THE UNIVERSITY of EDINBURGH informatics

University of Cyprus Department of Computer Science

Explaining Answers to Datalog Queries

Andreas Pieris

University of Edinburgh
University of Cyprus

based on joint work with Marco Calautti, Ester Livshits and Markus Schneider

Datalog 2.0, Dallas, October 11, 2024

Datalog: Another Success Story of LiCS

- Important recursive query language
- Benchmark for other query languages
- Has influenced the SQL3 standard
- Successfully used in many applications, e.g., code querying, web data extraction, business process, modeling and automation, ontological query answering, …
- Large projects and some companies are "Datalog-based"

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

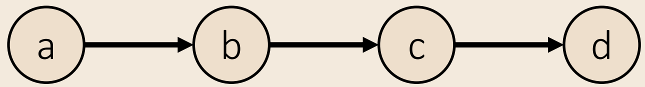
RDFOX

DeepReason.ai

RelationalAI

Datalog at First Glance

Edge	start	end
	a	b
	b	c
	c	d

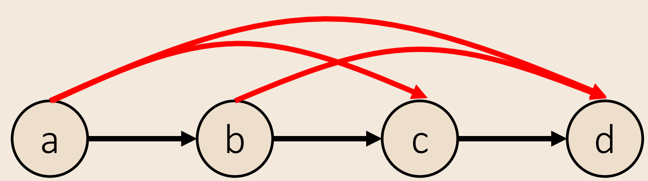



```
TrClosure(x,y) :- Edge(x,y)

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

Answer(x,y) :- TrClosure(x,y)
```

Answer	start	end
	a	b
	a	c
	a	d
	b	c
	b	d
	c	d



Syntax of Datalog

A Datalog rule is an expression of the form

$$\underbrace{R_0(\overline{x_0})}_{\text{head}} \text{ :- } \underbrace{R_1(\overline{x_1}),\ldots,R_n(\overline{x_n})}_{\text{body}}$$

- $n \geq 0$ - the body might be empty
- $R_0,\ldots,R_n$ are relation names
- $\overline{x_0},\ldots,\overline{x_n}$ are tuples of variables
- Each variable in the head occurs also in the body - safety condition

Syntax of Datalog

- **Datalog program P:** a finite set of Datalog rules
- **Extensional relation:** does not occur in the head of a rule of P
- **Intensional relation:** occurs in the head of some rule of P
- **Extensional schema:** the set of extensional relations of P
- **Intensional schema:** the set of intensional relations of P
- **Datalog query Q:** a pair of the form (P, Answer), where P is a Datalog program, and Answer a distinguished intensional relation (the output relation)

Semantics of Datalog

- Semantics: a mapping from databases of the extensional schema to databases of the intensional schema, and the answer is determined by the output relation

Edge	start	end
	a	b
	b	c
	c	d

→

Answer	start	end
	a	b
	a	c
	a	d
	b	c
	b	d
	c	d

- Equivalent ways for defining the semantics
  - Model-theoretic: logical sentences asserting a property of the result
  - Fixpoint: solution of a fixpoint procedure
  - Proof-theoretic: based on proof trees

Proof-theoretic Semantics of Datalog

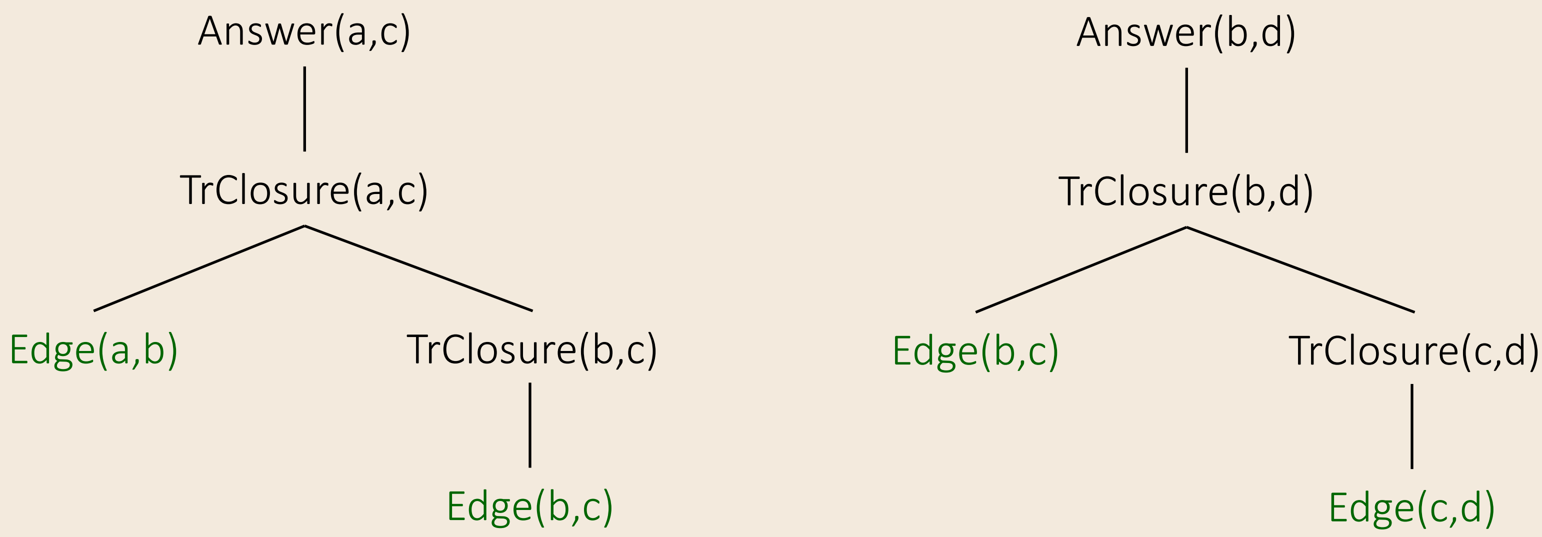
- Given a database D and a Datalog query Q = (P, Answer), we first define the output of P on D, denoted P(D), and then collect the content of the relation Answer in P(D)

- We define the notion of proof of a relational atom w.r.t. D and P, and then the output of P on D are all the atoms that can be proven - "proof-theoretic semantics"

Proof Tree by Example

D = {Edge(a,b), Edge(b,c), Edge(c,d)}

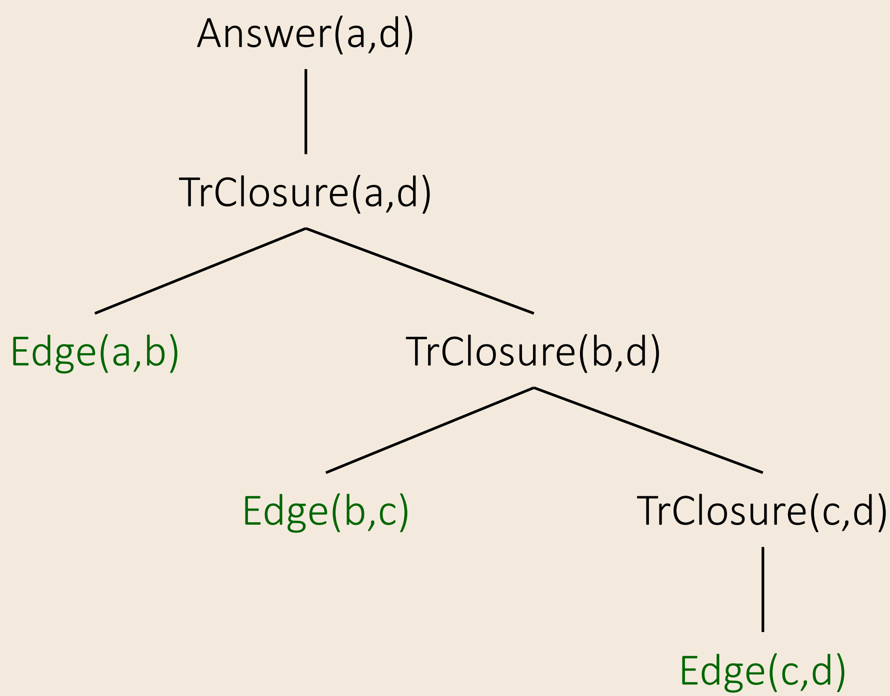
P = {
TrClosure(x,y) :- Edge(x,y)
TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)
Answer(x,y) :- TrClosure(x,y)
}



Proof Tree by Example

D = {Edge(a,b), Edge(b,c), Edge(c,d)}

P = {
TrClosure(x,y) :- Edge(x,y)
TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)
Answer(x,y) :- TrClosure(x,y)
}



Proof-theoretic Semantics of Datalog

- Given a database D and a Datalog query Q = (P, Answer), we first define the output of P on D, denoted P(D), and then collect the content of the relation Answer in P(D)

- We define the notion of proof of a relational atom w.r.t. D and P, and then the output of P on D are all the atoms that can be proved - "proof-theoretic semantics"

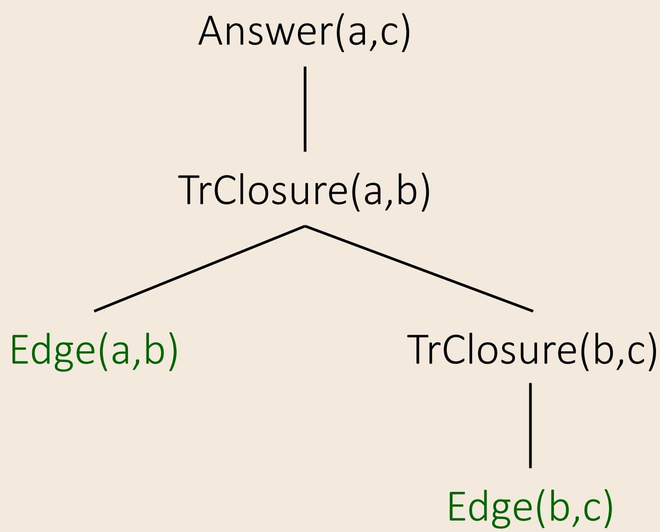
$P(D) = \{R(c_1,\ldots,c_n) : \text{there is a proof tree of } R(c_1,\ldots,c_n) \text{ w.r.t. } D \text{ and } P\}$

for a Datalog query Q = (P, Answer), $Q(D) = \{(c_1,\ldots,c_n) : Answer(c_1,\ldots,c_n) \in P(D)\}$

Explaining Answers to Datalog Queries

D = {Edge(a,b), Edge(b,c), Edge(c,d)}

P = {
TrClosure(x,y) :- Edge(x,y)
TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)
Answer(x,y) :- TrClosure(x,y)
}



Q = (P, Answer)

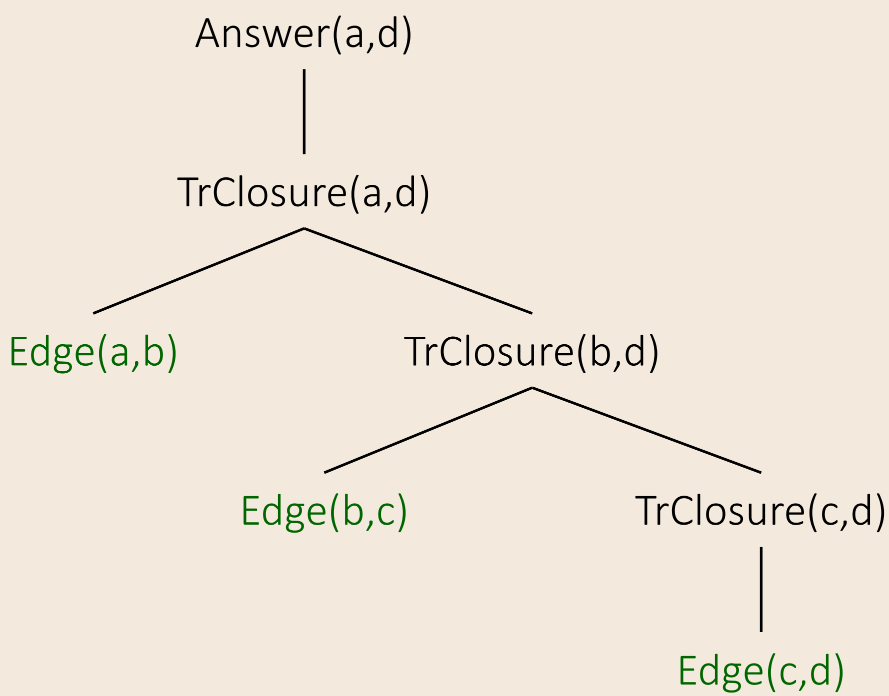
why (a,c) ∈ Q(D)?

{Edge(a,b), Edge(b,c)}

Explaining Answers to Datalog Queries

D = {Edge(a,b), Edge(b,c), Edge(c,d)}

P = {
TrClosure(x,y) :- Edge(x,y)
TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)
Answer(x,y) :- TrClosure(x,y)
}



Q = (P, Answer)

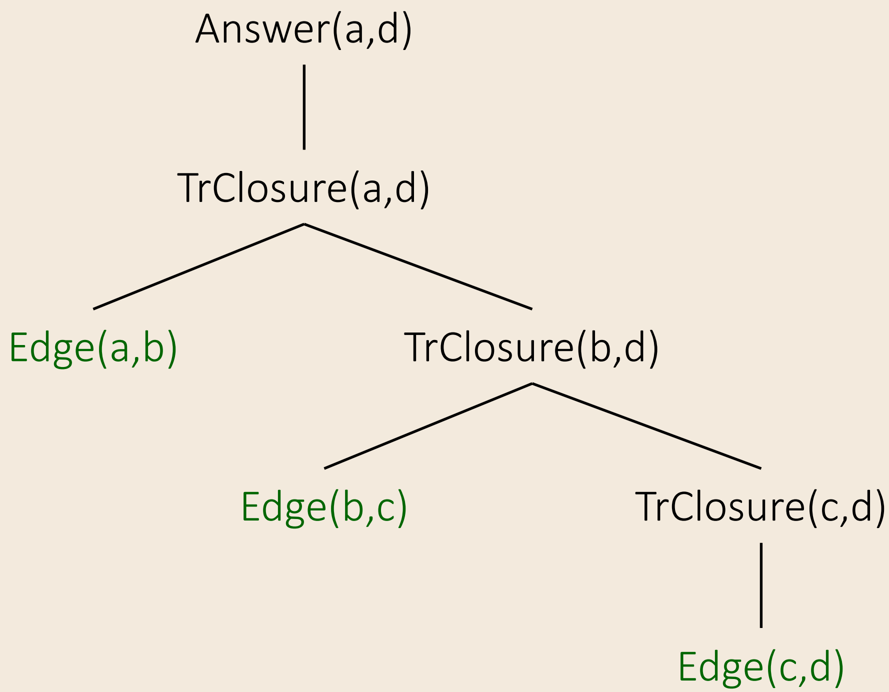
why (a,d) ∈ Q(D)?

{Edge(a,b), Edge(b,c), Edge(c,d)}

Explaining Answers to Datalog Queries

D = {Edge(a,b), Edge(b,c), Edge(c,d),
Edge(a,c)}

P = { TrClosure(x,y) :- Edge(x,y)
TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)
Answer(x,y) :- TrClosure(x,y) }



Q = (P, Answer)

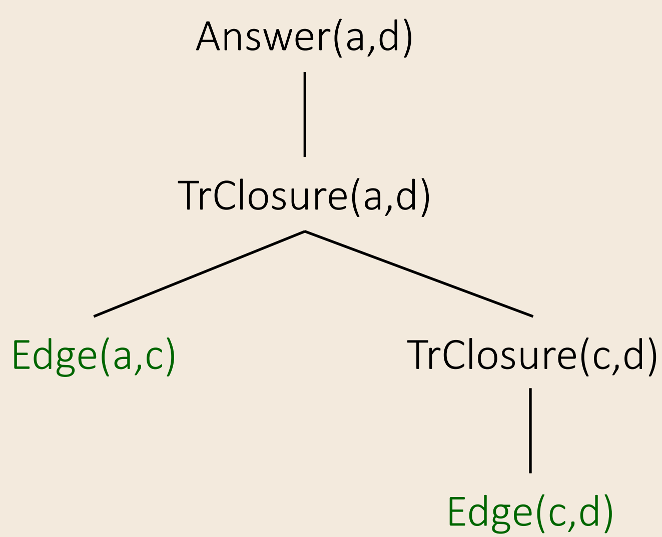
why (a,d) ∈ Q(D)?

{Edge(a,b), Edge(b,c), Edge(c,d)}

Explaining Answers to Datalog Queries

D = {Edge(a,b), Edge(b,c), Edge(c,d), **Edge(a,c)**}

P = {
TrClosure(x,y) :- Edge(x,y)
TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)
Answer(x,y) :- TrClosure(x,y)
}



Q = (P, Answer)

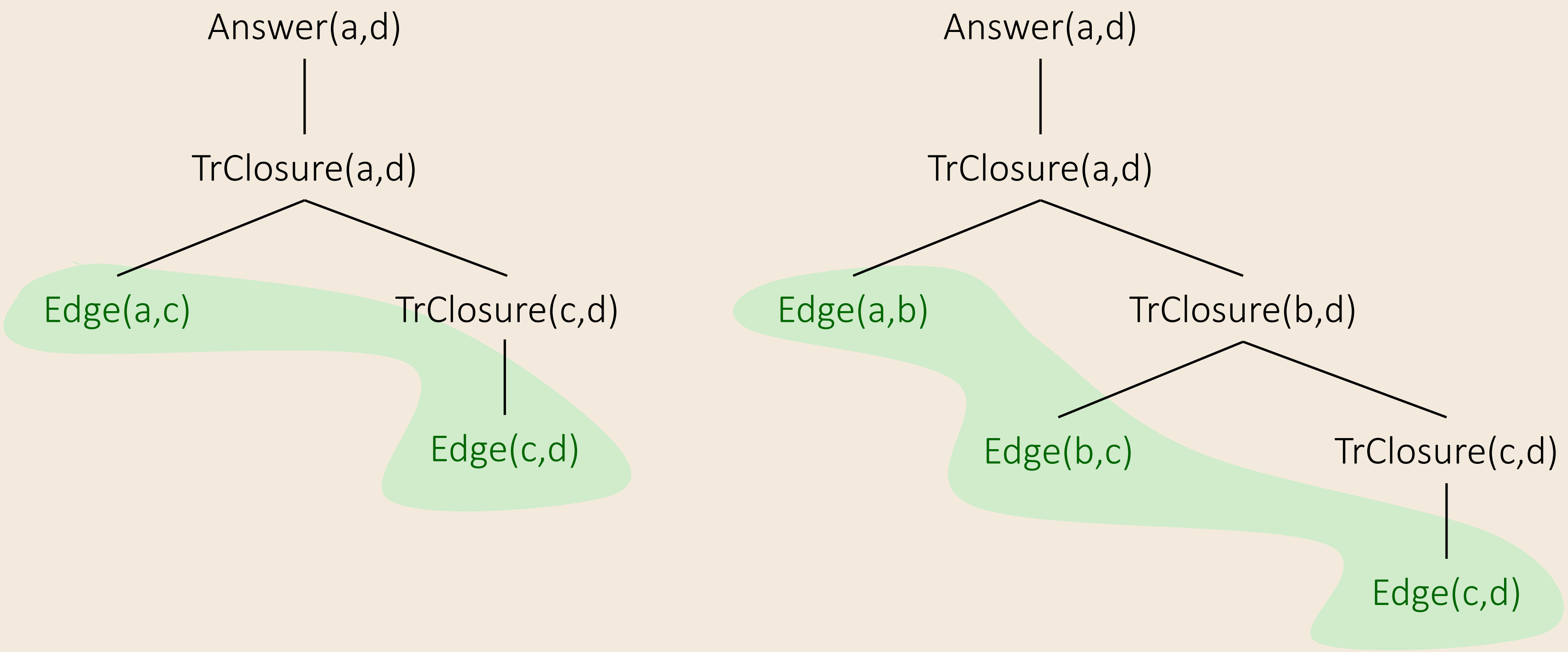
why (a,d) ∈ Q(D)?

{Edge(a,b), Edge(b,c), Edge(c,d)}

{Edge(a,c), Edge(c,d)}

Why-Provenance for Datalog Queries

The support of a proof tree T, denoted support(T), is the set of atoms labelling its leaves



Why-Provenance for Datalog Queries

Given a database D, a Datalog query Q = (P, Answer), and a tuple $(c_1,\ldots,c_n)$,

the why-provenance of $(c_1,\ldots,c_n)$ w.r.t. D and Q is the family of sets of atoms

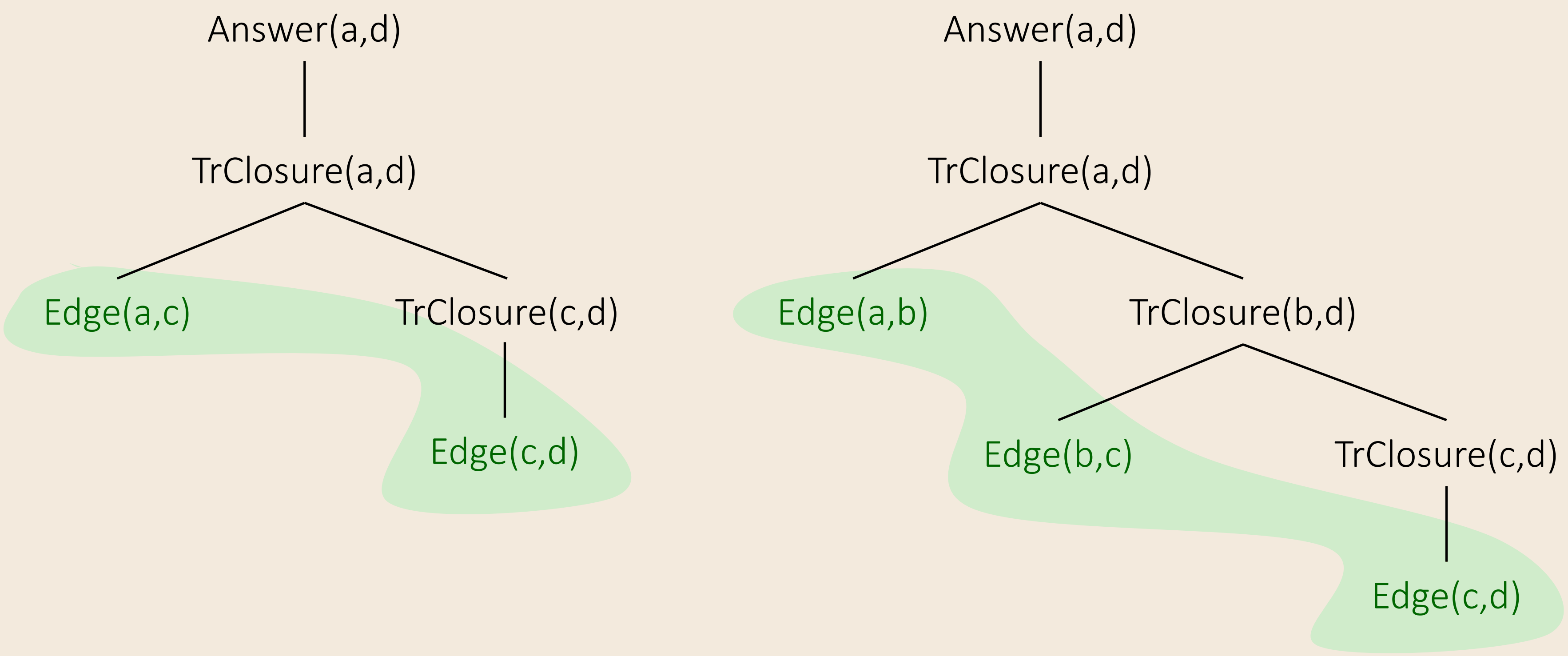
$$\text{why}((c_1,\ldots,c_n),D,Q) = \{\text{support}(T) : T \text{ is a proof tree of } \text{Answer}(c_1,\ldots,c_n) \text{ w.r.t. } D \text{ and } P\}$$

why-provenance can be alternatively defined using the framework of

provenance semirings by adopting the so-called why-provenance semiring

[Green, Karvounarakis, and Tannen, PODS 2007]; [Green, TCS 2011]

Why-Provenance for Datalog Queries



why((a,d),D,Q) = { {Edge(a,c), Edge(c,d)}, {Edge(a,b), Edge(b,c), Edge(c,d)} }

Complexity of Why-Provenance

Why-Provenance

Input: a database D, a Datalog query Q, a tuple $(c_1,\ldots,c_n)$, and $D' \subseteq D$

Question: $D' \in why((c_1,\ldots,c_n),D,Q)$?

Data complexity - D, $(c_1,\ldots,c_n)$, D' are part of the input, Q is fixed

Why-Provenance[Q]

Input: a database D, a tuple $(c_1,\ldots,c_n)$, and $D' \subseteq D$

Question: $D' \in why((c_1,\ldots,c_n),D,Q)$?

Data Complexity of Why-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2024]):

1. For every Datalog query Q, Why-Provenance[Q] is in NP
2. There is a Datalog query Q such that Why-Provenance[Q] is NP-hard

Proof Trees as Witnesses

For $n > 0$, let D_n be the database

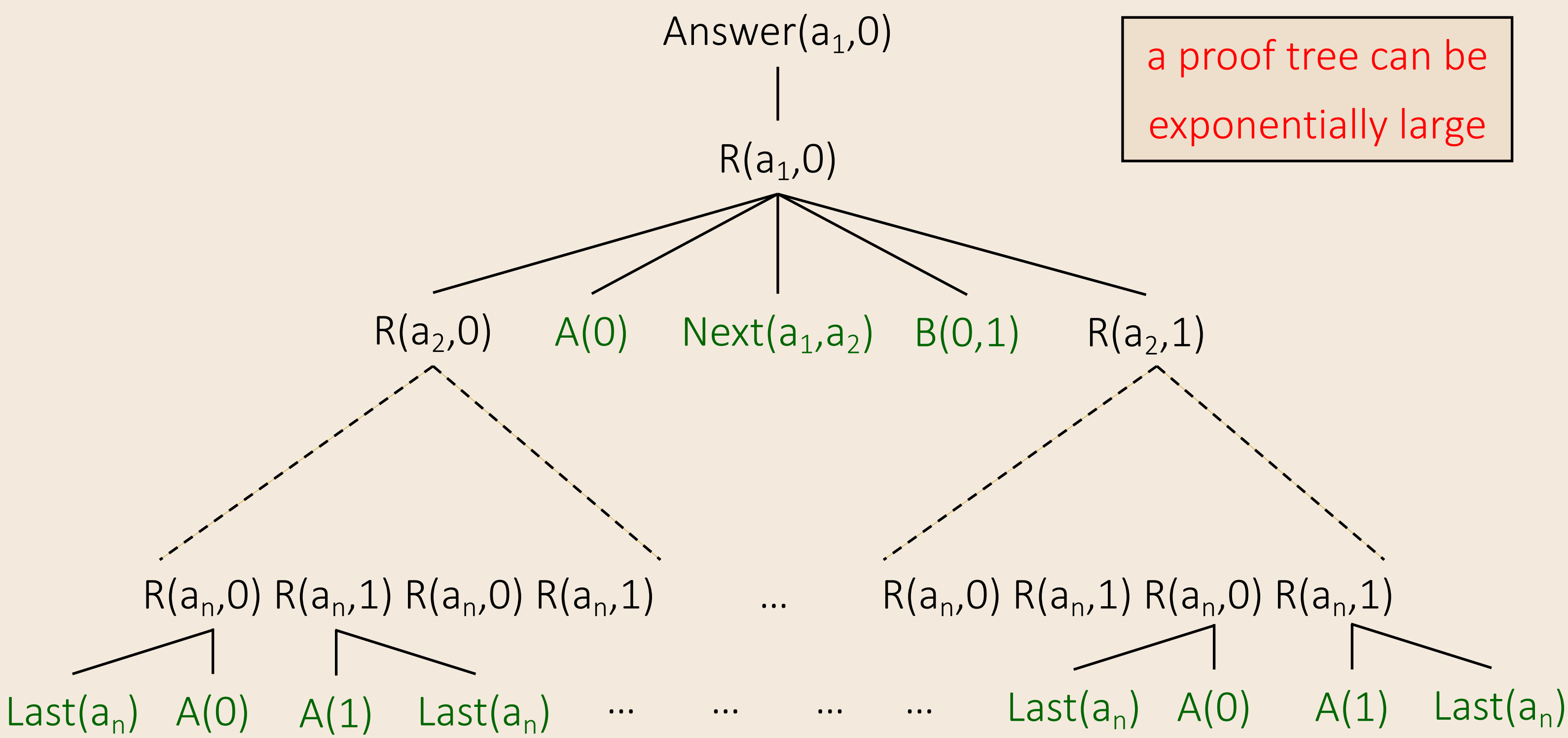
$\{Next(a_1, a_2), \ldots, Next(a_{n-1}, a_n)\}$

$\cup$

$\{A(0), A(1), B(0,1), Last(a_n)\}$

$$P = \left\{ \begin{array}{l} R(x,y) \text{ :- } A(y), Next(x,z), B(w_1,w_2), R(z,w_1), R(z,w_2) \\ R(x,y) \text{ :- } Last(x), A(y) \\ Answer(x,y) \text{ :- } R(x,y) \end{array} \right\}$$

a proof tree can be exponentially large



Data Complexity of Why-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2024]):

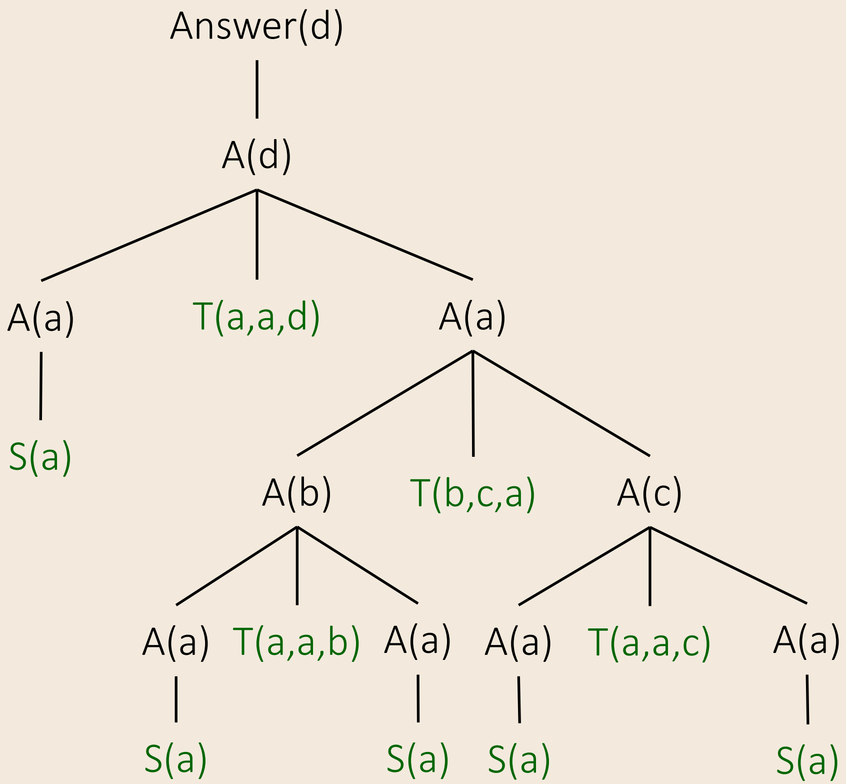
1. For every Datalog query Q, Why-Provenance[Q] is in NP
2. There is a Datalog query Q such that Why-Provenance[Q] is NP-hard

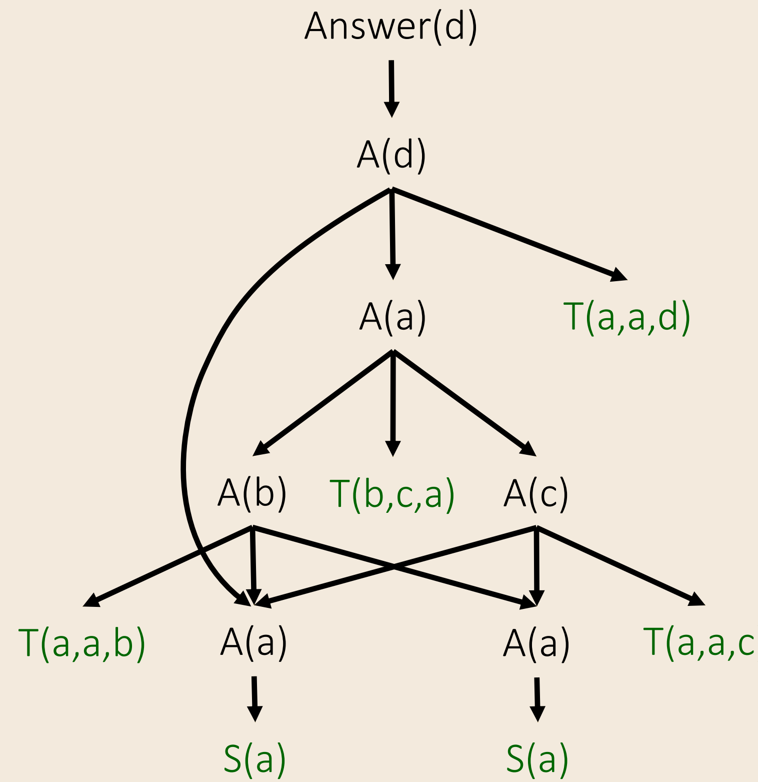
1. Upper bound via a compact representation of proof trees

Proof Directed Acyclic Graph (DAG)

D = {S(a), T(a,a,b), T(a,a,c), T(a,a,d), T(b,c,a)}

$$P = \left\{ \begin{array}{l} A(x) \text{ :- } S(x) \\ A(x) \text{ :- } A(y), A(z), T(y,z,x) \\ Answer(x) \text{ :- } A(x) \end{array} \right\}$$





Compact Representation of Proof Trees

Proposition: For every Datalog program P, there is a polynomial function **f** such that, for every database D, atom $R(c_1,\ldots,c_n)$, and $D' \subseteq D$, the following are equivalent:

1. There is a proof tree T of $R(c_1,\ldots,c_n)$ w.r.t. D and P with support(T) = D'
2. There is a proof DAG G of $R(c_1,\ldots,c_n)$ w.r.t. D and P with support(G) = D' and $|G| \leq \mathbf{f}(|D|)$

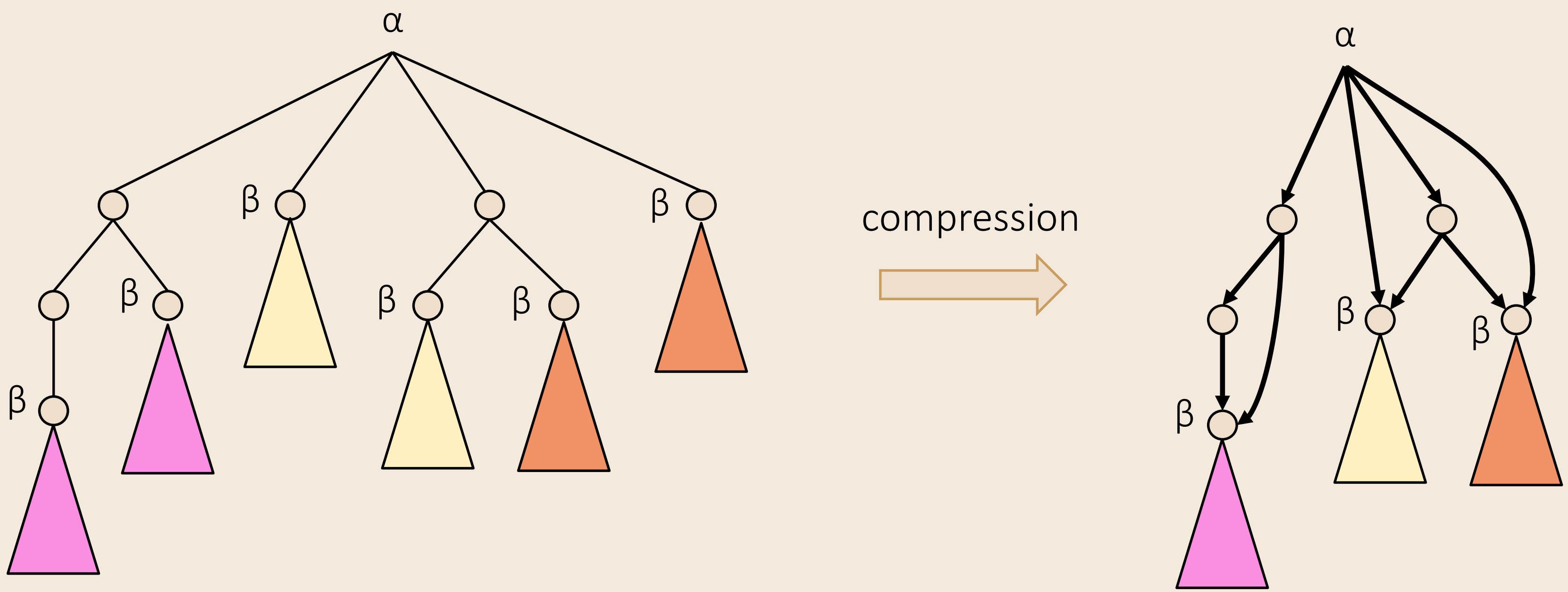
(1) ⇒ (2): The proof consist of three main steps:

1. reduce the depth of the proof tree
2. reduce the subtree count (number of subtrees rooted at nodes with the same label)
3. compression (reuse subtrees by folding the tree into a proof DAG)

Compact Representation of Proof Trees

Proposition: For every Datalog program P, there is a polynomial function **f** such that, for every database D, atom $R(c_1,\dots,c_n)$, and $D' \subseteq D$, the following are equivalent:

1. There is a proof tree T of $R(c_1,\dots,c_n)$ w.r.t. D and P with support(T) = D'
2. There is a proof DAG G of $R(c_1,\dots,c_n)$ w.r.t. D and P with support(G) = D' and $|G| \leq$ **f**$(|D|)$



Compact Representation of Proof Trees

Proposition: For every Datalog program P, there is a polynomial function **f** such that, for every database D, atom $R(c_1,\dots,c_n)$, and $D' \subseteq D$, the following are equivalent:

1. There is a proof tree T of $R(c_1,\dots,c_n)$ w.r.t. D and P with support(T) = D'
2. There is a proof DAG G of $R(c_1,\dots,c_n)$ w.r.t. D and P with support(G) = D' and $|G| \leq \mathbf{f}(|D|)$

(1) ⇒ (2): The proof consist of three main steps:

1. reduce the depth of the proof tree
2. reduce the subtree count (number of subtrees rooted at nodes with the same label)
3. compression (reuse subtrees by folding the tree into a proof DAG)

(2) ⇒ (1): We simply unfold the proof DAG

Data Complexity of Why-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2024]):

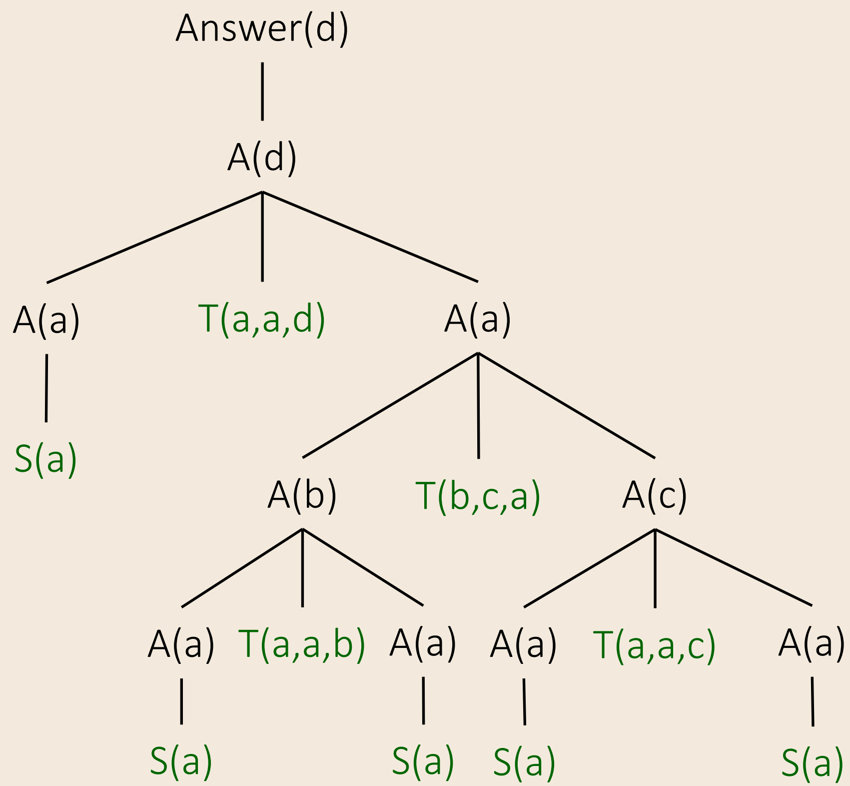
1. For every Datalog query Q, Why-Provenance[Q] is in NP
2. There is a Datalog query Q such that Why-Provenance[Q] is NP-hard

1. Upper bound via a compact representation of proof trees
2. Lower bound via a reduction from 3SAT

Conceptually Problematic Proof Trees

D = {S(a), T(a,a,b), T(a,a,c), T(a,a,d), T(b,c,a)}

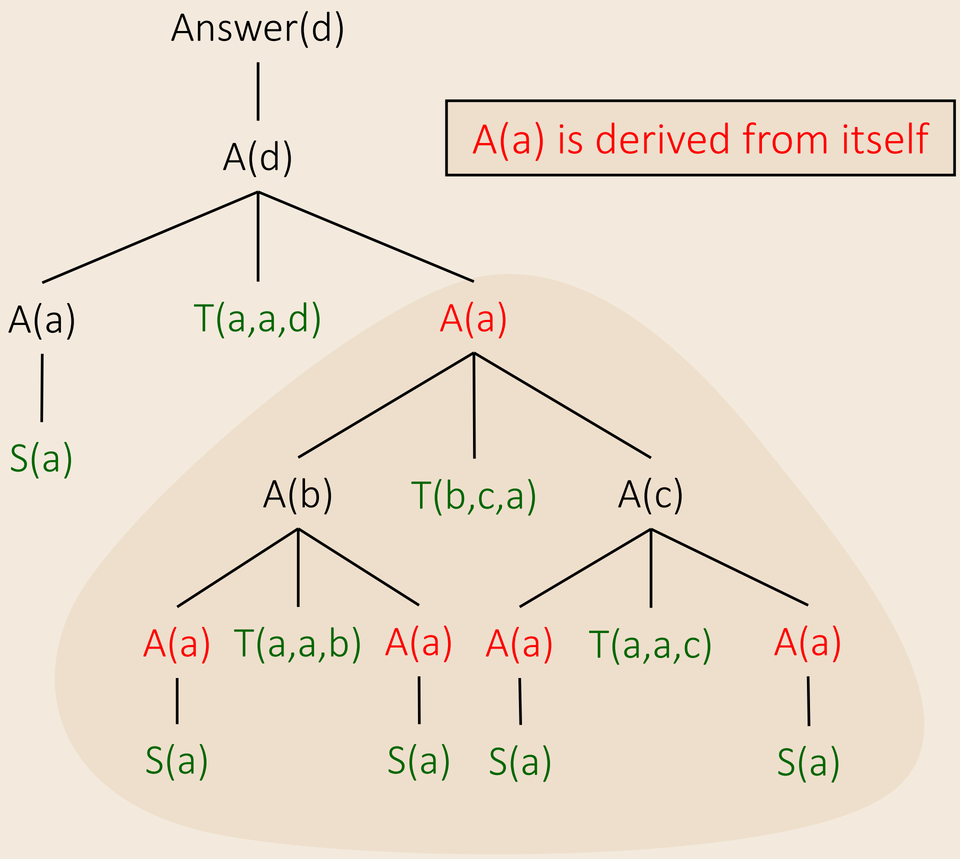
P = {
A(x) :- S(x)
A(x) :- A(y), A(z), T(y,z,x)
Answer(x) :- A(x)
}

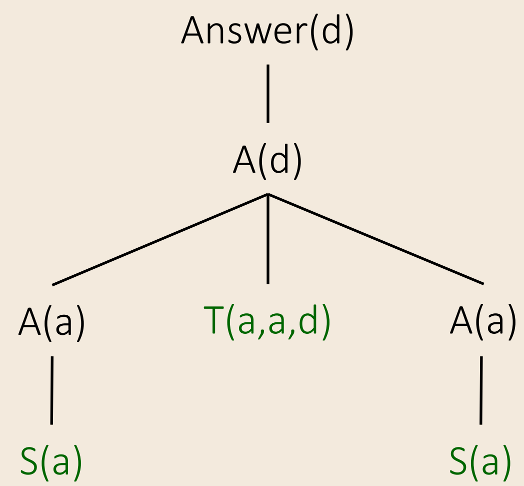


Conceptually Problematic Proof Trees

D = {S(a), T(a,a,b), T(a,a,c), T(a,a,d), T(b,c,a)}

P = { A(x) :- S(x)
A(x) :- A(y), A(z), T(y,z,x)
Answer(x) :- A(x) }





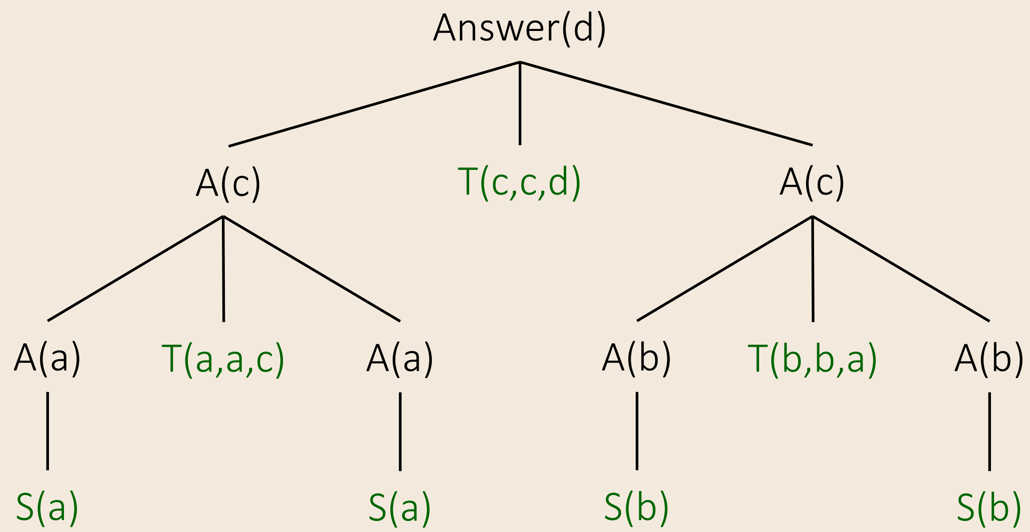
Refined Proof Trees

Non-recursive proof trees - an atom occurs at most once on a path

[Bourgaux, Bourhis, Peterfreund, and Thomazo, KR 2022]

D = {S(a), S(b), T(a,a,c), T(b,b,c), T(c,c,d)}

P = {
A(x) :- S(x)
A(x) :- A(y), A(z), T(y,z,x)
Answer(x) :- A(x)
}



Refined Proof Trees

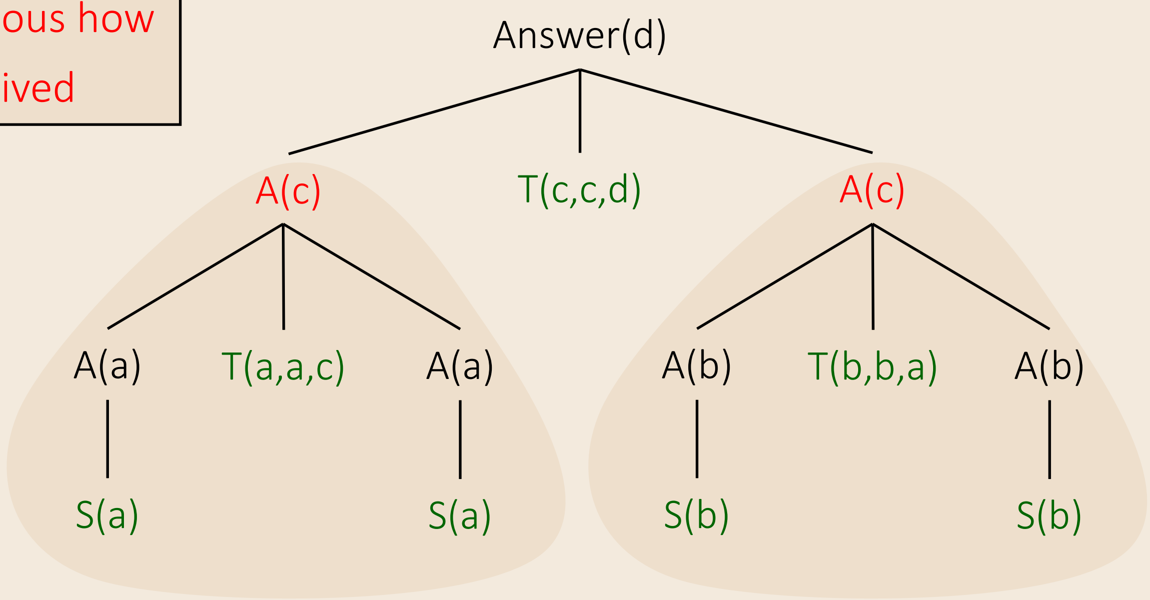
Non-recursive proof trees - an atom occurs at most once on a path

[Bourgaux, Bourhis, Peterfreund, and Thomazo, KR 2022]

D = {S(a), S(b), T(a,a,c), T(b,b,c), T(c,c,d)}

P = { A(x) :- S(x)
A(x) :- A(y), A(z), T(y,z,x)
Answer(x) :- A(x) }

but its ambiguous how A(c) is derived



Refined Proof Trees

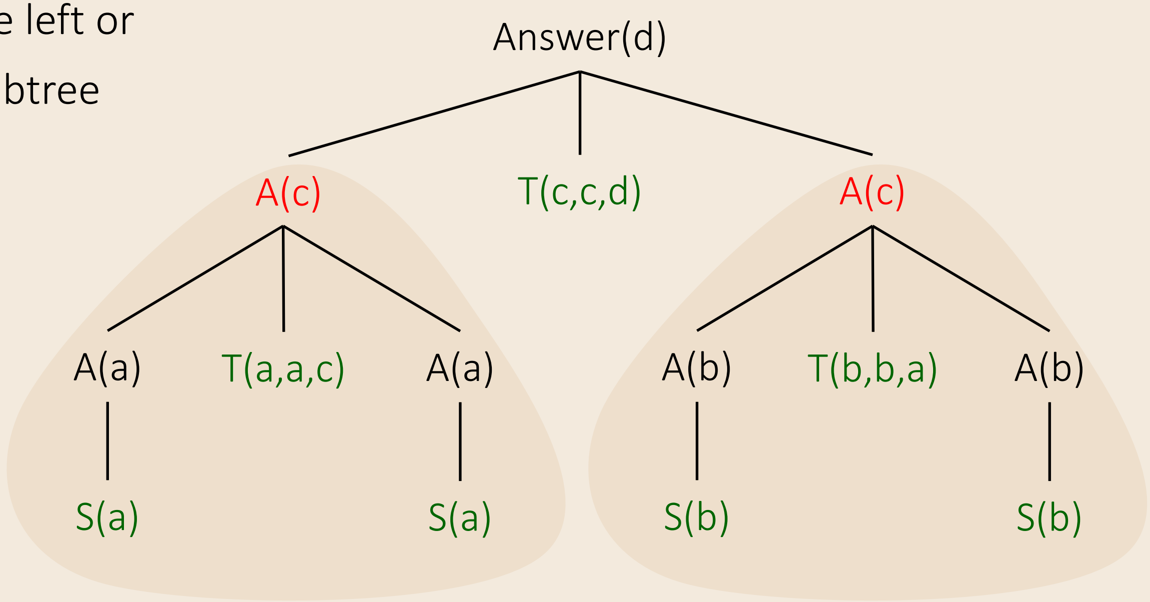
Non-recursive proof trees - an atom occurs at most once on a path

[Bourgaux, Bourhis, Peterfreund, and Thomazo, KR 2022]

D = {S(a), S(b), T(a,a,c), T(b,b,c), T(c,c,d)}

P = { A(x) :- S(x); A(x) :- A(y), A(z), T(y,z,x); Answer(x) :- A(x) }

use either the left or the right subtree



Refined Proof Trees

Non-recursive proof trees - an atom occurs at most once on a path

[Bourgaux, Bourhis, Peterfreund, and Thomazo, KR 2022]

Unambiguous proof trees - each occurrence of an atom has the same subtree

[Calautti, Livshits, P., and Schneider, AAAI 2024]

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2024 & AAAI 2024]**):**

Considering only non-recursive or unambiguous proof trees:

1. For every Datalog query Q, Why-Provenance[Q] is in NP
2. There is a Datalog query Q such that Why-Provenance[Q] is NP-hard

1. Upper bound via a compact representation of proof trees
2. Lower bound via a reduction from Hamiltonian Cycle

Recap

takeaway

- Explaining answers to Datalog queries according to why-provenance is intractable (NP-complete) in data complexity, even if the recursion is linear

- The space of proof trees can be refined without paying a price in complexity

…can we employ SAT solvers for explaining answers to Datalog queries?

Our Target

Given a database D and a Datalog query Q = (P, Answer),

- for a tuple $(c_1,\ldots,c_n)$,
- efficiently enumerate the members of the why-provenance of $(c_1,\ldots,c_n)$ w.r.t. D and Q
- relative to unambiguous proof trees

- **On-demand why-provenance:** instead of computing the why-provenance for all the query answers, focus on a given query answer $(c_1,\ldots,c_n)$ of interest [Elhalawati, Kroetzsch, and Mennicke, RuleML + RR 2022]
- **Incremental computation:** instead of computing the whole why-provenance in one-shot, which is very expensive, provide one explanation at a time (unlike Elhalawati et al.)
- **Conceptually meaningful explanations:** provide only members of the why-provenance supported by a conceptually meaningful derivation process (unlike Elhalawati et al.)

From Why-Provenance to SAT

Proposition(informal) ([Calautti, Livshits, P., and Schneider, AAAI 2024]):

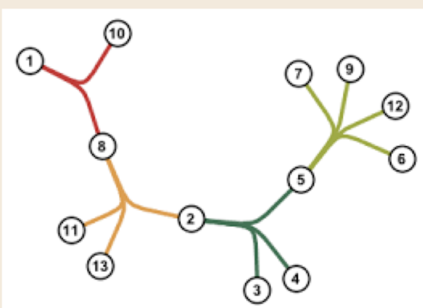
Given a database D, a Datalog query Q = (P, Answer), and a tuple $(c_1,\dots,c_n)$, there exists a Boolean formula ϕ in CNF such that:

1. ϕ can be computed in polynomial time in D and $(c_1,\dots,c_n)$
2. Each member of the why-provenance of $(c_1,\dots,c_n)$ w.r.t. D and Q relative to unambiguous proof trees corresponds to a truth assignment that satisfies ϕ

the construction of ϕ relies on an auxiliary data structure (the **downward closure** of Answer$(c_1,\dots,c_n)$ w.r.t. D and P), that is, a hypergraph that encodes all the proof trees of Answer$(c_1,\dots,c_n)$ w.r.t D and P in their compact representation

Why-Provenance via SAT Solvers

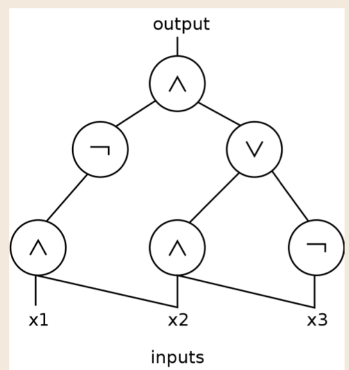
D, Q = (P, Answer), $(c_1,\ldots,c_n)$



downward closure of

Answer$(c_1,\ldots,c_n)$ w.r.t. D and P

add constraint for D'



Boolean formula

SAT Solver (Glucose)

Satisfying assignment

$x_1 = 1, x_2 = 0, \ldots$

Explanation D' of

$(c_1,\ldots,c_n)$ w.r.t. D and Q

Experimental Evaluation

https://gitlab.com/aaai24whyprov/datalog-why-provenance

- Several scenarios from the Datalog literature consisting of a query Q and a family of databases (varying in size) D[Q]
- For each query Q and database D from D[Q], we have selected 100 tuples from Q(D) uniformly at random, and for each tuple, we have incrementally computed its why-provenance w.r.t. D and Q relative to unambiguous proof trees
- Pre-processing: computing the downward closure is the expensive task, whereas the time for building the Boolean formula is negligible
- Enumeration: with the Boolean formula at hand, we can efficiently enumerate the members of the why-provenance relative to unambiguous proof trees - each explanation is produced in milliseconds

Recap

takeaway

- Explaining answers to Datalog queries according to why-provenance is intractable (NP-complete) in data complexity, even if the recursion is linear

- The space of proof trees can be refined without paying a price in complexity

- Encouraging results on using SAT solvers for the incremental computation of why-provenance relative to unambiguous proof trees

rest of the talk

- What about less informative notions of provenance (**whyminimal-provenance**), as well as more informative notions (**whymultiplicity-provenance**)?

WhyMinimal-Provenance for Datalog Queries

Given a database D, a Datalog query Q = (P, Answer), and a tuple $(c_1,\ldots,c_n)$,

the whyminimal-provenance of $(c_1,\ldots,c_n)$ w.r.t. D and Q is the family of sets of atoms

whymin($(c_1,\ldots,c_n)$,D,Q) = {support(T) : T is a minimal proof tree of Answer($c_1,\ldots,c_n$) w.r.t. D and P}

there is no proof tree T' of Answer($c_1,\ldots,c_n$) w.r.t. D and P with support(T') $\subset$ support(T)

whyminimal-provenance can be alternatively defined using the framework of

provenance semirings by adopting the semiring of positive Boolean expressions

[Green, Karvounarakis, and Tannen, PODS 2007]; [Green, TCS 2011]

Complexity of WhyMinimal-Provenance

WhyMinimal-Provenance

Input: a database D, a Datalog query Q, a tuple $(c_1,\ldots,c_n)$, and $D' \subseteq D$

Question: $D' \in whymin((c_1,\ldots,c_n),D,Q)$?

Data complexity - D, $(c_1,\ldots,c_n)$, D' are part of the input, Q is fixed

WhyMinimal-Provenance[Q]

Input: a database D, a tuple $(c_1,\ldots,c_n)$, and $D' \subseteq D$

Question: $D' \in whymin((c_1,\ldots,c_n),D,Q)$?

Data Complexity of WhyMinimal-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2025]):

1. For every Datalog query Q, WhyMinimal-Provenance[Q] is in PTIME
2. There is a Datalog query Q such that WhyMinimal-Provenance[Q] is PTIME-hard

Key Observation: WhyMinimal-Provenance is tightly related to query evaluation

Datalog Query Evaluation

Query-Evaluation

Input: a database D, a Datalog query Q, and a tuple $(c_1,\ldots,c_n)$

Question: $(c_1,\ldots,c_n) \in Q(D)$?

Data complexity - D and $(c_1,\ldots,c_n)$ are part of the input, Q is fixed

Query-Evaluation[Q]

Input: a database D and a tuple $(c_1,\ldots,c_n)$

Question: $(c_1,\ldots,c_n) \in Q(D)$?

Theorem:

1. For every Datalog query Q, Query-Evaluation[Q] is in PTIME
2. There is a Datalog query Q such that Query-Evaluation[Q] is PTIME-hard

WhyMinimal-Provenance vs. Query Evaluation

Proposition: Consider a database D, a Datalog query Q, and a tuple $(c_1,\ldots,c_n)$. For every $D' \subseteq D$, the following are equivalent:

1. $D' \in \text{whymin}((c_1,\ldots,c_n),D,Q)$?
2. $(c_1,\ldots,c_n) \in Q(D')$ and, for every atom $\alpha \in D'$, $(c_1,\ldots,c_n) \notin Q(D' \setminus \{\alpha\})$

Proposition: For every Datalog query Q, there exists a Datalog query Q* such that Query-Evaluation[Q] reduces in logarithmic space to WhyMinimal-Provenance[Q*]

Data Complexity of WhyMinimal-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2025]):

1. For every Datalog query Q, WhyMinimal-Provenance[Q] is in PTIME
2. There is a Datalog query Q such that WhyMinimal-Provenance[Q] is PTIME-hard

1. Upper bound via linearly many calls to query evaluation
2. Lower bound via a reduction from query evaluation

Refined Proof Trees

WhyMinimal-Provenance relative to non-recursive proof trees

= [Bourgaux, Bourhis, Peterfreund, and Thomazo, KR 2022]

WhyMinimal-Provenance

= [Calautti, Livshits, P., and Schneider, PODS 2025]

WhyMinimal-Provenance relative to unambiguous proof trees

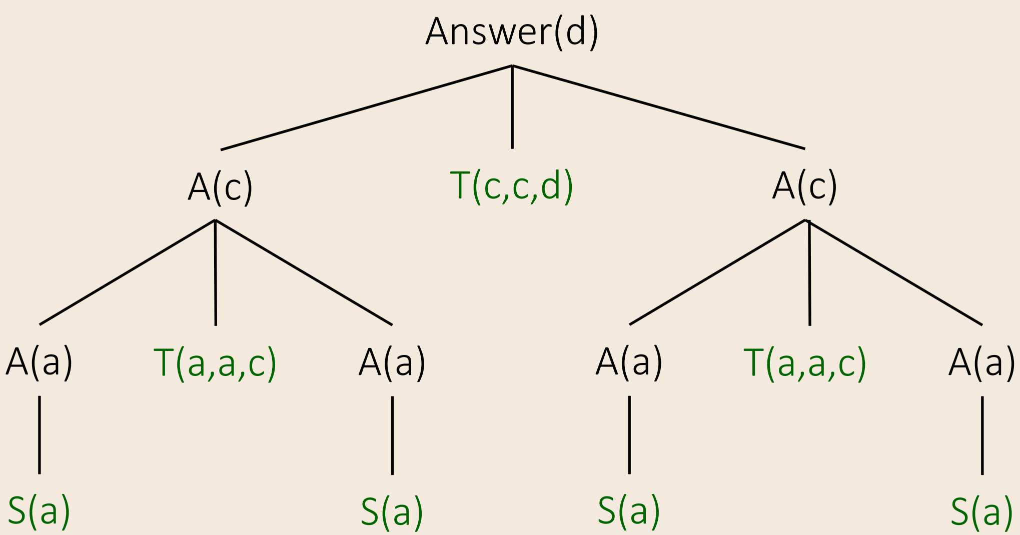
Corollary: Considering only non-recursive or unambiguous proof trees:

1. For every Datalog query Q, WhyMinimal-Provenance[Q] is in PTIME
2. There is a Datalog query Q such that WhyMinimal-Provenance[Q] is PTIME-hard

More Informative Explanations

D = {S(a), S(b), T(a,a,c), T(b,b,c), T(c,c,d)}

P = {
A(x) :- S(x)
A(x) :- A(y), A(z), T(y,z,x)
Answer(x) :- A(x)
}



Q = (P, Answer)

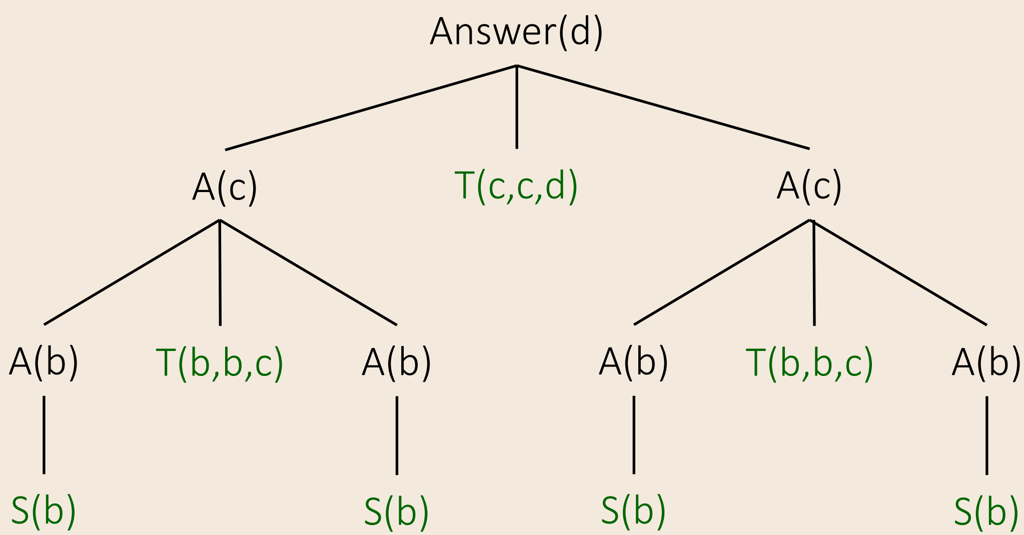
why (d) ∈ Q(D)?

{(S(a),4), (T(a,a,c),2), (T(c,c,d), 1)}

More Informative Explanations

D = {S(a), S(b), T(a,a,c), T(b,b,c), T(c,c,d)}

$$P = \left\{ \begin{array}{l} A(x) \text{ :- } S(x) \\ A(x) \text{ :- } A(y), A(z), T(y,z,x) \\ Answer(x) \text{ :- } A(x) \end{array} \right\}$$



Q = (P, Answer)

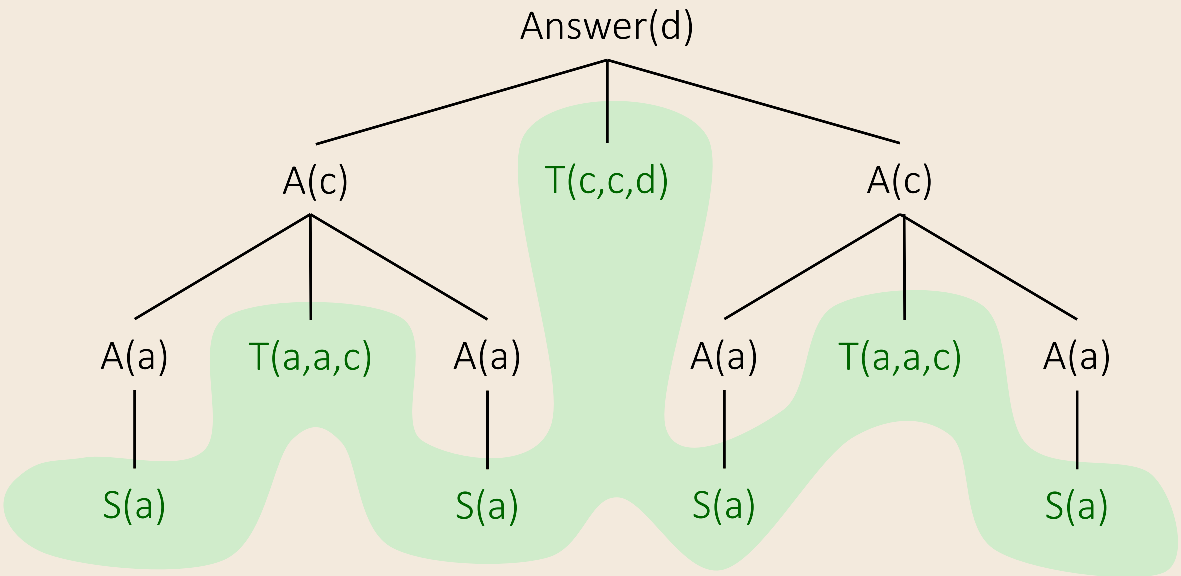
why (d) ∈ Q(D)?

{(S(a),4), (T(a,a,c),2), (T(c,c,d), 1)}

{(S(b),4), (T(b,b,c),2), (T(c,c,d), 1)}

WhyMultiplicity-Provenance for Datalog Queries

The bagsupport of a proof tree T, denoted bagsupport(T), is the bag of atoms labelling its leaves



{(S(a),4), (T(a,a,c),2), (T(c,c,d), 1)}

WhyMultiplicity-Provenance for Datalog Queries

Given a database D, a Datalog query Q = (P, Answer), and a tuple $(c_1,\ldots,c_n)$,

the whymultiplicity-provenance of $(c_1,\ldots,c_n)$ w.r.t. D and Q is the family of bags of atoms

$$\text{whymult}((c_1,\ldots,c_n),D,Q) = \{\text{bagsupport}(T) : T \text{ is a proof tree of Answer}(c_1,\ldots,c_n) \text{ w.r.t. } D \text{ and } P\}$$

whymultiplicity-provenance can be alternatively defined using the framework of provenance semirings by adopting the Boolean provenance polynomial semiring [Green, Karvounarakis, and Tannen, PODS 2007]; [Green, TCS 2011]

Complexity of WhyMultiplicity-Provenance

WhyMultiplicity-Provenance

Input: a database D, a Datalog query Q, a tuple $(c_1,\ldots,c_n)$, and a bag B

with D being the underlying set of B; integers are encoded in binary

Question: $B \in \text{whymult}((c_1,\ldots,c_n),D,Q)$?

Data complexity - D, $(c_1,\ldots,c_n)$, B are part of the input, Q is fixed

WhyMultiplicity-Provenance[Q]

Input: a database D, a tuple $(c_1,\ldots,c_n)$, and a bag B

Question: $B \in \text{whymult}((c_1,\ldots,c_n),D,Q)$?

Data Complexity of WhyMultiplicity-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2025]):

1. For every Datalog query Q, WhyMultiplicity-Provenance[Q] is in NP
2. There is a Datalog query Q such that WhyMultiplicity-Provenance[Q] is NP-hard

1. Upper bound via a hypergraph-theoretic approach

Hypergraph-theoretic Approach

Proposition (informal): Consider a database D, a Datalog query Q = (P, Answer), a tuple $(c_1,\ldots,c_n)$, and a bag B with D being the underlying set. The following are equivalent:

1. There is a proof tree T of Answer$(c_1,\ldots,c_n)$ w.r.t. D and P with bagsupport(T) = B
2. There exists a certain hyperpath in a directed hypergraph obtained from D and P

- The above proposition leads to an easy guess-and-check algorithm that runs in polynomial time in the combined size of D, $(c_1,\ldots,c_n)$, and B
- To show that the "check" step of the above algorithm can be performed in polynomial time, we had to show that the existence of an **Euler hyperpath** from a source node to a target node in a directed hypergraph can be checked in polynomial time
- The latter is shown by characterizing the existence of such an Euler hyperpath via some simple syntactic conditions that can be verified in polynomial time

Data Complexity of WhyMultiplicity-Provenance

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2025]):

1. For every Datalog query Q, WhyMultiplicity-Provenance[Q] is in NP
2. There is a Datalog query Q such that WhyMultiplicity-Provenance[Q] is NP-hard

1. Upper bound via a hypergraph-theoretic approach
2. Lower bound via a reduction from 3SAT

Non-Recursive Proof Trees

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2025]):

Considering only non-recursive proof trees:

1. For every Datalog query Q, WhyMultiplicity-Provenance[Q] is in PSPACE
2. There is a Datalog query Q such that WhyMultiplicity-Provenance[Q] is PSPACE-hard

1. Upper bound via a recursive algorithm that non-deterministically constructs a proof tree T with the right bagsupport in a depth-first fashion
2. Lower bound via a reduction from Q3SAT

Unambiguous Proof Trees

Theorem ([Calautti, Livshits, P., and Schneider, PODS 2025]):

Considering only unambiguous proof trees:

1. For every Datalog query Q, WhyMultiplicity-Provenance[Q] is in NP
2. There is a Datalog query Q such that WhyMultiplicity-Provenance[Q] is NP-hard

1. Upper bound via a compact representation of proof trees
2. Lower bound via a reduction from Hamiltonian Cycle

Summing Up

	Arbitrary	Non-Recursive	Unambiguous
WhyMultiplicity	NP	PSPACE	NP
Why	NP		
WhyMinimal	PTIME		

	Arbitrary	Non-Recursive	Unambiguous
WhyMultiplicity	NP		
Why			
WhyMinimal	NLOGSPACE		

Linear recursion: at most one intensional relation in rule-bodies

Summing Up

	Arbitrary	Non-Recursive	Unambiguous
WhyMultiplicity	NP	PSPACE	NP
Why	NP		
WhyMinimal	PTIME		

- Encouraging results on using SAT solvers for the incremental computation of why-provenance relative to unambiguous proof trees

- WhyMinimal-Provenance is tightly related to query evaluation → efficient Datalog engines can be used

Future Research

- Further development of the SAT-based approach for the incremental computation of explanations (downward closure, whymultiplicity-provenance)
- Study other interesting notions of provenance (Why and WhyMultiplicity + frequency)
- Explain answers to rule-based ontology-mediated queries (recursive queries with the additional feature of value invention)

Thank You!