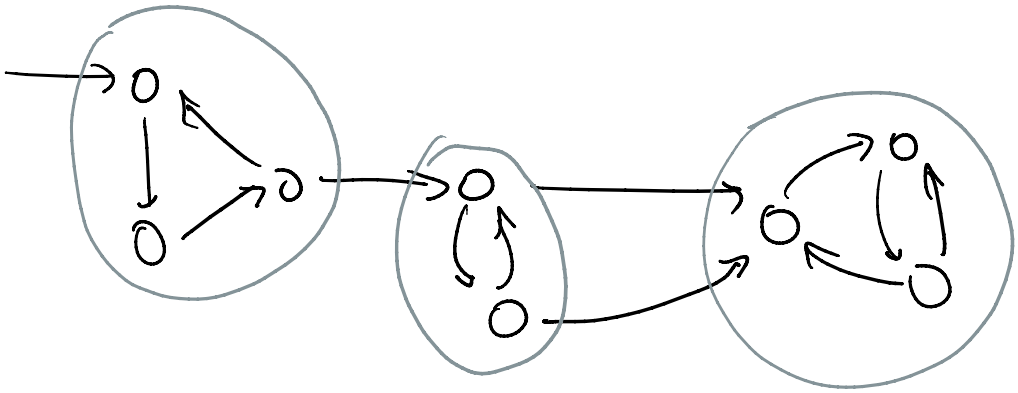
then parents in samegen.

(4)



sgen(x y) :- parent(z x) parent(w y) sgen(w z).

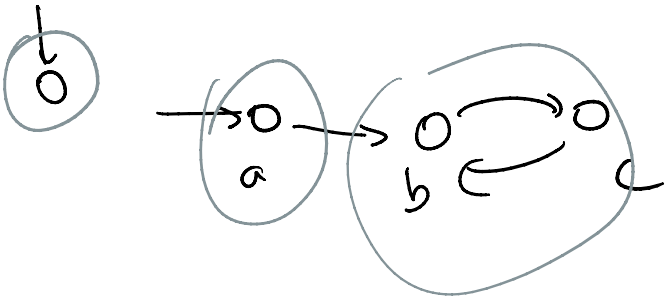
Example Strongly connected components



Defn: Vertices x & y are in the same SCC

if there is a path from x to y
& there is a path from y to x.

```
scc(x  y) :- path(x  y)  path(y  x)
path(x  y) :- edge(x  y)
path(x  z) :- path(x  y)  path(y  z)
```



Closed World Assumption

All that is true, I know to be true.

I know everything (that is true)

The Problem of Negation

Soln 1: If a rule has R(_) in the head then !R(_) cannot be in the body.

```
paradox(X) :- rel(x), ! paradox(x)
```

```
P1(X) :- rel(x), ! P2(x)
P2(x) :- P1(X).
```

← Soln 1 fails to eliminate this program

Soln 2 (Stratification)

```
S(e) ∘ edge
S(p) 1 path(x, y) :- edge(x  y)        ← s(p) ≥ s(e)
       path(x  z) :- edge(x  y) path(y  z).
S(n) 2 nopath(x y) :- ! path(x y)
```

s(n) : nopath(x, y) :- ! path(x, y)

s(p) ≥ s(e)

s(p) ≥ s(p)

s(n) > s(p)

+ strict!

---

nopath(x, y) :- ! path(x, y)

But what do x & y range over?!

nopath(x, y) :- vertex(x) vertex(y)

! path(x, y)
