Updated 2/23/2022

Topic 1: Data models and query languages
Unit 4: Datalog (continued)
Lecture 09

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

https://northeastern-datalab.github.io/cs7240/sp22/

2/15/2022

Pre-class conversations

- Last class Datalog recapitulation
- Calendar: two more classes on data models & query languages
- Reminder: It is time to start to hand in your first scribes

- today:
  - Recursion (Datalog)

Syntax of rules

- evaluates to true when relation R_i contains the tuple described by $args_i$
- e.g. Actor(344759,'Douglas','Fowley') is true

$R_i(args_i)$: relational predicate with arguments ("atom")

arithmetic predicate

```
Q2(f,l) :- Actor(u,f,l), Casts(u,x), Movie(x,y,z), z<'1940'.
```

head

(or consequent)
single IDB subgoal

body

(or antecedent)
conjunction of subgoals

{f,l}: head variables
{u,x,y,z}: existential variables

Alternative notation: Q(args) <- R1(args) AND R2(args) .... / or variables begin with a capital, predicates with lower-case (problem: can't have "Boston")
Source: Dan Suciu, CSE 554, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

```
Q(y) :- Movie(x,y,z), z<'1940'.
```

Meaning of a datalog rule is a logical statement:

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

```
Q(y) :- Movie(x,y,z), z<'1940'.
```

Meaning of a datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<'1940' then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

logically equivalent to

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<'1940'.

Meaning of a datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<'1940' then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

compare with RC

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<'1940'.

Meaning of a datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<'1940' then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

compare with RC

$$\{(y)\mid \exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940]\ \}$$

We want the smallest set Q with this property (why?)

Syntactic Constraints

$$Q(\mathbf{x}) :- R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m).$$

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}\left[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}\left[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)\right]\right]$$

Recall we want the smallest set Q with this property

Two restrictions:

1. Safety: every head variable should occur in the body at least once

R(x,z) :- S(x,y), R(y,x).

?

Syntactic Constraints

$$Q(\mathbf{x}) :- R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m).$$

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}\left[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}\left[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)\right]\right]$$

Two restrictions:

1. Safety: every head variable should occur in the body at least once

~~R(x,z) :- S(x,y), R(y,x).~~

forbidden rule: z not in body

2. The head predicate must be an IDB (Intensional) predicate
   (Body can include both EDBs and IDBs)

Arc(x,y) :- Arc(x,z),Arc(z,y).

?

Syntactic Constraints

$$Q(\mathbf{x}) :- R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m).$$

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}\left[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}\left[R_1(\mathbf{x}_1,\mathbf{y}_1) \wedge \cdots \wedge R_m(\mathbf{x}_m,\mathbf{y}_m)\right]\right]$$

Two restrictions:

1. Safety: every head variable should occur in the body at least once

~~R(x,z) :- S(x,y), R(y,x).~~

forbidden rule: z not in body

2. The head predicate must be an IDB (Intensional) predicate
   (Body can include both EDBs and IDBs)

~~Arc(x,y) :- Arc(x,z),Arc(z,y).~~

assuming Arc is EDB

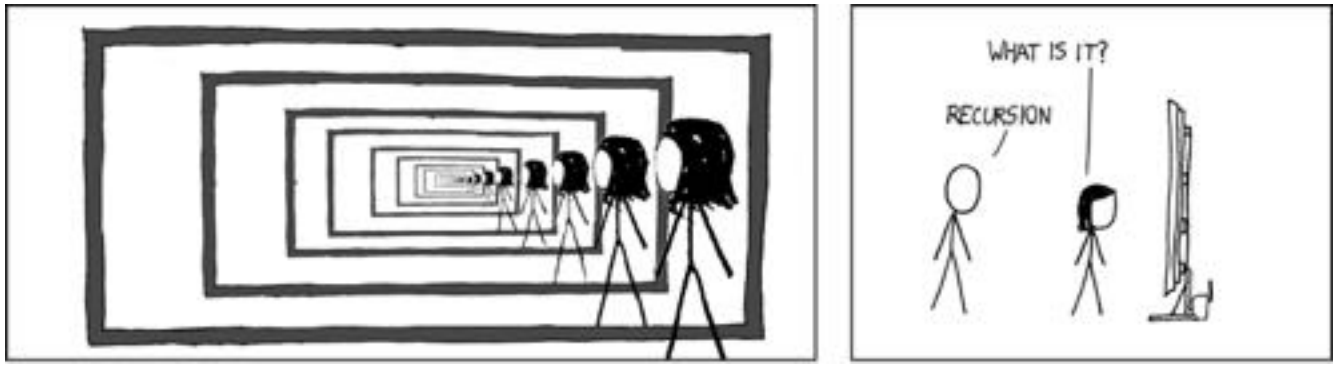
Outline: T1-4: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

Datalog program

- A datalog program consists of several rules
- Usually there is one distinguished predicate that's the output
- Importantly, rules may be recursive!

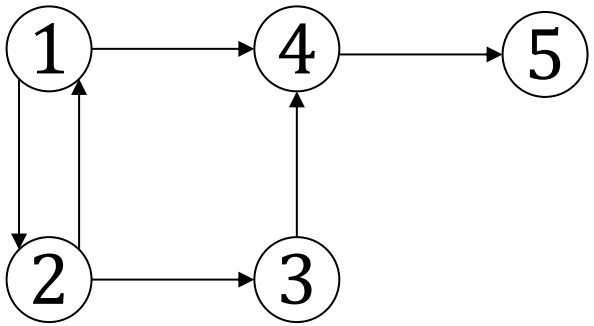
Recursion occurs when a thing is defined in terms of itself (self-repetition).



- (We later also see how to use recursion in SQL)

Figure Source: Fake XKCD: http://xkcdsw.com/1105

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

recursion due to head in rule body

A(S,T)

501

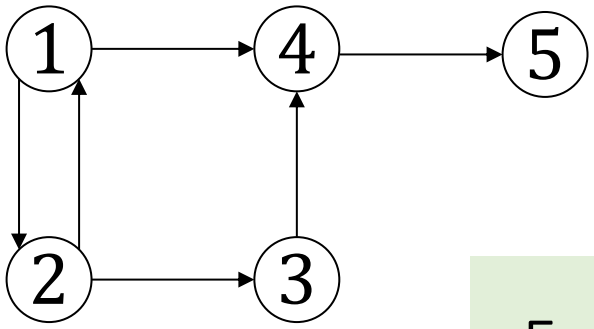
What does this query compute?

?

A

1	2
2	1
2	3
1	4
3	4
4	5

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

recursion due to head in rule body

Calculates all paths (transitive closure)

A(S,T)

501

A

1	2
2	1
2	3
1	4
3	4
4	5

For all nodes x and y:
If there is an arc from x to y,
then there is a path from x to y.

For all nodes x, z, and y:
If there is an arc from x to z, and there is a path from z to y
then there is a path from x to y.

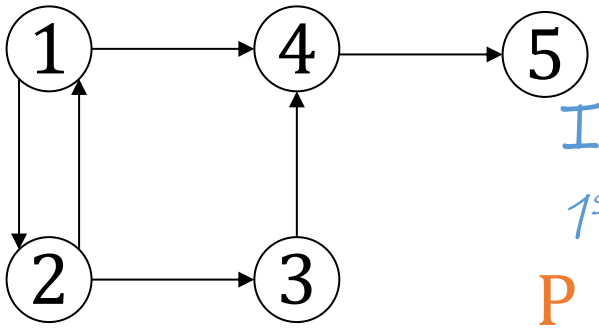
Example

A(S,T)

501

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body



Initially: P is empty

1st iteration

P

?

A

1	2
2	1
2	3
1	4
3	4
4	5

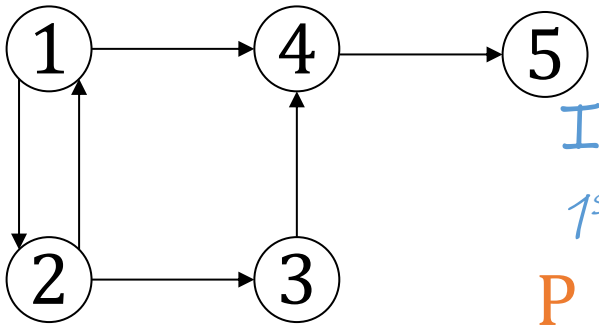
Example

A(S,T)

501

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body



Initially: P is empty

1st iteration

2nd iteration

P

1	2
2	1
2	3
1	4
3	4
4	5

P=A from 1st rule

P

?

A

1	2
2	1
2	3
1	4
3	4
4	5

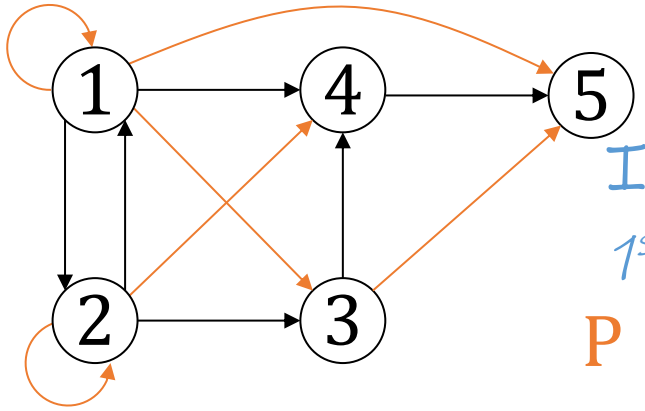
2nd rule generates nothing (because P is empty)

Example

EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

recursion due to head in rule body



Initially: P is empty

1st iteration

P

1	2
2	1
2	3
1	4
3	4
4	5

P=A from 1st rule

2nd iteration

P

1	2
2	1
2	3
1	4
3	4
4	5

1st rule

1	1
2	2
1	3
2	4
1	5
3	5

2nd rule

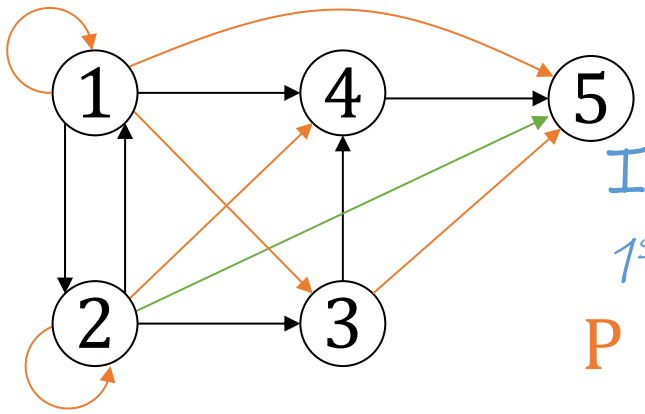
A

1	2
2	1
2	3
1	4
3	4
4	5

2nd rule generates nothing (because P is empty)

New facts from 2nd rule

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

recursion due to head in rule body

A(S,T)

501

A

1	2
2	1
2	3
1	4
3	4
4	5

Initially: P is empty

recall set semantics!

1st iteration

P

1	2
2	1
2	3
1	4
3	4
4	5

P=A from 1st rule

2nd rule generates nothing (because P is empty)

2nd iteration

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

1st rule

2nd rule

New facts from 2nd rule

3rd iteration = 4th iteration (No new facts)

P

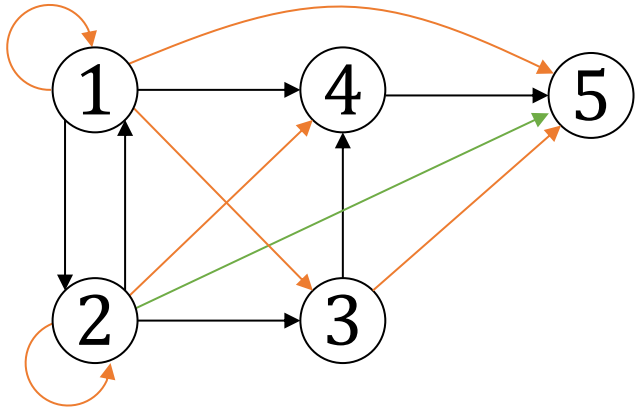
1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

1st + 2nd rule

1st rule

2nd rule

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

A(S,T)

501

In SQL ?

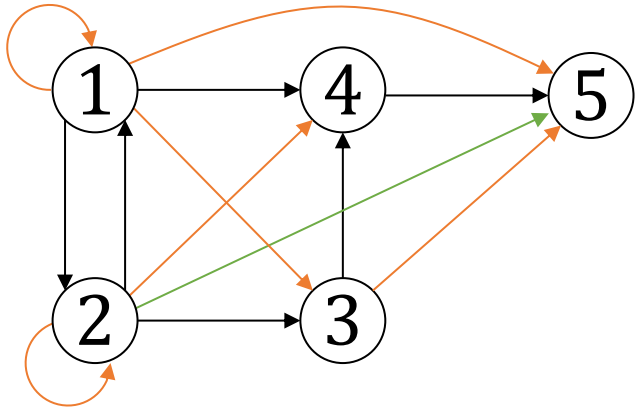
A

S	T
1	2
2	1
2	3
1	4
3	4
4	5

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

A(S,T)

501

A

S	T
1	2
2	1
2	3
1	4
3	4
4	5

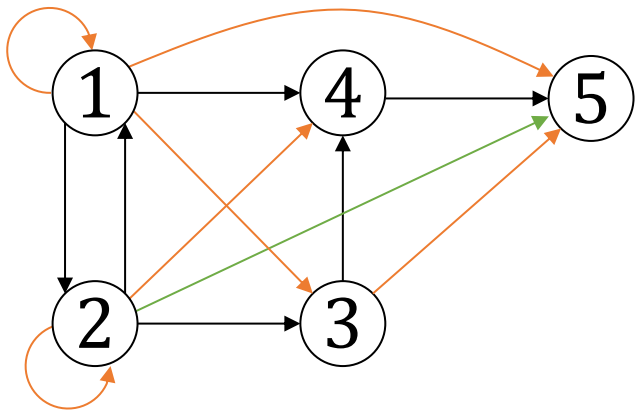
```
WITH RECURSIVE P AS (
      ?
   UNION
      ?

SELECT *
FROM P
```

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

A(S,T)

501

A

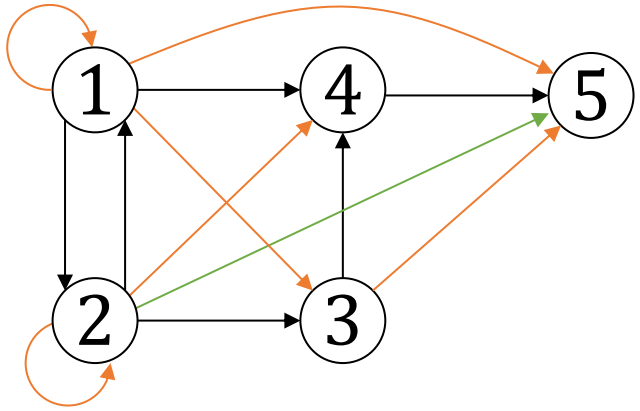
S	T
1	2
2	1
2	3
1	4
3	4
4	5

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A.S, P.T
    FROM A, P
    WHERE A.T = P.S)
SELECT *
FROM P
```

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), A(z,y).
```

A(S,T)

501

Strictly speaking, this process is iteration, not recursion:

A	S	T
	1	2
	2	1
	2	3
	1	4
	3	4
	4	5

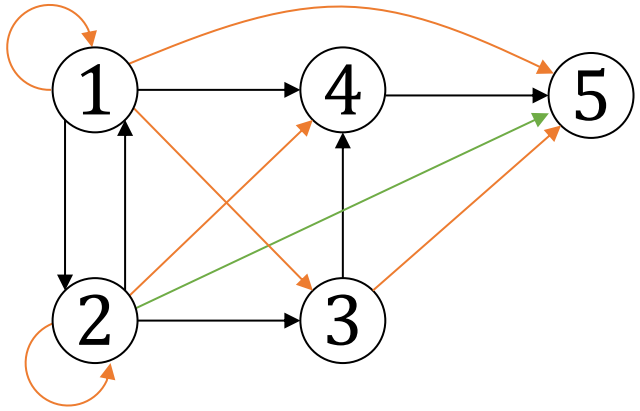
```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A.S, P.T
    FROM A, P
    WHERE A.T = P.S)
SELECT *
FROM P
```

Recursion and Iteration both repeatedly execute a set of instructions.

- Recursion (self-similarity) is when a statement in a function calls itself repeatedly.
- Iteration (reptition) is when a loop repeatedly executes until the controlling condition becomes false.

See also: https://www.postgresql.org/docs/14/queries-with.html#QUERIES-WITH-RECURSIVE
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example



EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), A(z,y).
```

A(S,T)

501

A

S	T
1	2
2	1
2	3
1	4
3	4
4	5

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A1.S, A2.T
    FROM A A1, A A2
    WHERE A1.T = A2.S)
SELECT *
FROM P
```

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

?

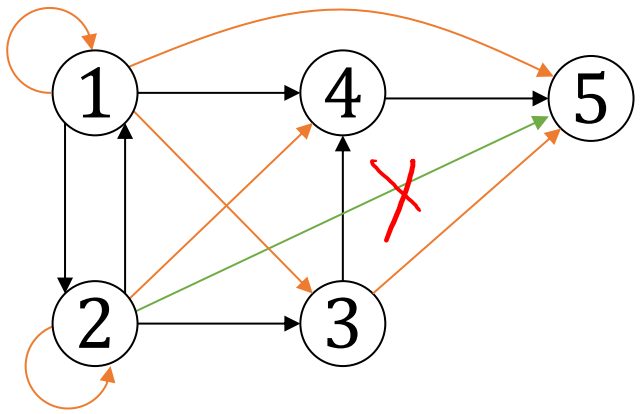
Example

EDB
IDB

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), A(z,y).
```

A(S,T)

501



A

S	T
1	2
2	1
2	3
1	4
3	4
4	5

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A1.S, A2.T
    FROM A A1, A A2
    WHERE A1.T = A2.S)
SELECT *
FROM P
```

P

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Challenge

???

- Write a query that finds the shortest path to each node from a starting node
- Create an interesting minimum database instance
- Show interesting variations
- https://www.postgresql.org/docs/14/queries-with.html

?

Which of these programs is recursive?

1

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
Invited(y) :- Relative('myself',y),Local(y).
```

?

2

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Invited(y) :- Relative('myself',y),Local(y).
```

?

3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
Visit(x,y) :- MayLike(x,y).
Close(x,z) :- Visit(x,y),Visit(z,y).
```

?

Dependency Graph

- The dependency graph of a Datalog program is the directed graph (V,E) where
  - V is the set of IDB predicates (relation names)
  - E contains an arc S ⟶ T whenever there is a rule with T in the head and S in the body

- A Datalog program is recursive if its dependency graph contains a cycle

Which of these programs is recursive?

1

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
Invited(y) :- Relative('myself',y),Local(y).
```

?

2

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Invited(y) :- Relative('myself',y),Local(y).
```

?

3

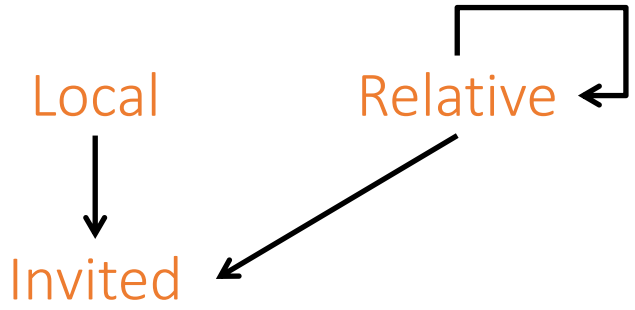
```
MayLike(x,y) :- Close(x,z),Likes(z,y).
Visit(x,y) :- MayLike(x,y).
Close(x,z) :- Visit(x,y),Visit(z,y).
```

?

Which of these programs is recursive?

1

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
Invited(y) :- Relative('myself',y),Local(y).
```



2

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Invited(y) :- Relative('myself',y),Local(y).
```

?

3

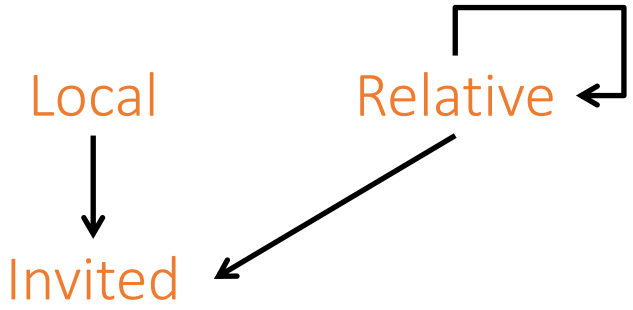
```
MayLike(x,y) :- Close(x,z),Likes(z,y).
Visit(x,y) :- MayLike(x,y).
Close(x,z) :- Visit(x,y),Visit(z,y).
```

?

Which of these programs is recursive?

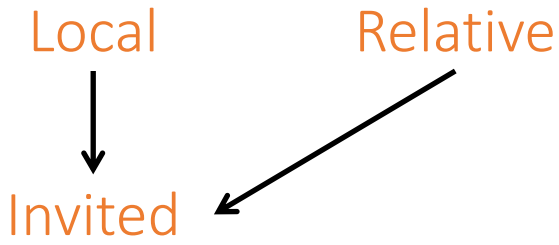
1

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
Invited(y) :- Relative('myself',y),Local(y).
```



2

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Invited(y) :- Relative('myself',y),Local(y).
```



3

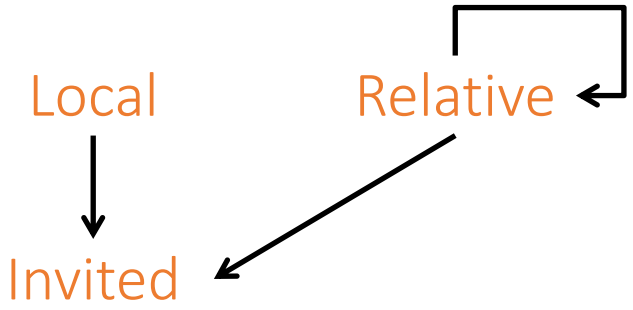
```
MayLike(x,y) :- Close(x,z),Likes(z,y).
Visit(x,y) :- MayLike(x,y).
Close(x,z) :- Visit(x,y),Visit(z,y).
```

?

Which of these programs is recursive?

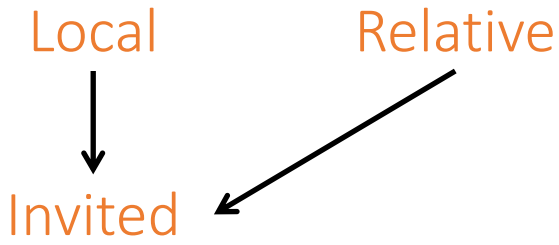
1

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
Invited(y) :- Relative('myself',y),Local(y).
```



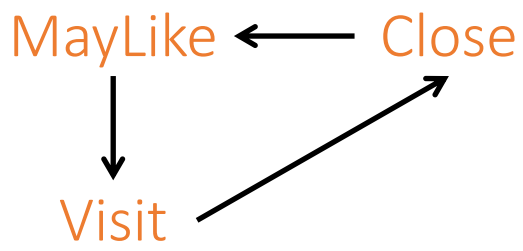
2

```
Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Invited(y) :- Relative('myself',y),Local(y).
```



3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
Visit(x,y) :- MayLike(x,y).
Close(x,z) :- Visit(x,y),Visit(z,y).
```



Outline: T1-4: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

Semantics of Datalog Programs

- Let **S** be a schema, **D** a database over **S**, and **P** be a Datalog program over **S** (i.e., all EDBs predicates belong to **S**)
- The result of evaluating **P** over **D** is a database **I** over the IDB schema of **P**
- We give 2 definitions:
  1. Fixpoint semantics — operative (think procedural)
  2. model-theoretic — declarative

1. Fixpoint semantics via the chase (operative definition)

Pseudo-code of a chase procedure:

Chase(P,D)

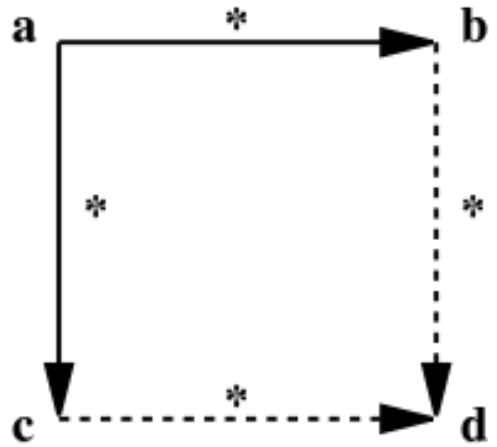
```
I := empty
repeat {
    if(D∪I satisfies all the rules of P), then return I
    Find a rule head(x) :- body(x,y) and constants a,b
       s.t. that D∪I contains body(a,b) but not head(a)
    I := I ∪ {head(a)}
}
```

Notice since rules are monotone, I is also monotonically increasing

Nondeterminism

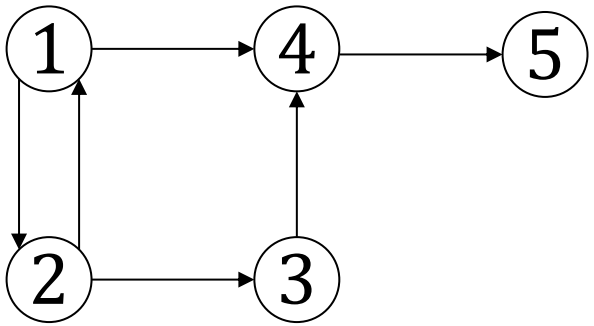
- Note: the chase is underspecified (i.e., not fully defined)
  - There can be many ways of choosing the next violation to handle
  - And each choice can lead to new violations, and so on
- We can view the choice of a new violation as nondeterministic

Church-Rosser property (defined for term reduction):
*If term **a** can be reduced to both **b** and **c**, then there must be a further term **d** (possibly equal to either b or c) to which both **b** and **c** can be reduced.*



Also see: https://en.wikipedia.org/wiki/Church%E2%80%93Rosser_theorem

Example




```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

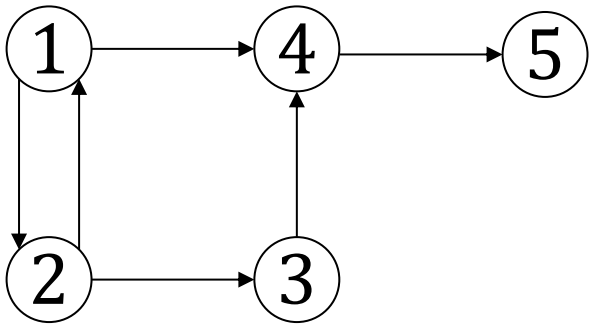
Arc

1	2
2	1
2	3
1	4
3	4
4	5

Path

Reachable

Example




```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

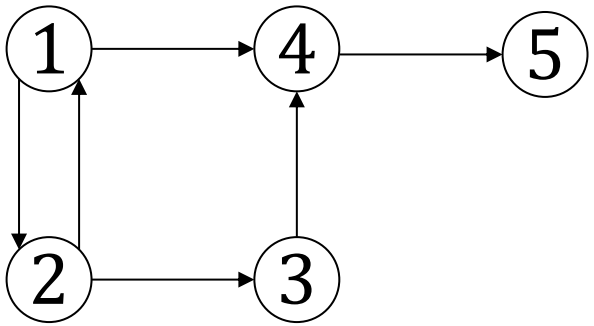
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2

Reachable

Example




```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

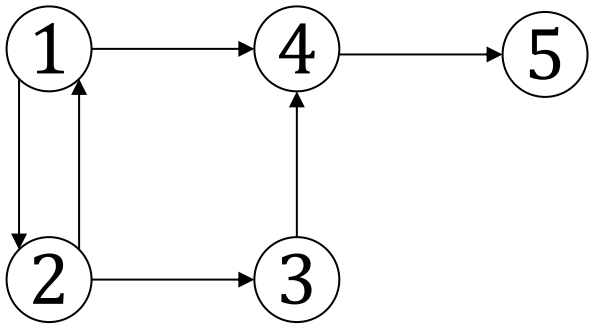
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2
2	1

Reachable

Example




```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

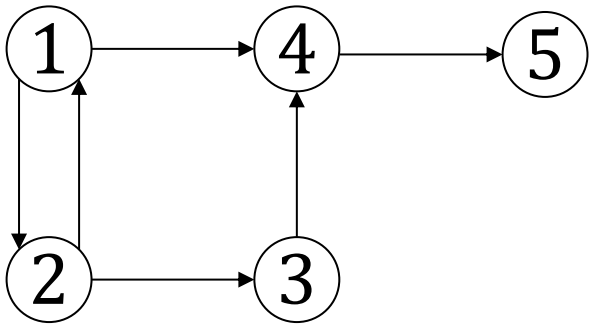
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2
2	1
2	3

Reachable

Example




```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

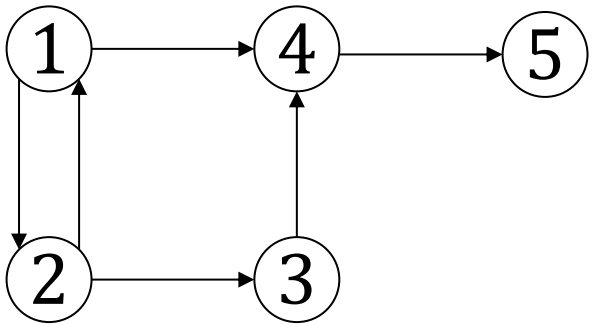
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2
2	1
2	3
1	1

Reachable

Example




```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2
2	1
2	3

Reachable

2

2. Minimal model semantics (model-theoretic definition)

- We say that IDB I is a model of Datalog program P (w.r.t. EDB D) if D∪I satisfies all the rules of P

$$\forall var[\text{Head}(IDB) \Leftarrow \text{Body}(EDB, IDB)]$$

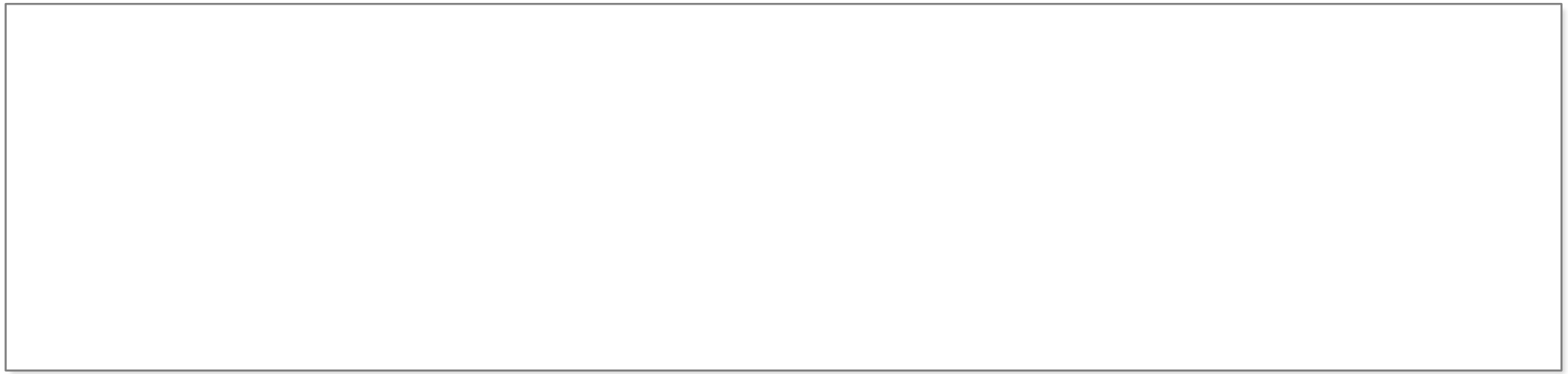
- We say that I is a minimal model if I does not properly contain any other model

- Theorem: there exists one minimal model

Illustration with our example

```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
```

1. Fixpoint semantics



2. Minimal model semantics: smallest Path s.t.

Illustration with our example

```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
```

1. Fixpoint semantics

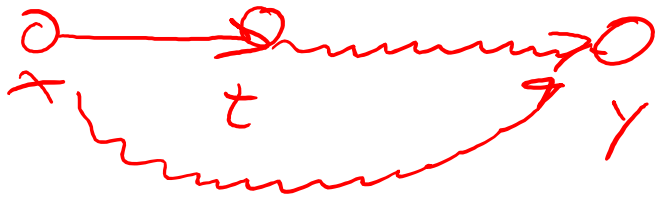
$$\text{Path}^{(0)} := \emptyset,\ t:=0$$

Repeat {

  inc(t)

  $\text{Path}^{(t)}(x,y) := \text{Arc}(x,y) \cup \Pi_{xy}(\text{Arc}(x,z) \bowtie \text{Path}^{(t-1)}(z,y))$

until $\text{Path}^{(t)} = \text{Path}^{(t-1)}$}



"immediate consequence operator"

2. Minimal model semantics: smallest relation Path s.t.

Illustration with our example

```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
```

1. Fixpoint semantics

- $Path^{(0)} := \emptyset$, t:=0
- Repeat {
  - inc(t)
  - $Path^{(t)}(x, y) := Arc(x,y) \cup \Pi_{xy}(Arc(x,z) \bowtie Path^{(t-1)}(z,y))$
- until $Path^{(t)} = Path^{(t-1)}$}

"immediate consequence operator"

2. Minimal model semantics: smallest relation Path s.t.

$$\forall x,y\ [Arc(x,y) \Rightarrow Path(x,y)] \wedge$$
$$\forall x,y,z\ [Arc(x,z) \wedge Path(z,y) \Rightarrow Path(x,y)]$$

Datalog Semantics & equivalence b/w the definitions

(nondeterministic)

1. The fixpoint semantics tells us how to compute a Datalog query
2. The minimal model semantics is more declarative: only says what we get

> THEOREM: For all Datalog programs P and DBs D
> there is a unique minimal model,
> and every chase returns this model

Proof sketch:

1. If $\mathbf{I}_1$ and $\mathbf{I}_2$ are models, so are $\mathbf{I}_1 \cap \mathbf{I}_2$
2. Every chase returns a model (finite)
3. Pick a chase and prove by induction: If $\mathbf{I}'$ is a model, then every intermediate $\mathbf{I}$ is contained in $\mathbf{I}'$ (monotonicity)

The minimal model is the *result*, denoted P(D)

Semantics Summary

1. Fixpoint-theoretic
   - Most "operational": Based on the immediate consequence operator for a Datalog program.

2. Model-theoretic
   - Most "declarative": Based on model-theoretic semantics of first order logic. View rules as logical constraints.

Semantics Summary

1. Fixpoint-theoretic
   - Most "operational": Based on the immediate consequence operator for a Datalog program.
   - Least fixpoint is reached after finitely many iterations of the immediate consequence operator.
   - Basis for practical, bottom-up evaluation strategy.
2. Model-theoretic
   - Most "declarative": Based on model-theoretic semantics of first order logic. View rules as logical constraints.
   - Given input DB D and Datalog program P, find the smallest possible DB instance D' that extends D and satisfies all constraints in P.

Expressiveness of Non-recursive Datalog

THEOREM: Non-recursive Datalog with built-in predicates (<,>,≤,≥,!=) has the same expressive power as the positive algebra {σ,π,×,∪}

If we restrict selection to $\sigma_{=}$ (i.e. selection with a single equality), this fragment is also called at times UCQs (Union of Conjunctive Queries) or USPJ (Union-Select-Project-Join) queries.

Monotonicity

- Can Datalog express difference?
  - Answer: No!
- Proof: Datalog is monotone, difference is not
  - That is, if D and D' are such that every relation of D is contained in the corresponding relation of D' ($D \subseteq D'$), then $P(D) \subseteq P(D')$

$$D \subseteq D' \quad \Rightarrow \quad P(D) \subseteq P(D')$$

Outline: T1-4: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3).

?

Alternative notations are "! Parent(y,x)" or "NOT Parent(y,x)" instead of "¬ Parent(y,x)"

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

Box(x) :- Item(x), ¬Box(x).

Item('ball') → ?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

Box(x) :- Item(x), ¬Box(x).

Item('ball') → Box('ball') ???

LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).

Item('ball') → ?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

```
Friend(x,y) :- Likes(x,y),¬Parent(y,x).
```

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

```
Box(x) :- Item(x), ¬Box(x).
```

Item('ball') → Box('ball') ???

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).
```

Item('ball') → LeftBox('ball') ???
unsafe!

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- Item(x), ¬LeftBox(x).
```

Item('ball') ?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

```
Friend(x,y) :- Likes(x,y),¬Parent(y,x).
```

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

```
Box(x) :- Item(x), ¬Box(x).
```

Item('ball') → Box('ball') ???

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).
```

Item('ball') → LeftBox('ball') ???
unsafe!

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- Item(x), ¬LeftBox(x).
```

Item('ball') → LeftBox('ball')
Item('ball') → RightBox('ball')

⇒ Adding negation to Datalog is not straightforward!

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

```
Friend(x,y) :- Likes(x,y),¬Parent(y,x).
```

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

```
Box(x) :- Item(x), ¬Box(x).
```

Item('ball') → ~~Box('ball')~~
no model

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).
```

Item('ball') → ~~LeftBox('ball')~~
unsafe!

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- Item(x), ¬LeftBox(x).
```

Item('ball') → LeftBox('ball')
Item('ball') → RightBox('ball')

Later discussed "stable model" semantics

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "NOT Parent(y,x)"

Negation in Datalog

- Various semantics have been proposed for supporting negation in Datalog that still allow tractability
- We will first look at two:
  - 1. Semipositive Datalog¬ (restricted): PTIME
  - 2. Stratified Datalog¬ (standard): PTIME
- We will later look at a more powerful (but intractable) semantics
  - Stable Models semantics (or answer set programming, ASP): NP-complete

1. Semipositive Programs and Safety

Friend(x,y) :- Likes(x,y), ¬Parent(y,x).

$$\text{Likes} - \pi_{y,x}\text{Parent}$$

A semipositive program is a program where only EDBs may be negated

- Semantics: same as ordinary Datalog programs
- Safety: rule is safe if every variable occurs in a positive (= unnegated) relational atom (ensures domain independence: the results of programs are finite and depend only on the actual contents of the database)

Exercise: Are following rules safe?

S(x) :- T(y), Arc(z,y), ¬Arc(x,y). ?

S(x) :- T(y), ¬T(x). ?

1. Semipositive Programs and Safety

```
Friend(x,y) :- Likes(x,y), ¬Parent(y,x).
```

$$\text{Likes} - \pi_{y,x}\text{Parent}$$

A semipositive program is a program where only EDBs may be negated

- Semantics: same as ordinary Datalog programs
- Safety: rule is safe if every variable occurs in a positive (= unnegated) relational atom (ensures domain independence: the results of programs are finite and depend only on the actual contents of the database)

Exercise: Are following rules safe?

```
S(x) :- T(y), Arc(z,y), ¬Arc(x,y).
```

unsafe

```
S(x) :- T(y), ¬T(x).
```

P(1)

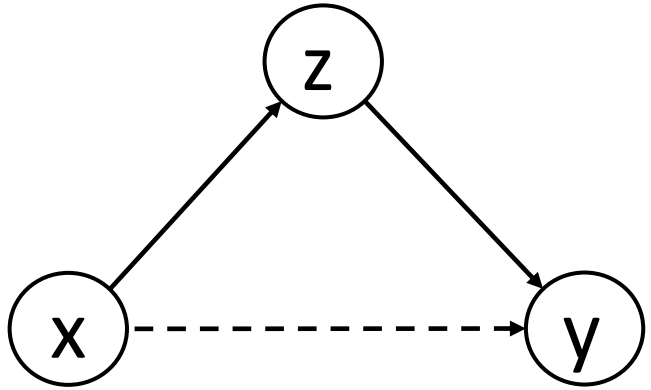
unsafe

1. Semipositive: Negated Atoms

Arc(Source,Target)

- We may put ¬ or ! or not in front of an atom, to negate its meaning.
- EXAMPLE: For any node x, return all nodes y that are two hops away, where y is not an immediate neighbor of x.

?

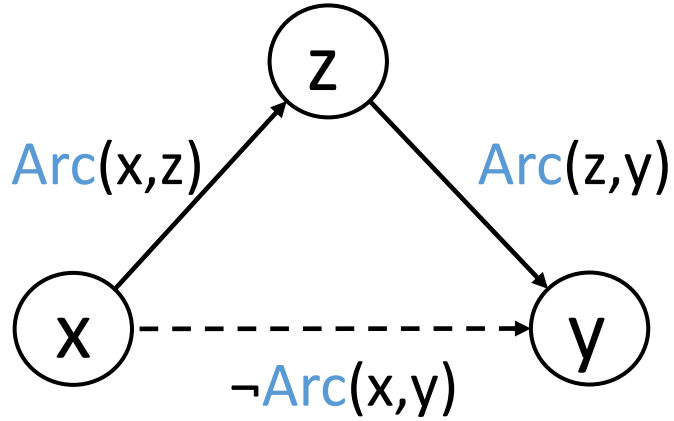


1. Semipositive: Negated Atoms

Arc(Source,Target)

- We may put ¬ or ! or not in front of an atom, to negate its meaning.
- EXAMPLE: For any node x, return all nodes y that are two hops away, where y is not an immediate neighbor of x.

```
TwoHopsAway(x,y) :- Arc(x,z), Arc(z,y), ¬Arc(x,y).
```



A(S,T)

501

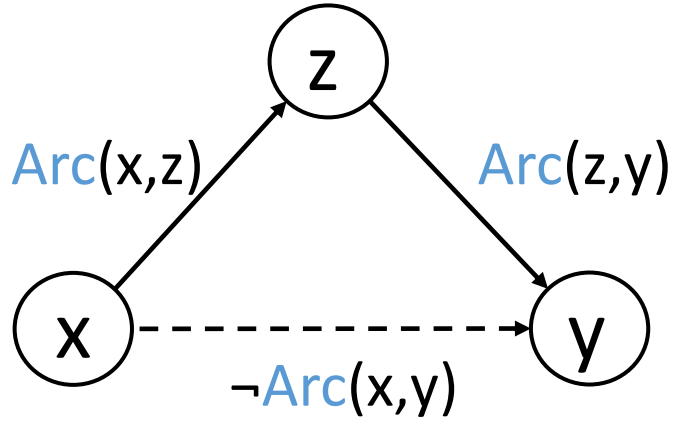
SQL ?

1. Semipositive: Negated Atoms

Arc(Source,Target)

- We may put ¬ or ! or not in front of an atom, to negate its meaning.
- EXAMPLE: For any node x, return all nodes y that are two hops away, where y is not an immediate neighbor of x.

TwoHopsAway(x,y) :- Arc(x,z), Arc(z,y), ¬Arc(x,y).



A(S,T)

501

```
SELECT A1.S, A2.T
FROM A A1, A A2
WHERE A1.T = A2.S
AND NOT EXISTS
    (SELECT *
    FROM A A3
    WHERE A3.S = A1.S
    AND A3.T = A2.T)
```

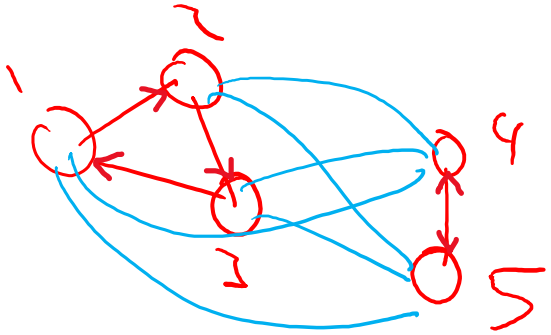
Exampe: beyond Semipositive

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

?

Node is basically ADom:
Node(x) :- Arc(x,y)
Node(y) :- Arc(x,y)



Exampe: beyond Semipositive

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

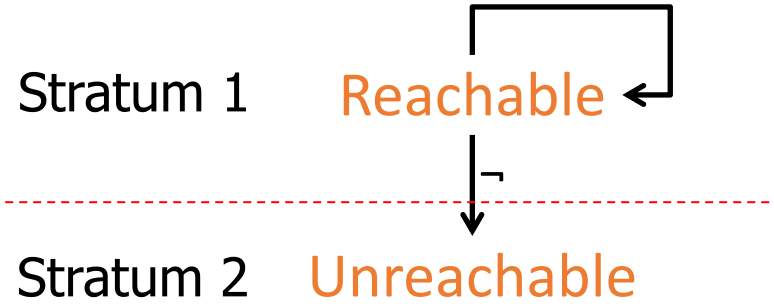
```
Reachable(x,y) :- Arc(x,y).
Reachable(x,y) :- Arc(x,z), Reachable(z,y).
```

Exampe: beyond Semipositive

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Arc(x,y).
Reachable(x,y) :- Arc(x,z), Reachable(z,y).
Unreachable(x,y) :- Node(x), Node(y), ¬ Reachable(x,y).
```



- Straightforward syntactic restriction.
- When the Datalog program is stratified, we can evaluate IDB predicates stratum-by-stratum
- Once evaluated, treat it as EDB for higher strata.

Precedence graph
- Nodes = IDB predicates
- Arc p→q if predicate q depends on p
- Label this arc "¬" if predicate p is negated

think: "topological sort"

Non-stratified example:

```
LeftBox(x) :- ¬LeftBox(x), Item(x).
```

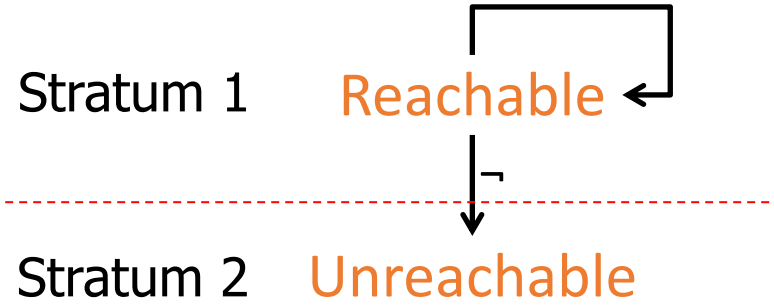
?

Exampe: beyond Semipositive

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Arc(x,y).
Reachable(x,y) :- Arc(x,z), Reachable(z,y).
Unreachable(x,y) :- Node(x), Node(y), ¬ Reachable(x,y).
```



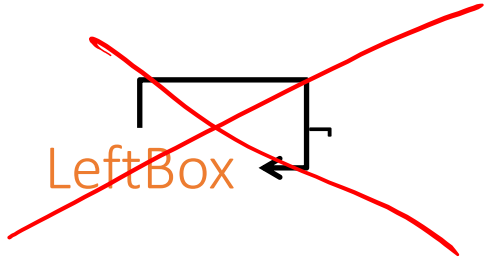
- Straightforward syntactic restriction.
- When the Datalog program is stratified, we can evaluate IDB predicates stratum-by-stratum
- Once evaluated, treat it as EDB for higher strata.

Precedence graph
- Nodes = IDB predicates
- Arc p→q if predicate q depends on p
- Label this arc "¬" if predicate p is negated

think: "topological sort"

Non-stratified example:

```
LeftBox(x) :- ¬LeftBox(x), Item(x).
```



2. Stratified Programs: Definition and Semantics

- DEFINITION: Let **P** be a Datalog program, **E** be the set of EDB predicates, and **I** be the set of IDB predicates. A stratification of **P** is a partitioning of the IDB predicates into disjoint sets $I_1,\ldots,I_k$ such that:
  - For i=1,...,k, every rule with head in I_i has posibie body predicates only from **E**, $I_1,\ldots, I_i$
  - For i=1,...,k, every rule with head in I_i has negated body predicates only from **E**, $I_1,\ldots, I_{i-1}$

- SEMANTICS:
  - For i=1,...,k:
    - Compute the IDBs of the stratum I_i, possibly via recursion
    - Add computed IDBs to the EDBs
  - Due to the definition of stratification, each E_i can be viewed as semipositive

2. Theorems on Stratification

- THEOREM 1: A program has a stratification if and only if its dependency graph does not contain a cycle with a "negated edge"
  - Dependency graph is defined as previously, except that edges can be labeled with negation
  - Hence, we can test for stratifiability efficiently, via graph reachability

```
A(x) :- B(x).
B(x) :- C(x).
C(x) :- ¬A(x).
```

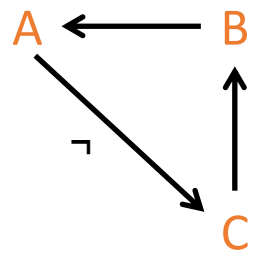
Stratified

?

2. Theorems on Stratification

- THEOREM 1: A program has a stratification if and only if its dependency graph does not contain a cycle with a "negated edge"
  - Dependency graph is defined as previously, except that edges can be labeled with negation
  - Hence, we can test for stratifiability efficiently, via graph reachability

```
A(x) :- B(x).
B(x) :- C(x).
C(x) :- ¬A(x).
```



- THEOREM 2: Non-recursive Datalog with negation can always be stratified ia the topological order

- THEOREM 3: Non-recursive Datalog with negation has the same expressive power as the algebra $\{\sigma_{=}, \pi, \times, \cup, -\}$
  - Extendable to RA if we add the comparison predicates <, >, !=, <=, >=

2. Stratification practice

Parent(P,C)

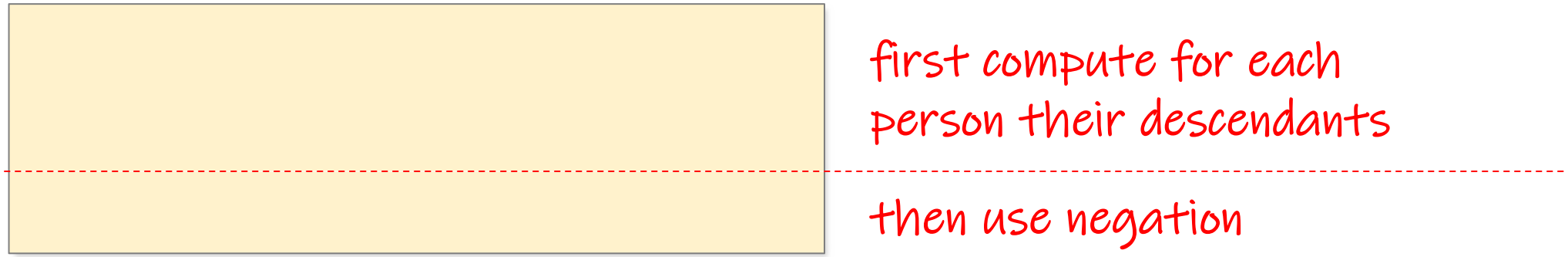
Q: Find all descendants of Alice,
who are not descendants of Bob

?

2. Stratification practice

Parent(P,C)

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



first compute for each person their descendants

then use negation

?

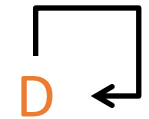
2. Stratification practice

Parent(P,C)

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

```
D(x,y) :- Parent(x,y).
D(x,z) :- Parent(y,z), D(x,y).
```

first compute for each person their descendants



then use negation

?

2. Stratification practice

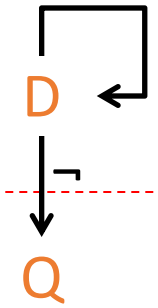
Parent(P,C)

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

```
D(x,y) :- Parent(x,y).
D(x,z) :- Parent(y,z), D(x,y).
Q(x) :- D('Alice',x), ¬D('Bob',x).
```

first compute for each person their descendants

then use negation



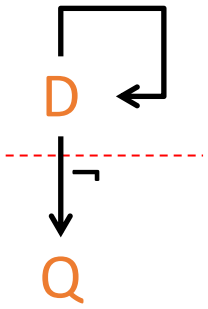
2. Stratification practice

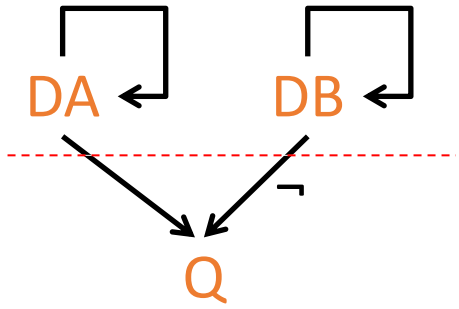
Parent(P,C)

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

```
D(x,y) :- Parent(x,y).
D(x,z) :- Parent(y,z), D(x,y).
Q(x) :- D('Alice',x), ¬D('Bob',x).
```

```
DA(y) :- Parent('Alice',y).
DA(y) :- Parent(x,y), DA(x).
DB(y) :- Parent('Bob',y).
DB(y) :- Parent(x,y), DB(x).
Q(x) :- DA(x), ¬DB(x).
```

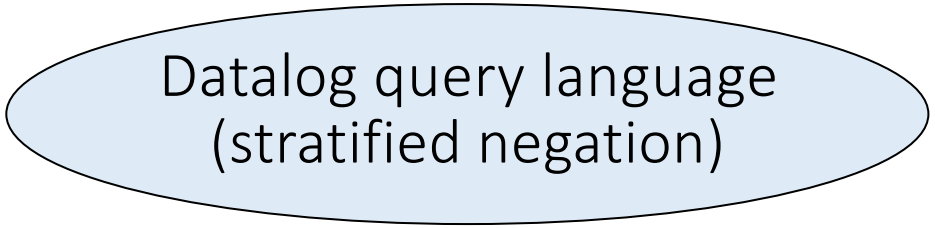




Outline: T1-4: Datalog

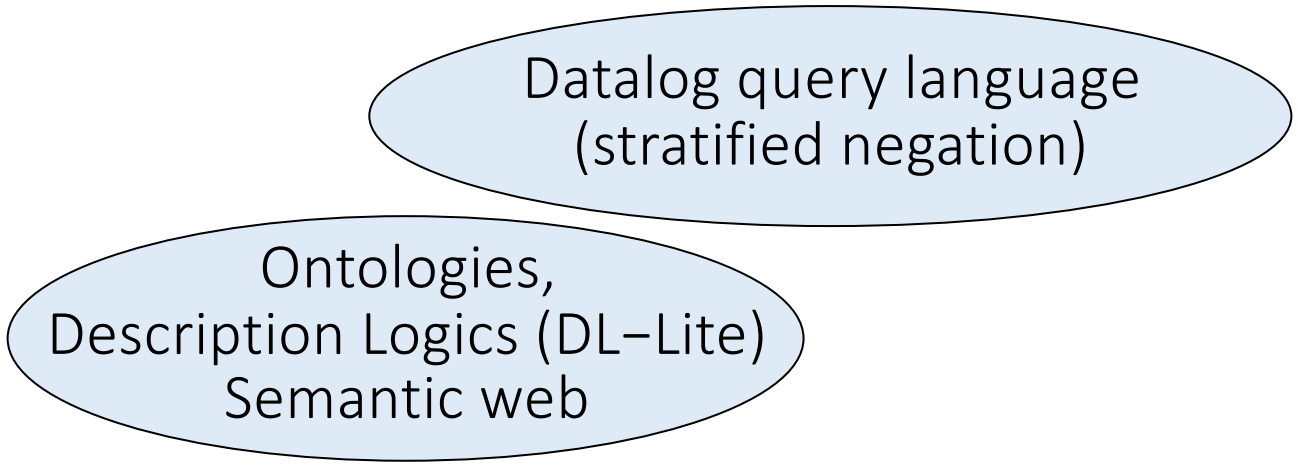
- Datalog
  - Datalog rules
  - Recursion
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

Datalog$^{\pm}$: background



- Much is possible with Datalog

Datalog$^{\pm}$: background

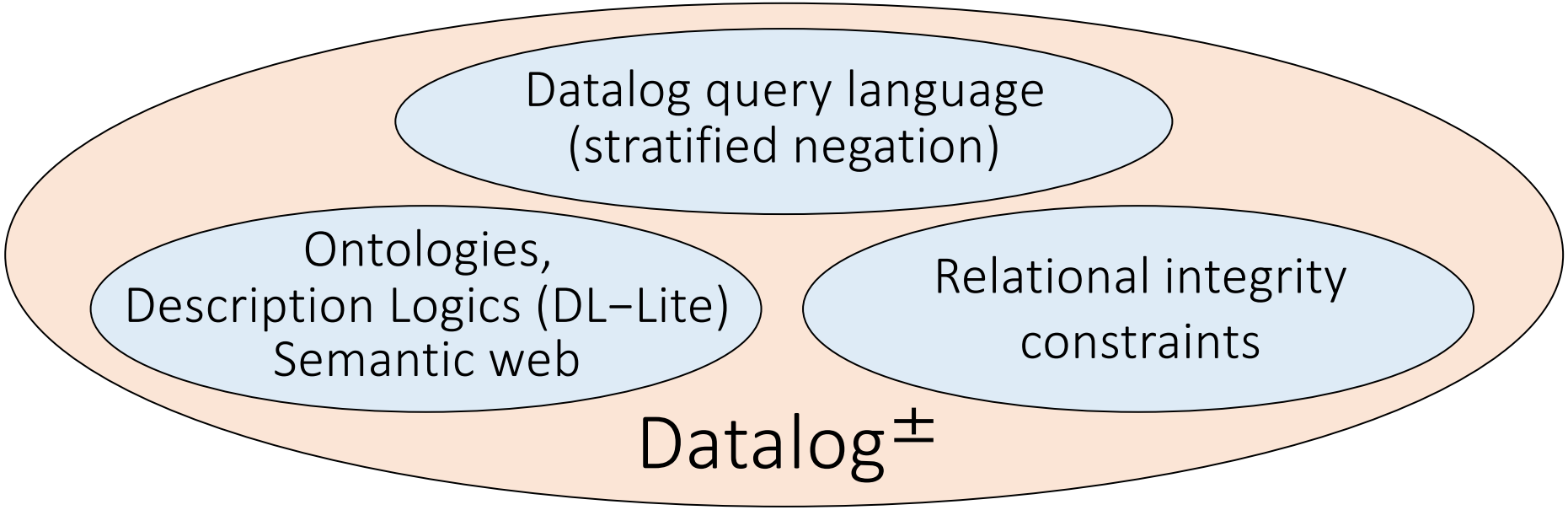


- Much is possible with Datalog
- Much is not (observed e.g. by [Patel-Schneider, Horrocks 2006])

Patel-Schneider, Horrocks. Position paper: A comparison of two modelling paradigms in the Semantic Web. WWW (Semantic Web track). 2006. https://dl.acm.org/doi/10.1145/1135777.1135784

Based on a presentation by Andrea Cali
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Datalog$^{\pm}$: goal



- Much is possible with Datalog
- Much is not (observed e.g. by Patel-Schneider, Horrocks 2006)
- Datalog$^{\pm}$ is a framework that extends Datalog with:
  - value invention (∃-variables in the head): TGDs (Tuple-Generating Dependencies)
  - equality predicate in the head: EGDs (Equality Generating Dependencies)
  - constant ⊥ in the head: negative constraints (disjointness)

Patel-Schneider, Horrocks. Position paper: A comparison of two modelling paradigms in the Semantic Web. WWW (Semantic Web track). 2006. https://dl.acm.org/doi/10.1145/1135777.1135784
Cali, Gottlob, Lukasiewicz, Marnette, Pieris. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. LICS 2010. https://doi.org/10.1109/LICS.2010.27
Based on a presentation by Andrea Cali
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Datalog and expressiveness for ontological reasoning

Assertion type	Datalog rule
Inclusion	$\text{emp}(X) \rightarrow \text{person}(X)$
(Inverse) role inclusion	$\text{reportsTo}(X, Y) \rightarrow \text{manages}(Y, X)$
Reflexive expansion	$\text{boss}(X) \rightarrow \text{manages}(X, X)$
Transitivity	$\text{manages}(X, Y), \text{manages}(Y, Z) \rightarrow \text{manages}(X, Z)$
Concept product	$\text{seniorEmp}(X), \text{emp}(Y) \rightarrow \text{higher}(X, Y)$
Participation	?
Disjointness	?
Functionality	?

Ontology assertion	Datalog$^{\pm}$ rule
Participation	$\text{boss}(X) \rightarrow \exists Y\ \text{reports}(Y, X)$
Disjointness	$\text{customer}(X), \text{boss}(X) \rightarrow \bot$
Functionality	$\text{reports}(X, Y1), \text{reports}(X, Y2) \rightarrow Y1 = Y2$

Based on a presentation by Andrea Cali
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Datalog$^{\pm}$ vs. DL

$$
\begin{aligned}
&FiveStar(X) \rightarrow Hotel(X),\\
&FiveStar(X), \text{not}\, Pool(X,Y) \rightarrow \exists Z\ Beach(X,Z),\\
&FiveStar(X), \text{not}\, Beach(X,Y) \rightarrow \exists Z\ Pool(X,Z),\\
&Beach(X,Y) \rightarrow \exists Z\ SwimOpp(X,Z),\\
&Pool(X,Y) \rightarrow \exists Z\ SwimOpp(X,Z),
\end{aligned}
$$

The above example corresponds to the following set of DL axioms, expressed in an extension of $\mathcal{ELHI}$ by nonmonotonic negation:

$$
\begin{aligned}
FiveStar &\sqsubseteq \text{Hotel},\\
FiveStar \sqcap \text{not}\exists Pool &\sqsubseteq \exists Beach,\\
FiveStar \sqcap \text{not}\exists Beach &\sqsubseteq \exists Pool,\\
\exists Beach &\sqsubseteq \exists SwimOpp,\\
\exists Pool &\sqsubseteq \exists SwimOpp,
\end{aligned}
$$

Source: Gottlob, Lukasiewicz, Pieris. Datalog+/-: Questions and Answers. AAAI 2014. https://www.aaai.org/ocs/index.php/KR/KR14/paper/viewPaper/7965
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Interesting Observations

- Exploiting schema knowledge in query answering is not trivial
- Languages and algorithms exist that allow for tractable query answering
- Applications in real-world scenarios are possible
  - Industrial applications in data integration, Semantic Web, ontological reasoning
  - Companies and Products: RelationalAI, Deepreason.ai, Oracle Semantic Technologies, IBM IODT, OntoDLV (Vienna)