Obscuring Provenance Confidential Information via Graph Transformation

Jamal Hussein[1,2](✉), Luc Moreau[1], and Vladimiro Sassone[1]

[1] Electronics and Computer Science, University of Southampton, Southampton, UK
[2] Department of Computer Science, University of Sulaimani, Sulaimani, Iraqi Kurdistan
{jah1g12,vs,l.moreau}@ecs.soton.ac.uk

Abstract. Provenance is a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing. In particular, the provenance of information is crucial in deciding whether information is to be trusted. PROV is a recent W3C specification for sharing provenance over the Web. However, provenance records may expose confidential information, such as identity of agents or specific attributes of entities or activities. It is therefore essential for confidential information to be obscured before sharing provenance. This paper describes PROV-GTS, a provenance graph transformation system, whose principled definition is based on PROV properties, and which seeks to avoid false independencies and false dependencies. PROV-GTS is shown to preserve graph integrity, to be terminating and to be confluent.

Keywords: Provenance · PROV model · Privacy · Anonymization · Graph transformation

1 Introduction

Provenance is a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing [1]. Provenance is crucial to validate the quality of data and to enable the reusability of data. It has been used in a variety of domains, including scientific workflow [2], healthcare [3], sensor networks [4], and access control [5]. For example, full provenance of medical decisions enriches medical history captured in health records and enables scientists to find the cause of diseases and care providers to improve health services [3]. Overall the provenance of information is essential to decide whether information is to be trusted [1].

However, provenance may include confidential information, such as agents identities, time information, specific attributes of, and relations between, entities, activities and agents. Such a confidential information must be obscured before exposing provenance, but this presents challenges given the graphical nature of provenance and associated graph inference [6]. Indeed, deleting a node or an edge containing confidential information may affect what can be inferred from a graph. For instance, if $a \rightarrow b$ and $b \rightarrow c$ for some transitive relation, confidential information in b, and subsequent deletion of b, will prevent us from inferring $a \rightarrow c$. Such a problem is being referred to as *false independency* [7] since the transformed graph may lead us to believe that a and c are unrelated (in the sense that one has no influence over the other [1]). Likewise, one needs to ensure that a transformed provenance graph does not enable the inference of nodes or edges that cannot be inferred from the original graph, a problem referred to as *false dependency* [8]. The problems of false dependency and independency have not been considered together by previous work on provenance privacy protection [5,9], but should critically be addressed in order to maintain the usefulness of provenance in establishing trust of data.

The model of provenance we adopt is the recently standardized PROV, aimed at sharing provenance information over the Web [6,10]. The richness of PROV requires a principled approach to defining a graph transformation and formalising its properties. To address the problem of false (in)dependency, we have established that, when a node needs to be deleted, we need to consider not only the edges incident to that node, but also the edges between its adjacent nodes. To do so, graph transformation rules need to be equipped with a variety of graph rewriting capabilities, such as negative application conditions (NAC) [11] and nested constraints [12].

The aim of this paper is to propose PROV-GTS, a provenance graph transformation system that prevents false dependencies by creating nodes and edges according to the semantics of PROV. Concretely, the contributions of the paper are threefold: (*i*) A principled definition of transformation rules that is based on the properties of PROV such as its inference rules. (*ii*) An approach to avoiding false independencies and false dependencies in the transformed graph. (*iii*) The system termination and confluence shows that the rules are parallel-independent (no inconsistency and all critical pairs are safe).

In Sect. 2, the most relevant approaches found in the literature are presented. The intuition of our approach to deleting nodes in PROV model is described in Sect. 3. In Sect. 4 the formal definition of graph transformation used in PROV-GTS is presented followed by the construction of the transformation rules in Sect. 5. The issue of inconsistency is resolved in Sect. 6. Nested graph predicate and the properties of PROV-GTS are presented in Sects. 7 and 8, respectively. Finally, we provide the conclusion and future work in Sect. 9.

2 Previous Work

Provenance graph transformations have been mainly used in two broad domains: provenance access control and scientific workflow run provenance. A provenance access control language has been proposed in [5] based on integrity criteria which reduces the original query entered by a user and deletes the paths that are in the original query but not in the reduced query. In [9], data, module, and structural privacy in scientific workflow have been examined. A module clustering approach has been proposed by creating new composite modules from the old ones preventing the visibility of private information while preserving completeness. However, clustering may require adding new dependencies which are not part of the original graph thus breaching the validity of the provenance information by adding false dependencies.

The issue of false dependencies has been solved in the following research works. However, the proposed approaches, by deleting extra information other than the sensitive nodes, have not been able to avoid false independency. A redaction-based graph grammar [13] for rewriting provenance graph replaces two or more nodes and the edges connecting them with a new node and applies node relabelling as necessary to hide sensitive information. The paper [14] shows how a variety of user requests such as abstracting, anonymizing, or hiding nodes may lead to provenance policy violations such as false dependencies, false independencies, or cyclic graphs. The paper suggests inventing new non-functional nodes when it is necessary and maintaining the essential relationships.

The approach proposed in [7,15] performs abstractions on provenance graphs by replacing a graph chunk by one node while avoiding adding false dependencies to the graph. In [16] an abstraction model has been proposed using node grouping. It replaces a set of sensitive nodes by a single node. The approach avoids cycles and invalid relations otherwise a set of nodes will be extended such that sink nodes in the set are of the same type, entity or activity. Finally the system replaces the set by a new node of the same type as the sink node.

3 Deleting Nodes in the PROV Model

PROV [10] defines a notion of graph, formed of nodes and edges, each equipped with an identifier and optionally decorated by attributes. Figure 1 illustrates the nodes and edges of the core PROV data model; they include three node types - entity, activity, and agent - and seven edge types which are wasDerivedFrom (der), wasGeneratedBy (gen), used (use), wasInformedBy (info), wasAttributedTo (attr), wasAssociatedWith (asso), and actedOnBehalfOf (del). We use the abbreviated labels of these edges throughout this paper to refer to them.

In the proposed system, the sensitive parts of the provenance graph are specified by a set of *restricted* nodes. Confidentiality levels will be used to represent *plain* ☐, *restricted* ▩, and *anonymous* ▨ nodes. The full description of these notations will be provided in Sect. 4.

The goal of our proposed approach is to obscure the confidential information by removing the restricted nodes. If a node cannot be deleted then it will be

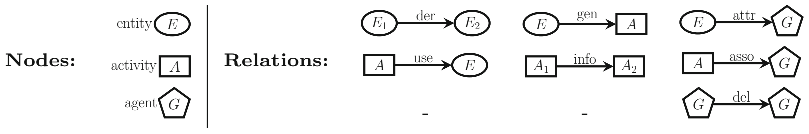


Fig. 1. Core PROV data model

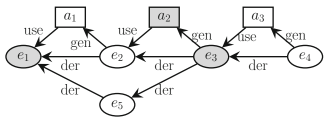
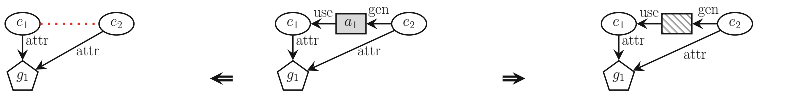


Fig. 2. A provenance graph



(a) Deleting a_1 (b) Original Graph (c) Anonymizing a_1

Fig. 3. Node anonymization

replaced by a less sensitive anonymous node. There are two reasons why we prefer removing nodes over anonymization. First, the topology of the graph can be used by an attacker, who has prior knowledge on the content of the graph, to infer the hidden identity and attributes of the nodes [17]. Second, if we only anonymise, then it is possible to have a graph redundant and useless nodes, which could have been removed.

However, deleting restricted nodes may result in omitting non-relevant information. For example, the implicit *info* edge between activities a_3 and a_2, in Fig. 2, disappears as a consequence of deleting e_3. Since this relation can be inferred, we need to ensure that we create the edge *info* between a_3 and a_2 before deleting e_3.

If the relations cannot be inferred then the node will be anonymized. In Fig. 3(b), no relation between e_1 to e_2 can be inferred; therefore, deleting the restricted activity a_1 will cut the path between e_1 and e_2 as shown in Fig. 3(a). Instead, a_1 will be anonymized as illustrated in Fig. 3(c). Node anonymization is carried out by obscuring the node's *id* and deleting all its attributes [18].

4 PROV Graph Transformation System (PROV-GTS)

Algebraic graph transformation approaches rely either on two gluing diagrams, referred to as double-pushout approach (DPO) [19], or one gluing diagram, referred to as single-pushout approach (SPO). SPO is capable of removing nodes and their incident edges from the graph, including dangling edges [20]. Algebraic approaches can be extended by additional application conditions, such as existence and non-existence of certain nodes and edges [21], as well as conditions that are repeated frequently in the original graph and known as nested conditions [22]. PROV-GTS is an algebraic graph transformation system that consists of a set of rules based on the single-pushout approach.

4.1 PROV Graph

Provenance graphs are typed, which means we need to define a type for each of the nodes, edges, and confidentiality levels. Let ν, ε, and ρ represent the node types, the edge types, and the confidentiality levels, respectively, where ν = [*entity, activity, agent*], ε = [*use, gen, der, info, asso, attr, del*], and ρ = [*plain, restricted, anonymous*]. By default, all graph nodes are *plain*, except those that have been annotated by the user as *restricted.* PROV-GTS either deletes the *restricted* nodes or makes them *anonymous*, according to the intuition of Sect. 3.

To avoid having too many rules and conditions to achieve a particular goal, we define *abstract* nodes and edges based on the core PROV data model [23]. For example, to say that a node has incoming edges, we use an abstract node and edge to construct a single condition, instead of defining multiple conditions for each type of incoming edges. These abstract nodes and edges, as shown in the hierarchies of Fig. 4, have been used to construct PROV-GTS rules, where *node* (the *triangle* in Fig. 4(a)) represents all node types, and *artifact* (the *diamond* in Fig. 4(a)) represents *entity* and *agent.* The top-level edge *link* shown in Fig. 4(c) represents all core PROV edges and *rel* represents each of *der*, *attr*, *use*, *asso*, and *del.* Regarding the confidentiality levels shown in Fig. 4(b), *any* represents the top-level ancestor of *plain*, *anonymous* and *restricted.* Three new sets $\bar{\nu}$, $\bar{\varepsilon}$, $\bar{\rho}$ are defined, where $\bar{\nu} = \nu \cup$ [*node, artifact*], $\bar{\varepsilon} = \varepsilon \cup$ [*link, rel*], and $\bar{\rho} = \rho \cup$ [*any*].

Definition 1 (Extended PROV Graph). *An extended PROV graph is a typed graph G = (N_G, E_G, s_G, d_G, p_G, c_G, h_G) where N_G is the set of nodes, E_G is the set of edges, s_G, d_G: $E_G \rightarrow N_G$ are functions which assign respectively a source and a target node to each edge, the function $p_G : N_G \rightarrow \bar{\rho}$ assigns confidentiality level to the nodes, and the functions $c_G : N_G \rightarrow \bar{\nu}$ and $h_G : E_G \rightarrow \bar{\varepsilon}$ map nodes and edges to their types, respectively.*

Given the type hierarchies of Fig. 4, we extend the definition of graph morphisms [24] with binary relations $\leq_N$, $\leq_E$, and $\leq_P$ which are implicitly defined in Fig. 4 by the subtype arrows. These relations are used in mapping nodes, edges, and confidentiality levels in PROV-GTS rules to their corresponding nodes, edges, and confidentiality levels in PROV graphs, respectively.

Definition 2 (Extended Graph-Morphism). *Let G and H be graphs. A morphism $f : G \rightarrow H$ is the mappings $f_N : N_G \rightarrow N_H$, $f_P : N_G \rightarrow N_H$ and $f_E : E_G \rightarrow E_H$, such that the diagrams below commute.*

A partial graph morphism $f : G \rightarrow H$ is a total graph morphism from some sub-graph K of G to H, where $N_K \subseteq N_G$ and $E_K \subseteq E_G$.

$$\begin{tikzcd} E_G \arrow[r, "f_E"] \arrow[d, "s_G, d_G"'] & E_H \arrow[d, "s_H, d_H"] \\ N_G \arrow[r, "f_N"] & N_H \end{tikzcd}$$

$$f_N \circ s_G = s_H \circ f_E$$
$$f_N \circ d_G = d_H \circ f_E$$

$$\begin{tikzcd} N_G \arrow[rr, "f_N"] \arrow[dr, "c_G"'] & \leq_N & N_H \arrow[dl, "c_H"] \\ & \bar{\nu} & \end{tikzcd}$$

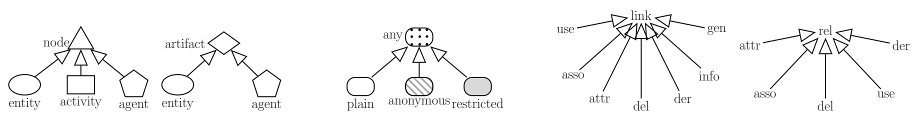
$$c_G \leq_N c_H \circ f_N$$

$$\begin{tikzcd} N_G \arrow[rr, "f_P"] \arrow[dr, "p_G"'] & \leq_P & N_H \arrow[dl, "p_H"] \\ & \bar{\rho} & \end{tikzcd}$$

$$p_G \leq_P p_H \circ f_P$$

$$\begin{tikzcd} E_G \arrow[rr, "f_E"] \arrow[dr, "h_G"'] & \leq_E & E_H \arrow[dl, "h_H"] \\ & \bar{\varepsilon} & \end{tikzcd}$$

$$h_G \leq_E h_H \circ f_E$$



(a) Node Hierarchies (b) Confidentiality Hierarchy (c) Edge Hierarchies

Fig. 4. Type hierarchies for PROV-GTS rules

Definition 3 (Extended Graph Category). *Let $Graph_p$ be the category of extended PROV graphs and extended partial graph-morphisms between them. We use Graph to denote the category of extended PROV graph and their extended (total) morphism.*

4.2 PROV Rules

The graph transformation modifies a graph G according to a rule of the form $r : L \rightarrow R$ by replacing an instance of L in G by R [24].

Definition 4 (PROV Rule). *A PROV graph transformation rule $r : L \rightarrow R$ is a partial morphism r in $Graph_p$; L and R are called the left-hand side (LHS) and the right-hand (RHS) side of the rule, respectively.*

The match of the rule r is given by a total morphism $m : L \rightarrow G$. The rule is applied by using the derivation $G \overset{r,m}{\Rightarrow} H$, which is given by the pushout of r and m using a single-pushout approach [20].

The rule r can be extended to have negative application conditions which forbid the existence of certain graph patterns before applying the rule [21].

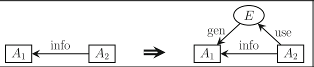
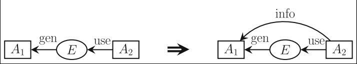
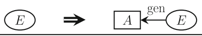
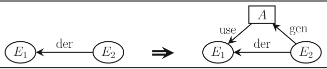
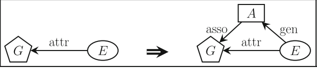
Definition 5 (Negative Application Condition). *A negative application condition (NAC) for the rule r is a total morphism $n : L \rightarrow N$ in Graph; n is satisfied by a graph morphism $m : L \rightarrow G$ if there exists no total morphism $p : N \rightarrow G$ such that $p \circ n = m$.*

Definition 6 (PROV Graph Transformation System). *A PROV graph transformation system* PROV-GTS *consists of a set of graph transformation rules set_R (possibly with NACs).*

5 Construction of PROV-GTS Rules

We use PROV properties prov1–5 (see Table 1) to construct the provenance graph transformation rules in a principled manner. All the inferences related to core PROV [6] are used to define PROV-GTS, except for [6, Inference 11] which is related to the time information of generation and usage relations and is outside the scope of this paper. While formally there is no explicit inference relevant to prov4, the narrative of PROV makes it clear that the existence of an activity can be inferred from the derivation relation. Table 1 provides a graphical representation of PROV properties, which we comment below.

Table 1. PROV model properties

Property	**Graph Patterns** ($C_i \rightarrow E_i$)	**Property**	**Graph Patterns** ($C_i \rightarrow E_i$)
prov1		prov2	 A1 gen E use A2 info A1 gen E use A2
prov3		prov4	
prov5			

– prov1: The existence of an entity generated by an activity and used by another can be implied from their communication (*info*) relation ([6, Inference 5]).
– prov2: The communication between two activities can be inferred if there exists an entity generated by one of the activities and used by the other ([6, Inference 6]).
– prov3: The existence of an entity implies the existence of the activity that generated it ([6, Inference 7]).
– prov4: The derivation (*der*) edge implies the existence of an activity which connects the generated and used entities [1].
– prov5: The attribution relation (*attr*) between an entity and an agent implies that there is an activity that generated the entity and is associated with the agent ([6, Inference 13]).

Definition 7 (PROV Property). *A PROV property* prov$i$ *is* $p : C_i \rightarrow E_i$ *in Graph for* $i = 1..5$ *where Ci and Ei are respectively premise and conclusion of inference rules, and p is the obvious inclusion morphism.*

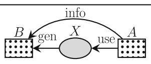
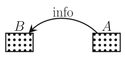
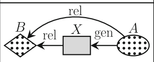
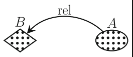
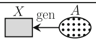
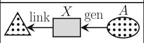
These properties are used for two purposes. First, to define conditions necessary to construct the *deletion rules*. Second, to create the inferred nodes and edges that are not explicitly in the PROV graph and required to trigger the deletion rules by defining a set of *creation rules*. For example, prov1 can be used not only to delete an entity if it is part of a communication relation, but also to infer the existence of the entity if its identity is unknown. Any restricted nodes that are not part of the patterns that are represented by those properties and cannot be inferred from them will be used to construct *anonymization rules*.

The following sections respectively introduce deletion, creation, and anonymization rules, consisting of LHS, RHS, and/or NACs based on the aforementioned properties. The rules are presented progressively, starting with the functionality, and continuing with the more involved versions, in Sect. 6 to deal with inconsistency, and in Sect. 7 with nested conditions.

5.1 Deletion Rules

Based on prov1, a restricted entity can be deleted when the *info* edge between the generating and using activities exists. Additionally, the prov3, 4, 5 are used

Table 2. Deletion rules

rule	L	R	NAC	Rationale
rule 1 ***delete-entity***				prov1
rule 2 ***delete-activity-in-out***				prov4–5
rule 3 ***delete-activity-no-out***				prov3

to form two activity deletion rules. By using the shapes and the colour patterns defined in Fig. 4, Table 2 illustrates the deletion rules and specifies the properties used in their construction in the 'Rationale' column. Note that the (labelled) nodes in L are fixed for the entire rule. The negative application condition for *rule 3* indicates that the rule is applicable if the restricted activity has no outgoing edges.

Definition 8 (Deletion Rule). *A deletion rule is a rule* $r : L \to R$ *in* $Graph_p$ *constructed from PROV property* prov$i$*, where* $L = E_i$*,* $R = C_i$ *and the nodes in* $N_{E_i} \backslash N_{C_i}$ *are restricted.*

5.2 Creation Rules

The *info* relation can be inferred with prov2. In addition, generating and using activity can be inferred by prov4–5. Furthermore, generating activities can be inferred with prov3, if the restricted entity has no outgoing edges. Using prov1, we can add an entity to *info* edges to enable the *delete-activity-no-out* rule. The creation rules are shown in Table 3.

Definition 9 (Creation Rule). *A creation rule is a rule* $r : L \to R$ *in* $Graph_p$ *constructed from PROV property* prov$i$ *where* $L = C_i$*,* $R = E_i$*, and one of the nodes in* N_{C_i} *is restricted.*

5.3 Anonymization Rules

The restricted nodes are not always part of the patterns used to construct the deletion rules, or the patterns that can be completed using the creation rules. If this is the case, the restricted nodes will be anonymized. Examples of the restricted nodes that cannot be removed include an entity with no incoming edges, an activity that generated an entity and used another without derivation edges. Since there are no properties that help eliminate agents, they are always anonymized. The patterns that have been used to construct the anonymization rules are shown in Table 4. The rules *create-entity-in-info* (Table 3) and *anonym-activity-in-info* (Table 4) share the same LHS and NAC but conflict in their RHS which will be addressed in Sect. 6.

Table 3. Creation rules

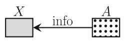
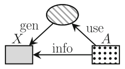
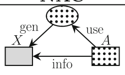
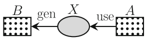
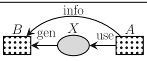
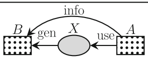
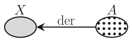
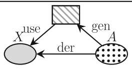
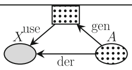
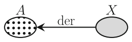
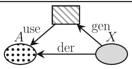
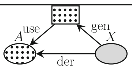
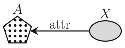
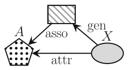
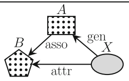
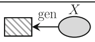
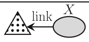
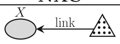
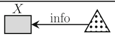
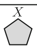
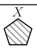
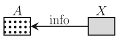
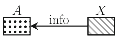
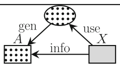
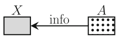
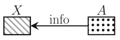
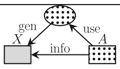
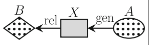
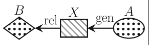
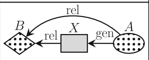
rule	L	R	NAC	Rationale
rule 4 ***create-entity-in-info***				prov1
rule 5 ***create-info-use-gen***				prov2
rule 6 ***create-activity-in-der***				prov4
rule 7 ***create-activity-out-der***				prov4
rule 8 ***create-activity-attr***				prov5
rule 9 ***create-activity-no-gen***				prov3

Table 4. Anonymization patterns

rule	L	R	NAC	Rationale
rule 10 ***anonym-entity-no-in***				entity with no incoming edge
rule 11 ***anonym-activity-no-in***				activity with no incoming edge
rule 12 ***anonym-agent***				agent
rule 13 ***anonym-activity-out-info***				outgoing info with no entity
rule 14 ***anonym-activity-in-info***				incoming info with no entity
rule 15 ***anonym-activity-no-rel***				gen and use with no der gen and asso with no attr

Definition 10 (Anonymization Rule). *The anonymization rules are rules $r : L \rightarrow R$ in $Graph_p$ constructed from a pattern and a NAC as listed in Table 4. They only match if the none of the deletion and creation rules do.*

Observe that the matching condition is beyond the expressive power of classical transformation systems. We will address this in Sect. 7 by formulating our system in terms of the more expressive nested rules.

6 Inconsistency in PROV-GTS

In order to avoid non-determinism in GTS, we must ensure that rules are independent of each other, and that the output of a transformation is not dependent

Table 5. Extra negative conditions to ensure consistency

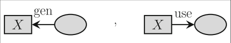
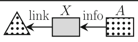
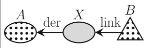
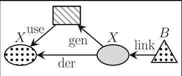
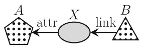
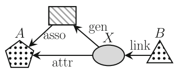
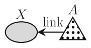
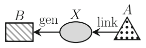
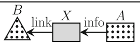
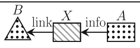
rule	conflicting rules	added NAC
rule 1 *delete-entity*	***rule 2-3*** *delete-activity-**	
rule 14 *anonym-activity-in-info*	***rule 4*** *create-entity-in-info*	

Table 6. Extended LHS-RHS to ensure consistency

rule	conflicting rules	extended L	extended R
rule 10 *anonym-entity-no-in*	***rule 7*** *create-activity-out-der*		
	rule 8 *create-activity-attr*		
	rule 9 *create-activity-no-gen*		
rule 2 *delete-activity-in-out*	***rule 14*** *anonym-activity-in-info*		

on the order in which rules are applied. With the rules as presented so far, these properties do not hold. For instance, restricted activities with incoming *info* edge but no outgoing edges can be anonymized by the rule *anonym-activity-in-info* but it also deleted by the rule *delete-activity-no-out*. In addition, the activities with incoming *info* and outgoing edges must be anonymized by the rule *anonym-activity-in-info*, however, it could be preceded by creating an unnecessary entity by the rule *create-entity-in-info*. The key to ensure the determinism of PROV-GTS is embedding appropriate positive or negative conditions in the transformation rules. Furthermore, deleting activities before entities when matchings of the *delete-entity* rule and the *activity-deletion-** rules overlap, may required more transformation steps. For example, in Fig. 2, deleting the activity a_2 before the entity e_3, requires an extra transformation step by adding an anonymous activity in place of a_2. To resolve this issue we prevent deleting activities until all linked restricted entities are processed by adding two NACs to the *activity-deletion-** rules. Some PROV-GTS rules are provided with extended LHS and RHS (see Table 5), or NACs (see Table 6), which must be added to the conflicting rules, to ensure consistency. There are other overlapped matchings between some of the anonymization rules resulting in critical pairs, but fortunately all these pairs are safe as shown in Sect. 8.3.

7 Nested Graph Predicate

The simple rule consisting of *LHS*, *RHS*, and *NAC*s is not always enough to define transformation rules. For example, in Fig. 2 the entity e_3 has been used in derivation of the two entities e_2 and e_5. The *entity-deletion* rule deletes e_3 based on one of the outgoing edges, ignoring the other relation. To delete these nodes, we have to check conditions that repeat frequently in the host graph and have a universal nature which can only be represented using nested graph predicates [25]. In our system, we adopt the approach defined in [25,26] limited to graph predicates of depth three and one rule application. Each nested rule consists of two parts: the nested graph predicate for rule matching, represented by a root (LHS) and a set of universal-existential pairs (u_i, e_i), and an *RHS* for rule application.

Definition 11 (Nested Graph Predicate). *A nested graph predicate is the tree-shaped diagram in the category Graph consisting of three nested levels:*

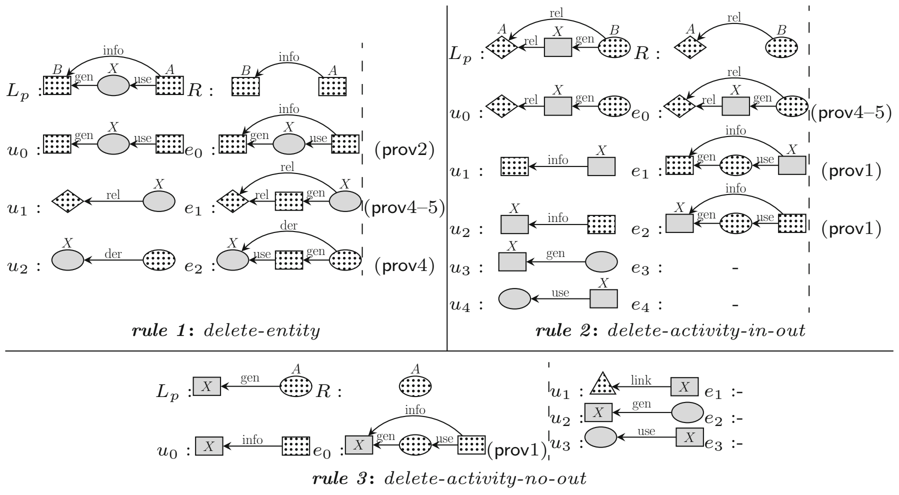
1. ***The Root*** L_p*: each nested predicate has only one root which must be satisfied existentially (that is, in the usual way). The root* L_p *plays the same role as LHS in simple rules.*
2. ***Universal Extensions****: each root* L_p *has at least one universal extension which must be satisfied universally (that is, each possible match is considered, one at the time). It consists of a finite set* $U(L_p)$ *which represents the universal extensions of the root* L_p*, where* $U(L_p) \neq \emptyset$ *and* $u_i \in U(L_p)$ *is ith universal extension.*
3. ***Existential Extensions****: each universal extension may have an associated existential extension, to be satisfied existentially for each match of the universal extension. We denote by* e_i *the existential extension of* u_i*.*

Definition 12 (Nested Predicate Satisfaction). *Let* p *be a nested graph predicate and* G *be a provenance graph,* p *satisfied by the graph* G *if*

- *The predicate* p *existentially satisfied by* $f : L_p \rightarrow G$*, and*
- *For each universal extension* $u_i \in U(L_p)$*,* p *universally satisfied by all* $g : u_i \rightarrow G$ *and* $k : L_p \rightarrow u_i$ *such that* $f = g \circ k$*, and*
- *For each existential extension* e_i*,* p *existentially satisfied by at least one* $h : e_i \rightarrow G$ *and* $l : u_i \rightarrow e_i$ *such that* $g = h \circ l$*.*

If e_i is non-empty, then u_i must be satisfied universally. If e_i is empty, then u_i is de facto a negative application condition NAC, in that no match for it can exist in the graph for p to be satisfied. The nested graph predicates of the deletion rules of PROV-GTS and the properties used in their construction are shown in Fig. 5.

The unlabelled nodes are fixed between u_i and e_i while the labelled nodes are fixed for the entire rule. The provided graph patterns for u_i and e_i represent only the required conditions. The full graphical representation can be obtained by $L \cup u_i$ (and $L \cup e_i$) such that N_L and $N_{u_i} \backslash [X]$ (also N_L and $N_{e_i} \backslash [X]$) are two disjoint sets where $X \in L$ is a restricted node. The completeness of these graph predicates is proven in Sect. 8.1 via Lemma 1.



L_p : B gen X use A info R : B info A
u_0 : gen X use e_0 : gen X use info (prov2)
u_1 : rel X e_1 : rel gen X rel (prov4–5)
u_2 : X der e_2 : X use gen der (prov4)
rule 1: delete-entity
L_p : A rel X gen B rel R : A rel B
u_0 : rel X gen e_0 : rel X gen rel (prov4–5)
u_1 : info X e_1 : gen use X info (prov1)
u_2 : X info e_2 : X gen use info (prov1)
u_3 : X gen e_3 : -
u_4 : use X e_4 : -
rule 2: delete-activity-in-out
L_p : X gen A R : A
u_0 : X info e_0 : X gen use info (prov1)
u_1 : link X e_1 :-
u_2 : X gen e_2 :-
u_3 : use X e_3 :-
rule 3: delete-activity-no-out

Fig. 5. Graph predicates for deletion rules

8 Properties of PROV-GTS

In PROV-GTS, provenance graphs are transformed by applying the rules again and again until no rule applies any more. In the following sections, the proofs of graph integrity, termination, and confluence are provided.

8.1 Graph Integrity

To trust the provenance information, it is important to show that the transformed graph is semantically correct. This can be done by proving that the rules do not create false-dependencies and do not result in false-independencies.

Theorem 1 (No False Dependency). *Suppose G is the original graph and G_T is the graph transformed by the PROV-GTS rules. Then $N_{G_T} \backslash N_G$ and $E_{G_T} \backslash E_G$ can be inferred from the graph G, i.e. there are no false dependencies.*

Proof (No False Dependency). Because of the way the creation rules are constructed (see Definition 9), the transformation rules add to the transformed graph G_T only what can be inferred from the original graph G. In addition, the modified rules in Table 5 do not affect this property, as they add the same positive condition to each of LHS and RHS of the conflicting rules, i.e. always $N_R \backslash N_L = N_{E_i} \backslash N_{C_i}$ and $E_R \backslash E_L = E_{E_i} \backslash E_{C_i}$ for all the creation rules $(r : L \to R)$ constructed from the PROV properties $(p : C_i \to E_i)$.

Lemma 1 (Completeness of Nested Graph Predicates). *Suppose G is the original graph and G_T is the graph transformed by the PROV-GTS rules. Deleting the restricted nodes in G_T, by the deletion rules shown in Fig. 5, will not affect what can be inferred from G.*

Proof (Completeness of Nested Graph Predicates). For the graph predicate to be complete, it must check that all possible relations between the nodes adjacent to the restricted nodes have been preserved. To this end, the nested graph predicates consist of universal-existential pairs that represent all PROV properties relevant to the type of the restricted node. For instance, in the *delete-entity* rule shown in Fig. 5, the PROV properties prov2, prov4–5, and prov4 have been used to construct the universal-existential pairs (u_0, e_0), (u_1, e_1), and (u_2, e_2), respectively. Therefore, the edges that are not incident to the restricted nodes, including the inferred ones, will not be affected by the node deletion.

Theorem 2 (No False Independency). *Suppose G is the original graph and G_T is the graph transformed by the PROV-GTS rules, $N_H = N_{G_T} \backslash N_G$ and $E_H = E_{G_T} \backslash E_G$. No false independency will be created as a consequence of deleting the restricted nodes.*

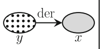
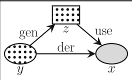
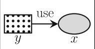
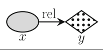
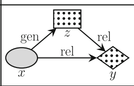
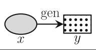
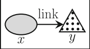
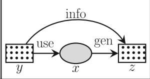
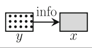
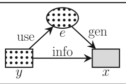
Proof (No False Independency). To prove that there is no false independency, it is sufficient to prove that all nodes in N_H are restricted nodes and the source or target of each edge in E_H is a restricted node in N_H.

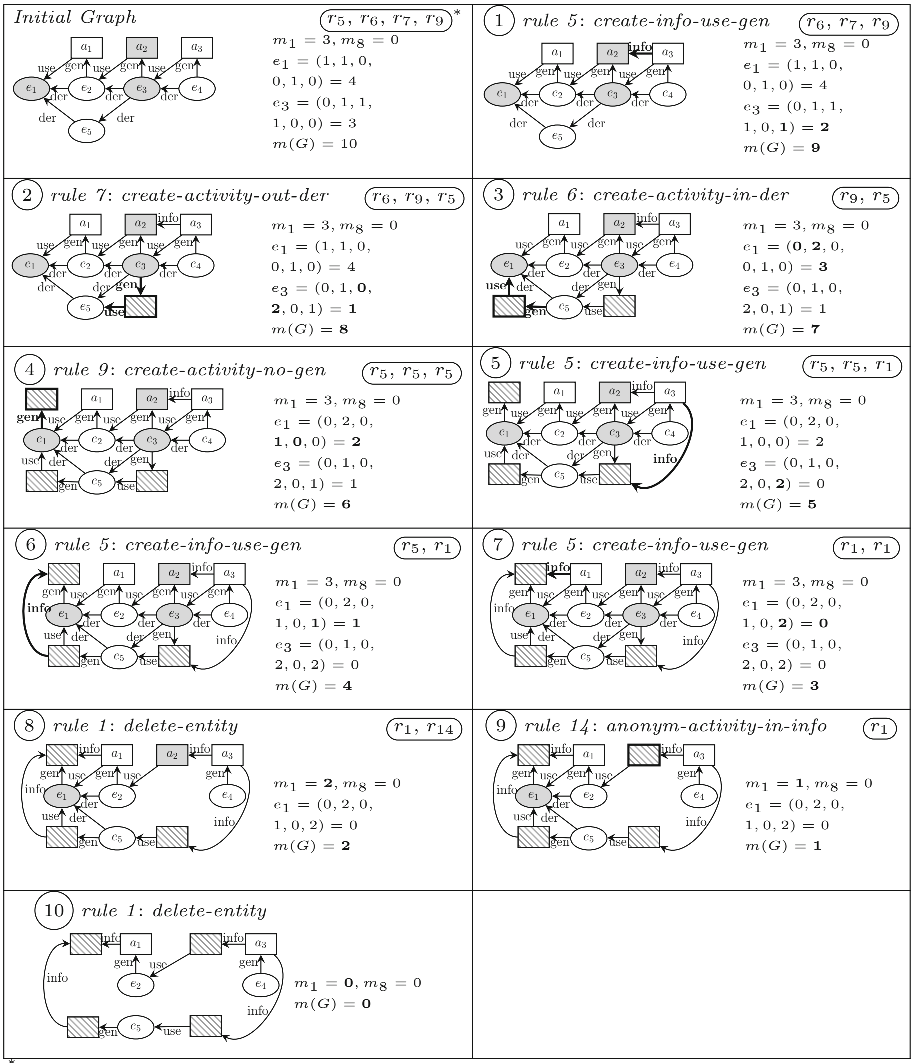
All nodes in N_H are restricted because the deletion rules of PROV-GTS, according to Definition 8, removes only restricted nodes. In addition, the proof of completeness of the nested graph predicates in Lemma 1 shows that only the edges incident to the restricted nodes will be deleted.

8.2 Termination

To ensure that the graph transformation in PROG-GTS always terminates, we use a termination count which is computed by counting the number of occurrences of the graph patterns shown in Table 7. Each pattern has a positive part R and may have a negative part N. The pattern P_1 is used to compute the number of *restricted* nodes, whereas patterns $P_2 \dots P_7$ are used to count the number of creation rule applications for each *restricted* entity. Suppose $m(G)$ is the termination count for graph G and m_1 and m_8 are the number of occurrences of P_1 and P_8 respectively, while $m_j^{e_i}$ is the number of occurrences of P_j for the

Table 7. Termination measurement patterns

P	R	N	P	R	N	P	R	N
P_1		-	P_2			P_3		-
P_4			P_5		-	P_6		
P_7		-	P_8					


Initial Graph
r5, r6, r7, r9 *
m1 = 3, m8 = 0
e1 = (1, 1, 0, 0, 1, 0) = 4
e3 = (0, 1, 1, 1, 0, 0) = 3
m(G) = 10
1 rule 5: create-info-use-gen
r6, r7, r9
m1 = 3, m8 = 0
e1 = (1, 1, 0, 0, 1, 0) = 4
e3 = (0, 1, 1, 1, 0, 1) = 2
m(G) = 9
2 rule 7: create-activity-out-der
r6, r9, r5
m1 = 3, m8 = 0
e1 = (1, 1, 0, 0, 1, 0) = 4
e3 = (0, 1, 0, 2, 0, 1) = 1
m(G) = 8
3 rule 6: create-activity-in-der
r9, r5
m1 = 3, m8 = 0
e1 = (0, 2, 0, 0, 1, 0) = 3
e3 = (0, 1, 0, 2, 0, 1) = 1
m(G) = 7
4 rule 9: create-activity-no-gen
r5, r5, r5
m1 = 3, m8 = 0
e1 = (0, 2, 0, 1, 0, 0) = 2
e3 = (0, 1, 0, 2, 0, 1) = 1
m(G) = 6
5 rule 5: create-info-use-gen
r5, r5, r1
m1 = 3, m8 = 0
e1 = (0, 2, 0, 1, 0, 0) = 2
e3 = (0, 1, 0, 2, 0, 2) = 0
m(G) = 5
6 rule 5: create-info-use-gen
r5, r1
m1 = 3, m8 = 0
e1 = (0, 2, 0, 1, 0, 1) = 1
e3 = (0, 1, 0, 2, 0, 2) = 0
m(G) = 4
7 rule 5: create-info-use-gen
r1, r1
m1 = 3, m8 = 0
e1 = (0, 2, 0, 1, 0, 2) = 0
e3 = (0, 1, 0, 2, 0, 2) = 0
m(G) = 3
8 rule 1: delete-entity
r1, r14
m1 = 2, m8 = 0
e1 = (0, 2, 0, 1, 0, 2) = 0
m(G) = 2
9 rule 14: anonym-activity-in-info
r1
m1 = 1, m8 = 0
e1 = (0, 2, 0, 1, 0, 2) = 0
m(G) = 1
10 rule 1: delete-entity
m1 = 0, m8 = 0
m(G) = 0
a1 a2 a3 e1 e2 e3 e4 e5
use gen der info

*these rounded rectangles are used to show the (other) applicable rules where r_i represents *rule i*.

Fig. 6. Step-by-step rule application with the termination count

restricted entity e_i where $j = 2..7$. The total number of pattern matchings can then be computed using the following equations:

$$m(G) = m_1 + \sum_{i=1}^{n} E_i + m_8 \text{ where } n \text{ is the number of restricted entities,}$$
$$E_i = m_2^{e_i} + m_4^{e_i} + m_6^{e_i} + f_i - m_7^{e_i} \text{ and}$$
$$f_i = (m_2^{e_i} + m_3^{e_i}) * (m_4^{e_i} + m_5^{e_i} + m_6^{e_i})$$

The example in Fig. 6 shows step-by-step PROV-GTS rules application with the termination count at each step. For conciseness, we use a vector notation $(m_2, \ldots, m_7)$ for e_i. In each step, one of the applicable rules are (randomly) chosen and applied on the graph.

8.3 Confluence

Confirming the termination of the graph transformation process is not enough. It is essential to guarantee that the graph rewriting always terminates with the same resulting graph despite the order in which the rules have been applied, i.e. confluent.

Definition 13 (Confluence). *A graph transformation system is called confluent if for all derivations* $G \overset{*}{\Rightarrow} H_1$ *and* $G \overset{*}{\Rightarrow} H_2$ *there is a graph* X *and the derivations* $H_1 \overset{*}{\Rightarrow} X$ *and* $H_2 \overset{*}{\Rightarrow} X$*.*

To prove the local confluence, it is enough to prove that all critical pairs in PROV-GTS are strictly confluent [24].

In PROV-GTS, any of the rules *anonym-activity-out-info*, *anonym-activity-in-info*, and *anonym-activity-no-rel*, when applies on the same activity, makes the other two inapplicable. The same conflict happens when the matchings of the rules *anonym-activity-no-in* and *anonym-activity-out-info* overlap. Since the above rules are anonimyzing activities, the resulting graph is the same for all rule applications. This proves that the system is confluent.

9 Conclusion

In this paper, PROV model properties have been used to construct a set of rewriting rules, which form the PROV graph transformation system (PROV-GTS). The relations are preserved by creating nodes and edges that lead to the deletion of restricted nodes. If this preservation is not possible or does not lead to node deletion, the restricted nodes will be anonymized. The integrity of provenance graph has been proven by showing that no false dependencies or independencies are generated by PROV-GTS, thereby the transformed graph can be trusted. The termination has been proven by defining a count that indicates the progress of graph transformation. We show that the system is confluent using termination proof and confluence of critical pairs.

The proposed system should be expanded to cover concepts that have not been included in this paper. First, new transformation rules must be defined to process the extended terms of the PROV model. In addition, it is important to preserve other concepts, such as the time sequence in which different operations in the provenance graph occur. Finally, the system must be integrated into provenance-based applications and then its functionality, in terms of obscurity and graph utility, must be evaluated by defining proper measurements.

References

1. Moreau, L., Missier, P.: PROV-DM: the PROV data model. Technical report, W3C Recommendation, W3C (2013). http://www.w3.org/TR/prov-dm/
2. Davidson, S.B., Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1345–1350. ACM (2008)
3. Kifor, T., Varga, L., Vazquez-Salceda, J., Alvarez, S., Willmott, S., Miles, S., Moreau, L.: Provenance in agent-mediated healthcare systems. IEEE Intell. Syst. **21**(6), 38–46 (2006)
4. Lim, H.S., Moon, Y.S., Bertino, E.: Provenance-based trustworthiness assessment in sensor networks. In: Proceedings of the Seventh International Workshop on Data Management for Sensor Networks, pp. 2–7. ACM (2010)
5. Cadenhead, T., Khadilkar, V., Kantarcioglu, M., Thuraisingham, B.: A language for provenance access control. In: Proceedings of the First ACM Conference on Data and Application Security and Privacy, pp. 133–144. ACM (2011)
6. Cheney, J., Missier, P., Moreau, L., DeNies, T.: Constraints of the PROV data model. Technical report, W3C Recommendation, W3C (2013). http://www.w3.org/TR/prov-constraints/
7. Missier, P.: Preserving Privacy in Shared Provenance Data. Computing Science. Newcastle University, Newcastle upon Tyne (2013)
8. Dey, S.C., Zinn, D., Ludäscher, B.: ProPub: towards a declarative approach for publishing customized, policy-aware provenance. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 225–243. Springer, Berlin Heidelberg (2011)
9. Davidson, S.B., Khanna, S., Roy, S., Stoyanovich, J., Tannen, V., Chen, Y.: On provenance and privacy. In: Proceedings of the 14th International Conference on Database Theory, pp. 3–10. ACM (2011)
10. Moreau, L., Missier, P., Cheney, J., Soiland-Reyes, S.: PROV-N: the provenance notation. Technical report, W3C Recommendation, W3C (2013). http://www.w3.org/TR/prov-n/
11. König, B., Stückrath, J.: Well-structured graph transformation systems with negative application conditions. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 81–95. Springer, Heidelberg (2012)
12. Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas, F.: M-adhesive transformation systems with nested application conditions. Part 1: Parallelism, concurrency and amalgamation. Math. Struct. Comput. Sci. 93–102 (2012)
13. Cadenhead, T., Khadilkar, V., Kantarcioglu, M., Thuraisingham, B.: Transforming provenance using redaction. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, pp. 93–102. ACM (2011)
14. Dey, S., Ludascher, B.: A declarative approach to customize workflow provenance. In: Proceedings of the Joint EDBT/ICDT 2013 Workshops, EDBT 2013, pp. 9–16. ACM, New York (2013)
15. Missier, P., Bryans, J., Gamble, C., Curcin, V., Danger, R.: PRovenance Graph Abstraction by Node Grouping. Computing Science. Newcastle University, Newcastle upon Tyne (2013)
16. Missier, P., Bryans, J., Gamble, C., Curcin, V., Danger, R.: ProvAbs: model, policy, and tooling for abstracting PROV graphs. In: Ludaescher, B., Plale, B. (eds.) IPAW 2014. LNCS, vol. 8628, pp. 3–15. Springer, Heidelberg (2015)

17. Blaustein, B., Chapman, A., Seligman, L., Allen, M.D., Rosenthal, A.: Surrogate parenthood: protected and informative graphs. Proc. VLDB Endowment **4**(8), 518–525 (2011)
18. Backstrom, L., Dwork, C., Kleinberg, J.: Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In: Proceedings of the 16th International Conference on World Wide Web, pp. 181–190. ACM (2007)
19. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation-part i: basic concepts and double pushout approach. In: Handbook of Graph Grammars, pp. 163–246 (1997)
20. Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A., Corradini, A.: Algebraic approaches to graph transformation-part ii: single pushout approach and comparison with double pushout approach. In: Handbook of Graph Grammars, pp. 247–312. Citeseer (1997)
21. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26**(3), 287–313 (1996)
22. Habel, A., Pennemann, K.-H.: Nested constraints and application conditions for high-level structures. In: Kreowski, H.-J., Montanari, U., Orejas, F., Rozenberg, G., Taentzer, G. (eds.) Formal Methods. LNCS, vol. 3393, pp. 293–308. Springer, Heidelberg (2005)
23. Moreau, L., Groth, P.: Provenance: an introduction to prov. Synth. Lect. Semant. Web Theor. Technol. **3**(4), 1–129 (2013)
24. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation, vol. 373. Springer, Heidelberg (2006)
25. Rensink, A., Kuperus, J.H.: Repotting the geraniums: on nested graph transformation rules. Electron. Commun. EASST **18**, 1–15 (2009)
26. Rensink, A.: Nested quantification in graph transformation rules. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 1–13. Springer, Heidelberg (2006)