Datalog 查询的 why-来源复杂度

Marco Calautti[1] , **Ester Livshits**[2] , **Andreas Pieris**[2,3] , **Markus Schneider**[2]

[1]米兰大学计算机科学系[2]爱丁堡大学信息学院[3]塞浦路斯大学计算机科学系

marco.calautti@unimi.it, ester.livshits@ed.ac.uk, apieris@inf.ed.ac.uk, m.schneider@ed.ac.uk

arXiv:2303.12773v1 [cs.DB] 22 Mar 2023

摘要

解释数据库查询结果为何产生，是实现可解释人工智能这一目标的一项重要任务。尤其是在当今，Datalog 等表达能力强的数据库查询语言在基于本体的应用开发中发挥着关键作用。解释查询结果的一种标准方法是所谓的 why-来源，它主要通过输入数据库的子集来提供有关查询结果见证的信息；这些子集足以推导出该结果。令我们惊讶的是，尽管 Datalog 查询的 why-来源概念早已提出并得到深入研究，其计算复杂度却仍未得到探索。本研究旨在填补 why-来源文献中这一明显空白。为此，我们明确刻画了 Datalog 查询及其关键子类的 why-来源数据复杂度。我们的研究表明，递归查询的 why-来源是一个难处理的问题，即使递归被限制为线性；而对于非递归查询，该问题则非常易处理。尽管如此，我们通过利用 SAT 求解器进行实验确认，在实践中实现（递归）Datalog 查询的 why-来源并非不切实际的目标。

1 引言

Datalog 于 20 世纪 80 年代从逻辑编程领域发展而来，成为一种基于逻辑的查询语言，此后得到了广泛研究（Abiteboul、Hull 和 Vianu，1995）。Datalog 这一名称体现了其设计初衷：为数据处理开发一种与 Prolog 对应的语言。它本质上扩展了合取查询的并集这一语言（该语言对应于关系代数中的选择-投影-连接-并集片段），加入了递归这一重要特性，而表达某些自然查询亟需这一特性。在众多应用中，Datalog 被广泛用于本体查询应答。尤其是，对于若干基于描述逻辑和存在规则的重要本体语言，本体查询应答可以归约为计算 Datalog 查询的问题（例如，参见 Eiter 等人，2012；Benedikt 等人，2022），进而可以利用 DLV（Leone 等人，2006）和 Clingo（Gebser 等人，2016）等高效的 Datalog 引擎。

与其他查询语言一样，解释为何会得到某个 Datalog 查询的结果，对于构建可解释且透明的数据密集型应用至关重要。一种标准方法是为查询答案提供此类解释的方式，即所谓的*why*-来源（Buneman、Khanna 和 Tan，2001）。其核心是收集输入数据库中所有足以推导出某个答案的子集。更准确地说，对于 Datalog 查询，答案元组 $\bar{t}$ 的 why-来源 t 是通过考察事实 Ans($\bar{t}$) 的所有可能证明树 T 得到的，其中 Ans 是相关 Datalog 查询的答案谓词；随后，收集所有标记这些证明树 T 叶节点的数据库事实。回顾一下，相对于数据库 D 和一组 Datalog 规则 Σ，事实 α 的证明树是一种树状表示，展示了如何从 D 出发并执行 Σ 中出现的规则来推导出 α（Abiteboul、Hull 和 Vianu，1995年）。

近期有一些研究探讨了 Datalog 查询的 why-来源概念。具体而言，相关工作包括：关于如何计算 why-来源的理论研究（Dam´asio、Analyti 和 Antoniou2013；Deutchet al. 2014）；旨在通过下近似 why-来源实现高效计算的尝试（Zhao、Subotic 和 Scholz 2020）；针对非递归 Datalog 查询这一受限情形的研究（Lee、Lud¨ascher 和 Glavic 2019）；通过将基实例化 Datalog 规则转换为方程组来计算 why-来源的尝试（Esparza、Luttenberger 和 Schlund 2014）；以及通过转换为存在规则，按需计算 why-来源的尝试（Elhalawati、Kr¨otzsch 和 Mennicke 2022）。

尽管上述研究已经探讨了 Datalog 查询的 why-来源概念，但令人意外的是，仍有一个根本性问题尚未得到探索：

主要研究问题：*Datalog* 查询的 *why*-来源的计算复杂度究竟是多少？

本研究旨在回答上述问题。为此，对于一个 Datalog 查询 Q，我们研究以下算法问题的复杂度，并将其命名为 Why-Provenance[Q]：给定一个数据库 D、关于 Q 在 D 上的一个答案 $\bar{t}$，以及 D 的一个子集 D'，D' 是否属于关于 D 和 Q 的 t 的 why-来源？确定上述判定问题的复杂度，将使我们能够理解针对数据库大小而言，Datalog 查询的 why-来源所固有的复杂度；而这正是实际使用 why-来源时所关注的。

我们的贡献。复杂度分析的结论是，通过 why-来源 来解释 Datalog 查询通常是一个

难处理的问题。具体而言，对于 Dat-alog 查询 Q，我们证明 Why-Provenance[Q] 属于 NP，而且存在一些查询使得该问题为 NP困难。我们进一步分析了当 Q 为线性（即递归受限为线性递归）或非递归时该问题的复杂度，旨在阐明递归这一特性是否会影响 why-来源 的固有复杂度。我们证明，将递归限制为线性并不会影响复杂度；也就是说，该问题属于 NP，且对于某些查询甚至是 NP困难的。然而，完全移除递归会显著降低复杂度；具体而言，我们证明该问题属于 AC_0。

显然，Datalog 查询的 why-来源概念，以及由此产生的问题 Why-Provenance[Q]，在很大程度上依赖于证明树的概念。然而，正如文献中已有讨论（例如，近期的研究 Bourgaux 等人，2022），有些证明树并不符合直觉，因为它们表示不自然的推导（例如，某个事实被用于推导自身，或某个事实以多种不同方式被推导出来），这也会影响 why-来源。为克服证明树这一概念上的局限，我们提出了无歧义证明树这一类别。在这类证明树中，某个事实的所有出现都必须通过同一种推导来证明。随后，我们研究了以无歧义证明树为基础的问题 Why-Provenance[Q]，并证明其复杂度保持不变。我们应当将此视为积极成果，因为我们克服了任意证明树的局限，同时没有增加复杂度。

最后，我们验证了无歧义证明树除了在概念上的优势外，还有助于利用现成的 SAT 求解器，高效计算 Datalog 查询的 why- 来源。具体而言，我们讨论了一种概念验证实现，该实现利用了最先进的 SAT 求解器 Glucose（例如，参见（ Audemard 和 Simon，2018）），并展示了基于 Datalog 文献中的查询和数据库所得的令人鼓舞的结果。

包含更多细节的扩展版本、实验情景以及源代码，可在 *https://gitlab.com/mcalautti/datalog-why-provenance* 查阅。

2 预备知识

我们考虑两个互不相交的可数无限集合**C** 和**V**，分别由常量和变量组成。我们也将常量和变量称为项。为简洁起见，给定一个整数$n > 0$，我们可以用 $[n]$ 表示整数集合 $\{1, \ldots, n\}$。

关系数据库。 模式 **S** 是一个有限的关系名（或谓词）集合，每个关系名（或谓词）都关联一个元数。我们用 R/n表示 R 的元数为 $n \geq 0$；也可以将 ar(R) 写作n。一个（关系）原子 α 是定义在**S** 上的表达式，其形式为$R(\bar{t})$，其中 $R/n \in \mathbf{S}$，且 $\bar{t}$ 是一个由项组成的 n元组。为简便起见，我们可以将元组视为其元素组成的集合。事实是只包含常量的原子。定义在**S** 上的数据库是一个由**S**上的事实构成的有限集合。数据库 D的活动域记作 dom(D)，是 D中常量构成的集合。

Datalog 程序的语法与语义。 一个模式 **S**上的 （*Datalog*）规则 σ 是如下形式的表达式

$$R_0(\bar{x}_0) \text{ :- } R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$$

对于 $n \geq 1$，其中 $R_i(\bar{x}_i)$ 是模式 **S** 上的一个（不含常量的）关系原子，适用于 $i \in \{0, \ldots, n\}$；并且 $\bar{x}_0$ 中的每个变量都出现在 $\bar{x}_k$ 中，适用于某个 $k \in [n]$。我们将 $R_0(\bar{x}_0)$ 称为 σ的头部，记作 head(σ)；将出现在 :– 符号右侧的表达式称为 σ的体部，记作 body(σ)，并可将其视为由其中的原子构成的集合。

定义在模式**S** 上的*Datalog* 程序是由定义在**S**上的 Datalog 规则构成的有限集合 Σ。若谓词 R 出现在Σ 中，且 Σ 中没有规则以 R 为头部，则称其为外延谓词；若 Σ中至少有一条规则以 R 为头部，则称其为内涵谓词。 Σ的外延（数据库）模式记作 edb(Σ)，是 Σ中所有外延谓词的集合；Σ的内涵模式记作 idb(Σ)，是 Σ中所有内涵谓词的集合。注意，根据定义，edb(Σ) $\cap$idb(Σ) $= \emptyset$。 Σ的模式记作 sch(Σ)，是 edb(Σ)$\cup$ idb(Σ)，通常是**S**的子集，因为**S** 中的某些谓词可能不会出现在 Σ中。

Datalog 程序中存在一些有趣的片段，它们以某种方式限制了递归，并在文献中得到了广泛研究。若 Datalog 程序 Σ 的每条规则 $\sigma \in \Sigma$ 中，body(σ) 在 idb(Σ) 上至多包含一个原子，即 body(σ) 至多提及一个内涵谓词，则称该程序为线性。粗略地说，线性 Datalog 程序只能进行线性递归。另一个重要的片段则完全禁止递归。若 Datalog 程序Σ 的谓词图（用于编码 sch(Σ) 中谓词之间的依赖关系）是无环的，则称该程序为非递归。回想一下，Σ 的谓词图中的节点是 sch(Σ) 的谓词；如果 Σ 中存在形如 $P(\bar{x})$:– $\ldots, R(\bar{y}) \ldots$的规则，则图中有一条从 R 到 P 的边。

Datalog 程序有一个优雅的性质：它具有三种等价的语义，即模型论语义、不动点语义和证明论语义（Abiteboul、Hull 和 Vianu，1995 年）。我们接下来回顾 Datalog 程序的证明论语义，因为它与 why-来源的概念更为接近。为此，我们需要引入事实的证明树这一关键概念；无论如何，它在我们的工作中都将发挥重要作用。对于数据库D 和 Datalog 程序 Σ，令 base(D, Σ) $= \{R(\bar{t}) \mid R \in$sch($\Sigma$) 且 $\bar{t} \in \text{dom}(D)^{\text{ar}(R)}\}$，它表示使用 sch($\Sigma$) 中的谓词和 dom($D$) 中的项所能构成的所有事实的集合。

定义 1 (**证明树**)。 考虑一个 *Datalog* 程序 Σ、一个基于 扩展数据库 (Σ)的数据库 D，以及一个基于 模式(Σ)的事实 α。关于 α 和 Σ，D 的证明树是一个有限标记根树$T = (V, E, \lambda)$，其中 $\lambda \colon V \to$基础 (D, Σ)，满足：

1. 如果 $v \in V$ 是根节点，那么 $\lambda(v) = \alpha$。
2. 如果 $v \in V$ 是叶节点，那么 $\lambda(v) \in D$。
3. 如果 $v \in V$ 是一个有 $n \geq 1$ 个子节点 u_1、.....、u_n 的节点，那么存在一条规则 $R_0(\bar{x}_0)$:– $R_1(\bar{x}_1) \ldots, R_n(\bar{x}_n) \in \Sigma$，以及一个函数 $h \colon \bigcup_{i \in [n]} \bar{x}_i \to C$，使得对于每个 $i \in [n]$，$\lambda(v) = R_0(h(\bar{x}_0))$，且 $\lambda(u_i) = R_i(h(\bar{x}_i))$。 ∎

本质上，事实 α 关于 D 和 Σ 的证明树表明，我们可以使用 D 和 Σ 证明 α；也就是说，我们可以从 D 出发，执行 Σ 中的规则来推导出 α。下面给出一个例子，用以说明证明树的概念；本文也将始终以此为例。

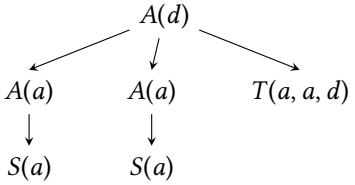
例 1. 考虑由以下内容组成的 *Datalog* 程序 Σ

$$A(x) \quad :- \quad S(x)$$
$$A(x) \quad :- \quad A(y), A(z), T(y, z, x)$$

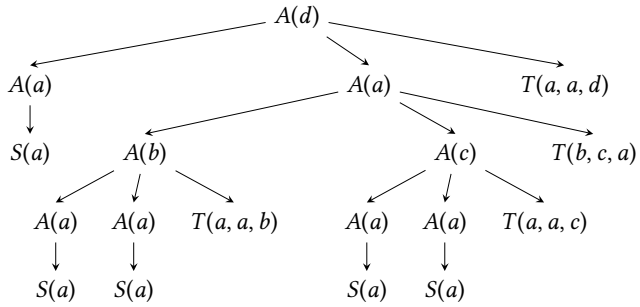
编码了路径可达性问题（*Cook 1974*）。谓词 S 表示源节点，A 表示从源节点可达的节点，而 T 表示可达性条件，也就是说，$T(y, z, x)$ 表示如果 y 和 z 都能从源节点到达，那么 x 也能从源节点到达。我们进一步考虑数据库

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

事实 $A(d)$ 相对于 D 和 Σ 的一个简单证明树如下：



下面是事实 $A(d)$ 相对于 D 和 Σ的另一个稍复杂一些的证明树：



请注意，上述只是 $A(d)$ 相对于 D 和 Σ的众多证明树中的两个。事实上，这样的证明树可以不断构造得更大，因此存在无穷多个：每当我们遇到一个标记为 $A(a)$的节点时，都可以选择应用递归规则，而不是规则 $A(x) :- S(x)$。 ■

现在，给定一个 Datalog 程序 Σ 和一个定义在 sch(Σ) 上的数据库 D， D 上 Σ的语义记作 $\Sigma(D)$，是集合

$$\Sigma(D) = \{\alpha \mid \text{there is a proof tree of } \alpha \text{ w.r.t. } D \text{ and } \Sigma\},$$

也就是说，使用 D 和 Σ可以证明的事实集合。

Datalog 查询。有了 Datalog 程序的语法和语义，现在可以直接回顾 Datalog 查询的语法和语义。*Datalog* 查询 是一个二元组 $Q = (\Sigma, R)$，其中 Σ 是一个 Datalog 程序， R 是 idb(Σ) 中的一个谓词。我们进一步称 Q 为线性（分别为非递归），如果程序 Σ 是线性的（分别为非递归）。现在，对于一个定义在 edb(Σ) 上的数据库 D， D 上 Q 的答案定义为以下元组集合

$$Q(D) = \{\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)} \mid R(\bar{t}) \in \Sigma(D)\},$$

也就是说，满足以下条件的元组 $\bar{t}$：事实 $R(\bar{t})$ 可以使用D 和 Σ证明。包含所有 Datalog 查询的类记为 Dat。我们还分别用 LDat 和 NRDat 表示线性 Datalog 查询类和非递归 Datalog 查询类。

3 Datalog 查询的 why-来源

如引言中所述，why-来源是一种解释为何会得到某个查询结果的标准方式。它实质上收集数据库的所有子集（不含不必要的原子），这些子集能够用于证明（或推导）某个查询结果。接下来，我们将这一简单想法形式化，然后介绍我们关注的主要问题。

给定一棵证明树 $T = (V, E, \lambda)$（针对某个数据库和 Datalog 程序中的某个事实）， T的支撑集 是集合

$$\mathsf{support}(T) = \{\lambda(v) \mid v \in V \text{ is a leaf of } T\},$$

本质上，这是标记证明树 T的叶节点的事实集合。注意，support(T) 是底层数据库的子集，因为根据定义，证明树的叶节点标记为数据库原子。下面给出 Datalog 查询的 why-来源的形式化定义。

定义 2(Datalog 的 why-来源)。考虑一个 *Datalog* 查询 $Q = (\Sigma, R)$、一个定义在 扩展数据库(Σ)上的数据库 D，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$。$\bar{t}$ 的 why-来源 D相对于 Q和的定义是事实集的集合族

$$\{\mathsf{support}(T) \mid T \textit{ is a proof tree of } R(\bar{t}) \textit{ w.r.t. } D \textit{ and } \Sigma\}$$

我们将其记为 why($\bar{t}, D, Q$)。 ■

直观地说，属于 why($\bar{t}$， D， Q) 的事实集 $D' \subseteq D$ 应理解为一个“真正的”理由，用来说明元组 $\bar{t}$ 为何是数据库 D 上的查询 Q 的答案，即 D' 解释了为何 $\bar{t} \in Q(D)$。所谓“真正的”，是指 D' 中的所有事实确实都用于推导出元组 $\bar{t}$，使其成为答案。下面给出一个简单的 why-来源示例。

例 *2.* 令 $Q = (\Sigma, A)$，其中 Σ 是如例 *1* 中将路径可达性问题编码为程序的程序，并令 D 为例 *1* 中的数据库。可以验证，关于 D 和 Q的单元组 *(d)* 的 *why*-来源由 {*S(a) T(a, a, d)*} 以及数据库 D 本身组成。前一个集合实际上是例 *1* 中给出的第一棵证明树的支撑集，而 D 是第二棵证明树的支撑集。回想一下， *A(d)* 关于 D 和Σ有无限多棵证明树，而 *why((d) D, Q)* 只包含两个集合。因此，一般而言，事实 *R(t̄)* 的证明树与 $\bar{t}$ 的 *why*-来源成员之间并不存在一一对应关系。 ■

我们希望明确计算相对于数据库和 Datalog 查询的某个元组的 why-来源这一问题的固有复杂度。为此，我们需要研究判定数据库的某个子集是否属于 why-来源的复杂度，也就是说，判定某个候选解释是否确实是一个解释。这引出了以下以 Datalog 查询类 C 为参数的算法问题；C 可以是 Dat、LDat 或 NRDat 等：

问题:	Why-来源[C]
输入:	来自 C 的 Datalog 查询 $Q = (\Sigma, R)$， 一个数据库 D 在扩展数据库(Σ)上， 一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，以及 $D' \subseteq D$。
问题:	Does $D' \in$ why($\bar{t}, D, Q$)?

我们的目标是研究上述问题并明确其复杂度。我们实际关注的是 Why-来源[C], 的数据复杂度，其中查询 Q是固定的，只有数据库 D、元组 $\bar{t}$和 D' 属于输入；也就是说，对于 C 中每个 $Q = (\Sigma,\ R)$，我们都考虑以下问题：

问题：Why-来源[Q]输入：一个关于扩展数据库(Σ)的数据库 D、一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，以及 $D' \subseteq D$。
问题： $D' \in \mathsf{why}(\bar{t}, D, Q)$ 是否成立?

按照惯例，如果对于 C 中的每个查询 Q，Why-来源[Q] 都属于某个复杂性类 C，那么问题 Why-来源[C]在数据复杂度意义下属于该复杂性类。另一方面，如果存在一个来自 C 的查询 Q，使得 Why-来源[Q] 对于某个复杂性类 C 是难的，那么问题 Why-来源[C] 在数据复杂度意义下对于该复杂性类是难的。

4 Why-来源的数据复杂度

本节旨在确定每个 C $\in$ {Dat、LDat、NRDat} 所对应的 Why-来源[C], 的数据复杂度。正如我们将看到的，分析的主要结论是：对于递归查询，即使递归是线性的，该问题通常也是难处理的；而对于非递归查询，该问题则非常易处理。我们首先关注递归查询。

4.1 递归查询

我们给出以下复杂度结果：

定理 3. Why-Provenance[C] 对每个类 C $\in$ {Dat,LDat}而言，在数据复杂度上都是 NP完全的。

请注意，why-来源问题与查询求值问题之间存在显著差异。众所周知，后者在数据复杂度上属于 PTIME；事实上，对于线性Datalog查询，它属于 NL（Dantsin 等人，2001）。要证明定理 3，只需证明：

- Why-Provenance[Dat]在数据复杂度上属于 NP。
- Why-Provenance[LDat]在数据复杂度上是 NP困难的。

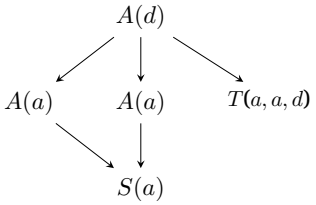
下界通过从 3SAT 进行归约来确立。实际上，我们设计了一个线性 Datalog 查询 Q，并给出了从 3SAT 到 Why-Provenance[Q] 的归约。现在我们来讨论上界所依据的关键要素。核心性质是：每当存在一棵证明树 T，能够证明给定输入数据库的子集属于 why-来源时，总能找到一种方法，将 T 紧凑地表示为一个多项式大小的有向无环图。由此便可得到一种简单的猜测-检验算法，其运行时间为多项式时间。接下来，我们将进一步阐述上述关键性质。

证明有向无环图。 我们首先介绍事实的证明有向无环图（DAG）的概念，它本质上是一个通用的
证明树概念的推广。回顾一下，若有向无环图 G 恰有一个没有入边的节点，则称其为有根，该节点即为根节点。若 G 的某个节点没有出边，则称其为叶节点。

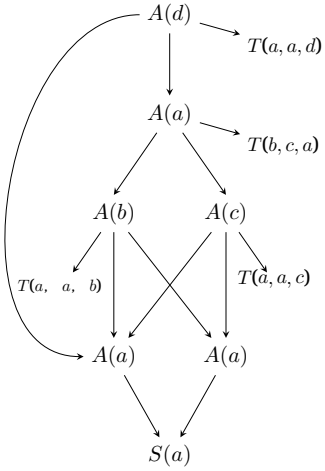
定义4 (证明有向无环图)。 考虑一个 *Datalog* 程序Σ、一个定义在扩展数据库(Σ)上的数据库 D，以及一个定义在模式(Σ)上的事实 α。α 相对于 D 和 Σ 的证明有向无环图是一个有限的带标签的有根有向无环图 $G = (V, E, \lambda)$，其中 λ： $V \to \mathsf{base}(D, \Sigma)$，满足：
1. 如果 $v \in V$ 是根节点，那么 $\lambda(v) = \alpha$。*2.* 如果 $v \in V$ 是叶节点，那么 $\lambda(v) \in D$。*3.* 如果 $v \in V$ 有 $n \geq 1$ 条出边 $(v,\ u_1)$ ..... $(v,\ u_n)$，那么存在一条规则 $R_0(\bar{x}_0)$: – $R_1(\bar{x}_1)$.....，$R_n(\bar{x}_n) \in \Sigma$以及一个函数 h： $\bigcup_{i\in[n]} \bar{x}_i \to C$，使得 $\lambda(v) = R_0(h(\bar{x}_0))$，并且对于 $i \in [n]$，$\lambda(u_i) = R_i(h(\bar{x}_i))$。 ■

证明树和证明有向无环图之间的关键区别在于，证明有向无环图可能会重复利用节点，以紧凑地表示证明树。以下示例展示了这一点。

示例3。 令 $Q = (\Sigma, A)$，其中 Σ 是例 *1* 中给出的程序，令 D 为例 *1* 中的数据库。事实 $A(d)$ 相对于 D 和 Σ 的一个简单证明有向无环图为



它以紧凑形式表示了例 *1* 中给出的第一棵证明树。下面是事实 $A(d)$ 相对于 D 和 Σ 的另一个稍复杂一些的证明有向无环图：



它显然表示了例 *1* 中的第二棵证明树。 ■

证明树的紧凑表示。 给定一个证明有向无环图 G（它是某个 Datalog 程序相对于某个数据库对某个事实所作的证明），我们将其支撑集定义为 support(G)，即为 G 的叶节点标记的事实集合。关键结果如下：

命题 5。 对于一个 *Datalog* 程序 Σ，存在一个多项式 f，使得对于每个定义在 扩展数据库(Σ) 上的数据库 D、定义在模式(Σ)上的事实 α，以及 $D' \subseteq D$，以下各项等价：
1. 存在一个关于 D 和 Σ 的 α 的证明树 T，满足支撑集(T) $= D'$。*2.* 存在一个关于 D 和Σ 的 α 的证明有向无环图 $G = (V, E, \lambda)$，满足 支撑集(G) $= D'$ 且 $|V| \leq f(|D|)$。

容易证明，通过将关于 D 和 Σ 的 α 的证明有向无环图 G“展开”为关于 D 和 Σ 的 α 的证明树 T，并使 support(T) = support(G)，即可由 (2) 推出 (1)。现在，由 (1) 推出 (2) 的方向则相当复杂，需要仔细构造，将关于 D 和 Σ 的 α 的证明树 T 转换为关于 D 和 Σ 的 α 的紧凑证明有向无环图 G，使得 support(T) = support(G)。该构造分为三个主要步骤，分别由引理6、7和8给出。

• 第一步是证明，具有支撑集 support(T) = D' 的、关于 D 和 Σ 的 T 证明树可以转换为一棵关于 D 和 Σ 的 T' 证明树，且其支撑集为 support(T') = D'，并具有“较小”的深度。回顾一下，有根树T的深度（记作 depth(T)）是从根节点到叶节点的最长路径长度。相应的引理如下：

引理 6。对于每个 *Datalog* 程序 Σ，都存在一个多项式 f，使得对于任意一个定义在 扩展数据库(Σ) 上的数据库 D、定义在 模式(Σ) 上的事实 α，以及 $D' \subseteq D$，如果存在一棵关于 D 和 Σ 的 T 证明树，且其支撑集为 支撑集(T) = D'，那么也存在这样一棵证明树 T'，其深度 (T') $\leq f(|D|)$。

• 第二步 是证明：对于深度“较小”的证明树 T，如果它是关于 D 和 Σ、支撑集为 support(T) = D' 的，那么可以将其转换为一棵子树数量“较小”的证明树 T'，它是关于 D 和 Σ、支撑集为 support(T') = D' 的。粗略地说，证明树 T 的子树数量是指：在 T 中，以标签相同的节点为根的子树中，彼此不同（就节点标签而言）的子树的最大数量。下面将这一概念形式化。

若存在一个双射 $h: V \to V'$，使得对于每个节点 $v \in V$，$\lambda(v) = \lambda'(h(v))$，并且对于任意两个节点 u、$v \in V$，$(u, v) \in E$ 当且仅当 $(h(u)\ h(v)) \in E'$，则称两棵有根树 $T = (V, E, \lambda)$ 和 $T' = (V', E', \lambda')$ 同构，记作 $T \approx T'$。显然，$\approx$ 是所有有根树集合上的一个等价关系。对于事实 α，我们进一步用 $T[\alpha]$ 表示 T 中根节点标签为 α 的所有子树的集合，即 $T[\alpha] = \{T[v] \mid v \in V$，且 $\lambda(v) = \alpha\}$，其中 $T[v]$ 是以 v 为根节点的 T 的子树。令 $T[\alpha]_{/\approx}$ 为 $T[\alpha]$ 关于 $\approx$ 的商集，也就是 $T[\alpha]$ 关于 $\approx$ 的所有等价类构成的集合。换言之，$T[\alpha]_{/\approx}$ 的每个成员都是 $T[\alpha]$ 中一组标签完全相同的树所构成的极大集合。于是，T 的子树数量记作 scount(T)，其定义为 $\max_{\alpha \in \{\lambda(v) \mid v \in V\}} \{|T[\alpha]_{/\approx}|\}$。关键引理如下：

引理 7. 对于每个 *Datalog* 程序 Σ 和一个多项式 f，都存在一个多项式 g，使得对于扩展数据库 *edb*(Σ) 上的每个数据库 D、模式 *sch*(Σ) 上的事实 α 以及 $D' \subseteq D$，如果存在一个关于 D 和 Σ 的 α 的证明树 T，满足支撑集(T) $= D'$ 且深度(T) $\leq f(|D|)$，那么也存在这样一棵证明树 T'，满足子树数量(T') $\leq g(|D|)$。

• 第三步 表明，可以将一棵关于 D 和 Σ 的 α 的证明树 T（其支撑集(T) = D'，且子树数量“较小”）转换为一个关于 D 和 Σ 的 α 的紧凑证明有向无环图 G（其支撑集(G) = D'）。以下是相应的引理：

引理 8. 对于每个 *Datalog* 程序 Σ 和多项式 f，都存在一个多项式 g，使得对于每个定义在扩展数据库(Σ)上的数据库 D、事实 α以及 $D' \subseteq D$，如果存在一棵关于 D和 Σ的 α的证明树 T，且其支撑集(T) = D'，并且 *scount*(T) $\leq f(|D|)$，那么就存在一张关于 D 和 Σ 的 α的证明有向无环图 $G = (V, E, \lambda)$，其支撑集(G) = D'，并且 $|V| \leq g(|D|)$。

现在很清楚，命题 5 中方向 (1) 蕴含 (2) 是引理 6、7 和 8 的直接推论。

4.2 非递归查询

现在我们将重点讨论非递归 Datalog 查询，并展示以下关于 why-来源的数据复杂度的结论：

定理 9. Why-来源[NRDat] 在数据复杂度下属于 AC_0 。

上述结果通过一阶可改写性得出。也就是说，给定一个非递归 Datalog 查询 $Q = (\Sigma, R)$，我们构造一个一阶查询 Q_{FO}，使得对于 Why-来源[Q]的每个输入实例——即一个定义在 edb(Σ) 上的数据库 D、一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，以及 D 的一个子集 D' ——事实 D' 属于 why($\bar{t}$, D, Q)，等价于 $\bar{t}$ 是查询 Q_{FO} 在 D' 上的一个答案。由于一阶查询求值在数据复杂度下属于 AC_0 （Vardi 1995），定理 9 得证。在深入讨论细节之前，我们先回顾一下一阶查询的基础知识。

一阶查询。一阶（*FO*）查询 Q 是形如 $\varphi(\bar{x})$ 的表达式，其中 φ 是一个一阶公式，$\bar{x}$ 是一个由（不一定互不相同的）变量组成的元组，而出现在 $\bar{x}$ 中的变量集合恰好是 φ的自由变量集合。数据库 D 上 Q 的答案 是元组集合 $Q(D) = \{\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|} \mid D \models \varphi[\bar{x}/\bar{t}]\}$，其中 $|\bar{x}|$表示 $\bar{x}$的长度，$\varphi[\bar{x}/\bar{t}]$ 表示将 $\bar{x}$ 中的变量替换为 $\bar{t}$中对应常量后得到的句子，而 $\models$ 表示标准的一阶逻辑蕴涵。令 var(φ) 表示出现在 φ中的变量集合。合取查询（*CQ*）是一个一阶查询 $\varphi(\bar{x})$，其中 φ 具有如下形式：$\exists \bar{y}(R_1(\bar{x}_1) \wedge \cdots \wedge R_n(\bar{x}_n))$，且 $\bar{x} \cap \bar{y} = \emptyset$ 和 $\bar{x}_i \subseteq \bar{x} \cup \bar{y}$。

一些准备工作。为了构造所需的一阶查询，我们需要一些辅助概念。事实 α 的规范形式 记作 can(α)，是将 α 中的每个常量 c 替换为变量 $\langle c \rangle$ 后得到的原子；也就是说，变量的名称由常量 c 唯一确定。给定一个 Datalog 查询 $Q = (\Sigma, R)$，若标记根树 $T = (V, E, \lambda)$ 是关于某个扩展数据库 edb(Σ) 和 Σ 上的某个数据库 D 中某个事实 $R(\bar{t})$ 的证明树，则称其为 Q-树 。由 Q-树诱导的合取查询的概念如下：

定义 10 (**诱导的合取查询**)。考虑一个 *Datalog* 查询$Q = (\Sigma, R)$ 和一棵 Q-树 $T = (V, E, \lambda)$，其中 $v \in V$ 是根节点，且 $\lambda(v) = R(c_1, \ldots, c_n)$。由 T 诱导的合取查询 记作 cq(T)，是合取查询 $\varphi_T(\langle c_1 \rangle, \ldots, \langle c_n \rangle)$，其形式为

$$\varphi_T = \exists \bar{x} \left(\bigwedge_{\alpha \in \mathsf{support}(T)} \mathsf{can}(\alpha) \right),$$

其中，$\bar{x}$ 由所有满足 $c \in \mathsf{dom}(\text{支撑集}(T)) \setminus \{c_1, \ldots, c_n\}$ 的 $\langle c \rangle$ 组成。我们将 合取查询(Q) = {合取查询(T) | T称为一棵 Q-树}。 ■

简单来说，cq(T) 是这样得到的合取查询：取 T 规范形式中标记叶节点的事实的合取，然后对所有不在标记 T 根节点的事实的规范形式中出现的变量进行存在量化。现在，给定两个合取查询 $\varphi(\bar{x})$ 和$\psi(\bar{y})$，如果它们同构，我们记作 $\varphi(\bar{x}) \approx \psi(\bar{y})$。显然，

$\approx$ 是合取查询集合上的一个等价关系。对于一个 Dat- alog 查询 Q，$\mathsf{cq}(Q)/\approx$ 是 $\mathsf{cq}(Q)$ 关于 $\approx$ 的商集，即 $\mathsf{cq}(Q)$ 关于 $\approx$ 的所有等价类组成的集合。令 $\mathsf{cq}^{\approx}(Q)$ 为这样一个合取查询集合：从 $\mathsf{cq}(Q)/\approx$ 的每个成员中任取一个代表元。于是：

引理 11. 对于每个非递归 *Datalog* 查询 Q，都有 $\mathsf{cq}^{\approx}(Q)$是有限的。

一阶重写。 对于非递归 Datalog 查询 $Q = (\Sigma, R)$，有 $\mathsf{cq}^{\approx}(Q)$，现在我们可以继续构造所需的一阶查询 Q_{FO}。

我们首先针对一个合取查询 $\varphi(\bar{y}) \in \mathsf{cq}^{\approx}(Q)$，构造一个一阶查询 $Q_{\varphi(\bar{y})} = \psi_{\varphi(\bar{y})}(x_1, \ldots, x_{\mathsf{ar}(R)})$，其中 $x_1, \ldots, x_{\mathsf{ar}(R)}$是互不相同的变量，且不会出现在 $\mathsf{cq}^{\approx}(Q)$ 的任何合取查询中，并满足以下性质：对于每个数据库 D和元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，$\bar{t} \in Q_{\varphi(\bar{y})}(D)$ 当且仅当 $\bar{t}$ 是 $\varphi(\bar{y})$ 在 D 上的答案；此外，所有 D 的原子都用于蕴含句子 $\varphi[\bar{y}/\bar{t}]$，也就是说，D 中没有其他事实，只有那些被用作见证、对应于出现在 $\varphi[\bar{y}/\bar{t}]$ 中的原子的事实。假设 φ 的形式为 $\exists \bar{z}(R_1(\bar{w}_1) \wedge \cdots \wedge R_n(\bar{w}_n))$。带有自由变量 $x_1, \ldots, x_{\mathsf{ar}(R)}$ 的公式 $\psi_{\varphi(\bar{y})}$ 具有以下形式

$$\exists \bar{y} \exists \bar{z} \left(\varphi_1 \wedge \varphi_2 \wedge \varphi_3\right),$$

其中，每个合取项定义如下。我们用 $\bar{x}$ 表示元组 $(x_1, \ldots\ldots, x_{\mathsf{ar}(R)})$，而 $\bar{u}_P$（其中 P是谓词）表示变量元组 $(u_1, \ldots\ldots, u_{\mathsf{ar}(P)})$。此外，对于两个变量元组 $\bar{u} = (u_1, \ldots\ldots, u_k)$ 和 $\bar{v} = (v_1, \ldots\ldots, v_k)$，$(\bar{u} = \bar{v})$ 是 $\bigwedge_{i=1}^{k}(u_i = v_i)$ 的简写。公式 φ_1 为

$$\bigwedge_{i \in [n]} R_i(\bar{w}_i) \wedge (\bar{x} = \bar{y}) \wedge \bigwedge_{\substack{u,v \in \mathsf{var}(\varphi), \\ u \neq v}} \neg(u = v)$$

这表示，φ 中的每个原子都应通过为 φ 中的不同变量赋予不同的值来满足。公式 φ_2 定义为

$$\bigwedge_{P \in \{R_1, \ldots, R_n\}} \neg \left(\exists \bar{u}_P \left(P(\bar{u}_P) \wedge \bigwedge_{\substack{i \in [n], \\ R_i = P}} \neg(\bar{w}_i = \bar{u}_P) \right) \right)$$

这实质上表示，对于 φ 中出现的每个谓词 P，底层数据库中谓词为 P 的原子仅限于那些用作 φ 中原子的见证的原子。最后，公式 φ_3 定义为

$$\bigwedge_{P \in \mathsf{edb}(\Sigma) \setminus \{R_1, \ldots, R_n\}} \neg \left(\exists \bar{u}_P \, P(\bar{u}_P)\right)$$

这表示底层数据库中不存在谓词未出现在 φ中的原子。

有了针对每个合取查询 $\varphi(\bar{y}) \in \mathsf{cq}^{\approx}(Q)$ 的一阶查询 $Q_{\varphi(\bar{y})}$，显然，所需的一阶查询 Q_{FO} 定义为 $\Phi(x_1 \ldots, x_{\mathsf{ar}(R)})$，其中 $\Phi = \bigvee_{\varphi(\bar{y}) \in \mathsf{cq}^{\approx}(Q)} \psi_{\varphi(\bar{y})}$，并且可以得到下一个技术结果：

引理 12。 给定一个非递归 *Datalog* 查询 $Q = (\Sigma, R)$、一个定义在 扩展数据库(Σ) 上的数据库 D 、 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，以及$D' \subseteq D$，有 $D' \in \mathsf{why}(\bar{t}, D, Q$ ）当且仅当 $\bar{t} \in Q_{FO}(D')$。

4.3 精化证明树

why-来源的标准概念依赖于不受任何限制的任意证明树。然而，正如文献中已有讨论（例如，参见近期研究 Bourgaux 等人，2022），有些证明树会给出反直觉解释。例如，例 1 中的第二棵证明树就是如此，因为事实 $A(a)$ 是从其自身推导出来的。现在，由这种不自然的证明树见证的 $\mathsf{why}(\bar{t}, D, Q)$ 中的一个成员 D'，可能被视为对 $_t$ 的反直觉解释，因为它无法对应于一种直观的推导过程——这种过程可以从证明树中提取，并从 D' 导出事实 $R(\bar{t})$。因此，我们需要考虑经过细化的证明树类别，以克服任意证明树在概念上的局限性。文献中考虑的两种依据充分的概念是非递归证明树和最小深度证明树（Bourgaux 等人，2022）。粗略地说，非递归证明树是不包含两个标记相同事实且其中一个是另一个后代的节点的证明树；而最小深度证明树则是在某个元组的所有证明树中深度最小的证明树。我们仅针对这些细化类别中的证明树分析了 why-来源的数据复杂度，并证明其保持不变。由于篇幅限制，本文省略了相关细节，详情见扩展版。

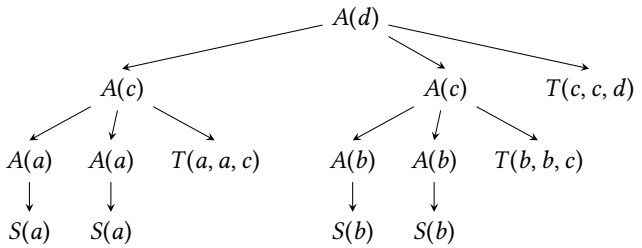
5 无歧义证明树

尽管非递归证明树和最小深度证明树是值得关注的核心类别，但这些类别中仍有一些证明树会被归为反直觉的情况。更确切地说，我们可以构造出既非递归又具有最小深度的证明树，但它们在某些事实的推导方式上存在歧义。

例 4. 令 $Q = (\Sigma, A)$，其中 Σ 是 *Datalog* 程序，它按照例 *1* 中的方式对路径可达性问题进行编码。另考虑数据库

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

以下是事实 $A(d)$ 相对于 D 和Σ 的一棵证明树，它既是非递归的，也是最小深度的，但存在上文提到的歧义问题：



事实上，有两个节点都标记着事实 $A(c)$，但它们的子树不同，因此，$A(c)$ 是如何推导出来的并不明确。因此，相对于非递归的最小深度证明树，数据库 D属于 D 和 Q 的 ***why-provenance***；由于上述证明树，它可能被归类为反直觉解释，因为它并不符合一种直观的推导过程——在这种过程中，每个事实都因一个明确的原因而只被推导一次。

■

上述讨论引出了一个新类别：无歧义证明树。在这类树中，同一事实的所有出现都必须通过相同的推导来证明。

定义 *13*（无歧义证明树）。考虑一个 *Datalog* 程序 Σ、一个定义在扩展数据库 Σ 上的数据库 D，以及一个定义在模式 Σ 上的事实 α。相对于 D 和 Σ，α 的一棵无歧义证明树是 α 相对于 D 和 Σ 的一棵证明树 $T = (V, E, \lambda)$，并且满足：对于所有 v，$u \in V(v) = \lambda(u)$ 蕴含 $T[v] \approx T[u]$。 ■

再看例 4，我们只需将 $A(d)$ 的第二个子节点的子树替换为其第一个子节点的子树（反之亦然），即可构造一棵相对于 D 和 Σ 的 $A(d)$ 无歧义证明树。现在，相对于无歧义证明树的 why-来源 可按预期定义：对于 Datalog 查询 $Q = (\Sigma, R)$、定义在扩展数据库 Σ 上的数据库 D，以及一个属于 D 上 $\mathsf{dom}(D)$ 的元组 $\bar{t} \in \mathsf{ar}(R)$，$\bar{t}$ 相对于 D和 Q相对于无歧义证明树 的 why-来源 是以下族

$$\{\mathsf{support}(T) \mid T \text{ is an unambiguous proof tree of } R(\bar{t}) \text{ w.r.t. } D \text{ and } \Sigma\}$$

记作 $\mathsf{why}^{\mathsf{UN}}(\bar{t}, D, Q)$。再次考虑例 4，$\mathsf{why}^{\mathsf{UN}}((d)\ D, Q)$由 $\{S(a)\ T(a,a,c)\ T(c,c,d)\}$和 $\{S(b)\ T(b,b,c)\ T(c,c,d)\}$组成。这正是人们对元组$(d)$在概念上直观的解释，而不是对整个数据库 D的解释。算法问题

$$\mathsf{Why\text{-}Provenance}_{\mathsf{UN}}[\mathsf{C}] \quad \text{and} \quad \mathsf{Why\text{-}Provenance}_{\mathsf{UN}}[Q]$$

以预期的方式定义。我们可以证明，why-来源的数据复杂度保持不变。

定理 14。以下结论成立：

1. 对每个类$\mathsf{C} \in \{\mathsf{Dat}, \mathsf{LDat}\}$，Why-来源$_{\mathsf{UN}}[\mathsf{C}]$ 在数据复杂度意义下是NP完全的。*2.* Why-来源$_{\mathsf{UN}}[\mathsf{NRDat}]$在数据复杂度意义下属于 AC_0。

对于第（1）项，我们证明 $\mathsf{Why\text{-}Provenance}_{\mathsf{UN}}[\mathsf{Dat}]$ 属于 NP，而 $\mathsf{Why\text{-}Provenance}_{\mathsf{NR}}[\mathsf{LDat}]$ 是 NP困难。后者通过从判定一个有向图是否存在哈密顿回路的问题归约得到。NP 上界依赖于这样一个刻画：事实 α 相对于数据库 D和 Datalog 程序 Σ 存在满足 $\mathsf{support}(T) = D' \subseteq D$ 的无歧义证明树，当且仅当存在一个多项式大小的所谓无歧义证明有向无环图 G，它相对于 D 和 Σ 满足 $\mathsf{support}(G) = D'$，且大小为多项式。值得注意的是，与任意证明树不同，我们可以直接从无歧义证明树 T 得到一个具有与 T相同支撑集的多项式大小无歧义证明有向无环图，而无需采取任何中间步骤来降低 T的深度或子树数量。这是因为根据定义，无歧义证明树的深度和子树数量都很“小”（事实上，子树数量为一）。第（2）项中的 AC_0 上界通过 FO 可重写性得到证明。目标一阶查询的构造方式与定理 9 的证明相同，但在 $\mathsf{cq}(Q)$ 的定义中只考虑无歧义证明树。

5.1 通过 SAT 求解器计算 why-来源

接下来，我们将讨论如何利用现成的 SAT 求解器，高效计算相对于无歧义证明树的元组的 why-来源。然后，我们将讨论一种概念验证实现，并报告初步实验评估中令人鼓舞的结果。需要强调的是，聚焦于无歧义证明树对取得这些令人鼓舞的结果至关重要，因为目前尚不清楚如何让基于 SAT 的实现适用于非无歧义证明树。这主要是因为，与其他类别的证明树不同，无歧义证明树的子树数量始终为一，这对于控制布尔公式的规模至关重要。

考虑一个 Datalog 查询 $Q = (\Sigma, R)$、一个扩展数据库 D over $\mathsf{edb}(\Sigma)$，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$。我们可以在关于 D 的多项式时间内构造一个布尔公式 $\phi_{(\bar{t},D,Q)}$，使得可以根据令 $\phi_{(\bar{t},D,Q)}$ 为真的真值赋值，计算 t 相对于 D 和 Q、针对无歧义证明树的 why-来源。这依赖于前文提到的如下刻画：存在一个无歧义证明树 $R(\bar{t})$，其相对于 D 和 Σ 的支撑集为 $\mathsf{support}(T) = D' \subseteq D$，当且仅当存在一个无歧义证明有向无环图 G，其 $R(\bar{t})$ 相对于 D和 Σ 的支撑集为 $\mathsf{support}(G) = D'$。公式 $\phi_{(\bar{t},D,Q)}$ 的形式为 $\phi_{graph} \wedge \phi_{acyclic} \wedge \phi_{root} \wedge \phi_{proof}$，其中 ϕ_{graph}用于验证真值赋值是否对应一个语法正确的带标记有向图 G，$\phi_{acyclic}$ 用于验证 G 是否无环，ϕ_{root} 用于验证 $R(\bar{t})$ 是否为 G 的唯一根节点，而ϕ_{proof} 用于验证 G 是否为无歧义证明有向无环图。

构造 $\phi_{(\bar{t},D,Q)}$ 的关键要素是所谓的的向下闭包（$\bar{t}$）相对于 D 和 Σ，该概念取自（Elhalawati、Kr¨otzsch 和 Mennicke，2022）。直观地说，它是一个超图，编码了 $R(\bar{t})$ 相对于 D 和 Σ的所有可能证明有向无环图。我们首先构造这个超图 H，这可以在关于 D大小的多项式时间内完成；然后在 H 的指引下构造公式 $\phi_{(\bar{t},D,Q)}$，该公式本质上是在超图 H中搜索一个无歧义的证明有向无环图。现在，对 $\phi_{(\bar{t},D,Q)}$ 的变量所作的真值赋值 τ 自然会产生一个数据库，记为 $\mathsf{db}(\tau)$。令 $[\![\phi_{(\bar{t},D,Q)}]\!]$ 为这一族

$$\left\{\mathsf{db}(\tau) \mid \tau \text{ is a satisfying assignment of } \phi_{(\bar{t},D,Q)}\right\}.$$

接下来，我们可以证明如下技术结果：

命题 15. 考虑一个 *Datalog* 查询 $Q = (\Sigma, R)$、一个定义在扩展数据库(Σ)上的数据库 D，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$。则有 $\mathsf{why}^{\mathsf{UN}}(\bar{t}, D, Q) = [\![\phi_{(\bar{t},D,Q)}]\!]$。

上述命题提供了一种借助现成 SAT 求解器，针对无歧义证明树计算元组 why-来源的方法。但是，这套机制在实际应用中表现如何？具体而言，我们感兴趣的是通过枚举 why-来源的成员来增量计算 why-来源，而不是一次性计算整个集合。本节余下部分将围绕这个问题给出初步解答。

5.2 一些实现细节

在介绍实验结果之前，我们先简要讨论实现中的一些有趣之处。下文中，固定一个 Datalog 查询 $Q = (\Sigma, R)$、一个定义在 $\mathsf{edb}(\Sigma)$上的数据库D，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$。

构造向下闭包。请注意，$\phi_{(\bar{t},D,Q)}$ 的构造依赖于 $R(\bar{t})$ 的向下闭包

关于 D 和 Σ。事实证明，可以通过执行稍作修改的 Datalog 查询 $Q_{\downarrow}$（在稍作修改的数据库 $D_{\downarrow}$上）来计算向下闭包的超边。换言之，在 $D_{\downarrow}$ 上执行 $Q_{\downarrow}$ 所得的答案与向下闭包的超边一致。因此，为了构造向下闭包，我们借助最先进的 Datalog 引擎，即 DLV 2.1.1 版（Adrian 等，2018）。请注意，我们基于执行 Datalog 查询的方法不同于（Elhalawati、Kr¨otzsch 和 Mennicke，2022）中的方法，后者使用了带集合项扩展的 Datalog。

构造公式。回想一下，$\phi_{(\bar{t},D,Q)}$ 由四个合取项组成，每个合取项负责一项特定任务。正如预期的那样，最繁重的任务是验证相关图是否无环（由公式 $\phi_{acyclic}$ 完成）。通过布尔公式检查有向图的无环性，是 SAT 文献中一个研究充分的问题。为此，我们采用顶点消去技术（Rankooh 和 Rintanen，2022）。这种方法的优点是，编码 $\phi_{acyclic}$ 所需的布尔变量数量为 $O(n \cdot \delta)$ 量级，其中 n 是图的节点数，而 δ 是图的所谓消去宽度，直观来说，它与图的连通程度有关。

增量构造 why-来源。回想一下，我们关注的是 why-来源的增量计算；与一次性计算整个集合相比，这种做法在实践中更有用。为此，我们需要一种方法，在不重复的情况下枚举 why-来源的所有成员。我们通过改造 SAT 文献中一种用于枚举布尔公式满足赋值的标准技术来实现这一点，该技术称为阻塞子句。首先，我们将 S 中所有出现在 $R(\bar{t})$ 相对于 D 和 Σ 的向下闭包中的 D 事实收集到一个集合中。然后，在请求 SAT 求解器给出 $\phi_{(\bar{t},D,Q)}$ 的任意满足赋值 τ 后，我们输出数据库 db(τ)，再构造“阻塞”子句 $\vee_{\alpha \in S} \ell_\alpha$，其中，若 $\alpha \in$ db(τ)，则 $\ell_\alpha = \neg x_\alpha$；否则为 $\ell_\alpha = x_\alpha$。随后，我们将该子句加入公式中，以表示其他满足赋值 τ' 不应产生 why-来源中的同一成员。这样就会从计算中排除先前计算出的解释。每当得到 why-来源的一个新成员时，我们就继续添加这样的阻塞子句，直到公式不可满足为止。

5.3 实验评估

现在，我们将通过实验评估上述基于 SAT 的方法。为此，我们考虑文献中的多种情景，每种情景由一个 Datalog 查询 $Q = (\Sigma, R)$和一组基于扩展数据库(Σ)的数据库 $\mathcal{D}$ 组成。

实验情景。 所有考虑的情景汇总于表 1。简要介绍如下：

传递闭包。 此情景计算图的传递闭包，并查询连通节点。数据库D_{bitcoin} 存储了比特币网络的一部分（Weber等人（2019）），而 D_{facebook} 存储了来自 Facebook 的不同“社交圈”（McAuley和Leskovec（2012））。

Doctors。 情景 Doctors-i（$i \in [7]$）曾用于（Elhalawati、Kr¨otzsch 和 Mennicke，2022），代表从一个知名的数据交换基准中获得的查询，该基准涉及存在规则（存在变量已替换为新常量）。所有此类情景共用一个包含 100K 条事实的数据库。

Galen。 此情景用于（Elhalawati、Kr¨otzsch 和 Mennicke，2022），实现了 ELK 演算（Kazakov、Kr¨otzsch 和 Simancik，2014），并查询所有通过 subClassOf关系关联的概念对。各数据库包含 Galen本体（The Oxford Library（2007））的不同部分。

Andersen。此情景用于（Fan、Mallireddy 和 Koutris，2022），实现了经典的 Andersen“指向”算法，用于确定过程式程序中的数据流，并查询所有满足 p 指向 v的指针 p 与变量 v 对。各数据库是对不同长度的程序语句进行编码后得到的。

CSDA. 此情景（上下文敏感数据流分析）取自（Fan、Mallireddy 和 Koutris（2022）），与 Andersen 类似，但它要查找程序中的空引用。数据库 D_{httpd}、$D_{\mathsf{postgresql}}$ 和 D_{linux} 分别存储了 httpd Web服务器、PostgreSQL数据库管理系统和 Linux内核的语句。

实验设置。对于每个由查询 $Q = (\Sigma, R)$ 和数据库族 $\mathcal{D}$ 构成的情景 s，以及每个 $D \in \mathcal{D}$，我们使用 DLV 计算了 $Q(D)$，然后从 $Q(D)$ 中均匀随机抽取五个元组 $\bar{t}^1_{s,D}$、.....、$\bar{t}^5_{s,D}$。接着，对于每个 $i \in [5]$，我们首先通过 Python 3 实现计算适配查询 $Q_{\downarrow}$ 和数据库 $D_{\downarrow}$，然后使用 DLV 实际计算 $R(\bar{t}^i_{s,D})$ 关于 D 和 Σ 的向下闭包；随后，通过 C++ 实现构造布尔公式 $\phi_{(\bar{t}^i_{s,D},D,Q)}$；最后，我们以该公式为输入，运行最先进的 SAT 求解器 Glucose （参见例如（ Audemard 和 Simon，2018））4.2.1 版，枚举 why$_{\mathsf{UN}}(\bar{t}^i_{s,D}, D, Q)$ 的成员。所有实验均在一台笔记本电脑上进行，配置为 Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 和 32GB 随机存取存储器，运行 Fedora Linux 37。Python 代码使用 Python 3.11.2 执行，C++代码使用 g++ 12.2.1 编译，并启用了 -O3优化标志。

实验结果。 由于篇幅限制，我们仅展示基于 Andersen 情景的结果。不过，最终结果与我们基于所有其他情景观察到的结果一致。

关于向下闭包和布尔公式的构造，我们在图 1 中报告了 Andersen 情景中每个数据库的总运行时间（请注意，这里有五个规模各异的数据库，因此我们绘制了五个图）。此外，每个图包含五个条形，分别对应随机选取的五个元组。每个条形表示构建向下闭包所需的时间加上构造布尔公式所需的时间。我们观察到，几乎所有时间都花在计算向下闭包上，而构建公式所需的时间可以忽略不计。因此，我们的工作重点应放在

情景	数据库	查询类型	规则数量
传递闭包	D_{bitcoin} (235K), D_{facebook} (88.2K)	线性，递归	2
Doctors-i, $i \in [7]$	D_1 (100K)	线性，非递归	6
Galen	D_1 (26.5K), D_2 (30.5K), D_3 (67K), D_4 (82K)	非线性，递归	14
Andersen	D_1 (68K), D_2 (340K), D_3 (680K), D_4 (3.4M), D_5 (6.8M)	非线性，递归	4
CSDA	D_{httpd} (10M), $D_{\text{postgresql}}$ (34.8M), D_{linux} (44M)	线性，递归	2

表 1：实验情景。

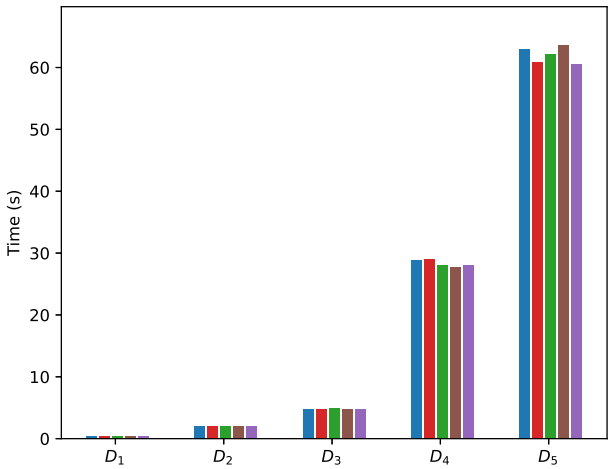


图 1：构建向下闭包和布尔公式。

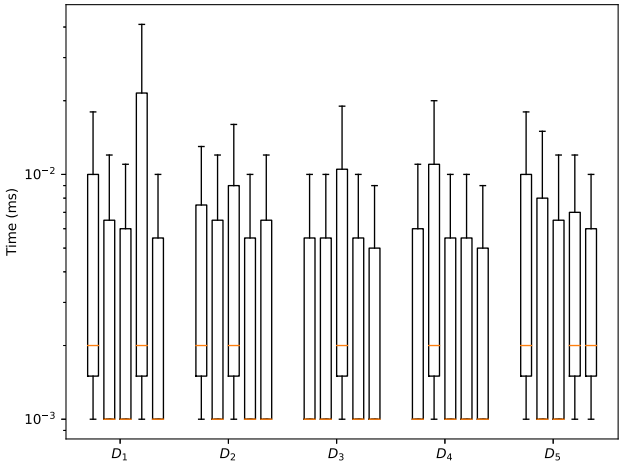


图 2：why-来源的增量计算。

改进向下闭包的计算上。此外，对于规模适中的数据库（包含 68K、340K 和 680K 个事实），总时间约为几秒，这十分令人鼓舞。至于我们考察的超大型数据库（包含 3.4M 和 6.8M 个事实），总时间介于半分钟和一分钟之间；考虑到查询的复杂度、数据库的庞大规模以及我们机器的有限性能，这同样令人鼓舞。

对于 why-来源的增量计算，我们在图 2 中给出了 Andersen 情景下每个数据库构建解释所需的时间，也就是当前 why-来源成员与下一个成员之间的时间（这一时间也称为延迟）。五个图分别汇总了为五个随机选取的元组构建 why-来源成员时的延迟（最多构建 10K 个成员，或超时 5 分钟）。我们使用箱线图，其中箱体的下边界和上边界分别表示第一和第三四分位数，即分别有 25% 和 75% 的延迟不超过该值；橙色线表示延迟中位数。此外，下须线和上须线分别表示最小和最大延迟。所有时间均以毫秒为单位，并采用对数刻度。可以看到，大多数延迟都低于 1 毫秒，中位数在微秒量级。因此，一旦布尔公式就绪，增量计算 why-来源的成员就会非常快。

6 结论

我们的工作的核心结论是：对于递归查询，why-来源问题通常是难处理的；而对于非递归查询，其数据复杂度则具有很强的易处理性。为了克服任意证明树在概念上的局限，我们考察了新的无歧义证明树类别，并证明它不会影响 why-来源问题的数据复杂度。有趣的是，我们通过实验确认，无歧义证明树有助于利用现成的 SAT 求解器，高效地计算 why-来源。请注意，我们通过聚焦于两种方法都能处理的设定，与（Elhalawati、Kr¨otzsch 和 Mennicke，2022）进行了初步比较。具体而言，我们使用了 Doctors-i、for $i \in [7]$ 情景，并测量了我们方法的端到端运行时间（而非延迟）。对于简单情景，两种方法的表现相当，运行时间在秒级。对于要求较高的情景（Doctors-$i$for $i \in \{1, 5, 7\}$），我们的方法通常更快。

若能以二分结果的形式，完整分类 why-来源问题的数据复杂度，将极具价值。这也有助于进一步揭示该问题的联合复杂度，即 Datalog 查询属于输入的一部分。最后，为了更好地了解我们的基于 SAT 的机制能否在实践中应用，进行更全面的实验评估至关重要。

参考文献

Abiteboul, S.; Hull, R.; and Vianu, V. 1995年。《数据库基础》。Addison-Wesley. Adrian, W. T.; Alviano, M.; Calimeri, F.; Cuteri, B.; Do- daro, C.; Faber, W.; Fusc`a, D.; Leone, N.; Manna, M.; Perri, S.; Ricca, F.; Veltri, P.; and Zangari, J. 2018. DLV 答案集编程系统：进展与应用。《K》¨《ünstliche Intell.》32(2-3):177–179. Audemard, G., and Simon, L. 2018. 关于 Glucose SAT 求解器。《Int. J. Artif. Intell. Tools》27(1):1840001:1–1840001:25. Benedikt, M.; Buron, M.; Germano, S.; Kappelmann, K.; and Motik, B. 2022. 重写无限追逐。《PVLDB》15(11):3045–3057. Bourgaux, C.; Bourhis, P.; Peterfreund, L.; and Thomazo, M. 2022. 重新审视 Datalog 的半环溯源。载于《KR》。 Buneman, P.; Khanna, S.; and Tan, W. C. 2001. 为什么以及在哪里：数据溯源的刻画。载于《ICDT》，316–330。 Cook, S. A. 1974. 关于时间-存储权衡的一个观察。《J. Comput. Syst. Sci.》9(3):308–316. Dam´asio, C. V.; Analyti, A.; and Antoniou, G. 2013. 逻辑编程的论证。载于《LPNMR》，530–542。 Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. 逻辑编程的复杂度与表达能力。《ACM Comput. Surv.》33(3):374–425. Deutch, D.; Milo, T.; Roy, S.; and Tannen, V. 2014. Datalog 溯源电路。载于《ICDT》，201–212。 Eiter, T.; Ortiz, M.; Simkus, M.; Tran, T.; and Xiao, G. 2012. Horn-SHIQ 加规则的查询重写。载于《AAAI》。 Elhalawati, A.; Kr¨otzsch, M.; and Mennicke, S. 2022. 用于按需计算 Datalog why-来源的存在规则框架。载于《RuleML》+《RR》。 Esparza, J.; Luttenberger, M.; and Schlund, M. 2014. Fp-solve：一种用于半环上不动点方程的通用求解器。载于《CIAA》，1–15。 Fan, Z.; Malliredy, S.; and Koutris, P. 2022. 深入理解 Datalog 系统的性能与设计。载于《Datalog 2.0》，166–180。 Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. 使用 clingo 5 轻松进行理论求解。载于《ICLP》，2:1–2:15。 Kazakov, Y.; Kr¨otzsch, M.; and Simancik, F. 2014. 非凡的 ELK—— 从多项式过程到使用 EL 本体进行高效推理。《J. Autom. Reason.》53(1):1–61。 Lee, S.; Lud¨ascher, B.; and Glavic, B. 2019. PUG：一个用于 why 和 why-not 溯源的框架及其实用实现。《VLDB J.》28(1):47–71。 Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006年。用于知识表示与推理的 DLV 系统。《ACM Trans. Com- put. Log.》7(3):499–562。

Leone, N.; Allocca, C.; Alviano, M.; Calimeri, F.; Civili, C.; Costabile, R.; Fiorentino, A.; Fusc`a, D.; Germano, S.; Laboccetta, G.; Cuteri, B.; Manna, M.; Perri, S.; Reale, K.; Ricca, F.; Veltri, P.; and Zangari, J. 2019年. 增强 DLV，以支持大规模推理。载于*LPNMR*，312–325。McAuley, J., and Leskovec, J. 2012年. 学习在自我中心网络中发现社交圈。载于*NIPS*，539–547。Rankooh, M. F., and Rintanen, J. 2022. 利用顶点消除法对无环性和可达性进行命题编码。载于*AAAI*，5861–5868。牛津图书馆。2007年. Galen本体。Vardi, M. Y. 1995年. 有界变量查询的复杂度。载于*PODS*，266–276。Weber, M.; Domeniconi, G.; Chen, J.; Weidele, D. K. I.; Bellei, C.; Robinson, T.; and Leiserson, C. E. 2019年. 比特币反洗钱：利用图卷积网络开展金融取证的实验。*CoRR*abs/1908.02591。Zhao, D.; Subotic, P.; and Scholz, B. 2020年. 调试大规模 Datalog：一种可扩展的溯源评估策略。*ACM Trans. Program.Lang. Syst.* 42(2):7:1–7:35。

why-来源的数据复杂度

本节中，我们将补充第 4 节缺失的细节。

A.1 递归查询

下面给出定理 3 的完整证明，为方便起见，先在此重述：

定理 3. Why-来源[C] 在数据复杂度意义下是 NP-完全的，其中每个类别 C $\in$ {Dat,LDat} 均如此。

要证明上述结果，只需证明：

- Dat 的 why-来源[在数据复杂度意义下属于 NP。]
- LDat 的 why-来源[在数据复杂度意义下是 NP困难的。

上界

我们的主要任务是证明命题 5（我们在下文重述），这将使我们能够设计一种猜测并检验过程，其运行时间关于数据库大小为多项式时间。

命题 5。 对于 *Datalog* 程序 Σ，存在一个多项式 f，使得对于每个定义在 扩展数据库(Σ) 上的数据库 D、定义在模式(Σ) 上的事实 α，以及 $D' \subseteq D$，以下两者等价：

1. 存在一个关于 D 和 Σ 的 α 证明树 T，使得支撑集(T) $= D'$。 *2.* 存在一个关于 D 和Σ 的 α 证明有向无环图 $G = (V, E, \lambda)$，使得支撑集(G) $= D'$，且 $|V| \leq f(|D|)$。

通过将关于 D 和 Σ 的 α 证明有向无环图 G “展开”为关于 D 和 Σ 的 α 证明树 T，并满足 support(T) $=$ support(G)，即可证明方向 (2) 蕴含 (1)。更准确地说，我们从根节点开始，到叶节点为止，使用广度优先搜索遍历 G 的节点。每当遇到一个有 k 条入边的节点 v 时，就创建其子有向无环图的 k 个副本。节点 v 的子有向无环图包含其自身 v 以及从 v 可达的每个节点；若 G 中存在边 (w_1, w_2)，则其中还包含一条边 (u_1, u_2)，其中 u_1 是 w_1 的一个副本， u_2 是 w_2 的一个副本。注意，这些副本保留了节点的标签。接着，我们将 v 的每条入边替换为一条指向其子有向无环图的某个独立副本根节点的边。注意，由于 G 是无环的，对 v 执行上述操作不会影响在 v 之前已经处理过的节点。显然，所得结果是一个树 T，其根节点 v 的标签与 G 的根节点标签相同，叶节点的标签也与 G 的叶节点标签相同（因此，对于该树的每个叶节点 v，都有 $\lambda(v) \in D$，因为证明有向无环图叶节点的标签也满足同样的性质）。此外，容易验证定义 1 的性质 (3) 成立，因为 G 满足定义4 中等价的性质 (3)，并且我们的副本保留了节点的标签。因此，所得的树是关于 D 和 Σ 的 α 证明树。

关于方向（1）蕴含（2），如正文所述，证明分为三个主要步骤，分别由引理6、7和8给出，我们接下来将证明这些引理。

引理 6. 对于每个 *Datalog* 程序 Σ，都存在一个多项式 f，使得对于扩展数据库 edb(Σ)上的每个数据库 D、模式 sch(Σ)上的事实 α，以及 $D' \subseteq D$，如果存在一棵证明树 T，证明 α 关于 D 和 Σ，且其支撑集(T) $= D'$，那么也存在这样一棵证明树 T'，其深度(T') $\leq f(|D|)$。

证明。 我们对 $f(|D|) = |\text{base}(D, \Sigma)| \times |D|$ 的 $n = \text{depth}(T)$ 进行归纳，以证明该结论。

基础情形。 对于任意 $n \leq |\text{base}(D, \Sigma)| \times |D|$，结论显然成立。

归纳步骤。 我们假设对于 $n \in \{|\text{base}(D, \Sigma)| \times |D|, \ldots\ldots, p\}$，结论成立，并证明结论对于$n = p + 1$也成立。设 T 是关于 D 和 Σ 的 α 证明树，其深度为 depth(T) $= p+1$。由于 $p+1 > |\text{base}(D, \Sigma)| \times |D|$，存在一条长度为 $p+1$ 的路径 $v_1 \to v_2 \cdots \to v_{p+2}$，它位于 T中，并且存在一个标签 β，使得沿该路径的 $k > |D|$ 节点 v_{i_1}、……、v_{i_k}的标签均为 β。（注意，$|\text{base}(D, \Sigma)|$ 是 T中不同标签数量的上界。）不失一般性，我们假设 $i_1 < i_2 < \cdots < i_k$。我们将证明，对于某个满足 $j < r$的 v_{i_j} 和 v_{i_r}，有

$$\text{support}(T[v_{i_j}]) = \text{support}(T[v_{i_r}]).$$

回想一下，对于节点 v， $T[v]$ 是以 v 为根节点的 T 子树。

一个显而易见的观察是，对于满足 T_2 是 T_1 的子树的 T_1，T_2，有 support(T_2) $\subseteq$ support(T_1)。因此，对于所有 $j < r$，我们有：

$$\text{support}(T[v_{i_r}]) \subseteq \text{support}(T[v_{i_j}]).$$

现在反设，对于每个 $j < r$,

$$\text{support}(T[v_{i_r}]) \subsetneq \text{support}(T[v_{i_j}]).$$

于是我们得出结论：

$$\text{support}(T[v_{i_k}]) \subsetneq \cdots \subsetneq \text{support}(T[v_{i_1}]).$$

这意味着 support($T[v_{i_1}]$) 包含 $k > |D|$ 个互不相同的事实，进而意味着 support(T) 至少包含 $k > |D|$ 个互不相同的事实。这与对于某个 $D' \subseteq D$，support(T) $= D'$ 的事实矛盾。

因此，对于某个 $j < r$，有：

$$\text{support}(T[v_{i_r}]) = \text{support}(T[v_{i_j}]).$$

现在，我们可以缩短 T 中的路径 $v_1 \to v_2 \cdots \to v_{p+2}$，并通过将子树 $T[v_{i_j}]$ 替换为子树 $T[v_{i_r}]$，得到另一棵支撑集 D' 相同的证明树 T_1。这里一个重要的观察是， T_1 仍然是关于 D 和 Σ 的 α 的证明树。由于我们没有修改根节点 v，因此仍有 $\lambda(v) = \alpha$。此外，T_1 的叶节点集合包含于 T 的叶节点集合中；因此，对于 T_1 的每个叶节点 v，都有 $\lambda(v) \in D$。最后，由于 $T[v_{i_r}]$ 是 T 的子树，它满足定义 1 的性质 (3)（也就是说，如果 v 是一个有 $n \geq 1$ 个子节点 u_1、……、u_n 的节点，那么存在一条规则 $R_0(\bar{x}_0) :\!-\ R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n) \in \Sigma$，以及一个函数 h：$\bigcup_{i \in [n]} \bar{x}_i \to \mathsf{C}$，使得 $\lambda(v) = R_0(h(\bar{x}_0))$，并且对每个 $i \in [n]$ 都有 $\lambda(u_i) = R_i(h(\bar{x}_i))$）。因此， T_1 的每个节点 v 也都满足这一性质（对于我们用节点 v_{i_r} 替换的节点 v_{i_j} 的父节点，该性质成立，因为 $\lambda(v_{i_j}) = \lambda(v_{i_r})$）。

显然，应用上述过程时，我们至少会消去一条长度为 $p+1$ 的路径，且不会引入任何新的长度为 $p+1$的路径。如果我们对每一条长度为 $p+1$的路径重复这一过程，最终将得到一棵关于 D 和 Σ 的 α 的证明树 T_2，其深度(T_2) $\leq p$，且支撑集(T_2) $=D'$。结论由归纳假设得出。 □

在证明引理 7 之前，我们先给出以下结果，其中不考虑证明树的支撑集。

引理 16。对于每个 *Datalog* 程序 Σ、基于扩展数据库(Σ)的数据库 D，以及基于模式(Σ)的事实 α，如果存在关于 D 和 Σ 的 α 的证明树T，那么也存在这样一棵证明树T'，使得对于 T' 中出现的每个事实 β，都有 $|T'[\beta]_{/\approx}| = 1$。

证明。设 $T = (V, E, \lambda)$ 是关于 D 和Σ 的 α 的一棵证明树。我们按如下方式构造另一棵关于 D 和 Σ 的α 的证明树 $T' = (V', E', \lambda')$，使其具有所需性质。令 S 为一个集合，对于 T 中出现的每个事实 β（即它是 V 中某个节点的标签），该集合包含 T 的一棵子树 $T[v]$，且这棵子树具有 $\lambda(v)$，并且在所有此类子树中深度最小；如果存在多棵这样的树，则任选一棵。对于每个 $1 \leq i \leq$ 深度(T)，令 S^i 为包含 S 中所有深度恰为 i 的树的集合。现在我们将通过归纳构造一个树集合，其中对 T 中出现的每个事实 β，都包含一棵具有代表性的树。然后，我们将使用 α 的代表树作为树 T'。

我们定义：

- $Z_1 = S^1$;
- $Z_{i+1} = \mathsf{Op}^{i+1}(Z_i) \cup Z_i$, for $i \geq 1$.

其中，$\mathsf{Op}^{i+1}(Z_i)$ 对 S^{i+1} 中的每棵树 T'' 都包含一棵通过以下过程从该树得到的树。设 T'' 的根节点的直接子节点为 u_1、.....、u_n。对于每个满足 $\lambda(u_j) = \beta$ 的子节点 u_j，我们用一棵根节点标签为 β 的 Z_i 树替换子树$T''[u_j]$。直观地说，这样一棵树的存在性是有保证的，因为S 对每个事实都包含一棵深度最小的子树；又因为$T''[u_j]$ 的深度至多为 i，集合 S 中就有一棵深度至多为 i、且以标签为 β 的节点为根节点的树。形式上，我们证明集合 Z_i具有以下性质：

1. 对于每个标签 β，如果 S^i 包含一棵根节点为 v 且满足 $\lambda(v) = \beta$ 的树，那么 Z_i 包含一棵根节点为 u 且满足 $\lambda(u) = \beta$ 的树。2. 对于每个标签 β，如果 Z_i 包含一棵根节点为 u 且满足 $\lambda(u) = \beta$ 的树，那么存在一棵根节点为 w 的树 T''，满足 $\lambda(w) = \beta$ 且 $T'' \in S^k$，其中某个 $k \leq i$。3. 对于每个标签 β，如果 Z_i 包含一棵根节点为 u 且满足 $\lambda(u) = \beta$ 的树，那么这样的树恰有一棵。4. 对于 Z_i 的每棵树 T''，如果 T'' 的一个叶节点为 u，那么$\lambda(u) \in D$。5. 对于 Z_i 的每棵树 T''，如果 T'' 的一个节点 u 有$n \geq 1$ 个子节点 u_1、.....、u_n，那么存在 Σ 中的一条规则$R_0(\bar{x}_0) :- R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$，以及一个函数 $h : \bigcup_{i \in n} \bar{x}_i \to \mathsf{C}$，使得 $\lambda(u) = R_0(h(\bar{x}_0))$，并且对于 $j \in [n]$，$\lambda(u_j) = R_j(h(\bar{x}_j))$。6. 对于 Z_i 的每棵树 T''，$|T''[\beta]_{/\approx}| = 1$，针对 T'' 中出现的每个事实 β。

我们通过对 i进行归纳法证明全部六个性质。

基础情形。对于 $i = 1$，前两个性质根据定义显然成立，因为 $Z_1 = S^1$。由于 S 中每个事实 β 都对应一棵树（即根节点标记为 β的树），S^1也是如此，因此第三个性质也成立。第四和第五个性质成立，是因为 S^1 中的每棵树（因此每棵树 Z_1）都是证明树的子树，而这些性质对于证明树成立。最后一个性质也成立，因为 Z_1 中的一棵树包含一个根节点 v 及其子节点 u_1、.....、u_n，不可能对某个 $j \in [n]$出现 $\lambda(v) = \lambda(u_j)$ 的情况（因为叶节点对应外延谓词，而根节点对应内涵谓词）。

归纳步骤。我们假设该命题对 $i = 1, \ldots, p$ 均成立，并证明它对 $i = p+1$ 也成立。第一个性质由构造可得，因为集合 $\mathsf{Op}^{p+1}(Z_p)$ 中，对于 S^{p+1} 的每棵树 T''，都包含另一棵具有相同根节点的树（我们只修改其子节点的子树）。此外，如果 T'' 根节点的子节点为 $u_1, \ldots, u_n$，那么对于每个 $r \in [n]$，子树 $T''[u_r]$（它也是原始 T 的子树）的深度至多为 p。因此，T 中标签为 $\lambda(u_r)$ 的最浅子树会在某个 $1 \leq k \leq p$ 对应的 S_k 中出现。根据归纳假设，集合 Z_k 包含一棵根节点为 v 的树，使得 $\lambda(v) = \lambda(u_r)$；又由于 $Z_k \subseteq Z_p$，这棵树也出现在 Z_p 中。因此，我们的构造定义良好。

第二个性质成立，因为每棵以 u 为根、满足 $\lambda(u) = \beta$ 的 Z_{p+1} 树，要么出现在 Z_p 中，要么由 S_{p+1} 中的一棵树得到。对于第一种情况，归纳假设意味着存在一棵树 T''，其根节点为 w，且 $\lambda(w) = \beta$ 位于 S^k 中，其中 $k \leq p$ 满足某个条件。对于第二种情况，根据 $\mathsf{Op}^{p+1}(Z_p)$ 的定义，存在一棵树 T''，其根节点为 w，且 $\lambda(w) = \beta$ 位于 S^{p+1} 中。

第三个性质成立，因为 S^{p+1} 中每个事实只对应一棵树，因此 $\mathsf{Op}^{p+1}(Z_p)$ 也如此。此外，根据归纳假设，Z_p 中每个事实只对应一棵树。我们将证明，不可能存在一个标签 β 和两棵树 T_1、T_2，使得：(1) 对于 T_1 的根节点 v_1，有 $\lambda(v_1) = \beta$；(2) 对于 T_2 的根节点 v_2，有 $\lambda(v_2) = \beta$；(3) $T_1 \in Z_p$；以及 (4) $T_2 \in \mathsf{Op}^{p+1}(Z_p)$。反设这样的两棵树确实存在。那么，$S^{p+1}$ 中包含一棵根节点为 v_3 的树 T_3，满足 $\lambda(v_3) = \beta$（T_2 就是由这棵树得到的）。此外，由归纳假设和性质 (2) 可知，对于某个 $k \leq p$，存在一棵根节点为 v_4 的树 T_4，它位于 S^k 中，且满足 $\lambda(v_4) = \beta$。由此可知，S 中有两棵树，其根节点都标记为 β——一棵深度为 $p+1$，另一棵深度为 $k \leq p$。这与以下事实矛盾：S 仅包含 T 中根节点标记为 β 的、深度最小的子树。

至于第四个性质，如前所述，T'' 中的每棵树 Z_{p+1} 要么出现在 Z_p 中，要么由S_{p+1}中的一棵树得到。对于第一种情况，该结论可直接由归纳假设推出。对于第二种情况，设 T'' 是 $\mathsf{Op}^{p+1}(Z_p)$ 中的一棵树。假设 T'' 的根节点是 u，其子节点是 u_1、.....、u_n。那么，对于某个 $j \in [n]$，T''的每个叶节点也是 $T''[u_j]$的叶节点；又由于根据构造，$T''[u_j]$是 Z_p 中的一棵树，因此根据归纳假设，我们有 $\lambda(u_j) \in D$。

根据归纳假设，Z_p 的每棵树都满足第五个性质。我们将证明，T'' 的每棵树也满足 $\mathsf{Op}^{p+1}(Z_p)$ 的第五个性质。S^{p+1} 中的每棵树都是 T 的子树；因此，它满足所需的性质（这是证明树的一个性质）。特别是，根节点满足该性质；而且由于我们没有修改根

节点的标签，也没有修改其子节点的标签，因此所得树T''的根节点也满足该性质。对于根节点的每个子节点，其子树都被替换为来自 Z_p 、根据归纳假设满足所需性质的一棵树，因此 T'' 的每个节点都满足该性质。

最后，对于也出现在 Z_p 中的 Z_{p+1} 树，根据归纳假设，最后一条性质成立。对于来自 $\mathrm{Op}^{p+1}(Z_p)$ 的 Z_{p+1} 的 T''树，根据归纳假设，对于出现在 T'' 中且不是根节点标签的每个标签 δ，最后一条性质都成立（因为我们用来自 Z_p 的树替换了根节点各子节点下的子树）。请注意，T'' 的根节点标签 β 不可能出现在 Z_p 的树中（尤其不可能作为其某个子节点的标签）。这是因为 $Z_p \subseteq Z_{p+1}$，并且我们已经证明了性质 (3)。因此，$|T''[\beta]_{/\approx}| = 1$ 也成立。至此，我们完成了对这六条性质的证明。

现在，根据定义，S 包含一棵树 T_1，其根节点标记为 α。假设这棵树的深度为 k，那么 $T_1 \in S^k$。性质 (1) 随即意味着，Z_k 包含一棵树T_2，其根节点标记为 α。现在只需证明这棵树是 α 关于 D 和 Σ 的一棵证明树，并满足所需性质，然后我们将定义 $T' = T_2$，至此证明完成。证明树的第一个性质（定义 1）显然成立，因为对于 T_2 的根节点 v，$\lambda(v) = \alpha$。由于集合 Z_i 分别满足性质 (4) 和 (5)，证明树的性质 (2) 和 (3) 也成立。因此，T_2 确实是 α 关于 D 和Σ的一棵证明树。集合 Z_i 的性质 (6) 意味着，对于 T_2 中出现的每个事实 β，都有 $|T_2[\beta]_{/\approx}| = 1$。因此，我们确实可以定义 $T' = T_2$，并得到所需的证明树。

□

现在我们着手证明引理 7，现将其重述如下：

引理 *7.* 对于每个 *Datalog* 程序 Σ 和一个多项式f，存在一个多项式 g，使得对于每个在 ***edb(Σ)*** 上的数据库 D、在 ***sch(Σ)*** 上的事实 α 以及 $D' \subseteq D$，如果存在一个关于 D 和 Σ 的、用于证明 α 的证明树 T，满足 ***support(T)*** $=D'$ 且 ***depth(T)*** $\leq f(|D|)$，那么也存在这样一棵证明树 T'，满足 ***scount(T')*** $\leq g(|D|)$。

证明。给定一棵关于 D 和 Σ 的、用于证明 α 的证明树 T，满足 $\mathrm{depth}(T) \leq f(|D|)$且 $\mathrm{support}(T) = D'$，我们分两步构造另一棵关于 D和 Σ的、用于证明 α的证明树 T'，满足 $\mathrm{scount}(T') \leq g(|D|)$且 $\mathrm{support}(T') = D'$。首先，对于 D'中的每个事实，我们在 T中选取一条从根节点到标记有该事实的叶节点的路径。然后，我们“冻结”这些路径（即，不在 T'中修改它们），以保留支撑集。然而，仅保留这些路径并不足够；我们还需要保留沿这些路径上每个节点的兄弟节点，才能得到有效的证明树，特别是满足定义 1 的最后一个性质。第二步是，对于每个兄弟节点 v 以及 $T[v]$中的每个事实 β，减少 $T[v][\beta]_{/\approx}$中的等价类数量。

设 v 是 T 的一个节点。容易观察到，如果对于某个事实 β，$\lambda(v) = \beta$，那么 $T[v]$ 就是关于 D 和 Σ的 β的证明树。接着，引理 16 表明，存在另一棵关于 D 和 Σ 的 β 的证明树 T_v，其根节点标记为 β，并且对于 T_v中出现的每个事实 δ，都有 $|T_v[\delta]_{/\approx}| = 1$。于是，我们可以用树 T_v 替换 T 的子树 $T[v]$。对每个兄弟节点都这样处理。显然，通过这一过程得到的树 T' 仍然是关于 D 和 Σ的 α 的证明树。这是因为我们没有修改根节点的标签；因此，定义 1 的性质 (1) 成立。此外，定义 1 的性质 (2) 也成立，因为 T' 的叶节点集合是 T 的叶节点集合的子集（这是由引理 16 证明中的构造所致）。最后，对于子树 T_v 中的每个节点，定义 1 的性质 (3) 都成立，因为如前所述，每个 T_v 都是关于 D 和 Σ' 的某个事实的证明树。此外，冻结路径上的每个节点也满足这一性质，因为对于每条这样的路径 $v_1 \to \ldots \to v_n$，如果 v_j 的子节点是 u_1、.....、u_m，那么其中一个子节点是 v_{j+1}，而我们不会修改它的标签，也不会修改其兄弟节点的标签（我们只替换它们下方的子树）。剩下只需证明 T' 满足所需性质。

为此，我们观察到：*(1)* 每个事实 β 在 T' 上沿冻结路径出现多项式次，且*(2)*这样的兄弟节点有多项式个。性质 (1) 成立，因为对于某个多项式 f，深度$(T) \leq f(|D|)$；因此，路径上的节点数为多项式级（至多为$f(|D|) + 1$）。此外，由于我们为 D'的每个事实冻结一条路径，因此共有 $|D'|$ 条路径。于是，每个事实在冻结路径上至多出现$[f(|D|) + 1] \times |D'|$ 次。性质 (2) 成立，因为冻结路径上的每个节点至多有 $b-1$个兄弟节点，其中 b 是 Σ中某条规则的体部所含原子数的最大值。因此，兄弟节点的数量至多为 $[f(|D|)+1]\times|D'|\times(b-1)$。根据我们的构造，对于出现在某个兄弟节点的子树 T''中的每个事实 β，都有 $|T''[\beta]_{/\approx}| = 1$。因此，

$$|T'[\beta]_{/\approx}| \leq [f(|D|) + 1] \times |D'| \times b$$

（冻结路径上的每个节点对应一个等价类，每个兄弟节点也对应一个等价类）。于是可得：

$$g(|D|) = [f(|D|) + 1] \times |D'| \times b.$$

至此，证明完毕。 □

最后，我们证明引理 8，并在此重述如下：

引理 *8.* 对于每个 *Datalog* 程序 Σ 和多项式f，存在一个多项式 g，使得对于扩展数据库Σ上的每个数据库 D、事实 α以及 $D' \subseteq D$，若存在一个关于 D和 Σ、支撑集为 ***support(T)*** $= D'$ 且 ***scount(T)*** $\leq f(|D|)$ 的 α 的证明树 T，则存在一个关于 D 和 Σ、支撑集为 ***support(G)*** $= D'$ 且 $|V| \leq g(|D|)$ 的 α的证明有向无环图 $G = (V, E, \lambda)$。

证明。我们的目标是构造一个有向无环图 $G = (V, E, \lambda)$，其中对于每个出现在 T 中的事实 β 以及 $T[\beta]_{/\approx}$ 的每个等价类，都包含一个表示该等价类的有向无环图。为此，我们首先针对每个出现在 T 中的事实 β 以及 $T[\beta]_{/\approx}$ 的每个等价类 C_β，向 V 添加 $v_1^{C_\beta}, \ldots, v_k^{C_\beta}$ 个节点，其中 k 是 T 的单个节点下 C_β 中树出现次数的最大值。显然，$k \leq b$，其中 b 是 Σ中某条规则体部内出现的原子数最大值。接着，对于每个 $j \in [k]$，我们定义 $\lambda(v_j^{C_\beta}) = \beta$。注意，树 T 本身属于某个等价类 $C \in T[\alpha]_{/\approx}$（因为根据证明树的定义，对于 T 的根节点 v，有 $\lambda(v) = \alpha$

树），并且这棵树不会出现在 T 的任何节点下；因此，对于这个等价类，有 $k = 0$。在这种情况下，我们向 V 添加一个节点 v^C。

接下来，如果对于每棵以 v 为根节点的 C 树 T'，$T'[u] \in C'$ 恰好有 p 个子节点 u 或 v，我们就向 E 添加边 $(v_j^{C_\beta}, v_1^{C'_{\beta'}}) \ldots, (v_j^{C_\beta}, v_p^{C'_{\beta'}})$。（注意，这一条件要么对 C 的所有树都成立，要么对其中任何一棵都不成立，因为 C 是一个等价类。）V 中的节点数以 $|\mathsf{base}(D,\Sigma)| \times f(|D|)$ 为界，因为 T 中出现的事实数以 $|\mathsf{base}(D,\Sigma)|$ 为界；对于每个事实，至多有 $f(|D|)$ 个等价类；而对于每个事实与其等价类的组合，至多有 b 个节点；因此，通过定义

$$g(|D|) \;=\; |\mathsf{base}(D,\Sigma)| \times f(|D|) \times b,$$

$|V| \leq g(|D|)$。还需证明 G 是一个证明有向无环图。

令 v 为证明树 T的根节点。如前所述，v 的子树（即 T 本身）属于 $T[\alpha]_{/\approx}$的某个等价类 C，并且在 V中有一个节点 v^C，满足 $\lambda(v^C) = \alpha$。显然，T 中不存在其他满足 $T[u] \in C$的节点 $u \neq v$；因此，根据 E的定义，节点 v^C 没有入边。相反，对于某个事实 β的其他任一等价类 C_β，如果 V 恰好包含与该等价类对应的 k 个节点 $v_1^{C_\beta}$、.....、$v_k^{C_\beta}$，那么根据定义，存在 T 中的一个节点 u，满足$\lambda(u) = \delta$，其中某个 δ具有 k个子节点 u_1、.....、u_k，且它们的子树都属于 C_β。此时，E 包含边（$v_1^{C_\delta}$，$v_1^{C_\beta}$）、.....、（$v_1^{C_\delta}$，$v_k^{C_\beta}$），其中 C_δ是子树 $T[u]$所属的等价类。由此可知，G 中恰有一个没有入边的节点，且该节点的标签为 α；因此，定义4的性质（1）成立。

我们可以类似地证明，每个叶节点 $v \in V$都对应于 T 中一个标签为相同事实的叶节点 v'。也就是说，如果 v是 G的叶节点，那么它的形式为 $v_j^{C_\beta}$，其中 β 是 T某个叶节点的标签，而 C_β 是一个等价类，其中包含仅有一个节点且该节点标记为 β 的树。由于 T是一棵证明树，对于 T的每个叶节点 v'，都有 $\lambda(v') \in D$，因此对于 G 的每个叶节点 v，都有 $\lambda(v) \in D$，于是定义4的性质 (2) 成立。

最后，我们证明定义4的性质（3）由 G 满足。设 $v_j^{C_\beta}$ 是 V 中一个具有 $n \geq 1$ 条出边 $(v_j^{C_\beta}, v_{j_1}^{C_{\delta_1}}) \ldots, (v_j^{C_\beta}, v_{j_n}^{C_{\delta_n}})$ 的节点。根据 V 的定义，存在 T 中的一个节点 u，满足 $\lambda(u) = \beta$，使得 $T[u] \in C_\beta$。根据 E 的定义，若等价类 $C_{\delta_1}, \ldots, C_{\delta_n}$ 中某个类 C（对应于某个事实 γ）恰好出现 p 次，那么对于 C_β 的每棵树（特别是 $T[u]$），该树的根节点（特别是节点 u）恰好有 p 个子节点 $u_1, \ldots, u_p$，使得对每个 $\ell \in [p]$ 都有 $T[u_\ell] \in C$。这也意味着，根据 V 的定义，对每个 $\ell \in [p]$ 都有 $\lambda(u_\ell) = \gamma$。由此可知，T 中存在一个节点 u，其子节点为 $u_1, \ldots, u_n$，使得 $\lambda(u) = \beta$，并且对每个 $i \in [n]$ 都有 $\lambda(u_i) = \delta_i$。由于 T 是一棵证明树，这意味着存在一条规则 $R_0(\bar{x}_0) \coloneq R_1(\bar{x}_1) \ldots, R_n(\bar{x}_n) \in \Sigma$，以及一个函数 $h : \bigcup_{i\in[n]} \bar{x}_i \to \mathsf{C}$，使得 $\lambda(v) = R_0(h(\bar{x}_0))$，并且对于 $i \in [n]$，$\lambda(u_i) = R_i(h(\bar{x}_i))$。因此，我们还有 $\lambda(v_j^{C_\beta}) = R_0(h(\bar{x}_0))$，并且对于每个 $i \in [n]$ 都有 $\lambda(v_{j_i}^{C_{\delta_i}}) = R_i(h(\bar{x}_i))$，性质（3）确实成立。由此我们得出，$G$ 是 α 关于 D 和 Σ 的一个证明有向无环图，且满足 $|V| \leq g(|D|)$。□

完成证明。借助上述技术性引理，现在很容易证明 NP 上界。固定一个 Datalog 查询 $Q = (\Sigma, R)$。给定一个定义在 edb(Σ) 上的数据库 D、一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$以及 D的一个子集 D'，要判定 $D' \in \mathsf{why}(\bar{t}, D, Q)$ 是否成立，我们只需检查是否存在一棵关于 D和 Σ 的 $R(\bar{t})$ 证明树 T，使得 $\mathsf{support}(T) = D'$。根据命题 5，这等价于存在一个关于 D 和 Σ 的 $R(\bar{t})$ 紧凑证明有向无环图 G，使得 $\mathsf{support}(G) = D'$。显然，只需猜测一个相对于 $|D|$而言大小为多项式、带标签的有向图 $G = (V, E, \lambda)$，再检查 G是否无环、有根，并且是关于 D 和 Σ的 $R(\bar{t})$ 证明有向无环图，且 $\mathsf{support}(G) = D'$，即可判定是否存在这样的证明有向无环图。由于这两个步骤都能在多项式时间内完成，Why-Provenance[Q]属于 NP，因此在数据复杂度下，Why-Provenance[Dat]属于 NP。

下界

我们接下来证明，Why-Provenance[LDat] 在数据复杂度下是 NP困难的。为此，我们需要证明存在一个线性 Datalog 查询 Q，使得问题 Why-Provenance[Q] 是 NP困难的。证明采用从 3SAT 出发的归约，输入是一个 3CNF 布尔公式 $\varphi = C_1 \wedge \ldots \wedge C_m$，其中每个子句恰好包含 3 个文字（一个布尔变量 v 或其否定 $\neg v$），并询问 φ 是否可满足。

线性Datalog查询。我们首先定义线性 Datalog 查询 $Q = (\Sigma, R)$。如果变量的名称无关紧要，那么我们用一个只在 Σ 中出现一次的新变量代替。为简便起见，我们在元组表达式中用分号代替逗号，以分隔具有不同语义含义的项。程序 Σ 如下：

σ_1 : $R(x) \coloneq \mathsf{Var}(x; z,\ ),\ \mathsf{Assign}(x,\ z),$
σ_2 : $R(x) \coloneq \mathsf{Var}(x;\ \ z), R(z),$
σ_3 : $\mathsf{赋值}(x,y) \coloneq\ C(x,y;\ ,;\ ,), \mathsf{赋值}(x,y),$
σ_4 : $\mathsf{赋值}(x,y) \coloneq\ C(\ ,;x,y;\ ,), \mathsf{赋值}(x,y),$
σ_5 : $\mathsf{赋值}(x,y) \coloneq\ C(\ ,;\ ,;x,y), \mathsf{赋值}(x,y),$
σ_6 : $\mathsf{赋值}(x,z) \coloneq \mathsf{Next}(x,y;z,)\ R(y),$
σ_7 : $\mathsf{赋值}(x,z) \coloneq \mathsf{Next}(x,y;z)\ R(y),$
σ_8 : $R(x) \coloneq \mathsf{Last}(x)$。

很容易验证，Σ 确实是一个线性 Datalog 程序。程序 Σ 背后的高层思路是：对于给定布尔公式 φ中出现的每个变量 v，非确定性地为其 v赋一个值（0 或 1），然后检查全局赋值是否使 φ 为真。规则 σ_1和 σ_2 负责为变量 v赋值 0 或 1；关系 Var 的最后两个位置始终分别存储值 0 和 1。规则 σ_3、σ_4和 σ_5 负责检查：对某个变量 v 的赋值，是否会使某个子句 C中提及 v 的文字为真

（因此也使 C 本身为真）。规则 σ_6 和 σ_7 负责在处理完某个变量 v后，转而考虑其后面的变量 u；关系 Next 为给定 3CNF 布尔公式中的变量提供了一个顺序。最后，在考虑完公式中的所有变量后，σ_8 会将我们带到最后一个变量；这是一个虚设变量，用来表示上述过程的结束。

从 3SAT到 Why-来源$[Q]$。现在我们通过从 3SAT 归约，证明 Why-来源$[Q]$ 是 NP困难的。考虑一个 3CNF布尔公式 $\varphi = C_1 \wedge \cdots \wedge C_m$，它有 n个布尔变量 $v_1, \ldots, v_n$。对于文字 ℓ，我们用 $\mathsf{v}(\ell)$ 表示 ℓ中出现的变量，并用 $\mathsf{b}(\ell)$ 表示数值 1（若 ℓ 是变量）或 0（若 ℓ 是变量的否定）。我们将 D_φ 定义为基于 edb(Σ) 的数据库

$$
\begin{aligned}
& \{\mathrm{Var}(v_i; 0, 1) \mid i \in [n]\} \\
\cup\ & \{\mathrm{Next}(v_i, v_{i+1}; 0, 1) \mid i \in [n-1]\} \\
\cup\ & \{\mathrm{Next}(v_n, \bullet; 0, 1), \mathrm{Last}(\bullet)\} \\
\cup\ & \{C(\mathsf{v}(\ell_1), \mathsf{b}(\ell_1); \mathsf{v}(\ell_2), \mathsf{b}(\ell_2); \mathsf{v}(\ell_3), \mathsf{b}(\ell_3)) \mid \\
& \qquad (\ell_1 \vee \ell_2 \vee \ell_3) \text{ is a clause of } \varphi\},
\end{aligned}
$$

它本质上存储了 φ 的子句，并为 φ 中出现的变量提供排序，其中 $\bullet$ 是一个虚设变量。我们可以给出以下引理，它本质上说明上述构造导出了一个正确的多项式时间归约，从 3SAT 到 Why-来源$[Q]$:

引理 17。 D_φ 可在关于 φ 的多项式时间内构造。此外，φ 可满足，当且仅当 $D_\varphi \in \mathsf{why}((v_1)\ D_\varphi, Q)$。

证明。显然，D_φ 属于扩展数据库(Σ)，并且 D_φ，(v_1) 可在关于 φ 的多项式时间内构造。现在我们证明，φ 可满足，当且仅当存在一棵证明树 T，其根节点为 $R(v_1)$，相对于 D_φ 和 Σ，满足 support$(T) = D_\varphi$。

我们先证明 (⇒) 方向。假设 φ 在真值赋值 μ 下可满足。对于每个变量 v_i，我们将形式为 $C(v_i,\ \mu(v_i);\ ;\ ;\ )$,$C(\ ;\ v_i,\ \mu(v_i);\ ,)$ 和 $C(\ ;\ ;\ ;v_i,\ \mu(v_i))$ 的事实集合记为$\bar{S}_{v_i}$。我们定义标记根树 $\bar{T} = (V,\ \bar{E},\ \bar{\lambda})$，其中根节点 $v \in V$标记为 $\lambda(v) = R(v_1)$，并递归定义如下:

1. 如果 $v \in V$ 被标记为 $\lambda(v) = R(v_i)$，其中 $i \in[n]$，那么 v 有两个子节点 u_1, u_2，其中 $\lambda(u_1)$ 是 D_φ 中唯一一个形如 $Var(v_i; 0, 1)$ 的事实，并且 $\lambda(u_2)$ =赋值$(v_i, \mu(v_i))$。2. 如果 $v \in V$被标记为 $\lambda(v)$ = 赋值$(v_i, \mu(v_i))$，其中 $i \in[n]$，且 S_{v_i} 非空，那么 v 有两个子节点 u_1, u_2，其中 $\lambda(u_1) = f$ 对应某个事实 $f \in S_{v_i}$，并且 $\lambda(u_2)$ =赋值$(v_i, \mu(v_i))$。然后我们从 S_{v_i}中移除 f。3. 如果 $v \in V$ 被标记为 $\lambda(v)$ = 赋值$(v_i, \mu(v_i))$，其中$i \in [n-1]$，且 S_{v_i} 为空，那么 v 有两个子节点u_1, u_2，其中 $\lambda(u_1)$ =Next$(v_i, v_{i+1}; 0, 1)$，并且 $\lambda(u_2) = R(v_{i+1})$。4. 如果 $v \in V$ 被标记为 $\lambda(v)$ = 赋值$(v_n, \mu(v_n))$，且 S_{v_n} 为空，那么 v 有两个子节点 u_1, u_2，其中$\lambda(u_1) =$ Next$(v_n, \bullet; 0, 1)$，并且并且 $\lambda(u_2) = R(\bullet)$。5. 如果 $v \in V$ 被标记为 $\lambda(v) = R(\bullet)$，那么 v 有一个子节点 u_1，其中 $\lambda(u_1)$ = 最后(•)。

可以验证，T 确实是关于 D_φ 和 Σ的 $R(v_1)$ 的证明树。特别地，根节点按构造被标记为$R(v_1)$，并且很容易看出，所有叶节点的标签都出现在 D_φ中。此外，由于 Σ 是线性的，在 T 的每个层级上至多存在一个非叶节点。该节点发出的边由上文第（1）– （5）项定义。第（1）项中定义的边，是通过考察规则 σ_1, σ_2 得到的；第（2）项中定义的边，是通过考察规则 $\sigma_3, \sigma_4, \sigma_5$ 得到的；第（3）项和第（4）项中定义的边，是通过考察规则 σ_6, σ_7 得到的。最后，第（5）项中定义的边，是通过考察规则 σ_8 得到的。因此，定义 1 的性质（3）成立。

关于 T 的支撑集，由于 μ 是一个满足赋值，φ 的每个子句都至少包含一个文字 ℓ_j，使得 $\mu(\mathsf{v}(\ell_j))$ = $\mathsf{b}(\ell_j)$。第（1）项确保我们遵循这个满足赋值（即，在对应变量 v_i 的节点 $R(v_i)$ 下，添加一个标记为 Assign$(v_i,\ \mu(v_i))$ 的节点）。第（2）项确保每当我们考察变量 v_i 时，都会访问谓词 C 上对应于包含变量 v_i 且符号正确的子句的每个原子（若 $\mu(v_i) = 1$，则为正；若 $\mu(v_i) = 0$，则为负）；也就是说，访问每个由对该变量的赋值所满足的子句。第（3）项和第（4）项确保我们遍历所有变量，因为谓词 Next 上的原子只允许我们从变量 v_i 移动到下一个变量 v_{i+1}。反过来，这又确保根据第（1）项，我们访问谓词 Var 上的每个原子；根据第（2）项，访问谓词 C 上的每个原子（因为如前所述，每个此类子句都包含一个文字，其符号与对应变量的赋值一致）；并根据第（3）项和第（4）项，访问谓词 Next 上的每个原子。最后，第（5）项确保我们访问原子 Last(•)。因此，我们得出 support$(T) = D_\varphi$。

接下来，我们证明（⇐）方向。假设 T =(V, E, λ) 是关于 D_φ和 Σ的 $R(v_1)$ 的一棵证明树，且 support$(T) = D_\varphi$。注意，由于 Σ的线性性质，在 T的每个层级（最后一层除外），恰好存在一个非叶节点。因此，这些节点在 T中构成一条路径 $u_0, u_1, \ldots$。注意，根据 Q的定义，u_0（即根节点）必然标记为事实 $R(v_1)$。然后，根据规则 σ_1 或 σ_2，根节点的子节点分别标记为 Var$(v_1; 0, 1)$ 和 Assign(v_1, b_1)（即节点 u_1），其中 $b_1 \in \{0, 1\}$ 为某个变量，因为只有这两种可能。此时，要么节点 u_2根据规则 σ_6 或 σ_7标记为 $R(v_2)$，要么对于某个 $j \geq 2$，$u_2, \ldots, u_j$ 均标记为 Assign(v_1, b_1)，而后 u_{j+1} 标记为 $R(v_2)$。接着，对 v_2 及其后所有变量也可作同样的推理。

由于谓词 Next 上的原子只考虑相邻的变量，因此沿着路径只能从 v_i移动到 v_{i+1}；又由于 support$(T) = D_\varphi$，集合 support(T) 包含 Next 上的所有原子，因此我们可以得出，所有变量都会被遍历。此外，对于每个变量 v_i，一旦我们为标记为 $R(v_i)$ 的节点的子节点选择标签 Assign$(v_i,\ b_i)$，就没有规则允许我们得到标签 Assign$(v_i,\ 1 - b_i)$，因此谓词 Assign 上的标签对应于变量 v_1、 .. 、 v_n 的一个真值赋值 μ。最后，由于谓词 C 上的所有原子都出现在支撑集中，规则 σ_3、 σ_4、 σ_5 意味着，对于每个子句 $(\ell_1 \vee \ell_2 \vee$

ℓ_3)，沿着路径的某个位置都会出现标签 Assign(v(ℓ_1)，b(ℓ_1))、Assign(v(ℓ_2)，b(ℓ_2)) 或 ssign(v(ℓ_3)，b(ℓ_3))。因此，μ是一个满足赋值。 □

根据引理 17，Why-来源[Q] 是 NP困难，因此，Why-来源[LDat]在数据复杂度上是 NP困难的。

A.2 非递归查询

现在我们聚焦于非递归 Datalog 查询，并给出定理 9 的完整证明，这里重述如下：

定理 9. Why-来源[NRDat] 在数据复杂度上属于 AC_0 。

给定一个非递归 Datalog 查询 Q，我们已在正文中说明如何构造一阶查询 Q_{FO} 。这里的主要任务是确立该构造的正确性，即引理 12，现重述如下：

引理 12. 给定一个非递归 *Datalog* 查询 $Q = (\Sigma, R)$、一个在 扩展数据库(Σ)上的数据库 D 、 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，以及$D' \subseteq D$，则 $D' \in \mathsf{why}(\bar{t}, D, Q)$ 当且仅当 $\bar{t} \in Q_{FO}(D')$。

证明。 我们首先讨论（⇒）方向。存在一个关于 D 和 Σ、支撑集为 $= D'$ 的 $R(\bar{t})$ 的证明树 T。因此，T 是一棵 Q-树，集合 cq(Q) 包含合取查询 cq(T)，也就是由 T 导出的合取查询；集合 cq≃(Q) 包含一个合取查询 $\varphi(\bar{y})$，它在变量重命名意义下与 cq(T) 相同。接下来，很容易证明 D' 满足句子 $\psi_{\varphi(\bar{y})}[\bar{x}/\bar{t}]$。这又意味着 $\bar{t} \in Q_{FO}(D')$。

现在我们讨论（⇐）方向。存在一个合取查询 $\varphi(\bar{y}) \in$ cq≃(Q)，使得 D' 满足句子 $\psi_{\varphi(\bar{y})}[\bar{x}/\bar{t}]$。接下来，很容易证明存在一个关于 D 和 Σ、支撑集为 $= D'$ 的 $R(\bar{t})$ 的证明树。这又意味着 $D \in \mathsf{why}(R(\bar{t})\ D,\ Q)$。 □

B 非递归证明树

正如论文正文中所讨论的（见第 4.3 节），标准的 why-来源概念在第 3 节中定义，并在第 4 节中进行了详尽分析；它不受任何限制，依赖于任意证明树。事实上，只要输入数据库 D 的一个子集是关于 D 和 Datalog 查询 $Q = (\Sigma,\ R)$ 的元组 $\bar{t}$ 的任意证明树的支撑集，该子集就属于该元组的 why-来源。 然而，正如文献中已有讨论（例如，近期的研究 Bourgaux 等人，2022 ），有些证明树并不符合直觉。例 1 中的第二棵证明树便是如此，因为事实 $A(a)$ 是从其自身推导出来的。现在，通过这种不自然的证明树作为见证，why($\bar{t}$，D，Q) 的一个元素 D' 可能会被归类为关于 t 的反直觉解释，因为它并不对应于直观的推导过程；从证明树中可以提取出这一过程，它从 D' 推导出事实 $R(\bar{t})$。因此，我们有必要考虑经过细化的证明树类别，以克服任意证明树在概念上的局限，从而得到概念上直观的解释。本节中，我们着重讨论非递归证明树这一类别。粗略地说，非递归证明树是指这样一种证明树：其中不存在两个标记为相同事实的节点，使得其中一个是另一个的后代；这反映了上述讨论，即使用一个事实推导其自身是一种反直觉现象。正式定义如下：

定义 *18*（非递归证明树）。考虑一个 *Datalog* 程序 Σ、一个定义在扩展数据库(Σ)上的数据库 D，以及一个定义在模式(Σ)上的事实α。相对于 D 和Σ， α 的非递归证明树是一个证明树 $T = (V, E, \lambda)$，用于证明相对于 D 和 Σ 的 α，并且满足：对于任意两个节点 v和 $u \in V$，如果在 T中存在一条从 v到 u 的路径，那么 $\lambda(v) \neq \lambda(u)$。 ■

现在，我们基于非递归证明树定义 why-来源。给定一个 Datalog 查询 $Q = (\Sigma, R)$、一个定义在扩展数据库(Σ)上的数据库D，以及一个属于D的元组 $\bar{t} \in \mathsf{dom}^{\mathsf{ar}(R)}$，相对于非递归证明树， t 相对于 D 和 Q 的*why*-来源 t D 和 Q 的定义为以下事实集族

$$\{\mathsf{support}(T) \mid T \text{ is a non-recursive proof tree of } R(\bar{t}) \text{ w.r.t. } D \text{ and } \Sigma\}$$

记作 why$_{\mathsf{NR}}(\bar{t}, D, Q)$。那么，算法问题

$$\mathsf{Why\text{-}Provenance}_{\mathsf{NR}}[\mathsf{C}] \quad \text{and} \quad \mathsf{Why\text{-}Provenance}_{\mathsf{NR}}[Q]$$

其定义方式与第 3 节中的完全相同，关键区别在于使用 why$_{\mathsf{NR}}(\bar{t}, D, Q)$，而不是 why($\bar{t}, D, Q$)；也就是说，问题在于给定的数据库子集是否属于 why$_{\mathsf{NR}}(\bar{t}, D, Q)$。接下来，我们研究每个类 C $\in$ {Dat，LDat，NRDat} 的 Why-Provenance$_{\mathsf{NR}}$[C] 的数据复杂度。正如任意证明树的情形，对于递归查询，即使将递归限制为线性，该问题通常仍然难处理；而对于非递归查询，该问题则非常易处理。我们首先关注递归查询。

B.1 递归查询

我们证明以下复杂度结果：

定理 19。 Why-Provenance$_{\mathsf{NR}}$[C] 在数据复杂度意义下是 NP-完全的，对每个类 C $\in$ {Dat、LDat} 均成立。

要证明定理 19，只需证明：

- Why-来源$_{\mathsf{NR}}$[Dat]在数据复杂度意义下属于 NP。
- Why-来源$_{\mathsf{NR}}$[LDat]在数据复杂度意义下是 NP困难的。

我们先关注上界。

上界

该证明与第 3 节中为 Why-来源[Dat] 建立的类似结果的证明相似。给定一个 Datalog 程序 Σ、一个定义在 edb(Σ) 上的数据库 D，以及一个定义在 sch(Σ) 上的事实α，我们首先定义 α 关于 D 和 Σ的非递归证明有向无环图的与这一概念。随后，我们证明一个类似于命题 5 的结果：存在一个关于 D 和 Σ 的非递归证明树 α，且 support(T) $= D' \subseteq D$，当且仅当存在一个多项式大小的非递归证明有向无环图 G，它关于 D 和 Σ，且 support(G) $= D'$。由此可得到一个在多项式时间内运行的猜测-检验算法。下面将上述高层次描述形式化。

定义20(非递归证明有向无环图)。 考虑一个 *Datalog* 程序 Σ, 一个建立在 扩展数据库(Σ)之上的数据库 D，以及一个事实

α 建立在 模式(Σ)之上。α 关于 D 和 Σ 的非递归证明有向无环图是一个证明有向无环图 $G=(V,E,\lambda)$，它是 α 关于 D 和 Σ 的证明，并且满足：对于任意两个节点 v、$u\in V$，如果在 G 中存在一条从 v 到 u 的路径，那么 $\lambda(v)\neq\lambda(u)$。 ■

命题 5 的类似结果如下：

命题 21. 对于 *Datalog* 程序 Σ，存在一个多项式 f，使得对于每个定义在 扩展数据库(Σ)上的数据库 D、定义在模式(Σ)上的事实α 以及 $D'\subseteq D$，以下条件等价：

1. 存在一个关于 D 和 Σ 的非递归证明树 T，其支撑集$(T)=D'$。

2. 存在一个关于 D 和 Σ 的非递归证明有向无环图 $G=(V,E,\lambda)$，用于证明 α，其支撑集$(G)=D'$，且 $|V|\leq f(|D|)$。

方向 (2) 推出 (1) 的证明方法是将非递归证明有向无环图 G “ 展开 ” 为一个非递归证明 T，使得 support(G) = support(T)。我们采用与命题 5 中方向 (2) 推出 (1) 的证明相同的“展开”构造，该构造保持非递归性。

现在我们继续证明 (1) 蕴含 (2)。其基本构造分为两个主要步骤，分别由下文的引理 22 和 23 概述。

•第一步是证明：对于 D 和 Σ，关于 α 的事实 T，若其支撑集为 support(T) $=D'$，则可将其非递归证明树 T 转换为另一棵非递归证明树 T'；该证明树关于 D 和 Σ，具有支撑集 support(T') $=D'$，且其子树数量 “ 较小 ”。

引理 22。对于每个 *Datalog* 程序 Σ，都存在一个多项式 f，使得对于每个定义在扩展数据库(Σ)上的数据库 D、定义在模式(Σ)上的事实 α，以及 $D'\subseteq D$，如果存在一棵关于 D 和 Σ 的非递归证明树 T，用于证明 α，且其支撑集(T) 为 $=D'$，那么也存在这样一棵证明树 T'，使得scount(T') $\leq f(|D|)$。

证明。 我们首先注意到，根据定义，非递归证明树 T 的深度“较小”。具体而言，由于路径 T 上任意两个节点的标签都不相同，因此路径长度至多等于标签的数量，即 |base(D, Σ)|，显然这是数据库 D大小的多项式。另一个关键观察是，引理 7 背后的构造将深度“较小”的证明树转换为具有相同支撑集、子树数量“较小”的证明树，同时保留了非递归性。因此，我们可以将引理 7 背后的构造应用于非递归证明树 T，得到一棵支撑集为 support(T) =support(T') 的非递归证明树T'，并且满足 scount(T') $\leq f(|D|)$，其中 f 是引理 7 给出的多项式。

□

• 第二步 表明，支撑集为 support(T) $=D'$ 、子树数量“较小”的 α 关于 D 和 Σ 的非递归证明树T，可以转换为支撑集为 support(G) $=D'$ 的 α 关于 D 和 Σ 的紧凑非递归证明有向无环图 G。

引理*23.* 对于每个 *Datalog* 程序 Σ 和多项式f，都存在一个多项式 g，使得对于 ***edb(Σ)*** 上的每个数据库 D、事实 α以及 $D'\subseteq D$，如果存在一个关于 D 和 Σ 的 α 的非递归证明树 T，且其支撑集 ***support(T)*** $=D'$，并且 ***scount(T)*** $\leq f(|D|)$，那么就存在一个关于 D和 Σ的 α的非递归证明有向无环图 $G=(V,E,\lambda)$，其支撑集 ***support(G)*** $=D'$，并且 $|V|\leq g(|D|)$。

证明。 我们采用引理 8 背后的构造，它将子树数量“较小”的证明树转换为支撑集相同且大小为多项式的非递归证明有向无环图，因为它保持非递归性。后者成立是因为，对于证明树中的每条路径，证明有向无环图中都存在一条具有相同标签的路径，反之亦然。 □

现在可以清楚地看出，命题 21 的方向 (1) 蕴含 (2) 是引理 22 和 23 的直接推论。

完成证明。现在我们可以完成以下论断的证明：Why-Provenance$_{\mathsf{NR}}$[Dat] 在数据复杂度下属于 NP。固定一个 Datalog 查询 $Q=(\Sigma,R)$。给定一个定义在扩展数据库(Σ)上的数据库D、一个元组 $\bar{t}\in \mathrm{dom}(D)^{\mathsf{ar}(R)}$，以及 D 的一个子集D'，要判定 $D'\in \mathrm{why}_{\mathsf{NR}}(\bar{t},D,Q)$ 是否成立，我们只需检查是否存在一个关于 D 和 Σ 的 $R(\bar{t})$ 的非递归证明树 T，使得 support(T) $=D'$。根据命题 21，这等价于存在一个关于 D 和 Σ 的 $R(\bar{t})$ 的多项式大小非递归证明有向无环图 G，且 support(G) $=D'$。可以通过一个在数据库大小上以多项式时间运行的非确定性算法，检查是否存在这样的非递归证明有向无环图；证明 Why-Provenance[Dat] 在数据复杂度下属于 NP 时（定理 3）也采用了这种方法。唯一的区别是，现在还需要检查猜测出的有向无环图是否为非递归图。为此，可以使用深度优先搜索遍历该有向无环图的节点，并记下当前路径上见过的所有标签。每当遇到一个新节点时，都要验证其标签不属于当前路径上已经见过的标签集合。如果属于该集合，则拒绝；否则，就将其标签加入集合并继续。每当我们沿有向无环图“回退”（从节点 u 移动到节点 v，且存在一条边 (v,u)）时，就从集合中移除子节点的标签。这需要 $O(|V|\cdot|E|)$ 时间，其中 V 和 E 分别是该有向无环图的节点集合和边集合。因此，Why-Provenance[Q] 属于 NP，从而 Why-Provenance[Dat]在数据复杂度下属于 NP。

下界

接下来，我们将证明 Why-Provenance$_{\mathsf{NR}}$[LDat] 在数据复杂度方面是 NP困难的。为此，我们需要证明存在一个线性 Datalog 查询 Q，使得问题 Why-Provenance$_{\mathsf{NR}}$[Q] 是 NP困难的。证明采用从 Ham-Cycle 问题进行归约的方法。该问题的输入是一个有向图 $G=(V,E)$，要求判断 G是否存在哈密顿回路，即是否存在一个环$v_1,\ldots,v_n,v_1$，使其位于 G中，且对于互不相同的整数 $i,j\in[n]$、$v_i\neq v_j$和 $V=\{v_1,\ldots,v_n\}$。

线性Datalog查询。 我们首先定义线性 Datalog 查询 $Q=$(Σ，路径)。如果变量名不重要，那么对于在 Σ中仅出现一次的新变量，我们使用。为滥用记法，我们在元组表达式中使用分号而非逗号来分隔

含义不同的项。程序 Σ 如下:

σ_1 : MarkedE(x) :– First(x)σ_2 : MarkedE(y) :– E(, ;x, y;), MarkedE(x), σ_3 : 路径(y) :– E(x, y; , ; z), MarkedE(z) N(x), σ_4 : 路径(y) :– E(x, y; , ;), 路径(x) N(x)。

- - -

很容易验证, Σ 确实是一个线性 Datalog 程序。

从Ham-Cycle 到 Why-Provenance$_{\mathsf{NR}}[Q]$。 我们现在通过从 Ham-Cycle 归约, 证明 Why-Provenance$[Q]$ 是 NP困难的。考虑一个有向图 $G = (V, E)$, 其中$E = \{e_1, \ldots, e_m\}$。令 D_G 为 edb(Σ) 上的数据库

$$\{\text{First}(1)\} \cup \{N(v) \mid v \in V\} \cup \{E(u, v; i, i+1; m+1) \mid i \in [m] \text{ and } e_i = (u, v)\},$$

它本质上存储了图 G , 并为其边规定了顺序。现在我们证明下一个引理, 该引理指出上述构造可得出一个从 Ham-Cycle 到 Why-Provenance$_{\mathsf{NR}}[Q]$的正确多项式时间归约。

引理 24。 以下结论成立:

1. D_G 可在关于 G的多项式时间内构造。 *2.* G 存在哈密顿回路, 当且仅当对于某个任意节点 $v^* \in V$, $D_G \in \mathsf{why}_{\mathsf{NR}}((v^*)\ D_G, Q)$。

证明。 显然, D_G 可以在关于 G大小的多项式时间内构造。接下来我们证明第（2）项。首先证明方向（⇒）。假设 G 存在哈密顿回路 $v_1, \ldots, v_n, v_{n+1}$, 且不失一般性, 设 $v_1 = v_{n+1} = v^*$。因此, G中存在边 $(v_1, v_2) \ldots, (v_{n-1}, v_n), (v_n, v_{n+1})$。我们定义带标签的有根树 $T = (V', E', \lambda)$: 对于根节点 $v \in V'$, 令$\lambda(v)$ = 路径(v_{n+1}) = 路径(v^*), 并递归定义如下:

1. 若 $v \in V'$ 满足 $\lambda(v)$ = Path(v_{i+1}), 对于 $i \in \{2, \ldots, n\}$, 则 v 有 3 个子节点 u_1、u_2、u_3, 其中 $\lambda(u_1)$ 是 D_G 中唯一一个形式为 $E(v_i, v_{i+1}; \cdot, \cdot; \cdot)$ 的事实, $\lambda(u_2)$ = Path(v_i), 且 $\lambda(u_3) = N(v_i)$。 2. 若 $v \in V'$ 满足 $\lambda(v)$ = Path(v_2), 则 v 有 3 个子节点u_1、u_2、u_3, 其中 $\lambda(u_1)$ 是 D_G 中唯一一个形式为 $E(v_1, v_2; \cdot, \cdot; m+1)$ 的事实, $\lambda(u_2)$ = MarkedE($m+1$), 且 $\lambda(u_3) = N(v_1)$。 3. 若 $v \in V'$ 满足 $\lambda(v)$ = MarkedE($i+1$), 对于 $i \in \{1, \ldots, m\}$, 则 v 有 2 个子节点 u_1、u_2, 其中 $\lambda(u_1)$ 是 D_G 中唯一一个形式为 $E(\cdot, \cdot; i, i+1; \cdot)$ 的事实, 且 $\lambda(u_2)$ = MarkedE(i)。 4. 若 $v \in V'$ 满足 $\lambda(v)$ = MarkedE(1), 则 v 只有一个子节点 u, 其中 $\lambda(u)$ = First(1)。

至此, T的构造完成。接下来我们证明, T 是关于 D_G和 Q 的 Path(v^*) 的非递归证明树, 且 support(T) = D_G; 这进而意味着, 正如所需, $D_G \in \mathsf{why}_{\mathsf{NR}}((v^*)\ D_G,\ Q)$。由于 Σ 是线性的, 因此在 T 的每一层至多存在一个非叶节点。此外, T 中从第 i 层到第$i+1$层的边（其中 $i \in \{0, \ldots, n-1\}$）以及对应节点的标签（如第（1）项中所定义）都是有效的, 因为它们是通过考虑规则 σ_4 得到的, 并且存在边 (v_i, v_{i+1}) $\in E$, 其中 $i \in \{2, \ldots, n\}$。 T中从第 $n-1$ 层到 n 层的边是通过考虑 σ_3得到的, 标签也同样有效, 因为 (v_1, v_2) $\in E$。然后, 对于最后一层以外的所有其余层, 我们考虑第（3）项中的 σ_2 ; 对于最后一层, 我们考虑第（4）项中的 σ_1。在这最后两种情况下, 很容易验证这些标签是有效的。重要的是, 根据 T的构造, 对于每个非叶节点, T 中不存在其他具有相同标签的节点, 因此该证明树显然是非递归的。关于支撑集, 根据第（1）项, $N(v_i)$ ∈support(T), 其中 $i \in \{2, \ldots, n\}$。根据第（2）项, $N(v_1)$ ∈support(T)。此外, 根据第（3）项, $E(u, v; i, i+1; m+1)$ ∈support(T), 其中 (u, v) = e_i, 且 $i \in [m]$。最后, 根据第（4）项, First(1) ∈ support(T)。因此, support(T) = D_G, 结论得证。

现在我们继续讨论方向（⇐）。假设$D_G \in \mathsf{why}_{\mathsf{NR}}((v^*)\ D_G, Q)$, 这进而意味着, 存在一个非递归证明树 $T = (V, E, \lambda)$, 它是关于（v^*）相对于 D_G和 Q 的证明树, 且满足 support(T) = D_G。令 n 为 G中的节点数, 并且不失一般性地假设 $n \geq 2$。注意, 由于 Σ的线性性质, 在 T的每一层（最后一层除外）, 都恰好存在一个非叶节点。令 u_i 为第 i 层的非叶节点。显然, $u_0, u_1, \ldots$ 是 T中的一条路径。注意, 根据 Q的定义, u_0（即根节点）必然标记为一个使用谓词 Path 的事实; 第 $i \in \{1, \ldots, n-1\}$层的u_i也如此。事实上, 反设T 中 $i \in \{1, \ldots, n-1\}$ 是第一层, 其中 u_i 未标记为一个使用谓词 Path 的事实。由于 σ_3 和 σ_4 是 Σ 中仅有的体部原子使用谓词 N的规则, 而且也只有这些规则的头部中出现了 Path, 因此 support(T) 至多包含 $i < n$ 个使用谓词 N的事实, 从而 support(T) $\neq$ D_G, 这就产生了矛盾。

由于 $u_0, \ldots, u_{n-1}$ 是 T 中的一条路径, 且这些节点都由使用 Path 的事实标记, 因此我们得出, 这些标签是通过使用 σ_4 得到的, 因为它是唯一一个在体部和头部中都含有谓词 Path 的规则。因此, 由于 body(σ_4) 中的原子 $E(x, y; , ;)$ 使用了 Σ 的一个外延谓词, 并且已知 T 是一棵证明树, 所以 $\lambda(u_i)$ = Path(v_i), 其中 v_i 是图 G 的某个节点。因此, 由于 T 是非递归的, Path(v_0)、Path(v_1) . . . 、Path(v_{n-1}) 都互不相同, 进而 $v_0, v_1, \ldots, v_{n-1}$ 也互不相同。于是, 再根据原子 $E(x, y; , ;)$ 出现在 body(σ_4) 中且带有一个外延谓词、T 是一棵证明树这一事实, 以及 D_G 的构造, 我们得出 $(v_{n-1}, v_{n-2}) \ldots, (v_1, v_0)$ 都是图 G 中的边。因此, $v_0, \ldots, v_{n-1}$ 是 G 中一条哈密顿路径的逆序。此外, 令 $S_i \subseteq$ support(T) 表示这样一个事实集合: 这些事实使用谓词 N , 并标记 T 中截至层级 i 的节点。由于 σ_4 的体部中有原子 $N(x)$, 我们得出 $S_{n-1} = \{N(v_1) \ldots, N(v_{n-1})\}$, 也就是说, S_{n-1} 包含图 G 的所有节点, 除了 v_0。

现在聚焦于 u_{n-1} of T 中的节点 u_{n-1} 。回想一下, $\lambda(u_{n-1})$ = Path(v_{n-1})。注意, u_{n-1}的子节点不能再用 σ_4 标记, 因为它包含两个体部原子 Path(x) $N(x)$, 其中 N 是一个外延谓词。因此, 由于 $v_0, \ldots, v_{n-1}$ 恰好是 G 的所有节点之一, u_{n-1} 的一个子节点必然会被事实 Path(v) 标记, 其中 v 必然与 $v_0, \ldots, v_{n-1}$中的某个节点重合, 并且

因此， T 不会是非递归的。所以，唯一剩下的规则是 σ_3。如前所述，在层级 $n-1$ 以内，支撑集 support(T) 中谓词为 N 的事实集合是 $S_{n-1} = \{N(v_1), \ldots, N(v_{n-1})\}$。因此，要使 support($T$) 也包含 Path($v_0$) $\in D_G$，在 T 中层级 $n-1$ 以下至少还必须有一个节点被 $N(v_0)$ 标记。由于 σ_3 没有谓词为 Path 的体部原子， u_{n-1} 的一个子节点必然被 $N(v_0)$ 标记。因此，借助 σ_4 的体部中的原子 $E(x, y; ; ; z)$，我们可以得出 G 中也存在边 (v_0, v_{n-1})，从而 $v_0, v_1, \ldots, v_{n-1}$，$v_0$ 是 G 中一个哈密顿回路的逆序，结论得证。

□

根据引理 24，Why-来源$_{\mathsf{NR}}$[Q]是 NP困难。因此，Why-来源$_{\mathsf{NR}}$[LDat]在数据复杂度上是 NP困难的。

B.2 非递归查询

现在我们聚焦于非递归 Datalog 查询，并给出以下关于相对于非递归证明树的 why-来源的数据复杂度的结论：

定理 25. Why-来源$_{\mathsf{NR}}$[NRDat]在数据复杂度上属于 AC_0 。

证明。 这可通过一阶可改写性来证明，方法与定理 9 中相同。事实上，目标一阶查询的构造与定理 9 的证明完全相同，关键区别在于，对于 Datalog 查询 Q，合取查询集合 cq(Q) 的定义仅考虑非递归证明树，即该集合为 {cq(T) | T 是一个非递归 Q-树}。

□

C 最小深度证明树

现在我们关注文献中研究过的另一类经过细化的证明树。回忆一下，有根树 T 的深度（记作 depth(T)）是从其根节点到叶节点的最长路径长度。给定一个 Datalog 程序Σ、一个在扩展数据库Σ上的数据库 D，以及一个事实 $\alpha \in \Sigma(D)$，令 min-tree-depth(α, D, Σ) 为整数

$$\min\{\mathsf{depth}(T) \mid T \text{ is a proof tree of } \alpha \text{ w.r.t. } D \text{ and } \Sigma\},$$

也就是说，它是关于 D 和Σ的 α 所有证明树中的最小深度。最小深度证明树的概念如下：

定义 26 (最小深度证明树)。 考虑一个 *Datalog* 程序 Σ、一个在扩展数据库 (Σ) 上的数据库 D，以及一个在 模式 (Σ) 上的事实 α。关于 D 和Σ 的 α 最小深度证明树，是指一个关于 D 和 Σ 的 α 的证明树 T，满足深度(T)等于最小-树-深度(α, D, Σ)。

■

相对于最小深度证明树的 why-来源按预期定义。给定一个 Datalog 查询 $Q = (\Sigma, R)$、一个在扩展数据库(Σ)上的数据库 D，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，$t$ 关于 D 和 Q 相对于最小深度证明树的*why*-来源定义为事实集的集合

$$\{\mathsf{support}(T) \mid T \text{ is a minimal-depth proof tree of } R(\bar{t}) \text{ w.r.t. } D \text{ and } \Sigma\}$$

记作 why$_{\mathsf{MD}}(\bar{t}, D, Q)$。那么，算法问题

$$\mathsf{Why\text{-}Provenance}_{\mathsf{MD}}[\mathsf{C}] \quad \text{and} \quad \mathsf{Why\text{-}Provenance}_{\mathsf{MD}}[Q]$$

的定义与预期一致。我们将研究每个类 C ∈{Dat、LDat、NRDat}中 Why-Provenance$_{\mathsf{MD}}$[C] 的数据复杂度。正如任意证明树和非递归证明树的情形所示，对于递归查询，即使递归被限制为线性，该问题通常仍然是难处理的；而对于非递归查询，则非常易处理。我们首先关注递归查询。

C.1 递归查询

我们给出以下复杂度结果：

定理 27。 Why-Provenance$_{\mathsf{MD}}$[C] 在数据复杂度方面是NP困难的，对每个类C ∈ {Dat,LDat}均成立。

要证明定理 19，只需证明：

- Why-Provenance$_{\mathsf{MD}}$[Dat]在数据复杂度方面属于 NP。
- Why-Provenance$_{\mathsf{MD}}$[LDat]在数据复杂度方面是 NP 困难的。

我们首先关注上界。

上界

证明与 Why-Provenance$_{\mathsf{NR}}$[Dat] 中类似结果的证明相似（见定理 19）。给定一个 Datalog 程序 Σ、一个定义在扩展数据库(Σ)上的数据库 D，以及一个定义在模式(Σ)上的事实 α，我们首先定义关于 α 和 D 的 Σ的最小深度证明有向无环图 这一概念。随后，我们证明一个类似于命题 21 的结果：存在一个关于 D 和 Σ、支撑集(T) $= D' \subseteq D$ 的 α 最小深度证明树，当且仅当存在一个多项式大小的最小深度证明有向无环图 G，它关于 D 和 Σ、支撑集(G) $= D'$。这进而使我们能够设计一种在多项式时间内运行的猜测并验证算法。下面将这一高层次描述形式化。

深度的概念可以自然地推广到有根有向无环图。具体而言，对于有根有向无环图 G， G的深度 （记作 depth(G)）定义为从 G 的根节点到 G的叶节点的最长路径长度。给定一个 Datalog 程序 Σ、一个定义在扩展数据库(Σ)上的数据库 D，以及一个事实 $\alpha \in \Sigma(D)$，令 min-dag-depth(α, D, Σ) 为整数

$$\min\{\mathsf{depth}(G) \mid G \text{ is a proof DAG of } \alpha \text{ w.r.t. } D \text{ and } \Sigma\},$$

也就是说，相对于 D 和 Σ， α 的所有证明有向无环图中的最小深度。在介绍最小深度证明有向无环图之前，我们先确立 min-dag-depth(α, D, Σ) 的一个关键性质，它将在我们的复杂度分析中发挥重要作用。

命题 28. 考虑一个 *Datalog* 程序 Σ、一个基于 扩展数据库(Σ) 的数据库 D ，以及一个事实 $\alpha \in \Sigma(D)$。整数最小 -DAG-深度(α, D, Σ) 可在关于 $|D|$ 的多项式时间内计算。

证明。 该证明依赖于 Datalog 中众所周知的即时后果算子。粗略地说，该算子通过不同步骤构造出所有能够从 D 出发并应用 Σ 中规则推导出的事实。具体而言，每个事实都是“尽可能早地”构造出来的；我们将证明，事实 α 首次得到的步骤恰好等于 min-dag-depth(α, D, Σ)。

一个事实 $R(\bar{t})$ 是直接后果 D 和 Σ 如果

- $R(\bar{t}) \in D$，或

- there exists a Datalog rule $R(\bar{x}) :- R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$ in Σ and a function $h : \bigcup_{i\in[n]} \bar{x}_i \to \mathbf{C}$ such that $\{R_1(h(\bar{x}_1)), \ldots, R_n(h(\bar{x}_n))\} \subseteq D$ and $h(\bar{x}) = \bar{t}$.

Σ 的直接后果算子定义为函数 T_Σ，其作用域为 sch(Σ) 上数据库的集合 S，映射到 S

$$T_\Sigma(D) = \{R(\bar{t}) \mid R(\bar{t}) \text{ is an immediate consequence of } D \text{ and } \Sigma\}.$$

接下来我们定义

$$T^0_\Sigma(D) = D,$$

并且对每个 $i > 0$,

$$T^i_\Sigma(D) = T_\Sigma(T^{i-1}_\Sigma(D)).$$

最后，我们定义

$$T^\infty_\Sigma(D) = \bigcup_{i\geq 0} T^i_\Sigma(D).$$

不难看出

$$T^\infty_\Sigma(D) = T^{|\mathsf{base}(D,\Sigma)|}_\Sigma(D),$$

这又意味着，$T_\Sigma(D)$ 可以在关于 D 大小的多项式时间内计算；例如参见（Dantsin 等人，2001）。因此，对于每个 $i \geq 0$，$T^i_\Sigma(D)$ 都可以在关于 D 大小的多项式时间内计算。注意，$T^\infty_\Sigma(D)$ =Σ(D)（Abiteboul、Hull 和 Vianu，1995）。

上述讨论连同以下辅助引理将证明我们的论断。对于事实 $\alpha \in T^\infty_\Sigma(D)$，我们将 $\mathsf{rank}_{D,\Sigma}(\alpha)$ 记作整数 $\min\{i \mid \alpha \in T^i_\Sigma(D)\}$。

引理 29。对于每个事实 $\alpha \in T^\infty_\Sigma(D)$，秩$_{D,\,\Sigma}(\alpha)$ = 最小-DAG-深度(α, D, Σ)。

证明。考虑任意事实 $\alpha \in T^\infty_\Sigma(D)$，并令 $n = \mathsf{min\text{-}dag\text{-}depth}(\alpha, D, \Sigma)$。我们对 n 使用归纳法。

基础情形。对于 $n = 0$，根据证明有向无环图和 T_Σ的定义，结论立即成立。

归纳步骤。存在一个关于 D和 Σ、用于证明 α的证明有向无环图 $G = (V, E, \lambda)$，其深度为 $\mathsf{depth}(G) = n$。令 $u_1, \ldots, u_k$ 为根节点 v的出邻居，并令 $\lambda(u_i) = R_i(\bar{u}_i)$，对每个 $i \in [k]$均成立。由于对每个 $i \in [k]$，$\mathsf{depth}(G) = n$，$\mathsf{min\text{-}dag\text{-}depth}(R_i(\bar{u}_i)\ D, \Sigma) \leq \mathsf{depth}(G_i) < n$，其中 G_i是 G 中以 u_i为根节点的子有向无环图。根据归纳假设，对每个 $i \in [k]$，$\mathsf{rank}_{D,\Sigma}(R_i(\bar{u}_i)) \leq \mathsf{min\text{-}dag\text{-}depth}(R_i(\bar{u}_i)\ D, \Sigma)$，因此，$\{R_1(\bar{u}_1) \ldots, R_k(\bar{u}_k)\} \subseteq T^{n-1}_\Sigma(D)$。此外，根据证明有向无环图的定义，存在规则 $R_0(\bar{x}_0) :- R_1(\bar{x}_1) \ldots, R_k(\bar{x}_k)$，属于 Σ，以及函数 $h : \bigcup_{i\in[n]} \bar{x}_i \to \mathsf{C}$，使得 $\alpha = R_0(h(\bar{x}_0))$ 且对每个 $i \in [k]$，$h(\bar{x}_i) = \bar{u}_i$。因此，$\mathsf{rank}_{D,\Sigma}(\alpha) \leq \max_{i\in[k]}\{\mathsf{rank}_{D,\Sigma}(R_i(\bar{u}_i))\} + 1 \leq n$。我们进一步注意到 $\alpha \notin T^{n-1}_\Sigma(D)$，否则，根据归纳假设可得 $\mathsf{min\text{-}dag\text{-}depth}(\alpha, D, \Sigma) < n$，这与假设矛盾。因此，$\mathsf{rank}_{D,\Sigma}(\alpha) = n$。 □

该结论由引理 29 得出。 □

最小深度证明有向无环图的核心概念如下：

定义 30(最小深度证明有向无环图)。考虑一个 *Datalog* 程序 Σ、一个基于 扩展数据库(Σ)的数据库 D，以及一个基于 模式(Σ)的事实 α。关于 D和Σ 的 α 的最小深度证明有向无环图，是指关于 D 和 Σ 的 α 的证明有向无环图 G，且满足 深度(G)与 最小-有向无环图-深度(α, D, Σ)相同。 ■

与命题 21 类似的结果如下：

命题 31。对于 *Datalog* 程序 Σ，存在一个多项式 f，使得对于任意定义在 扩展数据库(Σ)上的数据库 D、定义在模式(Σ)上的事实α 以及 $D' \subseteq D$，以下条件等价：

1. 存在一个关于 α、相对于 D 和 Σ 的最小深度证明树 T，使得 支撑集(T) = D'。*2.* 存在一个关于 α、相对于 D 和 Σ 的最小深度证明有向无环图 $G = (V, E, \lambda)$，其 支撑集(G) = D'，且 $|V| \leq f(|D|)$。

方向 (2) 推出 (1) 的证明方法是将最小深度证明有向无环图 G“展开”为一个最小深度证明树 T，使得 support(G) = support(T)。我们采用与命题 5 中方向 (2) 推出 (1) 的证明相同的“展开”构造，该构造保持了深度的最小性。现在我们继续证明方向 (1) 推出 (2)。其基础构造分为两个主要步骤，由下文给出的引理 32 和 33 刻画。

- 首先，第一步是证明，关于 D 和 Σ、支撑集为 support(T) = D' 的 α 的最小深度证明树 T，可以转换为关于 D 和 Σ、支撑集为 support(T') = D' 的 α 的最小深度证明树 T'，且该证明树的子树数量“较小”。

引理 32。对于每个 *Datalog* 程序 Σ，都存在一个多项式 f，使得对于扩展数据库 扩展数据库(Σ)上的每个数据库 D、模式 模式(Σ)上的每个事实 α，以及每个 $D' \subseteq D$，如果存在一棵关于 D 和 Σ、其支撑集(T) = D'的 α 的最小深度证明树 T，那么也存在一棵这样的证明树 T'，满足子树数量(T') $\leq f(|D|)$。

证明。我们首先说明，证明树 T的深度“较小”。具体而言，根据引理 6，存在一个多项式 f，以及一棵关于 D 和 Σ 的 α 的证明树 T'，其深度 depth(T') $\leq f(|D|)$。由于根据假设，T 具有最小深度，因此我们得出 depth(T) $\leq f(|D|)$。另一个关键观察是，引理 7 背后的构造可将一棵深度“较小”的证明树转换为一棵支撑集相同、子树数量“较小”的证明树，且保留深度的最小性。可以验证，将引理 7 背后的构造应用于 T 后得到的证明树 T' 满足 depth(T') $\leq$ depth(T)。因此，由于 T 是一棵最小深度证明树，T'也同样如此。于是，我们可以将引理 7 的构造应用于最小深度证明树 T，得到一棵支撑集为 support(T) =support(T') 的最小深度证明树 T'，满足 scount(T') $\leq f(|D|)$，其中 f 是引理 7 给出的多项式。 □

- 第二步 表明，关于 α 相对于 D 和 Σ、支撑集为 support(T) = D' 且子树数量较少的最小深度证明树T，可以转换为关于 α 相对于 D 和 Σ、支撑集为 support(G) = D'的紧凑最小深度证明有向无环图 G。

引理 33. 对于每个 *Datalog* 程序 Σ和多项式f，都存在一个多项式 g，使得对于每个数据库 D

在 *edb(Σ)* 上，给定事实 α和 $D' \subseteq D$，如果存在一个关于 D 和 Σ 的 α 的最小深度证明树 T，其支撑集*(T)* 为 $=D'$，且 *scount(T)* 为 $\leq f(|D|)$，那么就存在一个关于 D和 Σ 的 α 的最小深度证明有向无环图 $G = (V, E, \lambda)$，其支撑集*(G)* 为 $= D'$，且 $|V| \leq g(|D|)$。

证明。 我们采用引理 8 中的构造，它将子树数量“较小”的证明树转换为一个具有相同支撑集、大小为多项式的非递归证明有向无环图，因为它保持了深度的最小性。 □

现在很清楚，命题 31 的方向 (1) 蕴含 (2) 是引理 32 和 33 的直接推论。

完成证明。现在我们可以完成以下论断的证明：Why-Provenance$_{\mathsf{MD}}$[Dat] 在数据复杂度意义下属于 NP。固定一个 Datalog 查询 $Q = (\Sigma, R)$。给定一个定义在扩展数据库 edb(Σ) 上的数据库D 、一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，以及 D 的一个子集 D'，要判定 $D' \in \mathsf{why}_{\mathsf{MD}}(\bar{t}, D, Q)$ 是否成立，只需检查是否存在一个相对于 D 和 Σ 的、以 $R(\bar{t})$ 为根的最小深度证明树 T，使得 support(*T*) $= D'$。根据命题 31，这等价于存在一个大小为多项式的最小深度证明有向无环图 G ，它相对于 D 和 Σ 以 $R(\bar{t})$ 为根，且 support(*G*) $= D'$。可以通过一个在数据库大小上以多项式时间运行的非确定性算法，检查是否存在这样的非递归证明有向无环图；证明 Why-Provenance[Dat] 在数据复杂度意义下属于 NP（定理 3）时，也是采用了这种方法。唯一的区别是，现在还需要额外检查猜测的有向无环图 G 是否具有最小深度，即 depth(*G*) 是否等于 min-dag-depth($R(\bar{t})$ D, Σ)。接下来需要说明，后者可以在多项式时间内完成。根据命题 28，min-dag-depth($R(\bar{t})$ D, Σ) 可以在多项式时间内计算。现在，将 G 转换为一个边带权图 G' ，为每条边赋予权重 -1 ，然后运行 Dijkstra 多项式时间算法，求出两个节点之间路径的最小权重，即可计算 depth(*G*)。[1]因此，Why-Provenance$_{\mathsf{MD}}$[Q] 属于 NP，进而 Why-Provenance$_{\mathsf{MD}}$[Dat]在数据复杂度意义下属于 NP。

下界

接下来，我们将证明 Why-Provenance$_{\mathsf{MD}}$[LDat] 在数据复杂度方面是 NP困难的。为此，我们需要证明存在一个线性 Datalog 查询 Q ，使得问题 Why-Provenance$_{\mathsf{MD}}$[Q] 是 NP困难的。证明通过从 3SAT 归约而来，其输入是一个 3CNF 布尔公式 $\varphi = C_1 \wedge \ldots \wedge C_m$，其中每个子句恰好有 3 个文字（一个布尔变量 v 或其否定 $\neg v$），问题是判断 φ是否可满足。

线性Datalog查询。 我们首先定义线性 Datalog 查询 $Q = (\Sigma, R)$。实际上，我们改编了 Q在证明 Why-Provenance[LDat] 在数据复杂度方面是 NP困难这一事实时所用的查询（见定理 3），使得每棵证明树的深度都相同。和往常一样，我们使用如果变量名称并不重要，则使用该名称；并在元组表达式中用分号分隔语义含义不同的项。程序 Σ 如下：

σ_1 : $R(x)$:– Var(x; y, ; z), 赋值(x, y, z),σ_2 : $R(x)$:– Var(x; y; z), 赋值 (x, y, z),σ_3 : 赋值 (x, y, z) :– NextC(x; z, w; k, ℓ),$C(x, y;$, ; , ; z, w; k, ℓ), 赋值 (x, y, w),σ_4 : 赋值 (x, y, z) :– NextC(x; z, w; k, ℓ),σ_5 : 赋值 (x, y, z) :– NextC(x; z, w; k, ℓ),$C($, ; , ; x, y; z, w; k, ℓ), 赋值(x, y, w),σ' : 赋值(x, y, z) :– NextC(x; z, w; y,), 赋值(x, y, w),σ'' : 赋值(x, y, z) :– NextC(x; z, w; y), σ_6 : 赋值 (x, z, w) :– Next(x, y; z, ; w) $R(y)$,σ_7 : 赋值(x, z, w) :– Next(x, y; z; w) $P(y)$,σ_8 : $R(x)$:– 最后(x).

$$C(_, _, x, y, _, _, z, w, k, \ell), \mathrm{Assign}(x, y, w),$$

$$\mathrm{Assign}(x, y, w),$$

与定理 3 的证明中所用的 Datalog 程序相比，关键区别在于，现在粗略地说，规则 σ_1，σ_2 还会将第一个子句的编号附加到变量赋值上；关系 Var 的最后一个位置存储第一个子句的编号。规则 σ_3，σ_4，σ_5，σ'，σ''中，σ_3、σ_4 和 σ_5 改编自先前的证明，而 σ'、σ'' 则是新增的；这些规则强制子树在依次执行这些规则时始终执行m “步”；NextC 为公式的子句提供了一种顺序。很容易验证，Σ 确实是一个线性 Datalog 程序。

从 3SAT 到 Why-来源$_{\mathsf{MD}}$[Q]。 现在，我们通过从 3SAT 归约来证明 Why-来源$_{\mathsf{MD}}$[Q] 是 NP困难的。考虑一个含有 n 个布尔变量 v_1、.....、 v_n的 3CNF布尔公式 $\varphi = C_1 \wedge \cdots \wedge C_m$。对于一个文字 ℓ，我们用 v(ℓ) 表示 ℓ中出现的变量；如果 ℓ 是变量（分别为变量的否定），则用 b(ℓ) 表示数字 1（分别为 0）。我们将 D_φ 定义为 edb(Σ) 上的数据库

$$\begin{aligned}
&\{\mathrm{Var}(v_i; 0, 1; 1) \mid i \in [n]\} \\
\cup\ &\{\mathrm{Next}(v_i, v_{i+1}; 0, 1; m+1) \mid i \in [n-1]\} \\
\cup\ &\{\mathrm{Next}(v_n, \bullet; 0, 1; m+1), \mathrm{Last}(\bullet)\} \\
\cup\ &\{C(\mathsf{v}(\ell_1), \mathsf{b}(\ell_1); \mathsf{v}(\ell_2), \mathsf{b}(\ell_2); \mathsf{v}(\ell_3), \mathsf{b}(\ell_3); i, i+1; 0, 1) \mid \\
&\qquad i \in [m] \text{ with } C_i = (\ell_1 \vee \ell_2 \vee \ell_3)\} \\
\cup\ &\{\mathrm{NextC}(v_i; j, j+1; 0, 1) \mid i \in [n] \text{ and } j \in [m]\}.
\end{aligned}$$

我们可以证明下一个引理，它基本上说明上述构造给出了一个从 3SAT 到 Why-Provenance[Q] 的正确多项式时间归约：

引理 34. 以下结论成立：

1. D_φ 可以在 φ的多项式时间内构造。 *2.* φ 可满足，当且仅当 $D_\varphi \in \mathsf{why}((v_1)\ D_\varphi, Q)$。

证明。 不难看出， D_φ 可以在关于 φ大小的多项式时间内构造出来。我们接下来

[1]我们回顾一下，Dijkstra 算法仅当输入图是有向无环图时，才能在具有任意整数边标签的图上正确运行。事实上，对于一般图，计算两个节点之间的最长路径长度是 NP困难的。

证明第（2）项。但我们先陈述并证明一个辅助技术引理，该引理的核心内容是，关于 D_φ 和 Σ 的 $R(v_1)$的所有证明树具有相同的深度。

引理 35. 对于关于 D_φ 和Σ的每棵 $R(v_1)$ T 证明树，深度(T) $= n \cdot (m+2) + 1$。

证明。关于 D_φ 和 Σ 的 $R(v_1)$ 证明树，其根节点 v 标记为 $R(v_1)$，因此，其子节点必然标记为通过规则σ_1 或 σ_2得到的事实。在这些子节点中，只有一个（设为 u）标记为内涵事实，其形式为 Assign($v_1, \cdot$, 1)（常量 1 是由于规则σ_1, σ_2的体部中出现变量 z）。于是，u 的子节点必然标记为通过 σ_3, $\sigma_4, \sigma_5, \sigma', \sigma''$ 之一得到的事实（这是因为标记 u的事实中有常量 1）。此外，由于每条此类规则的体部中都出现原子 NextC($x; z, w; k, \ell$)，u只有一个子节点（设为 u'）标记为形式为 Assign($v_1, \cdot$, 2) 的内涵事实。按照相同的推理，从这个标签为 Assign($v_1, \cdot$, 2) 的节点出发，将使用规则 $\sigma_3, \sigma_4, \sigma_5, \sigma'$, σ'' 之一，并且第三个位置上的 id 将再次递增，直到与 $m+1$ 一致。到此为止，树中从根节点出发的最长路径长度为 m+1。现在不难看出，由于原子 Next(x, y; $z, ; w$) 中出现变量 w，唯一可应用的规则是 σ_6 或 σ_7。应用其中一条规则后，唯一一个标记为内涵事实的节点将标记为事实 $R(v_2)$（若 $n > 1$），否则标记为事实 R(•)，而深度总计为 $m + 1 + 1 = m +$2。因此，接下来将再次使用 σ_1 或 σ_8。对 φ的所有变量应用上述推理，关于 D_φ 和 Σ 的 $R(v_1)$ 证明树的深度总计恰为$n \cdot (m+2) + 1$，证毕。

□

有了引理 35，要证明我们的结论，只需证明：φ 可满足，当且仅当存在一棵证明树 T（无论其深度如何），相对于 D_φ 和 Σ，它是 $R(v_1)$ 的证明树，且 support(T) $= D_\varphi$。要证明这一最后的结论，不难看出如何调整定理 3 的证明。主要区别在于，我们还需要论证：当 $R(v_1)$ 的一棵证明树使用规则 σ_3、σ_4、σ_5、σ'、σ'' 时，因此证明树中会有一个节点标记为形如 Assign(v_i, val j) 的事实，如果为 v_i 选择的真值 val 不能使子句 C_j 为真，那么证明树将沿着 σ' 或 σ''前进，具体取决于 val 的取值。因此，无论哪种情况，通过规则 σ_3、σ_4、σ_5、σ'、σ''，任何证明树都能够触及数据库中所有谓词为 NextC 的事实。由于这些是相对于定理 3 的证明中所用事实而言唯一新增的事实（所有其他事实都只做了些微修改），如果 φ可满足，那么就存在一棵证明树 T，它相对于 D_φ 和 Σ 是 $R(v_1)$ 的证明树，并满足 support(T) $= D_\varphi$；反之亦然。 □

根据引理 34，Why-Provenance$_{\mathsf{MD}}[Q]$是 NP困难的。因此，Why-Provenance$_{\mathsf{MD}}$[LDat]在数据复杂度方面是 NP困难的。

C.2 非递归查询

现在我们聚焦于非递归 Datalog 查询，并给出以下关于相对于最小深度证明树的 why-来源数据复杂度的结论：

定理 36. Why-Provenance$_{\mathsf{MD}}$[NRDat] 在数据复杂度下属于 AC_0。

这可以通过一阶可重写性来证明，方法与定理 9 中相同。不过需要强调的是，对于 Datalog 查询 Q，目标一阶查询无法像非递归证明树的情形那样（参见定理 25），通过简单地细化 cq(Q) 定义所依据的证明树空间来获得。关键原因在于，最小深度并非针对所有证明树而言，而是针对某个特定事实的证明树而言。这种对事实（以及间接地对数据库）的依赖，使我们无法像处理非递归证明树时那样，简单地考虑一个细化后的证明树族，因此需要稍微复杂一些的构造。实际上，我们将调整定理 9 所依据的一阶查询。

一阶重写。考虑一个非递归的 Datalog 查询 $Q = (\Sigma, R)$。回顾定理 9 的证明，对于一个合取查询 $\varphi(\bar{y}) \in \mathsf{cq}^{\approx}(Q)$，我们定义了一个一阶查询 $Q_{\varphi(\bar{y})} = \psi_{\varphi(\bar{y})}(x_1, \ldots, x_{\mathsf{ar}(R)})$，其中 $x_1, \ldots, x_{\mathsf{ar}(R)}$ 是互不相同的变量，且不会出现在 cq$^{\approx}$(Q) 中的任何合取查询里，并具有以下性质：对于每个数据库 D 以及每个元组 $\bar{t} \in \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，$\bar{t} \in Q_{\varphi(\bar{y})}(D)$ 当且仅当 $\bar{t}$ 是 $\varphi(\bar{y})$ 在 D 上的答案；此外，D 的所有原子都用于蕴含句子 $\varphi[\bar{y}/\bar{t}]$，也就是说，D 中没有其他事实，只有那些被用作 $\varphi[\bar{y}/\bar{t}]$ 中出现的原子的见证的事实。我们将把 $Q_{\varphi(\bar{y})}$ 扩展为 $Q^+_\varphi(\bar{y}) = \psi^+_\varphi(\bar{y})(x_1, \ldots, x_{\mathsf{ar}(R)})$，使其还能够执行最小深度检查。回顾一下，当 φ 的形式为 $\exists \bar{z}(R_1(\bar{w}_1) \wedge \cdots \wedge R_n(\bar{w}_n))$ 时，具有自由变量 $\bar{x} = (x_1, \ldots, x_{\mathsf{ar}(R)})$ 的公式 $\psi_{\varphi(\bar{y})}$ 具有如下形式

$$\exists \bar{y} \exists \bar{z} \left(\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \right),$$

其中 φ_1 定义为

$$\bigwedge_{i \in [n]} R_i(\bar{w}_i) \wedge (\bar{x} = \bar{y}) \wedge \bigwedge_{\substack{u, v \in \mathsf{var}(\varphi), \\ u \neq v}} \neg(u = v)$$

φ_2 定义为

$$\bigwedge_{P \in \{R_1, \ldots, R_n\}} \neg \left(\exists \bar{u}_P \left(P(\bar{u}_P) \wedge \bigwedge_{\substack{i \in [n], \\ R_i = P}} \neg(\bar{w}_i = \bar{u}_P) \right) \right)$$

并且 φ_3 定义为

$$\bigwedge_{P \in \mathsf{edb}(\Sigma) \setminus \{R_1, \ldots, R_n\}} \neg \left(\exists \bar{u}_P \, P(\bar{u}_P) \right).$$

现在，公式$\psi^+_\varphi(\bar{y})$ 是

$$\exists \bar{y} \exists \bar{z} \left(\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \right),$$

其中，附加合取项 φ_4 定义如下。对于合取查询 $\chi(\bar{s}) \in \mathsf{cq}^{\approx}(Q)$，我们用 depth($\chi(\bar{s})$ Q) 表示整数

$$\min\{\mathsf{depth}(T) \mid T \text{ is a } Q\text{-tree with } \chi(\bar{s}) \approx \mathsf{cq}(T)\}$$

也就是说，在所有诱导的合取查询同构于 $\chi(\bar{s})$ 的 Q-树中，深度最小的那个。现在，公式 φ_4 是

$$\bigwedge_{\substack{\xi(\bar{s})\in \mathsf{cq}^{\approx}(Q)\\ \text{with } \xi=\exists \bar{u}(P_1(\bar{v}_1),\ldots,P_m(\bar{v}_m)),\\ \mathsf{depth}(\xi(\bar{s}),Q)<\mathsf{depth}(\varphi(\bar{y}),Q)}} \neg\exists\bar{s}\exists\bar{u}\Big(\bigwedge_{i\in[m]} P_i(\bar{v}_i)\wedge (\bar{x}=\bar{s})\wedge \bigwedge_{\substack{u,v\in \mathsf{var}(\varphi),\\ u\neq v}} \neg(u=v)\Big)$$

这表明不存在其他深度小于 $\varphi(\bar{y})$深度的合取查询 $\xi(\bar{s})\in \mathsf{cq}^{\approx}(Q)$，并且它证明了存在一棵深度为 d 的事实 $R(\bar{t})$的证明树，相对于 D和 Σ而言，其中 $\bar{t}$是由（$x_1, \ldots, x_{\mathsf{ar}(R)}$）见证的元组。

将一阶查询$Q^{+}_{\varphi\ (\bar{y})}$ 用于每个合取查询 $\varphi(\bar{y})\in \mathsf{cq}^{\approx}(Q)$后，显然，所需的一阶查询$Q^{+}_{FO}$定义为 $\Phi^{+}(x_1\ldots, x_{\mathsf{ar}(R)})$，其中

$$\Phi^{+} = \bigvee_{\varphi(\bar{y})\in \mathsf{cq}^{\approx}(Q)} \psi^{+}_{\varphi(\bar{y})}.$$

接下来我们证明该构造的正确性。

引理 37。 给定一个非递归 *Datalog* 查询 $Q = (\Sigma, R)$、一个定义在扩展数据库edb（Σ）上的数据库 D、 $\bar{t}\in \mathsf{dom}\ (D)^{\mathsf{ar}(R)}$, 以及$D'\subseteq D$, $D'\in \mathsf{why}_{\mathsf{MD}}\ (\bar{t}, D, Q)$ 当且仅当 $\bar{t}\in Q^{+}_{FO}\ (D')$ 。

证明。 我们先考虑（⇒）方向。根据假设，存在一棵关于 D 和 Σ 的 $R(\bar{t})$ 的最小深度证明树 T，使得 $\mathsf{support}(T) = D'$。根据引理 12，我们得到由 T 诱导的合取查询$\varphi(\bar{y})\in \mathsf{cq}^{\approx}(Q)$ 满足 $\bar{t}\in Q_{\varphi(\bar{y})}(D')$。因此，要证明 $\bar{t}\in Q^{+}_{\varphi\ (\bar{y})}(D')$，只需证明$\bar{t}\in Q'(D')$，其中 $Q' = \varphi_4(x_1,\ldots,x_{\mathsf{ar}(R)})$，且 φ_4为

$$\bigwedge_{\substack{\xi(\bar{s})\in \mathsf{cq}^{\approx}(Q)\\ \text{with } \xi=\exists \bar{u}(P_1(\bar{v}_1),\ldots,P_m(\bar{v}_m)),\\ \mathsf{depth}(\xi(\bar{s}),Q)<\mathsf{depth}(\varphi(\bar{y}),Q)}} \neg\exists\bar{s}\exists\bar{u}\Big(\bigwedge_{i\in[m]} P_i(\bar{v}_i)\wedge (\bar{x}=\bar{s})\wedge \bigwedge_{\substack{u,v\in \mathsf{var}(\varphi),\\ u\neq v}} \neg(u=v)\Big).$$

反设 $\bar{t}\notin Q'(D')$。这又意味着存在一个形如 $\exists\bar{u}(P_1(\bar{v}_1)\ldots, P_m(\bar{v}_m))$ 的合取查询 $\xi(\bar{s})\in \mathsf{cq}^{\approx}(Q)$，其 $\mathsf{depth}(\xi(\bar{s}), Q) < \mathsf{depth}(\varphi(\bar{y}), Q)$，并且句子

$$\exists\bar{s}\exists\bar{u}\Big(\bigwedge_{i\in[m]} P_i(\bar{v}_i)\wedge(\bar{t}=\bar{s})\wedge \bigwedge_{\substack{u,v\in \mathsf{var}(\varphi),\\ u\neq v}} \neg(u=v)\Big)$$

由 D' 满足。这意味着存在一个对 ξ 中变量的赋值h，使得对于每个 $u, v\in \mathsf{var}(\xi)$ 且满足 $u\neq v$ 的情况，都有 $h(\bar{s}) = \bar{t}$, $h(u)\neq h(v)$; 并且对于每个 $i\in[m]$, 都有 $P_i(h(\bar{v}_i))\in D'$。由于 $\xi(\bar{s})$ 是由一个深度为 $d = \mathsf{depth}(\xi(\bar{s}), Q)$ 的 Q-树 T' 诱导的，因此 T' 是关于 D 和 Σ 的 $R(\bar{t})$ 的一棵深度为 d 的证明树，使得

$$\mathsf{support}(T') = \{P_1(h(\bar{v}_1)),\ldots,P_m(h(\bar{v}_m))\}\subseteq D'.$$

然而，由于

$$\mathsf{depth}(T') = \mathsf{depth}(\xi(\bar{s}), Q) < \mathsf{depth}(\varphi(\bar{y}), Q)$$

and

$$\mathsf{depth}(\varphi(\bar{y}), Q)\leq \mathsf{depth}(T),$$

我们得出 $\mathsf{depth}(T') < \mathsf{depth}(T)$，这与 T 是最小深度证明树这一事实相矛盾。

现在我们继续讨论方向 (⇐)。根据假设，$\bar{t}\in Q^{+}_{FO}(D')$。因此，存在一个合取查询 $\varphi(\bar{y})\in \mathsf{cq}^{\approx}(Q)$，使得 $\bar{t}\in Q^{+}_{\varphi\ (\bar{y})}(D')$。显然，$\bar{t}\in Q_{FO}(D')$；由引理 12 的证明可知，$\varphi(\bar{y})$ 是由一棵 Q-树 T 所诱导的，而该树也是关于 D 和 Σ、支撑集为$(T) = D'$ 的 $R(\bar{t})$ 的证明树。不失一般性地，我们假设 $\mathsf{depth}(T) = \mathsf{depth}(\varphi(\bar{y}) Q)$，即在诱导出 $\varphi(\bar{y})$ 的那些证明树中，T 的深度最小。接下来只需证明 T 是一棵最小深度证明树。为导出矛盾，假设 T 不是一棵最小深度证明树。因此，存在另一棵证明树 T'，它是关于 D 和Σ 的 $R(\bar{t})$ 的证明树，且 $\mathsf{depth}(T') < \mathsf{depth}(T)$。令由 T' 诱导的合取查询 $\xi(\bar{s})\in \mathsf{cq}^{\approx}(Q)$。显然，

$$\mathsf{depth}(\xi(\bar{s}), Q)\leq \mathsf{depth}(T') < \mathsf{depth}(\varphi(\bar{y}), Q).$$

这使我们可以得出$\bar{t}\notin Q^{+}_{\varphi\ (\bar{y})}(D')$，这与假设矛盾。因此，$T$ 是关于 D 和 Σ 的 $R(\bar{t})$ 的最小深度证明树。由于 $\mathsf{support}(T) = D'$，我们得到 $D'\in \mathsf{why}_{\mathsf{MD}}(R(\bar{t})\ D,\ Q)$，正如所需。

□

D 无歧义证明树

本节中，我们为第 5 节的所有论断提供证明，并进一步详细介绍我们的实验评估。

D.1 定理 14 的证明

我们首先证明定理 14，为方便起见，现重述如下:

定理 14。 以下结论成立:

1. 为何-来源$_{\mathsf{UN}}$[C] 在数据复杂性方面是 NP-完全的，对每个类别 C $\in$ {Dat,LDat}均如此。*2.* 为何-来源$_{\mathsf{UN}}$[NRDat]在数据复杂性方面属于 AC_0 类。

我们分别证明定理 14 的命题 (1) 和命题 (2)。首先，我们聚焦于命题 (1)。

命题 (1) 的证明

我们的主要任务是证明 Why-来源$_{\mathsf{UN}}$[Dat] 属于 NP。Why-来源$_{\mathsf{UN}}$[LDat] 的 NP困难性可由 Why-来源$_{\mathsf{NR}}$[LDat], 的 NP困难性推出，而后者已在 B节中证明。后者源于这样一个观察：对于线性Datalog程序，非递归证明树与无歧义证明树相同。现在我们证明 NP 上界。

这一结果基于如下刻画：事实 α 相对于数据库 D 和 Datalog 程序 Σ，在满足 $\mathsf{support}(T) = D'\subseteq D$ 的条件下，存在一棵无歧义证明树，当且仅当存在一个所谓的 无歧义证明有向无环图 G，它相对于 α、D 和 Σ，满足 $\mathsf{support}(G) = D'$，且大小为多项式大小。进而，我们可以设计一种猜测-检验算法

该算法在多项式时间内运行。接下来，我们将上述高层次描述形式化。

对于有根有向无环图 $G=(V,E,\lambda)$ 和节点 $v\in V$，我们用 $G[v]$ 表示以 v为根节点的 G 的子有向无环图。此外，若存在双射 $h: V\to V'$，使得对于每个节点 $v\in V$， $\lambda(v)=\lambda(h(v))$；并且对于任意两个节点 u、 $v\in V$，$(u,v)\in E$ 当且仅当 $(h(u)\ h(v))\in E'$，则称两个有根有向无环图 $G=(V,E,\lambda)$ 和 $G'=(V',E',\lambda')$ 同构，记作 $G\approx G'$。有了上述定义，我们现在可以引入无歧义证明有向无环图这一关键概念。

定义 *38*（无歧义证明有向无环图）。考虑一个 *Datalog* 程序 Σ、一个在扩展数据库 Σ 上的数据库 D，以及一个在模式 Σ 上的事实 α。对于 D 和 Σ，关于 α 的无歧义证明有向无环图是关于 D 和 Σ 的 α 的证明有向无环图 $G=(V,E,\lambda)$，且满足：对于所有 v，若 $u\in V, \lambda(u)=\lambda(v)$，则 $G[u]\approx G[v]$。 ■

现在我们可以给出我们的刻画。

命题 39。对于一个 *Datalog* 程序 Σ，存在一个多项式 f，使得对于任意一个定义在 扩展数据库(Σ)上的数据库 D、定义在模式(Σ)上的事实α 以及 $D'\subseteq D$，以下条件等价：
1. 存在一个 T 的无歧义证明树，关于 D和 Σ，且其支撑集$(T)=D'$。*2.* 存在一个无歧义证明有向无环图 $G=(V,E,\lambda)$，它是 α 关于 D 和 Σ 的证明，且其支撑集$(G)=D'$，并且 $|V|\le f(|D|)$。

证明。 我们首先证明 (1) 蕴含 (2)。设 T是 α 关于 D 和 Σ 的无歧义证明树，其支撑集为 support(T) $=D'$。根据定义， T 的子树数量是“较小的”；事实上，对于 T的每个标签 α，$|T[\alpha]_{/\approx}|=1$。接着，我们采用引理 8 背后的构造，将子树数量“较小”的证明树转换为一个具有相同支撑集且大小为多项式的证明有向无环图，因为该构造保持无歧义性。

对于 (2) 推出 (1)，我们采用“展开”构造来证明命题 5 中的 (2) 推出 (1)，因为它也保留了无歧义性。 □

完成证明。有了命题 39，证明定理 14 的第 (1) 项就很直接了。事实上，我们可以采用一种猜测-检验算法，其思路与用于证明定理 3 的 NP 上界的算法类似。关键区别在于，这里我们还需要验证猜测得到的有向无环图 G 是否无歧义。为此，只需在猜测图 G的同时，对于 G 中具有相同标签的每一对节点 u,v，猜测一个从 $G[u]$ 的节点到 $G[v]$的节点的双射 $h_{(u,v)}$。根据命题 39， G 的节点数相对于 $|D|$是多项式级的，因此需要猜测的双射数量相对于 $|D|$也是多项式级的。有了上述双射后，只需验证每个双射 $h_{(u,v)}$ 都是 $G[u]\approx G[v]$ 的见证。这个检查可以在多项式时间内轻松完成。

第（2）项的证明

这可通过一阶可改写性来证明，方法与定理 9 中所用的相同。事实上，目标一阶查询的构造与定理 9 的证明中完全相同，唯一的区别在于，对于 Datalog 查询 Q，集合 CQs cq(Q) 是通过仅考虑无歧义证明树来定义的，即该集合为 {cq(T) | T 是一棵无歧义 Q- 树 }。

D.2 命题15的证明

我们的目标是证明命题 15。但首先，我们需要引入一些辅助概念和结果，以便随后正式定义布尔公式 $\phi_{(\bar{t},D,Q)}$。接下来，我们将证明命题 15

更精细的刻画。 这里所讨论的布尔公式依赖于一种比命题 39 所给出的刻画更精细的刻画。为此，我们需要定义一种新型的图，用以见证无歧义证明树的存在。

定义40 (压缩有向无环图)。 考虑一个 *Datalog* 程序 Σ、一个关于 扩展数据库(Σ) 的数据库 D，以及一个关于模式(Σ) 的事实 α。α 关于 D 和 Σ 的压缩有向无环图 是一个有根有向无环图 $G=(V,E)$，带有 $V\subseteq$基(D,Σ)，满足：
1. G 的根是 α。*2.* 如果 $\beta\in V$是叶节点，那么 $\beta\in D$。*3.* 如果 $\beta\in V$有 $n\ge 1$ 条出边 (β,γ_1) ..... (β,γ_n)，那么存在一条规则 $R_0(\bar{x}_0)\text{ :- }R_1(\bar{x}_1)\text{.....},R_m(\bar{x}_m)\in\Sigma$，以及一个函数 $h: \bigcup_{i\in[m]}\bar{x}_i\to C$，使得 $\beta=R_0(h(\bar{x}_0))$，且 $\{\gamma_i\}_{i\in[n]}=\{R_i(h(\bar{x}_i))\mid i\in[m]\}$。 ■

压缩有向无环图可以看作一种类似证明有向无环图的结构，其中标记为同一事实的节点不超过一个。上述定义使我们能够将命题 39 中给出的刻画细化如下：对于一个未标记的有向无环图 $G=(V,E)$，稍微滥用一下记号，我们将 support(G) $=\{v\in V\mid v$ 是 $G\}$的叶节点。

命题41. 对于 *Datalog* 程序 Σ、定义在扩展数据库(Σ)上的数据库 D、定义在模式(Σ)上的事实 α，以及数据库 $D'\subseteq D$，以下条件等价：
1. 存在一个关于 D 和 Σ 的 α 的无歧义证明树 T，使得支撑集$(T)=D'$。*2.* 存在一个关于 D 和 Σ的 α 的压缩有向无环图 G，使得支撑集$(G)=D'$。

证明。我们先证明 (1) 推出 (2)。根据命题 39，只需证明：若存在一个关于 D 和 Σ 的 α 的无歧义证明有向无环图 $G'=(V',E',\lambda')$，使得 support(G') $=D'$，那么就存在一个关于 D 和 Σ 的 α 的压缩有向无环图$G=(V,E)$，且 support(G) $=D'$。假设 $G'=(V',E',\lambda')$ 是一个关于 D 和 Σ 的 α 的无歧义证明有向无环图，且 support(G') $=D'$。由于 G' 无歧义，对于任意两个非叶节点 u、 $v\in V'$，若 $\beta=\lambda'(u)=\lambda'(v)$，则必有 $S=\{\lambda'(u_1)\ldots,\lambda'(u_n)\}=\{\lambda'(v_1)\ldots,\lambda'(v_m)\}$，其中 u_1、...、u_n 和 v_1、...、v_m 分别是 G' 中 u 和 v 的子节点。因此，对于标记某个非叶节点的每个事实 β，我们将上述（唯一的）集合 S 称为 G' 中 β 的论证。于是，构造一个关于 D 和 Σ 的 α 的压缩有向无环图 $G=(V,E)$，使得 support(G) $=D'$，是直接的。具体而言， G 的根节点是 α；若某个节点为 $\beta\in V$，并令 $S=\{\gamma_1,\ldots,\gamma_n\}$ 为其在 G'中的论证，则 G 的节点为 γ_1、...、 γ_n，其边为 (β,γ_1) ...，(β,γ_n)。

为证明（2）蕴含（1），我们采用一种“展开”构造，类似于命题 5 的证明中将证明有向无环图转换为证明树时所用的方法。具体而言，考虑一个关于 D and $G=$、支撑集为 V 的压缩有向无环图 $G=(V,E)$。

根据 G 的定义，对于 G 中每个非叶节点 β，其在 G 中的子节点 γ_1、.....、γ_n 满足：存在一条规则 $R_0(\bar{x}_0) :- R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m) \in \Sigma$，以及一个函数 $h : \bigcup_{i\in[m]} \bar{x}_i \to \mathsf{C}$，使得 $\beta = R_0(h(\bar{x}_0))$，且 $\{\gamma_i\}_{i\in[n]} = \{R_i(h(\bar{x}_i)) \mid i \in [m]\}$；对于上文任意选取的一对 (σ, h)，我们称 (σ, h) 为 G 中 β 的触发器。我们将 G 展开为关于 D and Σ、α 的无歧义证明树 $T = (V', E', \lambda')$，具体如下。我们向 T 添加一个节点 v，其标签为 $\lambda'(v) = \alpha$。接着，若 T 中的节点 v 带有标签 $\lambda'(v) = \beta$，令 (σ, h) 为 G 中 β 的触发器，其中 $\sigma = R_0(\bar{x}_0) :- R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$，则向 T 添加 m 个全新节点 u_1、.....、u_m；其中 u_i 的标签为 $\lambda'(u_i) = R_i(h(\bar{x}_i))$，适用于 $i \in [m]$，并向 T 添加边 $(v, u_1), \ldots, (v, u_m)$。根据构造，$T$ 是关于 D and Σ、α 的证明树。为说明 T 是无歧义的，注意根据压缩有向无环图的定义以及 T 的构造，对于 T 中任何两个标签相同的非叶节点 u、v，u 和 v 的子节点数量相同，分别为 u_1、.....、u_n 和 v_1、.....、v_n，并且对于 $i \in [n]$，$\lambda'(u_i) = \lambda'(v_i)$。显然，$\mathsf{support}(T) = D'$。

□

有了上述刻画，我们现在可以讨论如何构造布尔公式。

规则实例图与向下闭包。就我们的目的而言，有向超图 $\mathcal{H}$ 是一对 (V, E)，其中 V 是 $\mathcal{H}$ 的节点集合，E 是其超边集合，即形如 (α, T) 的有序对，其中 $\alpha \in V$，且 $\emptyset \subsetneq T \subseteq V$。对于 $\mathcal{H}$ 中的两个节点 u, v，如果 $u = v$，或者存在一系列形如 $(u_1, T_1), \ldots, (u_n, T_n)$ 的超边，满足 $u_1 = u, v \in T_n$，且对于 $1 < i \leq n$ 有 $u_i \in T_{i-1}$，我们就称 u 在 $\mathcal{H}$ 中可达 v。对于 $u \in V$，我们将由 $\mathcal{H}$ 得到的超图 (V', E') 记为 $\mathcal{H}_{\downarrow u}$，其中 V' 包含 u 以及所有可从 u 到达的节点，而超边则是 $\mathcal{H}$ 中所有涉及 V' 的超边。现在我们可以引入规则实例图的概念。

定义42 (规则实例图)。 考虑一个 *Datalog* 程序 Σ，以及一个定义在 扩展数据库(Σ)上的数据库 D。D 和 Σ 的规则实例图（GRI）是超图$\mathsf{gri}(D, \Sigma) = (V, E)$，其中 $V \subseteq \mathsf{base}(D, \Sigma)$，满足

1. 对每个 $\alpha \in D, \alpha \in V$。 *2.* 如果存在一条规则 $R_0(\bar{x}_0) : - R_1(\bar{x}_1) .., R_n(\bar{x}_n)$ 属于Σ，以及一个函数 $h : \bigcup_{i\in[n]} \bar{x}_i \to \mathbf{C}$，使得 $\alpha_i = R_i(h(\bar{x}_i)) \in V$，对于 $i \in [n]$，则 $\alpha_0 = R_0(h(\bar{x}_0)) \in V$，且 $(\alpha_0, \{\alpha_1, \ldots, \alpha_n\}) \in E$。

粗略地说，$\mathsf{gri}(D, \Sigma)$ 是一种结构，它“包含”了相对于 D 和 Σ 的 α 的所有可能压缩有向无环图。由于我们只关心找到特定事实 α 的压缩有向无环图，因此不必考虑完整的 $\mathsf{gri}(D, \Sigma)$，只需考虑 $\mathsf{gri}(D, \Sigma)$ 中包含 α 及其所有可达节点的子超图。形式化地，对于 Datalog 程序 Σ、定义在 $\mathsf{edb}(\Sigma)$ 上的数据库 D，以及 $\mathsf{gri}(D, \Sigma)$ 中的节点（即事实）α，α 相对于 D 和 Σ 的向下闭包是超图 $\mathsf{down}(D, \Sigma, \alpha) = \mathsf{gri}(D, \Sigma)_{\downarrow\alpha}$。换言之，向下闭包只保留 $\mathsf{gri}(D, \Sigma)$ 中与推导事实 α 相关的部分。容易验证，$\mathsf{down}(D, \Sigma, \alpha)$“包含”相对于 D 和 Σ 的 α 的所有压缩有向无环图，因此可得下一个技术结果：

引理 43。 考虑一个 *Datalog* 程序 Σ、定义在 扩展数据库(Σ) 上的数据库 D、定义在模式(Σ) 上的事实 α，以及相对于 D 和 Σ 的 α 的压缩有向无环图$G = (V, E)$。那么，对于 G 中每个具有出边 $(\beta, \gamma_1), \ldots, (\beta, \gamma_n)$ 的节点 $\beta \in V$，$(\beta, \{\gamma_1, \ldots, \gamma_n\})$ 都是 $\mathsf{down}(D, \Sigma, \alpha)$ 的一条超边。

布尔公式。 现在我们可以介绍所需的布尔公式了。对于 Datalog 查询 $Q = (\Sigma, R)$、定义在 $\mathsf{edb}(\Sigma)$ 上的数据库 D，以及元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，我们可以在 $|D|$ 中用多项式时间构造公式 $\phi_{(\bar{t},D,Q)}$，使得相对于无歧义证明树，t 关于 D 和 Q 的 why-来源可以根据使 $\phi_{(\bar{t},D,Q)}$ 为真的真值赋值计算得出。

令 $\mathsf{down}(D, Q, R(\bar{t})) = (V, E)$。$\phi_{(\bar{t},D,Q)}$ 的布尔变量集合由四个互不相交的变量集合 V_N、V_H、V_E 和 V_C 组成。V_N 中的每个变量都对应于 $\mathsf{down}(D, Q, R(\bar{t}))$ 的一个节点，即 $V_N = \{x_\alpha \mid \alpha \in V\}$；$V_H$ 中的每个变量都对应于 $\mathsf{down}(D, Q, R(\bar{t}))$ 的一条超边，即 $V_H = \{y_e \mid e \in E\}$；而 V_E 中的每个变量都对应于一条可从 $\mathsf{down}(D, Q, R(\bar{t}))$ 的某条超边中提取的“标准边”，即 $V_E = \{z_{(\alpha,\beta)} \mid (\alpha, T) \in E, \text{其中 } \beta \in T\}$。集合 V_C 将在后文讨论。大致而言，通过 $\phi_{(\bar{t},D,Q)}$ 的满足赋值而取真的 V_N 和 V_E 中的变量，将导出压缩有向无环图 G 的节点和边，该图对应于相对于 D 和 Q 的 $R(\bar{t})$；根据命题 41，这将推出 $\mathsf{support}(G) \in \mathsf{why}^{\mathsf{UN}}(\bar{t}, D, Q)$。

The formula $\phi_{(\bar{t},D,Q)}$ is of the form

$$\phi_{graph} \wedge \phi_{root} \wedge \phi_{proof} \wedge \phi_{acyclic}.$$

接下来，我们将讨论上述各个公式。下文中，我们用 $A \Rightarrow B$ 作为 $\neg A \vee B$的简写。第一个公式 ϕ_{graph} 负责确保 V_N 中变量的真值赋值与 V_E中变量的真值赋值保持一致；也就是说，如果某条边被声明为 G的一部分，那么这条边连接的两个节点也必须属于 G:

$$\phi_{graph} = \bigwedge_{z_{(\alpha,\beta)} \in V_E} (z_{(\alpha,\beta)} \Rightarrow x_\alpha) \wedge (z_{(\alpha,\beta)} \Rightarrow x_\beta).$$

第二个公式 ϕ_{root} 确保原子 $R(\bar{t})$确实是 G中的一个节点，是 G的根节点，并且其他任何作为 G 中节点的原子都不能是根节点（也就是说，它必须始终至少有一条入边）：

$$\phi_{root} = x_{R(\bar{t})} \wedge \left(\bigwedge_{z_{(\alpha,R(\bar{t}))} \in V_E} \neg z_{(\alpha,R(\bar{t}))} \right) \wedge \bigwedge_{\substack{x_\alpha \in V_N \\ \text{with } \alpha \neq R(\bar{t})}} \left(x_\alpha \Rightarrow \bigvee_{z_{(\beta,\alpha)} \in V_E} z_{(\beta,\alpha)} \right).$$

现在我们转到下一个公式 ϕ_{proof}。大致来说，该公式负责确保：每当内涵原子 α 是 G 中的一个节点时，它在 G 中就必须有正确的子节点。也就是说，它的子节点是来自 $\mathsf{down}(D, Q, R(\bar{t}))$ 中某条超边的节点，除此之外没有其他节点是它的子节点（这是确保 G 是一个压缩有向无环图所必需的）。这通过两个子公式实现。

第一部分负责为每个内涵原子 α 选择 $\mathsf{down}(D,\ Q,\ R(\bar{t}))$ 中的某条超边 $(\alpha,\ T)$；第二部分则确保，对于每条选定的超边 $(\alpha,\ T)$（每个内涵原子 α 对应一条），且仅有 T 中的节点是 G 中 α 的子节点：

$$\phi_{proof} = \bigwedge_{\substack{x_\alpha \in V_N \text{ with} \\ \alpha \text{ intensional}}} \left(x_\alpha \Rightarrow \bigvee_{y_{(\alpha,T)} \in V_H} y_{(\alpha,T)} \right) \wedge \bigwedge_{\substack{y_e \in V_H \\ \text{with } e=(\alpha,T)}} \left(\bigwedge_{z_{(\alpha,\beta)} \in V_E} y_e \Rightarrow \ell_{e,\beta} \right),$$

其中，对于超边 $e = (\alpha, T)$，$\ell_{e,\beta}$ 表示：若 $\beta \in T$，则为 $z_{(\alpha,\beta)}$；否则为 $\neg z_{(\alpha,\beta)}$。

备注。 虽然我们感兴趣的是为每个内涵原子 α选择恰好一个超边 (α, T)，但上述公式使用的是简单析取，而非异或。这没有问题，因为任何使两个变量 $y_{(\alpha,T_1)}$ 和 $y_{(\alpha,T_2)}$ 为真的真值赋值都不可能是满足赋值，因为 ϕ_{proof}中的第二个子公式（例如）要求变量 $z_{(\alpha,\beta)}$（其中 $\beta \in T_1$）为真，而所有其他变量都必须为假。因此，由于 $T_1 \neq T_2$，在考虑超边 (α, T_2) 时，这个子公式将不满足。

剩下要讨论的公式是 $\phi_{acyclic}$。这个最后的公式负责检查 G，也就是说，检查其边对应于 V_E 中真变量的图是否无环。通过布尔公式中的布尔变量对图进行编码后，再检查图的无环性，是 SAT 文献中研究得很充分的问题，因此存在多种不同的编码。对于我们的构造而言，使用最简单的编码就足够了（尽管它在实践中效率不高）；该编码只对图的传递闭包进行编码。不过，在实验评估中，我们将使用一种基于所谓顶点消除的更高效编码来实现该公式，这种编码可使公式规模降低数个数量级 $\phi_{acyclic}$ (Rankooh 和 Rintanen，2022)。

为了编码传递闭包，我们现在需要使用布尔变量集合 V_C，其中对于 $\mathsf{down}(D, Q, R(\bar{t}))$ 中的每两个节点 α, β，都有一个形如 $t_{(\alpha,\beta)}$ 的变量。直观地说，$t_{(\alpha,\beta)}$ 表示在 G 中是否存在一条从 α 到 β 的路径。有了这些变量，编写 $\phi_{acyclic}$ 就很简单了：它只需对底层图的传递闭包进行编码，然后检查是否不存在环：

$$\phi_{acyclic} = \left(\bigwedge_{z_{(\alpha,\beta)} \in V_E} z_{(\alpha,\beta)} \Rightarrow t_{(\alpha,\beta)} \right) \wedge \left(\bigwedge_{z_{(\alpha,\beta)} \in V_E, t_{(\beta,\gamma)} \in V_C} z_{(\alpha,\beta)} \wedge t_{(\beta,\gamma)} \Rightarrow t_{(\alpha,\gamma)} \right) \wedge \left(\bigwedge_{t_{(\alpha,\alpha)} \in V_C} \neg t_{(\alpha,\alpha)} \right).$$

至此，我们完成了布尔公式的构造。容易验证，该公式属于合取范式，并且可以在多项式时间内构造出来。接下来我们证明其正确性，即命题 15。对于从 $\phi_{(\bar{t},D,Q)}$ 到 $\{0,1\}$ 的变量中选取的真值赋值 τ，我们将数据库 $\{\alpha \in D \mid x_\alpha \in V_N \text{ 和 } \tau(x_\alpha) = 1\}$ 记作 $\mathsf{db}(\tau)$，也就是说，该数据库收集了所有对应变量在 $\phi_{(\bar{t},D,Q)}$ 中相对于 τ 为真的事实。最后，我们令 $[\![\phi_{(\bar{t},D,Q)}]\!]$ 为

$$\left\{ \mathsf{db}(\tau) \mid \tau \text{ is a satisfying assignment of } \phi_{(\bar{t},D,Q)} \right\}.$$

现在我们可以证明命题 15。为方便起见，我们在此重新列出该命题：

命题 15。 考虑一个 *Datalog* 查询 $Q = (\Sigma, R)$、一个定义在扩展数据库(Σ) 上的数据库 D，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$。则有 $\mathsf{why^{UN}}(\bar{t}, D, Q) = [\![\phi_{(\bar{t},D,Q)}]\!]$。

证明。 根据命题 41，只需证明：

引理 44. 对于每个数据库 $S \subseteq D$，以下条件等价：

1. 存在一个关于 D 和 Σ 的 $R(\bar{t})$ 的压缩有向无环图 G，其支撑集为 $G) = S$。*2.* 存在 $\phi_{(\bar{t},D,Q)}$ 的一个满足真值赋值 τ，使得 数据库$(\tau) = S$。

证明。 我们先证明蕴含关系 (1) $\Rightarrow$ (2)。假设 $G = (V, E)$ 是关于 D 和Σ 的 $R(\bar{t})$ 的压缩有向无环图，且其支撑集为 $\mathsf{support}(G) = S$。我们构造 $\phi_{(\bar{t},D,Q)}$ 的如下真值赋值 τ：

- 对于每个 $x_\alpha \in V_N$，如果 $x_\alpha(x_\alpha)$ 是 G 的一个节点，则 $\tau(x_\alpha) = 1$；否则 $\tau(x_\alpha) = 0$。
- 对于每个 $z_{(\alpha,\beta)} \in V_E$，如果 G 中存在一条边 (α, β)，则 $\tau(z_{(\alpha,\beta)}) = 1$；否则 $\tau(z_{(\alpha,\beta)}) = 0$。
- 对于每个 $y_{(\alpha,T)} \in V_H$, $\tau(y_{(\alpha,T)}) = 1$，如果 α 是 G 中一个具有出边 $(\alpha, \beta_1) \ldots, (\alpha, \beta_n)$ 的节点，且满足 $T = \{\beta_1, \ldots, \beta_n\}$，否则为 $\tau(y_{(\alpha,T)}) = 0$。
- 对于每个 $t_{(\alpha,\beta)} \in V_C$, $\tau(t_{(\alpha,\beta)}) = 1$，如果在 G 中存在一条从 α 到 β 的路径，否则为 $\tau(t_{(\alpha,\beta)}) = 0$。

现在我们声称 τ 使 $\phi_{(\bar{t},D,Q)}$ 为真，且 $\mathsf{db}(\tau) = S$。

观察 1。 根据引理 43，G 中的每个节点 α 都有一个变量 x_α 属于 $\phi_{(\bar{t},D,Q)}$。类似地，如果节点 α 在 G 中有出边 $(\alpha, \beta_1) \ldots, (\alpha, \beta_n)$，那么对于 $i \in [n]$，$z_{(\alpha,\beta_i)}$，以及$y_{(\alpha,T)}$（其中 $T = \{\beta_1, \ldots, \beta_n\}$），都是 $\phi_{(\bar{t},D,Q)}$的变量。

根据观察 1，每个事实 $\alpha \in$ 支撑集$(G) = S$ 都有一个对应的变量 x_α 位于 $\phi_{(\bar{t},D,Q)}$中。此外，根据 τ的构造，$\tau(x_\alpha) = 1$ 对每个 $\alpha \in S$都成立。进一步，对于 D 中不属于 S 的所有事实 β，这意味着 G中没有标记为 β的节点，因此 $\tau(x_\beta) = 0$。由此，$\mathsf{db}(\tau) = S$。

现在我们证明 τ 使 $\phi_{(\bar{t},D,Q)}$ 为真。我们将分别考察它的各个子公式。

ϕ_{graph}。 </style}显然，τ 使这个公式成立，因为当$\tau(z_{(\alpha,\beta)}) = 1$时，这意味着 α 和 β 是 G的节点，因此根据 τ的构造，$\tau(x_\alpha) = \tau(x_\beta) = 1$。

ϕ_{root}。 </style}由于 $R(\bar{t})$是 G的根节点，根据观察 1以及 τ的构造， $\tau(x_{R(\bar{t})}) = 1$。要理解这个公式为何

$$\left(\bigwedge_{z_{(\alpha,R(\bar{t}))} \in V_E} \neg z_{(\alpha,R(\bar{t}))} \right)$$

为真，因为 $R(\bar{t})$ 是根节点，没有任何入边，因此根据 τ 的构造，对于每个 $z_{(\alpha,R(\bar{t}))} \in V_E$， $\tau(z_{(\alpha,R(\bar{t}))}) = 0$。最后，关于公式

$$\bigwedge_{\substack{x_\alpha \in V_N \\ \text{with } \alpha \neq R(\bar{t})}} \left(x_\alpha \Rightarrow \bigvee_{z_{(\beta,\alpha)} \in V_E} z_{(\beta,\alpha)} \right),$$

如果对于某个 $\alpha \neq R(\bar{t})$， $\tau(x_\alpha) = 1$，那么根据 τ 的构造， α 是 G 的一个节点。根据观察 1，对于 G 中的每条边 (β, α)，我们有 $z_{(\beta,\alpha)}$ 是 $\phi_{(\bar{t},D,Q)}$ 的一个变量，且 $\tau(z_{(\beta,\alpha)}) = 1$，因此该析取式为真。

ϕ_{proof}。 我们首先考虑公式

$$\bigwedge_{\substack{x_\alpha \in V_N \text{ with} \\ \alpha \text{ intensional}}} \left(x_\alpha \Rightarrow \bigvee_{y_{(\alpha,T)} \in V_H} y_{(\alpha,T)} \right).$$

如果对于某个内涵事实 α， $\tau(x_\alpha) = 1$，这意味着 α 是 G 的一个节点。设 (α, β_1) ...，(α, β_n) 是 G 中 α 的出边。根据观察 1， $y_{(\alpha,T)}$ （其中 $T = \{\beta_1, \ldots, \beta_n\}$）是 $\phi_{(\bar{t},D,Q)}$ 的一个变量；根据 τ 的构造， $\tau(y_{(\alpha,T)}) = 1$，因此该析取式为真。现在我们考虑公式

$$\bigwedge_{\substack{y_e \in V_H \\ \text{with } e=(\alpha,T)}} \left(\bigwedge_{z_{(\alpha,\beta)} \in V_E} y_e \Rightarrow \ell_{e,\beta} \right).$$

如果 $\tau(y_e) = 1$，对于 $e = (\alpha, T)$，那么 α是 G中的一个节点，其出边为 $(\alpha, \beta_1), \ldots, (\alpha, \beta_n)$，其中 $T = \{\beta_1, \ldots, \beta_n\}$。根据观察 1， $z_{(\alpha,\beta_i)} \in V_E$，对于 $i \in [n]$；根据 τ的构造， $\tau(z_{(\alpha,\beta_i)}) = 1$，对于 $i \in [n]$。因此，所有形如 $y_e \Rightarrow \ell_{e,\beta}$ 的蕴含式（其中 $\beta \in \{\beta_1, \ldots, \beta_n\}$）都成立。至于其他蕴含式，即当 $\beta \notin \{\beta_1, \ldots, \beta_n\}$ 时，由于 α在 G中没有其他出边，根据 τ 的构造，对于所有其他事实 $\beta \notin \{\beta_1, \ldots, \beta_n\}$， $\tau(z_{(\alpha,\beta)}) = 0$。因此，整个公式都成立。

$\phi_{acyclic}$。 </style}公式为真的事实源于 G 的无环性、观察 1 以及 τ 的构造。

现在我们继续证明 (2) $\Rightarrow$ (1)。根据假设，存在一个真值赋值 τ，使得 $\phi_{(\bar{t},D,Q)}$ 为真且 $\mathsf{db}(\tau) = S$。我们定义辅助集合

$$\begin{aligned} \mathsf{nodes}(\tau) &= \{\alpha \mid x_\alpha \in V_N \text{ and } \tau(x_\alpha) = 1\} \\ \mathsf{edges}(\tau) &= \{(\alpha, \beta) \mid z_{(\alpha,\beta)} \in V_E \text{ and } \tau(z_{(\alpha,\beta)}) = 1\}. \end{aligned}$$

由于 τ使得 ϕ_{graph}为真，$\mathsf{edges}(\tau)$ 中出现的每个事实也都出现在 $\mathsf{nodes}(\tau)$ 中；因此， $G = (\mathsf{nodes}(\tau), \mathsf{edges}(\tau))$ 是一个定义良好的有向图。此外，由于 $\mathsf{db}(\tau) = S$， G 中所有没有出边的节点恰好就是 S 中的节点。最后，由于 τ使得 $\phi_{acyclic}$为真， G 是无环的；又由于 τ满足 ϕ_{root}， $R(\bar{t})$ 是G 中唯一没有入边的节点。因此， G 是一个有向无环图，其根节点是$R(\bar{t})$，叶节点恰好构成集合 S。剩下要论证的是， G 是相对于 D 和 Σ而言的 $R(\bar{t})$ 的一个压缩有向无环图。

考虑 G 中的一个节点 α，它是一个内涵事实。显然， $\tau(x_\alpha) = 1$，因此 ϕ_{proof} 中的析取式必须为真。于是，对于向下闭包 $\mathsf{down}(D, Q, R(\bar{t}))$ 中的某条超边 (α, T)，有 $\tau(y_{(\alpha,T)}) = 1$。特别地，由于 τ 使得

$$\bigwedge_{\substack{y_e \in V_H \\ \text{with } e=(\alpha,T)}} \left(\bigwedge_{z_{(\alpha,\beta)} \in V_E} y_e \Rightarrow \ell_{e,\beta} \right)$$

在 ϕ_{proof} 中为真，且 $\tau(y_{(\alpha,T)}) = 1$，我们必须有： τ 将 1 赋给所有形如 $z_{(\alpha,\beta)}$ 且满足 $\beta \in T$ 的变量，并将 0 赋给所有其他形如 $z_{(\alpha,\beta)}$ 且满足 $\beta \notin T$ 的变量。这意味着不存在其他形如 $y_{(\alpha,T')}$ 且满足 $T' = T$ 的变量被 τ 赋值为 1。因此，对于 G 中的每个内涵节点 α， α 都有出边 $(\alpha, \beta_1), \ldots, (\alpha, \beta_n)$，并且存在一条属于 $\mathsf{down}(D, Q, R(\bar{t}))$、形如 (α, T) 的超边，使得 $T = \{\beta_1, \ldots, \beta_n\}$。根据向下闭包的定义，后者意味着对于 G 中的每个内涵节点 α， α 都有出边 $(\alpha, \beta_1), \ldots, (\alpha, \beta_n)$，并且存在一条形如 $R_0(\bar{x}_0) \coloneq R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$ 的规则 $\sigma \in \Sigma$，满足 $m \geq n$；此外还存在一个函数 $h : \bigcup_{i \in [n]} \bar{x}_i \to \mathsf{C}$，使得 $R_0(h(\bar{x}_0)) = \alpha$，且$\{R_1(h(\bar{x}_1)), \ldots, R_m(h(\bar{x}_m))\} = \{\beta_1, \ldots, \beta_n\}$。因此， G 是关于 D 和 Σ的 $R(\bar{t})$ 的压缩有向无环图，正如所需。 □

Pro命题 15 可直接由引理 4 得出 4. □

D.3 实现细节

本节将进一步阐述论文正文中关于我们实现的讨论。下文中，固定一个 Datalog 查询 $Q = (\Sigma, R)$、一个定义在扩展数据库 Σ 上的数据库 D，以及一个元组 $\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$。

构造向下闭包。 回想一下， $\phi_{(\bar{t},D,Q)}$ 的构造依赖于相对于 D 和 Σ的 $R(\bar{t})$向下闭包。事实证明，只需执行一个经过轻微修改的 Datalog 查询 $Q_\downarrow$，并使用一个经过轻微修改的数据库 $D_\downarrow$，即可计算向下闭包的超边。换言之，在 $D_\downarrow$ 上执行 $Q_\downarrow$ 所得的答案与向下闭包的超边一致。为此，我们将对（Elhalawati、Kr¨otzsch 和 Mennicke，2022）中提出的一种现有方法稍作修改。在那篇论文中，作者研究了相对于标准证明树计算 Datalog 查询的 why-来源问题。然而，除了 $\mathsf{down}(D, Q, R(\bar{t}))$ 的构造外，他们计算一般树支撑集的方法与我们的方法有根本区别，因为他们采用的是基于存在规则的引擎，而非 SAT 求解器；目前尚不清楚如何将他们的方法改编用于我们的目的，因为我们需要检查底层树是否无歧义。此外，该方法

（Elhalawati、Kr¨otzsch 和 Mennicke，2022）会一次性计算全部支持集，而我们基于 SAT 求解器的方法可以枚举支持集，因此能够逐步构造 why-来源。尽管如此，两个方法都会构造 down($D, Q, R(\bar{t})$)，因此我们借鉴了（Elhalawati、Kr¨otzsch 和 Mennicke，2022）中的相关技术，并将在下文简要回顾。

主要思路是使用现有的 Datalog 引擎，计算查询 $Q_\downarrow$ 从 Q 经由 $D_\downarrow$的轻微修改版本 D得到的答案； $Q_\downarrow(D_\downarrow)$ 中的答案将与 down(D， Q， $R(\bar{t})$) 的所有超边一致。为此， $Q_\downarrow$ 的规则包含 Σ中的所有规则，这些规则负责推导 gri(D， Σ) 的所有节点；此外还有一组规则，负责使用这些节点构造 down(D， Q， $R(\bar{t})$) 的所有超边。形式化地，令 ω 为 Σ中谓词的最大元数，并令b 为 Σ中规则体所含原子的最大数量。我们定义两个新谓词：

• 元数为 $\omega + 1$ 的 CurNode，用于存储在评估 $Q_\downarrow$期间正在处理的 down(D， Q， $R(\bar{t})$) 当前节点。

•元数为 $(\omega+1) + b \times (\omega+1)$ 的 HEdge，用于存储在评估 $Q_\downarrow$ 期间正在构造的超边。

此外，对于原子 $\alpha = P(\bar{u})$，我们将 $\langle\alpha\rangle$ 表示为长度为 $\omega + 1$ 的元组，其形式为 c_P， $\bar{u}$， $\star$， . . .， $\star$，其中 c_P 和 $\star$ 是不属于 D 的常量。直观地说， $\langle\alpha\rangle$ 将原子 α 编码为一个固定长度的元组。最后，对于原子 α 和原子序列 β_1， . . .， β_n，其中 $n \leq b$，我们用 $\langle\alpha, \beta_1, \ldots, \beta_n\rangle$ 表示长度为 $(\omega + 1) + b \times (\omega + 1)$ 的元组，其形式为 $\langle\alpha\rangle$， $\langle\beta_1\rangle$， . . .， $\langle\beta_n\rangle$， $\star$， . . .， $\star$。

我们定义 Datalog 查询 $Q_\downarrow = (\Sigma'$， HEdge)，其中 $\Sigma' = \Sigma \cup \Sigma''$；对于每条形式为 $\sigma \in \Sigma$ $R_0(\bar{x}_0)$:– $R_1(\bar{x}_1) \ldots$， $R_n(\bar{x}_n)$ 的规则， Σ'' 包含这些规则

$$\sigma_1 = \mathsf{HEdge}(\langle R_0(\bar{x}_0), R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)\rangle) :- \mathsf{CurNode}(\langle R_0(\bar{x}_0)\rangle), R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$$

并且，对于每个 $i \in [n]$,

$$\sigma_2^{(i)} = \mathsf{CurNode}(\langle R_i(\bar{x}_i)\rangle) :- \mathsf{CurNode}(\langle R_0(\bar{x}_0)\rangle), R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n).$$

本质上，规则 σ_1会构造一个超边 (α， T)，它属于 down(D， Q， $R(\bar{t})$)；只要已知 α是 down(D， Q， $R(\bar{t})$)中的一个节点，并且 T中的所有原子都用于 gri(D， Σ)以生成 α。形式为 $\sigma_2^{(i)}$ 的规则会将新节点标记为属于 down(D， Q， $\bar{t}$)。请注意， Σ'' 中的规则包含常量，而按照我们的定义，规则不含常量。尽管如此，所有现有的 Datalog 引擎都支持含常量的规则；为使本节讨论简单明了，我们稍微放宽了规则的定义。只需向数据库添加一些辅助事实，就可以轻松地将上述规则集改写为不含常量的规则集。

数据库 $D_\downarrow$是 $D \cup \{\mathsf{CurNode}(\langle R(\bar{t})\rangle)\}$，这简单地说明 $R(\bar{t})$ 必须是 down($D, Q, R(\bar{t})$) 的一个节点。

很容易看出， $Q_\downarrow(D_\downarrow)$ 中的每个元组都编码了 down($D, Q, R(\bar{t})$) 的一条超边，因此，我们只需在 $D_\downarrow$ 上查询 $Q_\downarrow$，就可以构造 down($D, Q,$ $R(\bar{t})$)。

需要指出的是，我们对 $Q_\downarrow$ and $D_\downarrow$ 的定义与（Elhalawati、Kr¨otzsch 和 Mennicke，2022）的定义之间，主要区别在于：我们将 down($D, Q, R(\bar{t})$) 的节点和超边编码为元组，这使我们能够使用与回答原始查询 Q 相同的 Datalog 引擎，而不必使用支持存在规则等表达能力更强语言的外部引擎。

构造公式。关于公式 $\phi_{(\bar{t},D,Q)}$ 的构造，如前所述，出于效率考虑，我们对其子公式 $\phi_{acyclic}$ 采用了不同的编码。我们没有使用传递闭包，而是采用顶点消去技术（Rankooh 和 Rintanen，2022）。这种方法的优点是，对 $\phi_{acyclic}$ 进行编码所需的布尔变量数量为 $O(n \times \delta)$ 量级，其中 n 是底层图的节点数， δ 是该图所谓的消去宽度，粗略而言，它与底层图的连通程度有关。因此，只要消去宽度较低，我们就可以避免构造数量呈平方级增长的变量，这一过程代价高昂。

增量构造 why-来源。回顾一下，我们关注的是 why-来源的增量计算，因为与一次性计算整个集合相比，这种方式在实践中更有用。为此，我们需要一种不重复地枚举 why-来源所有成员的方法。我们通过改造 SAT 文献中用于枚举布尔公式满足赋值的一种标准技术来实现这一点，这种技术称为阻塞子句。我们首先将 S 中关于 D 和 Σ的 $R(\bar{t})$ 向下闭包中出现的 D 的所有事实收集到一个集合中。然后，在请求 SAT 求解器给出 $\phi_{(\bar{t},D,Q)}$ 的任意满足赋值 τ 后，我们输出数据库 db(τ)，接着构造“阻塞”子句 $\vee_{\alpha \in S} \ell_\alpha$，其中，若 $\alpha \in$ db(τ)，则 $\ell_\alpha = \neg x_\alpha$；否则为 $\ell_\alpha = x_\alpha$。随后，我们将此子句加入公式，以此表达不存在其他满足赋值 τ' 能产生 why-来源中的同一成员。这样就会将先前计算出的解释排除在计算之外。每当得到 why-来源的一个新成员时，我们都会继续添加这样的阻塞子句，直到公式不可满足。

D.4 实验评估

本节将进一步介绍我们基于 SAT 的方法的性能，并展示论文中所考虑的所有情景的实验评估结果；为求完整，我们再次报告 Andersen 场景。回想一下，在实验分析中，我们分别考虑两项主要任务：（1）构造向下闭包和布尔公式；（2）使用 SAT 求解器增量式计算 why-来源。

对于任务 1，我们在图 3 中为所考虑的每种情景分别绘制一张图，其中每张图报告该情景中每个数据库的总运行时间。此外，对于每张图以及其中考虑的每个数据库，

我们绘制了五个柱形，分别对应随机选取的五个元组。每个柱形显示构建向下闭包和构造布尔公式所需的时间之和。为便于展示，我们将基于 Doctors 的情景归在同一张图中（请回想一下，所有这类情景共用一个数据库）。

可以看出，在大多数情景中，运行时间约为几秒。对于基于传递闭包和 Doctors 的情景尤其如此，因为它们的查询最简单；而在 Galen 情景中，查询更复杂，因为涉及非线性递归，因此在最大的数据库 D_4 上耗时稍长。Andersen 和 CSDA 情景最具挑战性，因为两者都包含非常大的数据库。此外，尽管 Andersen 中的数据库比 CSDA 中的更小，但其查询的复杂度较高，涉及非线性递归，因此 Andersen 在最大数据库（680 万条事实）上的运行时间与 CSDA 在更大的数据库D_{httpd}（1000 万条事实）上的运行时间相当。当然，对于大得多的数据库$D_{\mathsf{postgresql}}$和 D_{linux}，运行时间会长得多，处理某些元组甚至需要 6 到 7 分钟。考虑到这些数据库的规模，我们认为最后这些情景的运行时间相当合理。正如论文正文中已讨论的，我们发现大部分时间都花在构建向下闭包上。

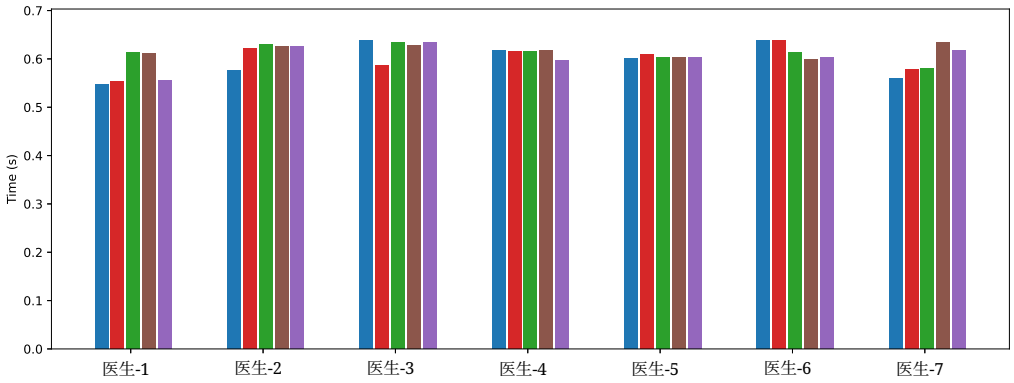
关于任务2，即 why-来源 的增量计算，我们在图4中为所考虑的每种情景各展示一张图。在情景s 的每张图中，我们报告了 s的每个数据库构建解释所需的时间，也就是当前 why-来源 元组与下一个元组之间的时间间隔（该时间也称为延迟）。每张图汇总了针对随机选出的五个元组分别构建 why-来源 元组所需的延迟（最多构建 10K 个元组，或达到 5 分钟超时）。我们使用箱线图，其中箱体的底边和顶边分别表示第一四分位数和第三四分位数，即分别有 25% 和 75 % 的延迟不超过该值；橙色线表示中位延迟。此外，底部和顶部须线分别表示最小延迟和最大延迟。所有时间均以毫秒为单位，并采用对数刻度。与任务1一样，我们将基于 Doctors 的情景汇总在一张图中。

正如论文正文中所观察到的，大多数延迟甚至低于 1 毫秒，中位数在微秒量级。因此，一旦布尔公式构造完成，增量计算 why-来源的成员就会非常快。最坏情况出现在 TransClosure 情景中，考察 Facebook 数据库时：这里也是仅有的两种情况中，支撑集的构造在能够构造出 10K 个支撑集之前超过了 5 分钟，即第三个和第五个元组。总体而言，在 Facebook 数据库的 TransClosure 情景中，平均延迟更高，有些支撑集最多需要 10 秒才能构造出来。我们认为，这与数据库 D_{facebook} 编码了一个高度连通的图有关，因此合取范式公式，尤其是编码无环性的公式 $\phi_{acyclic}$ 检查，变得相当庞大，因此对 SAT 求解器的要求也高得多。这在一定程度上是意料之中的，因为底层图的连通性越低，顶点消除技术检查无环性的效果就越好。我们通过使用取自（Fan、Mallireddy 和 Koutris（2022））的数据库进行其他实验，证实了这一点；这些数据库包含高度连通的合成图。在这种情况下，尽管构造向下闭包的效率很高（耗时为数秒），公式 $\phi_{acyclic}$ 的构造却会耗尽内存。因此，我们预计，在高度连通的输入图很常见的应用中，需要采用不同的方法来检查合取范式公式中的无环性。尽管如此，我们可以有把握地得出结论：在大多数情况下，计算 why-来源的成员都非常高效。

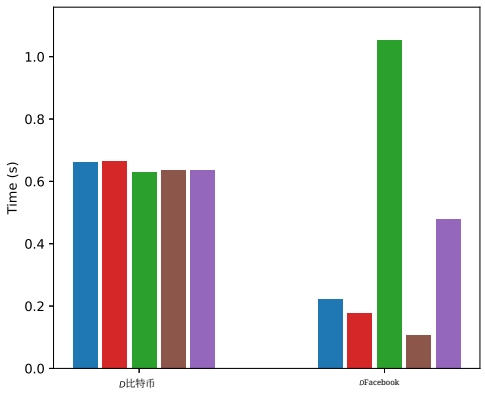
D.5 比较评估

如第 6 节所述，我们已与（Elhalawati、Kr¨otzsch 和 Mennicke 2022）进行了初步比较。本节最后将讨论此次比较评估的细节。首先需要澄清的是，我们的实现处理的是另一个问题。对于 Datalog 查询 $Q = (\Sigma,\ R)$、基于扩展数据库 D （Σ）的数据库，以及元组$\bar{t} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$，Elhalawati、Kr¨otzsch 和 Mennicke（2022）的方法旨在构建完整的 why($\bar{t},\ D,\ Q$) 集合，并针对这一目标进行了评估；而我们的方法旨在增量计算 why$^{\mathsf{UN}}$($\bar{t},\ D,\ Q$)，并针对这一目标进行了评估。不过，在某种情形下可以进行合理的比较，从而为这两种方法提供一些参考。当 Datalog 查询 Q 同时为线性且非递归时，集合 why($\bar{t},\ D,\ Q$) 和 why$^{\mathsf{UN}}$($\bar{t},\ D,\ Q$) 相同，因为相对于 D 和 Σ， $R(\bar{t})$ 的证明树显然没有歧义。因此，为了公平比较，我们将考察 Doctors-i 情景（$i \in [7]$）：其中包含一个线性且非递归的 Datalog 查询；同时考察我们方法的端到端运行时间（而非延迟），当然，不对待构建的 why-来源成员数量或总运行时间设限。

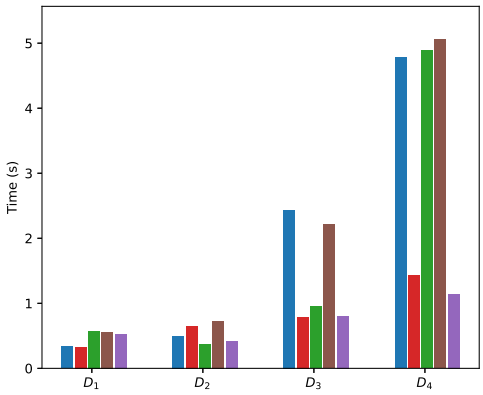
比较结果如图 5 所示。对于每种情景，我们展示了所提方法（蓝色）和（Elhalawati、Kr¨otzsch 与 Mennicke 2022）的方法（红色）在五个随机选取的元组上的运行时间；如果某个元组缺少对应的柱形，则表示执行时内存耗尽。我们观察到，对于简单情景，两种方法的性能相当，运行时间约为一秒。接下来，对于具有挑战性的情景，即 Doctors-i for $i \in \{1, 5, 7\}$，我们观察到，总体而言，所提方法更快。还可以看到，对于一些最具挑战性的情况，（Elhalawati、Kr¨otzsch 与 Mennicke 2022）的方法会耗尽内存。我们认为，这是因为该方法使用了规则引擎 VLog；VLog 面向基于存在规则的物化推理，而我们的方法依赖 Datalog 引擎（特别是 DLV），因此能够利用评估 Datalog 查询时通常采用的所有优化。例如，DLV 实现的魔集改写技术可以通过在规则求值过程中生成少得多的事实，大幅降低内存使用量；参见例如（Leone 等（2019））。



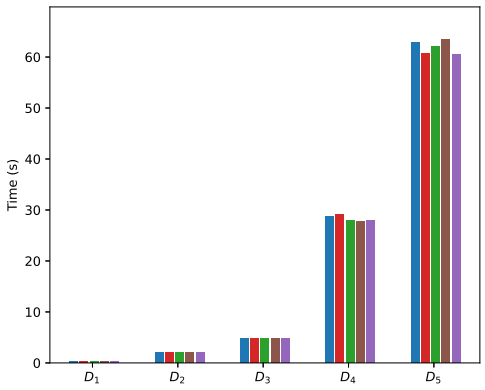
(a) Doctors



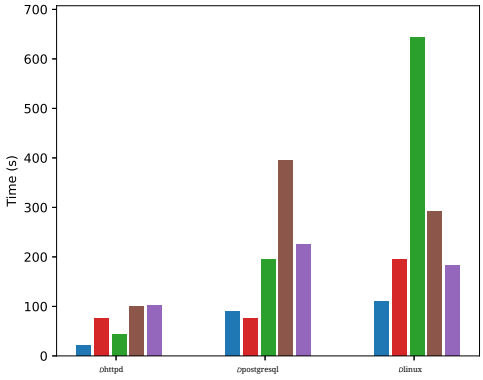
(b) 传递闭包



(c) Galen

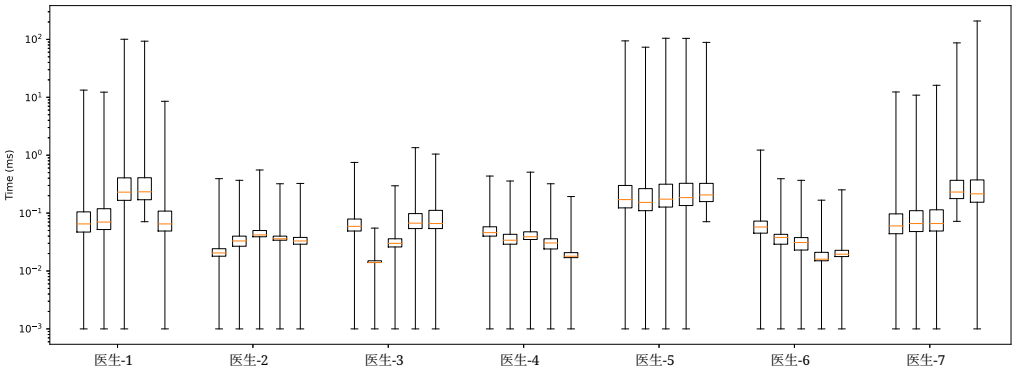


(d) Andersen

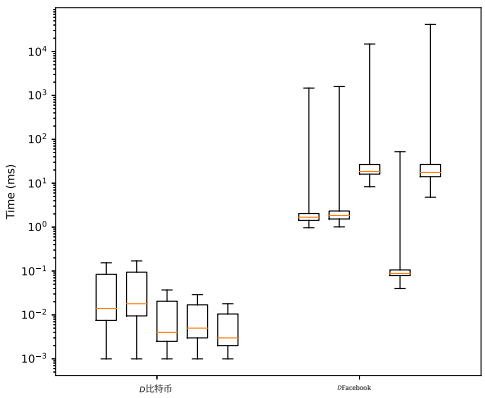


(e) CSDA

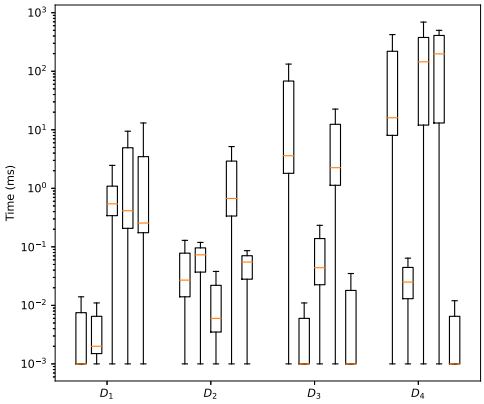
图 3：构建　　向下闭包和布尔公式（　　所有情景）。



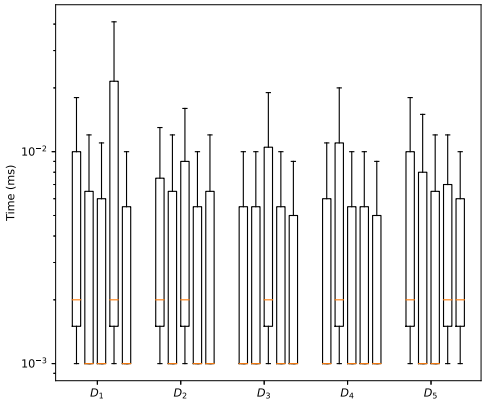
(a) Doctors



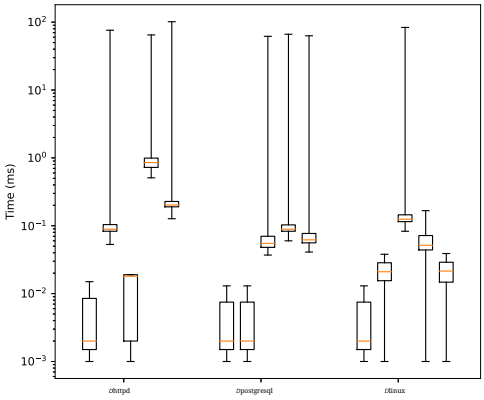
(b) 传递闭包



(c) Galen



(d) Andersen



(e) CSDA

图4：why-来源的增量计算（所有情景）。

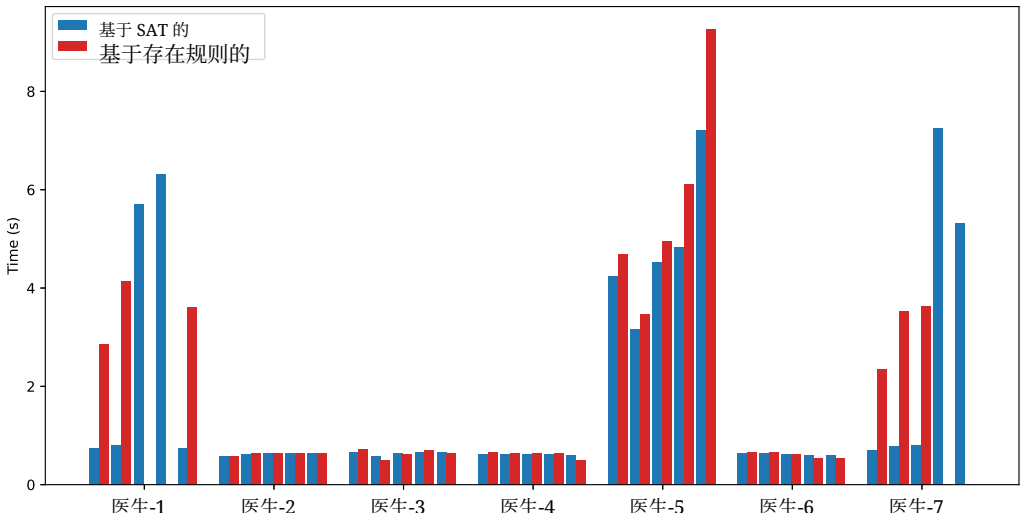


图 5：why-来源的端到端计算比较。