A Template-Based Graph Transformation System for the PROV Data Model*

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

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

Abstract. As data provenance becomes a significant metadata in validating the origin of information and asserting its quality, it is crucial to hide the sensitive information of provenance data to enable trustworthiness prior to sharing provenance in open environments such as the Web. In this paper, a graph rewriting system is constructed from the PROV data model to hide restricted provenance information while preserving the integrity and connectivity of the provenance graph. The system is formally established as a template-based framework and formalised using category theory concepts, such as functors, diagrams, and natural transformation.

Keywords: PROV data model, Graph transformation, Category theory, Graph template, Graph integrity.

1 Introduction

Provenance is defined as "a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing" [1]. Data provenance is vital in validating the origin of information and asserting its quality. Provenance has been adopted in many significant domains for different purposes, for example validating experimental results in scientific workflow systems [2,3] improving health services in healthcare [4], trustworthiness of data from sensor networks [5], and managing access control systems [6].

Sharing provenance may expose confidential information such as the medical history of a patient, the identity of an agent, and the bank account details of an individual. Prior to sharing provenance information in open environments such as the Web, we need an effective approach to deal with such private information. Regarding the graphical nature of provenance [7], certain parts of provenance graphs can be removed using algebraic graph transformation rules [8].

However, removing confidential parts from provenance graphs or creating new graph elements may affect the integrity of provenance graphs by causing

* This research was funded in part by the UK Research Council EPSRC for project 'Orchid', grant EP/I011587/1.

false independency and false dependency [9]. False independency happens when deleting items from the graph prevents us from inferring edges and nodes in the transformed graph that can be inferred from the original graph. While false dependency occurs if the items created by the graph transformation rules cannot be justified from the original graph. Preventing both false dependencies and false independencies have been guaranteed in the PROV graph transformation system PROV-GTS proposed in [10] based on the PROV data model. However, due to the complex nature and simultaneous edges of the PROV model graphs [11], the construction of PROV-GTS results in a large set of graph transformation rules, which makes validating the integrity of the provenance graph and proving properties such as termination and parallelism difficult.

In this paper, the new notion of rule templates is used to construct a template-based PROV-GTS by grouping the rules that consist of similar graph patterns in templates. This template-based construction enhances the readability of the rules and facilitates the proofs of various properties. The contributions of this paper are (i) A graph rewriting system constructed from the PROV data model to hide restricted provenance information while preserving the graph integrity. (ii) A template-based framework formalised using category theory concepts such as functors, diagrams, slice categories, and natural transformation. we adopt the same construction methodology provided in [10], but by defining simpler edge deletion rules rather than the complicated node deletion rules. To avoid false independency, creation rules are defined to preserve relations between the nodes adjacent to the source and target of the deleted edges.

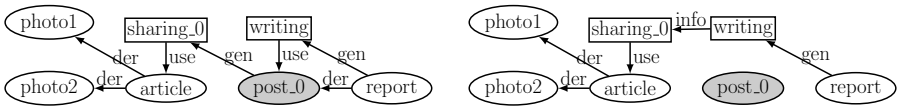
A running example is provided in Sect. 2. The related work is discussed in Sect. 3. The PROV data model is presented in Sect. 4. The typed provenance graph with inheritance is formalized in Sect. 5. The provenance graph transformation rules are described in Sect. 6. The transformation rules are grouped in templates in Sect. 7. Sect. 8 provides the proofs of graph integrity. Finally, the conclusion and future work is presented in Sect. 9.

2 A Running Example

The provenance graph representation, based on the W3C standardized PROV data model, consists of nodes and edges labelled with identifiers and characterized by optional attributes [12]. The PROV model graphs consists of the nodes: entity ⬭, activity ▭, and agent ⬠ and a set of edges connecting those nodes.

A concrete provenance graph is used as a running example based on the following scenario: A famous restaurant was targeted by its rivals via publishing a fake photo of one of the restaurant's products. The photo went viral via a post on a social network website and caused an outrage which badly affected the restaurant's reputation. The restaurant managers immediately started an investigation to track down the origin of that photo and prepare a report. They found that the photo had been fabricated using two different photos and it first appeared online as part of an article published on a website which mainly used by rumour-mongers. The corresponding provenance graph that describes that

situation is illustrated in Figure 1(a). To restore the restaurant's reputation, the managers decided to publish the report and use the associated provenance graph as an evidence. But they do not want to reveal any personal information, for instance, the entity *post_0* has attributes which can be used to expose the identity of the individual who initially posted the photo on the social network website. The template-based PROV-GTS is capable of isolating the node *post_0* without affecting the connectivity and integrity of the graph, i.e. no false (in)dependency, as shown in the transformed graph of Figure 1(b).



(a) The Original Graph (b) The Transformed Graph

Fig. 1. A Running Example

3 Related Work

Several graph redaction and sanitization approaches which are capable of obscuring private provenance information by deleting or abstracting sets of nodes have been proposed in the literature [13]. However, these approaches have not been fully built based on the fundamental theory of algebraic graph transformation [14] or adhesive high-level replacement (HLR) systems [15], which makes it difficult to look at various properties such as confluence and termination of rewriting systems [16].

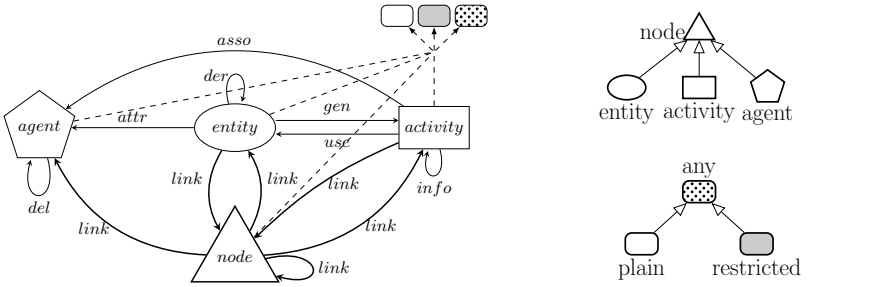
The graph grammar proposed in [8] performs edge and node contraction by replacing two nodes by a new abstract node and eliminating the edge connecting them. It also performs path redaction by replacing the entire path by an edge between the two end points of the path, however this may result in false independency since the system may eliminate the edges incident to the deleted nodes in the path. Abstracting, anonymizing, and hiding nodes by using a new non-functional node and then maintaining the essential relationships are presented in [9]. The system managed to avoid false dependency, however, hiding the non-sensitive nodes that violate integrity policy results in false independency.

A chunk of entities (or activities) containing sensitive information replaced by one abstract entity (or respectively abstract activity) is an approach suggested by [17]. The algorithm avoids cycles and invalid edges by expanding the set of abstracting nodes with new nodes. However, new relations may appear in the transformed graph, as a result of node abstraction, which are not part of the original graph resulting in false dependency. A graph transformation approach proposed in [6] restricts access to sensitive provenance information by removing nodes or replacing them with the new invented abstract nodes and edges. This approach may cause false dependency between the nodes adjacent to the abstracted set of nodes.

4 PROV Data Model and Rule Construction

The type graph of the core PROV model includes three node types, namely entity, activity, and agent and a set of edges as illustrated in Figure 2(a). In the proposed system, the sensitive parts of the provenance graph are specified by a set of *restricted* nodes, depicted as grey nodes ▭, while the non-restricted (*plain*) nodes are illustrated as white nodes ▭. These patterns represent the confidentiality level of the graph nodes and are illustrated in Figure 2(a) as data nodes with (*dashed*) edges from the graph nodes. Instead of using the confidentiality level nodes, we draw the graph nodes with the corresponding graph patterns that represent their confidentiality levels.

In addition to these concrete nodes types, an abstract graph node *node* △ and an abstract confidentiality level node *any* (depicted as dotted pattern ▭) have been added to the PROV type graph which are essential in constructing the template-based PROV-GTS. The (node and edge) labels are used for the purpose of clarification, they are not part of the system's formal description.



(a) PROV type graph (b) PROV-GTS inheritance graphs

Fig. 2. PROV type graph and PROV-GTS inheritance graphs

Relations between nodes in the PROV data model can be summarized or expanded using various inference rules. Some of these inference rules have been used to construct the proposed graph transformation templates. In order to guarantee that deleting certain edges will not affect the dependencies between other nodes in the graph, we construct conditional rules based on a set of properties prov1, prov2 and prov3 from the PROV data model, as shown in Figure 3. Each of prov1 and prov2 is constructed from the PROV model inference rules Inference 5 and Inference 6 respectively [11], while prov3 comes from the fact that the derivation (*der*) edge implies the existence of an activity which connects the generated and used entities [1]. These properties are used to build the required conditions for the deletion rules and to construct the creation rules. Suppose *Graph* is a category of provenance graphs as objects and inclusion morphisms as arrows.

Definition 1 (PROV property). *A PROV property* prov*i is* $p_i : C_i \to E_i$ *in Graph for* $i = 1..3$ *where* C_i *and* E_i *are respectively premise and conclusion of PROV properties, and* p_i *is the obvious inclusion morphism.*

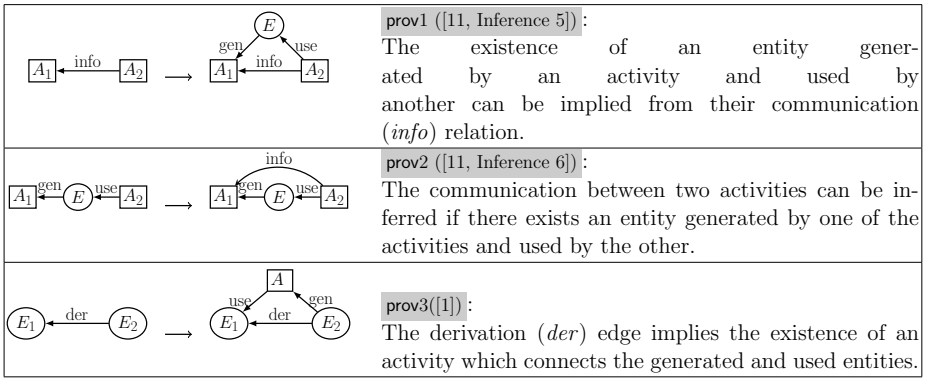
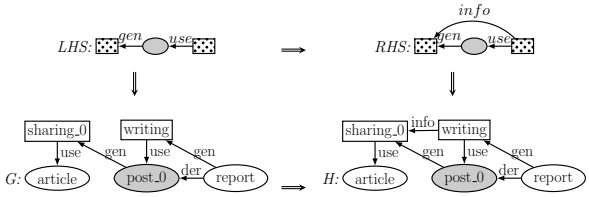


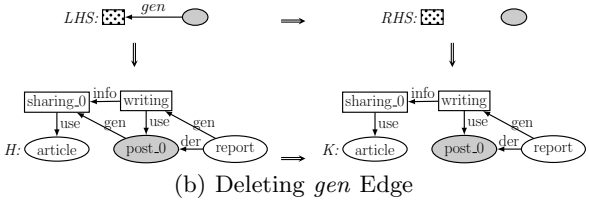
Fig. 3. PROV Model Properties

The single-pushout (SPO) approach of algebraic graph transformation defines transformation rules that consist of the left-hand side (*LHS*) and right-hand side (*RHS*). The rule is applied on a graph G by replacing a match of *LHS* in G by *RHS* resulting in a new graph H [18].

Deleting the edges connected to restricted nodes may result in omitting non-relevant information. In the graph of Figure 4, we show why creating some nodes and edges is important before deleting the edges connected to restricted nodes. For example, if we apply the deletion rule in Figure 4(b) before the creation rule in Figure 4(a), then we will no longer be able to infer the *info* edge between the activities *writing* and *sharing_0*, i.e. causing *false independency*. So, it is important to create nodes and edges as necessary to avoid losing information. But nodes and edges must be created according to the PROV data model, otherwise *false dependencies* may ensue. The LHS and RHS of each creation rule, such as the one in Figure 4(a), are constructed from one of the properties prov2 or prov3.



(a) Creating *info* Edge



(b) Deleting *gen* Edge

Fig. 4. Preserving Relations Before Deleting Edges

The PROV model allows simultaneous edges [11], for example, two or more activities may collaborate to generate an entity ([11, Constraint 39]), or an entity can be used by more than one activity. The simple transformation rules that consist only of LHS and RHS are not enough to preserve simultaneous relations. Regarding our running example, suppose another user shared the same post on a social media website which resulted in another entity *post_1* as shown in Figure 5. The edge *gen* between the entity *post_0* and the activity *sharing_0* can be deleted since the *info* relation between *writing* and *sharing_0* exists, but it is also part of another *use-gen* pattern between the nodes *sharing_1*, *post_0* and *sharing_0*. Deleting this *gen* edge prevents us from inferring the *info* edge between *sharing_1* and *sharing_0* (illustrated by the red dotted line). To tackle this issue, we need to check this kind of patterns universally rather than existentially using universal-existential conditions, as described in Sect. 6.

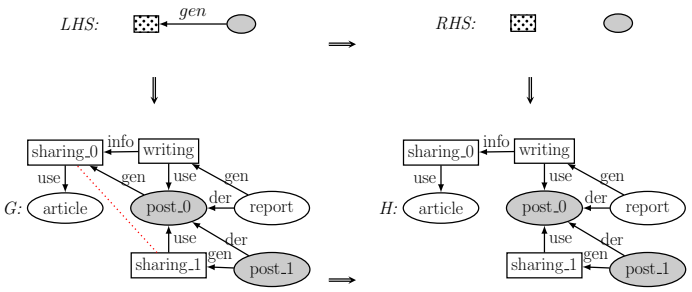


Fig. 5. Transforming a Provenance Graph with Simultaneous Edges

5 Typed PROV Graph with Inheritance

In construction of the conditional rules in template-based PROV-GTS some conditions are required to check frequent graph patterns and others are important to check non-existence of particular graph patterns, i.e. negative conditions [19]. In both cases, abstract nodes are essential to avoid having many graph patterns to satisfy a particular condition which allows us to merge similar conditions into one condition and group the rules with similar patterns in templates. We expand the type graph of the PROV model to have an abstract graph node and an abstract confidentiality level node via inheritance as shown in Figure 2(a). The inheritance graphs of nodes are illustrated in Figure 2(b).

Definition 2 (PROV type graph with abstract nodes). *A PROV type graph is a distinguished graph* $TG = (N_{TG}, E_{TG}, C_{TG}, B_{TG}, s_{TG}, t_{TG}, c_{TG}, d_{TG})$. *The sets* N_{TG} *and* C_{TG} *are called the graph nodes and confidentiality level nodes with abstract nodes respectively, and* E_{TG} *and* B_{TG} *are the graph edges and confidentiality level edges respectively. The functions* $s_{TG}, t_{TG} : E_{TG} \to N_{TG}$ *assign to graph edges their source and target graph nodes and* $c_{TG} : B_{TG} \to N_{TG}$ *and* $d_{TG} : B_{TG} \to C_{TG}$ *assign graph nodes and confidentiality level nodes to confidentiality level edges respectively.*

We adopt the definition of inheritance type graphs in [20]. Combining the PROV type graph with abstract nodes and the inheritance graphs result in PROV type graph with inheritance. Each node has an inheritance clan which represents its subtypes.

Definition 3 (PROV type graph with inheritance). *The PROV type graph with inheritance is* $ITG = (TG, IN, IC, AN, AC)$ *consists of a PROV type graph with abstract nodes* TG*, the inheritance graphs* IN *and* IC *with the same set of graph nodes and confidentiality level nodes as* TG*, and the sets* AN *and* AC *which represent the abstract graph nodes and abstract confidentiality level nodes respectively. The inheritance clans are defined by:*

- $\forall n \in N_{IN}$*:* $clan_{IN}(n) = \{n^{'} \in N_{IN} \mid \exists$ *path* $n^{'} \xrightarrow{*} n \in E_{IN}\} \cup \{n\}$
- $\forall c \in N_{IC}$*:* $clan_{IC}(c) = \{c^{'} \in N_{IC} \mid \exists$ *path* $c^{'} \xrightarrow{*} c \in E_{IC}\} \cup \{c\}$

The type graphs with inheritance can be flattened to ordinary graphs, called closure of type graph with inheritance, by adding normal edges implied by the subtype edges and removing the subtype edges [20]. In our system, the abstract graph node *node* can have an edge to each node in its inheritance clan. Also, each node in *clan(node)*, except *agent*, can have an edge to *node*. In addition, each graph node has an edge to the abstract confidentiality level node *any*. The PROV type graph with abstract nodes TG (as illustrated in Figure 2(a)) represents the closure of the PROV type graph with inheritance. So we can define the typed PROV graph and the corresponding morphism directly with respect to the type graph TG.

Definition 4 (Typed PROV graph with inheritance). *A typed PROV graph* $G_T = (G, type)$ *over* $ITG = (TG, IN, IC, AN, AC)$ *is an abstract instance of* TG*, i.e.* $(G, type : G \to TG)$ *consists of four morphisms* $type_{N_G} : N_G \to N_{TG}$*,* $type_{E_G} : E_G \to E_{TG}$*,* $type_{C_G} : C_G \to C_{TG}$ *and* $type_{B_G} : B_G \to B_{TG}$ *such that* $type_{N_G} \circ s_G = s_{TG} \circ type_{E_G}$*,* $type_{N_G} \circ t_G = t_{TG} \circ type_{E_G}$*,* $type_{N_G} \circ c_G = c_{TG} \circ type_{B_G}$ *and* $type_{C_G} \circ d_G = d_{TG} \circ type_{B_G}$*.*

Definition 5 (Typed PROV graph morphism with inheritance). *The morphism between two typed graphs with inheritance* $f : G \to H$ *consists of four functions:* $f_N : N_G \to N_H$*,* $f_E : E_G \to E_H$*,* $f_C : C_G \to C_H$ *and* $f_B : B_G \to B_H$ *such that* $f_N \circ s_G = s_H \circ f_E$*,* $f_N \circ t_G = t_H \circ f_E$*,* $f_N \circ c_G = c_H \circ f_B$*,* $f_C \circ d_G = d_H \circ f_B$*,* $type_{N_G} \subseteq clan_{IN}(type_{N_H} \circ f_N)$*,* $type_{C_G} \subseteq clan_{IC}(type_{C_H} \circ f_C)$*,* $type_{E_G} = type_{E_H} \circ f_E$ *and* $type_{B_G} = type_{B_H} \circ f_B$*. A partial graph morphism* $g : G \to H$ *is a total graph morphism from some sub-graph* K *of* G *to* H*, where* $N_K \subseteq N_G$ *and* $E_K \subseteq E_G$*.*

6 Conditional Graph Transformation Rules

The creation rules (e.g. Figure 4(a)) require negative application conditions to avoid an infinite number of rule applications. Similarly, the deletion rules (e.g.

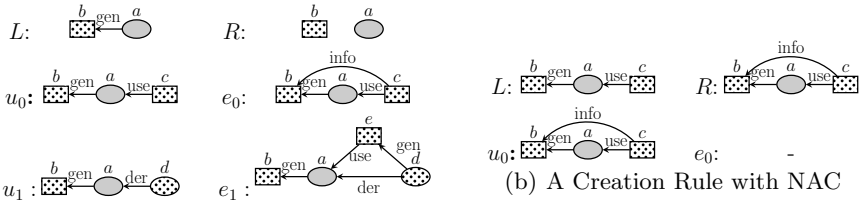
Figure 4(b)) need universal-existential conditions to deal with the simultaneous edges. To this end, the simple rules, which consist of LHS and RHS in the form of $r : L \to R$, must be extended by additional application conditions [16]. An application condition is a graph constraint $c(u, e)$ that is represented by a total morphism $(c : u \to e)$ in *Graph*. Suppose $r : L \to R$ is a simple rule and $c : u \to e$ is a graph constraint, then these possible conditions can be defined: (i) universal-existential $\forall_\exists$: if $L \to u$ is total and $u \hookrightarrow e$ is inclusion. (ii) negative application condition NAC: if $L \to u$ is total and e is empty. Now, a simple rule and a set of graph constraints can be combined to construct a conditional rule.

Definition 6 (Conditional rule). *A conditional rule $\nabla = (r, C, M)$ consists of a simple rule $r : L \to R$, a set of conditions $C = \{c_0, c_1 \dots\}$ and a set of morphisms $M = \{m_0, m_1, \dots\}$ where $c_i : u_i \to e_i$ and $m_i : L \to u_i$ are morphisms in Graph for $i = 0 \dots |C| - 1$.*

The conditional rule $\nabla = (r : L \to R, C, M)$ is applicable on a PROV graph G if the match $f : L \to G$ satisfies the set of conditions C.

Definition 7 (Conditional rule satisfaction). *Let $\nabla = (r : L \to R, C, M)$ be a conditional rule and G be a PROV graph, then $f : L \to G$ satisfies C if it (i) universally satisfies all $g : u_i \to G$ and $m_i : L \to u_i$ for each $c_i : u_i \to e_i$ such that $f = g \circ m_i$, and (ii) existentially satisfies at least one $h : e_i \to G$ such that $g = h \circ c_i$.*

In Figure 6 two rules are illustrated, a deletion rule and a creation rule. In Figure 6(a) the simple deletion rule shown in Figure 4(b) is extended with two universal-existential conditions, first to check that for each incoming *use* to the restricted entity there exists an *info* edge (u_0, e_0) and the second to ensure the existence of an activity for each incoming *der* edge (u_1, e_1). These two conditions are constructed from the PROV model properties prov2 and prov3, respectively. Similarly, in Figure 6(b) a negative application condition (u_0, e_0) is added to the creation rule in Figure 4(a) to avoid adding the same *info* edge again and again.



(a) A Deletion Rule with $\forall_\exists$

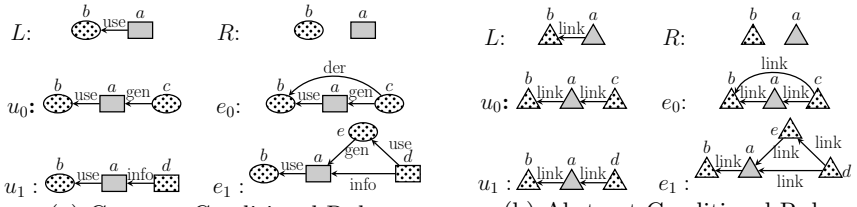
(b) A Creation Rule with NAC

Fig. 6. Universal-Existential and Negative Application Conditions

7 Grouping the Conditional Rules Using Templates

In this section, we define the notion of rule templates to group together the rules that have similar graph patterns in left- and right-hand sides, as well as the set of

conditions. For example, if we consider removing a *use* edge when the activity is *restricted*, we end up with another rule with two universal-existential conditions to ensure that for each incoming *gen* edge there exists a *der* edge and for each incoming *info* edge there is an entity, as depicted in Figure 7(a). The constructed rule has the same graph structure as the *gen* deletion rule in Figure 6(a) but with different nodes. We define a template with the same structure of the two rules by using the abstract node *node*, as shown in Figure 7(b).



(a) Concrete Conditional Rule (b) Abstract Conditional Rule

Fig. 7. Conditional Rules With Concrete and Abstract Nodes (template6)

The rule templates have the same structure as the conditional rules but consist only of abstract nodes and each rule template has a set of instance rules with the same structure. The template is instantiated using the nodes and edges from one of its rule instances to construct that rule. Regarding the rules with similar shapes, we define an abstract rule with the same shape but by replacing each concrete node with the abstract node *node*.

Definition 8 (Abstract conditional rule). *Abstract conditional rule is a conditional rule* $\nabla_A = (r_A : L_A \to R_A, C_A, M)$ *which is constructed from graphs that consist only of abstract graph nodes.*

Using the same construction methodology above, a set of rule templates, as shown in Figure 8, is constructed. Each rule template is accompanied by two tables, where each column in the first table with its corresponding column in the second table represent respectively the set of nodes and the set of edges of an instance of that template. Due to the limited space in this paper, we provide a subset of the actual set of templates and a subset of the rule instances for each template regarding only restricted entities and restricted activities. But this subset is sufficient to show the formal description of template-based PROV-GTS.

The rule templates are defined based on categorical constructions such as functors, natural transformation, co-cones, and pushouts [21,22]. The components of each rule template can be mapped to the corresponding components of the rule instances by a normal functor between the category that represent the template to the category that represents the rule instances. But also, due to the monomorphisms between the components of the rule instances, we can merge those components in a single rule graph and map the template to it by a constant functor rather than by a normal one.

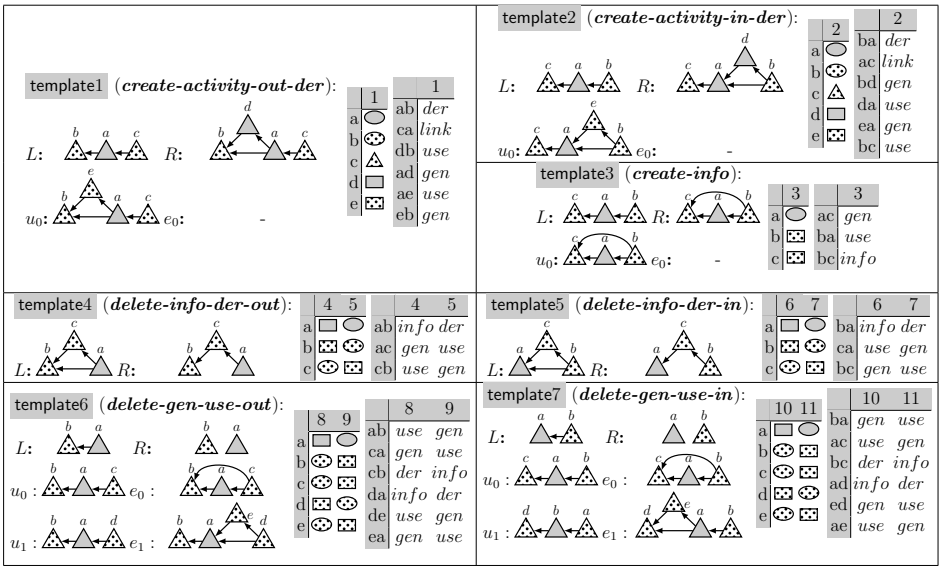


Fig. 8. Set of Rule Templates

Definition 9 (Rule graph). *Let* $\nabla = (r, C, M)$ *be a conditional rule with* $r : L \to R$, $C = \{c_i : u_i \to e_i\}$, *and* $M = \{m_i : L \to u_i\}$ *for* $i = 0 \ldots |C| - 1$ *then the rule graph* $\mathcal{R}$ *is the union of components of* ∇.

$$\mathcal{R} = L \cup R \cup \{ \bigcup_{i=0}^{|C|-1} u_i \} \cup \{ \bigcup_{i=0}^{|C|-1} e_i \}$$

Such that the following diagram commutes for each (u_i, e_i) *excluding* $e_i = \emptyset$*:*

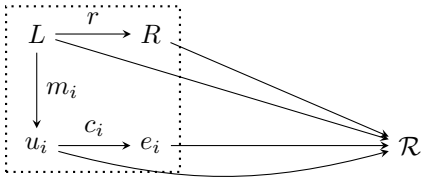


Figure 9 shows a graph transformation rule consisting of a left-hand side L, a right-hand side R, a negative condition u_0 (the only instance of template2), and the rule graph that is constructed from the components of that rule.

Suppose $Graph_{pITG}$ and $Graph_{ITG}$ represent the categories of typed graphs with inheritance with partial and total morphisms, respectively. A diagram of shape δ in $Graph_{ITG}$ is the homomorphism $D : \delta \to Graph_{ITG}$ where δ is a shape graph of the diagram D [21]. We define a diagram from a shape graph that represents an abstract conditional rule, to the rule itself and a constant functor from the shape graph to each rule graph. Then, the template is a set of natural transformation morphisms from the diagram to the constant functors.

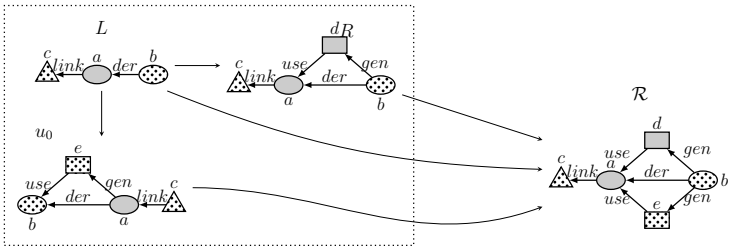
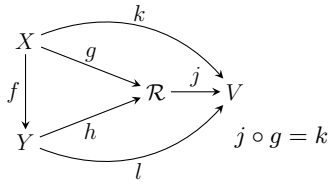


Fig. 9. Rule Graph for an Instance of template1

Definition 10 (Rule template). *Given a diagram functor $D : \delta \to \mathbf{Graph_{ITG}}$ on ∇_A and a set of constant functors $t : \delta \to \mathbf{Graph_{ITG}}$ on $\mathcal{R}$ where δ is a shape graph of the abstract conditional rule ∇_A and $\mathcal{R}$ is a rule graph, then the rule template τ is a set of natural transformation morphisms $\psi : D \to t$ such that for each $(f : X \to Y) \in \delta$, $\psi_X = \psi_Y \circ D(f)$ and $(\mathcal{R}, \psi)$ is a co-cone on D.*

Based on the above definition, $D(\delta)/\mathcal{R}$ is a slice category which is used in defining the template satisfaction below.

Definition 11 (Template satisfaction). *Given a diagram functor $D : \delta \to \mathbf{Graph_{pITG}}$ on $\nabla_A = (r_A : L_A \to R_A, C_A, M)$, a rule graph $\mathcal{R}$, and a morphism $j : \mathcal{R} \to V$ in $\mathbf{Graph_{pITG}}$ then V satisfies $\mathcal{R}$ if there exists a functor $F : D(\delta)/\mathcal{R} \to D(\delta)/V$ such that for each $f : X \to Y$ in $D(\delta)$ except $r_A : L_A \to R_A$ the following diagram commutes:*



Suppose the *writing* activity was not provided in the running example as illustrated in Figure 10 which shows an example of using a template for finding a match of a rule graph in a provenance graph. To apply the rule which is represented by the rule graph $\mathcal{R}$ as an instance of a template as defined above, we use the single-pushout approach but instead of direct modification, the rule application takes place through the template that represents the rule instance.

Definition 12 (Template application). *Let $\nabla_A = (r_A : L_A \to R_A, C_A, M_A)$ be an abstract conditional rule, S be a shape graph consisting of only $r_A : L_A \to R_A$. Given a rule graph $\mathcal{R}$, a diagram functor $\mathcal{D} : S \to \mathbf{Graph_{pITG}}$ on r_A and a constant functor $\mathcal{F} : S \to \mathbf{Graph_{pITG}}$ on $\mathcal{R}$, then there is a natural transformation $\eta : \mathcal{D} \to \mathcal{F}$ such that for $V, W \in \mathbf{Graph_{pITG}}$, V satisfies $\mathcal{R}$, there exists two co-cones (V, γ) and (W, μ) and the following diagram is a pushout:*

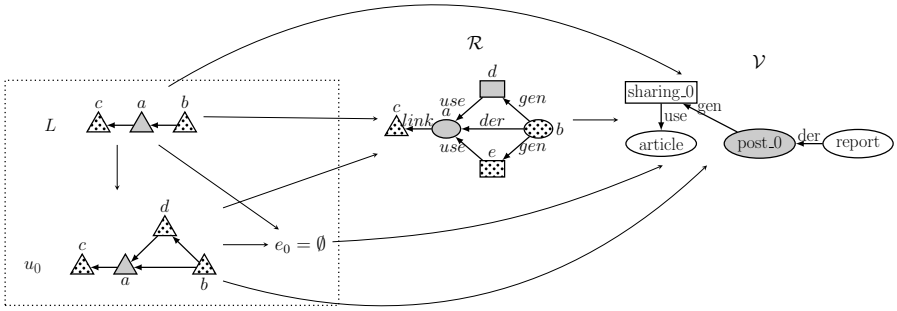
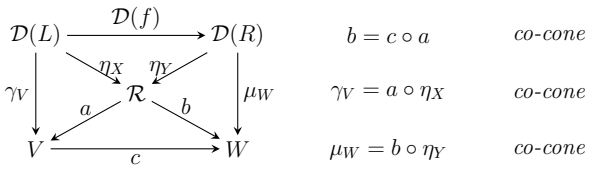


Fig. 10. An Example on Template Satisfaction using template1 and the Graph $\mathcal{V}$



For the creation templates, the triangle $\eta_X = \eta_Y \circ \mathcal{D}(f)$ represents a co-cone as well. An example of using template in transformation rule application is shown in Figure 11.

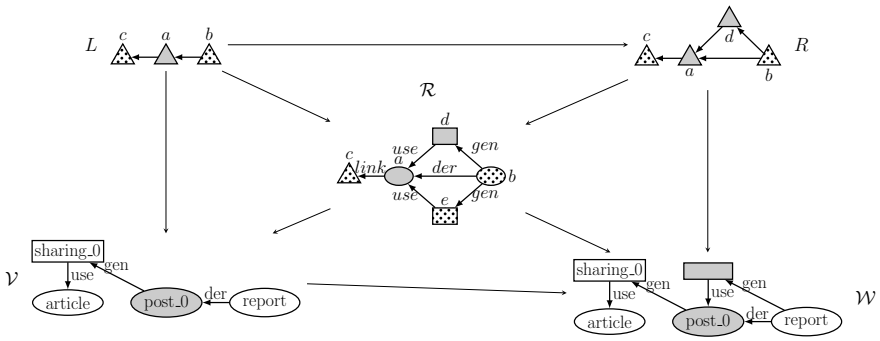


Fig. 11. An Example on Template Application using template2 and the Graph $\mathcal{V}$

The proposed graph transformation system is defined as a set of rule templates with one or more conditional rule instances for each template defined over the type graph with inheritance ITG.

Definition 13 (PROV Graph Transformation System). *A template-based PROV graph transformation system* PROV-GTS $= (ITG, TEMP, INS)$ *consists of a type graph with inheritance ITG, a set of rule templates $TEMP = \{\tau_{i=1...7}\}$ and a collection of rule instances INS where $I_i \in INS$ is a set of conditional rules belonging to the rule template τ_i.*

8 Provenance Graph Integrity

The integrity of provenance graphs is guaranteed by constructing the template-based PROV-GTS based on the PROV data model so that avoiding false dependencies and false independencies and preserving paths. Suppose G is the original graph and H is the graph transformed by template-based PROV-GTS rules.

Theorem 1 (No False Dependency). *The elements in $H \backslash G$ can be justified by the properties of PROV data model, i.e. there are no false dependencies.*

Proof (No False Dependency). The elements created by the creation templates via their rule instances must be constructed from the PROV data model properties such that they can be inferred from the original graph G. The creation templates are template1, template2 and template3. Each creation template has one rule instance. The LHS and RHS of the instances of template1 and template2 are inclusions to respectively premise and conclusion of prov3 while the rule instance of template3 is inclusion to prov2.

Theorem 2 (No False Independency). *Any two plain nodes in G are preserved in H along with all the edges between them including the edges that can be inferred from G.*

Proof (No False Independency). Since the creation templates create *activity*, *use*, *gen*, and *info* regarding the restricted entities, in the following we examine each rule instance related to restricted entities and show how they avoid false independency by providing adequate universal-existential conditions:

- template4, 5: These templates delete *der* only when there is an activity connecting between the two entities.
- template6, 7: The condition (u_0, e_0) checks the existence of an *info* for each incoming *use* when deleting a *gen* edge and for each outgoing *gen* when deleting a *use* edge. Similarly, the condition (u_1, e_1) ensures that for each incoming or outgoing *der* there exists an activity. These two universal-existential conditions are sufficient to check all possible graph patterns that include the deleted edge.

Theorem 3 (Path Preservation). *Any path between two plain nodes in G is preserved in H.*

Proof (Path Preservation). The proof of Theorem 2 shows that the relation between the nodes adjacent to restricted entities are preserved before deletion. Now we examine the situation when the restricted node is an activity.

- template4, 5: These templates delete an *info* edge when there exists an entity generated by the informant activity and used by the informed activity.
- template6, 7: The condition (u_0, e_0) checks the existence of a *der* edge for each incoming *gen* when deleting a *use* edge and for each outgoing *use* when deleting a *gen* edge and (u_1, e_1) ensures that for each incoming or outgoing *info* there exists an entity.

To proof the termination of our proposed system, the layered termination criteria defined in [23] can be used by distributing the seven templates, rather than the rule instances, over a number of deletion and non-deletion layers. The details will be provided in the forthcoming full version of this paper.

9 Conclusion

A template-based graph transformation system is proposed capable of isolating restricted nodes from provenance graphs as long as the integrity of the graph is not violated. The system consists of a set of graph transformation rule templates constructed based on the PROV data model. Each template has one or more rule instances and rule application happens using the template via the rule instance.

Any future development of the system should include processing the edges connected to the agent nodes as well as the agents that represent entities or activities in the same graph. In addition, the system can be expanded to maintain the provenance temporal information by preserving the sequences of operations such as derivation and generation in provenance graphs. Finally, analysing the rule applications that represent critical pairs is important to prove parallelism, local confluence, and confluence of the system. A different line of investigation is to look at some recent work on generic GTS which might be useful in this context, for example an alternative to rule templates could be variability-based graph transformation [24,25].

References

1. Moreau, L., Missier, P.: PROV-DM: The PROV Data Model. Technical report, W3C Recommendation, W3C, `http://www.w3.org/TR/prov-dm/` (2013)
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, ACM (2008) 1345–1350
3. Garijo, D., Corcho, O., Gil, Y.: Detecting common scientific workflow fragments using templates and execution provenance. In: Proceedings of the seventh international conference on Knowledge capture, ACM (2013) 33–40
4. Kifor, T., Varga, L., Vazquez-Salceda, J., Alvarez, S., Willmott, S., Miles, S., Moreau, L.: Provenance in agent-mediated healthcare systems. Intelligent Systems, IEEE **21**(6) (2006) 38–46
5. 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, ACM (2010) 2–7
6. Danger, R., Curcin, V., Missier, P., Bryans, J.: Access control and view generation for provenance graphs. Future Generation Computer Systems **49** (2015) 8–27
7. Moreau, L., Groth, P.: Provenance: An introduction to PROV. Synthesis Lectures on the Semantic Web: Theory and Technology **3**(4) (2013) 1–129
8. 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. SACMAT '11, New York, NY, USA, ACM (2011) 93–102

9. Dey, S., Zinn, D., Ludäscher, B.: Propub: Towards a declarative approach for publishing customized, policy-aware provenance. In Bayard Cushing, J., French, J., Bowers, S., eds.: Scientific and Statistical Database Management. Volume 6809 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2011) 225–243
10. Hussein, J., Moreau, L., Sassone, V.: Obscuring provenance confidential information via graph transformation. In: Trust Management IX. Volume 454 of IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg (2015) 1–17
11. Cheney, J., Missier, P., Moreau, L., DeNies, T.: Constraints of the PROV data model. Technical report, W3C Recommendation, W3C, http://www.w3.org/TR/prov-constraints/ (2013)
12. Moreau, L., Missier, P., Cheney, J., Soiland-Reyes, S.: PROV-N: The Provenance Notation. Technical report, W3C Recommendation, W3C, `http://www.w3.org/TR/prov-n/` (2013)
13. Cheney, J., Perera, R.: An analytical survey of provenance sanitization. arXiv preprint arXiv:1405.5777 (2014)
14. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series). Secaucus. Volume 373. NJ, USA: Springer-Verlag New York, Inc (2006)
15. Sassone, V., Sobocinski, P.: Congruences for contextual graph-rewriting. BRICS Report Series **11**(11) (2004)
16. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamental theory for typed attributed graphs and graph transformation based on adhesive hlr categories. Fundamenta Informaticae **74**(1) (2006) 31–61
17. Missier, P., Bryans, J., Gamble, C., Curcin, V., Danger, R.: Provabs: model, policy, and tooling for abstracting prov graphs. 5th Intl. Provenance and Annotation Workshop IPAW'14 (2014)
18. 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. (1997) 163–246
19. Löwe, M., König, H., Schulz, C., Schultchen, M.: Algebraic graph transformations with inheritance and abstraction. Science of Computer Programming (2015)
20. Bardohl, R., Ehrig, H., De Lara, J., Runge, O., Taentzer, G., Weinhold, I.: Node type inheritance concept for typed graph transformation. Technische Universität Berlin, Fakultät IV-Elektrotechnik und Informatik (2003)
21. Barr, M., Wells, C.: Category theory for computing science. Volume 49. Prentice Hall New York (1990)
22. Awodey, S.: Category theory. Oxford University Press (2010)
23. Ehrig, H., Ehrig, K., de Lara, J., Taentzer, G., Varro, D., Varro-Gyapay, S.: Termination criteria for model transformation. In Cerioli, M., ed.: Fundamental Approaches to Software Engineering. Volume 3442 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 49–63
24. Strüber, D., Rubin, J., Chechik, M., Taentzer, G.: A variability-based approach to reusable and efficient model transformations. In: Fundamental Approaches to Software Engineering. Springer (2015) 283–298
25. Taentzer, G., Plöger, J.: Rulemerger: Automatic construction of variability-based model transformation rules. In: Fundamental Approaches to Software Engineering. Volume 9633., Springer (2016) 122