Short and Simple Explanations for Datalog Derivations

Jelle Hellings *Yuchen You*

Department of Computing and Software
McMaster University

March 24, 2026

Understanding query results

Simple queries: Project-Select-Join

$$q := \pi_{columns}(\sigma_{conditions}(R_1 \times \cdots \times R_n))$$

To prove $\tau \in$ eval(q): find tuples $\tau_1 \in R_1, \ldots, \tau_n \in R_n$ that satisfy *conditions*:

- Easy to explain results: n facts!
- Easy to compute as side-effect of evaluation.

Understanding query results

Simple queries: Project-Select-Join

$$q := \pi_{columns}(\sigma_{conditions}(R_1 \times \cdots \times R_n))$$

To prove $\tau \in$ eval(q): find tuples $\tau_1 \in R_1, \ldots, \tau_n \in R_n$ that satisfy *conditions*:

- Easy to explain results: n facts!
- Easy to compute as side-effect of evaluation.

Recursive Queries: Datalog

Due to recursion, hard to understand how a tuple is derived:

- Explanation requires unknown number of facts.
- Many (possibly repeated) derivation steps.

Need for explaining recursive queries

Project-Select-Join queries only a fraction of real-world data-based computations.

Applications of Recursion

- Graph querying & reasoning over graphs.
- Automated reasoning (e.g., Prolog).
- Formal languages (e.g., context-free grammars)
- Data integration.
- ....

Explanations provide insight in queries (e.g., debugging) and in data (why?).

Goal of this work

Goal: Gain a basic understaning of how recursion (Datalog) can be explained.

Goal of this work

Goal: Gain a basic understaning of how recursion (Datalog) can be explained.

Refined Goal: Understand easy-to-use explanations.

Goal of this work

Goal: Gain a basic understaning of how recursion (Datalog) can be explained.

Refined Goal: Understand easy-to-use explanations.

Related Work: Why-provenance

Question Characterizing the set S of all possible explanations of a fact,
study properties of set S.

This work Focus on the properties of a *simple* member of S.

Datalog: Explaining a derivation

Definition

A Datalog program is a pair $\langle B, D \rangle$ with:

- B a set of Datalog rules defining *base facts*;
- D is a set of Datalog rules defining *derived facts*.

Datalog: Explaining a derivation

Definition

A Datalog program is a pair $\langle B, D \rangle$ with:

- B a set of Datalog rules defining *base facts*;
- D is a set of Datalog rules defining *derived facts*.

Definition

The *derivation tree* for fact $R(c_1, \ldots, c_n)$ is defined recursively as follows:

R is a base fact "$R(c_1, \ldots, c_n) \leftarrow \bullet$" $\in B$:
represent via a *leaf node*.

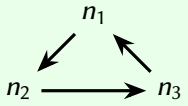
R is a derived fact derived using

$$\text{“}R(v_1, \ldots, v_n) \leftarrow S_1(w_{1,1}, \ldots, w_{1,m_1}) \wedge \cdots \wedge S_j(w_{j,1}, \ldots, w_{j,m_j}) \in D :$$

repesent via an *internal node* with children for facts $S_1, \ldots S_j$.

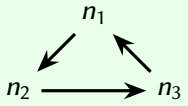
Datalog explanation via Derivation Trees: Example

$$\mathrm{TC}(v, w) \leftarrow \mathrm{Edge}(v, w);$$
$$\mathrm{TC}(v, w) \leftarrow \mathrm{TC}(v, u) \wedge \mathrm{TC}(u, w)$$



Datalog explanation via Derivation Trees: Example

$$\mathrm{TC}(v, w) \leftarrow \mathrm{Edge}(v, w);$$
$$\mathrm{TC}(v, w) \leftarrow \mathrm{TC}(v, u) \wedge \mathrm{TC}(u, w)$$

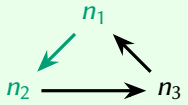


Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$

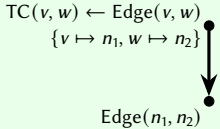
Datalog explanation via Derivation Trees: Example

$\mathrm{TC}(v, w) \leftarrow \mathrm{Edge}(v, w);$

$\mathrm{TC}(v, w) \leftarrow \mathrm{TC}(v, u) \wedge \mathrm{TC}(u, w)$



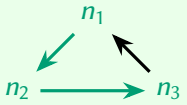
Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



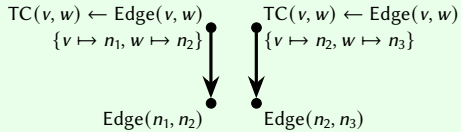
Datalog explanation via Derivation Trees: Example

$\mathrm{TC}(v, w) \leftarrow \mathrm{Edge}(v, w);$

$\mathrm{TC}(v, w) \leftarrow \mathrm{TC}(v, u) \wedge \mathrm{TC}(u, w)$



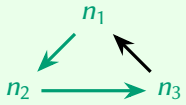
Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



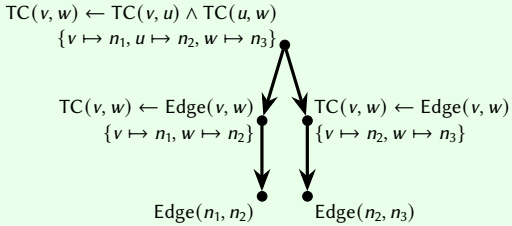
Datalog explanation via Derivation Trees: Example

$\mathrm{TC}(v, w) \leftarrow \mathrm{Edge}(v, w);$

$\mathrm{TC}(v, w) \leftarrow \mathrm{TC}(v, u) \wedge \mathrm{TC}(u, w)$

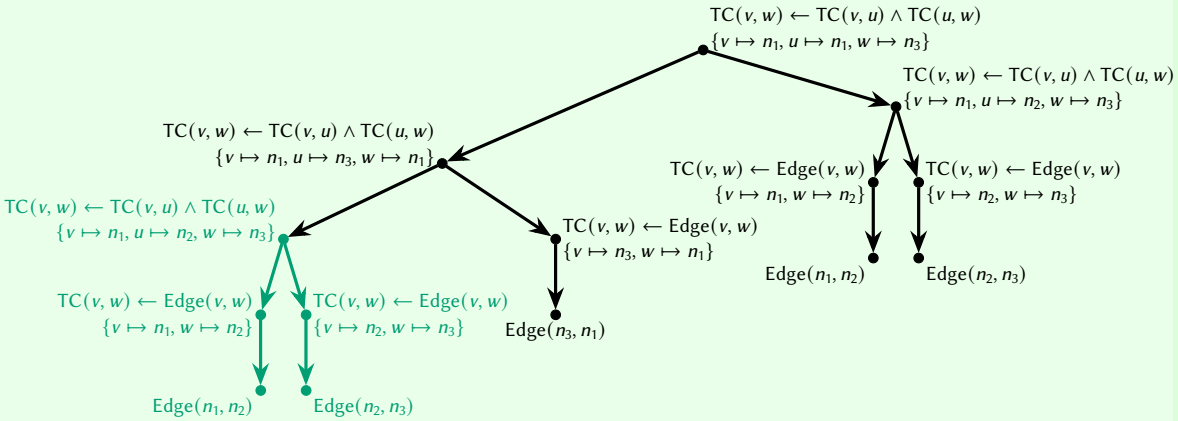


Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



Datalog explanation via Derivation Trees: Example

Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3 n_1 n_2 n_3$



Multiple widely-different derivations can exist for the same fact!

Short and simple Datalog explanations

Refined Goal: Understand easy-to-use explanations.

Decision: Use *derivation trees* to represent explanations
Natural fit for Datalog.

Short and simple Datalog explanations

Refined Goal: Understand easy-to-use explanations.

Decision: Use *derivation trees* to represent explanations
Natural fit for Datalog.

- ✖ Many derivation trees for a single fact.
- ✖ Derivation trees have no size bound.

Short and simple Datalog explanations

Refined Goal: Understand easy-to-use explanations.

Decision: Use *derivation trees* to represent explanations
Natural fit for Datalog.

- ✖ Many derivation trees for a single fact.
- ✖ Derivation trees have no size bound.

For easy-to-use explanations: choose *short and simple* derivation trees!

Research Problem: Characterize easy-to-use explanations.

Short and simple Datalog explanations

Research Problem: Characterize easy-to-use explanations.

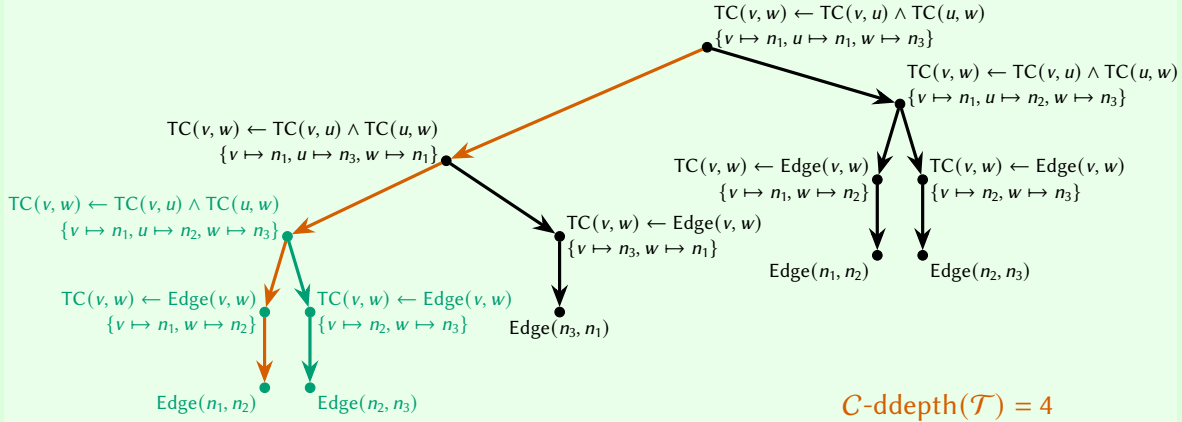
Cost measures for *derivation tree* $\mathcal{T}$

C-ddepth($\mathcal{T}$)	Depth of the tree?
C-deriv($\mathcal{T}$)	Size of the tree (internal nodes)?
C-base($\mathcal{T}$)	How many distinct base facts (leaves) are used?
C-uderiv($\mathcal{T}$)	How many distinct derived facts (internal nodes) are used?

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-ddepth($\mathcal{T}$) Depth of the tree?



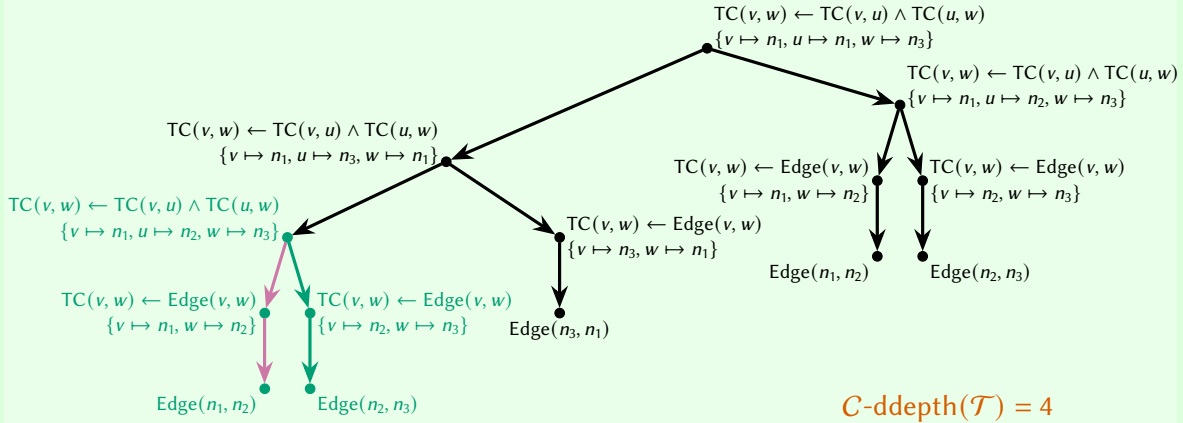
$\mathcal{C}$-ddepth($\mathcal{T}$) = 4

$\mathcal{C}$-ddepth($\mathcal{T}'$) = 2

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-ddepth($\mathcal{T}$) Depth of the tree?



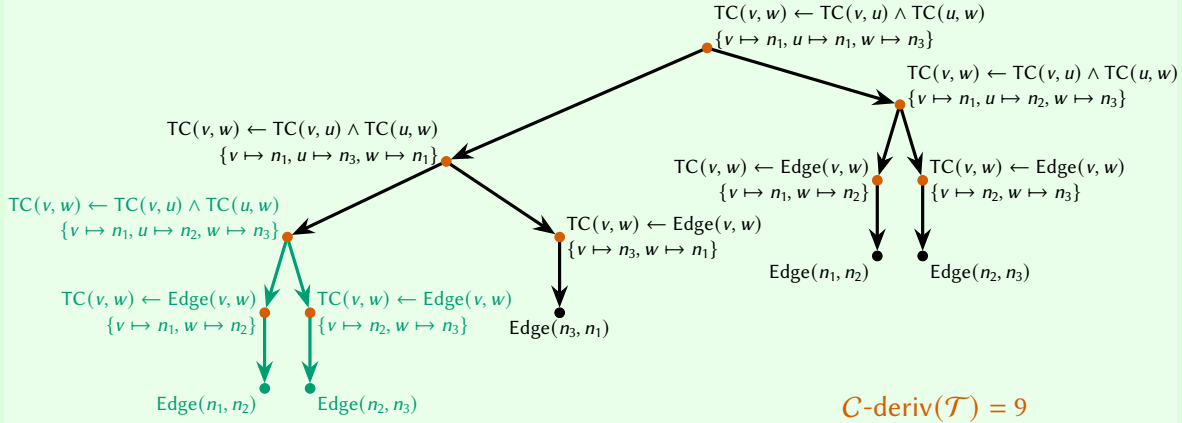
$\mathcal{C}$-ddepth($\mathcal{T}$) = 4

$\mathcal{C}$-ddepth($\mathcal{T}'$) = 2

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-deriv($\mathcal{T}$) Size of the tree (internal nodes)?



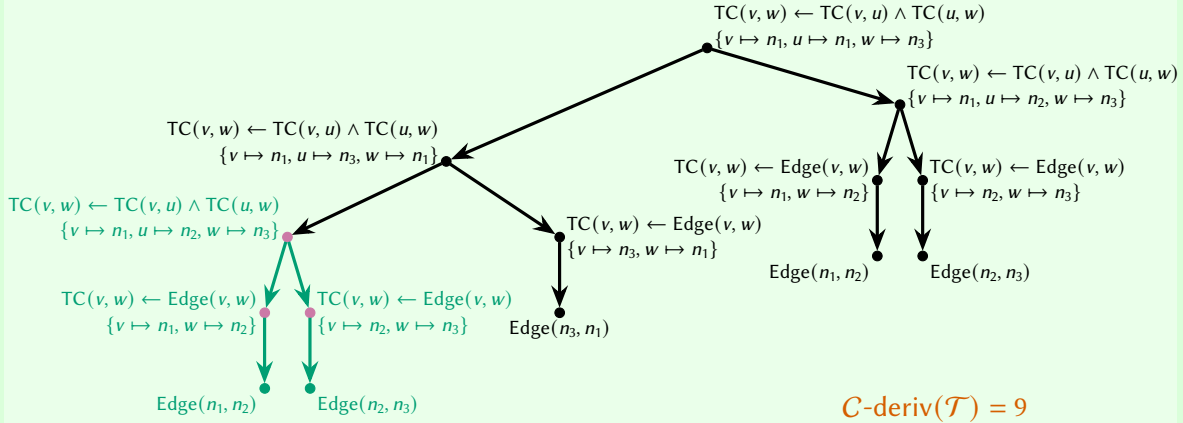
$\mathcal{C}$-deriv($\mathcal{T}$) = 9

$\mathcal{C}$-deriv($\mathcal{T}'$) = 3

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-deriv($\mathcal{T}$) Size of the tree (internal nodes)?



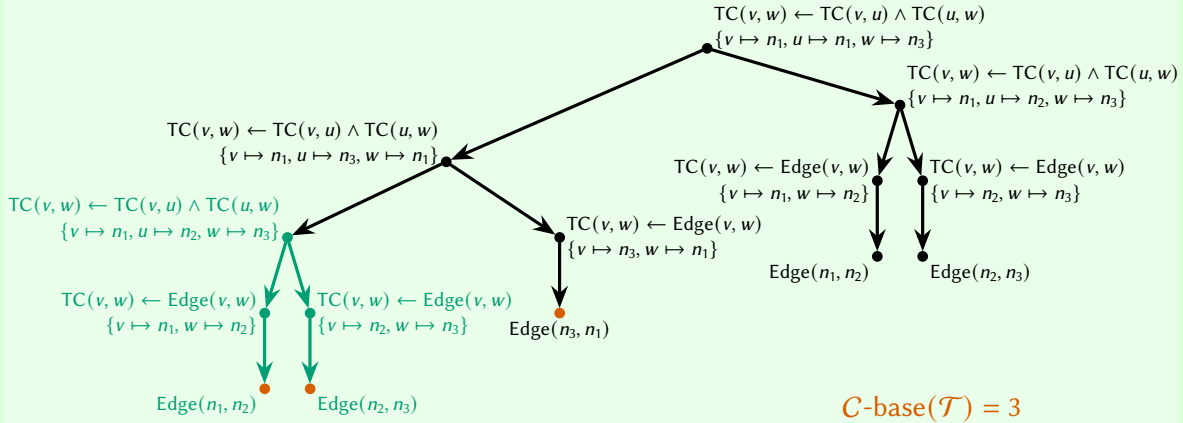
$\mathcal{C}$-deriv($\mathcal{T}$) = 9

$\mathcal{C}$-deriv($\mathcal{T}'$) = 3

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-base($\mathcal{T}$) How many distinct base facts (leaves) are used?



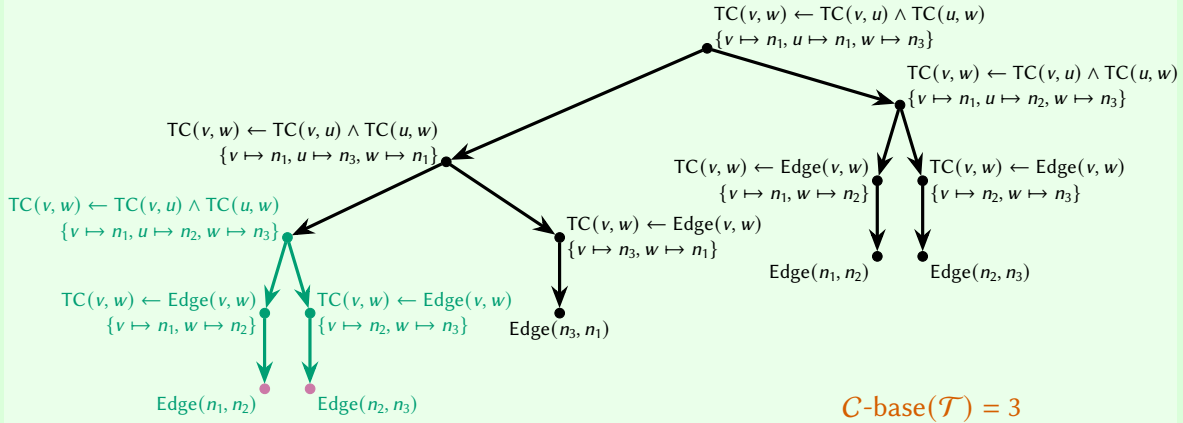
$\mathcal{C}$-base($\mathcal{T}$) = 3

$\mathcal{C}$-base($\mathcal{T}'$) = 2

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-base($\mathcal{T}$) How many distinct base facts (leaves) are used?



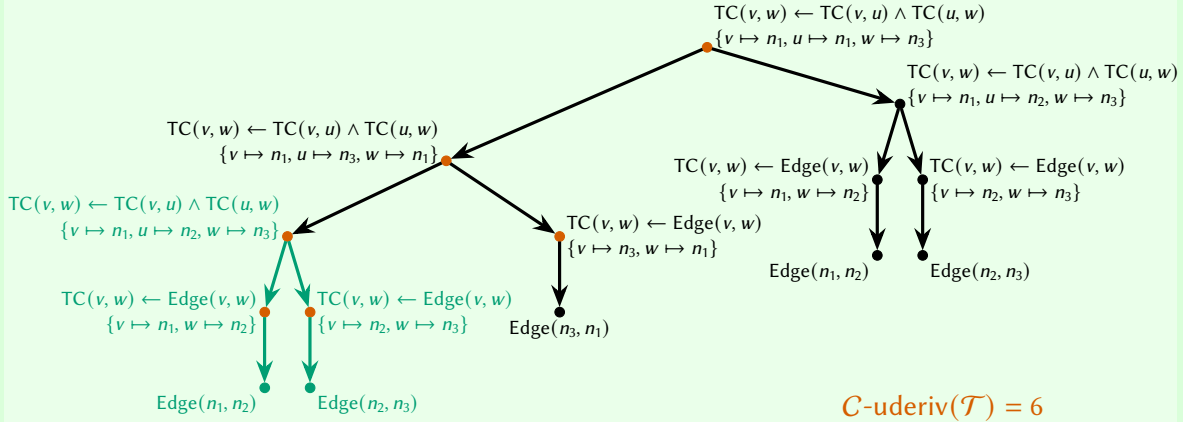
$\mathcal{C}$-base($\mathcal{T}$) = 3

$\mathcal{C}$-base($\mathcal{T}'$) = 2

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-uderiv($\mathcal{T}$) How many distinct derived facts (internal nodes) are used?



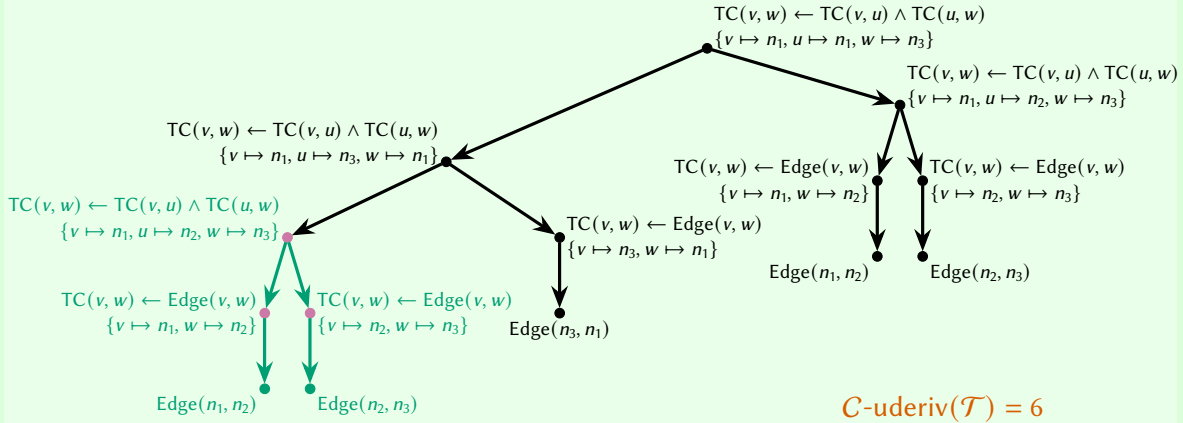
$\mathcal{C}$-uderiv($\mathcal{T}$) = 6

$\mathcal{C}$-uderiv($\mathcal{T}'$) = 3

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

$\mathcal{C}$-uderiv($\mathcal{T}$) How many distinct derived facts (internal nodes) are used?



$\mathcal{C}$-uderiv($\mathcal{T}$) = 6

$\mathcal{C}$-uderiv($\mathcal{T}'$) = 3

Short and simple Datalog explanations

Cost measures for *derivation tree* $\mathcal{T}$

C-ddepth($\mathcal{T}$) Depth of the tree?

C-deriv($\mathcal{T}$) Size of the tree (internal nodes)?

C-base($\mathcal{T}$) How many distinct base facts (leaves) are used?

C-uderiv($\mathcal{T}$) How many distinct derived facts (internal nodes) are used?

Related Work

Some works already cover some of these complexities, e.g.,

- Hellings (2025) looked at C-ddepth for context-free graph queries.
- Calautti et al. (2023) looked at some aspects of C-base derivations.

Comparing the complexity measures

Refined Goal: Understand easy-to-use explanations.

Theorem

Let $C_1, C_2 \in \{C\text{-base}, C\text{-deriv}, C\text{-uderiv}, C\text{-ddepth}\}$, $C_1 \neq C_2$, be two distinct cost measures.

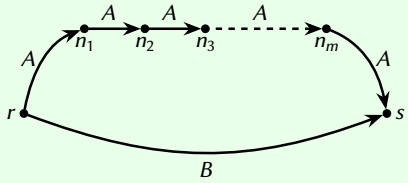
There exist a datalog program such that any derivation of fact $R(t)$ with minimal C_1 complexity does not have minimal C_2 complexity.

Proof (sketch).

Define base facts (graphs) and derivation rules such that a fact $R(t)$ has *exactly two distinct derivation trees* $\mathcal{T}$ and $\mathcal{T}'$ that differ in complexity C_1 and C_2. □

Comparing the complexity measures: Case $C_1 = C$-base

Base facts

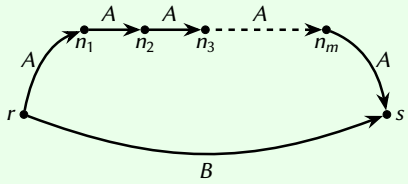


Derivation rules

$$\begin{aligned}
\text{Path-}A(v,w) &\leftarrow A(v,w),\\
\text{Path-}A(v,w) &\leftarrow A(v,u) \wedge \text{Path-}A(u,w)\\
\text{Path-}A'(v,w) &\leftarrow A(v,w),\\
\text{Path-}A'(v,w) &\leftarrow A(v,u) \wedge \text{Path-}A'(u,w);\\
\text{Path-}A''(v,w) &\leftarrow \text{Path-}A'(v,w);\\
\text{Ans}(v,w) &\leftarrow \text{Path-}A(v,w) \wedge \text{Path-}A''(v,w),\\
\text{Ans}(v,w) &\leftarrow \text{Path-}A(v,w) \wedge B(v,w).
\end{aligned}$$

Comparing the complexity measures: Case $C_1 = C$-base

Base facts



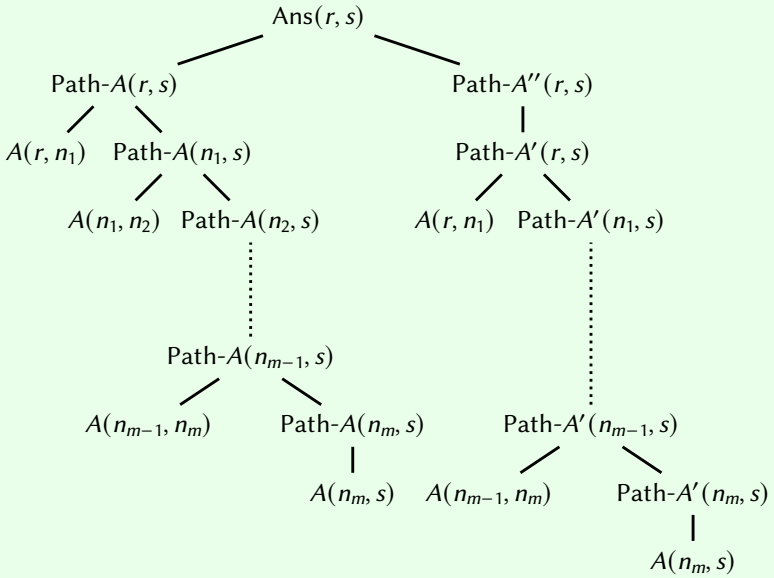
Derivation rules

$$\begin{aligned}
\text{Path-}A(v,w) &\leftarrow A(v,w),\\
\text{Path-}A(v,w) &\leftarrow A(v,u) \wedge \text{Path-}A(u,w)\\
\text{Path-}A'(v,w) &\leftarrow A(v,w),\\
\text{Path-}A'(v,w) &\leftarrow A(v,u) \wedge \text{Path-}A'(u,w);\\
\text{Path-}A''(v,w) &\leftarrow \text{Path-}A'(v,w);\\
\text{Ans}(v,w) &\leftarrow \text{Path-}A(v,w) \wedge \text{Path-}A''(v,w),\\
\text{Ans}(v,w) &\leftarrow \text{Path-}A(v,w) \wedge B(v,w).
\end{aligned}$$

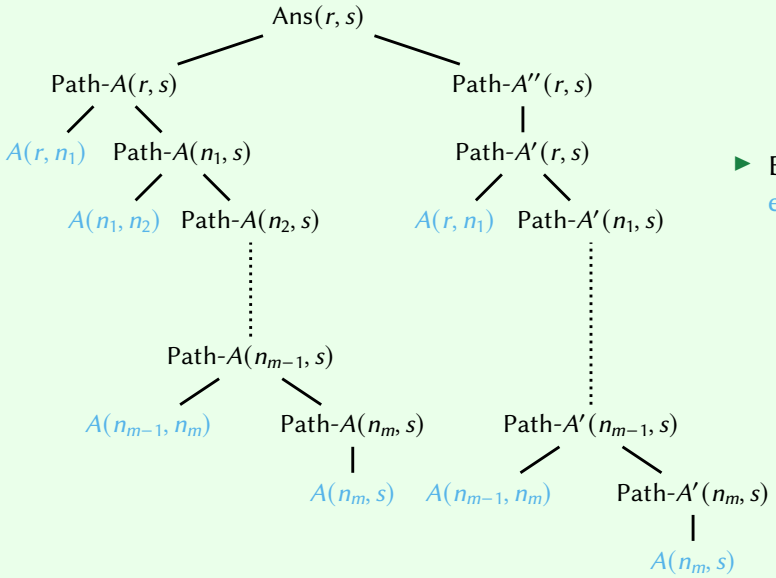
Explain fact $\text{Ans}(r,s)$

- Tree $\mathcal{T}$ explains via $\text{Ans}(v,w) \leftarrow \text{Path-}A(v,w) \wedge \text{Path-}A''(v,w)$.
- Tree $\mathcal{T}'$ explains via $\text{Path-}A(v,w) \wedge B(v,w)$.

Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$

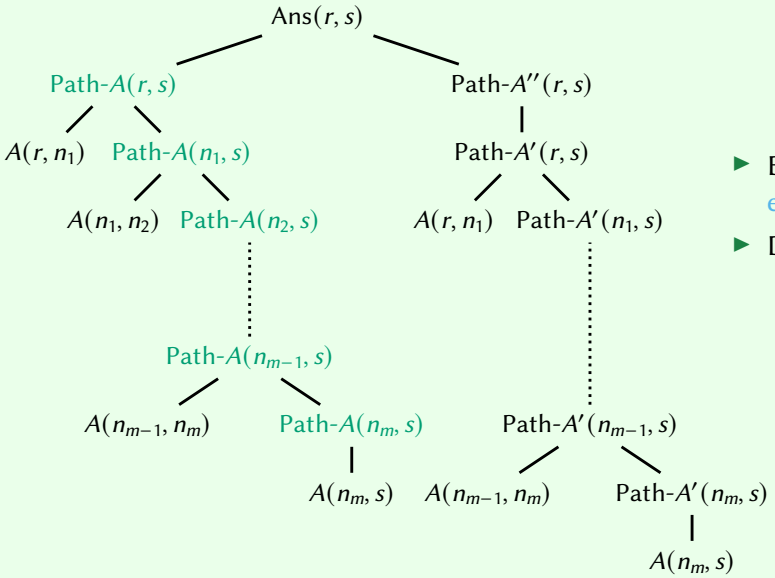


Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$



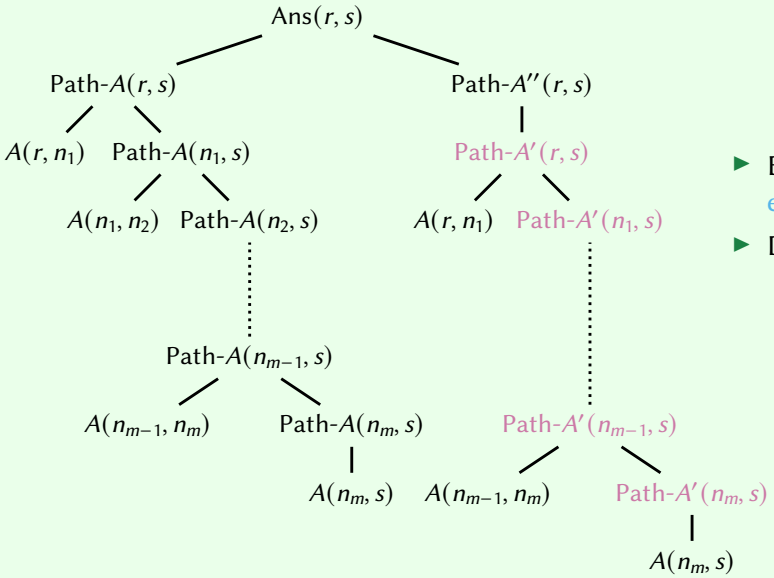
- Base facts:
  every A-edge: $m + 1$.

Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$



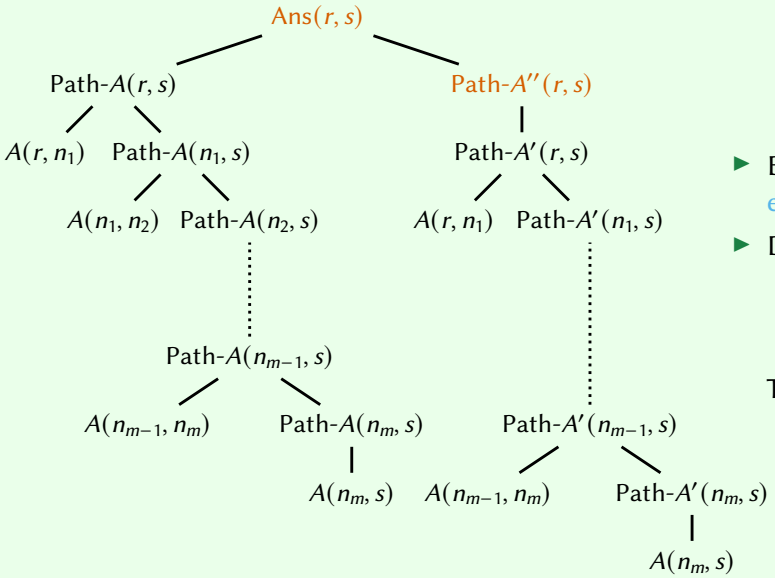
- Base facts:
  every A-edge: $m + 1$.
- Derivation steps:
  - Every Path-A term: $m + 1$.

Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$



- Base facts:
  every A-edge: $m + 1$.
- Derivation steps:
  - Every Path-A term: $m + 1$.
  - Every Path-A' term: $m + 1$.

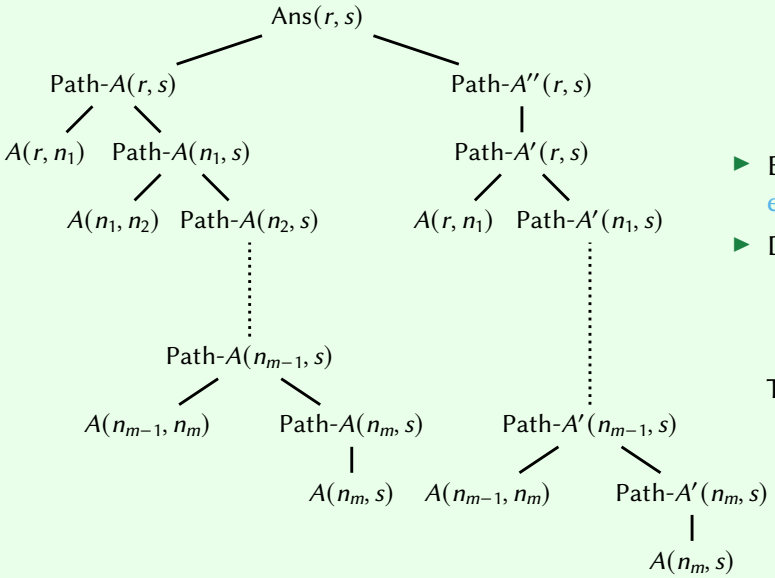
Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$



- Base facts:
  every A-edge: $m + 1$.
- Derivation steps:
  - Every Path-A term: $m + 1$.
  - Every Path-A' term: $m + 1$.
  - "Roots": 2

  Total: $2(m + 1) + 2$.

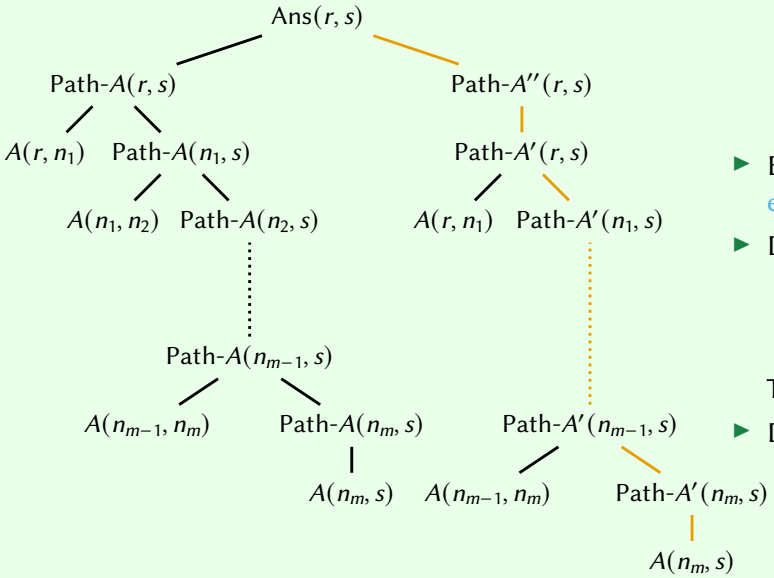
Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$



- Base facts:
  every A-edge: $m + 1$.
- Derivation steps:
  - Every Path-A term: $m + 1$.
  - Every Path-A' term: $m + 1$.
  - "Roots": 2

  Total: $2(m + 1) + 2$.

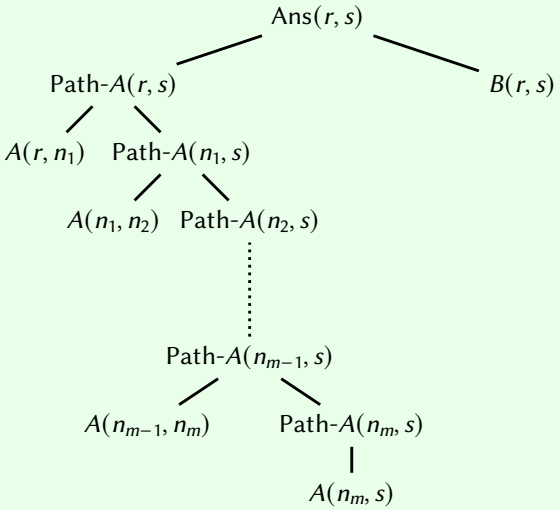
Comparing the complexity measures: $\mathcal{T}$ for explaining $\mathrm{Ans}(r, s)$



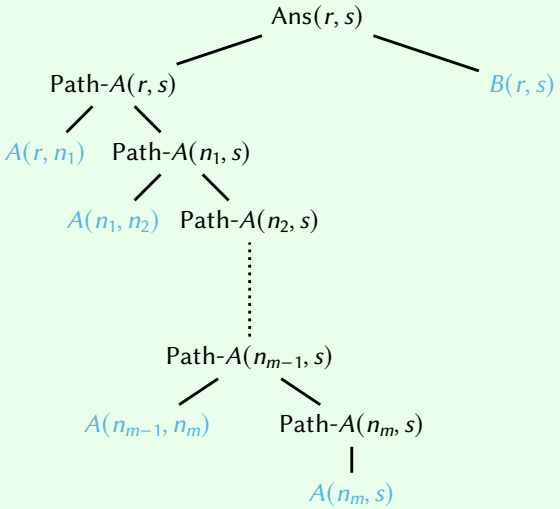
- Base facts:
  every A-edge: $m + 1$.
- Derivation steps:
  - Every Path-A term: $m + 1$.
  - Every Path-A' term: $m + 1$.
  - "Roots": 2

  Total: $2(m + 1) + 2$.
- Derivation depth: $m + 3$.

Comparing the complexity measures: $\mathcal{T}'$ for explaining $\mathrm{Ans}(r, s)$

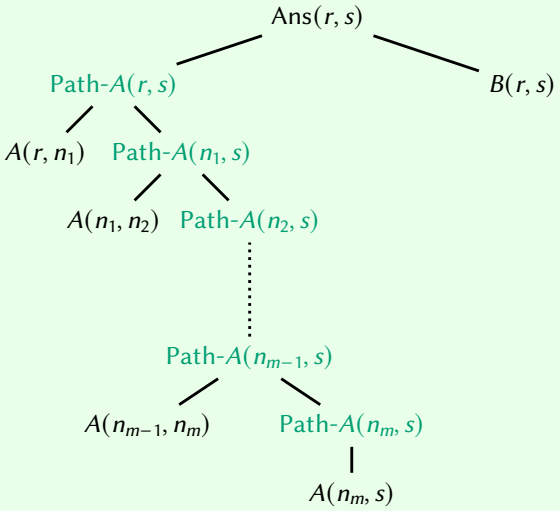


Comparing the complexity measures: $\mathcal{T}'$ for explaining $\mathrm{Ans}(r, s)$



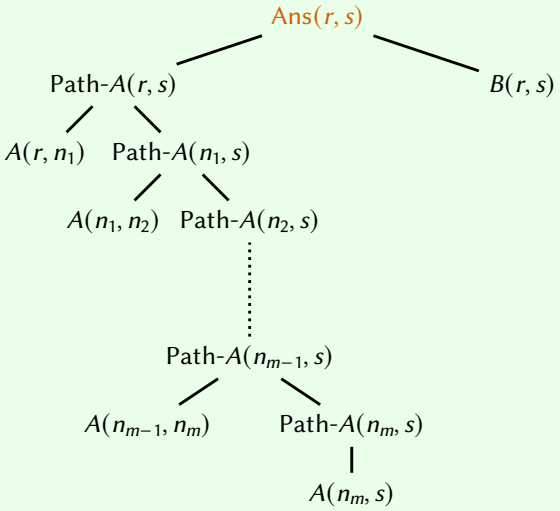
- Base facts:
  every A, B-edge: $m + 2$.

Comparing the complexity measures: $\mathcal{T}'$ for explaining $\mathrm{Ans}(r, s)$



- Base facts:
  every A, B-edge: $m + 2$.
- Derivation steps:
  - Every Path-A term: $m + 1$.

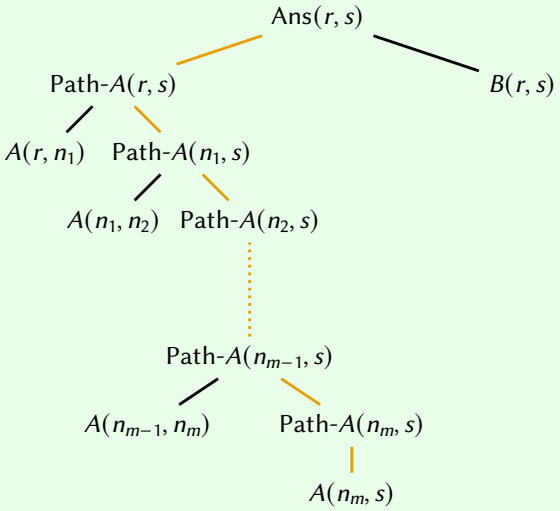
Comparing the complexity measures: $\mathcal{T}'$ for explaining $\mathrm{Ans}(r, s)$



- Base facts:
  every A, B-edge: $m + 2$.
- Derivation steps:
  - Every Path-A term: $m + 1$.
  - "Roots": 1

  Total: $(m + 1) + 1$.

Comparing the complexity measures: $\mathcal{T}'$ for explaining $\mathrm{Ans}(r, s)$



- Base facts:
  every A, B-edge: $m + 2$.
- Derivation steps:
  - Every Path-A term: $m + 1$.
  - "Roots": 1

  Total: $(m + 1) + 1$.
- Derivation depth: $m + 2$.

Comparing the complexity measures: Conclusion

	$\mathcal{T}$	$\mathcal{T}'$
C-base	$m+1$	$m+2$
C-ddepth	$m+3$	$m+2$
C-deriv	$2(m+1)+2$	$(m+1)+1$
C-uderiv	$2(m+1)+2$	$(m+1)+1$

Comparing the complexity measures: Conclusion

	$\mathcal{T}$	$\mathcal{T}'$
C-base	$m+1$	$m+2$
C-ddepth	$m+3$	$m+2$
C-deriv	$2(m+1)+2$	$(m+1)+1$
C-uderiv	$2(m+1)+2$	$(m+1)+1$

Other cases: Analogous.

Computing simple explanations

Strategy 1: Annotated evaluation

C-ddepth($\mathcal{T}$) Essentially standard semi-naive evaluation.

C-deriv($\mathcal{T}$) Semi-naive evaluation with a priority queue to guide low-cost derivation (essentially cross between Dijkstra's algorithm and semi-naive evaluation).

Computing simple explanations

Strategy 1: Annotated evaluation

C-ddepth($\mathcal{T}$) Essentially standard semi-naive evaluation.

C-deriv($\mathcal{T}$) Semi-naive evaluation with a priority queue to guide low-cost derivation (essentially cross between Dijkstra's algorithm and semi-naive evaluation).

Strategy 2: Brute force

C-base($\mathcal{T}$) For every subset S of the base facts:
Can I derive the fact using only S? $\longrightarrow$ *Keep smallest subset S.*

C-uderiv($\mathcal{T}$) For every subset S of all possible derived facts:
Can I derive the fact using only S? $\longrightarrow$ *Keep smallest subset S.*

Computing simple explanations

Strategy 1: Annotated evaluation

C-ddepth($\mathcal{T}$) Essentially standard semi-naive evaluation.

C-deriv($\mathcal{T}$) Semi-naive evaluation with a priority queue to guide low-cost derivation (essentially cross between Dijkstra's algorithm and semi-naive evaluation).

Polynomial data complexity.

Strategy 2: Brute force

C-base($\mathcal{T}$) For every subset S of the base facts:
Can I derive the fact using only S? $\longrightarrow$ *Keep smallest subset S.*

C-uderiv($\mathcal{T}$) For every subset S of all possible derived facts:
Can I derive the fact using only S? $\longrightarrow$ *Keep smallest subset S.*

NP-Complete, even with respect to small fixed-size Datalog programs.

Algorithm SNA-PQ (Semi-Naive Annotated with Priority Queue)

Algorithm SNA-PQ($\langle B, D \rangle$):

We write leaf-node(r) to denote the leaf node with label r.
result := a dictionary with $\{h \mapsto \text{leaf-node}(\text{“}h \leftarrow \bullet\text{”}) \mid \text{“}h \leftarrow \bullet\text{”} \in B\}$.
new := a priority queue initialized with facts $\{h \mid \text{“}h \leftarrow \bullet\text{”} \in B\}$.
/* This priority queue is ordered on the *derivation complexity* of facts. */
done := $\emptyset$.
while *new* $\neq \emptyset$ **do**
  Remove fact $S(d_1, \ldots, d_n)$ with lowest priority from *new* and add it to *done*.
  while there exists a derivation rule

$$\text{r} = \text{“}R(v_1, \ldots, v_m) \leftarrow S_1(w_{1,1}, \ldots, w_{1,m_1}) \wedge \cdots \wedge S_j(w_{j,1}, \ldots, w_{j,m_j})\text{”} \in D$$

  and variable assignment $\mu : \{w_{1,1}, \ldots, w_{j,m_j}\} \to \mathbb{U}$ such that

  i. there exists an i with $S(d_1, \ldots, d_n) = S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i}))$, $1 \leq i \leq j$;
  ii. for each i, $1 \leq i \leq j$, we have $S_i(\mu(w_{i,1}), \ldots, \mu(w_{i,m_i})) \in$ *done*; and
  iii. $R(\mu(v_1), \ldots, \mu(v_m)) \notin$ *done*

  do
    if $R(\mu(v_1), \ldots, \mu(v_m))$ is already in *new* **then**
      Compare this new derivation for $R(\mu(v_1), \ldots, \mu(v_m))$ with the one already stored in *result* and determine which derivation has the lowest *derivation complexity*. Update the priority for $R(\mu(v_1), \ldots, \mu(v_m))$ in *new* and update the derivation in *result* to represent the derivation with the lowest *derivation complexity*.
    else
      Add $R(\mu(v_1), \ldots, \mu(v_m))$ to the top of *new*.
      n := an internal node with label r, μ and j children
        $\mathit{result}(S_i(\mu(w_{i,1}), \ldots, \mu(w_{1,m_i}))), 1 \leq i \leq j$.
      Add $R(\mu(v_1), \ldots, \mu(v_m)) \mapsto n$ to *result*.
return *result*.

Algorithm SNA-PQ (Semi-Naive Annotated with Priority Queue)

Input Datalog program $\langle B, D \rangle$ with B the base facts, D the derivation rules.

Algorithm SNA-PQ (Semi-Naive Annotated with Priority Queue)

Input Datalog program $\langle B, D \rangle$ with B the base facts, D the derivation rules.

Initialize *result* := a dictionary:
mapping base facts in B onto leaf-nodes

Initialize *new* := a priority queue of facts, ordered on increasing derivation cost:
mapping base facts onto cost 0.

Initialize *done* := $\emptyset$.

Algorithm SNA-PQ (Semi-Naive Annotated with Priority Queue)

Input Datalog program $\langle B, D \rangle$ with B the base facts, D the derivation rules.

Initialize *result* := a dictionary:
mapping base facts in B onto leaf-nodes

Initialize *new* := a priority queue of facts, ordered on increasing derivation cost:
mapping base facts onto cost 0.

Initialize *done* := $\emptyset$.

Repeat While *new* $\neq \emptyset$:

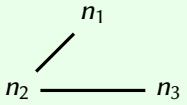
1. Take lowest-cost fact g from *new*.
2. Add g to *done*.
3. Produce facts h using g and other facts in *done* using D.
4. If $h \notin$ *done*: add h to *new* or update cost of h in *new*.

Example of Algorithm SNA-PQ

Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$

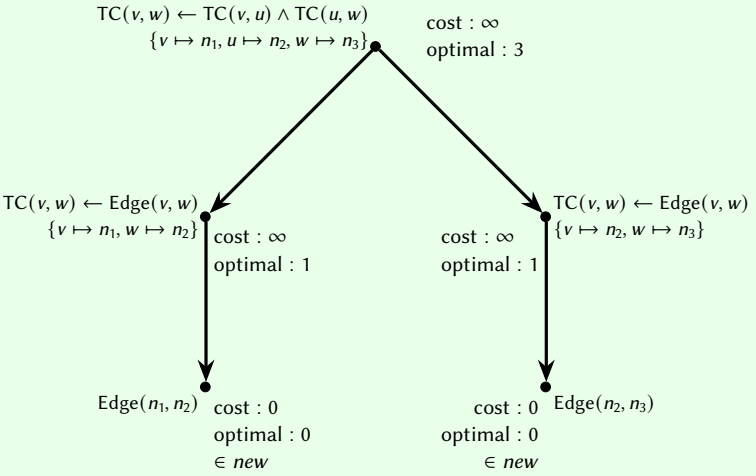
$\mathrm{TC}(v, w) \leftarrow \mathrm{Edge}(v, w);$

$\mathrm{TC}(v, w) \leftarrow \mathrm{TC}(v, u) \wedge \mathrm{TC}(u, w)$



Example of Algorithm SNA-PQ

Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



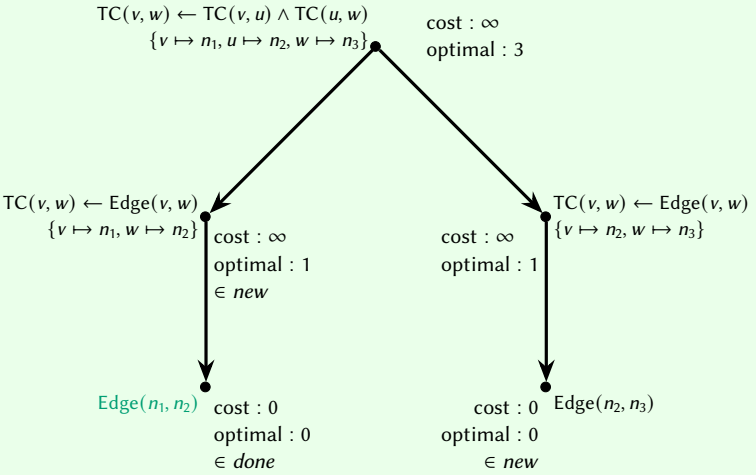
new

$\text{Edge}(n_1, n_2) : 0$

$\text{Edge}(n_2, n_3) : 0$

Example of Algorithm SNA-PQ

Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



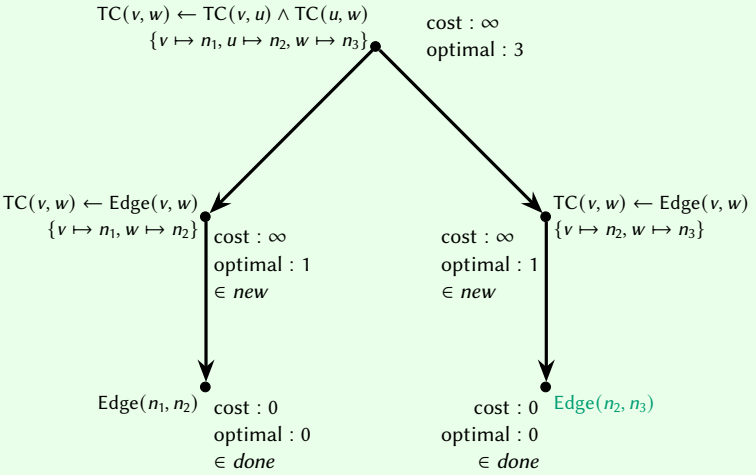
new

$\text{Edge}(n_2, n_3) : 0$

$\text{TC}(n_1, n_2) : 1$

Example of Algorithm SNA-PQ

Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



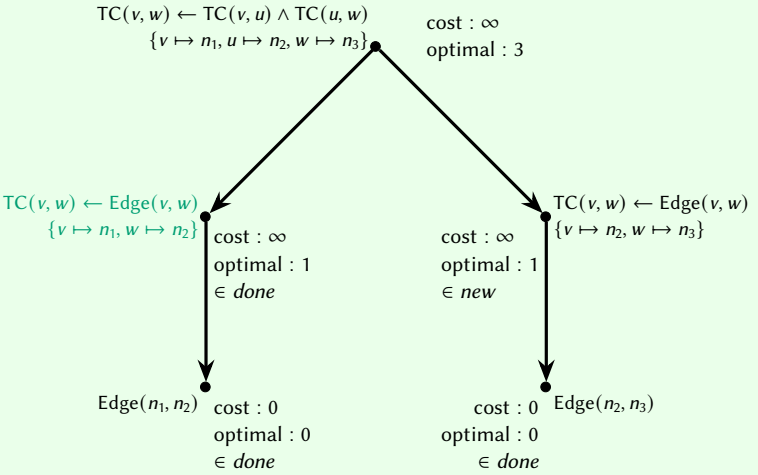
new

$\mathrm{TC}(n_1, n_2) : 1$

$\mathrm{TC}(n_2, n_3) : 1$

Example of Algorithm SNA-PQ

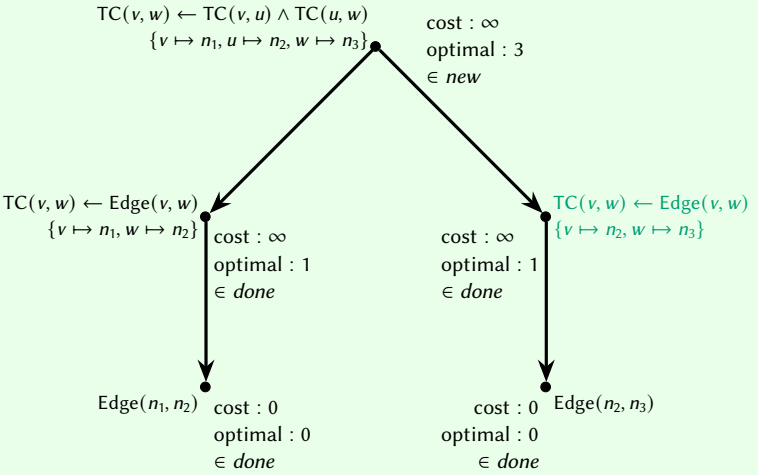
Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



$$\frac{new}{\mathrm{TC}(n_2, n_3) : 1}$$

Example of Algorithm SNA-PQ

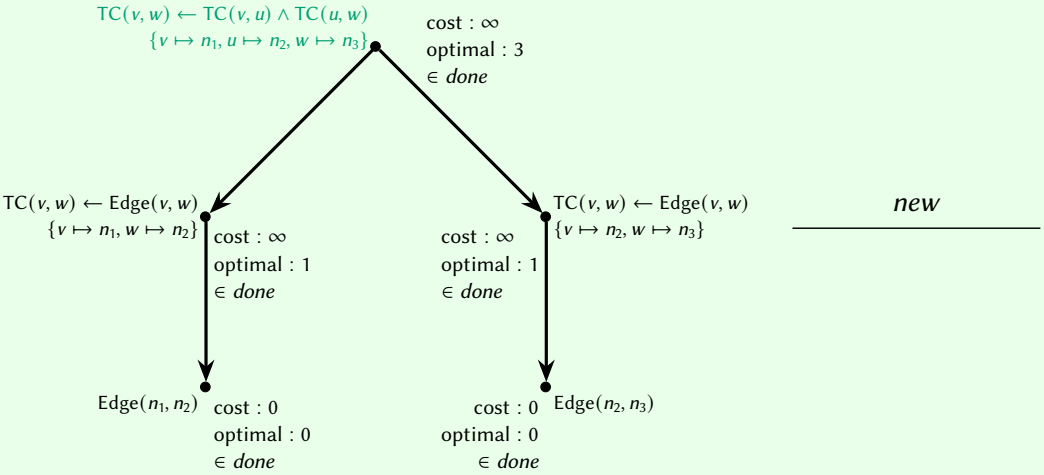
Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



$$\frac{new}{\mathrm{TC}(n_1, n_3) : 3}$$

Example of Algorithm SNA-PQ

Explain $TC(n_1, n_3)$ via the path $n_1 n_2 n_3$



Example of Algorithm SNA-PQ: updating cost in *new*

Assume we have two derivation rules for $G(x, y)$:

$$G(x, y) \leftarrow A(x, z), B(z, y)$$
$$G(x, y) \leftarrow C(x, z), D(z, y), E(x, y).$$

Assume we have

$$\text{optimal}(A(m, o)) = 4 \qquad \text{optimal}(B(o, n)) = 4;$$
$$\text{optimal}(C(m, o)) = 3 \qquad \text{optimal}(D(o, n)) = 3 \qquad \text{optimal}(E(m, n)) = 3.$$

Consider deriving $G(m, n)$

Example of Algorithm SNA-PQ: updating cost in *new*

Assume we have two derivation rules for $G(x, y)$:

$$G(x, y) \leftarrow A(x, z), B(z, y)$$
$$G(x, y) \leftarrow C(x, z), D(z, y), E(x, y).$$

Assume we have

$$\text{optimal}(A(m, o)) = 4 \qquad \text{optimal}(B(o, n)) = 4;$$
$$\text{optimal}(C(m, o)) = 3 \qquad \text{optimal}(D(o, n)) = 3 \qquad \text{optimal}(E(m, n)) = 3.$$

Consider deriving $G(m, n)$

- First, derive via $C(m, o), D(o, n), E(m, n)$ with cost 10
  because $C(m, o), D(o, n), E(m, n)$ have lower cost than $A(m, o), B(o, n)$.
  $\rightarrow$ Add $G(m, n)$ to *new* with cost 10.

Example of Algorithm SNA-PQ: updating cost in *new*

Assume we have two derivation rules for $G(x, y)$:

$$G(x, y) \leftarrow A(x, z), B(z, y)$$
$$G(x, y) \leftarrow C(x, z), D(z, y), E(x, y).$$

Assume we have

$$\text{optimal}(A(m, o)) = 4 \qquad \text{optimal}(B(o, n)) = 4;$$
$$\text{optimal}(C(m, o)) = 3 \qquad \text{optimal}(D(o, n)) = 3 \qquad \text{optimal}(E(m, n)) = 3.$$

Consider deriving $G(m, n)$

- First, derive via $C(m, o), D(o, n), E(m, n)$ with cost 10
  because $C(m, o), D(o, n), E(m, n)$ have lower cost than $A(m, o), B(o, n)$.
  $\rightarrow$ Add $G(m, n)$ to *new* with cost 10.
- Then, derive $G(m, n)$ via $A(m, o), B(o, n)$ with cost 9.
  $\rightarrow$ Update $G(m, n)$ in *new* to cost 9.

Correctness of Algorithm SNA-PQ

Prove invariants on the algorithm state

1. The facts $g \in \mathit{new}$ are ordered on increasing $\text{cost}(g)$.

2. For all facts g, we either have:
   - 2.1 fact g is in *done* with $\text{cost}(g) = \text{optimal}(g) \leq x$; or
   - 2.2 fact g is in *new* with $x \leq \text{cost}(g) = \text{optimal}(g)$; or
   - 2.3 $\text{cost}(g) > \text{optimal}(g) > x$ and, for every tree $\mathcal{T}$ representing a C-deriv-simple explanation of fact g and every fact $g' \in \mathcal{T}$ with $\text{cost}(g') > \text{optimal}(g')$, every path from g' to a leaf in $\mathcal{T}$ will reach a fact g'' such that $g'' \in (\mathit{new} \cup \mathit{done})$ and $\text{cost}(g'') = \text{optimal}(g'')$, and at least one of these reachable facts g'' is in *new*.

Correctness of Algorithm SNA-PQ

Prove invariants on the algorithm state

1. The facts $g \in \mathit{new}$ are ordered on increasing $\text{cost}(g)$.
   new is a priority queue on derivation cost.
2. For all facts g, we either have:
   2.1 fact g is in *done* with $\text{cost}(g) = \text{optimal}(g) \leq x$; or

   2.2 fact g is in *new* with $x \leq \text{cost}(g) = \text{optimal}(g)$; or

   2.3 $\text{cost}(g) > \text{optimal}(g) > x$ and, for every tree $\mathcal{T}$ representing a C-deriv-simple explanation of fact g and every fact $g' \in \mathcal{T}$ with $\text{cost}(g') > \text{optimal}(g')$, every path from g' to a leaf in $\mathcal{T}$ will reach a fact g'' such that $g'' \in (\mathit{new} \cup \mathit{done})$ and $\text{cost}(g'') = \text{optimal}(g'')$, and at least one of these reachable facts g'' is in *new*.

Correctness of Algorithm SNA-PQ

Prove invariants on the algorithm state

1. The facts $g \in \mathit{new}$ are ordered on increasing $\text{cost}(g)$.
   new is a priority queue on derivation cost.
2. For all facts g, we either have:
   2.1 fact g is in *done* with $\text{cost}(g) = \text{optimal}(g) \leq x$; or
   Fact g has been completed, optimal derivation found.
   2.2 fact g is in *new* with $x \leq \text{cost}(g) = \text{optimal}(g)$; or

   2.3 $\text{cost}(g) > \text{optimal}(g) > x$ and, for every tree $\mathcal{T}$ representing a C-deriv-simple explanation of fact g and every fact $g' \in \mathcal{T}$ with $\text{cost}(g') > \text{optimal}(g')$, every path from g' to a leaf in $\mathcal{T}$ will reach a fact g'' such that $g'' \in (\mathit{new} \cup \mathit{done})$ and $\text{cost}(g'') = \text{optimal}(g'')$, and at least one of these reachable facts g'' is in *new*.

Correctness of Algorithm SNA-PQ

Prove invariants on the algorithm state

1. The facts $g \in \mathit{new}$ are ordered on increasing $\text{cost}(g)$.
   new is a priority queue on derivation cost.
2. For all facts g, we either have:
   - 2.1 fact g is in *done* with $\text{cost}(g) = \text{optimal}(g) \leq x$; or
     Fact g has been completed, optimal derivation found.
   - 2.2 fact g is in *new* with $x \leq \text{cost}(g) = \text{optimal}(g)$; or
     Fact g has not been completed, optimal derivation already found.
   - 2.3 $\text{cost}(g) > \text{optimal}(g) > x$ and, for every tree $\mathcal{T}$ representing a C-deriv-simple explanation of fact g and every fact $g' \in \mathcal{T}$ with $\text{cost}(g') > \text{optimal}(g')$, every path from g' to a leaf in $\mathcal{T}$ will reach a fact g'' such that $g'' \in (\mathit{new} \cup \mathit{done})$ and $\text{cost}(g'') = \text{optimal}(g'')$, and at least one of these reachable facts g'' is in *new*.

Correctness of Algorithm SNA-PQ

Prove invariants on the algorithm state

1. The facts $g \in \mathit{new}$ are ordered on increasing $\text{cost}(g)$.
   new is a priority queue on derivation cost.
2. For all facts g, we either have:
   - 2.1 fact g is in *done* with $\text{cost}(g) = \text{optimal}(g) \leq x$; or
     Fact g has been completed, optimal derivation found.
   - 2.2 fact g is in *new* with $x \leq \text{cost}(g) = \text{optimal}(g)$; or
     Fact g has not been completed, optimal derivation already found.
   - 2.3 $\text{cost}(g) > \text{optimal}(g) > x$ and, for every tree $\mathcal{T}$ representing a $\mathcal{C}$-deriv-simple explanation of fact g and every fact $g' \in \mathcal{T}$ with $\text{cost}(g') > \text{optimal}(g')$, every path from g' to a leaf in $\mathcal{T}$ will reach a fact g'' such that $g'' \in (\mathit{new} \cup \mathit{done})$ and $\text{cost}(g'') = \text{optimal}(g'')$, and at least one of these reachable facts g'' is in *new*.
     Fact g has not been completed, optimal derivation not found:
     Fact g explained using facts g', which depend on a fact g'' which is not completed yet.

Computing simple explanations: Brute force

Strategy 2: Brute force

C-base($\mathcal{T}$) For every subset S of the base facts:
Can I derive the fact using only S? ⟶ Keep smallest subset S.

C-uderiv($\mathcal{T}$) For every subset S of all possible derived facts:
Can I derive the fact using only S? ⟶ Keep smallest subset S.

NP-Complete, even with respect to small fixed-size Datalog programs.

Computing simple explanations: Brute force

Let $C \in \{C\text{-base}, C\text{-deriv}, C\text{-uderiv}, C\text{-ddepth}\}$ be a complexity measure.

Definition (The *C-simple explanation problem*)

Given a Datalog program, fact f, and integer k,
is there an explanation $\mathcal{T}$ for the derivation of fact f with $C(\mathcal{T}) \leq k$.

Computing simple explanations: Brute force

Let $C \in \{C\text{-base}, C\text{-deriv}, C\text{-uderiv}, C\text{-ddepth}\}$ be a complexity measure.

Definition (The *C-simple explanation problem*)

Given a Datalog program, fact f, and integer k,
is there an explanation $\mathcal{T}$ for the derivation of fact f with $C(\mathcal{T}) \leq k$.

Theorem

The C-base-simple and the C-uderiv-simple explanation problems are NP-complete.

Computing simple explanations: Brute force

Theorem

The C-base-simple and the C-uderiv-simple explanation problems are NP-complete.

This is already true for simple Datalog programs:

- *all facts have arity at-most two;*
- *all derivaton rules have at-most two terms; and*
- *with only seven derivation rules.*

Computing simple explanations: Brute force

Theorem

The C-base-simple and the C-uderiv-simple explanation problems are NP-complete.

Proof (sketch)

Reduce the NP-complete vertex cover problem to these explanation problems:

- The nodes v get encoded by base facts $Node(v)$.
- The i-th edges (m, n) get encoded by $EndPoint(i, m) \leftarrow \bullet$ and $EndPoint(i, n) \leftarrow \bullet$.
- An order on edges get encoded by $SymbolOrder(i, j)$ with $j = i + 1$ edge identifiers.
- The vertex cover problem gets encoded by seven derivation rules whose C-base-simple or C-uderiv-simple explanations answer the vertex cover problem.

Conclusion

Goal: Gain a basic understaning of how recursion (Datalog) can be explained.
Refined Goal: Understand easy-to-use explanations.

- Introduced four distinct complexity measures for Datalog explanations.
- Introduced four distinct types of simple explanations.
- Evaluation algorithms for two types of simple explanations.
- NP-completeness results for the evaluation of the other two types of explanations.

Encouraging results: some types of explanations are practically feasible.

Conclusion

Goal: Gain a basic understaning of how recursion (Datalog) can be explained.
Refined Goal: Understand easy-to-use explanations.

- Introduced four distinct complexity measures for Datalog explanations.
- Introduced four distinct types of simple explanations.
- Evaluation algorithms for two types of simple explanations.
- NP-completeness results for the evaluation of the other two types of explanations.

Encouraging results: some types of explanations are practically feasible.

Future work

Practical high-performance top-down and goal-oriented evaluation algorithms:
Ongoing: Evaluation engine that uses relational algebra for non-recursive parts,
Integrating *explanation tree construction* into relational algebra and Datalog.