Why(-Not)-Provenance for Datalog with Negation

Bart Bogaerts[1,2] , **Marco Calautti**[3] , **Andreas Pieris**[4,5] , **Samuele Pollaci**[2,1] , **Robbe Van den Eede**[1,2]
[1]KU Leuven
[2]Vrije Universiteit Brussel
[3]University of Milano
[4]University of Cyprus
[5]University of Edinburgh
{bart.bogaerts, robbe.vandeneede}@kuleuven.be, marco.calautti@unimi.it, apieris@ed.ac.uk, samuele.pollaci@vub.be

Abstract

Datalog is a powerful rule-based language with numerous applications in databases and knowledge representation. Explaining why a fact belongs to the output of a Datalog program over a database is an essential task towards explainable and transparent data-intensive applications. A standard way of explaining a fact is the so-called why-provenance, which provides witnesses in the form of subsets of the input database that as a whole can be used to derive that fact. While why-provenance for Datalog has been extensively studied in the literature, the analogous notion for Datalog with negation remains unexplored. We extend why-provenance to Datalog with negation under the standard well-founded and stable model semantics, inherited from Logic Programming, by building on justification theory. We then perform a thorough data complexity analysis of the underlying explainability problem and show that it is in general intractable for both well-founded and stable model semantics; in particular, it is NP-complete, which is the best that we can hope for since the problem is already NP-complete for positive Datalog.

1 Introduction

Datalog emerged as a logic-based language rooted in Logic Programming for defining expressive queries over relational databases, and has since been studied extensively (Abiteboul, Hull, and Vianu 1995). A Datalog program Σ is a finite set of rule-like expressions that (inductively) define new relations starting from a relational database D; we write $\Sigma(D)$ for the output of Σ on D. Understanding why a fact α belongs to $\Sigma(D)$, i.e., why α is derived from D and Σ, is essential for explaining the derivation process that leads to α. A standard approach to providing such explanations is *why-provenance* (Buneman, Khanna, and Tan 2001), where the idea is to identify all the subsets of the database that, taken together, suffice to derive a given fact.

The notion of why-provenance is closely connected with the *proof-theoretic semantics* of Datalog, which defines $\Sigma(D)$ as the set of facts that admit a proof tree w.r.t. D and Σ, that is, a tree-like representation of a derivation of a fact by starting from D and executing the rules of Σ. The support of such a proof tree of a fact α, i.e., the set of database facts that label its leaves, forms an explanation of α. In other words, the why-provenance of α w.r.t. D and Σ is the family of sets of atoms obtained by considering all the proof trees of α w.r.t. D and Σ, and then collecting their supports.

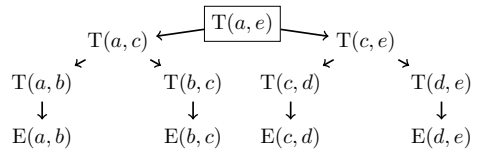

Figure 1: An example proof tree.

Example 1. *Consider the Datalog program* Σ *consisting of*

$$\mathrm{T}(x, y) \text{ :- } \mathrm{E}(x, y) \qquad \mathrm{T}(x, z) \text{ :- } \mathrm{T}(x, y), \mathrm{T}(y, z),$$

defining the binary relation T *that stores the transitive closure of a graph with edges stored via the relation* E*. A proof tree* P *of the fact* $\mathrm{T}(a, e)$ *w.r.t. the database*

$$D_{\mathsf{ex}} = \{\mathrm{E}(a, b), \mathrm{E}(a, c), \mathrm{E}(b, c), \mathrm{E}(c, d), \mathrm{E}(d, e)\}$$

and Σ *is depicted in Figure 1. The support of* P *is the set of facts* $\{\mathrm{E}(a, b), \mathrm{E}(b, c), \mathrm{E}(c, d), \mathrm{E}(d, e)\}$. ∎

Why-provenance has been studied intensively. There are theoretical studies (Damásio, Analyti, and Antoniou 2013; Deutch et al. 2014; Khamis et al. 2022), attempts to underapproximate it towards an efficient computation (Zhao, Subotic, and Scholz 2020), studies on the restricted setting of non-recursive Datalog (Lee, Ludäscher, and Glavic 2019), attempts to compute it by transforming the grounded Datalog rules to a system of equations (Esparza, Luttenberger, and Schlund 2014), and attempts to compute it on demand (Calautti et al. 2024c; Elhalawati, Krötzsch, and Mennicke 2022). The complexity of why-provenance for Datalog, i.e., of the problem of deciding whether a subset of the database belongs to the why-provenance of a fact, has been explored recently (Calautti et al. 2024b; 2024a).

Adding Negation. Adding the useful feature of negation to Datalog is very natural (Abiteboul, Hull, and Vianu 1995). Of course, the extension of Datalog with negation (or simply Datalog$^\neg$) comes with the complication of properly defining the semantics. The two most commonly used semantics, inherited from Logic Programming, are the well-founded semantics (Gelder, Ross, and Schlipf 1991) and stable model

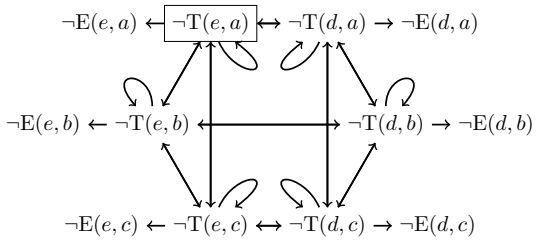

Figure 2: An example justification for the negative fact $\neg\mathrm{T}(e, a)$ for the transitive closure program.

semantics (Gelfond and Lifschitz 1988). However, despite the extensive research effort dedicated to Datalog$^\neg$ under the above semantics, the problem of explainability via notions analogous to why-provenance for positive Datalog remains unexplored. Note that the well-founded semantics is typically defined via a fixpoint approach, whereas the stable model semantics via a model-theoretic approach. Therefore, without having the equivalent proof-theoretic semantics of Datalog$^\neg$ in place, it is not clear how the notion of why-provenance for Datalog$^\neg$ can be defined in a way that leads to a natural generalization of why-provenance for positive Datalog. This leads to the following research question:

How can the proof-theoretic semantics of Datalog$^\neg$ under the well-founded and stable semantics be defined in order to obtain natural notions of why-provenance for Datalog$^\neg$?

Once we have an answer to the above question, the next task is to understand how complex it is to compute the why-provenance of a fact w.r.t. a database and a Datalog$^\neg$ program under the well-founded and stable model semantics. This leads to our second main research question:

What is the complexity of why-provenance for Datalog$^\neg$ under the well-founded and stable model semantics w.r.t. the size of the input database?

Methodology and Contributions. To tackle the first question, we build on *justification theory*, a theory that was originally developed by Denecker to explain the declarative view of Logic Programming as a logic of inductive definitions (Denecker and De Schreye 1993; Denecker 1993). He introduced the concept of justification as a mathematical abstraction for the induction process. Later, justification theory was generalized to a unifying framework for capturing a variety of different semantics of a range of non-monotonic logics (Denecker, Brewka, and Strass 2015).

Towards the proof-theoretic semantics of Datalog$^\neg$, we translate justification theory into the Datalog world. In this setting, a justification can be seen as a generalization of a proof tree, extending it in different aspects. This yields an elegant framework for defining the proof-theoretic semantics of Datalog$^\neg$ programs under different semantics, and thus obtaining natural notions of why-provenance in a uniform way. Interestingly, this framework also allows us to obtain natural notions of *why-not-provenance* for Datalog with negation under different semantics, with the aim of explaining why a fact is *not* derived. Figure 2 shows a justification of $\neg\mathrm{T}(e, a)$ under both the well-founded and stable model semantics. Its support contains $\neg\mathrm{E}(x, y)$ with $x \in \{d, e\}$ and $y \in \{a, b, c\}$, i.e., it provides as a reason for why there is no path from e to a the fact that there is a cut between $\{d, e\}$ and $\{a, b, c\}$ that is not crossed in the right direction.

After translating justification theory into the Datalog setting and defining the proof-theoretic semantics of Datalog$^\neg$, which then allow us to define why(-not)-provenance, we focus on the second question concerning complexity. In particular, for a Datalog$^\neg$ program Σ, we study the following decision problem: given a database D, a fact α in a model of D and Σ under the well-founded (resp., stable model) semantics, and a set of literals L, is it the case that L belongs to the why(-not)-provenance of α w.r.t. D and Σ under the well-founded (resp., stable model) semantics? We show that no matter the underlying semantics, all the problems of interest are NP-complete, which is the best that we can hope for since the problem is already NP-complete for positive Datalog programs (Calautti et al. 2024b).

Related Work. The problem of explaining entailments for the stable semantics has already been studied (Fandinno and Schulz 2019), with multiple approaches using an alternative notion of "justification". Closer to our notion of justification are LABAS justifications (Schulz and Toni 2016; 2013) and off-line justifications (Pontelli, Son, and Elkhatib 2009; Pontelli and Son 2006), even though the latter corresponds to a special kind of justifications called unambiguous. Causal graph justifications (Cabalar, Fandinno, and Fink 2014; Cabalar and Fandinno 2016) and why-not-provenance justifications (Damásio, Analyti, and Antoniou 2013) provide counterfactual explanations. These differ in nature from our explanations, as they include literals that are false in the given stable model, but whose truth would entail the falsity of the explained literal.

Recently, Eiter and Geibinger (2025) introduced a sequent calculus for answer set entailment, which they argue to provide a framework for explanations. However, their proposed explanations (which they refer to as future work) are natural language arguments derived from sufficiently simple proofs in their sequent calculus. Moreover, their proofs are significantly more complex than our justifications.

To the best of our knowledge, our framework is the first to uniformly define proof-theoretic semantics for Datalog with negation under different semantics while generalizing why-provenance for positive Datalog, and our work is the first to study the complexity of the associated provenance problem.

2 Preliminaries

We assume disjoint, countably infinite sets $\mathbf{C}$ and $\mathbf{V}$ of *constants* and *variables*, respectively. We refer to constants and variables as *terms*. For $n > 0$, we write $[n]$ for $\{1, \ldots, n\}$.

Relational Databases. A *schema* $\mathbf{S}$ is a finite set of relation names (or predicates) with associated arity. We write R/n to say that R has arity $n \geq 0$; we may also write $\mathsf{ar}(R)$ for n. A *(relational) atom* over $\mathbf{S}$ is an expression of the form $R(\bar{t})$, where $R/n \in \mathbf{S}$ and $\bar{t}$ is an n-tuple of terms. By abuse of notation, we may treat tuples as the *set* of their elements. A *fact* is an atom with only constants. A *database* D over

$\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of D, denoted $\mathsf{dom}(D)$, is the set of constants that appear in D.

Syntax of Datalog$^\neg$. A *literal* over a schema $\mathbf{S}$ is either an atom over $\mathbf{S}$ (i.e., *positive literal*) or an atom over $\mathbf{S}$ preceded with the negation symbol $\neg$ (i.e., a *negative literal*). A Datalog$^\neg$ rule is defined as plain Datalog rules, with the difference that also negative literals can appear in the body as long as their variables occur in a positive literal. Formally, a *(Datalog$^\neg$) rule* σ over a schema $\mathbf{S}$ is an expression

$$R_0(\bar{x}_0) \text{ :- } R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n), \\ \neg R_{n+1}(\bar{x}_{n+1}), \ldots, \neg R_{n+m}(\bar{x}_{n+m}),$$

for $n \geq 1$ and $m \geq 0$, where $R_i(\bar{x}_i)$ is a (constant-free) relational atom over $\mathbf{S}$ for $i \in \{0, \ldots, n+m\}$, and each variable that occurs in σ, occurs in $\bar{x}_k$ for some $k \in [n]$; the latter is known as the *safety condition*. We refer to $R_0(\bar{x}_0)$ as the *head* of σ, denoted $\mathsf{head}(\sigma)$, and to $R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$, $\neg R_{n+1}(\bar{x}_{n+1}), \ldots, \neg R_{n+m}(\bar{x}_{n+m})$ as the *body* of σ. A *Datalog$^\neg$ program* over $\mathbf{S}$ is a finite set Σ of Datalog$^\neg$ rules over $\mathbf{S}$. A predicate R occurring in Σ is called *intensional* if there is a rule in Σ with R in its head and *extensional* otherwise. The *extensional (database) schema* of Σ, denoted $\mathsf{edb}(\Sigma)$, is the set of all extensional predicates in Σ, while the *intensional schema* of Σ, denoted $\mathsf{idb}(\Sigma)$, is the set of all intensional predicates in Σ; by definition, $\mathsf{edb}(\Sigma) \cap \mathsf{idb}(\Sigma) = \emptyset$. The *schema* of Σ, denoted $\mathsf{sch}(\Sigma)$, is the set $\mathsf{edb}(\Sigma) \cup \mathsf{idb}(\Sigma)$, which is in general a subset of $\mathbf{S}$.

3 Proof-theoretic Semantics of Datalog$^\neg$

We introduce the well-founded and stable model semantics of Datalog$^\neg$ inherited from Logic Programming. However, instead of recalling the standard definitions from (Gelder, Ross, and Schlipf 1991; Gelfond and Lifschitz 1988), we present alternative definitions that rely on the notion of justification graph inherited from justification theory (Denecker and De Schreye 1993; Denecker, Brewka, and Strass 2015; Marynissen 2022), which will anyway play a key role in our development. The notion of justification graph generalizes the well-known notion of proof tree for Datalog programs, and thus, what we are going to introduce can be seen as the *proof-theoretic semantics* of Datalog$^\neg$. To the best of our knowledge, this is the first time that this framework is translated into Datalog terminology, and hence, the first time the well-founded and stable model semantics of Datalog$^\neg$ programs are defined by following a proof-theoretic approach. We see this as a central conceptual contribution of our work.

3.1 Relativized Semantics of Datalog$^\neg$

We start by introducing the so-called relativized semantics of Datalog$^\neg$, which will then allow us to define in a uniform way the well-founded and stable model semantics. We first give the high-level idea underlying this relativized semantics. To do so, we need to recall 3-valued interpretations.

3-Valued Interpretations. For a Datalog$^\neg$ program Σ and a database D over $\mathsf{edb}(\Sigma)$, let $\mathsf{base}^+(D, \Sigma)$ be the set of all facts (i.e., positive literals) that can be formed using predicates of $\mathsf{sch}(\Sigma)$ and constants of $\mathsf{dom}(D)$ and $\mathsf{base}^-(D, \Sigma)$ be the set of negative literals $\{\neg\alpha \mid \alpha \in \mathsf{base}^+(D, \Sigma)\}$; we write $\mathsf{literals}(D, \Sigma)$ for the set $\mathsf{base}^+(D, \Sigma) \cup \mathsf{base}^-(D, \Sigma)$.

We consider 3-valued interpretations with the three truth values being *false* (f), *unknown* (u), and *true* (t). We further adopt the truth order $\leq_T$ over $\{\mathsf{f}, \mathsf{u}, \mathsf{t}\}$ such that $\mathsf{f} \leq_T \mathsf{u}$ and $\mathsf{u} \leq_T \mathsf{t}$. Moreover, following classical 3-valued logic, $\neg\mathsf{t} = \mathsf{f}$, $\neg\mathsf{f} = \mathsf{t}$, and $\neg\mathsf{u} = \mathsf{u}$. A *3-valued interpretation I for D and Σ* is a pair (T, P), where $T \subseteq P \subseteq \mathsf{base}^+(D, \Sigma)$ such that, for every positive literal $R(\bar{t}) \in \mathsf{base}^+(D, \Sigma)$ with $R \in \mathsf{edb}(\Sigma)$, the following holds: $R(\bar{t}) \in D$ iff $R(\bar{t}) \in T$ iff $R(\bar{t}) \in P$. Intuitively, T is the set of true atoms, and P is the set of "possible" (or "non-false") atoms; hence, the atoms in $P \setminus T$ are unknown and the atoms in $\mathsf{base}^+(D, \Sigma) \setminus P$ are false. Whenever $T = P$, the interpretation I is *2-valued*. Henceforth, by interpretation we mean 3-valued interpretation. It will be made explicit whenever we want to refer to a 2-valued interpretation. Given an atom $\alpha \in \mathsf{base}^+(D, \Sigma)$ and an interpretation $I = (T, P)$ for D and Σ, the *value of α in I*, denoted $I(\alpha)$, is f if $\alpha \notin P$, u if $\alpha \in P \setminus T$, and t if $\alpha \in T$. For a literal $\neg\alpha \in \mathsf{base}^-(D, \Sigma)$, we define $I(\neg\alpha)$ as $\neg I(\alpha)$. By convention, $I(x) = x$ for each $x \in \{\mathsf{f}, \mathsf{u}, \mathsf{t}\}$.

High-level Description. The semantics of a Datalog$^\neg$ program Σ is essentially a function that assigns to each database D over $\mathsf{edb}(\Sigma)$ a set of interpretations for D and Σ. Thus, the relativized semantics in question should provide different ways for assigning to each database D over $\mathsf{edb}(\Sigma)$ a set of interpretations for D and Σ. The key ingredients towards the relativized semantics are:

1. The justification graph of a literal $\ell \in \mathsf{literals}(D, \Sigma)$ w.r.t. D and Σ, which is a (possibly infinite,) labeled, rooted directed graph whose nodes are labeled with literals from $\mathsf{literals}(D, \Sigma)$ and the root is labeled ℓ.
2. The support of a justification graph G, relative to a branch evaluation function BEval, consisting of all the values that are assigned by BEval to the complete branches of G. A complete branch is a sequence of literals that labels a finite path from the root to a leaf or an infinite path starting at the root; BEval assigns to each complete branch of G a literal or a truth value from $\{\mathsf{f}, \mathsf{u}, \mathsf{t}\}$.
3. The truth value of a justification graph G in an interpretation I for D and Σ, relative to BEval, which is the least value, according to $\leq_T$, among the truth values $I(\ell)$ for all elements ℓ of the support of G (relative to BEval) in I.
4. The supported truth value of a literal $\ell \in \mathsf{literals}(D, \Sigma)$ in an interpretation I for D and Σ, relative to BEval, which is the greatest value (w.r.t. $\leq_T$) over the truth values of all justifications of ℓ w.r.t. D and Σ in I (relative to BEval).

Having the supported value of a literal, we define the semantics of a Datalog$^\neg$ program Σ, relative to a branch evaluation function BEval, as the function $[\![\cdot]\!]^\Sigma_{\mathsf{BEval}}$ that assigns to each database D over $\mathsf{edb}(\Sigma)$ the set of all the interpretations I for D and Σ such that, for each literal $\ell \in \mathsf{literals}(D, \Sigma)$, the supported truth value of ℓ in I (relative to BEval) coincides with $I(\ell)$. Thus, different branch evaluation functions lead to different semantics; hence the term "relativized semantics". We now formalize the above discussion.

Justification Graph. To define this notion we need a mechanism for grounding the given Datalog$^\neg$ rules. Consider a Datalog$^\neg$ program Σ and a database D over $\mathsf{edb}(\Sigma)$. Let σ be a rule of Σ. A *ground instance of σ relative to D* is obtained from σ by simply replacing each occurrence of a variable x in σ with a constant $c_x \in \mathsf{dom}(D)$. We write $\mathsf{ground}^+(D, \Sigma)$ for the set of all ground instances of rules in Σ relative to D. A member of $\mathsf{ground}^+(D, \Sigma)$ is called a *ground rule of Σ relative to D*. We further need the notion of complementary ground rule. Formally, a *complementary ground rule of Σ relative to D* is an expression of the form $\neg\alpha \text{ :- } \neg\ell_1, \ldots, \neg\ell_m$, for $m \geq 1$, where $\alpha \in \mathsf{base}^+(D, \Sigma)$ and $\ell_j \in \mathsf{literals}(D, \Sigma)$ for each $j \in [m]$ (note that, for an atom β, $\neg\neg\beta = \beta$), such that the following holds: assuming that $\left\{\alpha \text{ :- } \beta^i_1, \ldots, \beta^i_{n_i}\right\}_{i\in[m]}$ is the set of all the rules of $\mathsf{ground}^+(D, \Sigma)$ with α as their head, $(\ell_1, \ldots, \ell_m) \in \bigtimes_{i\in[m]} \left\{\beta^i_1, \ldots, \beta^i_{n_i}\right\}$. Intuitively, a complementary ground rule for $\neg\alpha$ gives us a direct reason why α should be false, similar to how a ground rule gives us a direct reason why α should be true. Indeed, α should be false in case all the rules that can derive it are "blocked". This blocking is achieved here by selecting (at least) one literal from each rule with α as head. We write $\mathsf{ground}^-(D, \Sigma)$ for the set of all complementary ground rules of Σ relative to D. We also write $\mathsf{ground}(D, \Sigma)$ for $\mathsf{ground}^+(D, \Sigma) \cup \mathsf{ground}^-(D, \Sigma)$. We are now ready to properly define the notion of justification graph of a literal w.r.t. a database and a Datalog$^\neg$ program. Slightly generalizing the notation for interpretations to databases, if ℓ is a literal over $\mathsf{edb}(\Sigma)$, we write $D(\ell)$ to denote t if ℓ is true in D (i.e., if ℓ is positive and $\ell \in D$ or $\ell = \neg\alpha$ for an atom α and $\alpha \notin D$) and f otherwise.

Definition 1 (Justification Graph). *Consider a Datalog$^\neg$ program Σ, a database D over $\mathsf{edb}(\Sigma)$, and a literal $\ell \in \mathsf{literals}(D, \Sigma)$. A* justification graph of ℓ w.r.t. D and Σ *is a labeled, rooted directed graph $G = (V, E, \lambda, r)$, with $r \in V$ and $\lambda : V \to \mathsf{literals}(D, \Sigma)$, such that:*

1. *It holds that $\lambda(r) = \ell$.*
2. *If $v \in V$ is a leaf, then the literal $\lambda(v)$ is over $\mathsf{edb}(\Sigma)$ and $D(\lambda(v)) = \mathsf{t}$.*
3. *If $v \in V$ has $n \geq 1$ children $u_1, \ldots, u_n$, then there exists a rule $\ell_0 \text{ :- } \ell_1, \ldots, \ell_n$ in $\mathsf{ground}(D, \Sigma)$ such that $\lambda(v) = \ell_0$ and for each $i \in [n]$, $\lambda(u_i) = \ell_i$.* ∎

Henceforth, we simply say justification instead of justification graph, and we refer to a justification of some literal w.r.t. a database D and a Datalog$^\neg$ program Σ as (D, Σ)-justification. For a justification $G = (V, E, \lambda, r)$, we write $\mathsf{rt}(G)$ for its root. When we speak about a *path* in G, this can be either a finite or infinite path. For a finite (resp., infinite) path $P = v_1, \ldots, v_n$ (resp., $P = v_1, v_2, \ldots$) in G, we write $\lambda(P)$ for its labels, i.e., the finite (resp., infinite) sequence of literals $\lambda(v_1), \ldots, \lambda(v_n)$ (resp., $\lambda(v_1), \lambda(v_2), \ldots$). A path is *maximal* if it is either infinite or of the form $v_1, \ldots, v_n$ with v_n a leaf. Here is an example of a justification:

Example 2. *We want to define a relation of "no return" that collects pairs of nodes (x, y) such that there is a path from x to y, but there is no path from y to x. Such a relation is defined by the Datalog$^\neg$ program Σ_{ex} consisting of the rules*

$$\begin{array}{rcl} \mathrm{T}(x, y) & \text{:-} & \mathrm{E}(x, y), \\ \mathrm{T}(x, z) & \text{:-} & \mathrm{T}(x, y), \mathrm{T}(y, z), \\ \mathrm{NoReturn}(x, y) & \text{:-} & \mathrm{T}(x, y), \neg\mathrm{T}(y, x). \end{array}$$

Consider again the database

$$D_{\mathsf{ex}} = \{\mathrm{E}(a, b), \mathrm{E}(a, c), \mathrm{E}(b, c), \mathrm{E}(c, d), \mathrm{E}(d, e)\}$$

from Section 1. A justification of $\mathrm{NoReturn}(a, e)$ w.r.t. D_{ex} and Σ_{ex}, denoted G_{ex}, is given in Figure 3.[1] *Note that the left part of this justification corresponds to the proof tree from Figure 1, while the right part to the justification from Figure 2. The right part is cyclic and, as intuitively explained above, represents a cut in the graph that cannot be crossed. We will later see that this is indeed a "good" justification (under both well-founded and stable model semantics).* ∎

Relativized Justification Support. Consider a (D, Σ)-justification $G = (V, E, \lambda, r)$. A *branch of G* is a finite or infinite sequence of literals $B = \ell_1, \ldots, \ell_n$ or $B = \ell_1, \ell_2, \ldots$, respectively, for which there is a maximal path P of nodes in G with $B = \lambda(P)$. We say that the branch B of G is witnessed by the path P in G. For a node $v \in V$, let $\mathsf{branch}(G, v)$ be the set of all branches of G witnessed by a path in G starting at v. A *branch evaluation* is simply a function BEval that assigns to every (finite or infinite) sequence L of literals either a literal occurring in L or a truth value from $\{\mathsf{f}, \mathsf{u}, \mathsf{t}\}$. We can now define the notion of support of a justification relative to some branch evaluation.

Definition 2 (Justification Support). *Let* BEval *be a branch evaluation. Consider a (D, Σ)-justification $G = (V, E, \lambda, r)$ and a node $v \in V$. The* support of G at v relative to BEval *is defined as the set*

$$\mathsf{support}_{\mathsf{BEval}}(G, v) \;=\; \{\mathsf{BEval}(B) \mid B \in \mathsf{branch}(G, v)\}.$$

We use $\mathsf{support}_{\mathsf{BEval}}(G)$ to denote the support of G relative to BEval*, defined as $\mathsf{support}_{\mathsf{BEval}}(G, \mathsf{rt}(G))$.* ∎

Truth Value of a Justification. We now define the truth value of a justification in an interpretation relative to a branch evaluation. For a set $V \subseteq \{\mathsf{f}, \mathsf{u}, \mathsf{t}\}$ of truth values, we write $\min V$ (resp., $\max V$) for the least (resp., greatest) value of V according to the truth order $\leq_T$.

Definition 3 (Justification Truth Value). *Let* BEval *be a branch evaluation. Consider a (D, Σ)-justification $G = (V, E, \lambda, r)$, a node $v \in V$, and an interpretation I for D and Σ. The* value of G at v in I relative to BEval *is*

$$\mathsf{val}_{\mathsf{BEval}}(G, v, I) \;=\; \min\left\{I(\ell) \mid \ell \in \mathsf{support}_{\mathsf{BEval}}(G, v)\right\}.$$

The value of G in I relative to BEval*, denoted $\mathsf{val}_{\mathsf{BEval}}(G, I)$, is defined as $\mathsf{val}_{\mathsf{BEval}}(G, \mathsf{rt}(G), I)$.* ∎

[1] Since there are six ground rules with $T(e, a)$ as head, in a proper justification, each negative fact should have six children (with possibly duplicate labels). However, for readability, we omit such duplicate children. By duplicating all nodes with a negative label, this figure can be turned into a proper justification graph. The same holds for examples that follow.

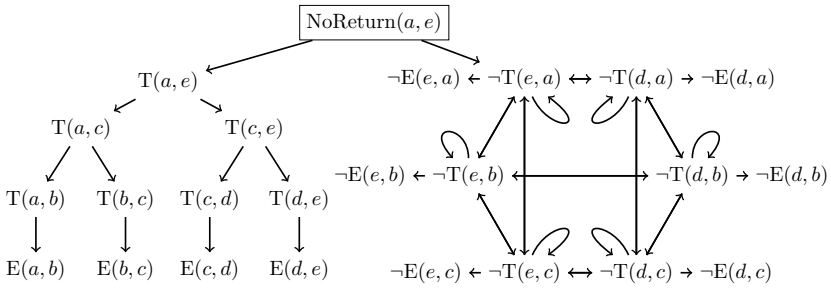

Figure 3: A good justification for the well-founded model of D_{ex} and Σ_{ex}, relative to wf.

Supported Truth Value of a Literal. We proceed to define the supported truth value of a literal in an interpretation relative to a branch evaluation.

Definition 4 (Literal Supported Truth Value). *Let* BEval *be a branch evaluation. Consider a Datalog$^\neg$ program* Σ *and a database* D *over* $\mathsf{edb}(\Sigma)$*. Let* $\ell \in \mathsf{literals}(D, \Sigma)$ *and* I *an interpretation for* D *and* Σ*. The* supported value of ℓ in I relative to BEval *is defined as*

$$\mathsf{sval}_{\mathsf{BEval}}(\ell, I) \;=\; \max\{\mathsf{val}_{\mathsf{BEval}}(G, I) \mid G \text{ is a justification of } \ell \text{ w.r.t. } D \text{ and } \Sigma\}. \quad \blacksquare$$

The Relativized Semantics. We can now define the semantics of a Datalog$^\neg$ program relative to a branch evaluation.

Definition 5 (Relativized Semantics of Datalog$^\neg$). *Let* BEval *be a branch evaluation. Consider a Datalog$^\neg$ program* Σ*. The* semantics of Σ relative to BEval *is the function* $[\![\cdot]\!]^{\Sigma}_{\mathsf{BEval}}$ *that assigns to every database* D *over* $\mathsf{edb}(\Sigma)$ *the following set of interpretations for* D *and* Σ*:*

$$[\![D]\!]^{\Sigma}_{\mathsf{BEval}} = \{I \mid \mathsf{sval}_{\mathsf{BEval}}(\ell, I) = I(\ell), \text{ for each literal } \ell \in \mathsf{literals}(D, \Sigma)\}. \quad \blacksquare$$

3.2 Well-Founded and Stable Model Semantics

With the relativized semantics in place, we can now define the well-founded and stable model semantics of Datalog$^\neg$ by choosing the right branch evaluation. In fact, we are going to define the well-founded and stable branch evaluations.

An infinite sequence $L = \ell_1, \ell_2, \ldots$ of literals is called

- *positive* if there exists $i \geq 1$ such that, for every $j \geq i$, $\ell_j \in \mathsf{base}^+(D, \Sigma)$,
- *negative* if there exists $i \geq 1$ such that, for every $j \geq i$, $\ell_j \in \mathsf{base}^-(D, \Sigma)$, and
- *mixed* otherwise, i.e., if, for every $i \geq 1$, there exist $j, k \geq i$ such that $\ell_j \in \mathsf{base}^+(D, \Sigma)$ and $\ell_k \in \mathsf{base}^-(D, \Sigma)$.

Since a branch of a justification G is, by definition, a sequence of literals, we can talk about positive, negative, and mixed infinite branches of G. The two branch evaluations of interest are defined by specifying the value (a literal or a truth value of $\{\mathsf{f}, \mathsf{u}, \mathsf{t}\}$) assigned to every finite branch, positive infinite branch, negative infinite branch, and mixed infinite branch of a justification. Interestingly, the definitions of these two branch evaluations are almost identical; they only differ in their treatment of mixed infinite branches.

Definition 6 (Well-Founded and Stable Branch Evaluations). *The* well-founded branch evaluation wf *and* stable branch evaluation st *are defined as follows:*

- *if* $L = \ell_1, \ldots, \ell_n$*, then* $\mathsf{wf}(L) = \mathsf{st}(L) = \ell_n$*,*
- *if* L *is a positive infinite branch, then* $\mathsf{wf}(L) = \mathsf{st}(L) = \mathsf{f}$*,*
- *if* L *is a negative infinite branch, then* $\mathsf{wf}(L) = \mathsf{st}(L) = \mathsf{t}$*,*
- *if* $L = \ell_1, \ell_2, \ldots$ *is a mixed infinite branch, then* $\mathsf{wf}(L) = \mathsf{u}$ *and* $\mathsf{st}(L) = \ell_i$*, where* $i > 0$ *is the smallest integer such that* ℓ_1 *and* ℓ_i *have different signs (i.e., one is a positive literal and the other a negative literal).* $\blacksquare$

Having the relativized semantics of Datalog$^\neg$ (see Definition 5) and the above concrete branch evaluations in place, we can easily define the "proof-theoretic" well-founded and stable model semantics of Datalog$^\neg$ programs. Although the well-founded semantics are straightforwardly defined by simply instantiating the branch evaluation, we need to be a bit more careful with the definition of the stable model semantics. This is because the standard stable model semantics, which we are interested in, considers only 2-valued interpretations, whereas the relativized semantics of Datalog$^\neg$ programs considers 3-valued interpretations. Thus, to properly define the stable model semantics, in addition to instantiating the branch evaluation in the relativized semantics, we should keep only the 2-valued interpretations.

Definition 7 (Semantics of Datalog$^\neg$ Programs). *Consider a Datalog$^\neg$ program* Σ*.*

- *The* well-founded semantics *of* Σ *is the function* $[\![\cdot]\!]^{\Sigma}_{\mathsf{wf}}$*.*
- *The* stable model semantics *of* Σ *is the function* $[\![\cdot]\!]^{\Sigma}_{\mathsf{st},2}$ *such that, for every database* D *over* $\mathsf{edb}(\Sigma)$*,*

$$[\![D]\!]^{\Sigma}_{\mathsf{st},2} = \{I \mid I \in [\![D]\!]^{\Sigma}_{\mathsf{st}} \text{ is a 2-valued interpretation of } D \text{ and } \Sigma\}. \quad \blacksquare$$

It is crucial to say that, given a Datalog$^\neg$ program Σ and a Database D over $\mathsf{edb}(\Sigma)$, $[\![D]\!]^{\Sigma}_{\mathsf{wf}}$ consists of a single interpretation for D and Σ, which is the unique well-founded model of D and Σ. Moreover, $[\![D]\!]^{\Sigma}_{\mathsf{st},2}$ consists, in general, of several 2-valued interpretations that are the stable models of D and Σ. Let us also note that $[\![D]\!]^{\Sigma}_{\mathsf{st}}$ is precisely the set of 3-valued or partial stable models of D and Σ. The above statements are folklore results in the justification theory literature and made recently explicit by (Marynissen 2022) (see Theorem 2.7.12), building on earlier results of (Denecker 1993).

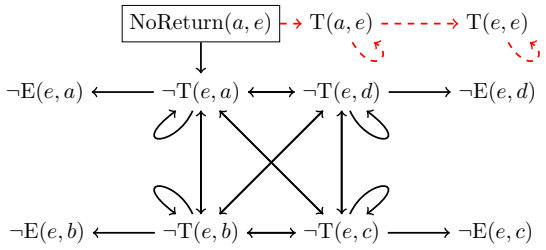

Figure 4: A justification that is not good for the well-founded model of D_{ex} and Σ_{ex}, relative to wf.

In the rest of the paper, we consider 2-valued stable models; this work is not concerned about 3-valued stable models. Thus, for brevity, we will write $[\![\cdot]\!]^{\Sigma}_{\mathsf{st}}$ meaning $[\![\cdot]\!]^{\Sigma}_{\mathsf{st},2}$.

4 Why-Provenance for Datalog$^{\neg}$

Consider a Datalog$^{\neg}$ program Σ and a database D over $\mathsf{edb}(\Sigma)$. For an interpretation I in the chosen semantics (we refer to arbitrary semantics, not necessarily well-founded or stable model semantics) of Σ on D and $\alpha \in \mathsf{base}^{+}(D, \Sigma)$, we want to explain why α is true in I. This is achieved via the notion of relativized why-provenance. Intuitively, the relativized why-provenance of α in I is the set that contains the relativized supports of all the justifications of α w.r.t. D and Σ that "comply" with I. We proceed to formalize this discussion. We first define what we mean by saying a justification "complies" with an interpretation.

Definition 8 (Good Justification). *Let* BEval *be a branch evaluation. Consider a Datalog*$^{\neg}$ *program* Σ *and a database* D *over* $\mathsf{edb}(\Sigma)$*. A* (D, Σ)*-justification* G *is* good for I relative to BEval*, where* I *is an interpretation for* D *and* Σ*, if it holds that* $\mathsf{val}_{\mathsf{BEval}}(G, I) = \mathsf{t}$. ∎

Equivalently, G is good for I relative to BEval if, for every $x \in \mathsf{support}_{\mathsf{BEval}}(G)$, $I(x) = \mathsf{t}$. Note that if G is good for I relative to BEval, then $\mathsf{support}_{\mathsf{BEval}}(G) \cap \{\mathsf{f}, \mathsf{u}\} = \emptyset$. It is also easy to verify that, for an interpretation $I \in [\![D]\!]^{\Sigma}_{\mathsf{BEval}}$ and an atom $\alpha \in \mathsf{base}^{+}(D, \Sigma)$, $I(\alpha) = \mathsf{t}$ iff there is a justification of α w.r.t. D and Σ that is good for I relative to BEval. This justifies the idea of using the justifications of α w.r.t. D and Σ that are good for I relative to BEval to explain why α is true in an interpretation $I \in [\![D]\!]^{\Sigma}_{\mathsf{BEval}}$.

Definition 9 (Relativized Why-Provenance for Datalog$^{\neg}$). *Let* BEval *be a branch evaluation. Consider a Datalog*$^{\neg}$ *program* Σ*, a database* D *over* $\mathsf{edb}(\Sigma)$*, an atom* $\alpha \in \mathsf{base}^{+}(D, \Sigma)$*, and an interpretation* $I \in [\![D]\!]^{\Sigma}_{\mathsf{BEval}}$*. The* why-provenance of α in I w.r.t. D and Σ relative to BEval *is*

$$\{\mathsf{support}_{\mathsf{BEval}}(G) \setminus \{\mathsf{t}\} \mid G \text{ is a justification of } \alpha \text{ w.r.t. } D \text{ and } \Sigma \text{ that is good for } I \text{ relative to } \mathsf{BEval}\}$$

which we denote by $\mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$. ∎

Intuitively, a set $L \in \mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$ should be interpreted as a "real" reason why α belongs to an interpretation $I \in [\![D]\!]^{\Sigma}_{\mathsf{BEval}}$. By "real" we mean two things: on the one hand, the set L is sufficient for α to hold (formalized in Proposition 1 below) and, on the other hand, that all the literals of L are actually used in a derivation of the atom α. We remark that if α is true in I, then $\mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma) \neq \emptyset$, i.e., there is at least one "real" reason for α. Also, if α is false in I, then $\mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$ is guaranteed to be empty, which is to be expected. Let us stress that we remove the truth value t from the relativized supports that are eventually collected as the members of why-provenance since it has no explanatory value. Moreover, in this way, our notion of why-provenance for Datalog$^{\neg}$ properly generalizes the notion of why-provenance for positive Datalog. Henceforth, for brevity, we write $\mathsf{support}^{-\mathsf{t}}_{\mathsf{BEval}}(G)$ for the set $\mathsf{support}_{\mathsf{BEval}}(G) \setminus \{\mathsf{t}\}$. Here is a simple example of why-provenance for the concrete cases where $\mathsf{BEval} \in \{\mathsf{wf}, \mathsf{st}\}$.

Example 3 (Example 2 continued)**.** *The example justification* G_{ex} *has only finite and negative infinite branches, meaning that* wf *and* st *will coincide for this justification (this actually holds for all stratified programs). With* $I^{wf}_{D,\Sigma}$ *being the well-founded model of* D *and* Σ *(i.e., the only element of* $[\![D]\!]^{\Sigma}_{\mathsf{wf}}$*), then* G_{ex} *is good for* $I^{wf}_{D_{\mathsf{ex}},\Sigma_{\mathsf{ex}}}$ *relative to* wf*, and thus*

$$\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G_{\mathsf{ex}}) = \{\mathrm{E}(a,b), \mathrm{E}(b,c), \mathrm{E}(c,d), \mathrm{E}(d,e), \neg\mathrm{E}(e,a), \neg\mathrm{E}(e,b), \neg\mathrm{E}(e,c), \neg\mathrm{E}(d,a), \neg\mathrm{E}(d,b), \neg\mathrm{E}(d,c)\}$$

belongs to $\mathsf{why}_{\mathsf{wf}}(\mathrm{NoReturn}(a,e), D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$*. There are actually further justifications of* $\mathrm{NoReturn}(a,e)$ *that are good for* $I^{wf}_{D_{\mathsf{ex}},\Sigma_{\mathsf{ex}}}$ *relative to* wf*. For instance, we also have that*

$$\{\mathrm{E}(a,c), \mathrm{E}(c,d), \mathrm{E}(d,e), \neg\mathrm{E}(e,a), \neg\mathrm{E}(e,b), \neg\mathrm{E}(e,c), \neg\mathrm{E}(e,d)\}$$

belongs to $\mathsf{why}_{\mathsf{wf}}(\mathrm{NoReturn}(a,e), D_{\mathsf{ex}}, \Sigma_{\mathsf{ex}})$*. However, there are justifications of* $\mathrm{NoReturn}(a,e)$ *that are* not *good for* $I^{wf}_{D_{\mathsf{ex}},\Sigma_{\mathsf{ex}}}$ *relative to* wf*; Figure 4 shows such a justification due to the infinite positive branches (dashed edges).* ∎

Coming back to our discussion that a set of literals $L \in \mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$ should be seen as a "real" reason why α belongs to an $I \in [\![D]\!]^{\Sigma}_{\mathsf{BEval}}$, we would like to recall that, inspired by (Glavic 2021), the authors of (Bogaerts, Jakubowski, and Van den Bussche 2024) introduced a notion of sufficiency as the minimal requirement for any provenance notion. Intuitively, if $L \in \mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$, then α is not only true in I, but also in every other BEval model for any other database that is consistent with L. We can transfer this notion to our setting and show that it holds.

Proposition 1. *Let* BEval *be a branch evaluation. Consider a Datalog*$^{\neg}$ *program* Σ*, a database* D*,* $\alpha \in \mathsf{base}^{+}(D, \Sigma)$*, and* $I \in [\![D]\!]^{\Sigma}_{\mathsf{BEval}}$*. Assume that* $L \in \mathsf{why}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$*,* D' *is a database such that* $\mathsf{dom}(D) = \mathsf{dom}(D')$*, and* $I' \in [\![D']\!]^{\Sigma}_{\mathsf{BEval}}$*. If* $I'(\ell) = \mathsf{t}$ *for each* $\ell \in L$*, then* $I'(\alpha) = \mathsf{t}$.

4.1 Problems of Interest

We are interested in pinpointing the inherent complexity of the problem of computing the why-provenance of an atom relative to wf and st. To this end, we concentrate on two

decision problems, one for each branch evaluation. Before we introduce those problems, let us concentrate for a moment on the notion of why-provenance relative to wf. As already discussed above, given a Datalog$^\neg$ program Σ and a database D over $\mathsf{edb}(\Sigma)$, it holds that $[\![D]\!]^\Sigma_{\mathsf{wf}}$ consists of a single interpretation for D and Σ, which is the unique well-founded model of D and Σ; we denote this interpretation as $I^{wf}_{D,\Sigma}$. With this in mind, one may wonder whether we need to explicitly refer to this interpretation in the definition of why-provenance relative to wf, or we can simply talk about the why-provenance of an atom. Note that in the case of positive Datalog, we simply talk about the why-provenance of an atom α, meaning implicitly the why-provenance of α in the unique minimal model of the given database and Datalog program. It turns out that this is actually possible. Such an alternative definition of why-provenance relative to the branch evaluation wf is provided by the following technical lemma, in which we call a justification G *good relative to* wf if it holds that $\mathsf{support}_{\mathsf{wf}}(G) \cap \{\mathsf{f}, \mathsf{u}\} = \emptyset$.

Lemma 1. *Consider a Datalog$^\neg$ program Σ, a database D over $\mathsf{edb}(\Sigma)$, and an atom $\alpha \in \mathsf{base}^+(D, \Sigma)$. It holds that $\mathsf{why}_{\mathsf{wf}}(\alpha, I^{wf}_{D,\Sigma}, D, \Sigma)$ coincides with the family of sets*

$$\{\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G) \mid G \text{ is a justification of } \alpha \text{ w.r.t. } D \text{ and } \Sigma \text{ that is good relative to } \mathsf{wf}\}.$$

By Lemma 1, we can simply refer to the *why-provenance of α w.r.t. D and Σ relative to* wf, which we denote by $\mathsf{why}_{\mathsf{wf}}(\alpha, D, \Sigma)$. As said above, our goal is to pinpoint the inherent complexity of the problem of computing the why-provenance of an atom relative to wf or st. To this end, we need to study the complexity of recognizing whether a certain set of literals belongs to the why-provenance relative to the branch evaluation of interest, i.e., whether a set of literals is an explanation. More precisely, we are interested in the *data complexity* of the problems where the Datalog$^\neg$ program Σ is fixed. This leads to the following two problems; recall that we are only concerned about 2-valued stable models and, by abuse of notation, we write $[\![D]\!]^\Sigma_{\mathsf{st}}$ for $[\![D]\!]^\Sigma_{\mathsf{st},2}$.

PROBLEM :	$\mathsf{Why\text{-}Provenance}_{\mathsf{wf}}[\Sigma]$
INPUT :	A database D over $\mathsf{edb}(\Sigma)$, an atom $\alpha \in \mathsf{base}^+(D, \Sigma)$, and a set of literals $L \subseteq \mathsf{literals}(D, \Sigma)$.
QUESTION :	Does $L \in \mathsf{why}_{\mathsf{wf}}(\alpha, D, \Sigma)$?

PROBLEM :	$\mathsf{Why\text{-}Provenance}_{\mathsf{st}}[\Sigma]$
INPUT :	A database D over $\mathsf{edb}(\Sigma)$, an atom $\alpha \in \mathsf{base}^+(D, \Sigma)$, an interpretation $I \in [\![D]\!]^\Sigma_{\mathsf{st}}$, and a set of literals $L \subseteq \mathsf{literals}(D, \Sigma)$.
QUESTION :	Does $L \in \mathsf{why}_{\mathsf{st}}(\alpha, I, D, \Sigma)$?

We say that $\mathsf{Why\text{-}Provenance}_{\mathsf{BEval}}$ is in a complexity class C in data complexity if, for every Datalog$^\neg$ program Σ, $\mathsf{Why\text{-}Provenance}_{\mathsf{BEval}}[\Sigma]$ is in C, and C-hard in data complexity if $\mathsf{Why\text{-}Provenance}_{\mathsf{BEval}}[\Sigma]$ is C-hard, for some Σ.

5 Complexity of Why-Provenance

The goal of this section is to pinpoint the data complexity of $\mathsf{Why\text{-}Provenance}_{\mathsf{BEval}}$, for $\mathsf{BEval} \in \{\mathsf{wf}, \mathsf{st}\}$. We show that the problems in question are intractable.

Theorem 1. $\mathsf{Why\text{-}Provenance}_{\mathsf{BEval}}$ *is NP-complete in data complexity, for each* $\mathsf{BEval} \in \{\mathsf{wf}, \mathsf{st}\}$.

The NP-hardness is inherited from the NP-hardness (in data complexity) of the why-provenance problem for positive Datalog (Calautti et al. 2024b); note that the latter is true even if the recursion is linear, i.e., each Datalog rule mentions at most one intensional predicate is in its body. The non-trivial task is to show the upper bound, i.e., to prove that for every Datalog$^\neg$ program Σ, $\mathsf{Why\text{-}Provenance}_{\mathsf{BEval}}[\Sigma]$, where $\mathsf{BEval} \in \{\mathsf{wf}, \mathsf{st}\}$, is in NP. Let us first discuss how this is shown for the simpler case of well-founded semantics.

5.1 Well-Founded Semantics

Our goal is to establish the following technical result, which states that if there is a justification of some atom that is good relative to wf, then there is one with the same support relative to wf of polynomial size in the size of the database.

Proposition 2. *For every Datalog$^\neg$ program Σ, there is a polynomial function $p_\Sigma : \mathbb{N} \to \mathbb{N}$ such that, for every database D over $\mathsf{edb}(\Sigma)$, atom $\alpha \in \mathsf{base}^+(D, \Sigma)$, and set of literals $L \subseteq \mathsf{literals}(D, \Sigma)$, the following are equivalent:*

1. *There is a justification G of α w.r.t. D and Σ that is good relative to* wf *with $L = \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G)$.*
2. *There is a justification $G = (V, E, \lambda, r)$ of α w.r.t. D and Σ that is good relative to* wf *with $L = \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G)$ and having at most $p_\Sigma(|D|)$ nodes, i.e., $|V| \leq p_\Sigma(|D|)$.*

This leads to an easy guess-and-check algorithm that runs in polynomial time for $\mathsf{Why\text{-}Provenance}_{\mathsf{wf}}[\Sigma]$, where Σ is a Datalog$^\neg$ program, and thus, $\mathsf{Why\text{-}Provenance}_{\mathsf{wf}}[\Sigma]$ is in NP. In particular, given a database D over $\mathsf{edb}(\Sigma)$, an atom $\alpha \in \mathsf{base}^+(D, \Sigma)$, and a set of literals $L \subseteq \mathsf{literals}(D, \Sigma)$:

1. Guess a labeled rooted directed graph $G = (V, E, \lambda, r)$, with $\lambda : V \to \mathsf{literals}(D, \Sigma)$, such that $|V| \leq p_\Sigma(|D|)$.
2. Verify that G is a justification of α w.r.t. D and Σ that is good relative to wf with $L = \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G)$, i.e., all leaves are in D and no cycle contains a positive fact.

Clearly, both steps take polynomial time in $|D|$, as needed.

Proof Idea for Proposition 2

We proceed to discuss the high-level idea underlying Proposition 2. We first observe that the direction (2) $\Rightarrow$ (1) holds trivially. Let us then discuss the proof idea for (1) $\Rightarrow$ (2). This direction is shown by proving that a justification G of α w.r.t. D and Σ that is good relative to wf can always be converted into a justification G' of α w.r.t. D and Σ that is good relative to wf such that $\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G')$ and G' has polynomially many nodes in $|D|$. The conversion of G into G' is performed via three steps:

Step 1: Unraveling. We first unravel G into a tree-shaped justification G^u that is good relative to wf such that $\mathsf{support}_{\mathsf{wf}}(G) = \mathsf{support}_{\mathsf{wf}}(G^u)$. Note that by tree-shaped we mean that G^u is a (potentially infinite) rooted

directed tree, i.e., a rooted directed acyclic graph whose underlying undirected graph is a tree, with root $\mathsf{rt}(G)$.

Step 2: Path Shortening. Then, for each literal $\ell \in \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^u)$, we fix an arbitrary leaf v_ℓ of G^u that is labeled with ℓ, and then carefully shorten the unique path from the root of G^u to v_ℓ so that (i) the shortened path from the root of G^u to v_ℓ has only polynomially many nodes in $|D|$, and (ii) the obtained rooted tree G^s remains a tree-shaped justification of α w.r.t. D and Σ that is good relative to wf with $\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^u) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^s)$.

Step 3: Disambiguation. Finally, we reduce the size of the part of G^s other than the shortened paths. To this end, we consider all the subtrees of G^s rooted at a node that is not part of a shortened path, but so that its parent occurs in a shortened path, and convert them into unambiguous graphs, i.e., graphs where each node has a different label (and thus, only polynomially many nodes can occur since there are only polynomially many labels w.r.t. $|D|$) in such a way that the obtained rooted directed graph G^d is a justification of α w.r.t. D and Σ that is good relative to wf and $\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^s) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^d)$. We stress that during the disambiguation of a subtree of G^s we may lose some of the labels that label its leaves. However, the final support is still preserved since all the literals of $\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^s)$ occur in $\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^d)$ due to the shortened paths that are all present in G^d and the labels of their leaves precisely form $\mathsf{support}^{-\mathsf{t}}_{\mathsf{wf}}(G^s)$.

At this point, let us note that a result analogous to Proposition 2 has been established in (Calautti et al. 2024b), where it is shown that why-provenance for Datalog without negation is in NP, by relying on a construction that converts a proof tree of an atom into a "small" one that has the same support. Despite some high-level similarities, our construction is significantly more involved since we need to deal with justifications of which the unravelling is an *infinite* tree-shaped justification, whereas for positive Datalog one only needs to deal with finite proof trees.

5.2 Stable Model Semantics

We now proceed with the more complex case of stable model semantics. Let I be a stable model of a database D and a Datalog$^\neg$ program Σ. As for the well-founded semantics, the goal is to show that if there is a justification of some atom w.r.t. D and Σ that is good for I relative to st, then there is one with the same support that is "small".

Proposition 3. *For every Datalog$^\neg$ program Σ, there is a polynomial function $p_\Sigma : \mathbb{N} \to \mathbb{N}$ such that, for every database D over $\mathsf{edb}(\Sigma)$, interpretation $I \in [\![D]\!]^\Sigma_{\mathsf{st}}$, atom $\alpha \in \mathsf{base}^+(D, \Sigma)$, and set of literals $L \subseteq \mathsf{literals}(D, \Sigma)$, the following are equivalent:*

1. *There is a justification G of α w.r.t. D and Σ that is good for I relative to st with $L = \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G)$.*
2. *There is a justification $G = (V, E, \lambda, r)$ of α w.r.t. D and Σ that is good for I relative to st with $L = \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G)$ and $|V| \leq p_\Sigma(|D|)$.*

As for Proposition 2, Proposition 3 leads to an easy guess-and-check algorithm that runs in polynomial time for the problem $\mathsf{Why\text{-}Provenance}_{\mathsf{st}}[\Sigma]$, where Σ is a Datalog$^\neg$ program, and thus, $\mathsf{Why\text{-}Provenance}_{\mathsf{st}}[\Sigma]$ is in NP, as needed.

Proof Idea for Proposition 3

We proceed to discuss the high-level idea underlying the direction (1) $\Rightarrow$ (2) of Proposition 3; the direction (2) $\Rightarrow$ (1) holds trivially. This is shown by proving that a justification G of α w.r.t. D and Σ that is good for I relative to st can always be converted into a justification G' of α w.r.t. D and Σ that is good for I relative to st such that $\mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G')$ and G' has polynomially many nodes in $|D|$. The conversion of G into G' is done via an intricate version of the 3-step construction described above for showing Proposition 2. The key reason for the additional complexity is the fact that we now need to carefully deal with infinite paths in G that witness mixed infinite branches since, unlike the support of G relative to wf, the support of G relative to st may contain literals that use an intensional predicate due to mixed infinite branches, and those literals should be preserved. The conversion of G into G' is performed by applying the following four steps:

Step 1: Unraveling. We first unravel G into a tree-shaped justification G^u that is good for I relative st such that $\mathsf{support}_{\mathsf{st}}(G) = \mathsf{support}_{\mathsf{st}}(G^u)$.

Step 2: Path Shortening. Then, for each literal $\ell \in \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^u)$, we fix a path P_ℓ starting at $\mathsf{rt}(G^u)$ that witnesses a branch B_ℓ of G^u with $\mathsf{st}(B_\ell) = \ell$ and we define v_ℓ to be the first node on P_ℓ with label ℓ. Note that if ℓ is over an intensional predicate, then B_ℓ will be a mixed infinite branch, and if ℓ is over an extensional predicate, then P_ℓ is finite and v_ℓ is its leaf. We then carefully shorten, for each literal $\ell \in \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^u)$, the unique path from the root of G^u to v_ℓ in such a way that (i) each shortened path has only polynomially many nodes in $|D|$, and (ii) the obtained rooted tree G^s remains a tree-shaped justification of α w.r.t. D and Σ that is good for I relative to st with $\mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^u) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^s)$. Let us stress that to ensure the last equality, we need to take care of subtleties related to mixed infinite branches that are not present when we deal with well-founded semantics.

Step 3: Looping. Observe that, for $\ell \in \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^s)$ that mentions an intensional predicate, although the subpath $\mathsf{rt}(G^u), \ldots, v_\ell$ of the infinite path P_ℓ fixed in the previous step has been shortened, its infinite tail P^t_ℓ starting at v_ℓ remains unchanged. This step is responsible for making such an infinite tail P^t_ℓ finite by carefully deleting an infinite subtree rooted at some node v^ℓ_{end} of P^t_ℓ, and then connecting the parent u^ℓ of v^ℓ_{end} to a properly chosen ancestor v^ℓ_{start} of it that has the same label as v^ℓ_{end}, thus creating the loop $C_\ell = v^\ell_{start}, \ldots, u^\ell, v^\ell_{start}$, in such a way that (i) C_ℓ has only polynomially many nodes in $|D|$, and (ii) the obtained rooted graph G° is a justification of α w.r.t. D and Σ that is good for I relative to st with $\mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^s) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^\circ)$. Note that such a looping step is not needed for the well-founded semantics since we do not deal with mixed infinite branches.

Step 4: Disambiguation. Finally, we reduce the size of the part of G° other than the shortened paths and the created loops, in order to obtain a justification graph G^d of α w.r.t. D and Σ that is good for I relative to st and $\mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^s) = \mathsf{support}^{-\mathsf{t}}_{\mathsf{st}}(G^d)$. This step proceeds as the analogous disambiguation step in the construction for the well-founded semantics, modulo some subtleties that are again related to mixed infinite branches.

6 Why-Not-Provenance for Datalog$^\neg$

So far, we have focused on the question why an atom α is true in an interpretation of interest I. When the latter does not hold, it is natural to ask why α is *not* true in I. Whenever we deal with 2-valued interpretations (e.g., in the case of stable model semantics), α is not true in I iff $\neg\alpha$ is true in I. Therefore, the question why α is not true in I is the same as asking why $\neg\alpha$ is true in I. This tells us that the why-not-provenance of α in I can be defined as the why-provenance of $\neg\alpha$ in I. However, when we deal with arbitrary 3-valued interpretations (such as in the case of well-founded semantics), the situation is a bit more subtle. An atom α that is not true in a 3-valued interpretation I can be false or unknown in I, and thus, no justification of $\neg\alpha$ that is good for I relative to the branch evaluation of interest might exist. This need not be a problem since, for a branch evaluation BEval, Definition 5 ensures that, whenever α is not true in $I \in [\![D]\!]^\Sigma_{\mathsf{BEval}}$ for a database D and a Datalog$^\neg$ program Σ, its supported value in I is either f or u. This leads to the notion of possible justification, which is analogous to that of good justification.

Definition 10 (Possible Justification). *Let* BEval *be a branch evaluation. Consider a Datalog$^\neg$ program* Σ *and a database* D *over* $\mathsf{edb}(\Sigma)$*. A* (D, Σ)*-justification* G *is* possible for I relative to BEval*, where* I *is an interpretation for* D *and* Σ*, if* $\mathsf{val}_{\mathsf{BEval}}(G, I) \geq \mathsf{u}$. ∎

Equivalently, G is possible for I relative to BEval if, for every $x \in \mathsf{support}_{\mathsf{BEval}}(G)$, $I(x) \geq \mathsf{u}$. Hence, we can use the justifications of $\neg\alpha$ w.r.t. D and Σ that are possible for I relative to BEval, instead of the good ones, to explain why α is not true in an interpretation $I \in [\![D]\!]^\Sigma_{\mathsf{BEval}}$.

Definition 11 (Relativized Why-Not-Provenance). *Let* BEval *be a branch evaluation. Consider a Datalog$^\neg$ program* Σ*, a database* D *over* $\mathsf{edb}(\Sigma)$*,* $\alpha \in \mathsf{base}^+(D, \Sigma)$*, and an interpretation* $I \in [\![D]\!]^\Sigma_{\mathsf{BEval}}$*. The* why-not-provenance of α in I w.r.t. D and Σ relative to BEval *is defined as*

$$\{\mathsf{support}_{\mathsf{BEval}}(G) \setminus \{\mathsf{t}, \mathsf{u}\} \mid G \text{ is a justification of } \neg\alpha \text{ w.r.t. } D \text{ and } \Sigma \text{ that is possible for } I \text{ relative to } \mathsf{BEval}\}$$

which we denote by $\mathsf{whynot}_{\mathsf{BEval}}(\alpha, I, D, \Sigma)$. ∎

6.1 Problems of Interest

We are interested in pinpointing the inherent data complexity of the problem of computing the why-not-provenance of an atom relative to the branch evaluations wf and st. Before introducing the problems of interest, let us establish a result, analogous to Lemma 1, that provides an alternative definition of why-not-provenance relative to wf that does not explicitly refer to the underlying well-founded model; we call a justification G *possible relative to* wf if $\mathsf{f} \notin \mathsf{support}_{\mathsf{wf}}(G)$.

Lemma 2. *Consider a Datalog$^\neg$ program* Σ*, a database* D *over* $\mathsf{edb}(\Sigma)$*, and an atom* $\alpha \in \mathsf{base}^+(D, \Sigma)$*. It holds that* $\mathsf{whynot}_{\mathsf{wf}}(\alpha, I^{wf}_{D,\Sigma}, D, \Sigma)$ *coincides with the family of sets*

$$\{\mathsf{support}_{\mathsf{wf}}(G) \setminus \{\mathsf{t}, \mathsf{u}\} \mid G \text{ is a justification of } \neg\alpha \text{ w.r.t. } D \text{ and } \Sigma \text{ that is possible relative to } \mathsf{wf}\}.$$

By Lemma 2, we can simply refer to the *why-not-provenance of* α *w.r.t.* D *and* Σ *relative to* wf, denoted by $\mathsf{whynot}_{\mathsf{wf}}(\alpha, D, \Sigma)$. As said above, our goal is to pinpoint the inherent complexity of the problem of computing the why-not-provenance of an atom relative to wf or st. To this end, as for why-provenance, we study the complexity of recognizing whether a certain set of literals belongs to the why-not-provenance relative to the branch evaluation of interest. More precisely, we are interested in the *data complexity* of the problems where the Datalog$^\neg$ program Σ is fixed. This leads to the problems $\mathsf{Why\text{-}Not\text{-}Provenance}_{\mathsf{wf}}[\Sigma]$ and $\mathsf{Why\text{-}Not\text{-}Provenance}_{\mathsf{st}}[\Sigma]$ defined as expected. We can further define, for each $\mathsf{BEval} \in \{\mathsf{wf}, \mathsf{st}\}$, when the problem $\mathsf{Why\text{-}Not\text{-}Provenance}_{\mathsf{BEval}}$ is C-complete in data complexity, for a complexity class C, in the obvious way. Then:

Theorem 2. $\mathsf{Why\text{-}Not\text{-}Provenance}_{\mathsf{BEval}}$ *is NP-complete in data complexity, for each* $\mathsf{BEval} \in \{\mathsf{wf}, \mathsf{st}\}$.

The above result is obtained via a proof similar to the one given for Theorem 1. For $\mathsf{BEval} = \mathsf{st}$, the proof is actually the same since in this case the notions of good and possible justification coincide. For $\mathsf{BEval} = \mathsf{wf}$, the proof is the same modulo the fact that we need to work with justifications of which the support does not contain the value f instead of the values u and f, and some further details concerning the disambiguation step.

7 Conclusion and Future Work

We have introduced the notion of why(-not)-provenance for Datalog with negation and studied the data complexity of the associated decision problem under the well-founded and stable model semantics. Interestingly, the proposed framework based on justification theory is sufficiently powerful to cover a plethora of other semantics by appropriately choosing the branch evaluation. For instance, we can directly use the framework to obtain the notion of why(-not)-provenance for partial stable model semantics (Przymusinski 1991), or for co-inductive programming semantics (Simon et al. 2006).

Justification theory allows using alternative branch evaluations for defining the same semantics; see, for example, Propositions 1 and 2 in (Marynissen, Bogaerts, and Denecker 2021). From a why(-not)-provenance perspective, this opens up opportunities for future research concerning the quality of the obtained explanations. In particular, different branch evaluations will result in different definitions of why(-not)-provenance, and thus, we can explore how to define alternative explanations. We are currently working in this direction for the case of stable model semantics.

Acknowledgements

This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (projects G064925N & G0B2221N and fellowship 11A2R26N).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Bogaerts, B.; Jakubowski, M.; and Van den Bussche, J. 2024. Postulates for provenance: Instance-based provenance for first-order logic. *Proc. ACM Manag. Data* 2(2):95.

Buneman, P.; Khanna, S.; and Tan, W. C. 2001. Why and where: A characterization of data provenance. In *ICDT*, 316–330.

Cabalar, P., and Fandinno, J. 2016. Justifications for programs with disjunctive and causal-choice rules. *Theory Pract. Log. Program.* 16(5–6):587–603.

Cabalar, P.; Fandinno, J.; and Fink, M. 2014. Causal graph justifications of logic programs. *Theory Pract. Log. Program.* 14(4–5):603–618.

Calautti, M.; Livshits, E.; Pieris, A.; and Schneider, M. 2024a. Below and above why-provenance for datalog queries. *Proc. ACM Manag. Data* 2(5):211:1–211:21.

Calautti, M.; Livshits, E.; Pieris, A.; and Schneider, M. 2024b. The complexity of why-provenance for datalog queries. *Proc. ACM Manag. Data* 2(2):83.

Calautti, M.; Livshits, E.; Pieris, A.; and Schneider, M. 2024c. Computing the why-provenance for datalog queries via SAT solvers. In *AAAI*.

Damásio, C. V.; Analyti, A.; and Antoniou, G. 2013. Justifications for logic programming. In *LPNMR*, 530–542.

Denecker, M., and De Schreye, D. 1993. Justification semantics: A unifiying framework for the semantics of logic programs. In *LPNMR*, 365–379.

Denecker, M.; Brewka, G.; and Strass, H. 2015. A formal theory of justifications. In *LPNMR*, 250–264.

Denecker, M. 1993. *Knowledge Representation and Reasoning in Incomplete Logic Programming*. Ph.D. Dissertation, Katholieke Universiteit Leuven, Belgium.

Deutch, D.; Milo, T.; Roy, S.; and Tannen, V. 2014. Circuits for datalog provenance. In *ICDT*, 201–212.

Eiter, T., and Geibinger, T. 2025. A sequent calculus for answer set entailment. In *IJCAI*, 4463–4472.

Elhalawati, A.; Krötzsch, M.; and Mennicke, S. 2022. An existential rule framework for computing why-provenance on-demand for datalog. In *RuleML+RR*.

Esparza, J.; Luttenberger, M.; and Schlund, M. 2014. Fpsolve: A generic solver for fixpoint equations over semirings. In *CIAA*, 1–15.

Fandinno, J., and Schulz, C. 2019. Answering the "why" in answer set programming – a survey of explanation approaches. *Theory Pract. Log. Program.* 19(2):114–203.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/LSP*, 1070–1080.

Glavic, B. 2021. Data provenance. *Found. Trends Databases* 9(3-4):209–441.

Khamis, M. A.; Ngo, H. Q.; Pichler, R.; Suciu, D.; and Wang, Y. R. 2022. Convergence of datalog over (pre-) semirings. In *PODS*, 105–117. ACM.

Lee, S.; Ludäscher, B.; and Glavic, B. 2019. PUG: a framework and practical implementation for why and why-not provenance. *VLDB J.* 28(1):47–71.

Marynissen, S.; Bogaerts, B.; and Denecker, M. 2021. On the relation between approximation fixpoint theory and justification theory. In *IJCAI*, 1973–1980.

Marynissen, S. 2022. *Advances in Justification Theory*. Ph.D. Dissertation, Department of Computer Science, KU Leuven.

Pontelli, E., and Son, T. C. 2006. Justifications for logic programs under answer set semantics. In *Logic Programming*, 196–210.

Pontelli, E.; Son, T. c.; and Elkhatib, O. 2009. Justifications for logic programs under answer set semantics. *Theory Pract. Log. Program.* 9(1):1–56.

Przymusinski, T. C. 1991. Stable semantics for disjunctive programs. *New Gener. Comput.* 9(3/4):401–424.

Schulz, C., and Toni, F. 2013. Aba-based answer set justification. *Theory Pract. Log. Program.* 13(4–5 (Online Supplement)).

Schulz, C., and Toni, F. 2016. Justifying answer sets using argumentation. *Theory Pract. Log. Program.* 16(1):59–110.

Simon, L.; Mallya, A.; Bansal, A.; and Gupta, G. 2006. Coinductive logic programming. In *ICLP*, 330–345.

Zhao, D.; Subotic, P.; and Scholz, B. 2020. Debugging large-scale datalog: A scalable provenance evaluation strategy. *ACM Trans. Program. Lang. Syst.* 42(2):7:1–7:35.