Updated 1/23/2020

T1: Data models and query languages
L5: Datalog continued

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp20)

https://northeastern-datalab.github.io/cs7240/sp20/

Version 1/21/2020

Outline: 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

Syntactic Constraints

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

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

The rule stands for the following logical formula:

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

Restrictions:

- Safety: every variable in x should occur in the body at least once

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

Syntactic Constraints

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

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

The rule stands for the following logical formula:

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

Restrictions:

- Safety: every variable in x should occur in the body at least once

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

  forbidden rule: z not in body

- The head predicate R must be an IDB predicate
  (Body can include both EDBs and IDBs)

  $\mathrm{Edge}(x,y) :- \mathrm{Edge}(x,z),\mathrm{Edge}(z,y)$

Syntactic Constraints

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

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

The rule stands for the following logical formula:

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

Restrictions:

- Safety: every variable in x should occur in the body at least once

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

forbidden rule: z not in body

- The head predicate R must be an IDB predicate
  (Body can include both EDBs and IDBs)

~~$Edge(x,y) :- Edge(x,z),Edge(z,y)$~~

assuming Edge is EDB

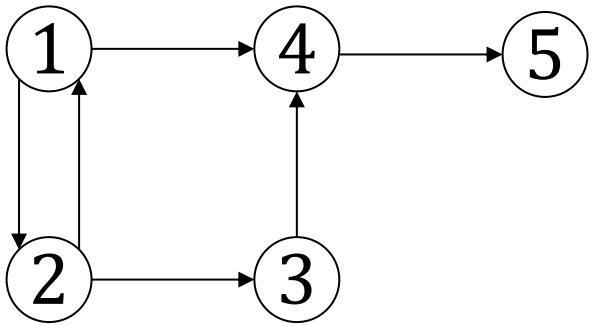
Outline: Datalog

- 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

Datalog program

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

Example



EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`

recursion due to head in rule body

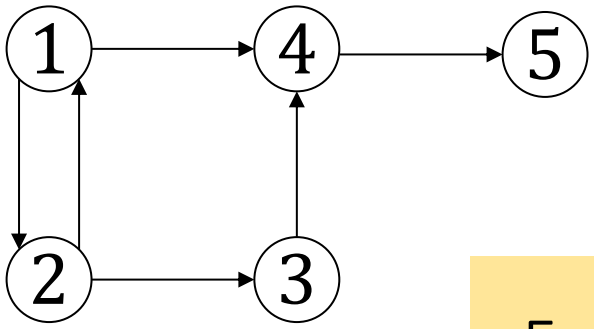
What does it compute?

?

E

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

Example



EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`

recursion due to head in rule body

What does it compute?

Calculates all paths (transitive closure)

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

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

E

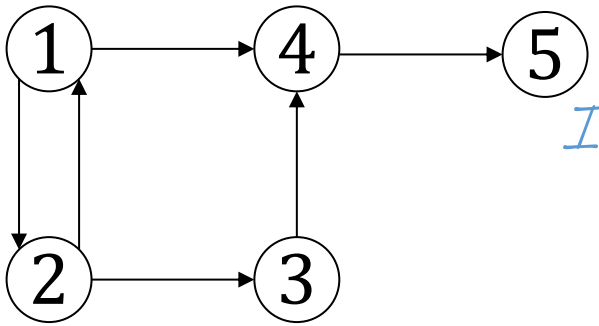
1	2
2	1
2	3
1	4
3	4
4	5

Example by Dan Suciu

Example

EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`



Initially: P is empty

recursion due to head in rule body

What does it compute?

E

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

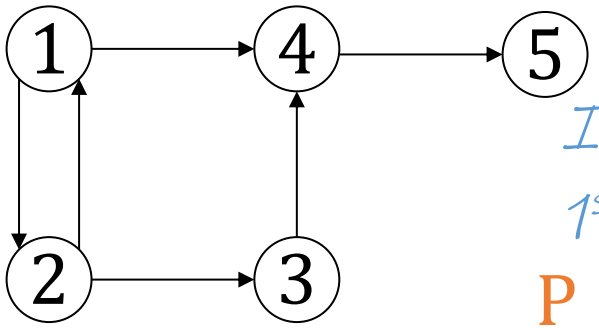
Example

EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`

recursion due to head in rule body

What does it compute?



Initially: P is empty

1st iteration

P

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

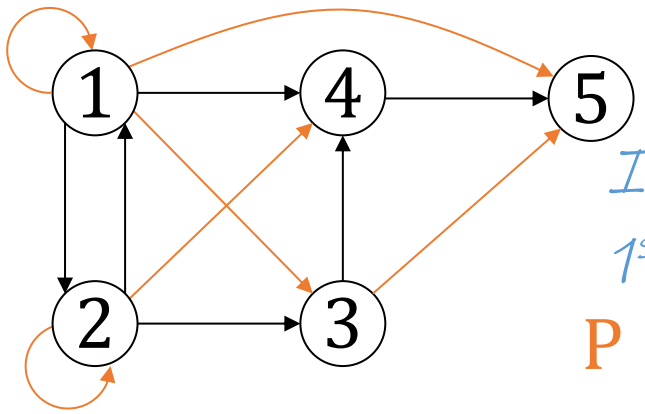
P=E from 1st rule

E

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

2nd rule generates nothing (because P is empty)

Example



EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`

recursion due to head in rule body

What does it compute?

recall set semantics!

E

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

Initially: P is empty

1st iteration

P

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

P=E 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

1st rule

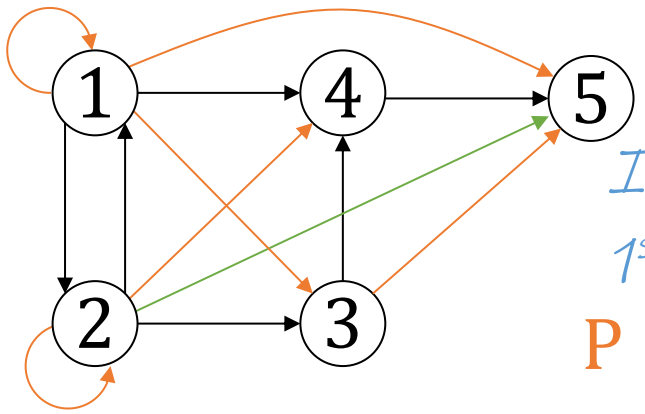
1	1
2	2
1	3
2	4
1	5
3	5

2nd rule

New facts from 2nd rule

Example by Dan Suciu

Example



EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`

recursion due to head in rule body

What does it compute?

recall set semantics!

E

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

Initially: P is empty

1st iteration

P

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

P=E 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

3rd 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
2	5

1st + 2nd rule

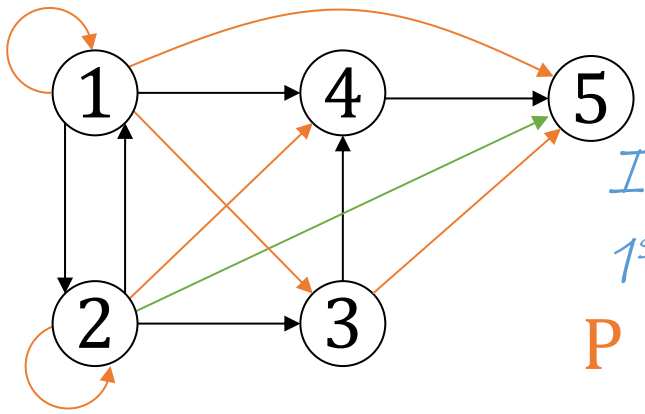
1st rule

2nd rule

New facts from 2nd rule

Example by Dan Suciu

Example



EDB `P(x,y) :- E(x,y)`

IDB `P(x,y) :- E(x,z), P(z,y)`

recursion due to head in rule body

What does it compute?

recall set semantics!

E

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

Initially: P is empty

1st iteration

P

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

P=E 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

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

New facts from 2nd rule

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)
```

?

Example by Benny Kimelfeld

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 edge $R \longrightarrow S$ whenever there is a rule with S in the head and R 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)
```

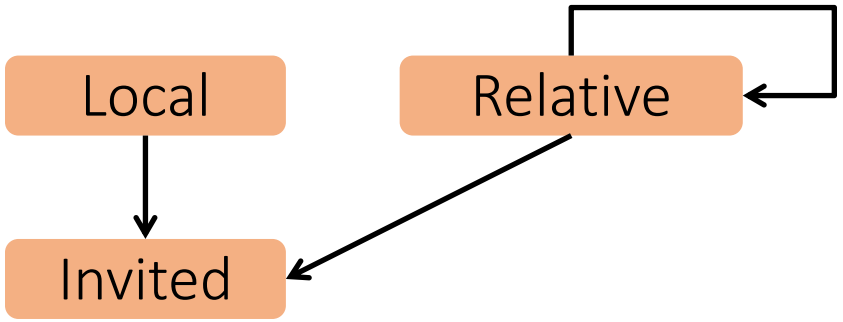
?

Example by Benny Kimelfeld

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)
```

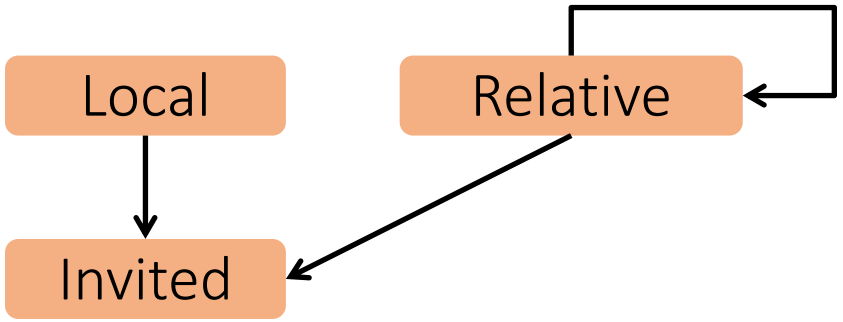
?

Example by Benny Kimelfeld

Which of these programs is recursive?

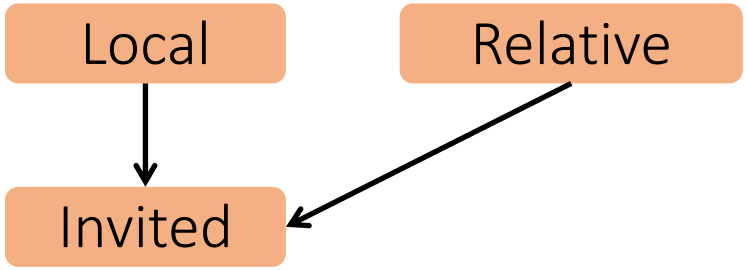
1

```
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)
```

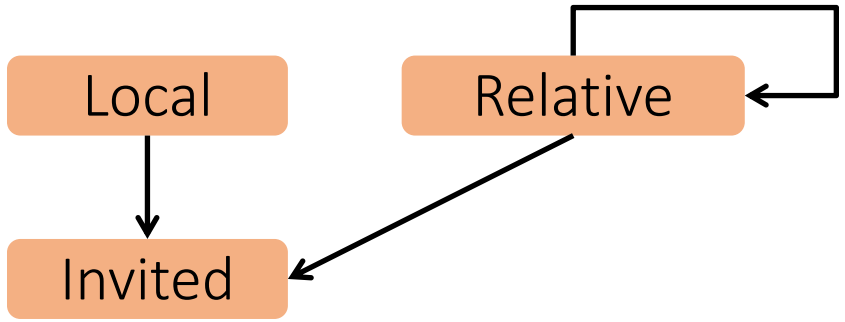
?

Example by Benny Kimelfeld

Which of these programs is recursive?

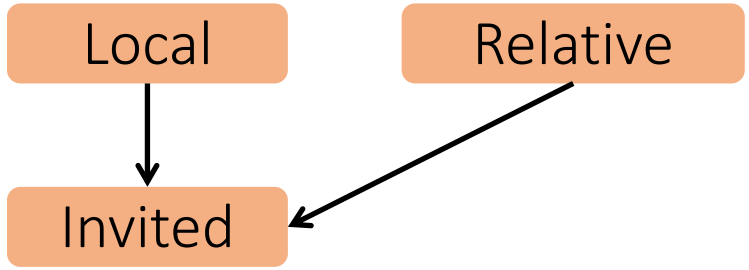
1

```
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)
```



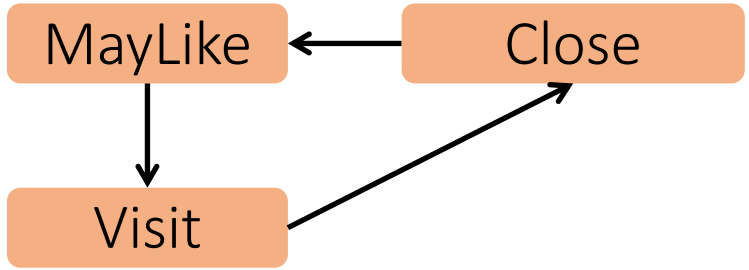
2

```
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)
```



Example by Benny Kimelfeld

Outline: 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

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
  2. model-theoretic

1. Fixpoint semantics via the chase (operative definition)

A chase procedure is a program of the following form:

Chase(P,D)

- I := empty
- while(true) {
  - if(D∪I satisfies all the rules of P), then return I
  - Find a rule head(**x**) :- body(**x**,**y**) and tuples **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

Example

Person(id, gender, country)
Parent(parent, child)

(To redo with Parent(child,parent) and Descendent..)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative *Invited*

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative *Invited*

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1

Invited

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1

Invited

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1

Invited

1

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1

Invited

1

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2

Invited

1

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2

Invited

1

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2
1	3

Invited

1

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2
1	3

Invited

1

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2
1	3

Invited

1
2

Example by Benny Kimelfeld

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2
1	3

Invited

1
2

Example

Person(id, gender, country)
Parent(parent, child)

```
Relative(x,x) :- Person(x,y,z)
Relative(x,y) :- Relative(x,z),Parent(z,y)
   Invited(y) :- Relative('1',y)
```

Person

1	F	IL
2	M	AU
3	F	NL
4	M	IL
5	F	FR

Parent

1	2
2	3

Relative

1	1
1	2
1	3

Invited

1
2
3

2. Minimal model semantics

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

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

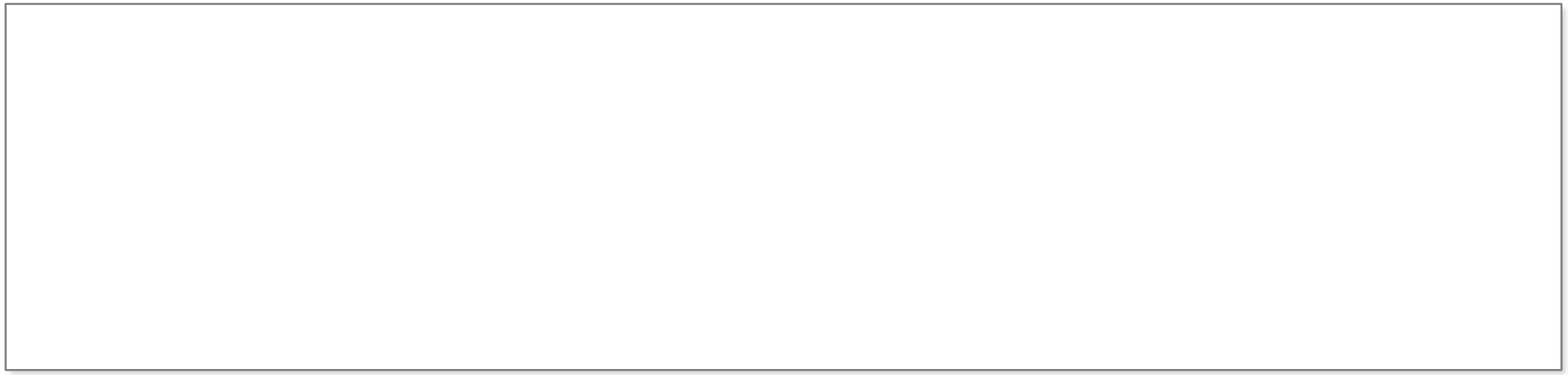
- We say that I is a minimal model if I does not properly contain any other model

- Theorem: there exists one minimal model

Illustration with our example

```
Path(x,y) :- Edge(x,y)
Path(x,z) :- Edge(x,y), Path(y,z)
```

1. Fixpoint semantics



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

Illustration with our example

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

1. Fixpoint semantics

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

"immediate consequence operator"

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

Illustration with our example

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

1. Fixpoint semantics

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

"immediate consequence operator"

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

$$\forall x,y\ [\text{Edge}(x,y) \Rightarrow \text{Path}(x,y)] \wedge$$
$$\forall x,y,z\ [(\text{Edge}(x,z) \wedge \text{Path}(z,y)) \Rightarrow \text{Path}(x,y)]$$

Datalog Semantics & equivalence b/w the definitions

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
3. Pick a chase and prove by induction: If $\mathbf{I}'$ is a model, then every intermediate $\mathbf{I}$ is contained in $\mathbf{I}'$

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 has the same expressive power as the algebra $\{\sigma_{=}, \pi, \times, \rho, \cup\}$ (where $\sigma_{=}$ means selection with a single equality)

This fragment is often called "positive RA" or USPJ (union-select-project-join)

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: 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

What is the Semantics?

Buddy(x,y) :- Likes(x,y),¬Parent(y,x)

?

What is the Semantics?

Buddy(x,y) :- Likes(x,y),¬Parent(y,x)

LeftBox(x) :- ¬LeftBox(x), Item(x)

?

What is the Semantics?

Buddy(x,y) :- Likes(x,y),¬Parent(y,x)

LeftBox(x) :- ¬LeftBox(x), Item(x)

Item('ball') ⟶ ? LeftBox('ball')

Left(x) :- ¬Right(x), Item(x)

Right(x) :- ¬Left(x)

?

What is the Semantics?

```
Buddy(x,y) :- Likes(x,y),¬Parent(y,x)
```

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

Item('ball') ⟶ ? LeftBox('ball')

```
Left(x) :- ¬Right(x), Item(x)
Right(x) :- ¬Left(x)
```

unsafe

```
Left(x) :- ¬Right(x), Item(x)
Right(x) :- ¬Left(x), Item(x)
```

?

What is the Semantics?

```
Buddy(x,y) :- Likes(x,y),¬Parent(y,x)
```

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

Item('ball') ⟶ ? LeftBox('ball')

```
Left(x) :- ¬Right(x), Item(x)
Right(x) :- ¬Left(x)
```

unsafe

```
Left(x) :- ¬Right(x), Item(x)
Right(x) :- ¬Left(x), Item(x)
```

Item('ball') → LeftBox('ball')

Item('ball') → RightBox('ball')

Adding negation to Datalog is not straightforward!

Negation in Datalog

- Various semantics have been proposed for supporting negation in Datalog
  - In a way that makes sense
- We will look at two:
  - *Semipositive* programs (restricted)
  - *Stratified* programs (standard)

Semipositive Programs

Buddy(x,y) :- Likes(x,y),¬Parent(y,x)

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 relational atom
  - Guarantees domain independence
  - Are following rules safe?

S(x) :- T(y), Edge(z,y), !Edge(x,y) ?

S(x) :- T(y), ! T(x) ?

Semipositive Programs

Buddy(x,y) :- Likes(x,y),¬Parent(y,x)

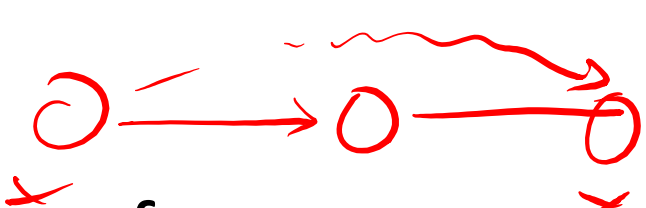
A semipositive program is a program where only EDBs may be negated

- Semantics: same as ordinary Datalog programs
- Safety: rule is safe if every variable occurs in a positive relational atom
  - Guarantees domain independence
  - Are following rules safe?

~~S(x) :- T(y), Edge(z,y), !Edge(x,y)~~

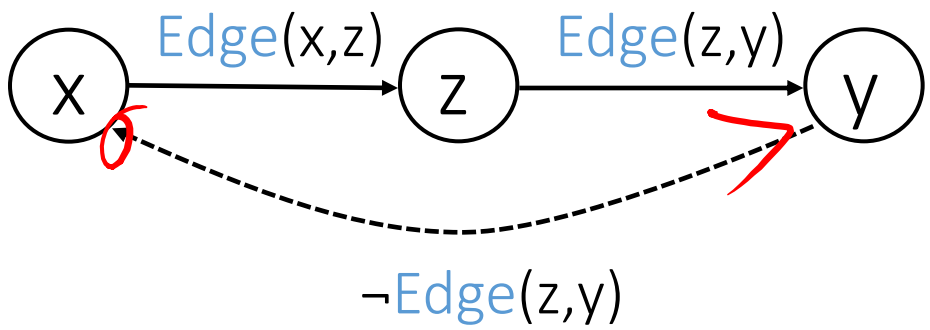
~~S(x) :- T(y), ! T(x)~~

Negated Atoms



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

TwoHop(x,y) :- Edge(x,z), Edge(z,y), ¬Edge(x,y)



Alternative notations: ¬R(args) instead of !R(args)

Stratified Programs

- Let P be a Datalog program and E_0 be set of EDB predicates
- A *stratification* of P is a partitioning of the IDB predicates into disjoint sets $E_1,\ldots,E_k$ where:
  - For $i=1,\ldots,k$, every rule with head in E_i has body predicates only from $E_0,\ldots,E_i$
  - For $i=1,\ldots,k$, every rule with head in E_i can have negated body predicates only from $E_0,\ldots,E_{i-1}$

Practice: unconnected nodes

Compute all pairs of disconnected nodes in a graph.

Practice: unconnected nodes

Compute all pairs of disconnected nodes in a graph.

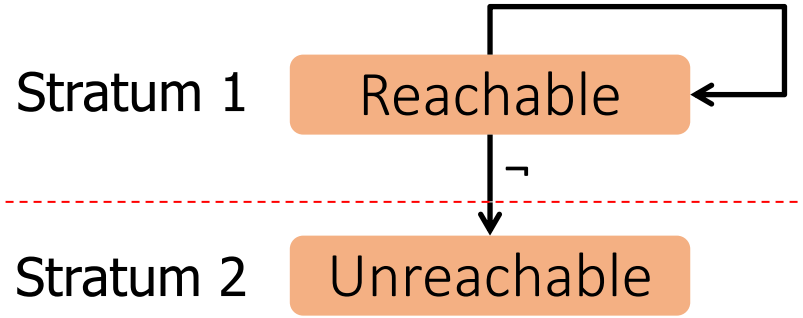
```
Reachable(x,y) :- Edge(x,y)
Reachable(x,y) :- Edge(x,z), Reachable(z,y)
```

Practice: unconnected nodes

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

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Edge(x,y)
Reachable(x,y) :- Edge(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 lowest-stratum-first.
- Once evaluated, treat it as EDB for higher strata.
  think: "topological sort"

Precedence graph :

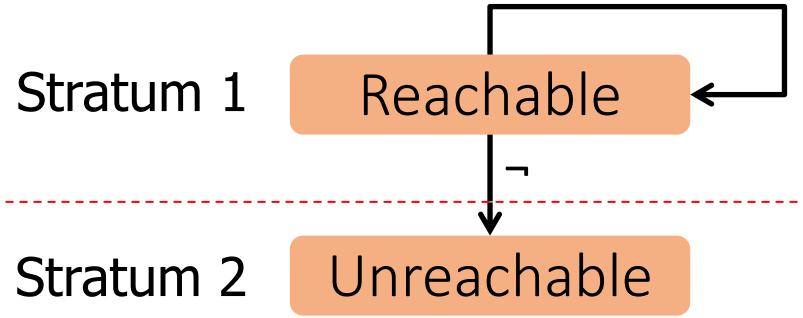
- Nodes = IDB predicates.
- Edge *(p,q)* if predicate *q* depends on *p*.
- Label this arc "!" if the predicate p is negated.

Practice: unconnected nodes

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

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Edge(x,y)
Reachable(x,y) :- Edge(x,z), Reachable(z,y)
Unreachable(x,y) :- Node(x), Node(y), !Reachable(x,y)
```

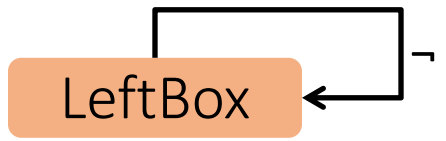


- Straightforward syntactic restriction.
- When the Datalog program is stratified, we can evaluate IDB predicates lowest-stratum-first.
- Once evaluated, treat it as EDB for higher strata.
  think: "topological sort"
- Non-stratified example:

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

Precedence graph :

- Nodes = IDB predicates.
- Edge *(p,q)* if predicate *q* depends on *p*.
- Label this arc "!" if the predicate p is negated.



Example

Person(id, gender, country)
Fake(id)
Parent(parent, child)
Spouse(person1, person2)
Likes(person1,person2)

```
RealPerson(x) :- Person(x,y,z),!Fake(x)
  Relative(x,x) :- RealPerson(x)
  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)
    Buddy(x,y) :- !Relative(x,y),Likes(x,y)
    Buddy(x,y) :- !Relative(x,y),Buddy(x,z),Buddy(z,y)
```

Example by Benny Kimelfeld

A possible stratification

Person(id, gender, country)
Fake(id)
Parent(parent, child)
Spouse(person1, person2)
Likes(person1,person2)

1

```
RealPerson(x) :- Person(x,y,z),!Fake(x)
Relative(x,x) :- RealPerson(x)
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)
```

2

```
Buddy(x,y) :- !Relative(x,y),Likes(x,y)
Buddy(x,y) :- !Relative(x,y),Buddy(x,z),Buddy(z,y)
```

Example by Benny Kimelfeld

Another possible stratification

Person(id, gender, country)
Fake(id)
Parent(parent, child)
Spouse(person1, person2)
Likes(person1,person2)

1
```
RealPerson(x) :- Person(x,y,z),!Fake(x)
```

2
```
Relative(x,x) :- RealPerson(x)
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)
```

3
```
Buddy(x,y) :- !Relative(x,y),Likes(x,y)
Buddy(x,y) :- !Relative(x,y),Buddy(x,z),Buddy(z,y)
```

Example by Benny Kimelfeld

Semantics of Stratified Programs

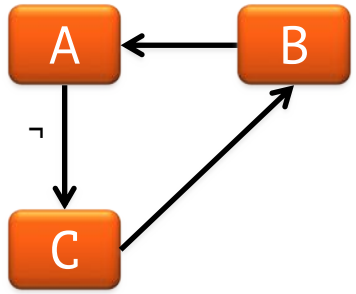
- For i=1,...,k:
  - Compute the IDBs of the stratum E_i
  - Add computed IDBs to the EDBs

- Then, due to the definition of stratification, each E_i can be viewed as semipositive
- Does the result depend on the specific stratification of choice?
  - Answer on the next slide

Theorems on Stratification (1)

- THEOREM 1: All stratifications are equivalent
  - That is, they give the same result on every input
- THEOREM 2: 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)
```



Theorems on Stratification (2)

- THEOREM 3: Non-recursive Datalog programs with negation are stratifiable
  - Via the topological order
- THEOREM 4: Nonrecursive Datalog with negation has the same expressive power as the algebra $\{\sigma_{=}, \pi, \times, \rho, \cup, \setminus\}$
  - Extendable to RA if we add the predicates >, <

Practice

ParentChild(p,c)

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

?

Practice

ParentChild(p,c)

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

first compute for each
person their descendants

---

then use negation

?

Practice

ParentChild(p,c)

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

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

first compute for each person their descendants

then use negation

?

Practice

ParentChild(p,c)

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

```
D(x,y) :- ParentChild(x,y).
D(x,z) :- D(x,y), ParentChild(y,z).
Q(x) :- D('Alice',x), !D('Bob',x).
```

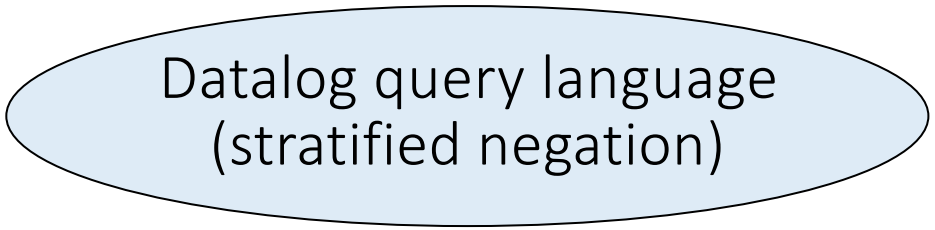
first compute for each person their descendants

then use negation

Outline: 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

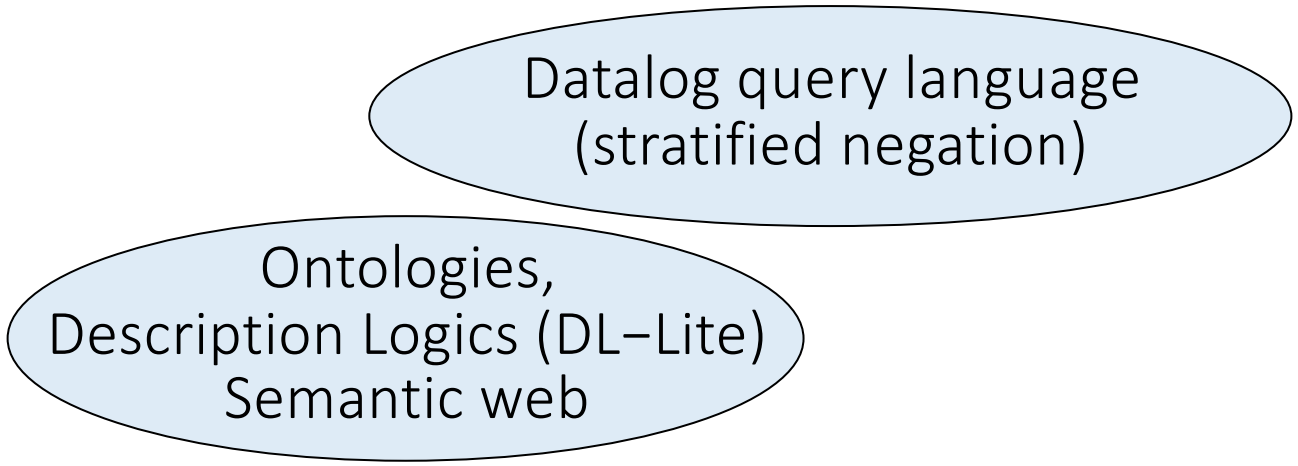
Datalog$^{\pm}$: background



- Much is possible with Datalog

Slide adopted from Andrea Cali

Datalog$^{\pm}$: background

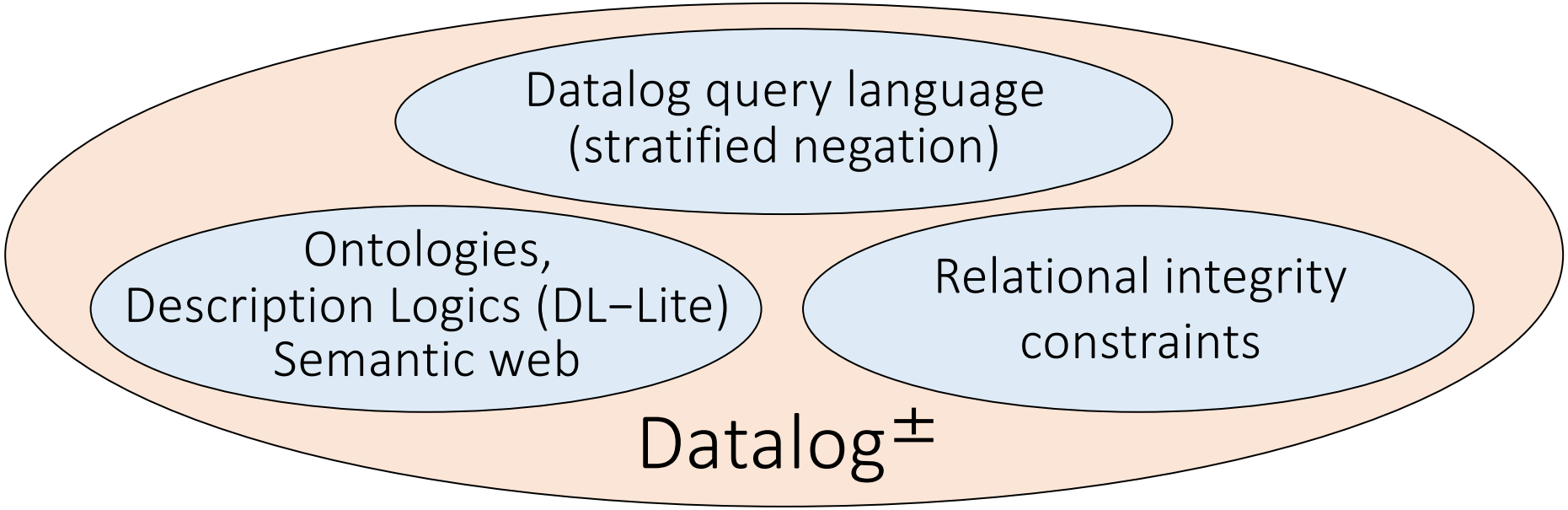


- Much is possible with Datalog
- Much is not (observed e.g. by Patel-Schneider, Horrocks 2006)

Patel-Schneider, Horrocks. A comparison of two modelling paradigms in the Semantic Web. WWW (Semantic Web track). 2006.

Slide adopted from Andrea Cali

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. A comparison of two modelling paradigms in the Semantic Web. WWW (Semantic Web track). 2006.
Cali, Gottlob, Lukasiewicz, Marnette, Pieris. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. LICS 2011.
Slide adopted from Andrea Cali

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	$\text{reports}(X, Y1), \text{reports}(X, Y2) \rightarrow Y1 = Y2$

Slide adopted from Andrea Cali

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:

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

Gottlob, Lukasiewicz, Pieris. Datalog+/-: Questions and Answers. AAAI 2014.

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)

Outline: 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