Towards Constraint Provenance Games

Sean Riddle Sven Köhler Bertram Ludäscher

Department of Computer Science, University of California, Davis, CA 95616

{swriddle, svkoehler, ludaesch}@ucdavis.edu

Abstract

Provenance for positive queries is well understood and elegantly handled by *provenance semirings* [GKT07], which subsume many earlier approaches. However, the semiring approach does not extend easily to *why-not* provenance or, more generally, first-order queries with negation. An alternative approach is to view query evaluation as a game between two players who argue whether, for given database I and query Q, a tuple t is in the answer $Q(I)$ or not. For first-order logic, the resulting *provenance games* [KLZ13] yield a new provenance model that coincides with provenance semirings (*how provenance*) on positive queries, but also is applicable to first-order queries with negation, thus providing an elegant, uniform treatment of earlier approaches, *including* why-not provenance and negation. In order to obtain a finite answer to a why-not question, provenance games employ an active domain semantics and enumerate tuples that contribute to failed derivations, resulting in a domain dependent formalism. In this paper, we propose *constraint provenance games* as a means to address this issue. The key idea is to represent infinite answers (e.g., to why-not questions) by finite constraints, i.e., equalities and disequalities.

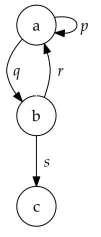
1. Introduction

Consider the relation $\texttt{hop}(x, y)$ in Fig. 1a and query $Q_{\texttt{3hop}} :=$

$$r_1: \quad \texttt{3hop}(X,Y) :- \texttt{hop}(X,Z_1), \texttt{hop}(Z_1,Z_2), \texttt{hop}(Z_2,Y).$$

$Q_{\texttt{3hop}}$ asks for pairs of nodes that are reachable via exactly three edges ("hops"). If we ask *why* and *how* a tuple such as $\texttt{3hop}(\texttt{a},\texttt{a})$ came about, we can use polynomials over a provenance semiring [GKT07, KG12] to get a precise answer, here: $p^3 + 2pqr$. In Fig. 1a we see that one can "go" from node $\texttt{a}$ to itself in three hops in distinct ways: (i) by using the edge p (= $\texttt{hop}(\texttt{a},\texttt{a})$, a self-loop) three times: $p{\cdot}p{\cdot}p$, or p^3 for short, (ii) by using the p edge once, followed by q (= $\texttt{hop}(\texttt{a},\texttt{b})$) and then r (= $\texttt{hop}(\texttt{b},\texttt{a})$), so $p{\cdot}q{\cdot}r$, or (iii) by following q, r, and then p, i.e., $q{\cdot}r{\cdot}p$. Since semiring provenance is commutative, $p{\cdot}q{\cdot}r + q{\cdot}r{\cdot}p = 2pqr$ as shown in the figure. Many prior provenance approaches can be understood as special provenance semirings: e.g., Trio provenance [BSHW06], why-provenance [BKT01], and lineage [CWW00], all yield coarser version of the provenance $p^3 + 2pqr$ of $\texttt{3hop}(\texttt{a},\texttt{a})$, i.e., $p + 2pqr$, $p + pqr$, and pqr, respectively [KG12].

Provenance through Games. In Fig. 1c we see that $\texttt{3hop}(\texttt{c},\texttt{a})$ is absent, so $\texttt{3hop}(\texttt{c},\texttt{a})$ is false. We cannot use semiring provenance to explain *why-not*, since the approach is not defined for negative queries and extensions for negation (or set-difference) are not obvious [GP10, GIT11, ADT11a, ADT11b]. On the other hand, if an approach *can* explain the provenance of $\neg A$, this naturally provides a why-not explanation for A. In [KLZ13] we proposed an alternative model of provenance that naturally supports negation. Consider the graph in Fig. 1d. It can be understood as the move graph of a *query evaluation game* in which two players argue whether or not



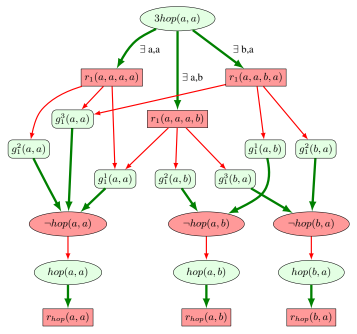
(a) input I ...

hop		
a	a	p
a	b	q
b	a	r
b	c	s

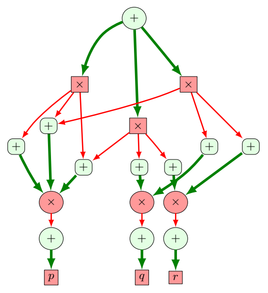
(b) ... annotated.

3hop		
a	a	$p^3 + 2pqr$
a	b	$p^2q + q^2r$
a	c	pqs
b	a	$p^2r + qr^2$
b	b	pqr
b	c	qrs

(c) $\texttt{3hop}$ with provenance.



(d) The game provenance of $\texttt{3hop}(\texttt{a},\texttt{a})$ ...



(e) ... is $p^3 + 2pqr$.

Figure 1: Each edge $\texttt{hop}(x, y)$ in the input graph I in (a) is annotated (p, q, r, ...) in (b). The answer to $Q_{\texttt{3hop}}$ is shown in (c) with provenance polynomials [KG12]. The game provenance [KLZ13], e.g., of $\texttt{3hop}(\texttt{a},\texttt{a})$ in (d) corresponds to the semiring provenance polynomial in (c): see (e).

a tuple $t \in Q(I)$. If a player wants to prove that $t = \texttt{3hop}(\texttt{a},\texttt{a})$ is in $Q_{\texttt{3hop}}$, she needs to move to a ground rule r with t in the head, thereby claiming that this rule instance is deriving t. In Fig. 1d, there are three choices, starting from the root node $\texttt{3hop}(\texttt{a},\texttt{a})$: the move to $r_1(a,a,a,a)$, to $r_1(a,a,a,b)$, or to $r_1(a,a,b,a)$. Here $r_1(x,y,z_1,z_2)$ identifies ground instances of r_1. There are two $\forall$-quantified variables X and Y occurring in the head and body, and two (implicitly) $\exists$-quantified variables Z_1 and Z_2, occurring only in r_1's body. By moving to a ground instance of r_1 in the game, the player tries to pick values for the $\exists$-quantified variables that make the rule body true while deriving t in the head. For r_1, the middle edge $\texttt{hop}(z_1, z_2)$ fixes the bindings of Z_1 and Z_2. For the given database instance I, there are three choices that "work": (a,a), (a,b), and (b,a). This means that there are exactly three different ways to obtain $\texttt{3hop}(\texttt{a},\texttt{a})$ via r_1 over input I: if we choose the p-$\texttt{hop}$ (a,a) as the middle edge, we have $p{\cdot}p{\cdot}p$; for the q-$\texttt{hop}$ (a,b) we have $p{\cdot}q{\cdot}r$; and for the r-$\texttt{hop}$ (b,a) we have $q{\cdot}r{\cdot}p$.[1] The opponent can now challenge each of these claims, by selecting a subgoal

[1] Game provenance [KLZ13] can distinguish $p{\cdot}q{\cdot}r$ and $q{\cdot}r{\cdot}p$ and is thus even more fine-grained than the provenance semirings in [GKT07, KG12].

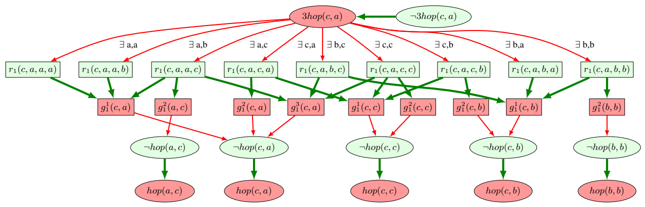


Figure 2: Why-not provenance for `3hop`(`c`, `a`) using provenance games.

g_1^i in the body of r_1, thus claiming that g_1^i is false and hence that the r_1 instance doesn't derive t. The first player can counter and demonstrate that g_1^i is true by selecting a rule instance or fact as evidence for g_1^i. The game proceeds in rounds until some player cannot move and thus loses (the opponent wins). In [KLZ13] it was shown how the provenance of a tuple t can be obtained via a regular path query over a solved game graph like the one in Fig. 1d: e.g., $p^3 + 2pqr$ for `3hop`(`a`, `a`) is represented by a solved game as shown in Fig. 1e: for positive queries, solved games represent semiring provenance by noting that won (green) and lost (red) positions correspond to "$+$" and "$\times$" operations, respectively (leaves represent input annotations, here: p, q, r, s) [KLZ13].

Why-Not Provenance and the Many Ways to Fail. Since games are inherently symmetric (one player's win is the opponent's loss and vice versa), the approach yields an elegant provenance model that unifies why and why-not provenance. Consider the (dark, red) node `3hop`(`c`, `a`) in Fig. 2. The color coding indicates that the position `3hop`(`c`, `a`) is lost (the atom is false), i.e., *all* outgoing moves to a node $r_1(x, y, z_1, z_2)$ lead to a position that is won for the opponent. There are 9 such positions, e.g., r_1(`c`, `a`, `c`, `b`) is one of them (third from the right). Recall that an instance of r_1 means that one can do a 3-hop from x to y (here: `c` to `a`) via intermediate nodes z_1 and z_2 (here: `c` and `b`). However, in the given database I in Fig. 1(a), there is no `hop`(`c`, z) – neither for z = `b` nor for any other z, since there are no outgoing moves from `c`. In this case, the opponent can successfully attack the goals in the body. Note how the *why-not* provenance of `3hop`(`c`, `a`) in Fig. 2 is similar but different from the *why* provenance of `3hop`(`a`, `a`) in Fig. 1: In order to show that `3hop`(`c`, `a`) is false, one has to show that *all* possible ways that it could be true are failing, i.e., for *all* z_1, z_2, the ground instances r_1(`c`, `a`, z_1, z_2) do *not* derive `3hop`(`c`, `a`) (since at least one goal in r_1's body is always false). In constrast, to prove that `3hop`(`a`, `a`) is true, it is sufficient to find *some* ground instance r_1(`a`, `a`, z_1, z_2) whose body is true. Earlier we saw that there are exactly three such instances, corresponding to $p \cdot p \cdot p + p \cdot q \cdot r + q \cdot r \cdot p$ ($= p^3 + 2pqr$).

Domain Dependence of Provenance Games. As seen, `3hop`(`a`, `a`) has three derivations, represented by the first provenance polynomial in Fig. 1(c) and the game provenance in Fig. 1(d) and (e). How many ways are there to show that `3hop`(`c`, `a`) is false (why-not provenance), or equivalently, that ¬ `3hop`(`c`, `a`) is true? If we annotate the leaves of the game graph in Fig. 2 with identifiers $u_1, \ldots, u_5$ for the five different `hop` tuples *missing* in I, we can construct a provenance expression that represents the many ways why `3hop`(`c`, `a`) is *not* in the answer. While this answer provides a comprehensive, instance-based why-not explanation, it also exhibits a problem with the current approach: In order to obtain finite (why and why-not) provenance answers for all first-order queries, game provenance employs an *active domain* semantics: e.g., the provenance game for $Q_{\texttt{3hop}}(I)$ considers only ground instances of r_1 over the active domain $adom(I) = \{a, b, c\}$. If additional elements $d, e, \ldots$ are added to I (e.g., via a disconnected graph component), the why-not provenance in Fig. 2 becomes incomplete and the provenance has to be recomputed for the larger domain.

Constraint Provenance Games. We propose to solve the problem of domain dependence by modifying provenance games so that they can handle certain infinite relations that can be finitely represented. For example, in addition to the finitely many reasons why `3hop`(`c`, `a`) fails over the active domain $adom(I)$, there are infinitely many others, if we consider new constants $d, e, \ldots$ outside of $adom(I)$. For example, let relation $R = \{a, b\}$ have two tuples $R(a)$ and $R(b)$. If we want to know *why-not* $R(c)$, we just point to $c \notin R$. But we could also return a more general answer for *why-not* $R(x)$ and say that $\neg R(x)$ is true for all x with $x \neq a \land x \neq b$ (not just for $x = c$). This approach is inspired by Chan's *Constructive Negation* [Cha88], a form of constraint logic programming [Stu95]. The key idea is to represent (potentially infinite) relations through constraints, i.e., Boolean combinations of equalities $x = c$ and disequalities $x \neq c$.

Overview and Contributions. Section 2 briefly explains how first-order queries are translated into games and how provenance is extracted from solved games. In Section 3 we describe the construction of *constraint provenance games*; additional details and examples are contained in the appendix. Our main contributions are: (i) game provenance provides a uniform treatment of why and why-not provenance for first-order logic (= relational algebra with set-difference); (ii) for positive queries, the approach captures the most informative semiring provenance [GKT07, KG12]; (iii) we develop a constraint provenance framework which yields *domain independent* provenance expressions, extending prior results [KLZ13]; and (iv) we implemented a prototype of constraint provenance games.

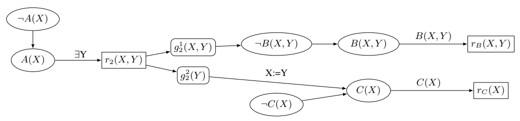
2. Provenance through Games

We first sketch how a query Q over database I gives rise to a game $G_{Q(I)}$ and how to obtain provenance from the solved game $G^{\lambda}_{Q(I)}$. Consider, e.g., input relations $\texttt{B}(X, Y)$ and $\texttt{C}(Y)$ and a relational query $Q_{\texttt{ABC}}$ with set-difference: $\texttt{A} \leftarrow \pi_X(\texttt{B} \bowtie (\pi_Y(\texttt{B}) \setminus \texttt{C}))$. It is well-known that any relational algebra query can be translated into a non-recursive Datalog$^\neg$ program. Here, we have $Q_{\texttt{ABC}}$ =

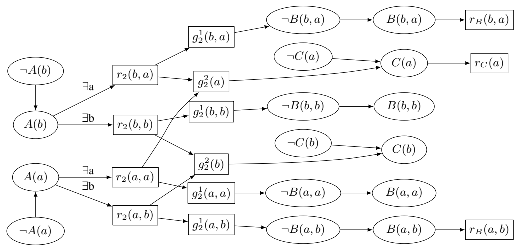
$$r_2: \quad \texttt{A}(X) \text{ :– } \texttt{B}(X, Y), \neg\texttt{C}(Y).$$

The key idea of provenance games is to understand query evaluation as a game between players I and II who argue whether or not a tuple is in the answer. In [KLZ13] we showed that the solved game is a representation of *why* (*why-not*) provenance of *answer* tuples (*missing* tuples), respectively. Fig. 3a shows the game template for $Q_{\texttt{ABC}}$: to prove that $\texttt{A}(x)$ is true, player I needs to find a rule instance of r_2, say $\texttt{A}(x)$:– $\texttt{B}(x, y), \neg\texttt{C}(y)$ which derives the desired tuple $\texttt{A}(x)$ and whose choice y for the $\exists$-quantified variable Y in the body satisfies all literals (subgoals) in the rule body. In the game template in Fig. 3a this corresponds to a move from $\texttt{A}(X)$ to $r_2(X, Y)$ while choosing a suitable domain value y for the $\exists$-quantified variable Y. Player II can challenge this claim by "attacking" one of the subgoals g in the rule body. If player I chose the "wrong" y for the instance $r_2(x, y)$, then II can always attack at least one subgoal that falsifies the body. The game continues in turns, until a player cannot move and loses, and the opponent wins.

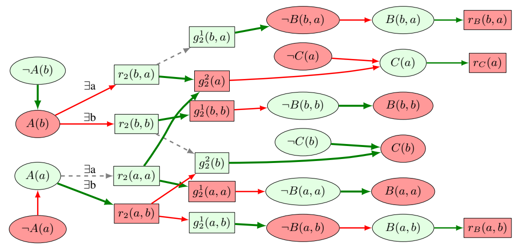
A game *template* G_Q for query Q contains *literal nodes* (oval; for atoms or their negation), *rule nodes* (boxes; for Datalog$^\neg$ rules), and *goal nodes* (rounded boxes; subgoals of rules): see Fig. 3a. Edge labels indicate a *condition* for a move: e.g., the label "$\exists Y$" between a literal node, say $\texttt{A}(X)$, and a rule node, say $r_2(X, Y)$, requires a player to pick a value y for the $\exists$-quantified variable Y when moving from an atom to the rule that derives it. Similarly, a condition "$X{:=}Y$" means that the current choice of Y becomes



(a) Game template for $Q_{\mathtt{ABC}}$: $\mathtt{A}(X) \mathrel{:-} \mathtt{B}(X,Y), \neg\mathtt{C}(Y)$.



(b) Instantiated $Q_{\mathtt{ABC}}$ game on $I = \{\mathtt{B}(a,b), \mathtt{B}(b,a), \mathtt{C}(a)\}$.



(c) Solved game: *lost* positions are (dark) red; *won* positions are (light) green. Provenance edges (= good moves) are solid. Bad moves are dashed and *not* part of the provenance. $\mathtt{A}(a)$ is *true* ($\mathtt{A}(b)$ is *false*) as it is *won* (*lost*) in the solved game; the game provenance explains *why* (*why-not*).

Figure 3: Provenance game for $Q_{\mathtt{ABC}}$. The well-founded model of $\mathtt{win}(X) \mathrel{:-} \mathtt{M}(X,Y), \neg\mathtt{win}(Y)$, applied to move graph $\mathtt{M}$, solves the game.

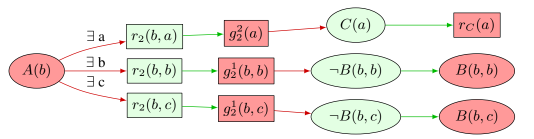


Figure 4: Altered subgraph of Fig. 3c after adding c to the active domain.

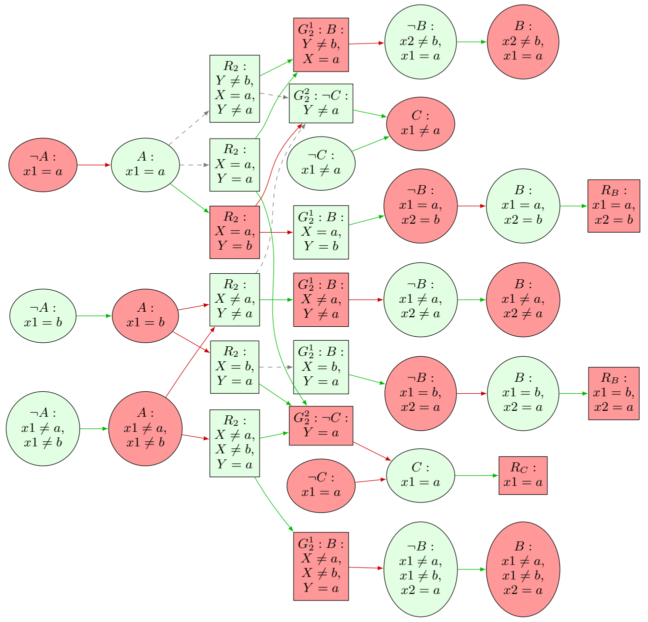


Figure 5: Constraint provenance game for $Q_{\mathtt{ABC}}$. Unlike in Figure 3, nodes may represent finite or infinite sets here.

the new binding for X; a condition "$\mathtt{B}(X,Y)$" means that a move is possible only if $\mathtt{B}(X,Y)$ is true in I for the current X, Y values.[2]

Given database I, a template can be instantiated yielding a *game graph* $G_{Q(I)}$ as in Fig. 3b. Note how template variables (e.g., Y) have been replaced by domain values (a or b), and that conditional edges (e.g., labeled "$\mathtt{C}(X)$") became unconditional edges (e.g., $\mathtt{C}(a) \rightarrow r_{\mathtt{C}}(a)$) or no edge at all (e.g., from $\mathtt{C}(b)$), depending on whether or not the condition holds in I. To extract why(-not) provenance from a game graph $G_{Q(I)}$ as in Fig. 3b, we need to *solve* the game first, i.e., determine which positions are *won* (light green) or *lost* (dark red); see Fig. 3c. There is a surprisingly simple and elegant solution: the (unstratified) Datalog$^{\neg}$ rule $Q_{\mathtt{wm}}$:=

$$\mathtt{win}(X) \mathrel{:-} \mathtt{move}(X,Y), \neg\mathtt{win}(Y)$$

when evaluated under the well-founded semantics [VGRS91] solves the game! Thus we can use $Q_{\mathtt{wm}}$ as a "game engine" to solve the provenance game with a move relation given by $G_Q(I)$.[3]

Finally, the solved game is a labeled graph $G^{\lambda}_{Q(I)}$, i.e., each node carries a new label λ, indicating whether a position is won (light green) or lost (dark red). As shown in [KLZ13], only edges from won to lost nodes (green) and lost to won nodes (red) are part of the provenance; other edges (grey, dashed in Fig. 3c) correspond to "bad moves" (invalid arguments in the query evaluation game) and are excluded from the provenance. The provenance subgraph of $G^{\lambda}_{Q(I)}$ thus consists only of edges that are matched by the regular path queries $(\mathtt{g.r})^{+}$ and $\mathtt{r.(g.r)}^{*}$, i.e., alternating sequences of green (winning) and red (delaying) moves [KLZ13].

3. Constraint Provenance Games

Consider the solved game graph of Fig. 3c. If the value $\mathtt{c}$ were added to the active domain, the provenance would be *incomplete*: e.g., to explain why-not $A(\mathtt{b})$ there are two $\exists\mathtt{a}, \exists\mathtt{b}$ branches emanating from $A(\mathtt{b})$. However, with $\mathtt{c}$ in the active domain there is a third $\exists\mathtt{c}$ branch via $r_2(\mathtt{b},\mathtt{c})$: see Fig. 4. We show that a modified game construction (Fig. 5) based on *constraints* can be used to automatically include such extensions of the active domain, thereby eliminating the domain dependence of the original approach.

Similarly, one could conclude from Fig. 2 that the absence of $\mathtt{3hop}(\mathtt{c},\mathtt{a})$ from the query answer is due entirely to the absence of $\mathtt{hop}(\mathtt{a},\mathtt{c})$, $\mathtt{hop}(\mathtt{c},\mathtt{a})$, $\mathtt{hop}(\mathtt{c},\mathtt{c})$, $\mathtt{hop}(\mathtt{c},\mathtt{b})$, and $\mathtt{hop}(\mathtt{b},\mathtt{b})$. Also this explanation, however, is complete only relative to the active domain: if $\mathtt{d}$ was introduced into the domain, new why-not answers such as $r_1(\mathtt{c},\mathtt{a},\mathtt{d},\mathtt{d})$ would have to be added to the provenance graph in Fig. 2. The new version of the provenance game (Fig. 9), however, takes care of this via a more general *constraint node* R_1 : $X{\neq}\mathtt{a}$, $X{\neq}\mathtt{b}$, $Z_1{\neq}\mathtt{c}$, $Z_1{\neq}\mathtt{a}$, $Z_1{\neq}\mathtt{b}$, $Z_2{\neq}\mathtt{c}$, $Z_2{\neq}\mathtt{a}$, $Z_2{\neq}\mathtt{b}$, $Y{\neq}\mathtt{c}$.

In constraint provenance games, nodes stand for *sets* of ground nodes. A constraint tuple such as "$\mathtt{3hop}(x,y)$: $x{=}\mathtt{a}, y{=}\mathtt{b}$" may stand for a single tuple (here: $\mathtt{3hop}(\mathtt{a},\mathtt{b})$), or for (possibly infinitely) many: e.g., "$\mathtt{3hop}(x,y)$: $x{\neq}\mathtt{a}, x{\neq}\mathtt{b}, y{=}\mathtt{a}$" stands for the set $\{\, \mathtt{3hop}(x,\mathtt{a}) \mid x \in D \setminus \{\mathtt{a},\mathtt{b}\} \,\}$ over any underlying domain D (finite or infinite).

[2] Readers familiar with logic programming semantics may recognize that provenance games mimic a form of SLD(NF) resolution.

[3] Indeed, both our prototypes use $Q_{\mathtt{wm}}$ to compute (constraint) provenance.

To that end, a tuple $R(\bar{X})$ with variables $\bar{X} = X_1, \ldots, X_n$ is associated with a Boolean expression over equalities of the form $X = c$, or disequalities of the form $X \neq c$. Thus, each (dis)equality is between a variable from $\bar{X}$ and a constant $c \in adom(I)$.

Since nodes in a constraint game no longer correspond to a single concrete value, but a constrained set, a tuple node being won in the solved game corresponds to the presence of all tuples which satisfy the node's constraint (the node is said to *admit* the tuples), and if lost to their absence. The advantage of this approach is that one can query provenance of tuples involving elements not in the active domain and provenance answers will stay correct in light of changes in the active domain.

Figure 9 (in the appendix) encompasses the why-not explanations that involve only the active domain, as well as the infinite explanations that can be generated when one considers values outside the active domain. The table below shows each explanation involving hypothetical tuples in the active domain and the corresponding rule node in Fig. 9. The rule nodes in Fig. 9 can be considered to be numbered from 1 on the left to 15 on the right. Each rule node is won, which agrees with the fact that each of the paths shown in Fig. 6 is only hypothetical.

Since all rule firings which would derive `3hop(c, a)` are won/unsatisfied, the tuple `3hop(c, a)` does not exist and the node indicating its positive presence in Fig. 9 (the source node) is accordingly lost.

Each of the rule nodes referenced in the table, which explain the negative provenance of a rule firing grounded in the active domain, also captures the rule non-satisfaction of an infinite set of possible variable bindings to elements possibly outside the active domain. Any constraint that has a variable that is only disequality-constrained represents an infinite set of firings. Consider the rule node: $R_1 : X{\neq}\mathtt{a}, X{\neq}\mathtt{b}, Z_1{=}\mathtt{a}, Z_2{=}\mathtt{a}, Y{=}\mathtt{a}$. This corresponds to the (hypothetical) 3hop path $\mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a} \xrightarrow{p} \mathtt{a} \xrightarrow{p} \mathtt{a}$ and the situation in which the edge t exist (see first row of Fig. 6). However, it also explains why the rule firing $d \rightarrow a \rightarrow a \rightarrow a$ is not successful. The explanation is the failure of the first goal of the rule. In the case of $X{=}c$, it represents that there are no outgoing edges from c. In the case of $X{=}d$ or any other invented value this is trivially true.

This shows that constraint provenance games do not suffer from the same problems as their fully-grounded counterparts. Provenance can be queried for any imaginable tuple, including one not in the active domain, and the provenance presented is still correct in the presence of a growing active domain.

$r_1(X, Y, Z_1, Z_2)$ [Fig. 2]	$X \rightarrow Z_1 \rightarrow Z_2 \rightarrow Y$ [Fig. 7]	Why−Not Provenance	R_1 Node [Fig. 9]
$r_1(\mathtt{c, a, a, a})$	$\mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a} \xrightarrow{p} \mathtt{a} \xrightarrow{p} \mathtt{a}$	$t \Rightarrow t{\cdot}p{\cdot}p$	2
$r_1(\mathtt{c, a, a, b})$	$\mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a} \xrightarrow{q} \mathtt{b} \xrightarrow{r} \mathtt{a}$	$t \Rightarrow t{\cdot}q{\cdot}r$	3
$r_1(\mathtt{c, a, a, c})$	$\mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a} \overset{u}{\dashrightarrow} \mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a}$	$t, u \Rightarrow t{\cdot}u{\cdot}t$	7
$r_1(\mathtt{c, a, c, a})$	$\mathtt{c} \overset{v}{\dashrightarrow} \mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a} \xrightarrow{p} \mathtt{a}$	$t, v \Rightarrow v{\cdot}t{\cdot}p$	14
$r_1(\mathtt{c, a, b, c})$	$\mathtt{c} \overset{w}{\dashrightarrow} \mathtt{b} \xrightarrow{s} \mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a}$	$t, w \Rightarrow w{\cdot}s{\cdot}t$	6
$r_1(\mathtt{c, a, c, c})$	$\mathtt{c} \overset{v}{\dashrightarrow} \mathtt{c} \overset{v}{\dashrightarrow} \mathtt{c} \overset{t}{\dashrightarrow} \mathtt{a}$	$t, v \Rightarrow v{\cdot}v{\cdot}t$	12
$r_1(\mathtt{c, a, c, b})$	$\mathtt{c} \overset{v}{\dashrightarrow} \mathtt{c} \overset{w}{\dashrightarrow} \mathtt{b} \xrightarrow{r} \mathtt{a}$	$v, w \Rightarrow v{\cdot}w{\cdot}r$	15
$r_1(\mathtt{c, a, b, a})$	$\mathtt{c} \overset{w}{\dashrightarrow} \mathtt{b} \xrightarrow{r} \mathtt{a} \xrightarrow{p} \mathtt{a}$	$w \Rightarrow w{\cdot}r{\cdot}p$	4
$r_1(\mathtt{c, a, b, b})$	$\mathtt{c} \overset{w}{\dashrightarrow} \mathtt{b} \overset{x}{\dashrightarrow} \mathtt{b} \xrightarrow{r} \mathtt{a}$	$w, x \Rightarrow w{\cdot}x{\cdot}r$	1

Figure 6: The nine r_1-instances in the first column correspond to those in Fig. 2 from left to right. The `3hop`-path is shown in the second column, with missing/hypothetical edges (dashed) t, u, v, w, x and existing edges p, q, r, s; see Fig. 7. The third column shows the why-not provenance of `3hop(c, a)`: e.g., if an edge t from `c` to `a` *were* present, there *would be* two derivations $t{\cdot}p{\cdot}p$ and $t{\cdot}q{\cdot}r$. The last column identifies the R_1 rule node (labeled from 1 to 15, left to right) in Fig. 9 which subsumes the corresponding rule node in Fig. 2.

4. Conclusions

In earlier work we proposed provenance games as an elegant and novel approach to unify why and why-not provenance [KLZ13]. The problem of domain *dependence* for why-not answers led us to develop our domain *independent* extension using concepts from constructive negation [Cha88]. This approach increases the complexity of individual nodes, but has the advantage that provenance can be queried that is not limited to the active domain, and a constraint provenance graph is still correct when considering a program executed under a larger active domain, unlike can occur in non-constraint games. This domain independent extension of provenance games [KLZ13] to use constraints is implemented as a prototype that uses a Datalog$^\neg$ engine to solve games via $Q_{\mathtt{wm}}$ (Sec. 2), and the Z3 theorem prover to simplify constraints (most figures in the paper and appendix are automatically generated by our prototype).

Acknowledgments. Supported in part by NSF awards IIS-1118088 and ACI-0830944.

References

[ADT11a] Y. Amsterdamer, D. Deutch, and V. Tannen. On the Limitations of Provenance for Queries With Difference. In *TaPP*, Heraklion, Crete, 2011.

[ADT11b] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, pp. 153–164. ACM, 2011.

[BKT01] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pp. 316–330. Springer, 2001.

[BSHW06] O. Benjelloun, A. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pp. 953–964, 2006.

[Cha88] D. Chan. Constructive Negation Based on the Completed Database. In *ICLP/SLP*, pp. 111–125, 1988.

[CWW00] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM (TODS)*, 25(2):179–227, 2000.

[GIT11] T. Green, Z. Ives, and V. Tannen. Reconcilable differences. *Theory of Computing Systems*, 49(2):460–488, 2011.

[GKT07] T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pp. 31–40, 2007.

[GP10] F. Geerts and A. Poggi. On database query languages for k-relations. *Journal of Applied Logic*, 8(2):173–185, 2010.

[KG12] G. Karvounarakis and T. J. Green. Semiring-annotated data: queries and provenance. *ACM SIGMOD Record*, 41(3):5–14, 2012.

[KLZ13] S. Köhler, B. Ludäscher, and D. Zinn. First-Order Provenance Games. In *In Search of Elegance in the Theory and Practice of Computation*, pp. 382–399. Springer, 2013.

[Stu95] P. J. Stuckey. Negation and constraint logic programming. *Information and Computation*, 118(1):12–33, 1995.

[VGRS91] A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649, 1991.

A. Why-Not `3hop(c, a)` Dissected

Consider the input graph in Fig. 1a and its why-not provenance for 3hop(c, a) in Fig. 2. The graph encodes the reasons why 3hop(c, a) is not in the answer. Moving from the lost 3hop(c, a) in Fig. 2, there are nine possible rule instantiations $r_1(\mathtt{c}, \mathtt{a}, z_1, z_2)$, all of which represent a reason why there is no 3hop(c, a) via intermediate nodes $z_1, z_2 \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. To better understand these why-not explanations, consider the input graph in Fig. 7. It contains the original database instance I plus a number of *hypothetical* (or *missing*) edges (dotted), with labels t, u, v, w, and x. These missing edges correspond to the failed leaf nodes in Fig. 2. The table in Fig. 6 contains the why-not provenance, with different combinations of missing edges as preconditions for a derivation of 3hop(c, a).

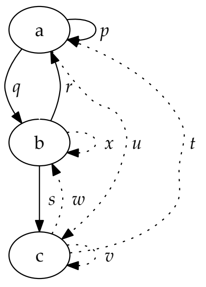


Figure 7: Input graph I with five additional, hypothetical edges (dashed).

B. Constraint Game Construction

Consider the query $Q_{\mathtt{ABC}}$. To build the game, each ground tuple in the program such as B(a, b) is replaced by a constraint B: x_1=a, x_2=b (a conjunction).

First, the subgraph for EDB predicates is created. The remainder of the game is constructed iteratively similar to query execution. For rules whose subgoals are all on EDB predicates, goal/rule nodes/edges are generated. For IDB predicates that were only in the head of EDB-only rules, tuple nodes are generated. Goal and rule nodes/edges are added for rules when the subgraph for all their subgoals has been generated, and for predicates when the subgraph for all the rules deriving into it has been generated.

For each EDB predicate, an expression is generated that is a disjunction of all tuples in the predicate. This expression and its negation are both processed to produce orthogonal DNF expressions (i.e., the conjunction of any two disjuncts in the expression is unsatisfiable). Tuple nodes $t_+ = \mathtt{P} : c$ and $t_- = \mathtt{!P} : c$ and an edge (t_-, t_+) are added to the graph for each disjunct in the constraint.

Those EDB nodes created from a positive expression disjunct are connected negative to positive and positive to a new sink node. Those from a negative disjunct are connected negative to positive, the positive node being a sink.

Orthogonalization is applied to the tuple constraints to ensure that each variable-free tuple is admitted by exactly one node.

Rule nodes are created to which connect IDB tuple nodes for the head predicate and which connect to goal nodes representing the uses of predicates in subgoals of the rule. A rule node is generated for each combination of body tuple nodes such that, if variables in the tuple node constraints were renamed as in the rule, the constraints would be satisfiable when conjuncted. The rule node is given this simplified conjunction as a constraint, each goal node is created with an edge to its originating tuple node, and the rule node is connected to all these goals.

When all rules deriving a predicate have been processed, tuple nodes for the predicate are created. All constraints for rule nodes corresponding to these rules are disjuncted and this expression is restricted to the variables in the rule node.[4] This expression is then treated like that of an EDB predicate: it is simplified and converted to orthogonal DNF. A pair (positive and negative) of tuple nodes is created for each disjunct in the DNF. Edges are created from positive tuple nodes to rule nodes if the tuple node constraint (with variables renamed appropriately) when conjuncted with the rule node constraint can be satisfied.

A player selecting a goal node for goal g with conjunction e argues that a tuple agreeing with e can be used to satisfy g. A player currently 'at' a rule node is fighting the implicit claim that this rule firing is satisfied and creates the tuple in question. To rebut this claim, the player moves to a goal node claimed to be unsatisfied. The goal, if unsatisfied, will be lost; the rule node will be won iff at least one goal is unsatisfied. This provides the desired semantics for the rule node.

A detailed example using the game in Fig. 5 can be found in the next section.

Constraint provenance games improve grounded provenance games by making them domain independent. To return to our motivating example, consider Fig. 5. Observe that the won/lost states are effectively the same as in Fig. 3c, but compressed into constraint nodes that apply to more than one tuple. If one is interested in why the firing $r_2(b, c)$ was not sufficient to derive $A(b)$, then one just has to find the node admitting this rule firing ($r2 : X{\neq}a,\ Y{\neq}a$). The subgraph of this node reachable using provenance edges will explain why rule firings admitted by this node are invalid.

Example Consider the example $Q_{\mathtt{ABC}}$ corresponding to the constraint game in Fig. 5. After all EDB facts of B and C have been processed, the rule is processed. Intuitively, a way to show the presence of $\mathtt{A}(X)$ is to select a node which represent the presence of tuples in B and a node for the absence of tuples in C, which conjunctively correspond to a valid rule firing deriving $\mathtt{A}(X)$. This is equivalent to evaluating the $\exists Y$ from the game template (see Fig. 3a) without having to enumerate all possible assignments of values to Y. Expressions that are not satisfiable in conjunction represent insoluble join conditions between the goals.

When creating nodes for the rule, one could consider the combination $\mathtt{!B} : x_1{=}a, x_2{=}b$ and $\mathtt{C} : x_1{\neq}a$. Goal nodes are created for these ($g_2^1 : \mathtt{B} : X{=}a, Y{=}b$ and $g_2^2 : \mathtt{!C} : Y{\neq}a$, respectively) and since $X{=}a \wedge Y{=}b \wedge Y{\neq}a$ is satisfiable, a rule node $r_2 : X{=}a, Y{=}b$ is created and edges are drawn from the rule node to each goal node and from each goal to the corresponding tuple node. To contrast, the combination $\mathtt{!B} : x_1{=}b, x_2{=}a$ and $\mathtt{C} : x_1{\neq}a$ would not be satisfiable after renaming and conjunction.

Consider the (valid) rule firing $\mathtt{A(a)} :- \mathtt{B(a, b)}, \neg\mathtt{C(b)}$. In constructing the game, the node $\mathtt{!B} : x_1{=}a, x_2{=}\mathtt{b}$ is used for the first goal as this node has the only expression to agree with $\mathtt{B}(a, b)$. A goal node is created signifying the use of this conjunction in the context of this goal: $g_2^1 : \mathtt{B}: X{=}a, Y{=}\mathtt{b}$. Consider the conjunction of the expressions of nodes $g_2^1 : \mathtt{B}: X{=}\mathtt{a}, Y{=}\mathtt{b}$ and $g_2^2 : \mathtt{!C}: Y{\neq}\mathtt{a}$. It can be satisfied, so a rule node is created representing this combination of goal nodes. The corresponding expression is the simplified conjunction of all the goal expressions used.

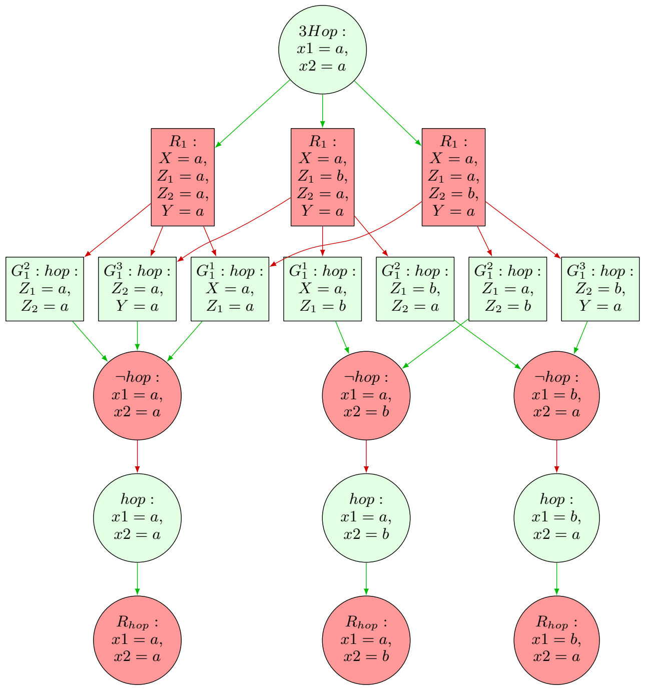
The rule firing $r_2 : X{=}\mathtt{a}, Y{=}\mathtt{b}$ is lost because both the connected goal nodes g_2^1 and g_2^2 are won (ultimately because B(a, b) is in the EDB and C(a) is not, respectively).

An expression for A/1 is generated by disjuncting all the expressions for rule nodes deriving into A/1.[5] This expression is then restricted to X (yielding $X{=}a \vee X{=}b \vee X{\neq}a$). Orthogonalization ensures that each tuple will correspond to a single conjunction: $(X{=}a) \vee (X{=}b) \vee (X{\neq}\mathtt{a}, X{\neq}\mathtt{b})$.

[4] All other variables are replaced with true.

[5] This yields $(Y{\neq}\mathtt{b}, X{=}a, Y{\neq}\mathtt{a}) \vee (X{=}\mathtt{a}, Y{=}\mathtt{a}) \vee (X{=}a, Y{=}b) \vee (X{\neq}\mathtt{a}, Y{\neq}\mathtt{a}) \vee (X{=}b, Y{=}a) \vee (X{\neq}\mathtt{a}, X{\neq}\mathtt{b}, Y{=}\mathtt{a})$

Edges are added between rule head tuple nodes and matching rule nodes where appropriate. The tuple node $A:\ x1{\neq}\mathtt{a},\, x1{\neq}\mathtt{b}$ is connected to $r_2:\ X{\neq}\mathtt{a}, Y{\neq}\mathtt{a}$ because $X{\neq}\mathtt{a}, X{\neq}\mathtt{b}, X{\neq}\mathtt{a}, Y{\neq}\mathtt{a}$ is satisfiable. Selecting the tuple node $\mathtt{A}$: $x_1{=}\mathtt{a}$ claims that there exists a rule firing with no unsatisfied goals that demonstrates the truth of the expression $x_1{=}\mathtt{a}$. In this case, the optimal move would be to $r_2:\ X{=}\mathtt{a}, Y{=}\mathtt{b}$, where it would be the opponent's job to show an unsatisfied goal. In this case, no such goal exists, so the opponent loses the rule node, causing the initial player to win this tuple node.



$3Hop:$ $x1 = a,$ $x2 = a$
$R_1:$ $X = a,$ $Z_1 = a,$ $Z_2 = a,$ $Y = a$
$R_1:$ $X = a,$ $Z_1 = b,$ $Z_2 = a,$ $Y = a$
$R_1:$ $X = a,$ $Z_1 = a,$ $Z_2 = b,$ $Y = a$
$G_1^2 : hop:$ $Z_1 = a,$ $Z_2 = a$
$G_1^3 : hop:$ $Z_2 = a,$ $Y = a$
$G_1^1 : hop:$ $X = a,$ $Z_1 = a$
$G_1^1 : hop:$ $X = a,$ $Z_1 = b$
$G_1^2 : hop:$ $Z_1 = b,$ $Z_2 = a$
$G_1^2 : hop:$ $Z_1 = a,$ $Z_2 = b$
$G_1^3 : hop:$ $Z_2 = b,$ $Y = a$
$\neg hop:$ $x1 = a,$ $x2 = a$
$\neg hop:$ $x1 = a,$ $x2 = b$
$\neg hop:$ $x1 = b,$ $x2 = a$
$hop:$ $x1 = a,$ $x2 = a$
$hop:$ $x1 = a,$ $x2 = b$
$hop:$ $x1 = b,$ $x2 = a$
$R_{hop}:$ $x1 = a,$ $x2 = a$
$R_{hop}:$ $x1 = a,$ $x2 = b$
$R_{hop}:$ $x1 = b,$ $x2 = a$

Figure 8: Constraint game provenance of $3hop(a, a)$. This tuple is generated by three independent successful rule firings, the lost rectangular r_1 nodes. Each of these requires the satisfaction of all goals to be successful, as the position is lost only if all children are won. The player would then proceed to argue that goal is not satisfied (i.e., that the appropriate tuple is not present). In this case, as in general for why provenance over positive queries, constraint provenance is identical with grounded game provenance.

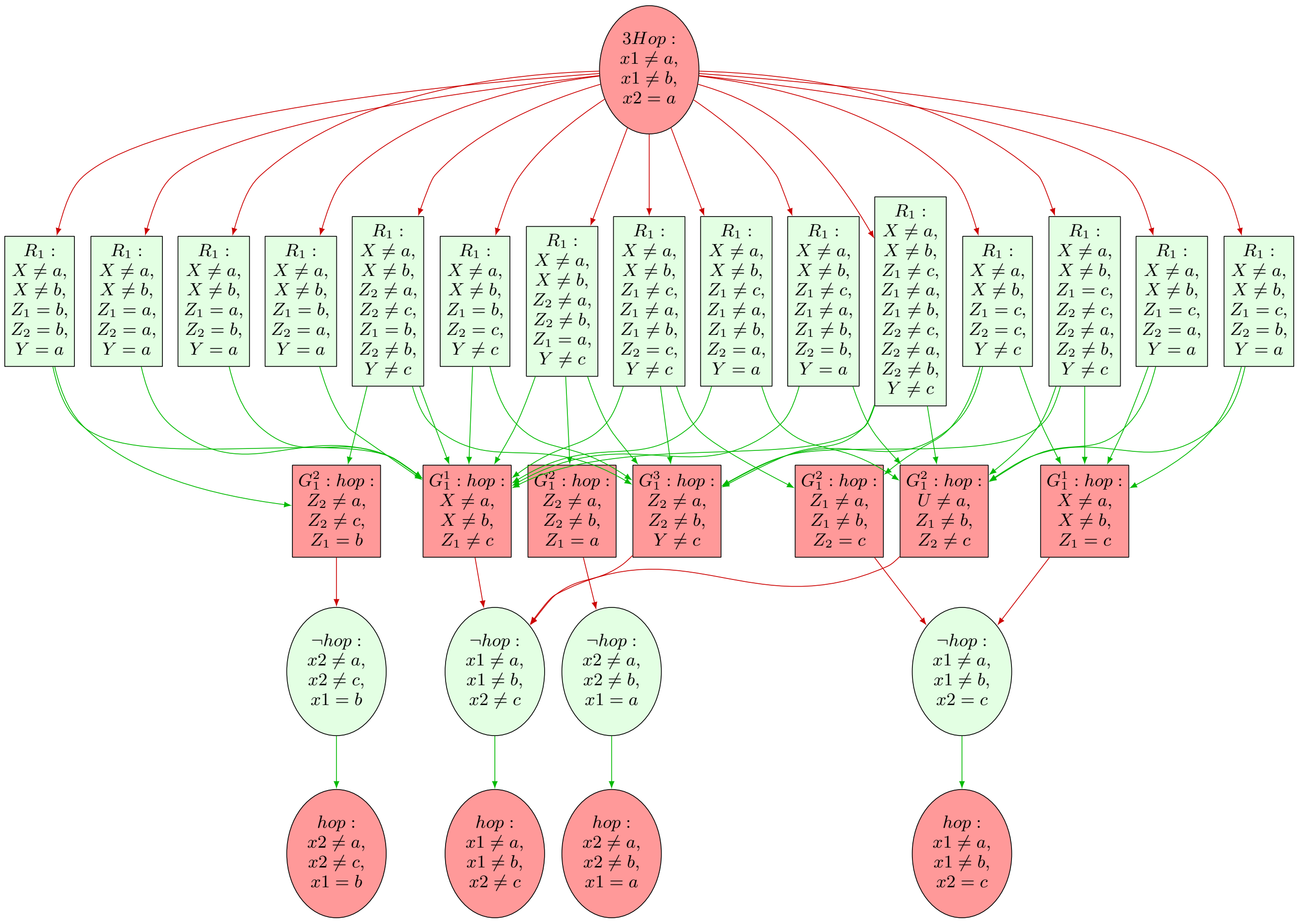

$3Hop: x1 \neq a, x1 \neq b, x2 = a$
$R_1: X \neq a, X \neq b, Z_1 = b, Z_2 = b, Y = a$
$R_1: X \neq a, X \neq b, Z_1 = a, Z_2 = a, Y = a$
$R_1: X \neq a, X \neq b, Z_1 = a, Z_2 = b, Y = a$
$R_1: X \neq a, X \neq b, Z_1 = b, Z_2 = a, Y = a$
$R_1: X \neq a, X \neq b, Z_2 \neq a, Z_2 \neq c, Z_1 = b, Z_2 \neq b, Y \neq c$
$R_1: X \neq a, X \neq b, Z_1 = b, Z_2 = c, Y \neq c$
$R_1: X \neq a, X \neq b, Z_2 \neq a, Z_2 \neq b, Z_1 = a, Y \neq c$
$R_1: X \neq a, X \neq b, Z_1 \neq c, Z_1 \neq a, Z_1 \neq b, Z_2 = c, Y \neq c$
$R_1: X \neq a, X \neq b, Z_1 \neq c, Z_1 \neq a, Z_1 \neq b, Z_2 = a, Y = a$
$R_1: X \neq a, X \neq b, Z_1 \neq c, Z_1 \neq a, Z_1 \neq b, Z_2 = b, Y = a$
$R_1: X \neq a, X \neq b, Z_1 \neq c, Z_1 \neq a, Z_1 \neq b, Z_2 \neq c, Z_2 \neq a, Z_2 \neq b, Y \neq c$
$R_1: X \neq a, X \neq b, Z_1 = c, Z_2 = c, Y \neq c$
$R_1: X \neq a, X \neq b, Z_1 = c, Z_2 \neq c, Z_2 \neq a, Z_2 \neq b, Y \neq c$
$R_1: X \neq a, X \neq b, Z_1 = c, Z_2 = a, Y = a$
$R_1: X \neq a, X \neq b, Z_1 = c, Z_2 = b, Y = a$
$G_1^2: hop: Z_2 \neq a, Z_2 \neq c, Z_1 = b$
$G_1^1: hop: X \neq a, X \neq b, Z_1 \neq c$
$G_1^2: hop: Z_2 \neq a, Z_2 \neq b, Z_1 = a$
$G_1^3: hop: Z_2 \neq a, Z_2 \neq b, Y \neq c$
$G_1^2: hop: Z_1 \neq a, Z_1 \neq b, Z_2 = c$
$G_1^2: hop: U \neq a, Z_1 \neq b, Z_2 \neq c$
$G_1^1: hop: X \neq a, X \neq b, Z_1 = c$
$\neg hop: x2 \neq a, x2 \neq c, x1 = b$
$\neg hop: x1 \neq a, x1 \neq b, x2 \neq c$
$\neg hop: x2 \neq a, x2 \neq b, x1 = a$
$\neg hop: x1 \neq a, x1 \neq b, x2 = c$
$hop: x2 \neq a, x2 \neq c, x1 = b$
$hop: x1 \neq a, x1 \neq b, x2 \neq c$
$hop: x2 \neq a, x2 \neq b, x1 = a$
$hop: x1 \neq a, x1 \neq b, x2 = c$

Figure 9: The why-not provenance of $3hop(c, a)$. The provenance is represented in the failure of the claim that $3hop(c, a)$ is in the answer. This is argued over the Boolean expression defining $3hop(x, y)$. A move from the source node to a child represents the choice of a Boolean expression that is sufficient to capture a rule deriving $3hop(c, a)$. The opponent counters with a subset of this conjunction that is claimed not to be true. The game continues until it reaches the EDB. There exists no equivalent grounded provenance game.