

Database Theory

Datalog

Original Motivation

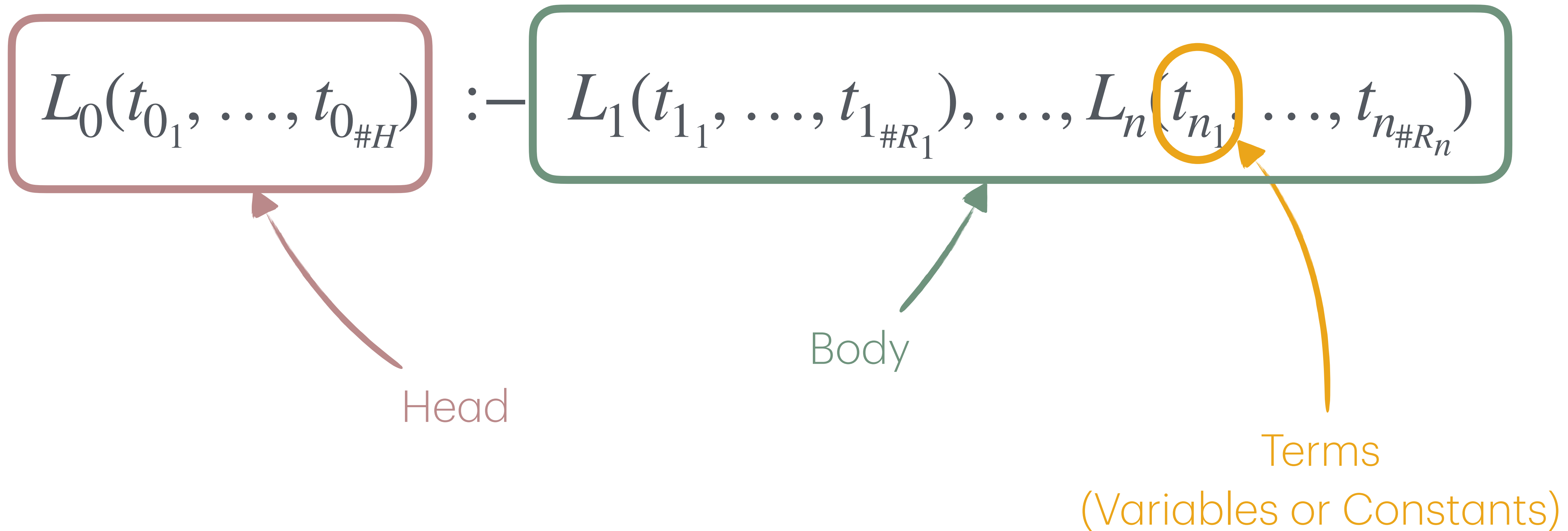
- ◆ Keep the logical foundations of relational calculus
- ◆ Have a simple and intuitively understandable language that can be used by domain experts rather than programmers.
- ◆ Support recursion. This is key for hierarchical data or graph-like structures.
- ◆ Combine all of these with efficiency in evaluation.

Syntax of Datalog

Datalog Rule

$$L_0(t_{0_1}, \dots, t_{0_{\#H}}) \stackrel{\text{"If"}}{:-} L_1(t_{1_1}, \dots, t_{1_{\#R_1}}), \dots, L_n(t_{n_1}, \dots, t_{n_{\#R_n}})$$

Datalog Rule



Datalog Rule

$$L_0(t_{0_1}, \dots, t_{0_{\#H}}) \text{ :- } L_1(t_{1_1}, \dots, t_{1_{\#R_1}}), \dots, L_n(t_{n_1}, \dots, t_{n_{\#R_n}})$$

IDB vs EDB

We partition relation names into to *intensional* and *extensional* predicates (IDBs and EDBs).

A relation name is an IDB if it occurs in the head of a rule.

Intuitively, EDBs are those predicates that come from the database.

Datalog Rule

$$L_0(t_{0_1}, \dots, t_{0_{\#H}}) \text{ :- } L_1(t_{1_1}, \dots, t_{1_{\#R_1}}), \dots, L_n(t_{n_1}, \dots, t_{n_{\#R_n}})$$

$vars(L)$:= the set of variables in literal L

$vars(r)$:= the set of all variables in the rule

$vars(body(r))$:= the set of all variables in the rule body

Datalog Rule

$$L_0(t_{0_1}, \dots, t_{0_{\#H}}) \text{ :- } L_1(t_{1_1}, \dots, t_{1_{\#R_1}}), \dots, L_n(t_{n_1}, \dots, t_{n_{\#R_n}})$$

Important Restrictions

1. Variables in the head must occur in the body,
i.e, $\text{vars}(L_0) \subseteq \text{vars}(\text{body}(r))$.
2. EDB predicates cannot occur in the head of rules.

Datalog Rule

We will simplify notation by using \bar{t}_i to represent the list of terms $t_{i_1}, \dots, t_{i_{\#R_i}}$

$$L_0(t_{0_1}, \dots, t_{0_{\#H}}) \text{ :- } L_1(t_{1_1}, \dots, t_{1_{\#R_1}}), \dots, L_n(t_{n_1}, \dots, t_{n_{\#R_n}})$$

becomes

$$L_0(\bar{t}_0) \text{ :- } L_1(\bar{t}_1), \dots, L_n(\bar{t}_n)$$

Datalog Program

A **Datalog program** is simply a set of Datalog rules.

A **Datalog query** is a pair (Π, P) where Π is a Datalog program and P is an IDB relation name.

Example Program:

$$CoAuthor(y, x) :- Write(x, p), Write(y, p)$$
$$HasEN(x) :- HasEN(y), CoAuthor(y, x)$$
$$HasEN(Erdos).$$

Syntactic convention
for empty body

HasEN and *CoAuthor* are IDBs. When used with a database we expect the EDB relation *Write* to be defined in the database.

Operational Semantics

Some Notation

- ◆ In this context it will be convenient to introduce new notation to check whether a specific tuple is in a specific relation of a database D .
- ◆ For relation name R , we write $R(c_1, \dots, c_\ell) \in D$ to mean that the tuple (c_1, \dots, c_ℓ) is in R^D

In the same sense we will also sometimes define databases via sets of atoms, e.g., the set

$$\{R(1,2), R(2,3), S(a,b)\}$$

Defines the following database:

Relation R	
A	B
1	2
2	3

Relation S	
X	Y
A	B

Bodies are CQs

$$L_0(\bar{t}_0) \text{ :- } L_1(\bar{t}_1), \dots, L_n(\bar{t}_n)$$

The intended meaning of such a rule is that L_0 should contain all the tuples in

$$\{ \bar{t}_0 \mid L_1(\bar{t}_1) \wedge \dots \wedge L_n(\bar{t}_n) \}$$

We will refer to this query of the body of rule r as **bodyCQ**(r).

Immediate Consequence

Let Π be a Datalog program and D a database.

A relational atom $R(c_1, \dots, c_\ell)$ is an *immediate consequence* of Π and D if:

- ♦ $R(c_1, \dots, c_\ell) \in D$, or
- ♦ there exists a rule $r = R(y_1, \dots, y_\ell) : - L_1(\bar{x}_1), \dots, L_n(\bar{x}_n)$ such that (c_1, \dots, c_ℓ) is an answer of **bodyCQ**(r) on D .

Immediate Consequence — Example

Say we have the following program and database

$CoAuthor(y, x): - \quad Write(x, p), Write(y, p)$

Write

Author	Paper
A1	P7
A2	P8
A2	P7
A3	P8

$$\text{bodyCQ}(r) = \{ (y, x) \mid Write(x, p) \wedge Write(y, p) \}$$

Mapping $x \mapsto A1, y \mapsto A2, p \mapsto P7$ satisfies the CQ.

- ➔ $(A2, A1)$ is an answer of $\text{bodyCQ}(r)$ on D .
- ➔ $CoAuthor(A2, A1)$ is an immediate consequence of this program and database.

Immediate Consequence Operator

Intuitively, we can define Datalog semantics by adding all immediate consequences to the databases until there are no more new immediate consequences.

The *immediate consequence operator* T_{Π} is a function that maps a database to a new database as follows:

$$T_{\Pi}(D) = \{ R(\bar{a}) \mid R(\bar{a}) \text{ is an immediate consequence of } \Pi \text{ and } D \}$$

We call D a *fixpoint* of T_{Π} if $T_{\Pi}(D) = D$.

Datalog Fixpoint Semantics

Starting from a Datalog program Π and database D we can now define the result of applying the program to the database as follows:

$$\text{Let } T_{\Pi}^0(D) = D \quad \text{and} \quad T_{\Pi}^{i+1}(D) = T_{\Pi}(T_{\Pi}^i(D))$$

$$\text{and } T_{\Pi}^{\infty} = \bigcup_{i \geq 0} T_{\Pi}^i(D)$$

We write $\Pi_{\text{fp}}(D)$ for *result of program Π applied to D* and define it as

$$\Pi_{\text{fp}}(D) := T_{\Pi}^{\infty}(D)$$

Example

Recall our example program

$CoAuthor(y, x): - \quad Write(x, p), Write(y, p)$
 $HasEN(x) \quad : - \quad HasEN(y), CoAuthor(y, x)$
 $HasEN(Erdos).$

Write

Author	Paper
A1	P7
A2	P8
A2	P7
Erdos	P8

$$T_{\Pi}^1(D) = D \cup \{ HasEN(Erdos), CoAuthor(Erdos, A2), CoAuthor(A2, Erdos), CoAuthor(A1, A2), CoAuthor(A2, A1) \}.$$

Now the **bodyCQ** of the second rule has a new answer on $T_{\Pi}^1(D)$: $(Erdos, A2)$

Example

Recall our example program

$CoAuthor(y, x): - \quad Write(x, p), Write(y, p)$
 $HasEN(x) \quad : - \quad HasEN(y), CoAuthor(y, x)$
 $HasEN(Erdos).$

Write

Author	Paper
A1	P7
A2	P8
A2	P7
Erdos	P8

$$T_{\Pi}^1(D) = D \cup \{ HasEN(Erdos), CoAuthor(Erdos, A2), CoAuthor(A2, Erdos), CoAuthor(A1, A2), CoAuthor(A2, A1) \}.$$

Now the **bodyCQ** of the second rule has a new answer on $T_{\Pi}^1(D)$: $(Erdos, A2)$

$$T_{\Pi}^2(D) = T_{\Pi}^1(D) \cup \{ HasEN(A2) \} \quad T_{\Pi}^3(D) = T_{\Pi}^2(D) \cup \{ HasEN(A1) \} \quad T_{\Pi}^4(D) = T_{\Pi}^3(D)!$$

Finiteness of $T_{\Pi}^{\infty}(D)$

Observe that if $R(c_1, \dots, c_{\ell})$ is an immediate consequence of Π and D , then every c_i must either come from the database or be a fixed part of the program.

Let $Dom(\Pi, D)$ be the set of all constants that occur in at least one of Π and D .

Since both Π and D are finite, so is $Dom(\Pi, D) \Rightarrow$ Finitely many $R(c_1, \dots, c_{\ell})$ to add.

Proposition

Let r be the number of IDBs in Π and a be their maximum arity,

Then $T_{\Pi}^{\infty}(D) = T_{\Pi}^n(D)$ where $n \leq r \cdot |Dom(\Pi, D)|^a$.

Datalog Semantics

The *(fixpoint) semantics of a Datalog query* $q = (\Pi, R)$ are as follows:

$$q(D) := \{(a_1, \dots, a_{\#R}) \in Dom^{\#R} \mid R(a_1, \dots, a_{\#R}) \in \Pi_{fp}(D)\}$$

In our running example $q = (\Pi, HasEN)$ returns the set of all authors with finite Erdos number.

$$\begin{aligned} CoAuthor(y, x) &: - \quad Write(x, p), Write(y, p) \\ HasEN(x) &: - \quad HasEN(y), CoAuthor(y, x) \\ HasEN(Erdos) &. \end{aligned}$$

Logical Semantics

FO Theories

- ◆ Recall, a set Φ of first-order sentences is called a (first-order) theory.
- ◆ A database D is a model of theory Φ if $D \models \varphi$ for every $\varphi \in \Phi$.
- ◆ First-order formulas and databases are always defined over some schema \mathcal{S} .
We will not explicitly mention the schema every time here. We always talk about databases and theories over a single shared schema here.

Datalog as a FO Theory

For Datalog rule $r = R(\bar{x}) :- L_1(\bar{x}_1), \dots, L_n(\bar{x}_n)$ we define

$$\varphi_r \quad := \quad \forall z_1, z_2, \dots, z_m. (L_1(\bar{x}_1) \wedge \dots \wedge L_n(\bar{x}_n)) \rightarrow R(\bar{x})$$

where $\{z_1, \dots, z_m\} = \text{vars}(r)$

Example

$$r = \text{HasEN}(x) \quad :- \quad \text{HasEN}(y), \text{CoAuthor}(y, x)$$

$$\text{vars}(r) = \{x, y\}$$

$$\varphi_r = \forall x, y (\text{HasEN}(y) \wedge \text{CoAuthor}(x, y)) \rightarrow \text{HasEN}(x)$$

Datalog as a FO Theory

For Datalog program Π we define the theory

$$\Phi_{\Pi} \quad := \quad \{ \varphi_r \mid r \in \Pi \}$$

This theory captures logically what we mean by the rules of the program.

A model of this theory satisfies every rule. That is for every “match” in a body, the corresponding head atom has to also be true.

Datalog as a FO Theory

Formally, this means we are interested in models D' such that:

♦ $D' \supseteq D$, and

models extend the
database.

♦ $D' \models \Phi_\Pi$

“all of our rules
are satisfied”

Problem:

there are infinitely many such D' !
Which one determines our query answers?

Minimal Models

For program Π and database D we define the set of **minimal models**:

$$\text{MinMod}(\Pi, D) := \{ D' \mid D \subseteq D', D' \text{ is a } \subseteq\text{-minimal model of } \Phi_{\Pi} \}$$

Minimal Models

For program Π and database D we define the set of **minimal models**:

$$\text{MinMod}(\Pi, D) := \{ D' \mid D \subseteq D', D' \text{ is a } \subseteq\text{-minimal model of } \Phi_{\Pi} \}$$

Example

$$\varphi_r = \forall x, y (HasEN(y) \wedge CoAuthor(x, y)) \rightarrow HasEN(x)$$

$$D = \{ HasEN(Erdos), CoAuthor(Erdos, A1), CoAuthor(A3, A4) \}$$

Minimal Models

For program Π and database D we define the set of **minimal models**:

$$\text{MinMod}(\Pi, D) := \{ D' \mid D \subseteq D', D' \text{ is a } \subseteq\text{-minimal model of } \Phi_\Pi \}$$

Example

$$\varphi_r = \forall x, y (HasEN(y) \wedge CoAuthor(x, y)) \rightarrow HasEN(x)$$

$$D = \{ HasEN(Erdos), CoAuthor(Erdos, A1), CoAuthor(A3, A4) \}$$

Is $D \cup \{ HasEN(A1), HasEN(A3), HasEN(A4) \}$ a minimal model of φ_r ?

Minimal Models

For program Π and database D we define the set of **minimal models**:

$$\text{MinMod}(\Pi, D) := \{ D' \mid D \subseteq D', D' \text{ is a } \subseteq\text{-minimal model of } \Phi_\Pi \}$$

Example

$$\varphi_r = \forall x, y (HasEN(y) \wedge CoAuthor(x, y)) \rightarrow HasEN(x)$$

$$D = \{ HasEN(Erdos), CoAuthor(Erdos, A1), CoAuthor(A3, A4) \}$$

Is $D \cup \{ HasEN(A1), HasEN(A3), HasEN(A4) \}$ a minimal model of φ_r ?

No — It is a model (simply check the assignments for x, y in φ_r) but not minimal.

Minimal Models

For program Π and database D we define the set of **minimal models**:

$$\text{MinMod}(\Pi, D) := \{ D' \mid D \subseteq D', D' \text{ is a } \subseteq\text{-minimal model of } \Phi_\Pi \}$$

Example

$$\varphi_r = \forall x, y (HasEN(y) \wedge CoAuthor(x, y)) \rightarrow HasEN(x)$$

$$D = \{ HasEN(Erdos), CoAuthor(Erdos, A1), CoAuthor(A3, A4) \}$$

Only $D \cup \{ HasEN(A1) \}$ is a **minimal** model of φ_r !

Minimal Models

For program Π and database D we define the set of **minimal models**:

$$\text{MinMod}(\Pi, D) := \{ D' \mid D \subseteq D', D' \text{ is a } \subseteq\text{-minimal model of } \Phi_{\Pi} \}$$

Theorem

For every Datalog program Π and database D :

$$| \text{MinMod}(\Pi, D) | = 1$$

A Proof Sketch

Let $H = D \cup \{ R(\bar{c}) \mid R \text{ is an IDB, } \bar{c} \in \text{Dom}^{\#R} \}$. It is easy to check that $H \models \Phi_\Pi$: every possible consequent of an implication is in H . Hence $|\text{MinMod}(\Pi, D)| \geq 1$.

Assume $|\text{MinMod}(\Pi, D)| = \ell > 1$ and let D_1, \dots, D_ℓ be the minimal models.

We show towards a contradiction that $D' = D_1 \cap \dots \cap D_\ell$ is also a model of Φ_Π and $D' \supseteq D$:

1. For every $1 \leq i \leq \ell$, we have $D \subseteq D_i$. Hence $D \subseteq D'$.
2. For every $\varphi_r \in \Phi_\Pi$, if the left-hand side of an implication (for some assignment to the universally quantified variables) is in some D_i , then the right-hand side will also be in D_i . Hence, if the left-hand side of an implication is in D' , so is the right-hand side.

Minimal Model Semantics

We can use this to define the result of a Datalog program Π on database D in an alternative way:

$$\Pi_{\text{mm}}(D) := \text{the } \subseteq\text{-minimal model of } \Phi_D \text{ that contains } D$$

The definition of semantics of a Datalog query $q = (\Pi, R)$ remains unchanged, except for a change in what we consider the result of a program:

$$q(D) := \{(a_1, \dots, a_{\#R}) \in \text{Dom}^{\#R} \mid R(a_1, \dots, a_{\#R}) \in \Pi_{\text{mm}}(D)\}$$

Connecting the Two Semantics

Theorem

For every Datalog program Π and database D
 $\Pi_{\text{fp}}(D) = \Pi_{\text{mm}}(D)$.

Why?

As long as $\forall z_1, z_2, \dots, z_m. (L_1(\bar{x}_1) \wedge \dots \wedge L_n(\bar{x}_n)) \rightarrow R(\bar{x})$ is not satisfied, the immediate consequence operator will add the respective atom $R(\bar{c})$ for which the left-hand side of the implication was true. Hence $\Pi_{\text{fp}}(D)$ is a model of Φ_{Π} . Furthermore, by definition $D \subseteq \Pi_{\text{fp}}(D)$.

The immediate consequence operator cannot add any other atoms (hence, minimality).

Complexity of Datalog

Data/Combined Complexity

Combined Complexity:

Datalog-Eval is the problem, given input Datalog query $q = (\Pi, R)$ and database D , whether $q(D) \neq \emptyset$

Data Complexity:

In data complexity, we are interested in the complexity for fixed query, that is, only the database is the input.

Formally: for fixed Datalog query $q = (\Pi, R)$, q-Datalog-Eval is the problem, given database D , whether $q(D) \neq \emptyset$.

\mathcal{C} -hardness in data complexity means \mathcal{C} -hardness of q-Datalog-Eval for some q .

We say Datalog-Eval is in \mathcal{C} in data complexity if q-Datalog-Eval is in \mathcal{C} for every q ,

A Simple Algorithm

From our discussion of semantics above we have $\bar{a} \in q(D) \iff R(\bar{a}) \in T_{\Pi}^{\infty}(D)$.

Remember from before that $\Pi_{\text{fp}}(D) = T_{\Pi}^n(D)$ for some $n \leq \ell \cdot |Dom(\Pi, D)|^a$.

Computing T_{Π}^i from T_{Π}^{i-1} requires for each rule r :

$$O(\underbrace{Dom(\Pi, D)^{vars(r)}}_{\text{Number of assignments from variables to domain.}} \cdot \underbrace{|T_{\Pi}^{i-1}(D)|}_{\text{Time to check if the assignment is satisfied in } T_{\Pi}^{i-1}(D)}) \text{ time}$$

Recall that $|T_{\Pi}^{i-1}(D)|$ is at most $\ell \cdot |Dom(\Pi, D)|^a$!

A Simple Algorithm

From our discussion of semantics above we have $\bar{a} \in q(D) \iff R(\bar{a}) \in T_{\Pi}^{\infty}(D)$.

Remember from before that $\Pi_{\text{fp}}(D) = T_{\Pi}^n(D)$ for some $n \leq \ell \cdot |Dom(\Pi, D)|^a$.

Computing T_{Π}^i from T_{Π}^{i-1} requires:

$$O(|\Pi| \cdot Dom(\Pi, D)^{maxvars} \cdot \ell \cdot |Dom(\Pi, D)|^a) \text{ time}$$

Repeating at most n times we can compute $T_{\Pi}^{\infty}(D)$ – and therefore decide $\bar{a} \in q(D)$ – in time

$$O(|\Pi| \cdot Dom(\Pi, D)^{maxvars} \cdot \ell^2 \cdot |Dom(\Pi, D)|^{2a})$$

A Simple Algorithm

$$O(|\Pi| \cdot \text{Dom}(\Pi, D)^{\text{maxvars}} \cdot r^2 \cdot |\text{Dom}(\Pi, D)|^{2a})$$

Combined Complexity

The inputs are Π and D .

Both *maxvars* and *a* depend on Π ,
i.e., our algorithm requires
exponential time in combined
complexity.

Data Complexity

The program Π is fixed, only the
database D is an input.

All the exponents in our time bound
are thus fixed, i.e., they are constants.

Our algorithm requires polynomial
time in data complexity!

This algorithm is very naïve.
Can we do significantly better?

Complexity of Datalog

Theorem

Datalog-Eval is ExpTime-complete.

Datalog-Eval is PTime-complete in data-complexity

Combined Complexity

We show ExpTime-hardness by reducing from acceptance in an exponential time Turing machine.

Let $L \in \text{ExpTime}$, and let $M = (Q, \Sigma, \delta, q_0, q_T)$ be a deterministic TM that decides L in exponential time.

Reminder TM:
 Q .. states
 Σ ...alphabet
 δ ...transition function
 q_0 ...start state
 q_T ...accept state

Our goal: On input string w , compute in polynomial time a database D and Datalog query $q = (\Pi, \text{Accept})$ such that

$$M \text{ accepts } w \iff q(D) \neq \emptyset$$

Combined Complexity

High-level overview of the reduction:

Any transition $\delta(q, a) = (q', b, dir)$ can be understood as a rule:

if at time t in state q the head reads a ,
then at time $t + 1$, the state is q' , the symbol a under the head gets
replaced by b , and the head is moved according to dir

We can express these statements easily in Datalog. If we can follow these computations for 2^m steps (where m is $|w|^k$ for some $k \in \mathbb{N}$), we can capture the full computation of M on w .

Constructing a Long Chain

For each $i \in [m - 1]$:

$$Succ^{i+1}(z, \bar{x}, z, \bar{y}) : - Succ^i(\bar{x}, \bar{y}), Low^1(z)$$

$$Succ^{i+1}(z, \bar{x}, z, \bar{y}) : - Succ^i(\bar{x}, \bar{y}), High^1(z)$$

$$Succ^{i+1}(z, \bar{x}, v, \bar{y}) : - Succ^1(z, v), Low^i(\bar{x}), High^i(\bar{y})$$

$$High^{i+1}(x, \bar{y}) : - High^1(x), High^i(\bar{y})$$

$$Low^{i+1}(x, \bar{y}) : - Low^1(x), Low^i(\bar{y})$$

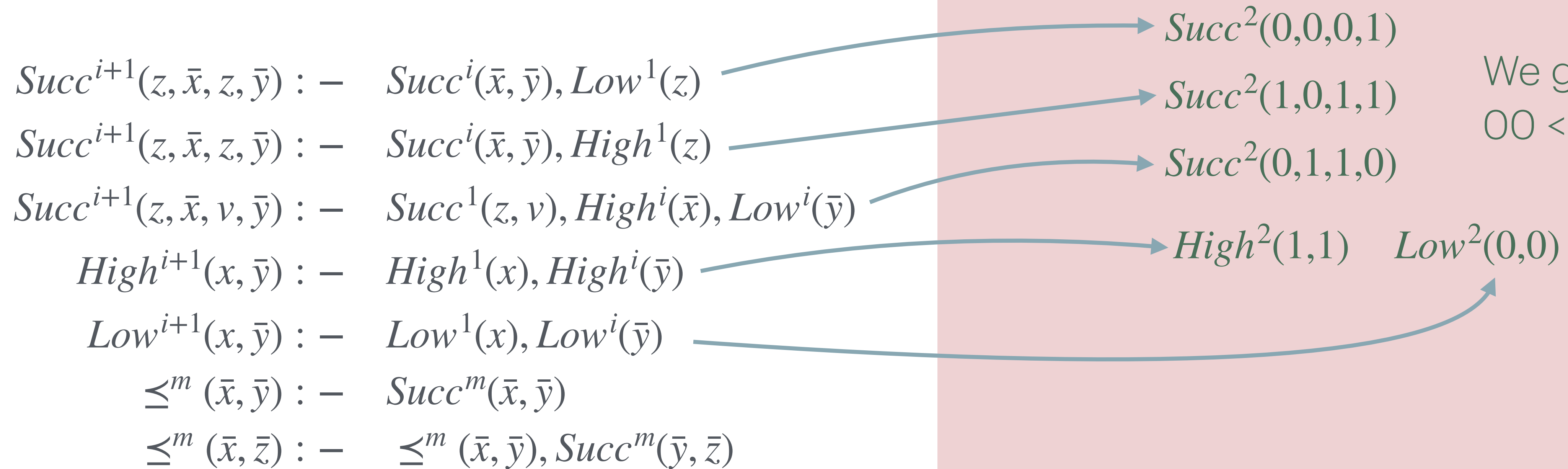
$$\leq^m (\bar{x}, \bar{y}) : - Succ^m(\bar{x}, \bar{y})$$

$$\leq^m (\bar{x}, \bar{z}) : - \leq^m (\bar{x}, \bar{y}), Succ^m(\bar{y}, \bar{z})$$

Intuitively, this is a compact way to construct a sequence of length 2^m .

Example

$Succ^1(0,1), High(1), Low(0)$



We get the ordering
 $00 < 01 < 10 < 11$

Example

$$\begin{aligned}
 Succ^{i+1}(z, \bar{x}, z, \bar{y}) &: - & Succ^i(\bar{x}, \bar{y}), Low^1(z) \\
 Succ^{i+1}(z, \bar{x}, z, \bar{y}) &: - & Succ^i(\bar{x}, \bar{y}), High^1(z) \\
 Succ^{i+1}(z, \bar{x}, v, \bar{y}) &: - & Succ^1(z, v), High^i(\bar{x}), Low^i(\bar{y}) \\
 High^{i+1}(x, \bar{y}) &: - & High^1(x), High^i(\bar{y}) \\
 Low^{i+1}(x, \bar{y}) &: - & Low^1(x), Low^i(\bar{y}) \\
 \leq^m(\bar{x}, \bar{y}) &: - & Succ^m(\bar{x}, \bar{y}) \\
 \leq^m(\bar{x}, \bar{z}) &: - & \leq^m(\bar{x}, \bar{y}), Succ^m(\bar{y}, \bar{z})
 \end{aligned}$$

$$Succ^1(0,1), High(1) \boxed{Low(0)}$$



$$Succ^2(0,0,0,1)$$

$$\boxed{Succ^2(1,0,1,1)}$$

$$Succ^2(0,1,1,0)$$

$$High^2(1,1) \quad Low^2(0,0)$$

We get the ordering
00 < 01 < 10 < 11



$$Succ^2(0,0,1,0,1,0)$$

Example

$$\begin{aligned}
 Succ^{i+1}(z, \bar{x}, z, \bar{y}) &: - & Succ^i(\bar{x}, \bar{y}), Low^1(z) \\
 Succ^{i+1}(z, \bar{x}, z, \bar{y}) &: - & Succ^i(\bar{x}, \bar{y}), High^1(z) \\
 Succ^{i+1}(z, \bar{x}, v, \bar{y}) &: - & Succ^1(z, v), High^i(\bar{x}), Low^i(\bar{y}) \\
 High^{i+1}(x, \bar{y}) &: - & High^1(x), High^i(\bar{y}) \\
 Low^{i+1}(x, \bar{y}) &: - & Low^1(x), Low^i(\bar{y}) \\
 \leq^m(\bar{x}, \bar{y}) &: - & Succ^m(\bar{x}, \bar{y}) \\
 \leq^m(\bar{x}, \bar{z}) &: - & \leq^m(\bar{x}, \bar{y}), Succ^m(\bar{y}, \bar{z})
 \end{aligned}$$

$$Succ^1(0,1), High(1), Low(0)$$



$$Succ^2(0,0,0,1)$$

$$Succ^2(1,0,1,1)$$

$$Succ^2(0,1,1,0)$$

$$High^2(1,1) \quad Low^2(0,0)$$

We get the ordering
00 < 01 < 10 < 11



$$Succ^2(0,0,1,0,1,0)$$

⋮

$$Succ^2(0,0,0,0,0,1)$$

⋮

The Database

We only need a very basic database from which our successor relationship can be constructed.

$$D = \{ Succ^1(0,1), High^1(1), Low^1(0) \}$$

Note that the database in this construction is *independent* of the input w !

The Starting State

For input word $w = a_0a_1\cdots a_\ell$ we add the following rules:

$$\begin{aligned}
 State_s(\bar{x}) &: - Low^m(\bar{x}) && \text{"At time 0, the machine is in the starting state"} \\
 Symbol_{a_0}(\bar{x}, \bar{x}) &: - Low^m(\bar{x}) && \text{"At time 0, the symbol } a_0 \text{ is on cell 0 of the tape"} \\
 Symbol_{a_1}(\bar{x}_0, \bar{x}_1) &: - Low^m(\bar{x}_0), Succ^m(\bar{x}_0, \bar{x}_1) && \text{"At time 0, the symbol } a_1 \text{ is on cell 1 of the tape"} \\
 &\vdots \\
 Symbol_{a_\ell}(\bar{x}_0, \bar{x}_\ell) &: - Low^m(\bar{x}_0), Succ^m(\bar{x}_0, \bar{x}_1), \dots Succ^m(\bar{x}_{\ell-1}, \bar{x}_\ell) \\
 Symbol_{\sqcup}(\bar{x}_0, \bar{y}) &: - Low^m(\bar{x}_0), Succ^m(\bar{x}_0, \bar{x}_1), \dots Succ^m(\bar{x}_{\ell-1}, \bar{x}_\ell), \leq^m(\bar{x}_\ell, \bar{y}) \\
 Head(\bar{x}, \bar{x}) &: - Low^m(\bar{x})
 \end{aligned}$$

We use $Symbol_a(t, c)$ to express that at time t cell c contains symbol a . $|\Sigma|$ many

Similarly, $Head(t, c)$ means that at time t , the head is over cell c .

Transitions as Rules

For every transition $\delta(q, a) = (q', b, \rightarrow)$ add the rules:

$$State_{q'}(\bar{z}) : - \quad State_q(\bar{x}), Head(\bar{x}, \bar{y}), Symbol_a(\bar{x}, \bar{y}), Succ^m(\bar{x}, \bar{z})$$

$$Symbol_b(\bar{z}, \bar{y}) : - \quad State_q(\bar{x}), Head(\bar{x}, \bar{y}), Symbol_a(\bar{x}, \bar{y}), Succ^m(\bar{x}, \bar{z})$$

$$Head(\bar{z}, \bar{v}) : - \quad State_q(\bar{x}), Head(\bar{x}, \bar{y}), Symbol_a(\bar{x}, \bar{y}), Succ^m(\bar{x}, \bar{z}), Succ^m(\bar{y}, \bar{v})$$

It is straightforward to adapt the *Head* rule to the other directions.

Inertia and Acceptance

The final missing part is to preserve unchanged symbols over time:

$$Symbol_a(\bar{v}, \bar{y}) : - \quad Symbol_a(\bar{x}, \bar{y}), Head(\bar{x}, \bar{z}), \leq^m (\bar{y}, \bar{z}), Succ^m(\bar{x}, \bar{v})$$

$$Symbol_a(\bar{v}, \bar{y}) : - \quad Symbol_a(\bar{x}, \bar{y}), Head(\bar{x}, \bar{z}), \leq^m (\bar{z}, \bar{y}), Succ^m(\bar{x}, \bar{v})$$

The first rule propagates the cells that come before the head to the next timepoint, the second propagates the cells after the head.

Finally, we check whether our simulation of the machine reaches the accepting state q_{\top}

$$Accept : - \quad State_{q_{\top}}(\bar{x})$$

Data Complexity

For data complexity we need to reduce some problem to answering a fixed Datalog query.

That is, we give a fixed query $q = (\Pi, R)$ that doesn't depend on the problem input. For every input w , the reduction computes a database D such that

$$q(D) \neq \emptyset \iff w \text{ is an accepting input}$$

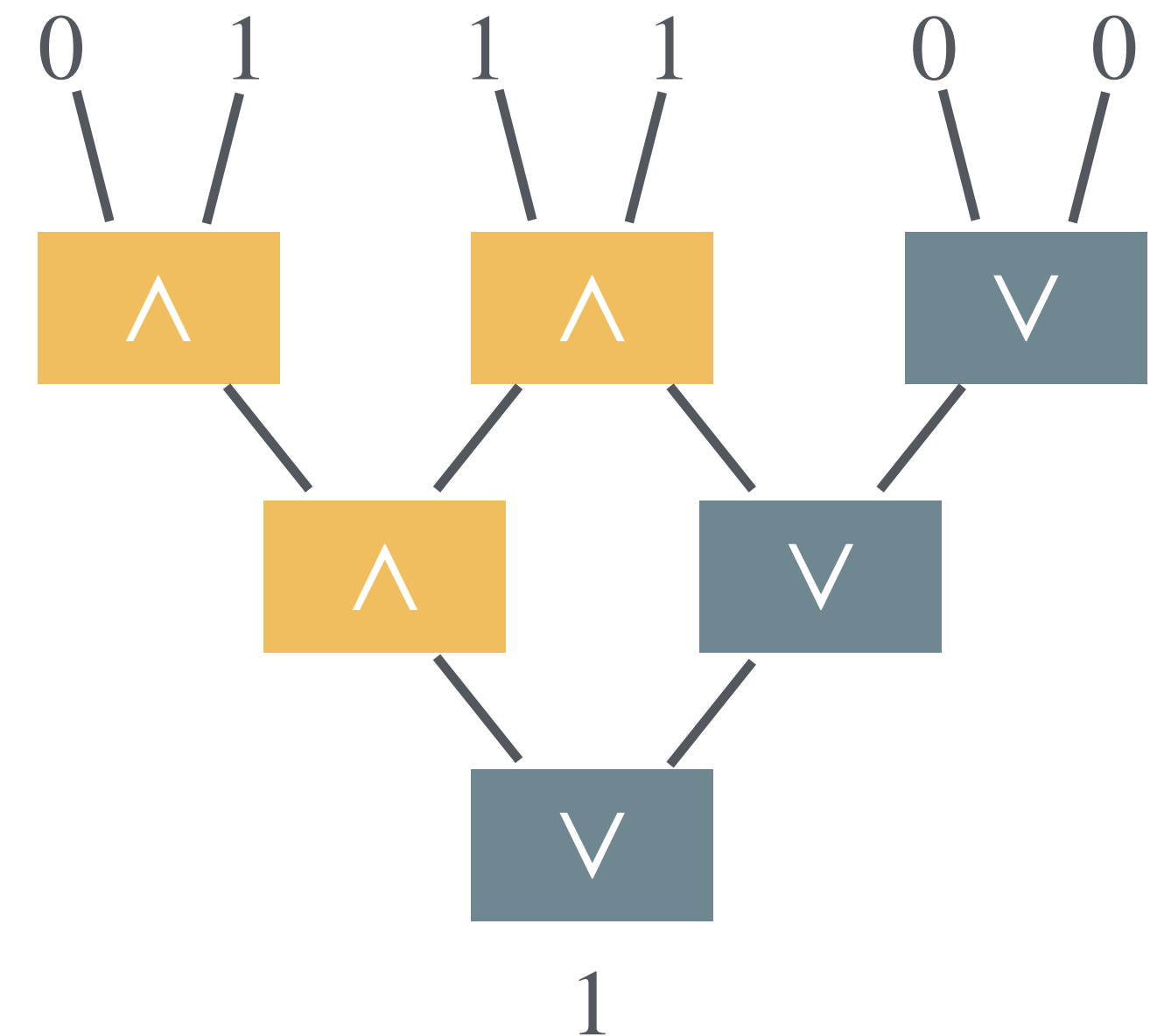
Data Complexity

We will reduce from monotone Boolean-circuit evaluation (MCVP)

MCVP

Input: An n -input, monotone Boolean circuit C , and a string $w \in \{0,1\}^n$

Output: accepts iff C outputs true on input w .



Data Complexity

We fix the Datalog query $q = (\Pi, \mathbf{Accept})$ with program Π :

$$True(x) : - \quad Or(x, y, z), True(y)$$
$$True(x) : - \quad Or(x, y, z), True(z)$$
$$True(x) : - \quad And(x, y, z), True(y), True(z)$$
$$\mathbf{Accept} : - \quad End(x), True(x)$$

The program simulates a monotone Boolean circuit specified in the database.

To complete the proof we need to give a *LogSpace* algorithm that takes a circuit as input and outputs a corresponding database.

Stratified Negation

Motivation

There are many natural situations where we would like to extend Datalog with the option to negate body atoms.

$$\textit{Unassigned}(E) \text{ :- } \textit{Employee}(E), \neg \textit{WorksOn}(E, _P) .$$

Negation in Datalog

$$P : - \neg Q$$

$$Q : - \neg P$$

What is supposed to happen here?

Key problem:

Relation is defined on the basis of its own negation

Semipositive Programs

A Datalog program with negation is **semipositive** if only EDB relation names are negated.

Say we have EDB relation name *Knows*, then the following program is semipositive:

$$\textit{Stranger}(x, y) : - \neg \textit{Knows}(x, y)$$

Semantics of semipositive programs are straightforward:

1. For every negated EDB introduce relation name R_{\neg} and add $R_{\neg}^D = \text{Dom}^{\#R} \setminus R^D$ to the database
2. replace every occurrence of $\neg R(\bar{x})$ with $R_{\neg}(\bar{x})$
3. The program now has no more negation, evaluate as usual.

Stratified Negation

Semipositive is too restrictive, but often programs can be seen as a sequence of multiple semipositive programs!

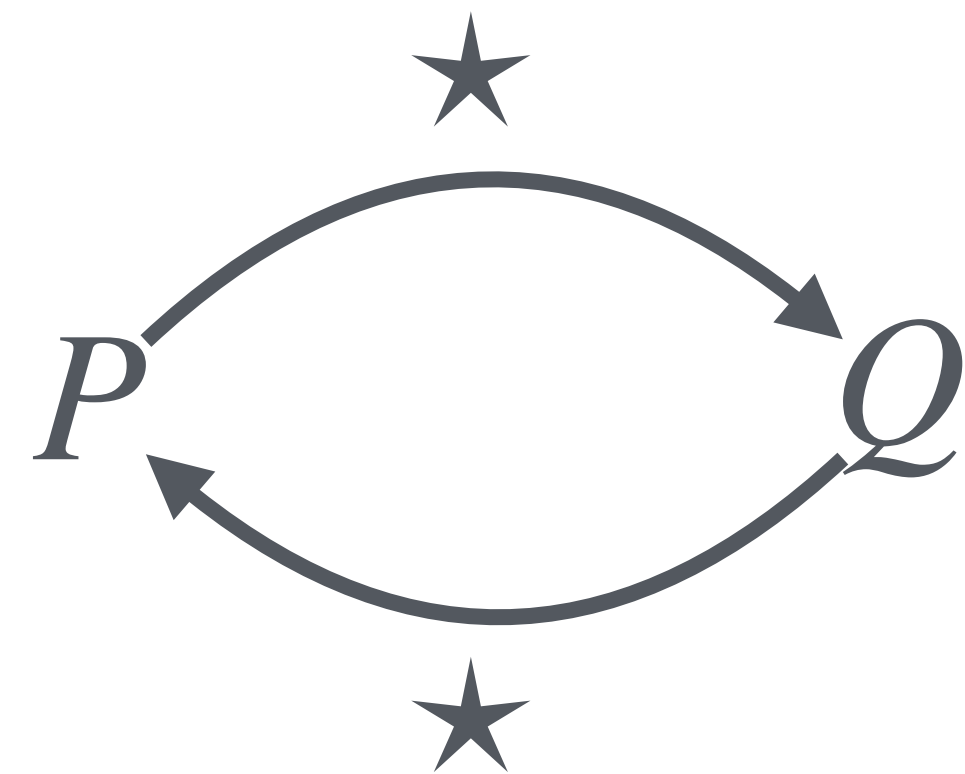
Dependency Graph of program Π is a directed graph:

- ◆ Vertices $:=$ relation names in Π
- ◆ There is an arc $p \rightarrow q$ if there is a rule in Π where p is the head relation symbol, and q is in the body.
- ◆ We mark arcs $p \rightarrow q$ with a star \star if q in the body is negated.

Dependency Graph

$$P : - \neg Q$$

$$Q : - \neg P$$

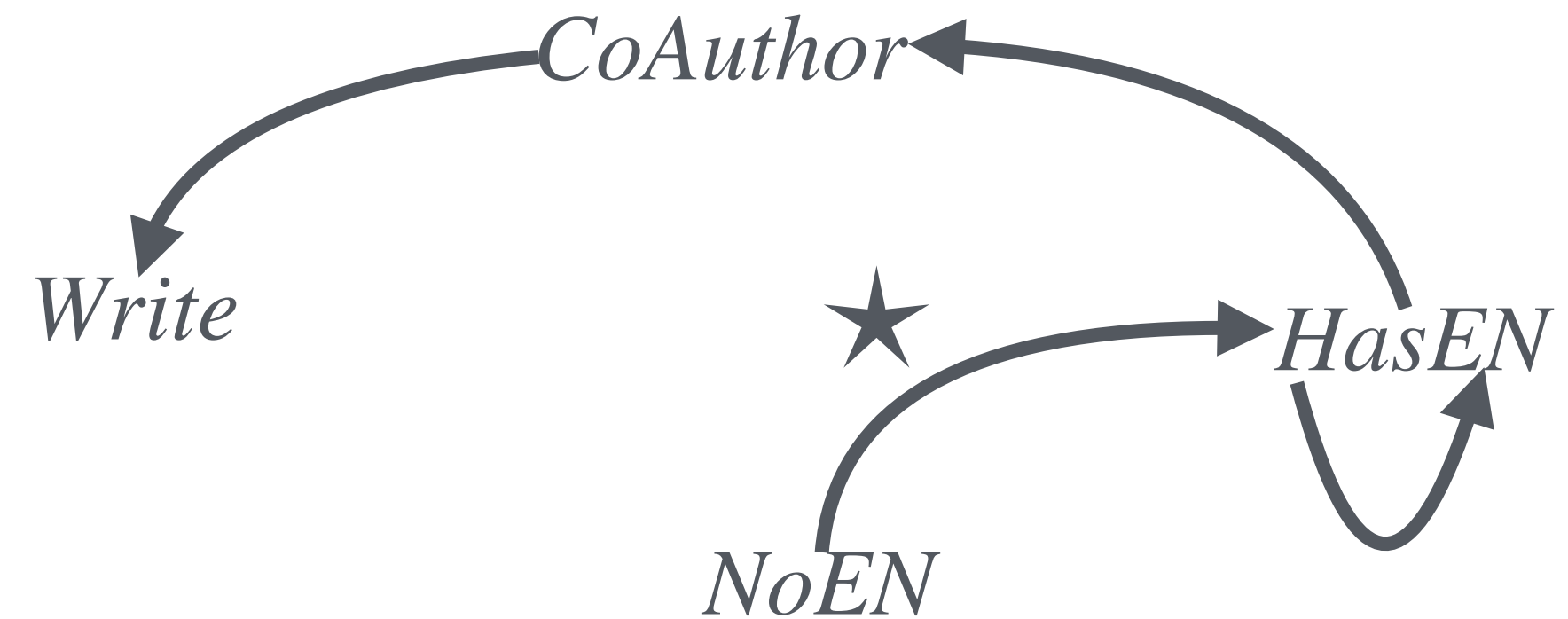


$$CoAuthor(y, x) : - \quad Write(x, p), Write(y, p)$$

$$HasEN(x) : - \quad HasEN(y), CoAuthor(y, x)$$

$$NoEN(x) : - \quad \neg HasEN(x)$$

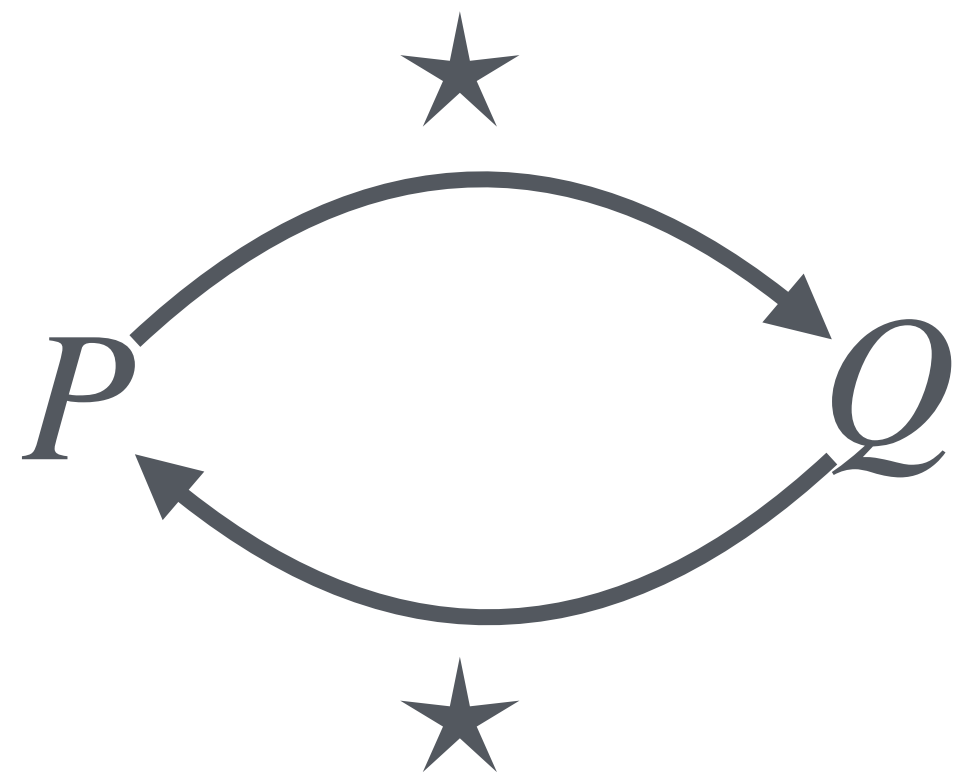
$$HasEN(Erdos).$$



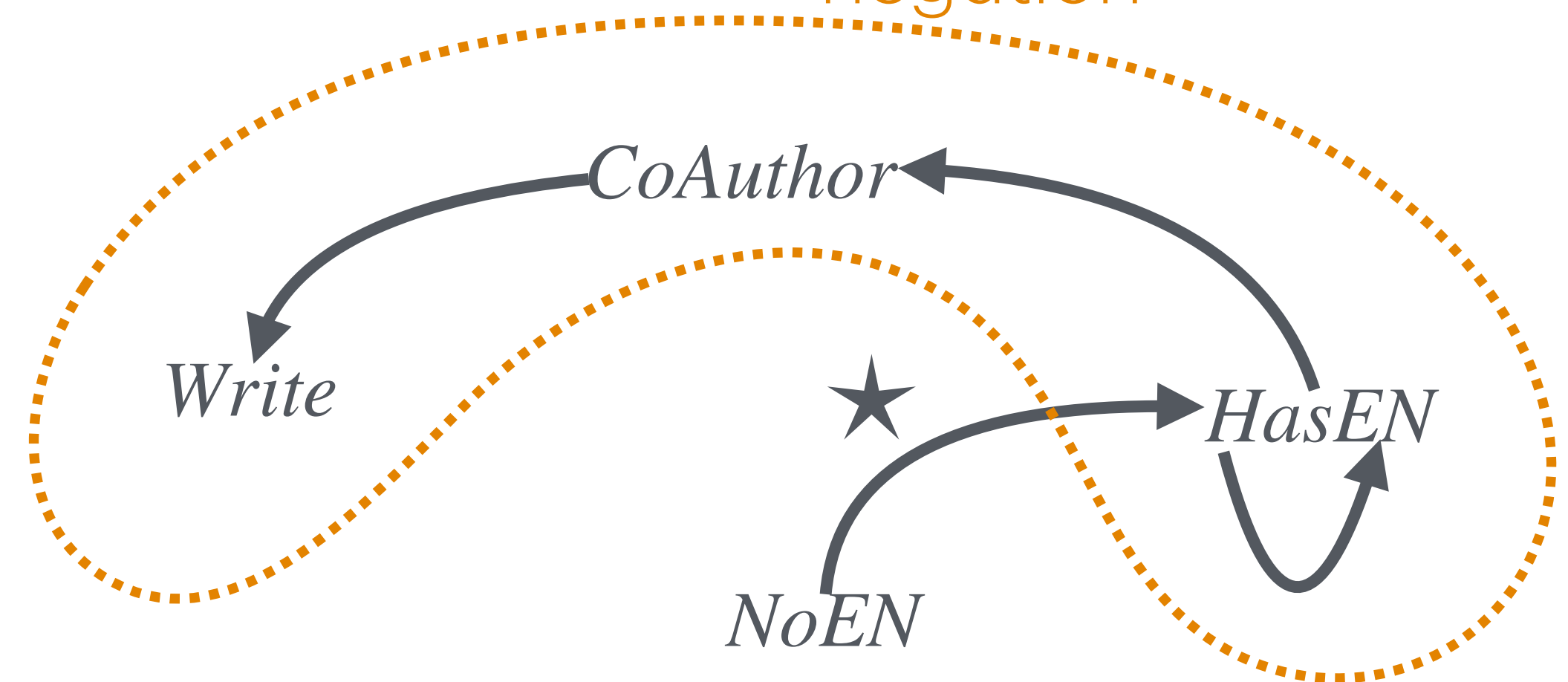
Stratifiable Programs

If the dependency graph of Π has no directed cycle that contains an edge marked with \star we call Π stratifiable.

Intuition: we can solve this "stratum" before checking negation



Not stratifiable



Stratifiable

Stratification

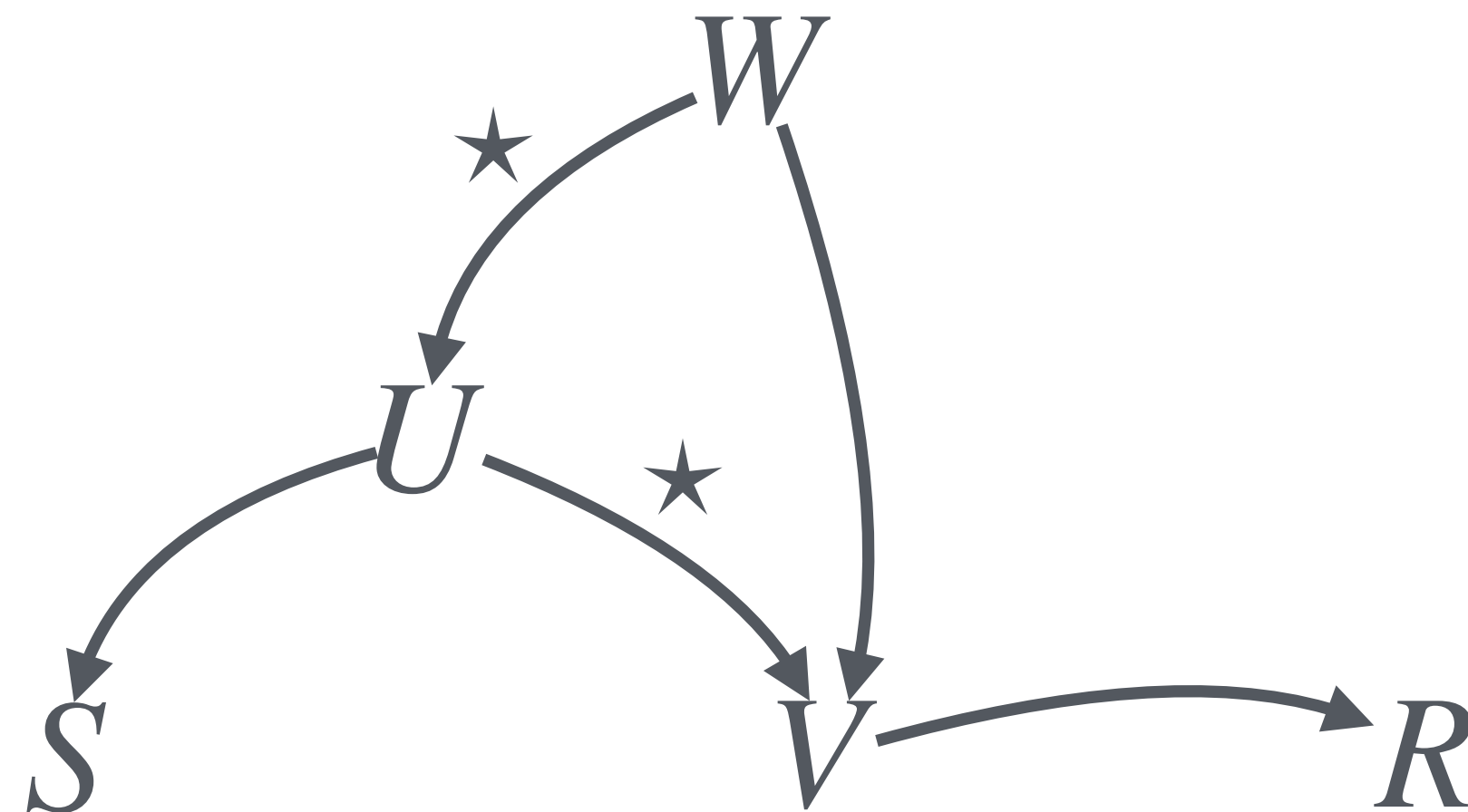
A stratification is a function $\sigma : \mathbf{Rel} \rightarrow \mathbb{N}$ that assign to every relation name a number such that:

- ♦ If $R \in \mathbf{Rel}$ is an EDB, then $\sigma(R) = 0$.
- ♦ If there is a rule with R in its head and P in its body, then $\sigma(R) \geq \sigma(P)$.
- ♦ If there is a rule with R in its head and $\neg P$ in its body, then $\sigma(R) > \sigma(P)$

Example

.....
: If $R \in \mathbf{Rel}$ is an EDB, then :
: $\sigma(R) = 0$. :
.....

$$\begin{aligned} V(x, y) &: - R(x, x), R(y, y) \\ U(x, y) &: - S(x, y), S(y, z), \neg V(x, y) \\ W(x, y) &: - \neg U(x, y), V(y, x) \end{aligned}$$



Level 0: R, S, V

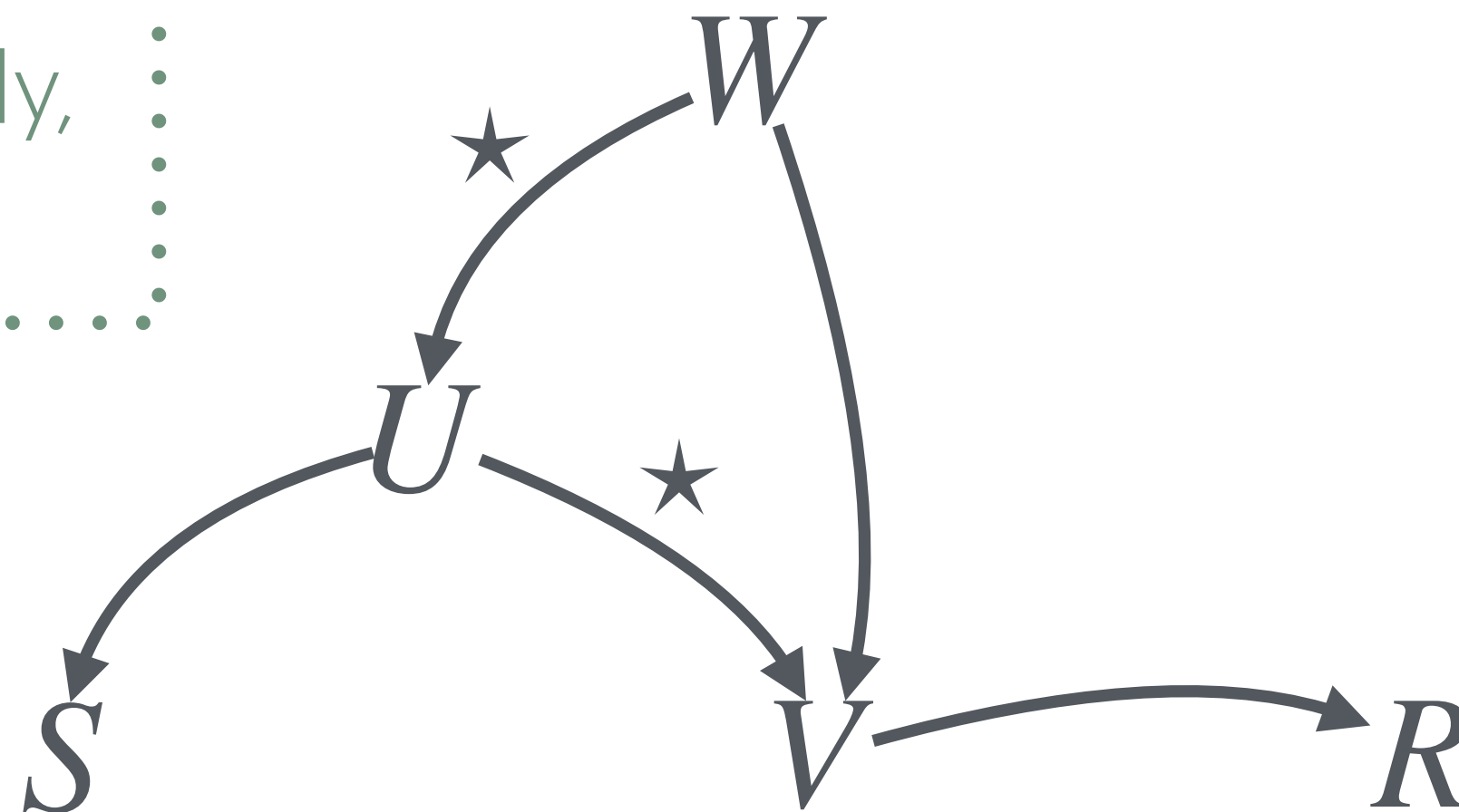
Example

$$V(x, y) \quad : - \quad R(x, x), R(y, y)$$

$$U(x, y) \quad : - \quad S(x, y), S(y, z), \neg V(x, y)$$

$$W(x, y) \quad : - \quad \neg U(x, y), V(y, x)$$

.....
: If there is a rule with R in its
: head and $\neg P$ in its body,
: then $\sigma(R) > \sigma(P)$
:



Level 1: U

Level 0: R, S, V

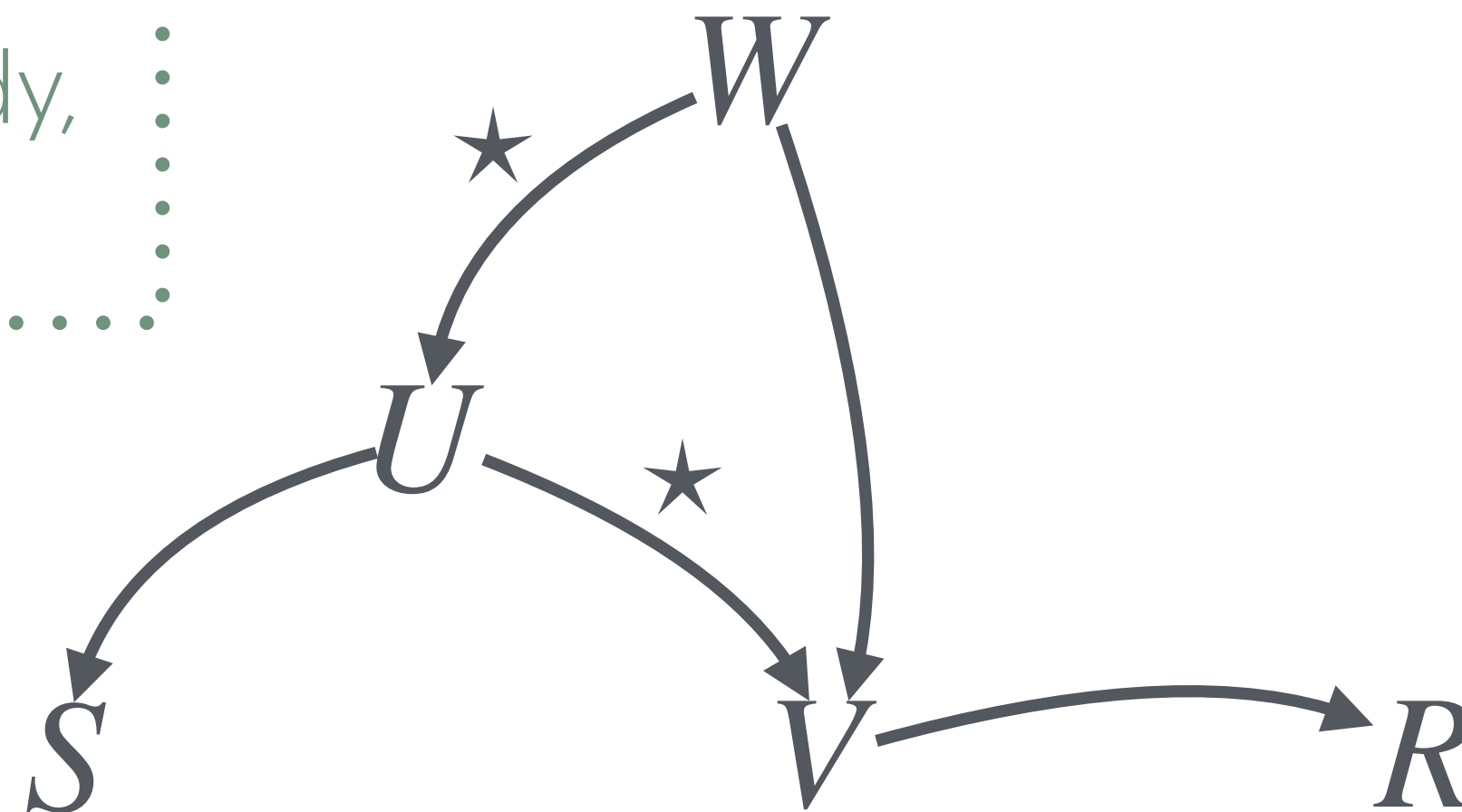
Example

$$V(x, y) \quad : - \quad R(x, x), R(y, y)$$

$$U(x, y) \quad : - \quad S(x, y), S(y, z), \neg V(x, y)$$

$$W(x, y) \quad : - \quad \neg U(x, y), V(y, x)$$

.....
: If there is a rule with R in its
: head and $\neg P$ in its body,
: then $\sigma(R) > \sigma(P)$
:



Level 2: W

Level 1: U

Level 0: R, S, V

Stratified Negation — Semantics

For a given stratification σ , partition the program Π into programs Π_1, Π_2, \dots :

Rule r with head relation name R is in program $\Pi_{\sigma(R)}$.

$$\begin{array}{lll} \Pi_1 & V(x, y) & : - \quad R(x, x), R(y, y) \\ \Pi_2 & U(x, y) & : - \quad S(x, y), S(y, z), \neg V(x, y) \\ \Pi_3 & W(x, y) & : - \quad \neg U(x, y), V(y, x) \end{array}$$

Define $D_0 = D$, $D_i = \Pi_i(D_{i-1})$. The result of a stratified program $\Pi_\sigma(D) := D_\ell$ where ℓ is the maximum level assigned by σ .

Now every program Π_i is semipositive: every negated atom occurs in a lower stratum
 \Rightarrow it becomes an EDB in this stratum-wise execution.

Stratified Negation — Semantics

One problem is left, semantics depend on a specific stratification σ .
However, actually any stratification leads to the same result, providing us with robust semantics for Datalog with stratified negation.

Theorem

Let σ_1, σ_2 be two stratifications of Datalog program Π . Then $\Pi_{\sigma_1}(D) = \Pi_{\sigma_2}(D)$.

Conclusion

The Real World



Classic Datalog engine with some extra features.

<https://souffle-lang.github.io/index.html>



In particular, clingo — a popular ASP solvers — can be used as a Datalog engine with extra features.

<https://potassco.org/clingo/>

Try online:

<https://potassco.org/clingo/run/>

The Real World



Modern Datalog engine that additionally supports stratified aggregation, value invention, and many QoL extensions.

<https://knowsys.github.io/nemo-doc/>

Try it in the browser:

<https://tools.iccl.inf.tu-dresden.de/nemo/>



Datalog-based language that compiles to SQL.

Specialised for data analysis.

<https://logica.dev/>

Negation

Well-founded Negation

Three-values logic where atoms can also have “undefined” truth.

For example, in mutual recursion, both atoms have undefined truth.

Stable Negation

Consider anything that is not necessarily true as false. Programs can have multiple incomparable models!

$P \text{ :- } \neg Q$
 $Q \text{ :- } \neg P$ has two stable models:
 $\{P\}$ and $\{Q\}$

Datalog[±]

Explore the details
in a literature research!

We commonly need to deal with incomplete data.
We might need to invent new values!

$$\exists x \textit{Manager}(x, y) : - \textit{Employee}(y)$$

Creates a much more powerful language (the $+$).
So powerful that answering queries becomes undecidable.

We therefore study restrictions that allow for value invention but
keep the language decidable (the $-$).

Restricted Datalog

Restrictions to standard Datalog have also been studied.

Some examples:

- ◆ ***Linear Datalog***

All rules have at most one IDB in their body. This guarantees that any recursion is “linear”, i.e., there are no joins on recursion.

PSpace-complete in combined complexity, NL-complete in data complexity.

- ◆ ***Non-recursive Datalog***

No recursion is allowed, i.e., no cycles in the dependency graph at all.

In the Codd’s Theorem exercise you will see that non-recursive Datalog with negation this is equivalent to relational algebra.