On the Chase for All Provenance Paths With Existential Rules

Abdelraouf Hecham, Pierre Bisquert, Madalina Croitoru

On the Chase for All Provenance Paths With Existential Rules

Abdelraouf Hecham[1], Pierre Bisquert[2], Madalina Croitoru[1]

[1]GraphIK INRIA, University of Montpellier
[2] GraphIK INRIA, French Institute of Research in Agronomy (INRA)

Abstract. In this paper we focus on the problem of *how lineage* for existential rules knowledge bases. Given a knowledge base and an atomic ground query, we want to output all minimal provenance paths of the query (i.e. the sequence of rule applications that generates an atom from a given set of facts). Obtaining all minimal provenance paths of a query using forward chaining can be challenging due to the simplifications done during the rule applications of different chase mechanisms. We build upon the notion of Graph of Atoms Dependency (GAD) and use it to solve the problem of provenance path loss in the context of forward chaining with existential rules. We study the properties of this structure and investigate how different chase mechanisms impact its construction.

1 Introduction

Provenance is used in many information management systems [3, 16, 17] and describes where data came from, how it was derived and how it was updated over time [12]. In this paper we focus on the problem of *how lineage* [12] that, given a knowledge base and a ground query, outputs the provenance paths of the query (i.e. the sequences of rule applications that generate a query from a given set of facts). Given that in a provenance path certain rule applications are unnecessary for provenance justification, it is usually assumed that one is interested in minimal provenance paths. Unlike existing work that focuses on obtaining only one provenance path, the novelty of this work consists in obtaining all provenance paths to a ground query. This problem is relevant in many practical applications such as explanation [9], abduction [13], debugging [4] and notably, defeasible reasoning. In [11], the authors have stumbled upon this problem as query answering in defeasible reasoning with existential rules became unsound due to provenance path loss (not all provenance paths could be extracted). This unexpected behavior as we will show in this paper is due to the *order in which rules are applied* and to the *type of forward chaining mechanism* used.

Forward chaining (a.k.a. *chase*) is the exhaustive application of a set of rules on a set of facts. In this paper we focus on classes of existential rules where forward chaining is finite while backward chaining might be infinite. Different types of chases have been defined in the literature (Oblivious[5], Skolem [14], Restricted [8], etc.), each chase provides a more powerful restriction test for

detecting when to stop. While these tests are crucial for the chase to stop, they might induce a loss of rule applications depending on the order in which the rules are applied. This loss is not picked up by existing work on provenance path extraction which only addressed the problem of obtaining one path, as it was implicitly assumed that obtaining all provenance paths is not a difficult task but a mere enumeration of the first. Unfortunately this is not the case as shown in the following example:

Example 1 *Consider a knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ *and a query* $q = t(b)$ *where the set of facts* $\mathcal{F} = \{p(a), q(b), s(b)\}$*, and the set of rules* $\mathcal{R} = \{R_1 : p(X) \rightarrow r(X, Y)$*,* $R_2 : \{p(X) \wedge s(Y) \rightarrow p(Y)$*,* $R_3 : q(X) \rightarrow r(X, Y)$*,* $R_4 : r(X, Y) \rightarrow t(X)\}$*.*

Extracting the provenance paths for the query q using backward chaining [4] is not possible as it is infinite. To extract provenance paths using forward chaining the state of the art uses a chase graph [6] (also called derivation tree [1]). A chase graph is a directed graph consisting of a set of nodes representing the facts of the chase and having an arrow from a fact u to v iff v is obtained from u (possibly with other atoms) by the application of a rule in $\mathcal{R}$.

The saturated set of facts $\mathcal{F}^* = \mathcal{F} \cup \{r(a, Y_1)$, $p(b)$, $r(b, Y_2)$, $r(b, Y_3)$, $t(a)$, $t(b)\}$ is obtained using an Oblivious chase. This is represented by the chase graph in Figure 1. From the chase graph we can find that there is only one provenance path for $t(b)$ which is applying R_3 on $q(b)$ then R_4 on the resulting $r(b, Y_2)$. However, we can see that by applying $R4$ on the atom $r(b, Y_3)$ we get $t(b)$, which gives us another provenance path that does not show in the chase graph. This loss of provenance path is due to the order in which rules are applied. When the chase applies the rule R_4 on $r(b, Y_3)$ it generates the atom $t(b)$ but this atom is considered redundant as $t(b)$ already exists. This rule application is hence considered not useful and the resulting atoms are not added to the chase graph.

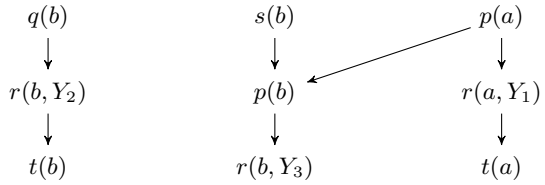

Fig. 1. Chase graph for $\mathcal{F}$ w.r.t. $\mathcal{R}$ of Example 1

In [11], a solution to the provenance path loss problem limited to the restricted chase has been proposed by defining a combinatorial structure called Graph of Atom Dependency (GAD). In this paper we build upon that work and extend the GAD for other types of chase, define its construction algorithms in Section 3, define the provenance path extraction algorithm, study its properties and prove its soundness and completeness in Section 4.

2 Preliminaries

Existential rules extend the Datalog language [7] with existential variables in the conclusion of the rules (also called tuple generating dependencies – TGDs) and generalise certain fragments of Description Logics by allowing n-ary predicates as well as cyclic structures [6]. We consider a first-order logical (FOL) language with constants but no other function symbol based on a vocabulary V composed of an infinite set of predicates, an infinite set of constants, an infinite set of variables and an infinite set of existential 'fresh' variables (called 'nulls', which act as placeholders for unknown constants). Different constants represent different values (unique name assumption) while *different fresh variables may represent the same value*. An atomic formula (or atom) is of the form $p(t_1 \dots t_k)$, where p is a predicate and t_i are variables or constants in V. $\top$ and $\bot$ are also allowed and considered themselves atoms. For a formula Φ, we note $terms(\Phi)$ and $vars(\Phi)$ respectively the terms and variables occurring in Φ. We denote variables by uppercase letters $X, Y, Z, \dots$, constants by lowercase letters $a, b, c, \dots$, nulls with numbered uppercase letter $Y_1, Y_2, \dots$, and predicate symbols by lowercase letters p, q, r, s, etc. We use FOL classical entailment and equivalence, noted $\models$ and $\equiv$ respectively.

A fact F is a ground atom (an atom with only constants and nulls). An existential rule (or a tuple generating dependency) R is a closed formula of the form $\forall \boldsymbol{X}, \boldsymbol{Y} \big(\mathcal{H}(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\ C(\boldsymbol{X}, \boldsymbol{Z})\big)$ where $\boldsymbol{X}, \boldsymbol{Y}$ are tuples of variables, $\boldsymbol{Z}$ is a tuple of existential variables, and $\mathcal{H}, C$ are finite non empty conjunctions of atoms respectively called *premise* and *conclusion* of R. We omit quantifiers when there is no ambiguity, and we use the form $R = (\mathcal{H}, C)$ to represent a rule. The frontier of $R = (\mathcal{H}, C)$ noted $fr(R)$ is the set of variables occurring in both $\mathcal{H}$ and C: $fr(R) = vars(\mathcal{H}) \cap vars(C)$. Given a set of variables $\boldsymbol{X}$ and a set of terms T, a substitution of $\boldsymbol{X}$ by T is a mapping from $\boldsymbol{X}$ to T. Let $\pi : \boldsymbol{X} \rightarrow T$ be a substitution, and Φ be a formula, $\pi(\Phi)$ denotes the set of atoms obtained from Φ by replacing each occurrence of $X \in \boldsymbol{X} \cap terms(\Phi)$ by $\pi(X)$. A homomorphism from a set of atoms S to a set of atoms S' is a substitution of $vars(S)$ by $terms(S')$ such that $\pi(S) \subseteq S'$ (S maps to S' by π).

A rule $R = (\mathcal{H}, C)$ is said to be applicable to a set of facts $\mathcal{F}$ if there is a homomorphism π from $\mathcal{H}$ to $\mathcal{F}$. In that case, the application of R to $\mathcal{F}$ according to π adds to the set $\mathcal{F}$ the conclusion C with constants and possibly new fresh existential variables. More precisely, the application produces a set of facts $\alpha(\mathcal{F}, R, \pi) = \mathcal{F} \cup \pi^{safe}(C)$, where $\pi^{safe}(X) = \pi(X)$ if X belongs to the frontier, and is a fresh variable otherwise. This rule application is said to be redundant if $\alpha(\mathcal{F}, R, \pi) \equiv \mathcal{F}$. The application of R to $\mathcal{F}$, $\alpha(\mathcal{F}, R, \pi)$ w.r.t to π, is also denoted by $R_\pi(\mathcal{F})$. Given a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$ the application of all rules $\mathcal{R}$ on the facts $\mathcal{F}$ is denoted $\mathcal{R}(\mathcal{F})$. Please note that we denote by Π the set of homomorphisms. A knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ is composed of a set of facts $\mathcal{F}$ and a set of rules $\mathcal{R}$. A query q is an *atom* without fresh or free variables. We consider the boolean query answering problem for atomic ground queries that checks whether $\mathcal{KB} \models q$ (i.e. if $\mathcal{R}(\mathcal{F}) \models q$).

The approach in this paper relies on the notion of hypergraphs and hyperpaths. We use the classical definitions of hyperedges and hypergraphs [10, 15]: a *directed hyperedge* $e \in \mathcal{E}$ is an ordered pair $e = (U, W)$ of non empty disjoint subsets of vertices $U, W \in 2^{\mathcal{V}}$; U is the tail of e while W is its head noted $tail(e)$ and $head(e)$ respectively. A *directed edge-labeled hypergraph* is a tuple $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ where $\mathcal{V}$ is a set of vertices (or nodes), $\mathcal{E} \subseteq 2^{\mathcal{V}} \times 2^{\mathcal{V}}$ is a set of directed *hyperedges* (or edges) and $\mathcal{L} : \mathcal{E} \rightarrow L$ is a labeling function that maps each edge $e \in \mathcal{E}$ with an element of the labeling set L.

We define a path $P_{s/t}$ of length k in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ from a node $s \in \mathcal{V}$ to a node $t \in \mathcal{V}$ as a sequence of hyperedges $\langle e_1, \ldots, e_k \rangle$ such that: $s \in tail(e_1)$, $t \in head(e_k)$, and $\forall 1 < i \leq k, head(e_{i-1}) \cap tail(e_i) \neq \emptyset$. We say that two nodes $v_i, v_j \in \mathcal{V}$ are connected if there is a path P_{v_i/v_j} from v_i to v_j. In a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a hyperpath $\Theta_{S/t}$ from $S \subseteq \mathcal{V}$ to $t \in \mathcal{V}$ is a hypergraph $\mathcal{H}_p = (\mathcal{V}_p, \mathcal{E}_p)$ satisfying the following conditions: (1) $\mathcal{E}_p \subseteq \mathcal{E}$, (2) $S \cup \{t\} \subseteq \mathcal{V}_p = \bigcup_{e \in \mathcal{E}_p} (tail(e) \cup head(e))$, and (3) $\forall v \in \mathcal{V}_p$, v is connected to t. A hyperpath $\Theta_{S/t} = (\mathcal{V}_p, \mathcal{E}_p)$ from $S \subseteq \mathcal{V}$ to $t \in \mathcal{V}$ is said to be *minimal* w.r.t. to $\mathcal{V}_p$ and $\mathcal{E}_p$ if no other hyperpath $\Theta'_{S/t} = (\mathcal{V}'_p, \mathcal{E}'_p)$ from S to t exits s.t.: $\mathcal{V}'_p \subset \mathcal{V}_p$ and $\mathcal{E}'_p \subset \mathcal{E}_p$. We denote by $BS(v) = \{e \in \mathcal{E} | v \in head(e)\}$ the backward star (incoming edges) of a node $v \in \mathcal{V}$.

In order to clearly define hypergraphs and how we draw them in this paper, let us consider the following Example 2 that illustrates the notion of hypergraph and hyperedge. In Figure 3 we give the equivalent bipartite depiction of the hypergraph in Figure 2. For clarity reasons we will use the bipartite depiction throughout the paper.

Example 2 *Consider a hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ *with* $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$, $\mathcal{E} = \{\varepsilon_1, \varepsilon_2\}$ *such that* $\varepsilon_1 = (\ \{v_1\}, \{v_3, v_4, v_5\}\ )$ *and* $\varepsilon_2 = (\ \{v_1, v_2\}, \{v_3\}\ )$, *and* $\mathcal{L} = \{\ (\varepsilon_1,\ label_{\varepsilon_1}),\ (\varepsilon_2,\ label_{\varepsilon_2})\ \}$. *In this hypergraph we have* $tail(\varepsilon_2) = \{v_1, v_2\}$ *(please note that* $tail(\varepsilon_2)$ *is depicted in the upper half of the hyperedge* ε_2 *in Figure 3). A path from* v_1 *to* v_4 *is a sequence of hyperedges* $P_{v_1/v_4} = \langle \varepsilon_2 \rangle$. *A hyperpath from* $\{v_1\}$ *to* v_4 *is the hypergraph* $\Theta_{\{v_1\}/v_4} = (\mathcal{V}_\Theta, \mathcal{E}_\Theta)$ *s.t.* $\mathcal{V}_\Theta = \{v_1, v_3, v_4, v_5\}$ *and* $\mathcal{E}_\Theta = \{\varepsilon_1\}$.

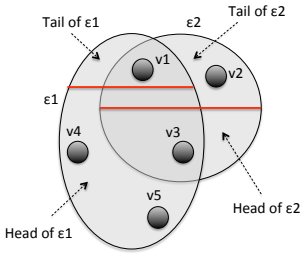

Fig. 2. Hypergraph in Example 2

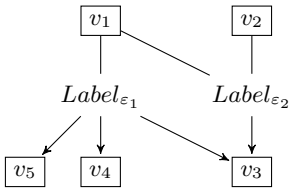

Fig. 3. Bipartite depiction of the hypergraph in Example 2

3 Constructing the Graph Of Atom Dependency

In this section we present the notion of Graph of Atom Dependency (GAD), then we define the algorithms that can be used to construct it, and finally we explain how chase variants can impact its construction. This hypergraph structure will be used in order to construct all provenance paths as detailed in Section 4.

3.1 Provenance Paths

In order to define the notion of provenance path we first need to define the notion of derivation of a set of facts with respect to a set of rules.

Definition 1 (Derivation of $\mathcal{F}$ with respect to $\mathcal{R}$) *Given a set of facts $\mathcal{F}$, and a set of rules $\mathcal{R}$, a **derivation of $\mathcal{F}$ with respect to** $\mathcal{R}$ is a (potentially infinite) sequence $\mathcal{D}$ of D_i s.t. D_i is a tuple $(\mathcal{F}_i, R_i, \pi_i)$ composed of a set of facts $\mathcal{F}_i$, a rule $R_i = (\mathcal{H}_i, C_i)$ and a homomorphism π_i from $\mathcal{H}_i$ to $\mathcal{F}_i$ where: $D_0 = (\mathcal{F}, \emptyset, \emptyset)$, and $\mathcal{F}_i = \alpha(\mathcal{F}_{i-1}, R_i, \pi_i)$.*

In a tuple $D_i = (\mathcal{F}_i, R_i, \pi_i)$ we denote by $fact(D_i) = \mathcal{F}_i$, $rule(D_i) = R_i$ and $homorph(D_i) = \pi_i$ the facts, rule and homomorphism of D_i respectively.

In this paper we are interested in the notion of provenance path of a query. Given a query q and a set of facts $\mathcal{F}$, the provenance of the query q from the facts $\mathcal{F}$ with respect to a set of rules $\mathcal{R}$ is a finite derivation of $\mathcal{F}$ with respect to $\mathcal{R}$ that ends with a set of atoms containing q.

Definition 2 (Provenance path from $\mathcal{F}$ to an atom F w.r.t. $\mathcal{R}$) *A provenance path $\mathcal{PP}$ from the set of facts $\mathcal{F}$ to the atom F with respect to a set of rules $\mathcal{R}$ is a finite derivation of $\mathcal{F}$ with respect to $\mathcal{R}$ s.t.: $\mathcal{PP} = \langle D_0, \ldots, D_n \rangle$ and $F \in fact(D_n)$.*

Example 3 *Let us consider a simple knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ where $\mathcal{F} = \{p(a), r(a)\}$, $\mathcal{R} = \{R_1 : p(X) \wedge r(X) \rightarrow s(X) \wedge t(X),\ R_2 : t(X) \rightarrow q(X),\ R_3 : p(X) \rightarrow u(X)\}$. A possible derivation of $\mathcal{F}$ w.r.t. $\mathcal{R}$ is:*
$\langle (\mathcal{F}, \emptyset, \emptyset)\ ,\ (\mathcal{F}_1 = \mathcal{F} \cup \{s(a), t(a)\}, R_1, \pi_1 = \{X \rightarrow a\}),$
$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{u(a)\}, R_3, \pi_2 = \{X \rightarrow a\}),$
$(\mathcal{F}_3 = \mathcal{F}_2 \cup \{q(a)\}, R_2, \pi_3 = \{X \rightarrow a\})\rangle$.

The provenance path from $\mathcal{F}$ to $q(a)$ is the sequence
$\mathcal{PP}_{\mathcal{KB}} = \langle (\mathcal{F}, \emptyset, \emptyset)\ ,\ (\mathcal{F}_1 = \mathcal{F} \cup \{s(a), t(a)\}, R_1, \pi_1),\ (\mathcal{F}_1 \cup \{q(a)\}, R_2, \pi_3)\rangle$.

3.2 Graph of Atom Dependency (GAD)

A Graph of Atom Dependency (GAD) [11] is a hypergraph where the set of nodes corresponds to the set of atoms and the set of labeled edges corresponds to rule applications labeled by the rule and the corresponding homomorphisms.

Definition 3 (Graph of Atom Dependency) *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$, a Graph of Atom Dependency of $\mathcal{KB}$ is a directed edge-labeled hypergraph that allows repeated edges $\mathcal{H}_{\mathcal{KB}} = (\mathcal{V}_{\mathcal{KB}}, \mathcal{E}_{\mathcal{KB}}, \mathcal{L}_{\mathcal{KB}})$ where:*

- $\mathcal{V}_{\mathcal{KB}}$ *is a set of* ***ground atoms*** *s.t.* $\mathcal{F} \subseteq \mathcal{V}_{\mathcal{KB}}$ *($\mathcal{V}_{\mathcal{KB}}$ contains $\mathcal{F}$ and all generated atoms from $\mathcal{F}$ using $\mathcal{R}$).*
- $\mathcal{E}_{\mathcal{KB}} \subseteq 2^{\mathcal{V}_{\mathcal{KB}}} \times 2^{\mathcal{V}_{\mathcal{KB}}}$ *is a set of hyperedges.*
- $\mathcal{L} : \mathcal{E}_{\mathcal{KB}} \to \mathcal{R} \times \Pi$ *is a labeling function that maps each edge $e \in \mathcal{E}_{\mathcal{KB}}$ to a tuple (R, π) where $R \in \mathcal{R}$ and $\pi \in \Pi$, s.t. $head(e) = \alpha(tail(e), R, \pi)$.*

Example 4 *Let us consider the knowledge base in Example 3, $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ where $\mathcal{F} = \{p(a), r(a)\}$, $\mathcal{R} = \{R_1 : p(X) \wedge r(X) \to s(X) \wedge t(X)$, $R_2 : t(X) \to q(X)$, $R_3 : p(X) \to u(X)\}$. Figure 4 describes the Graph of Atom Dependency of the derivation in Example 3, that is $GAD_{\mathcal{KB}}(\mathcal{F}, \mathcal{R}) = (\mathcal{V}_{\mathcal{KB}}, \mathcal{E}_{\mathcal{KB}}, \mathcal{L}_{\mathcal{KB}})$:*

- $\mathcal{V}_{\mathcal{KB}} = \{p(a), r(a), s(a), t(a), u(a), q(a)\}$
- $\mathcal{E}_{\mathcal{KB}} = \{e_1 = (\ \{p(a), r(a)\}, \{s(a), t(a)\}\ )$, $e_2 = (\ \{p(a)\}, \{u(a)\}\ )$, $e_3 = (\ \{t(a)\}, \{q(a)\}\ )\ \}$
- $\mathcal{L}_{\mathcal{KB}} = \{(\ e_1, (R_1, \pi_1)\ )$, $(\ e_2, (R_3, \pi_2)\ )$, $(\ e_3, (R_2, \pi_3)\ )\}$

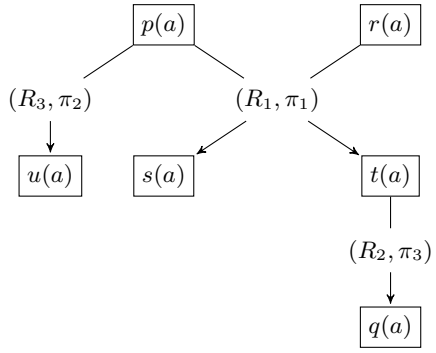

Fig. 4. Graph of Atom Dependency $GAD_{\mathcal{KB}}$ (Example 3)

3.3 Chase Variants for GAD

In this section we will describe how to build the Graph of Atom Dependency using a breadth-first forward chaining algorithm (chase) [5]. We describe the effects of different variants of the chase on the resulting GAD.

Different kinds of chase can be defined by using different derivation reducers. A derivation reducer σ is a function that, given a derivation $\mathcal{D}$ of $\mathcal{F}$ w.r.t. $\mathcal{R}$ returns a sequence of sets of facts such that $\forall D_i \in \mathcal{D}, \sigma(D_i) \subseteq fact(D_i)$. We call $\sigma-chase$ a chase relying on some derivation reducer σ. It generates a possibly infinite derivation σ–$chase(\mathcal{F}, \mathcal{R})$ of $D'_i = (\sigma(D_i), R_i, \pi)$. We say that a (possibly infinite) derivation obtained by a $\sigma-chase$ is complete when any further rule application on that derivation would produce the same set of facts. Since we place ourselves in a context where the chase is finite (for example, concrete Finite Expansion Set classes for Skolem and Restricted chases [2]), then we can extract all provenance paths without loss. This will be detailed below.

The algorithm to construct the $GAD = (\mathcal{V}, \mathcal{E})$ using a chase $\sigma-chase(\mathcal{F}, \mathcal{R})$ is straightforward (as described by Algorithm 1): for each rule application, if it

generates new facts (according to the chase derivation reducer), then a hyperedge between the involved atoms and the generated ones is added. If, on the other hand, the generated facts are not considered new according to the chase derivation reducer (these atoms already exists) then a procedure that handles atoms that are considered the same is called. This procedure is specific to the type of chase as each chase defines same atoms differently. The algorithm is polynomial in the size of the saturated knowledge base. The call to the procedure $HandleSameAtoms$ is what differentiates a Graph of Atom Dependency from a chase graph.

Algorithm 1 GAD construction with chase

Function `ChaseGAD` $(\sigma - chase(\mathcal{F}, \mathcal{R}))$
  input : $\sigma - chase(\mathcal{F}, \mathcal{R})$: the chase
  output: $GAD = (\mathcal{V}, \mathcal{E})$: Graph of Atom dependency w.r.t. $\mathcal{F}$ and $\mathcal{R}$
  $\mathcal{V} \leftarrow \mathcal{F}$; $\mathcal{E} \leftarrow \emptyset$; $GAD \leftarrow (\mathcal{V}, \mathcal{E})$;
  foreach $D_i = (\mathcal{F}_i, R_i = (H_i, C_i), \pi_i) \in \sigma - chase(\mathcal{F}, \mathcal{R})$ **do**
    if $\sigma(D_i) \neq (\mathcal{F}_{i-1})$ **then**
      foreach $v \in \pi_i(C_i)$ *and* $v \notin (\mathcal{F}_{i-1})$ **do**
        Add v to $\mathcal{V}$;
      end
    end
    $HandleSameAtoms(F_{i-1}, D_i, GAD)$;
  end
  return GAD;

In a chase graph, if the atom v has been generated before the atom w and w is considered the same as v then w is removed along with the subtree rooted in w. This is problematic as it removes some provenance paths as detailed in Example 5. In what follows we define the $HandleSameAtoms$ algorithm for each different kind of chase: oblivious, skolem and restricted.

Oblivious Chase The oblivious chase $\sigma_{obl} - chase$ (also called naive chase) [5] relies on the oblivious derivation reducer denoted by σ_{obl} and is defined as follows: for any derivation $\mathcal{D}$, $\sigma_{obl}(D_1) = \mathcal{F}_1$ and $\forall D_i = (\mathcal{F}_i, R_i, \pi_i) \in \mathcal{D}$:

$$\sigma_{obl}(D_i) = \begin{cases} \mathcal{F}_{i-1} \cup \pi_i^{safe}(C_i) & \text{if } \forall j < i, \pi_j \neq \pi_j \text{ or } R_j \neq R_i \\ \mathcal{F}_{i-1} & \text{otherwise} \end{cases}$$

Essentially, the oblivious chase ensures that a rule R is applied according to a homomorphism π only if it has not already been applied according to the same homomorphism. For this chase, two atoms are considered the same if they are exactly the same (i.e. redundant). Due to the simplicity of the test performed by the oblivious chase, the $HandleSameAtoms$ procedure (defined in Algorithm 2) for this chase simply ensures that for any rule application, if an edge representing it has not already been created, then it creates it. This algorithm is polynomial in the size of $\mathcal{F}_{i-1}$.

Algorithm 2 Handle same atoms for Oblivious chase

Procedure `HandleSameAtoms` $(\mathcal{F}_{i-1}, D_i, GAD)$
  input : F_{i-1} : set of facts, $D_i(\mathcal{F}_i, \mathcal{R}_i = (H_i, C_i), \pi_i)$: element of the chase, $GAD = (\mathcal{V}, \mathcal{E})$: graph of atom dependency
  if $e = (\pi_i(H_i), \pi_i(C_i)) \notin \mathcal{E}$ **then**
    if *e does not create a cycle* **then**
      Add e to $\mathcal{E}$;
    end
  end

Example 5 *Let us consider* $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ *from Example 1. A possible derivation for the oblivious chase of* $\mathcal{F}$ *w.r.t.* $\mathcal{R}$ *is:*
$\sigma_{obl}-chase(\mathcal{F}, \mathcal{R}) =< (\mathcal{F}, \emptyset, \emptyset)$,
$(\mathcal{F}_1 = \mathcal{F} \cup \{r(a, Y_1)\}, R_1, \pi_1 = \{X \to a\})$,
$(\mathcal{F}_2 = \mathcal{F}_1 \cup \{p(b)\}, R_2, \pi_2 = \{X \to a, X \to b\})$,
$(\mathcal{F}_3 = \mathcal{F}_2 \cup \{r(b, Y_2)\}, R_3, \pi_3 = \{X \to b\})$,
$(\mathcal{F}_4 = \mathcal{F}_3 \cup \{t(a)\}, R_4, \pi_4 = \{X \to a, Y \to Y_1\})$,
$(\mathcal{F}_5 = \mathcal{F}_4 \cup \{t(b)\}, R_4, \pi_5 = \{X \to b, Y \to Y_2\})$,
$(\mathcal{F}_6 = \mathcal{F}_5 \cup \{r(b, Y_3)\}, R_1, \pi_6\{X \to b\})$,
$(\mathcal{F}_7 = \mathcal{F}_6, R_4, \pi_7 = \{X \to b, Y \to Y_3\}) >$

The chase graph and GAD resulting from the oblivious chase σ_{obl}*–chase*$(\mathcal{F}, \mathcal{R})$ *are shown in Figure 1 and Figure 5 respectively. As described before, the chase graph can only find one minimal provenance path* $\mathcal{PP}_1$ *for* $t(b)$ *whereas the GAD can find another minimal provenance path* $\mathcal{PP}_2$ *that the chase graph lost due to the fact that the application of* R_4 *on* $r(b, Y_3)$ *generates what the oblivious chase considers a redundant atom:*

- $\mathcal{PP}_1 =< (\mathcal{F}, \emptyset, \emptyset)$, $(\mathcal{F}_1 = \mathcal{F} \cup \{r(b, Y_2)\}, R_3, \pi_3)$, $(\mathcal{F}_2 = \mathcal{F}_1 \cup \{t(b)\}, R_4, \pi_5) >$
- $\mathcal{PP}_2 =< (\mathcal{F}, \emptyset, \emptyset)$, $(\mathcal{F}_1 = \mathcal{F} \cup \{p(b)\}, R_2, \pi_2)$, $(\mathcal{F}_2 = \mathcal{F}_1 \cup \{r(b, Y_3)\}, R_3, \pi_6)$, $(\mathcal{F}_3 = \mathcal{F}_2 \cup \{t(b)\}, R_4, \pi_7) >$.

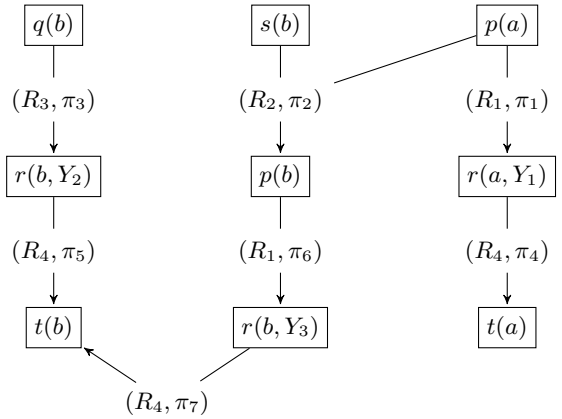

Fig. 5. Graph of atom dependency generated for Example 1

The following proposition states that for any rule application generated by an oblivious chase, there exists an edge representing it in the generated GAD using Algorithm 2, meaning that no rule application is lost.

Proposition 1 (GAD $\sigma_{obl}-chase(\mathcal{F},\mathcal{R})$ Completeness) *Given a knowledge base $\mathcal{KB}=(\mathcal{F},\mathcal{R})$ and $GAD=(\mathcal{V},\mathcal{E})$ generated by an oblivious chase, $\forall D_i=(\mathcal{F}_i,R_i=(\mathcal{H}_i,C_i),\pi_i)\in\sigma_{obl}\text{–}chase(\mathcal{F},\mathcal{R}),\exists e\in\mathcal{E}$ such that $e=(\pi_i(\mathcal{H}_i),\pi_i(C_i))$.*

Proof (Sketch). We prove this by construction, since for any $D_i=(\mathcal{F}_i,R_i=(\mathcal{H}_i,C_i),\pi_i)\in\sigma_{obl}\text{–}chase(\mathcal{F},\mathcal{R})$, $HandleSameAtoms$ is called (as per Algorithm 1), if $e=(\pi_i(H_i),\pi_i(C_i))\notin\mathcal{E}$ then it is added, otherwise, it already exists.

Skolem/Frontier Chase In the frontier chase $\sigma_{fr}-chase$ two applications $\alpha(\mathcal{F},R,\pi)$ and $\alpha(\mathcal{F},R,\pi')$ of the same rule add the same atoms if they map frontier variables identically ($\forall X\in fr(R),\pi(X)=\pi'(X)$). The frontier derivation reducer denoted by σ_{fr} is defined as follows: for any derivation $\mathcal{D}$, $\sigma_{fr}(D_1)=\mathcal{F}_1$ and $\forall D_i=(\mathcal{F}_i,R_i,\pi_i)\in\mathcal{D}$:

$$\sigma_{fr}(D_i)=\begin{cases}\mathcal{F}_{i-1}\cup\pi_i^{safe}(C_i) & \text{if } \forall j<i,\pi_j|_{fr(R_j)}(C_j)\neq\pi_i|_{fr(R_i)}(C_i)\\ & \text{and } R_j\neq R_i\\ \mathcal{F}_{i-1} & \text{otherwise}\end{cases}$$

The frontier chase is equivalent to the skolem chase [14] that relies on a skolemisation of the rules by replacing each occurrence of an existential variable Y with a functional term $f_Y^R(\boldsymbol{X})$, where $\boldsymbol{X}=fr(R)$ are the frontier variables of R; the oblivious chase is then run on skolemized rules. Frontier chase and skolem chase yield isomorphic results [2], in the sense that they generate exactly the same atoms, up to a bijective renaming of variables by skolem terms.

The oblivious chase is strictly 'weaker' than the frontier chase [2] meaning that if $\sigma_{obl}-chase(\mathcal{F},\mathcal{R})$ is finite then $\sigma_{fr}-chase(\mathcal{F},\mathcal{R})$ is also finite. In the frontier chase, two atoms are considered the same if they have the same constants (and possibly different freshly generated constants). The $HandleSameAtoms$ procedure (defined in Algorithm 3) for the frontier chase is more general than the one for the oblivious chase and might result in different GADs. This algorithm is polynomial in the size of the $\mathcal{F}_{i-1}$.

Algorithm 3 Handle same atoms for Frontier chase

Procedure `HandleSameAtoms` $(\mathcal{F}_{i-1},D_i,GAD)$
- **input** : F_{i-1} : set of facts, $D_i(\mathcal{F}_i,\mathcal{R}_i=(H_i,C_i),\pi_i)$: element of the chase, $GAD=(\mathcal{V},\mathcal{E})$: graph of atom dependency
- **if** $\exists j,\pi_j,R_j$ *s.t.* $j<i$ *and* $\pi_j|_{fr(R_j)}(C_j)=\pi_i|_{fr(R_i)}(C_i)$ **then**
  - **if** $e=(\pi_i(H_i),\pi_j(C_i))\notin\mathcal{E}$ *and does not create a cycle* **then**
    - Add e to $\mathcal{E}$;
  - **end**
- **else**
  - **if** $e=(\pi_i(H_i),\pi_i(C_i))\notin\mathcal{E}$ *and does not create a cycle* **then**
    - Add e to $\mathcal{E}$;
  - **end**
- **end**

Similarly to Prop. 1, Proposition 2 states that no rule application is lost, even if it does not generate new atoms, it is still added to the edges of the GAD.

Proposition 2 (GAD $\sigma_{fr}-chase(\mathcal{F},\mathcal{R})$ Completeness) *Given a knowledge base $\mathcal{KB} = (\mathcal{F},\mathcal{R})$ and $GAD = (\mathcal{V},\mathcal{E})$ generated by a frontier chase, $\forall D_i = (\mathcal{F}_i, R_i = (\mathcal{H}_i, C_i), \pi_i) \in \sigma_{fr}-chase(\mathcal{F},\mathcal{R}), \exists e \in \mathcal{E}$ such that $e = (\pi_i(\mathcal{H}_i), \pi_i(C_i))$ or $e = (\pi_i(\mathcal{H}_i), \pi_j(C_i))$ where π_j and π_i map frontier variables of R identically.*

Proposition 3 expresses the structural link between GADs obtained thanks to oblivious and frontier chases.

Proposition 3 *Let $GAD_{\sigma_{obl}} = (\mathcal{V}_{\sigma_{obl}}, \mathcal{E}_{\sigma_{obl}})$ and $GAD_{\sigma_{fr}} = (\mathcal{V}_{\sigma_{fr}}, \mathcal{E}_{\sigma_{fr}})$ be two Graphs of Atom Dependency for $(\mathcal{F},\mathcal{R})$ generated by a complete oblivious chase and a complete frontier chase. If $GAD_{\sigma_{obl}}$ and $GAD_{\sigma_{fr}}$ are finite, then $|\mathcal{V}_{\sigma_{fr}}| \leq |\mathcal{V}_{\sigma_{obl}}|$ and $|\mathcal{E}_{\sigma_{obl}}| = |\mathcal{E}_{\sigma_{fr}}|$.*

Proof (Sketch). Given that frontier is stronger than the oblivious chase [2], some generated atoms are judged redundant by the frontier chase while considered new by the oblivious one, thus $|\mathcal{V}_{\sigma_{fr}}| \leq |\mathcal{V}_{\sigma_{obl}}|$. Furthermore, since rule applications are not lost given Propositions 1 and 2, then $|\mathcal{E}_{\sigma_{obl}}| = |\mathcal{E}_{\sigma_{fr}}|$.

Restricted chase The restricted chase $\sigma_{res}-chase$ (also called standard chase) [8] uses the restricted derivation reducer denoted by σ_{res} and defined as follows: for any derivation $\mathcal{D}$, $\sigma_{res}(D_1) = \mathcal{F}_1$ and $\forall D_i = (\mathcal{F}_i, R_i, \pi_i) \in \mathcal{D}$:

$$\sigma_{res}(D_i) = \begin{cases} \mathcal{F}_{i-1} \cup \pi_i^{safe}(C_i) & \text{if } \mathcal{F}_{i-1} \not\models \pi_i^{safe}(C_i) \\ \mathcal{F}_{i-1} & \text{otherwise} \end{cases}$$

The restricted chase relies on the notion of useful homomorphism. For a rule $R = (\mathcal{H}, C)$ and a set of facts $\mathcal{F}$, a homomorphism π from $\mathcal{H}$ to $\mathcal{F}$ is said to be useful if it cannot be extended to a homomorphism from $\mathcal{H} \cup C$ to $\mathcal{F}$, meaning that $\pi^{safe}(H \cup C)$ does not exist in $\mathcal{F}$. The frontier chase is strictly weaker than the restricted chase, thus, the *HandleSameAtoms* procedure (defined in Algorithm 4) for the restricted chase is more general than the one for the frontier chase and might result in different GADs (as described in Example 6). Furthermore, the restricted chase checks only for local redundancy, meaning that the order in which rules are applied affects the resulting set of atoms as described in Example 6. This algorithm is polynomial in the size of the $\mathcal{F}_{i-1}$.

Similarly to Prop. 1 and 2, no rule application is lost for the restricted chase.

Proposition 4 (GAD $\sigma_{res}-chase(\mathcal{F},\mathcal{R})$ Completeness) *Given a knowledge base $\mathcal{KB} = (\mathcal{F},\mathcal{R})$ and $GAD = (\mathcal{V},\mathcal{E})$ generated by a restricted chase, $\forall D_i = (\mathcal{F}_i, R_i = (\mathcal{H}_i, C_i), \pi_i) \in \sigma_{res}-chase(\mathcal{F},\mathcal{R}), \exists e \in \mathcal{E}$ such that $e = (\pi_i(\mathcal{H}_i), \pi_i(C_i))$ or $e = (\pi'(\mathcal{H}_i), \pi'(C_i))$ where π' is a homomorphism such that $\pi'(\mathcal{H}_i \cup C_i) \subseteq \mathcal{F}_{i-1}$.*

Algorithm 4 Handle same atoms for Restricted chase

Procedure `HandleSameAtoms` $(\mathcal{F}_{i-1}, D_i, GAD)$
  input : F_{i-1} : set of facts, $D_i(\mathcal{F}_i, \mathcal{R}_i = (\mathcal{H}_i, C_i), \pi_i)$: element of the chase, $GAD = (\mathcal{V}, \mathcal{E})$: graph of atom dependency
  if $\exists \pi'$ *s.t.* $\pi'(\mathcal{H}_i \cup C_i) \subseteq \mathcal{F}_{i-1}$ **then**
    if $e = (\pi'(\mathcal{H}_i), \pi'(C_i)) \notin \mathcal{E}$ *and does not create a cycle* **then**
      Add e to $\mathcal{E}$;
    end
  else
    if $e = (\pi_i(\mathcal{H}_i), \pi_i(C_i))$ *does not create a cycle* **then**
      Add e to $\mathcal{E}$;
    end
  end

Example 6 *We will consider the knowledge base* $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ *such that* $\mathcal{F} = \{p(a)\}$ *and the set of rules* $\mathcal{R} = \{R_1 : p(X) \rightarrow r(X,Y) \wedge q(Y),\ R_2 : p(X) \rightarrow r(X,Y),\ R_3 : r(X,Y) \rightarrow q(Y)\}$. *The GAD generated by the frontier chase for this example is exactly the same as the one generated by the oblivious chase regardless of the order in which the rules are applied at each breadth-first derivation. On the other hand, the order of rule applications affects the GAD generated by a restricted chase as shown in Figures 6 and 7.*

Let $GAD_{\sigma_{res}}$ *be the Graph of Atom Dependency generated by first applying the rule* R_1*:* $\alpha(\mathcal{F}, R_1, \pi_1)$ *gives* $\{r(a, Y_1), q(Y_1)\}$*, which are considered new as these atoms are not contained in* $\mathcal{F}$ *(*$\mathcal{F} \not\models \{r(a, Y_1), q(Y_1)\}$*). Hence* $\mathcal{F}_2 = \mathcal{F} \cup \{r(a, Y_1), q(Y_1)\}$.

Then R_2 *is applied:* $\alpha(\mathcal{F}_2, R_2, \pi_2)$ *generates* $\{r(a, Y_2)\}$*, which is considered redundant as the chase maps it to* $\{r(a, Y_1)\}$ *(the fresh variable* Y_2 *is mapped to the fresh variable* Y_1*). We have* $\mathcal{F} \models \{r(a, Y_2)\}$*, so it is not added and the chase continues.*

However, in $GAD'_{\sigma_{res}}$*, the rule* R_1 *is applied after* R_2*. First applying* R_2 *(*$\alpha(\mathcal{F}, R_2, \pi_1)$*) gives* $r(a, Y_1)$ *which is new as* $F \not\models \{r(a, Y_1)\}$*. So* $\mathcal{F}_2 = \mathcal{F} \cup \{r(a, Y_1)\}$.

Then R_1 *is applied (*$\alpha(\mathcal{F}_2, R_1, \pi_2)$*), generating* $\{r(a, Y_2), q(Y_2)\}$ *which are considered new as this set of atoms cannot be mapped to any existing atoms (*Y_2 *cannot be mapped to* Y_1 *as there is no* $q(Y_1)$*). We have* $F_2 \not\models \{r(a, Y_2), q(Y_2)\}$*, so* $\mathcal{F}_3 = \mathcal{F}_2 \cup \{r(a, Y_2), q(Y_2)\}$.

4 Obtaining Provenance Paths

The intuition behind the use of the GAD is that, for a given GAD and a given query, there is a one-to-one mapping, up to provenance path equivalence, between the set of hyperpaths to q and the set of provenance paths to q. Therefore, once the GAD constructed (by considering the different chase mechanisms) the problem of obtaining all provenance paths can be transformed into the problem of generating all hyperpaths of q in the GAD.

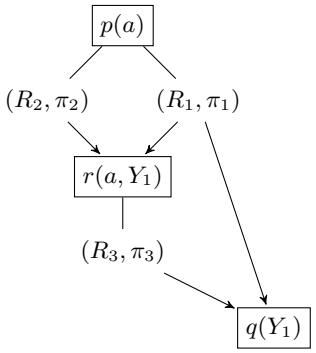

Fig. 6. $GAD_{\sigma_{res}}$ (Example 6) where R_1 is applied before R_2

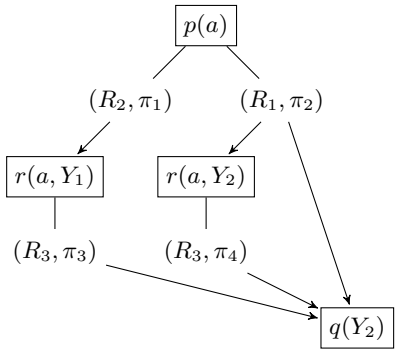

Fig. 7. $GAD'_{\sigma_{res}}$ (Example 6) where R_2 is applied before R_1

Let us first define the notion of provenance path minimality and equivalence. We recall that a provenance path is a sequence $\mathcal{PP}$ of D_i such that D_i is a tuple $(\mathcal{F}_i, R_i, \pi_i)$. We say that two provenance paths $\mathcal{PP}$ and $\mathcal{PP}'$ from a set of facts $\mathcal{F}$ to q are **equivalent** iff they have the same set of atoms and the same set of applied rules (along with their respective homomorphisms) i.e. $\bigcup_{D \in \mathcal{PP}} fact(D) = \bigcup_{D' \in \mathcal{PP}'} fact(D')$ and $\bigcup_{D \in \mathcal{PP}} (rule(D), homorph(D)) = \bigcup_{D' \in \mathcal{PP}'} (rule(D'), homorph(D'))$. We denote that $\mathcal{PP}$ and $\mathcal{PP}'$ are equivalent by $\mathcal{PP} \simeq \mathcal{PP}'$. Please note that $\simeq$ is an equivalence relation (as it is obviously reflexive, symmetric and transitive). Therefore it induces a partition of the set of all provenance paths. A provenance path $\mathcal{PP}$ from a set of facts F to q is said to be **minimal** w.r.t. a set of rules $\mathcal{R}$ and homomorphisms Π if no other provenance path $\mathcal{PP}'$ from F to q exits s.t. $\bigcup_{D' \in \mathcal{PP}'} fact(D') \subset \bigcup_{D \in \mathcal{PP}} fact(D)$ and $\bigcup_{D' \in \mathcal{PP}'} (rule(D'), homorph(D')) \subset \bigcup_{D \in \mathcal{PP}} (rule(D), homorph(D))$. The following property trivially holds.

Proposition 5 *If a provenance path $\mathcal{PP}$ is equivalent to another minimal provenance path $\mathcal{PP}'$ then $\mathcal{PP}$ is minimal.*

Provenance paths are constructed from the hyperpaths of the GAD. The following proposition shows that for every hyperpath of the GAD we can construct an equivalent provenance path. This will ensure the soundness of the hyperpath generation with respect to the problem of generating all provenance paths.

Proposition 6 (Hyperpath Soundness w.r.t. a Provenance Path) *Let GAD be a Graph of Atom Dependency generated by applying a $\sigma - chase(\mathcal{F}, \mathcal{R})$ on a set of facts $\mathcal{F}$ w.r.t. a set of rules $\mathcal{R}$. If there exists a hyperpath $\Theta_{\mathcal{F}/t}$ in GAD from $\mathcal{F}$ to a fact $t \in fact(D)$ s.t. $D \in \sigma - chase(\mathcal{F}, \mathcal{R})$, then there exist a provenance path $\mathcal{PP}$ from $\mathcal{F}$ to t.*

Proof (Sketch). Since GAD is acyclic by definition, then $\Theta_{\mathcal{F}/t}$ is acyclic. If $\Theta_{\mathcal{F}/t}$ is acyclic then a valid ordering of its hyperedges is possible [10]. Based on this valid ordering we can then generate the sequence in the provenance path.

Please note that for a given GAD and for a given hyperpath the provenance paths that can be constructed from $\Theta_{\mathcal{F}/t}$ are equivalent (i.e. they belong to the same class of $\simeq$).

Proposition 7 (Hyperpath Soundness) *Given the GAD for the knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$ and a hyperpath $\Theta_{\mathcal{F}/t}$ in GAD from $\mathcal{F}$ to a fact $t \in fact(D)$ s.t. $D \in \sigma - chase(\mathcal{F}, \mathcal{R})$, if two provenance paths $\mathcal{PP}_1$ and $\mathcal{PP}_2$ are generated from $\Theta_{\mathcal{F}/t}$ then $\mathcal{PP}_1 \simeq \mathcal{PP}_2$.*

Proof (Sketch). Since *GAD* is acyclic by definition, then $\Theta_{\mathcal{F}/t}$ is acyclic. If $\Theta_{\mathcal{F}/t}$ is acyclic then a valid ordering of its hyperedges is possible. In fact, $\Theta_{\mathcal{F}/t}$ can have different valid orderings of its hyperedges. Provenance paths generated from these valid orderings contain the same facts and rule applications since all ordering are for the same hyperedges. Thus, the generated provenance paths are equivalent.

Let us now show that the completeness holds. More precisely we can show that for a given knowledge base and a minimal provenance path there exists an equivalent hyperpath in the GAD associated to the knowledge base.

Proposition 8 (Hyperpath Completeness) *Given a knowledge base $\mathcal{KB} = (\mathcal{F}, \mathcal{R})$, a query q and $\mathcal{PP}$ a minimal provenance path for q in $\mathcal{KB}$, there exists a hyperpath $\Theta_{\mathcal{F}/q}$ in the GAD of $\mathcal{KB}$.*

Proof (Sketch). We prove this by contradiction. Let us suppose that there exists a minimal provenance path $\mathcal{PP}$ for q in $\mathcal{KB}$ such that no hyperpath $\Theta_{\mathcal{F}/q}$ can be constructed in the associated *GAD*. This means that a rule application in the provenance path is not present in the hyperpath. This means that in the construction of the GAD this rule application has not been considered. This is impossible given the results of the completeness of GAD construction using different chase variants (Propositions 1, 2 and 4).

Similar to above, for a given knowledge base and a class of $\simeq$ minimal provenance paths there exists a hyperpath in the associated GAD. The above propositions show the soundness and completeness of minimal provenance path generation with respect to hyperpath finding in a GAD.

Please note that the GAD construction is chase sensitive. For each chase (*oblivious*, *frontier* and *restricted*) a different GAD can be constructed, as shown in the previous section. For a given knowledge base the *oblivious*-generated GAD can be infinite, and for the same knowledge base the *frontier*-generated GAD is finite. The same result shows for *frontier* chase and restricted chase. Therefore if the GAD is finite then, for any of the above chase methods, for a given query we can generate all minimal provenance paths supporting this query. To construct all non-equivalent minimal provenance paths from a set of facts S to a fact t we only need to find all minimal hyperpaths from S to t. For this we need to compute all paths (sequence of hyperedges) from S to t. The recursive function FP defined in Algorithm 5 computes all paths that connect a subset of $\mathcal{F}$ to

an atom t using backward branching; we then use these paths in the procedure FindAllHyperpaths in order to construct the hyperpaths. Please note that Algorithm 5 is based on a modification of [15] to take into account hyperedges rather than hyperarcs. The modification does not affect its complexity which is polynomial in the size of the nodes of the hypergraph.

Algorithm 5 Find Paths & Hyperpaths

Function `FP` (S, t)
- **input :** S : source nodes, t: target node
- **output:** paths: set of all paths between S and t
- paths $\leftarrow \{\}$;
- **if** $t \in S$ **then**
  - **return** paths;
- **end**
- **if** $BS(t)$ *is equal to* $\emptyset$ **then**
  - **return** null;
- **end**
- **foreach** $e \in BS(t)$ **do**
  - path $\leftarrow \{e\}$;
  - tmp $\leftarrow \{\}$;
  - **foreach** $v \in tail(e)$ **do**
    - tmp $\leftarrow$ `FP` $(S, v) \times$ tmp;
  - **end**
  - paths $\leftarrow$ paths $\cup$ (path $\times$ tmp);
- **end**
- **return** paths;

Procedure `FindAllHyperpaths` (S, t)
- **input :** S : source nodes, t: target node
- **output:** hyperpaths: set of all hyperpaths between S and t
- hyperpaths $\leftarrow \{\}$; paths $\leftarrow FP(S, t)$;
- **foreach** *path* $\in$paths **do**
  - $\mathcal{V} \leftarrow S$;
  - $\mathcal{E} \leftarrow path$;
  - **foreach** $e \in \mathcal{E}$ **do**
    - Add $head(e)$ and $tail(e)$ to $\mathcal{V}$;
  - **end**
  - Add $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ to hyperpaths;
- **end**

5 Discussion

In this paper we studied the problem of generating all minimal provenance paths for an atomic ground query in the context of a knowledge base expressed using existential rules. As we have shown, this problem can be tricky as it was implicitly assumed that obtaining all provenance paths can be reduced to obtaining one path. However, given the restriction test of different chase, provenance path loss can occur in certain cases depending on the order in which rules are applied. This provenance path loss can be critical in applications such as defeasible reasoning [11]. To resolve this problem, we extended the notion of a graph of atom

dependency, and showed how the chase choice impacts its construction. We then used this graph to generate all minimal provenance paths for a given atom.

For future work directions we aim to define an optimized algorithm for *conjunctive* atomic queries. The ideas developed in this paper can be used to tackle this issue however performance can optimized when paths intersect. We also plan to consider the core chase, and investigate the use of the GAD in the backward chaining reasoning; more precisely, we plan to study if this could lead to a beneficial combination of backward and forward chaining.

References

1. T. Arora, R. Ramakrishnan, W. G. Roth, P. Seshadri, and D. Srivastava. Explaining program execution in deductive systems. In *Deductive and Object-Oriented Databases*, pages 101–119. Springer, 1993.
2. J.-F. Baget, F. Garreau, M.-L. Mugnier, and S. Rocher. Extending acyclicity notions for existential rules. In *ECAI*, pages 39–44, 2014.
3. P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550. ACM, 2006.
4. R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez. A theoretical framework for the declarative debugging of datalog programs. In *Semantics in Data and Knowledge Bases*, pages 143–159. Springer, 2008.
5. A. Calı, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Proc. of KR*, pages 70–80, 2008.
6. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:57–83, 2012.
7. S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, 1989.
8. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
9. W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. *AI magazine*, 13(3):57, 1992.
10. G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2):177–201, 1993.
11. A. Hecham, M. Croitoru, and P. Bisquert. Argumentation-based defeasible reasoning for existential rules. In *Proceedings of AAMAS '17*, 2017. To appear.
12. R. Ikeda and J. Widom. Data lineage: A survey. 2009.
13. A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. *Handbook of logic in artificial intelligence and logic programming*, 5:235–324, 1998.
14. B. Marnette. Generalized schema-mappings: from termination to tractability. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 13–22. ACM, 2009.
15. S. Nguyen and S. Pallottino. Hyperpaths and shortest hyperpaths. In *Combinatorial Optimization*, pages 258–271. Springer, 1989.
16. C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):797–808, 2008.
17. J. Widom. Trio: A system for data, uncertainty, and lineage. *Managing and Mining Uncertain Data*, 35, 2008.