UNIVERSITY OF SOUTHAMPTON

PROV-GTS: A Template-based PROV Graph Transformation System for Obscuring Provenance Confidential Information

by

Jamal A. Hussein

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Faculty of Physical and Applied Sciences
Electronics and Computer Science

12th June 2017

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Jamal A. Hussein

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. Provenance can be used to trace the source of ingredients in food industry, record the intermediate and final results of scientific workflows, and follow the origin of online posts and news. Data provenance becomes a significant metadata in validating the origin of information and asserting its quality. It has been adopted in many significant domains for different purposes, for example, validating experimental results in scientific workflow systems, improving health services in healthcare, trustworthiness of data from sensor networks, and managing access control systems. In particular, the provenance of information is crucial in deciding whether information is to be trusted.

In order to establish trust and confirm the quality and originality using provenance information, it is important to share provenance among collaborators in scientific workflow systems or publicly over open environments such as the Web. PROV is a recent W3C specification for sharing provenance over the Web.

However, 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. It is therefore crucial for the sensitive and confidential information of provenance data to be obscured to enable trustworthiness prior to sharing provenance in open environments such as the Web.

This research work describes PROV-GTS, a provenance graph transformation system, whose principled definition is based on PROV properties, and which seeks to preserve graph integrity by avoiding false independencies and false dependencies while obscuring restricted provenance information. The system is formally established as a template-based framework and formalised using category theory concepts, such as functors, diagrams, and natural transformation. PROV-GTS is shown to preserve graph connectivity, to be terminating and to be confluent with deterministic and consistent rule applications. The performance evaluation based on real-world provenance graphs demonstrates a high connectivity preservation with minimum graph reduction.

Acknowledgements

First and foremost I would like to express my sincere gratitude and thanks to my supervisors Professor Vladimiro Sassone and Professor Luc Moreau for their guidance, encouragement, patient and continuous support in overcoming numerous obstacles I have been facing throughout the period of this study.

I would like to thank all the provenance group members for their support especially for providing me with provenance datasets which helped me in conducting the performance evaluation of the system.

Last but not the least, I would like to thank my parents for supporting me spiritually throughout my life, and most importantly I would like to thank my wife for her love, patient, encouragement and understanding despite the difficulties.

Contents

List of Figures

List of Tables

Glossary

$Graph$ The category of PROV graphs with reverse indicator and total graph morphisms.

$Graph_p$ The category of PROV graphs with reverse indicator and partial graph morphisms.

ID_A each object A in a category has an identity arrow (morphism). 131

∇ A conditional rule $\nabla = (r, C, M)$. 61

∇_A An abstract conditional rule $\nabla_A = (r_A, C_A, M)$. 79

$frwd(G)$ a function which returns the graph G after reversing all its edges, otherwise it returns G itself.

$g \circ f$ the composition of two arrows (morphisms) f and g such that $cod(f) = dom(g)$. 131

AC A set that consists of the abstract confidentiality level node *any*. 54

AG The PROV type graph with abstract nodes which consists of the nodes and edges of the CG plus the abstract graph and the confidentiality level node.

AN The set of abstract graph nodes consists of the two nodes *node* and *artifact*. 54

CG The PROV type graph with confidentiality level nodes which consists of the nodes and edges of the core PROV data model plus a set of data nodes represent the confidentiality level of the graph nodes.

G^{-1} The reverse PROV graph where all edges are reversed.

G_T Typed PROV graph with inheritance.

$Graph_{ITG}$ The category of typed PROV graphs with inheritance with total morphisms. 58

$Graph_{pITG}$ The category of typed PROV graphs with inheritance with partial morphisms. 58

IC confidentiality inheritance graph. 55

IN Node inheritance graph. 55

ITG PROV type graph with inheritance.

TG The PROV type graph which consists of the nodes and edges of the core PROV data model.

$\forall_\exists$ universal-existential conditions.

$\overline{AG}$ Closure of PROV type graph with inheritance.

$cod(f)$ codomain of the arrow (morphism) f. 131

$dom(g)$ domain of the arrow (morphism) f. 131

provi A PROV property $p_i : C_i \to E_i$. 59

Category Category is a mathematical structure consists of a collection of objects and composable arrows (morphisms) with identity arrows and associativity. ix, 131–136

Confluence Confluence means that a sequence of rule applications are always result in the same graph despite the order in which they were applied. 95

diagram A diagram in category $\mathcal{C}$ of shape J is the functor $D : F(J) \to \mathcal{C}$ where $F(J)$ is the free category on the graph J. 135

false-dependency adding to a graph nodes and/or edges that cannot be justified from the graph based on PROV data model semantics. 5

false-independency happens when naïve deletion of an edge cut the dependency between some nodes which are adjusted to the source and the target of the deleted edge. 4

Free Category For any given directed graph G there is a free category $F(G)$ with the nodes of G as objects and the paths in G as arrows. x, 135

functor A functor $F : \mathcal{C} \to \mathcal{D}$ between the categories $\mathcal{C}$ and $\mathcal{D}$ consists of two operations $F_0 : C_0 \to D_0$ and $F_1 : C_1 \to D_1$, such that for each $f : A \to B$ in C_1 there is $F_1(f) : F_0(A) \to F_0(B)$ in D_1. 135

Isomorphism An arrow $f : A \to B$ in category $\mathcal{C}$ is *iso* (isomorphic) if there exist the inverse arrow $f^{-1} : B \to A$ in $\mathcal{C}$ such that $f^{-1} \circ f = ID_A$ and $f \circ f^{-1} = ID_B$. 133

Monomorphism An arrow $f : A \to B$ in category $\mathcal{C}$ is *mono* (monomorphic) if for any object C and any pair of arrows $g, h : C \to A$, $f \circ g = f \circ h$ implies $g = h$. 133

Morphisms morphism is an arrow between two objects in a category.

Natural Transformation natural transformation are morphisms between functors. x, 135

Parallel Independence Two rule applications are parallel independent if applying any one of them does not prevent the other to be applicable. 92

Pushout The pushout of two arrows $f : A \to B$ and $g : A \to C$ in any category $\mathcal{C}$ is given by the arrows $f^{'} : B \to P$ and $g^{'} : A \to P$ such that $f \circ g^{'} = g \circ f^{'}$. ix, 134

Sequential Independence Two rule applications are sequentially independent if one of them does not create elements that are required by the match of the other rule application. 93

Slice categories A slice category is $\mathcal{C}/A$ with a set of arrows from any object in $\mathcal{C}$ to A as objects and a set of arrows $f : B \to C$ from $g : A \to B$ to $h : A \to C$. ix, 132

Acronyms

DPO double-pushout. 18

LHS the Left-Hand Side of a graph transformation rule. 6,

NAC Negative Application Condition.

OPM Open Provenance Model. 1,

PROV W3C standardized PROVenance data model. 1,

PROV-DM PROV data Model.

PROV-GTS PROV Graph Transformation System. 2,

PROV-N PROV Notation.

PROV-O PROV Ontology.

RDF Resource Description Framework. 1,

RHS the Right-Hand Side of a graph transformation rule. 6,

SPO single-pushout. 18

W3C World Wide Web Consortium. 1,

XML eXtensible Markup Language. 1,

Chapter 1

Introduction

Nowadays, the Internet is so pervasive that it affects every aspect in our daily lives, from a breaking news story posted on a social media website to a new food product appearing on an online food store. Due to the vast amount of intertwined data available online, tracing the origin of a piece of information or confirming the quality of a trending product becomes a big challenge for both researchers and consumers. So, maintaining a record that shows the intermediate information and entities and the participants that influenced the various processes that lead to the current state of an object (digital, physical or conceptual) becomes significant for trust management and quality assurance. Such a record is known as provenance (Moreau 2010; Moreau and Groth 2013).

The word "provenance" stems from the French "provenir" meaning "to come from" and defined as "the place of origin of something" according to the Cambridge dictionary. Originally, the concept of provenance is related to works of arts where it refers to the documented history used in tracing the origin of a piece of art or artefact (Vázquez-Salceda, Álvarez, Kifor et al. 2008). Provenance can be used to trace the source of ingredients in the food industry (Morgan, Marsden and Murdoch 2008) and to record the intermediate and final results of scientific workflows (Davidson and Freire 2008). In addition, as social media, especially Facebook with 1.6 million users, becomes the main source of news including a lot of fake and misleading stories (Cellan-Jones 2016), provenance can also be used to follow the origin of online posts and news (Barbier, Feng, Gundecha et al. 2013). Provenance enables computer systems to capture the execution steps of computer applications that lead to the results they produce. To represent and interchange provenance information, it is important to have standard and domain-agnostic representations of provenance using known formats as RDF and XML (Groth and Moreau 2013), such that processes like maintaining, querying and retrieving data provenance can be managed by computer systems (Karvounarakis, Ives and Tannen 2010). Such models include the open provenance model (OPM) (Moreau, Clifford, Freire et al. 2011) and W3C standardized PROV data model (Lebo, Sahoo, McGuinness et al. 2013). Provenance can be stored in data provenance repositories with querying facilities which enables researchers and service providers to retrieve the provenance information (Groth, Jiang, Miles et al. 2006). Then, this provenance information can be used for various purposes, for example reproducing the same results using the same execution steps in scientific workflow systems (Freire, Koop, Santos et al. 2008).

Provenance is crucial to validate the origin and quality of data and to enable the reusability of data (Huynh, Ebden, Ramchurn et al. 2014). It has been adopted in many significant domains for different purposes, for example, validating experimental results in scientific workflow systems (Garijo, Corcho and Gil 2013; Davidson and Freire 2008), improving health services in healthcare (Kifor, Varga, Vazquez-Salceda et al. 2006; Kifor, Varga, Álvarez et al. 2006), trustworthiness of data from sensor networks (Lim, Moon and Bertino 2010), and managing access control systems (Danger, Curcin, Missier et al. 2015; Cadenhead, Khadilkar, Kantarcioglu et al. 2011a). For example, full provenance of medical decisions enriches the medical history captured in health records and could enable scientists to find the cause of diseases and care providers to improve health services (Kifor, Varga, Vazquez-Salceda et al. 2006). Overall, the provenance of information is essential to decide whether information is to be trusted (Moreau and Missier 2013).

In order to establish trust and confirm the quality and originality using provenance information, it is important to share provenance among collaborators in scientific workflow systems (Nagappan and Vouk 2008) or publicly over open environments such as the Web (Moreau 2010).

However, 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, we need an effective approach to deal with such private information. Provenance can be represented as directed graphs that explain how the entities and the processes involved in the generation of an artifact are related to each other (Moreau, Clifford, Freire et al. 2011). Regarding this graphical nature of provenance, the nodes that are located in certain parts of provenance graphs, which expose confidential information, can be annotated as restricted nodes. Then, these restricted nodes can be removed using graph transformation rules (Cadenhead, Khadilkar, Kantarcioglu et al. 2011b). In this thesis, a template-based provenance graph transformation system (PROV-GTS) is proposed capable of deleting the edges connected to restricted nodes in an attempt to isolate those nodes from the provenance graph as long as the graph integrity is not affected. The graph integrity is affected if, for example, the graph transformation process adds graph components that can not be justified by the original graph or results in creating false paths. Some nodes cannot be fully isolated due to the violation of graph integrity. If this is the case then the node will be replaced by an anonymous node.

Section 1.1 presents the PROV data model and principle of algebraic graph transformation. In Section 1.2, restricting provenance information while preserving graph integrity is discussed. Multiple, and possibly simultaneous, events in PROV data model are investigated in Section 1.3. The rationale of organizing rules in abstract templates is analysed in Section 1.4. Section 1.5 presents the research statement and the list of contributions. The papers published so far on the thesis's topic are provided in Section 1.6. Section 1.7 demonstrates the layout of this dissertation.

1.1 PROV Data Model and Graph Transformation

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" (Moreau and Missier 2013). The components of W3C standardized core PROV model can be represented as a directed graph, referred to as the PROV type graph, which consists of three node types: entity, activity, and agent as well as seven edge types. Entities are physical, digital, or conceptual things which were generated, used, consumed, and modified by activities, possibly under the responsibilities of some agents (Lebo, Sahoo, McGuinness et al. 2013). The PROV data model obeys a set of constraints and inference rules (Moreau, Huynh and Michaelides 2014). The PROV model constraints provide a logic specification that can be used to validate provenance, while the set of inference rules shows how some relations can be inferred from existing relations in the provenance graphs (Cheney, Missier, Moreau et al. 2013).

Provenance can be saved in databases, for instance by storing a record for each component (node or edge) of provenance graphs in database tables (Buneman, Chapman and Cheney 2006). Most of the database systems efficiently support querying languages that enable users looking up records that satisfy certain constraints. In addition, logic-based methods provides a formal and declarative semantics for representing and reasoning about provenance using predicates for expressing information and rules for deducing new information based on a given set of facts (Zhao and Lu 2008). But we chose graph-based representation of provenance since in computer science graphs represent a natural and intuitive mechanism for illustrating complex situations. They have been used almost everywhere, for example graphs are used for visualising UML diagrams, data flow diagrams, Petri nets, and indeed provenance. Rule-based alteration of graphs is the prime concept of graph transformation where each rule application represents a graph transformation step (Ehrig, Ehrig, Prange et al. 2006b). Graph transformation is used to model state changes of systems in many fields of computer science such as software engineering (Baresi and Heckel 2002), distributed systems (Taentzer, Goedicke and Meyer 2000), programming languages (Andries, Engels, Habel et al. 1999; Habel and Plump 2001), and visual modelling (Varró, Varró and Pataricza 2002).

1.2 Provenance Confidential Information and Graph Integrity

Due to the potential enormous amount of provenance information generated and stored in provenance databases (Chapman, Jagadish and Ramanan 2008; Rozsnyai, Slominski and Doganata 2011; Cheah, Plale, Kendall-Morwick et al. 2012) and the sensitivity of some of this information (Hasan, Sion and Winslett 2007; Davidson, Khanna, Roy et al. 2010), the process of hiding parts of this information is essential for any system to provide access to the provenance data (Anand, Bowers and Ludäscher 2012). This process must be capable of obscuring the sensitive information while preserving the integrity and accuracy of the information no matter how big the provenance data is. Data provenance can be represented as graphs with directed edges and typed nodes. Therefore, hiding parts of provenance graphs via graph transformations has been used in many areas such as access control and scientific workflow systems (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a; Davidson, Khanna, Roy et al. 2011b).

Provenance may include confidential information, such as agents' identities, time information, specific attributes of, and relations between, entities, activities and agents. For instance, suppose there is an online article which has been written based on fake information posted online and targeted the reputation of a restaurant. Figure 1.1 shows a diagram that describes the participants involved in generating the online post and also a report prepared by the restaurant's managers based on that online post. Data provenance can be used to describe the processes and entities involved in fabricating that online post. Then, this provenance data can be used as evidence to restore the reputation of the restaurant. However, the provenance information may expose confidential information of various participants involved in creating that online post. For example, the node that expose the identity of the user who initially shared that post on a social media network must not be revealed when publishing the provenance data. The above situation is used for constructing the provenance graph OnlinePost which will be used as a running example as described in Chapter 3.

Such a confidential information must be obscured before exposing provenance, but this poses problems given the graphical nature of provenance and associated inference rules (Cheney, Missier, Moreau et al. 2013). Indeed, obscuring a node or an edge containing confidential information may affect what can be inferred from a graph. For instance, if $a \longrightarrow b$ and $b \longrightarrow c$ for some transitive relation, confidential information in b, and subsequent deletion of the edge $a \longrightarrow b$ or the edge $b \longrightarrow c$, will prevent us from inferring $a \longrightarrow c$. Such a problem is being referred to as *false-independency* (Missier 2013) 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 (Moreau and Missier 2013)). For example, suppose the node *post* in Figure 1.1 has attributes that reveal login credentials of the user who initially share the post on a social media website. We could delete those

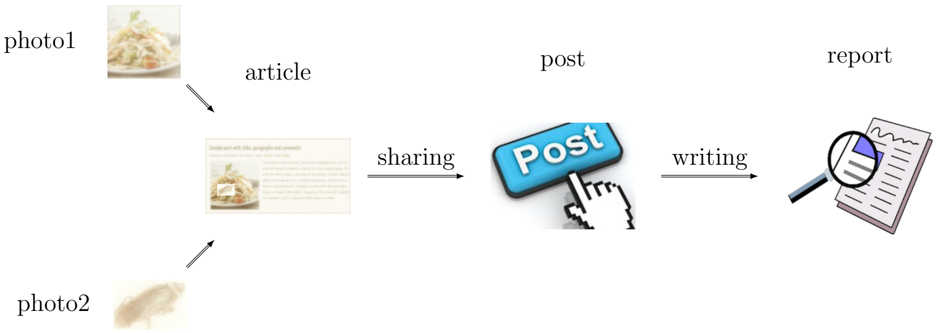


Figure 1.1: The online post diagram that describes a *report* prepared based on an online *post* that shared an *article* which contains a fake photo fabricated from two unrelated photos (*photo1* and *photo2*)

attributes and keep the node by anonymizing it, but these presents challenges as we will explain the reasons of it in Section 1.3. But also, naive deletion of the node *post* will cut any dependency between the report and the posted *article* which may result in *false independency*. 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* (Dey, Zinn and Ludäscher 2011a). For example, we cannot infer any immediate and non-transitive dependency between each of the nodes *photo1* and *photo2* and the node *post*, as there are no direct relations between them. The problems of false dependency and false independency have not been considered together by previous work on provenance privacy protection (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a; Davidson, Khanna, Roy et al. 2011b), but should critically be addressed in order to maintain the usefulness of provenance in establishing trust of data.

So, it is important to create nodes and edges as necessary to avoid *false independency*. But nodes and edges must be created according to the PROV data model, otherwise *false dependencies* may ensue.

To summarise, any provenance graph transformation system must preserve the graph integrity and connectivity based on the following requirements:

1. No false dependency: Any element created by the system must be justified from the original graph.
2. No false independency: The information that can be inferred from the original graph are not affected by the graph rewriting; Only the edges that directly connected to restricted nodes are deleted.
3. Path preservation: Every path between the non-restricted nodes in the original graph are still accessible and semantically correct in the transformed graph, probably with less information.

4. No false paths: The graph transformation process does not result in false paths.

5. No excessiveness: Only the edges connected to the restricted nodes are removed.

6. graph validity preservation: the transformed graph is always a valid provenance graph.

Based on the specifications of the core of W3C PROV data model, we construct the provenance graph transformation system PROV-GTS which consists of a set of template-based algebraic graph transformation rules. In its simplest form, the graph transformation rule consist of a *left-hand side(LHS)* and a *right-hand side (RHS)*. The left-hand side of each rule consists of the minimum graph pattern required by the rule to be applicable. Any extra conditions needed by the rule to accomplish its task are attached to it as additional conditions: negative application conditions (NAC) or the conditions that used to check multiple graph patterns referred to as universal-existential conditions. This rule extension is crucial to preserve graph integrity by avoiding deleting edges before ensuring that all relations are preserved. It prevents adding the same node or edge again and again. It guarantee consistency between the graph transformation rule applications, which secures the system's termination and confluence.

1.3 Multiple Events in the PROV Data Model

The richness of the PROV model requires a principled approach to defining a graph transformation and formalising its properties. To address the problem of false independency and false dependency, we have established that, when an edge needs to be deleted, we need to consider the edges incident to the source and target of the deleted edge. To do so, graph transformation rules need to be equipped with a variety of graph rewriting capabilities, such as negative application conditions (NAC) (König and Stückrath 2012) and universal-existential conditions (nested constraints) (Ehrig, Golas, Habel et al. 2014).

The PROV model allows multiple events represented by graph edges of the same type to occur on the same node, for example, two or more activities may involve in generation of an entity (Cheney, Missier, Moreau et al. 2013, Constraint 39); alternatively, an entity can be used by more than one activity (none of these usage events must consume the entity if it prevents the other usage to occur), and more than one activity may be associated with an agent. The simple transformation rules, which consist only of LHS and RHS, are not enough to avoid false independency and preserve paths, since they deal with single graph patterns existentially. To tackle this issue, we need to check this kind of patterns universally rather than existentially using universal-existential conditions.

The aim of PROV-GTS rules is to isolate the restricted nodes from the graph by deleting their incident edges one by one up to isolation. The graph transformation rules must have enough conditions that ensure preserving relations between the nodes adjacent to the source and target of the deleted edge. This preservation is important to maintain the integrity of provenance information by avoiding false independency and false dependency. In addition, since provenance represents the lineage of an artifact or thing, the paths that lead to a specific state are important when querying over paths, such that after hiding a sensitive node its predecessors are still approachable via path traversal queries (Blaustein, Chapman, Seligman et al. 2011). In PROV-GTS, creating nodes and edges leads to path preservation, such that if there exist a path between two nodes in the original graph there will be still a path connecting them in the target graph transformed by PROV-GTS. However, not all paths can be preserved, since some nodes cannot be fully isolated from the provenance graph as they are required for path preservation. So, isolating the nodes that reveal private and sensitive information may clash with the connectivity and integrity measures. In this case, we choose to preserve those nodes rather deleting them. This trade-off between privacy protection and connectivity preservation is important to maintain the usefulness of provenance graphs. However, preserving those nodes may expose sensitive information such as attributes and URLs associated with them. If this is the case then the nodes will be anonymised. Node anonymization is done by replacing the restricted node with a less-sensitive anonymous node of the same type and is carried out by obscuring the node's *id* and deleting all its attributes (Backstrom, Dwork and Kleinberg 2007).

We could anonymize all the restricted nodes keeping the topology of the provenance graph intact. But this presents challenges for the following reasons:

1. The relations between provenance nodes may represent sensitive information. Anonymizing such nodes keeps those relations intact which may somehow breach the sensitivity and privacy of provenance information. So deleting as much edges as possible is important to reduce the risk of exposing private information.

2. Anonymizing all the restricted nodes may result in a graph with many anonymous and redundant nodes which could have been deleted without losing any information rather than restricted nodes and their incident edges.

1.4 Grouping Graph Transformation Rules in Templates

The construction of PROV-GTS rules, regarding the core PROV data model and based on its properties, results in a large set of graph transformation rules. There are at least two deletion rules for each edge type, in addition to a number of creation rules. This construction resulted in twenty-four graph transformation rules, subject to expansion if we consider the full PROV data model. Each two rule applications are either parallel independent or they represent a safe critical pair. The applications of two rules are parallel independent if their matches in the host graph do not overlap in such a way that the application of one rule prevents the other rule from been applicable. In addition to parallelism and safety of critical pairs, some of these rules consist of graphs of similar topologies. By defining abstract nodes which will supersede the concrete nodes of PROV data model via inheritance, the rules with similar topologies are grouped in rule templates. This grouping process resulted in only eight rule templates. Each rule template consists of an abstract rule and one or more rule instances from the twenty-four graph transformation rules.

Category theory constructions such as graph homomorphisms and pushout are used in formalizing the graph transformation rule definitions, matching, and applications. Other constructions such as functors, sliced categories, and natural transformation morphisms are used in the formal definition of the proposed template-based graph transformation system. Functors are used to map between the rule templates and their instance rules, sliced category to formalize the definition of the provenance graph over a particular type graph, and natural transformations to construct the pushout diagrams that will be used in applying the templates on a provenance graph via the rule instances.

In PROV-GTS, rule application is the composition of the template instantiation and rule application operations according the following steps:

1. find an image (a match) of the LHS of the abstract rule of a template in the source provenance graph;
2. check if that image matches any of the rule instances of that template;
3. ensure that the provenance graph satisfies all the conditions of the rule instance via the abstract rule of the template;
4. apply the rule by replacing the image of LHS of the abstract rule of the template by the RHS of the rule instance, i.e. deleting the elements that are in LHS but not in RHS and creating the elements that are in RHS but not in LHS.

These template-based construction and rule application do not only reduce the complexity and improve scalability of the system in terms of the processing time, but also enhance the readability of the rules and facilitate the proofs of various properties. Instead of considering a large set of rules, analysing various graph transformation properties such as termination, critical pairs, parallelism, and confluence can be done on template basis. The system performance is evaluated by performing several metrics, such as the amount of information preserved by the system, the number of transformation steps applied in order to convert a provenance graph, and the average degree of the graph produced by PROV-GTS.

1.5 Research Statement and Contributions

> **A deterministic graph rewriting system is constructed from the PROV data model capable of hiding restricted provenance information while preserving graph integrity. The system is formally established as a terminating, parallel independent, and confluent set of algebraic graph transformation rules defined as a template-based framework and formalized using category theory concepts. The system evaluation shows high connectivity preservation with minimum graph reduction.**

The list of contributions:

1. A principled definition of PROV graph transformation rules, which are capable of obscuring restricted provenance information while maintaining graph integrity by preserving paths and avoiding false independencies and false dependencies.
2. A template-based framework that abstracts 24 graph transformation rules into 8 rule patterns, allowing a more concise definition of the transformation system and a tractable way of proving its properties.
3. The proof that the graph transformation system is terminating and confluent with deterministic and consistent rule applications.
4. A performance evaluation that demonstrates a high connectivity preservation with minimum graph reduction (in real-world provenance graphs, 10% restricted nodes result in 90% graph connectivity).

The rules are classified, according to the tasks they perform, as creation and deletion rules. The creation rules create nodes and edges based on properties from the PROV data model while the deletion rules delete the edges connected to the restricted nodes.

An approach is designed capable of maintaining graph integrity by avoiding false independencies and false dependencies and preserving paths. By preserving relations between the nodes adjacent to the restricted nodes using creation rules, we avoid false independencies, while creating nodes and edges according to the PROV model is important to avoid false dependencies. However, if the node cannot be fully isolated, since deleting some edges may result in broken paths and disconnected graphs, the node will be anonymized so that paths between the nodes of the transformed graph are preserved.

Each pair of simultaneous rule applications in the system are either parallel independent or their matches represent a safe critical pair. This leads to a system which is locally confluent. Furthermore, a layered-based rule application approach shows that the system is terminating. The local confluence and termination indicate that the system is also confluent. This results in deterministic graph transformation system with consistent rule applications.

The system is modelled as a template-based framework using category theory concepts such as functors, diagrams, sliced categories, and natural transformation. A simple construction of the provenance graph transformation system would have resulted in numerous graph transformation rules with some rules sharing similar graph patterns. Instead, those rules are grouped in rule templates using abstract nodes with inheritance and based on category theory constructions such as diagrams to represent the rule templates, functors to define the relationship between rule templates and rule instances, and natural transformation to find the match of a rule in a provenance graph via its template diagram.

The performance of the system is evaluated using several measurements such as

- the amount of preserved information in the transformed graph referred to as graph connectivity;
- the number of transformation steps required to convert a provenance graph;
- average degree;
- normalization degree (to what extent the graph is saturated based on the PROV inference rules. The less the information that can be inferred from the provenance graph the more the graph is saturated).

The evaluation is fulfilled using the measurements above in comparison with the number of restricted nodes in the graph according to the following hypotheses:

- The connectivity measure is highly affected by the number of restricted nodes.
- The number of transformation steps is mostly affected by the number of restricted nodes.
- The degrees of the restricted nodes determine the number of deletion steps required to isolate the restricted nodes.
- The number of the required creation steps is inversely proportional to the graph's normalization degree.

To conduct these experiments, provenance graphs generated by the following three different applications are used:

- AtomicOrchid (Jiang, Fischer, Greenhalgh et al. 2014; Fischer, Jiang and Moran 2012);

- W3C PROV working group email archive (Moreau, Groth, Herman et al. 2011);
- Ride share application (Packer and Moreau 2015).

For each dataset, an appropriate criterion has been used to annotate some nodes of the graphs as restricted based on either privacy or analysis requirements of the application. The system's performance evaluation using different provenance datasets and based on variety of measurements shows high connectivity preservation with minimum graph reduction.

1.6 Publications

The major parts of the research work presented in this dissertation are published in the papers below:

- Jamal Hussein, Luc Moreau and Vladimiro Sassone (2015). "Obscuring Provenance Confidential Information via Graph Transformation". In: *Trust Management IX: 9th IFIP WG 11.11 International Conference, IFIPTM 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*. Vol. 454. IFIP Advances in Information and Communication Technology. Springer International Publishing, pp. 109–125. ISBN: 978-3-319-18491-3. DOI: 10.1007/978-3-319-18491-3_8. URL: http://link.springer.com/chapter/10.1007/978-3-319-18491-3_8
- Jamal Hussein, Vladimiro Sassone and Luc Moreau (2016). "A template-based graph transformation system for the PROV data model". In: *Seventh International Workshop on Graph Computation Models*. URL: http://eprints.soton.ac.uk/397032/

1.7 Thesis Layout

This thesis is organized as follows:

- Chapter 2 (Background): The PROV data model and graph transformation concepts are discussed. The areas of provenance graph transformation are presented. The most issues relevant to graph reduction and sanitization approaches are presented followed by a review of the related work conducted so far in the area of provenance graph rewriting.
- Chapter 3 (Running Examples): In this chapter, two running examples are presented.

- Chapter 4 (Construction of Provenance Transformation Rules): In this chapter, a variety of PROV data model properties such as its inference rules have been used to justify the graph transformation rules in PROV-GTS.
- Chapter 5 (Template-based Graph Transformation System): The notion of rule templates has been formally described and used to group the rules with similar graph shapes.
- Chapter 6 (Termination, Parallelism, and Confluence of PROV-GTS): The properties, such as termination and confluence, of the template-based PROV-GTS have been presented and formally proved in this chapter.
- Chapter 7 (Provenance Graph Integrity and Connectivity): In this chapter, the proofs of graph integrity are provided.
- Chapter 8 (Evaluation and Performance): Evaluation and system's performance are demonstrated in this chapter.
- Chapter 9 (Conclusion and Future Work): The conclusion and future work.

Chapter 2

Background

2.1 Introduction

Provenance is the history of a resource used in variety of domains such as scientific workflow, healthcare, and intelligence (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a). The Open Provenance Model OPM is the first community-driven model aimed at exchanging provenance information between systems (Moreau, Ludäscher, Altintas et al. 2008; Moreau, Clifford, Freire et al. 2011). The OPM succeeded by the W3C standardized PROV data model aimed at exchanging provenance over the Web (Moreau and Missier 2013). Processes such as registering, representing, and evaluating provenance information, which are carried out by provenance-based systems, are important in estimating the originality and quality of an object (Moreau, Clifford, Freire et al. 2011). The vast majority of research works on provenance have been conducted in the fields of database, workflow, and e-science. Provenance allows regenerating scientific results; following changes in curated databases; and building trust for the information over the Semantic Web (Moreau 2010).

There are various ways to represent and store provenance data; graphically as direct typed graphs where nodes act as provenance components and edges represent their relations (Moreau 2010), or literally by using notations such as Turtle, *PROV-N*, or XML. The provenance notation *PROV-N*, which relies on a functional-syntax style, uses predicates and ordered list of terms to represent provenance data (Moreau, Missier, Cheney et al. 2013).

Open provenance model OPM is the model that aimed at exchanging information between systems based on shared provenance model. The OPM is designed during the course of three activities known as Provenance Challenges (Moreau, Freire, Futrelle et al. 2008; Moreau, Clifford, Freire et al. 2011). In the First Provenance Challenge, using a Functional Magnetic Resource Imaging workflow, a set of pre-defined provenance queries were executed (Moreau, Ludäscher, Altintas et al. 2008). The purpose of the Second Provenance Challenge was to set up interoperability between systems by exchanging provenance information. This led to the first version of OPM 1.0 and then version 1.01. The third Provenance Challenge resulted in OPM version 1.1 (Simmhan, Groth and Moreau 2011).

The PROV data model (PROV-DM) is the descendant of OPM model. It aims at exchanging provenance information on the Web. It defines provenance as “a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing” (Moreau and Missier 2013).

Graph transformations is of paramount importance in many areas such as software engineering, distributed systems, and Petri-nets. Regarding graphical representation of provenance information, graph transformation is convenient for creating or deleting certain parts of provenance graphs by using rule-based graph modification. Provenance graph transformations have been mainly used in two broad domains: provenance access control and scientific workflow run provenance. In provenance access control, graph transformations has been used to hide the sensitive parts of the provenance graphs based on policies which restrict access to the sensitive information according to designated users’ privileges (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a). In scientific workflow systems, providing different views via graph rewriting rules is essential to avoid overwhelming users with non-relevant information (Davidson, Khanna, Roy et al. 2011b). In both domains, graph transformations play a key role to minimize the size of graphs by using various techniques such as graph reduction and sanitization by deleting nodes and edges (Chebotko, Chang, Lu et al. 2008), anonymization by replacing a node or group of nodes by an anonymous node (Dey, Zinn and Ludäscher 2012), and grouping by abstracting sets of nodes (Danger, Curcin, Missier et al. 2015).

It is important to ensure that graph transformation systems do not delete extra information or add false dependencies to the graph. Relation preservation may be used to maintain the dependencies between the remaining part of the graph and ensuring that only the relevant information has been removed. However, this preservation either has not been supported or it cannot be justified by the properties of the provenance model used in constructing the provenance rewriting systems found in the literature (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a). So, the previously proposed approaches either remove from the graph additional and non-relevant dependencies (Blaustein, Chapman, Seligman et al. 2011) or add false dependencies (Chen, Edwards, Nelson et al. 2015).

In this chapter we introduce variety of requirements of provenance graph transformations as issues and we show how these properties are preserved by some and violated by others. These requirements include: No false dependency, No false independency, Path preservation, No false paths, No excessiveness, and graph validity preservation.

The definition of PROV model is presented in Section 2.2. Some of the PROV model inference rules are provided in Subsection 2.2.2. In Section 2.3 the concept of algebraic graph transformation is defined. Section 2.4 demonstrates the areas of provenance graph transformation. The provenance graph transformation techniques are provided in Section 2.5. The issues regarding provenance graph transformation are investigated in Section 2.6. Section 2.7 presents a set of most relevant approaches found in the literature.

2.2 The PROV Data Model

PROV data model is a W3C recommendation and the descendant of OPM model aimed at exchanging provenance information on the Web. Provenance can be used for various reasons, such as specifying the ownership of an object, understanding how data were gathered so it can be used efficiently, determining the originality of the information so it can be trusted, capturing the steps of obtaining a scientific result so it can be reproduced. PROV information can be presented in different perspectives, first perspective is agent-centered provenance, which specifies the people or organizations involved in processing the information. The second perspective is object-centered provenance which focuses on tracking the origin of the information. The last perspective is process-centered provenance which captured the steps taken to produce the information (Gil, Miles, Belhajjame et al. 2013).

The PROV Ontology *PROV-O* consists of a set of classes, properties, and restrictions which provides the foundation to represent and interchange provenance information in different domains (Lebo, Sahoo, McGuinness et al. 2013). In the description of PROV components two different representations have been defined: graphical representation using direct typed graphs and literal representations like *PROV-N* (Moreau, Missier, Cheney et al. 2013). In graphical representation, the diagrammatic shapes oval, rectangle, and pentagon are used to represent entities, activities, and agents respectively. Also there are rectangle-shaped data nodes use to represent nodes' and edges' attributes. *PROV-N* is a standard machine-processable and human-readable notation that is part of the PROV data model and is composed of a predicate name and an ordered list of terms in functional style syntax (Moreau, Missier, Cheney et al. 2013).

2.2.1 PROV data model components

The PROV model consists of a set of terms; each term has an identifier and additional attributes that further describe what the identifier denotes. Figure 2.1 illustrates the basic terms of the core PROV data model which consists of three node types - entity, activity, and agent - and seven edges which are wasDerivedFrom (der), wasGeneratedBy (gen), used (use), wasInformedBy (info), wasAttributedTo (attr), wasAssociatedWith (assoc), and actedOnBehalfOf (del). We use the abbreviations of these edges throughout this thesis to refer to them.

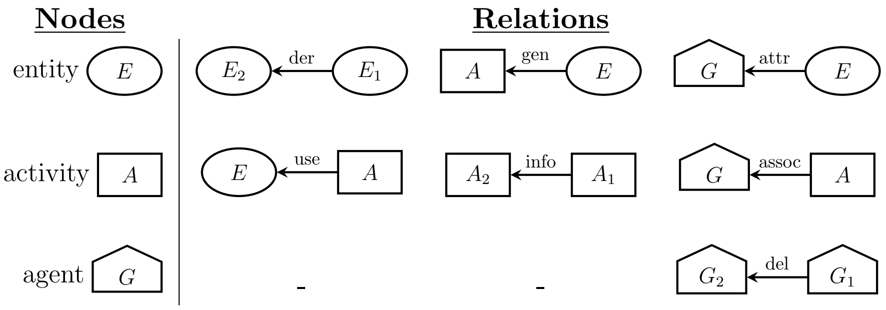


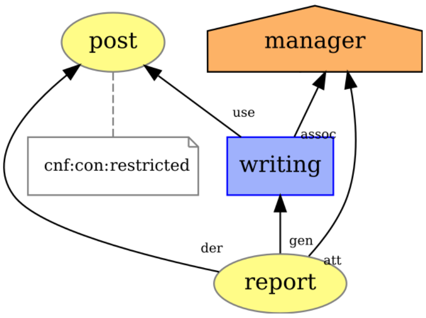
FIGURE 2.1: Core PROV data model terms which consists of three nodes (*entity*, *activity* and *agent*) and seven edge types

Each PROV model node has an identifier and a list of additional elements, here is an example in *PROV-N* format with the graph representation to the right:

```
document
prefix ex <http://example/>
prefix cnf <cnf:>

entity(ex:report)
entity(ex:post, [cnf:con="restricted"])
activity(ex:writing, -, -)
agent(ex:manager)

wasDerivedFrom(ex:report, ex:post)
wasGeneratedBy(ex:report, ex:writing,-)
used(ex:writing, ex:post, -)
wasAttributedTo(ex:report, ex:manager)
wasAssociatedWith(ex:writing, ex:manager, -)
endDocument
```



The two keywords: `document` and `endDocument` indicate the beginning and the end of *PROV-N* documents. The `prefix` keyword is used to declare two prefixes: `ex` and `cnf` followed by four nodes: two entities, one activity, and one agent. These nodes are connected via a number of edges. Each one of these terms has a number of parameter. For example, regarding the second entity, the first parameter is its identifier (`ex:post`) and it is always required. The second represents the list of attributes which contains only one attribute `con` with the string `"restricted"` as its value. Using attributes is optional and there may be more than one attribute separated by commas between the brackets.

2.2.2 PROV Model Inference Rules

The construction of the graph transformation rules of the proposed rewriting system (PROV-GTS), depends on properties from PROV data model such as its inference rules, which we provide a brief description of some of them in this section. Relations between nodes in PROV model can be summarized or expanded using various inference rules. A new relation between two PROV components can be inferred from the existing relations. The new inferred relations may involve new components (nodes and/or edges).

In Figure 2.2, the following PROV data model inferences are presented (Cheney, Missier, Moreau et al. 2013):

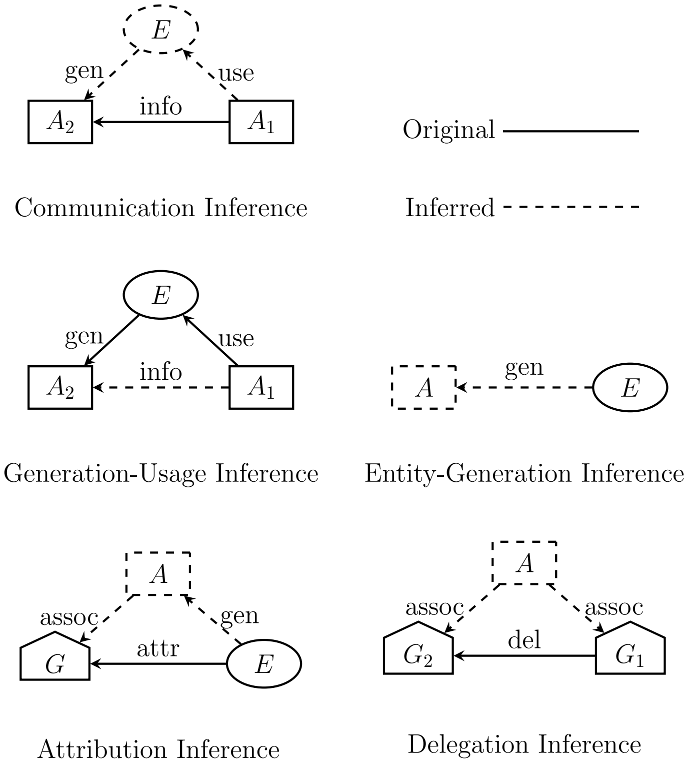


FIGURE 2.2: The PROV data model inference rules

1. Communication Inference: The existence of an entity generated by an activity and used by another can be implied from their communication (*info*) relation (Cheney, Missier, Moreau et al. 2013, Inference 5).

2. Generation-Usage Inference: The communication between two activities can be inferred if there exists an entity generated by one of the activities and used by the other (Cheney, Missier, Moreau et al. 2013, Inference 6).

3. Entity-Generation Inference: The existence of an entity implies the existence of the activity that generated it (Cheney, Missier, Moreau et al. 2013, Inference 7).

4. Attribution Inference: 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 (Cheney, Missier, Moreau et al. 2013, Inference 13).

5. Delegation Inference: The delegation relation (*del*) between two agents in the context of some activity implies that both agents are associated with that activity (Cheney, Missier, Moreau et al. 2013, Inference 14).

2.3 Algebraic Graph Transformation

Graph transformation acquired a remarkable importance over other rewriting techniques, for example using strings (Brill 1995; Satta and Henderson 1997) and trees (Keller, Perkins, Payton et al. 1984; Zanibbi, Blostein and Cordy 2002). It provides well-established analysis approaches for termination, confluence and parallelism (Ehrig, Ehrig, Prange et al. 2006b). In addition, several tools with different degree of expressive power have been developed for implementing graph transformation (Taentzer 2004; Rensink, Boneva, Kastenberg et al. 2010; Varró, Asztalos, Bisztray et al. 2008). Graph transformation techniques are playing an essential role in many applied areas, such as software engineering (Baresi and Heckel 2002), distributed systems (Taentzer, Goedicke and Meyer 2000), model transformation (Mens and Van Gorp 2006), and Petri-nets (Ehrig, Engels, Kreowski et al. 1999). Intuitively, graphs are a very powerful way to represent complicated situation naturally. Several approaches have been defined to model graph transformation, but the most important one is the algebraic approach. Algebraic graph transformation approaches formulate the graph transformation step by two gluing diagrams (pushouts) (Ehrig, Pfender and Schneider 1973) or one gluing diagram (Broek 1991).

Algebraic graph transformation approaches are based upon a category theory construction known as pushout which is used to model the concept of glueing of graphs. There exist two algebraic approaches, the *double-pushout (DPO)* approach (Ehrig, Pfender and Schneider 1973) and the *single-pushout (SPO)* approach (Löwe and Ehrig 1991). The *double-pushout (DPO)* approach models graph transformation rule using two gluing diagrams (i.e. pushouts). In contrast, the *single-pushout (SPO)* approach uses one gluing diagram in category of graphs and partial graph morphisms. Graph transformation systems consist of a set of graph transformation rules. In its simplest form, based on the *single-pushout (SPO)* approach, A graph transformation rule consists of a *left-hand side(*LHS*)* and a *right-hand side (*RHS*)*. It is possible to have several matches of the left-hand side in a graph but only one of them is chosen. The rule is applied on graph G, referred to as the original graph, by replacing the match of LHS in G by RHS resulting in a new graph H (Ehrig, Heckel, Korff et al. 1997), referred to as the transformed graph. The graph transformation rules can be used to delete or create nodes and edges, that is deletion and creation rules respectively.

The *single-pushout (SPO)* approach modifies a graph according to a rule of the form ($r : L \rightarrow R$), where L is the left-hand side (LHS) and R is the right-hand side (RHS) (Löwe

1993). The *SPO* rules are applied by finding a match of L in the graph G and then deleting $L \backslash R$ and creating $R \backslash L$ by performing the following steps:

1. Find a match of L in G.
2. Ensure that G satisfied all application conditions of r.
3. Remove from G the part of L that is not in R and create the part of R that is not in L by gluing it producing graph H.

$$\begin{tikzcd}
L \arrow[r] \arrow[d, "\text{m}"'] \arrow[dr, phantom, "(1)"] & R \arrow[d] \\
G \arrow[r] & H
\end{tikzcd}$$

Single Pushout Approach

The algebraic approach has been used in applications such as software engineering where simple labelled graphs are not adequate to implement modelling techniques like UML. Therefore, the classical representation of graphs has been expanded to support typed and attributed graphs in the categories and systems of high-level structures HLR (Ehrig, Prange and Taentzer 2004).

2.4 Areas of Provenance Graph Transformation

2.4.1 Provenance Access Control

In the digital world, provenance reveals information about the origin of a piece of data involving all the relations that lead to that piece of data. The sensitivity of provenance data may be more or less than the sensitivity of the data itself. The current access control systems focus on the data items only, while in provenance the focus must be on the relationships between these items as well. These different sensitivities require two different security policies, one for the data and the other for the provenance (Braun, Shinnar and Seltzer 2008). Graph transformations are used to apply the access control policies and prevent users from accessing sensitive provenance information.

2.4.2 Scientific Workflows

Real-world provenance systems consist of a huge amount of provenance information indicated by nodes and edges. Presenting provenance information as a static graph makes it difficult for users to catch related information and explore the relations between nodes. Providing specialized views of provenance graphs by hiding the irrelevant nodes and edges using provenance graph transformation is important for scientific workflow schemes (Dey, Zinn and Ludäscher 2011b; Anand, Bowers and Ludäscher 2012). Scientific workflows use provenance to formally describe the representation, computation, and analysis of scientific processes such that scientific results can be monitored, interpreted, and reproduced, and problems can be easily diagnosed (Chebotko, Chang, Lu et al. 2008).

2.5 Provenance Graph Reduction and Sanitization Techniques

There are many techniques that have been used in building provenance graph transformation systems such as graph reduction and sanitization, abstraction, and anonymization. Each graph transformation technique may violate one or more of the requirements of provenance graph transformation. Here we discuss these techniques:

2.5.1 Reduction and sanitization

The easiest way for implementing graph transformation is by deleting from the provenance graph the nodes and edges that reveal private and confidential information. However, naive deletion of nodes and edges may affects the integrity of the provenance graph by causing false independency. So, sometimes it is important to preserve relations, which may consist of creating nodes and edges, to avoid such graph integrity violation.

2.5.2 Abstraction and node grouping

The abstraction approach replaces a set of nodes with an abstract node. Usually, the set of the abstracted nodes represents a sub-graph with nodes that expose provenance sensitive information. Replacing a set of nodes by one single node may result in many issues such as invalid cycles, repeated edges and false dependency (Cheney and Perera 2014). To avoid these issues, the approaches that depend on this technique ensures that the transformed graph is valid by performing one of these:

- Redirecting and relabelling edges (Cadenhead, Khadilkar, Kantarcioglu et al. 2011b).
- Expanding the group of nodes to contains extra nodes (Chen, Edwards, Nelson et al. 2015).

2.5.3 Anonymization

Anonymization is the process of replacing a node with an anonymous or fictitious node hiding the identity and the attributes of the original node (Dey, Zinn and Ludäscher 2011a; Dey, Zinn and Ludäscher 2011b; Dey, Zinn and Ludäscher 2012; Dey and Ludäscher 2013).

2.6 Issues with Provenance Graph Transformation

Removing confidential and irrelevant parts from provenance graphs or creating new graph elements may affect the integrity of provenance graphs by causing false independency and false dependency (Dey, Zinn and Ludäscher 2011a). In addition, hiding certain parts of the graph may result in breaking paths between the remaining nodes in the transformed graph or creating false paths as a result of node abstraction and grouping. Moreover, in order to avoid violating graph integrity and generate valid provenance graphs some approaches require expanding abstracted subgraphs to contain new nodes that were not part of the sensitive nodes which result in hiding more information than what is required to be abstracted.

In the following, a set of preservation properties are provided which must be taken into account as requirements of provenance graph transformation systems:

No false dependency False dependency occurs if the items created in the transformed graph cannot be justified from the original graph based on the properties of the OPM or PROV model. This can be avoided by restricting the creation of nodes and edges to the ones that can be inferred from the original graph based on the OPM or PROV model properties such as the inference rules which have been described in Section 2.2.2.

No false independency 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. So, we need to consider that the graph reduction regarding a single node or edge will not affect the relations between the other nodes in the graph.

Path preservation The integrity and usefulness of provenance graphs require preserving the paths between non-sensitive nodes in the transformed graphs. To ensure integrity of provenance information, it is crucial for the various paths in the graph to be semantically correct. In addition, it is important when rewriting provenance graphs to preserve the same semantic paths, although with less information. So, the paths that we want to preserve is not the normal graph paths. They represent the line-ages of artifacts, which must still be reachable even after hiding some of the sensitive nodes and edges along the paths. This preservation can be done by preventing false independencies and avoiding edge deletions regarding certain restricted nodes when they affects the paths between the other nodes in the original graph.

To recap, path preservation requires avoiding the following:

- false independency: Path preservation is achieved by avoiding false independencies along paths.
- deleting edges that break paths between the nodes that are adjacent to the source and the target of the deleted edge. This can be avoided by anonymizing those restricted nodes rather than deleting their connected edges.

No false paths It is crucial to ensure that there are no false paths between the nodes in a transformed graph as a result of graph rewriting. False paths can be avoided by preventing false dependency. In addition, it is important to ensure that replacing a set of nodes by a single node is performed in a way that it will not result in new paths that cannot be justified by the original graph. A path is a false path if there exists at least one false dependency along the path.

No excessiveness In order to avoid false independency and preserve paths as part of integrity and validity preservation of provenance information, it is important for any graph transformation system to produce minimal provenance graphs that preserve the graph integrity. In order to establish this, the system must avoid hiding or abstracting nodes and edges other than what have been annotated as sensitive and confidential.

Graph validity preservation The transformation approach must provide a valid transformed graph over the same type graph that the graph transformation system is based upon such as OPM or PROV data model.

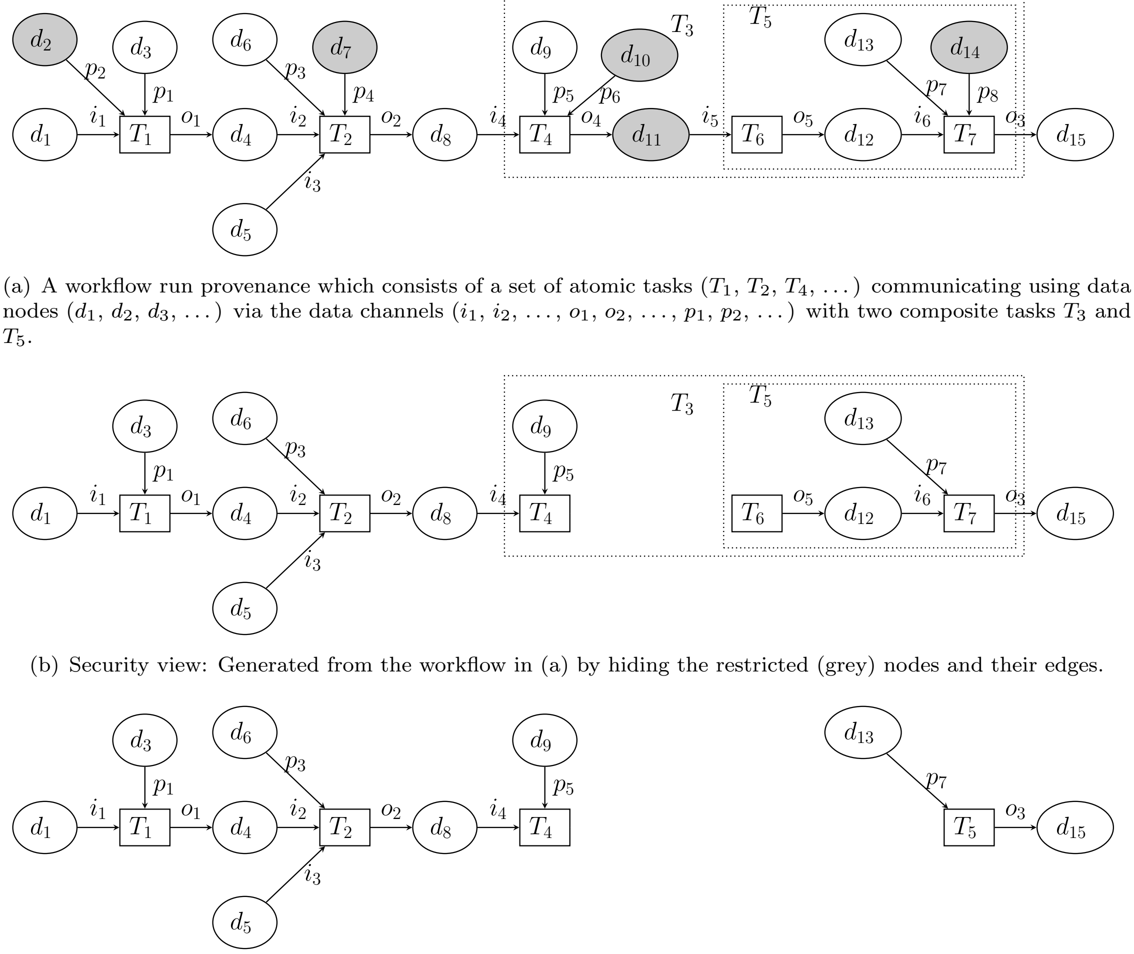
2.7 Reduction and Sanitization Approaches

Several graph reduction and sanitization approaches which are capable of obscuring private provenance information by deleting or abstracting set of nodes have been proposed in the literature (Cheney and Perera 2014). However, no one of these approaches has been fully built based on the fundamental theory of algebraic graph transformation (Ehrig, Ehrig, Prange et al. 2006b) or adhesive high-level replacement (HLR) systems (Sassone and Sobocinski 2004), which makes it difficult to look at various properties such as concurrency, parallelism, confluence, and termination of graph rewriting systems (Ehrig, Ehrig, Prange et al. 2006a).

Although provenance graph transformations have not yet been addressed in detail by the previously proposed approaches, we present the following research papers which used different techniques to design the rewriting approaches.

Workflow (Chebotko, Chang, Lu et al. 2008; Chebotko, Lu, Chang et al. 2010) These papers propose formalisations of the notions of security views and abstraction views. A security view of a worlkflow run provenance is the provenance view restricted to all the information that users are authorised to access, while abstraction views enable users to focus on certain parts of provenance information using composite tasks and subworkflows. The abstraction views and security views are integrated in a framework to enable users to examine provenance at different abstraction levels while preserving their security policies. An example of workflow run provenance, a security view, and a secure abstraction view are shown in Figure 2.3. The example represents an intragenomic recombination analysis scientific workflow, as dipected in Figure 2.3(a), and consists of a set of tasks (T_1, T_2, T_3, $\ldots$). The tasks are communicating with each other by passing data (d_1, d_2, d_3, $\ldots$) via the data channels (i_1, i_2, $\ldots$, o_1, o_2, $\ldots$, p_1, p_2, $\ldots$). This workflow is hierarchical and consists of two kinds of tasks; composite and atomic, for example the composite task T_5 consists of the two atomic tasks T_6 and T_7, as well as some data nodes and data channels. The tasks and data nodes are analogous to activities and entities of the OPM and PROV data model, while the data channels represent the edges.

Security views are generated from workflow run provenance by hiding the sensitive edges and/or nodes, for example the sensitive data nodes d_2, d_7, d_{10}, d_{11}, and d_{14} have been hidden as illustrated in Figure 2.3(b). Abstraction views are constructed from workflow run provenance by replacing non-relevant atomic tasks along with the data channels connecting them by their composite task, for example the tasks T_6, T_7, and data node d_{12} have been replaced by the composite task T_5 as shown in Figure 2.3(c). This approach causes false independency. For example in the security view (Figure 2.3(b)), the relation between the activities T_4 and T_6 is broken as a result of deleting d_{11}, and consequently breaking any path that consists of d_{11}. In addition, replacing nodes with composite tasks, as in the abstraction view of Figure 2.3(c), may cause more nodes to be abstracted than the sensitive nodes, i.e. excessiveness. For example, only d_{14} is sensitive among the nodes of the composite task T_5 but d_{12} was also deleted.



(a) A workflow run provenance which consists of a set of atomic tasks (T_1, T_2, T_4, ...) communicating using data nodes (d_1, d_2, d_3, ...) via the data channels (i_1, i_2, ..., o_1, o_2, ..., p_1, p_2, ...) with two composite tasks T_3 and T_5.

(b) Security view: Generated from the workflow in (a) by hiding the restricted (grey) nodes and their edges.

(c) Abstraction view: Constructed from (a) by hiding the restricted nodes and replacing the non-relevant atomic tasks (T_6 and T_7) and the data node d_{12} by the composite task T_5.

FIGURE 2.3: Security and abstraction views of a scientific workflow run provenance which represents an intragenomic recombination analysis (Chebotko, Chang, Lu et al. 2008)

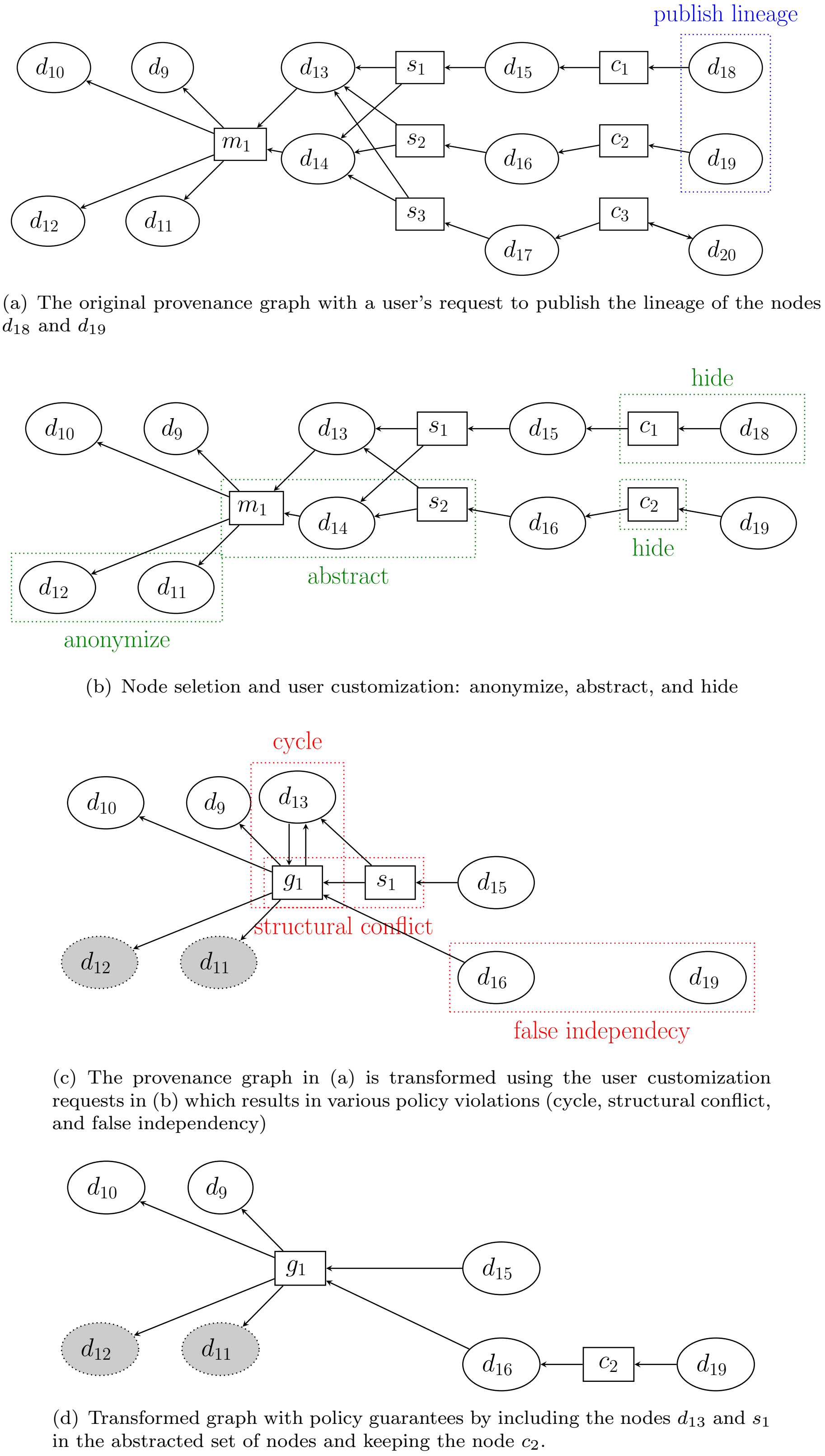
ProPub (Dey, Zinn and Ludäscher 2011a; Dey, Zinn and Ludäscher 2011b; Dey, Zinn and Ludäscher 2012; Dey and Ludäscher 2013)

These papers propose Provenance Publisher ProPub which specifies the provenance policies that must be obeyed when publishing provenance. The approach repairs any violation and reconcile conflicts to these policies by rejecting any inconsistent user requests. To do so, it checks and enforces integrity constraints based on a given conflict resolution strategy. This provenance model is used OPM graph notations with a structural constraint that the graph is bipartite, i.e. no edges between the same type of nodes. The model is based on user customization requests, such as anonymizing, abstracting or hiding certain parts of provenance graphs.

In Figure 2.4(a) a provenance graph with a user's request to publish the lineage of two nodes (d_{18} and d_{19}) is presented. The user customization requests, i.e. anonymizing, abstracting and hiding certain part of the graph, are provided in Figure 2.4(b). Applying these customization requests on the provenance graph results in a graph with various policy violations such as creating cycles, structural conflict and false independency as shown in Figure 2.4(c). Abstracting the nodes m_1, d_{14} and s_2 by replacing them with the node g_1 results in an invalid cycle, since the node d_{13} has incoming and outgoing edges to the set of abstracted nodes. In addition, the abstraction causes an invalid edge between the nodes s_1 and g_1 (based on the structural constraint mentioned above). To tackle these issues, the system includes the nodes d_{13} and s_1 in the abstracted set of nodes. To resolve the problem of false independency between d_{19} and d_{16} the system keeps the node c_2 which connects between the two nodes. Expanding the set of abstracted nodes and keeping some nodes result in a graph with policy guarantees. This model avoids false dependency by creating non-functional nodes between the nodes adjacent to the set of abstracted nodes and preserves paths by keeping some nodes that are supposed to be deleted (Dey, Zinn and Ludäscher 2011b). However, expanding the set of nodes to be abstracted results in hiding extra information, i.e excessiveness.

Framework (Cadenhead, Kantarcioglu and Thuraisingham 2011)

The paper (Cadenhead, Kantarcioglu and Thuraisingham 2011) addresses how to apply access control policies over a provenance graph and how to transform it to satisfy a set of redaction policies by providing a unified framework which allows protecting and sharing provenance information. The approach provides policies as graph operations over provenance graphs using regular expression queries. Graph rewriting rules are defined consisting of LHS, RHS, and the embedding element with two sub-elements; pre- and post-processing conditions which describe respectively how LHS and RHS are connected to the provenance graph. These conditions are necessary to that this graph rewriting approach always returns a valid (OPM) provenance graph, i.e. no invalid cycles



(a) The original provenance graph with a user's request to publish the lineage of the nodes d_{18} and d_{19}

(b) Node seletion and user customization: anonymize, abstract, and hide

(c) The provenance graph in (a) is transformed using the user customization requests in (b) which results in various policy violations (cycle, structural conflict, and false independency)

(d) Transformed graph with policy guarantees by including the nodes d_{13} and s_1 in the abstracted set of nodes and keeping the node c_2.

FIGURE 2.4: Graph transformation based on various user customization requests (Dey and Ludäscher 2013)

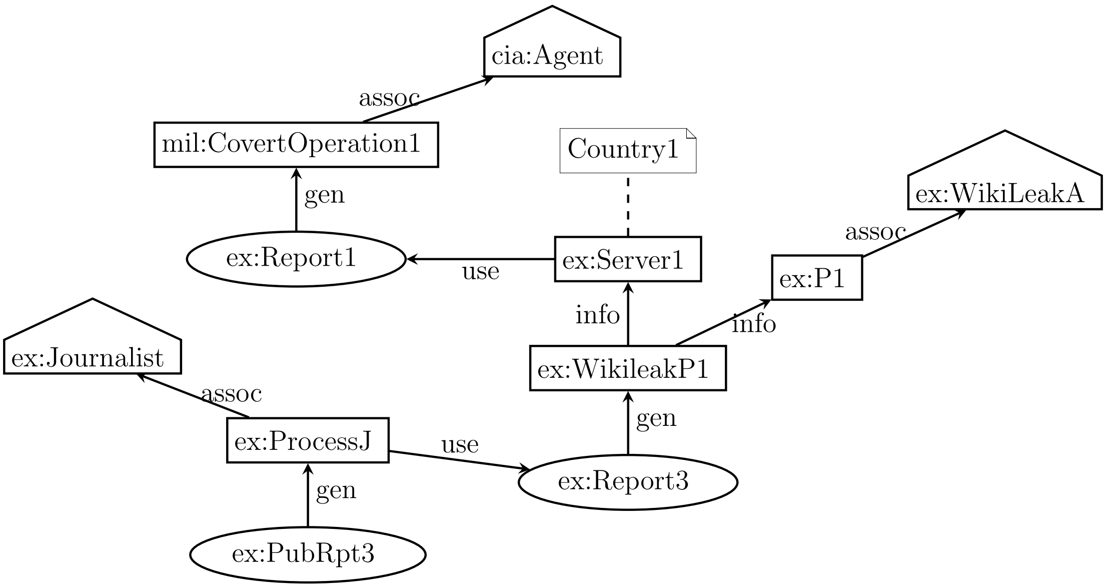


FIGURE 2.5: A provenance graph of an intelligence report (Cadenhead, Kantarcioglu and Thuraisingham 2011)

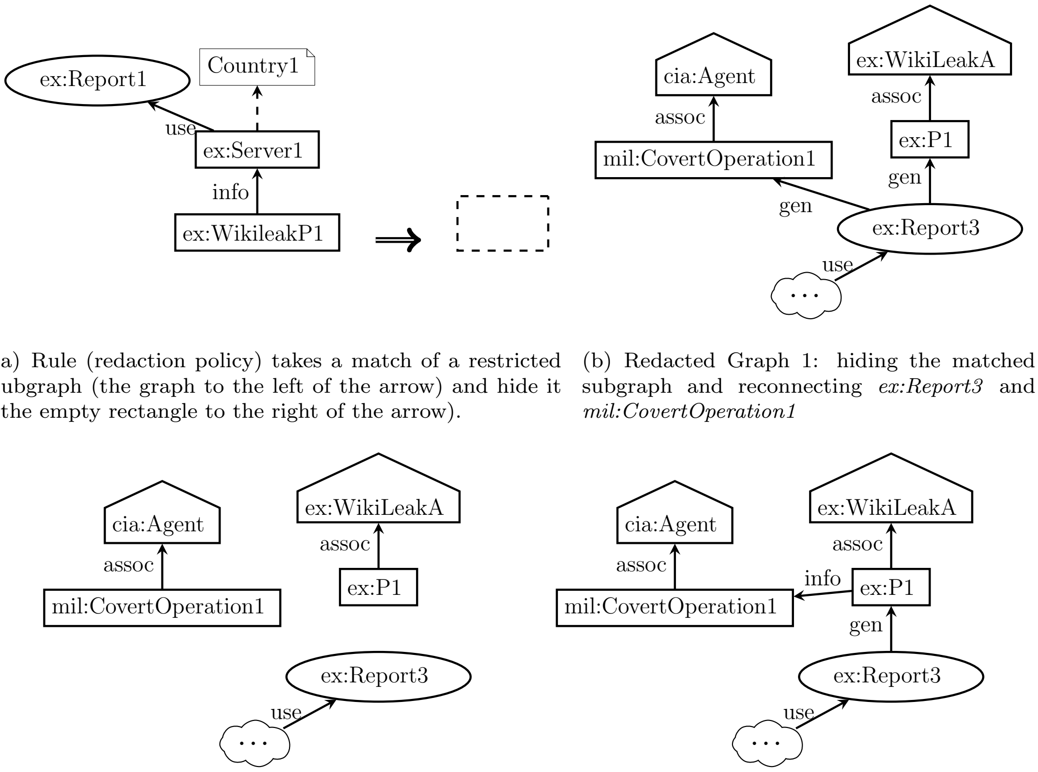
or false dependencies. Figure 2.5 shows an example which represents the provenance of an intelligence report.

Applying the same rewriting rule but with different embedding instructions results in different transformed graphs, as shown in Figure 2.6. The rule (redaction policy) shown in Figure 2.6(a), which matches the path that consists of the nodes *ex:WikiLeakP1*, *ex:Server1* and *ex:Report1*, is applied on the above graph three times, each time with a different embedding instruction resulted in three different redacted graphs.

However, this approach, when applied on the graph in Figure 2.5, causes false independency and broken paths as shown in Figure 2.6(c). It also results in false dependency and false paths after connecting *ex:Report3* and *mil:CovertOperation1*, and *ex:P1* and *mil:CovertOperation1*, as shown in Figure 2.6(b) and Figure 2.6(d), respectively.

Language (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a; Cadenhead, Khadilkar, Kantarcioglu et al. 2011b)

In (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a) an access control language to support provenance is proposed by defining a policy language which allows specification of policies over data items and their relationships in a provenance graph. This policy language uses regular expression to include the relationships between data items in the provenance graph which can be used to verify the quality of the data items and as access control mechanism. In this paper, provenance acts as an access control language and as an integrity mechanism for giving access to the data. The provenance graph in Figure 2.7 represents three medical stages which have been taken from a record of a patient. These medical stages are checkup, follow-up, and a surgery. In each stage, the patient record is



(a) Rule (redaction policy) takes a match of a restricted subgraph (the graph to the left of the arrow) and hide it (the empty rectangle to the right of the arrow).

(b) Redacted Graph 1: hiding the matched subgraph and reconnecting *ex:Report3* and *mil:CovertOperation1*

(c) Redacted Graph 2: hiding the matched subgraph without reconnecting the remaining nodes

(d) Redacted Graph 3: hiding the matched subgraph and reconnecting *ex:P1* and *mil:CovertOperation1*

FIGURE 2.6: Graph transformations: three redacted graphs in (b), (c) and (d) are generated based on the graph redaction policy in (a) which has been applied on the graph of Figure 2.5 (Cadenhead, Kantarcioglu and Thuraisingham 2011)

updated by deriving a new record from the old one. The patients records are represented by the entities (or artifacts in OPM terminology) $med : Doc1\_i$ for $i = 1 \dots 4$ which are updated by physicians or a surgeon represented by *agent* nodes in the graph. An access control policy as a regular expression is embedded to SPARQL provenance queries over provenance graphs which are used to retrieve paths after restricting access to some resources in the graph. Common types of queries such as *why-provenance*, which allows us to specify all the resources reachable from a particular node, are used. For example, in Figure 2.8 a *why-provenance* query is presented which retrieves the paths that led to generating the patient's record $med : Doc1\_2$.

A provenance query was entered by a user, converted to a regular expression which was evaluated against a specified access control policy resulting in a reduced query which was translated to a provenance graph. The approach deletes all nodes and paths that are in the graph and come from the original query but are not in the graph comes from the reduced query. The access control policy has been applied on the query of Figure 2.8 and results in the reduced query and the reduced transformed graph that are shown in Figure 2.9.

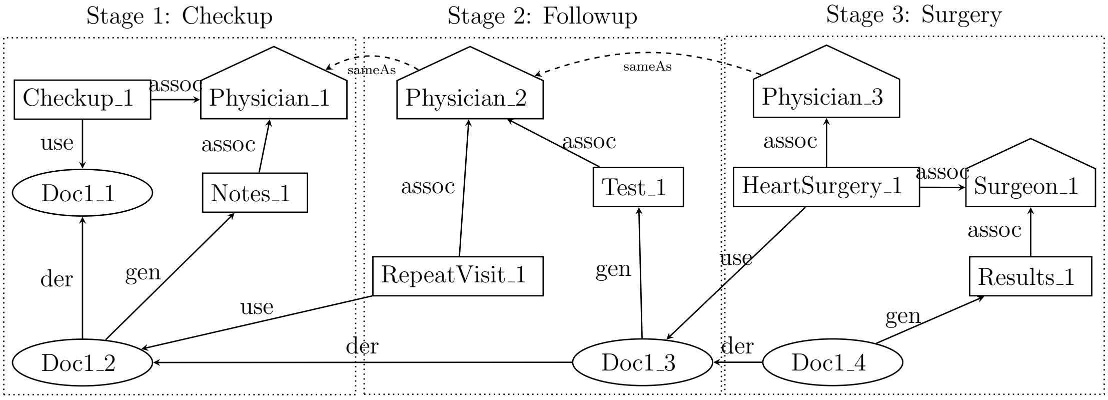
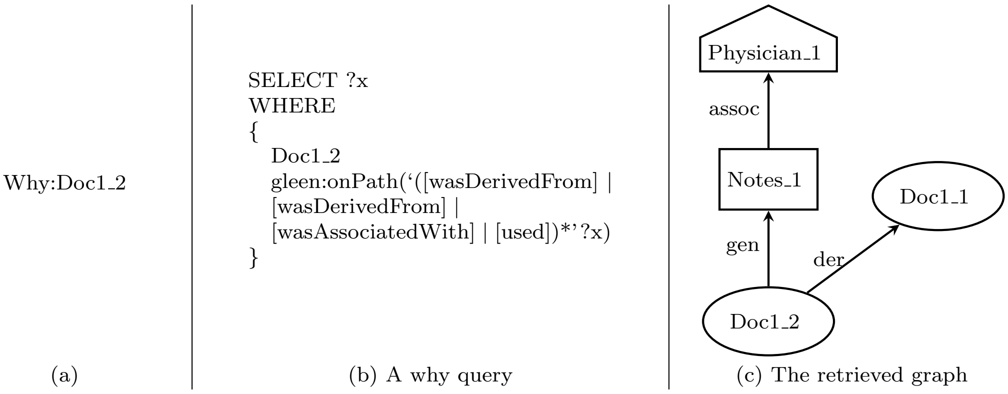


FIGURE 2.7: A provenance graph which represents three medical stages from a record of a patient (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a)



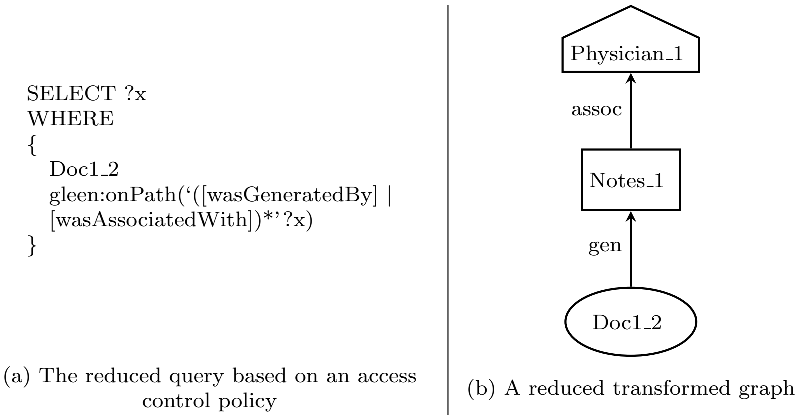
```
SELECT ?x
WHERE
{
   Doc1_2
   gleen:onPath('([wasDerivedFrom] |
   [wasDerivedFrom] |
   [wasAssociatedWith] | [used])*' ?x)
}
```

(a)

(b) A why query

(c) The retrieved graph

FIGURE 2.8: A why query which retrieves the paths reachable from the node *Doc1_2* (Cadenhead, Khadilkar, Kantarcioglu et al. 2011a)



```
SELECT ?x
WHERE
{
   Doc1_2
   gleen:onPath('([wasGeneratedBy] |
   [wasAssociatedWith])*' ?x)
}
```

(a) The reduced query based on an access control policy

(b) A reduced transformed graph

FIGURE 2.9: A resource protected by a policy (Cadenhead, Khadilkar, Kantarcioglu et al. 2011b)

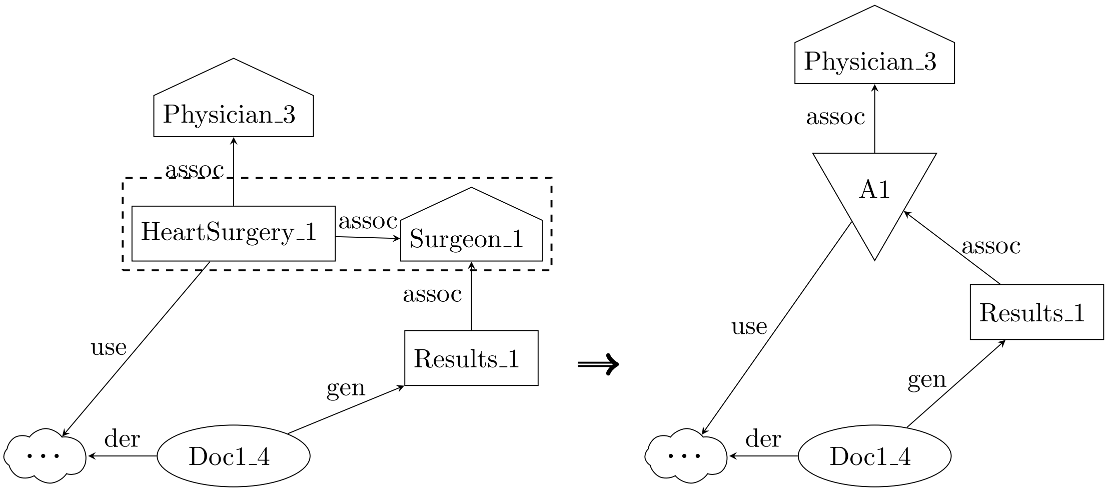
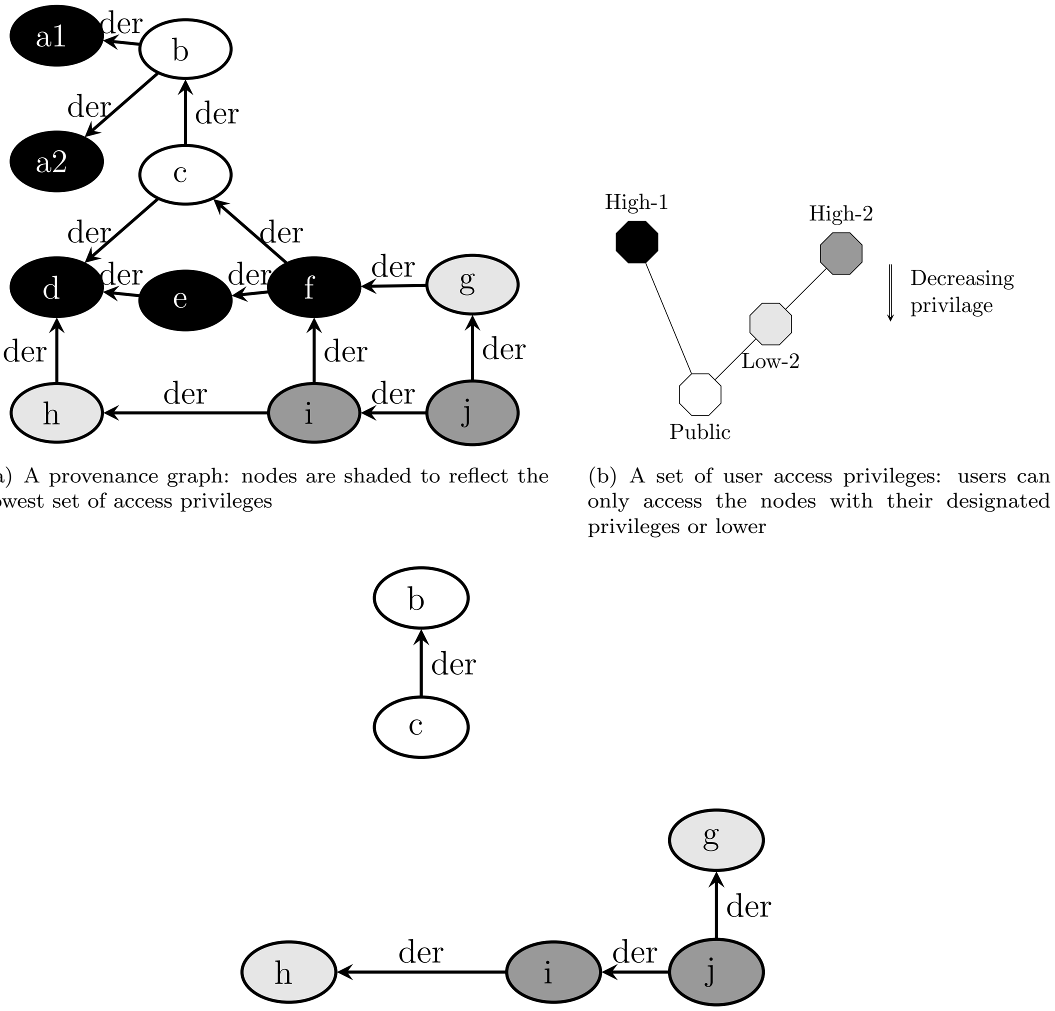


Figure 2.10: Edge contraction: replacing the *wasAssociatedWith* and the nodes *HeartSurgery_1* and *Surgeon_1* by the abstract node *A1* (Cadenhead, Khadilkar, Kantarcioglu et al. 2011b)

In (Cadenhead, Khadilkar, Kantarcioglu et al. 2011b) the above model is expanded by proposing a graph grammar approach for rewriting provenance graphs using reduction technique. This system returns a valid provenance graph using four graph operations, edge contraction, vertex contraction, path contraction, and node relabelling. Edge contraction replaces two nodes and an edge between them with a new node and serves as a basis for the other two contractions: vertex and path. To maintain the validity of provenance graphs, the approach deletes the loops that resulted from edge contraction. Vertex contraction is the operation of replacing two arbitrary vertices and a shared edge between them with a new node. Path contraction is the operation of replacing a set of edges in a path with a single edge connecting between the two end-points of the path. Node relabelling is the process of replacing a labelled node with a node with a new label.

Figure 2.10 represents an example of edge contraction where the edge *wasAssocitedWith (assoc)* and its two end points (*HeartSurgery_1* and *Surgeon_1*) are replaced by an abstract node (▽). In OPM's point of view, the transformed graph is not valid due to the new abstract node type and the edges connected to it. As a consequence, the model cannot prevent false dependencies or false paths.

Surrogate (Blaustein, Chapman, Seligman et al. 2011)
The approach in (Blaustein, Chapman, Seligman et al. 2011) describes techniques for interposing alternate, less sensitive nodes and edges, called surrogate nodes and edges, as needed to protect sensitive graph components while maximizing graph connectivity and giving users as much information as possible. The approach preserves as much connectivity as possible by creating a protected account G' from the original graph G. Utility and opacity measures are used to analyse the protecting account, i.e. the transformed graph. The utility measure determines the amount of information in the



(a) A provenance graph: nodes are shaded to reflect the lowest set of access privileges

(b) A set of user access privileges: users can only access the nodes with their designated privileges or lower

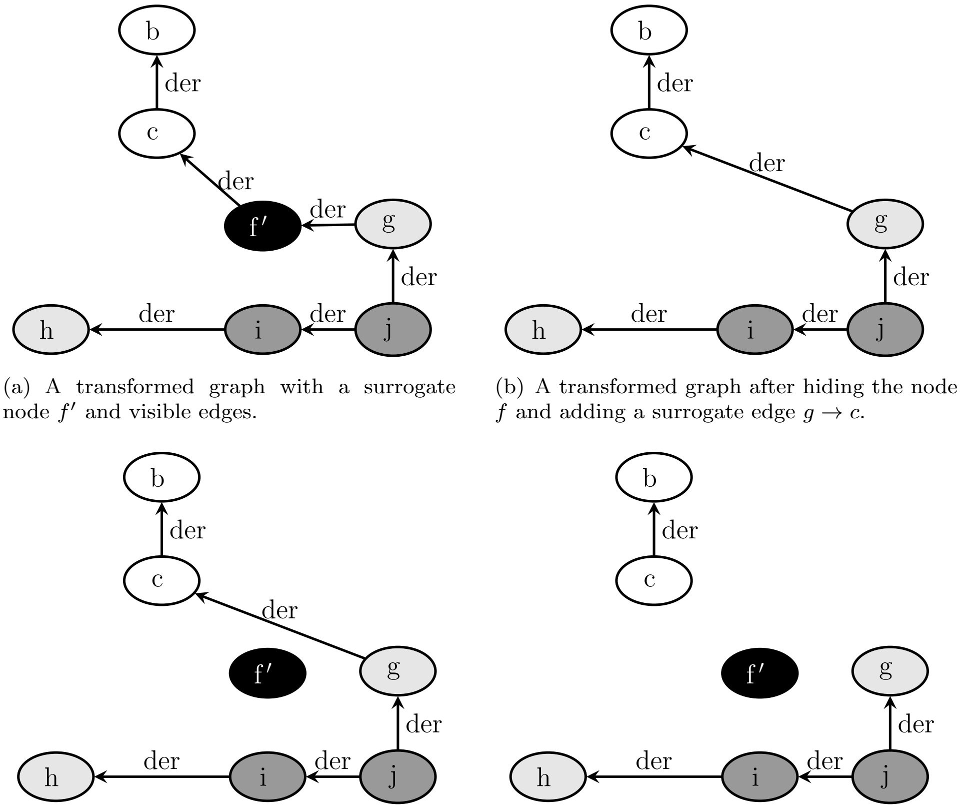
(c) A naively transformed graph available to a user with High-2 privileges

FIGURE 2.11: A sample graph, user privileges, and naively protected account (Blaustein, Chapman, Seligman et al. 2011)

protected account in comparison with the original graph, while the opacity measure indicates the likelihood that an attacker can recreate the topology of the original graph from the protected account.

In Figure 2.11, a sample graph is shown where nodes are shaded to reflect the lowest set of access privileges required (a), sample set of user privileges; each nickname (High-1, etc.) summarizes a predicate expressing privileges (b), a naively protected account G' available to a user with High-2 privileges (c).

Surrogate nodes are less sensitive nodes that are created by omitting or changing features of the original node for the users lacking access privileges to that node. So, each node in the protected graph (the transformed graph) is same as a node in the original graph or it is defined as a surrogate of it.
Figure 2.12 shows four examples of protected accounts, in (a) with a surrogate node f'



(a) A transformed graph with a surrogate node f' and visible edges.

(b) A transformed graph after hiding the node f and adding a surrogate edge $g \rightarrow c$.

(c) A transformed graph with a surrogate node f' after hiding the connected edges.

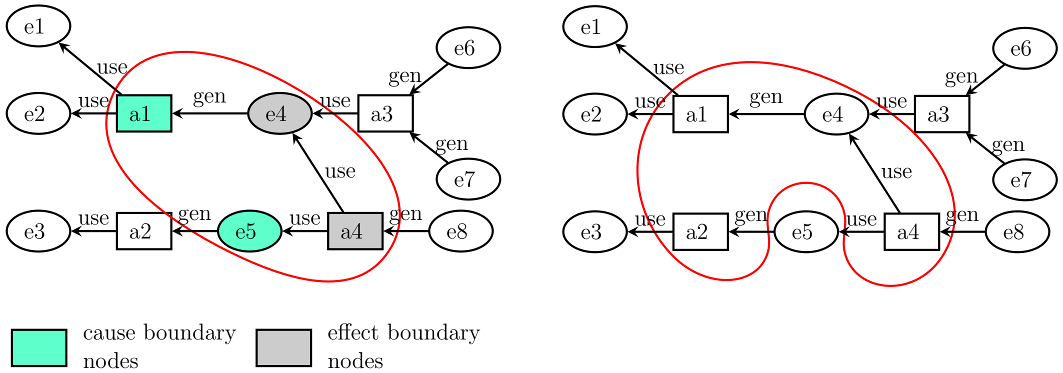
(d) A transformed graph with a surrogate node f' and a surrogate edge $g \rightarrow c$

FIGURE 2.12: Four different protected accounts (Blaustein, Chapman, Seligman et al. 2011).

and visible edges. In the graph in (b), the node f is hidden with a surrogate edge $g \rightarrow c$. While (c) provides a surrogate node f' and a surrogate edge $g \rightarrow c$ and the protected account in (d) provides a graph with surrogate node f' and hidden edges. The graphs in (b) and (c) show how a false dependency ensues by creating an edge between the nodes g and c. It is clear from (d) that the paths and dependencies are not preserved by this algorithm.

ProvAbs (Missier 2013; Missier, Bryans, Gamble et al. 2013; Missier, Bryans, Gamble et al. 2015)

In (Missier 2013; Missier, Bryans, Gamble et al. 2013; Missier, Bryans, Gamble et al. 2015) 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 and ensures convexity and acyclicity using closure operation such that there are no paths lead out of the set of nodes and back in again. If that does not preserve the validity of the provenance graph, then the set of nodes will be extended such that sink (boundary) 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. However, the resulting set after



(a) A convex boundary with cause and effect boundary nodes

(b) A broken set: the node e_5 has an incoming edge to and an outgoing edge from the boundary area

FIGURE 2.13: Chunk of nodes (Missier 2013)

the closure and extend operations may include extra nodes that were not part of the nodes that required abstraction causing excessiveness.

The algorithm is restricted to the entity and activity node types and to the causal edges *used* and *wasGeneratedBy* with no cyclic dependencies. An unbroken chunk of nodes is the set of nodes where no node outside the set has incoming and outgoing edges to the set, as illustrated in Figure 2.13(a). Abstracting a chunk of nodes may cause creating cycles if there exists a node outside the set with incoming and outgoing edges to the set, this chunk of nodes is called a broken set as shown in Figure 2.13(b).

Two types of node chunks are defined: a-chunk and e-chunk. A-chunk is an unbroken set of nodes where all boundary nodes are activities, while e-chunk is an unbroken set of nodes where all boundary nodes are entities. The algorithm replaces an a-chunk or e-chunk of nodes by an activity or an entity respectively.

It is possible for the chunk to generate false dependency by creating paths between nodes in the transformed graph that where not in the original graph. To avoid this, the chunk must be causally-total, i.e paths between any two nodes in the chunk are also in the chunk. To this end, additional nodes, that are not initially required to be abstracted, may need to be included in the abstraction, thus **excessiveness**. An example of transformation steps is shown in Figure 2.14.

Despite the above expansion, the algorithm may still creates **false dependencies** (hence **false paths**) when a set of nodes replaces by an abstract entity and the set has outgoing and incoming edges to and from activities resulting in an *wasInformedby (info)* relationship between the two activities.

In addition, the algorithm may result in different transformed graphs if we consider a-chunk or e-chunk for the same set of abstracting nodes in the same provenance graph, which is a non-deterministic graph transformation.

TACLP (Danger, Curcin, Missier et al. 2015)

An access control language for provenance data TACLP (Transformation-oriented Access Control Language for Provenance) is suggested in (Danger, Curcin, Missier et al. 2015). The paper presents an algorithm that transforms graphs based on a set of access control restrictions. The algorithm produces valid graphs that provide the maximum amount of information while it preserves privacy by allowing graph transformations according to user specifications and provenance inference rules. TACLP allows users to define subgraphs to be transformed, with three different levels of abstractions (namely hide, minimal and maximal). Two different operations are used to transform graphs: by removing a set of nodes or replacing them with a new abstract node. The set of nodes is partitioned in such a way that removal or replacement of each partition will not cause false dependencies by defining causality-preserving partitions.

The algorithm defines a type graph consists of node and edge types of *OPM* model and an abstract node and an abstract edge. In addition, the provenance graph defined in this paper can have indirect edges using the transitivity of *wasDerivedFrom (der)* edge in OPM.

Both removal and replacement are based on the notion of external effects and causes w.r.t. to a set of nodes as shown in Figure 2.15(a). The algorithm reconnects the external effects and causes as necessary such that the causal relationships among the remaining nodes will not be affected. Transformation examples for removal and replacement are shown in Figure 2.15(b).

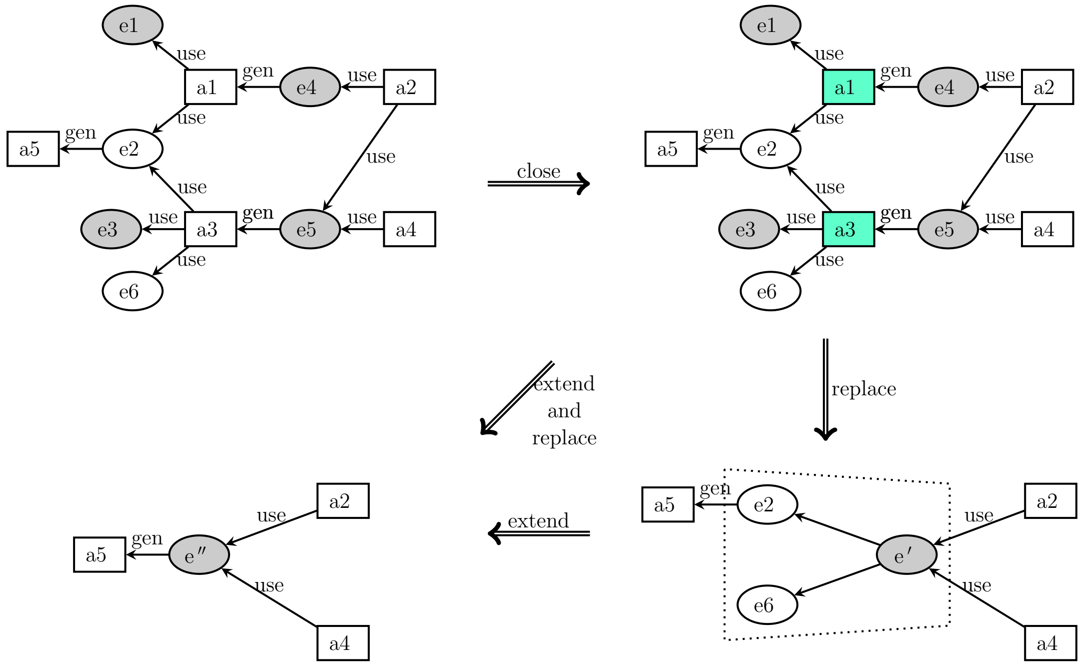
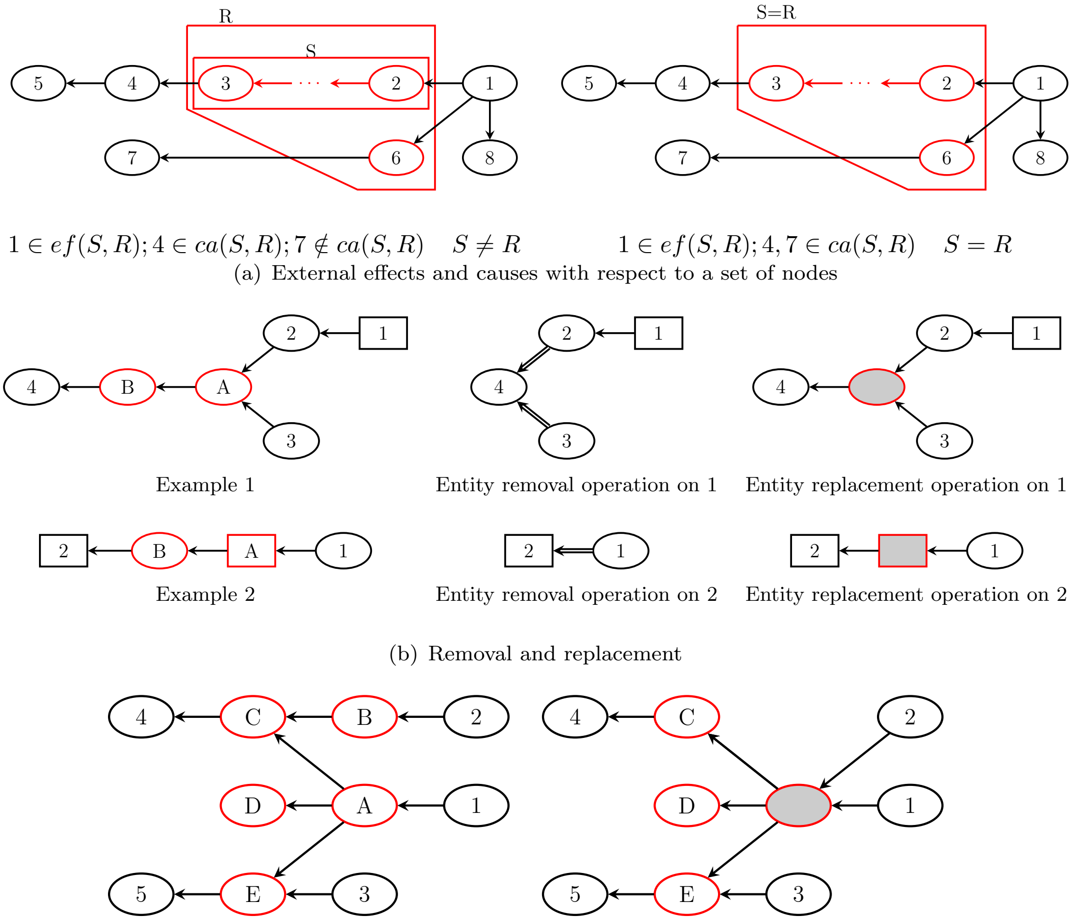


FIGURE 2.14: Graph transformation in Prov-Abs (Missier, Bryans, Gamble et al. 2015)


R
S
5
4
3
· · ·
2
1
7
6
8
S=R
5
4
3
· · ·
2
1
7
6
8
2
1
4
B
A
3
Example 1
2
1
4
3
Entity removal operation on 1
2
1
4
3
Entity replacement operation on 1
2
B
A
1
Example 2
2
1
Entity removal operation on 2
2
1
Entity replacement operation on 2
4
C
B
2
D
A
1
5
E
3
4
C
2
D
1
5
E
3

$1 \in ef(S,R); 4 \in ca(S,R); 7 \notin ca(S,R) \quad S \neq R$ $\qquad$ $1 \in ef(S,R); 4,7 \in ca(S,R) \quad S = R$

(a) External effects and causes with respect to a set of nodes

(b) Removal and replacement

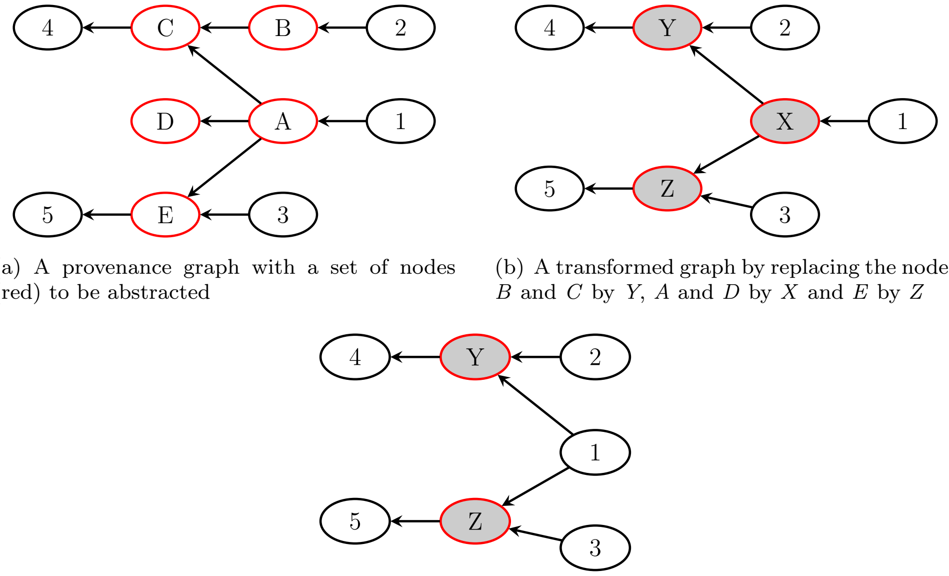
(c) False Paths: replacing the nodes *A* and *B* by an abstract node results in a false path between the nodes *2* and *5*

FIGURE 2.15: Graph transformation in TACLP (Danger, Curcin, Missier et al. 2015)

When removing a set of nodes, the nodes in external effects and causes of the set are reconnected using indirect edges. In case of replacing the set of nodes with an abstract node, edges from the abstract node to each node in external causes and from each node in external effects to the abstract node are generated. However, this transformation may introduce false dependencies and false paths. For example, abstracting the red-coloured nodes (A, B, C, D and E) in Figure 2.15(c) by an abstract node results in false paths from node *'2'* to each of the nodes *'D'*, *'E'* and *'5'*.

The algorithm tackles this problem by defining causality-preserving partitions which are used to avoid producing paths between external effect nodes and external cause nodes. In Figure 2.16, different partitions of a set of nodes in the graph shown in Figure 2.16(a) have been replaced by abstract nodes. Having different ways to partition the same set of nodes results in different transformed graphs as shown in Figure 2.16(b) and Figure 2.16(c).

However, it is still possible to generate a false dependency when an artifact (entity) is used as the abstract node and there are process nodes (activities) in both external effects



(a) A provenance graph with a set of nodes (red) to be abstracted

(b) A transformed graph by replacing the nodes *B* and *C* by *Y*, *A* and *D* by *X* and *E* by *Z*

(c) A transformed graph by replacing the nodes *A*, *B*, *C* and *D* by *Y*, and *E* by *Z*

FIGURE 2.16: Transformation with causality preserving partitions (Danger, Curcin, Missier et al. 2015)

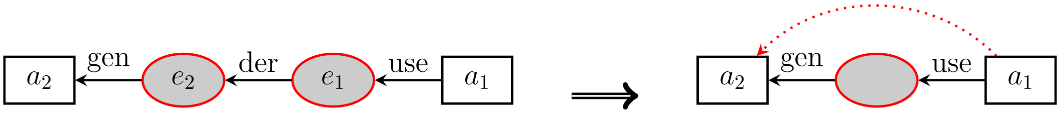
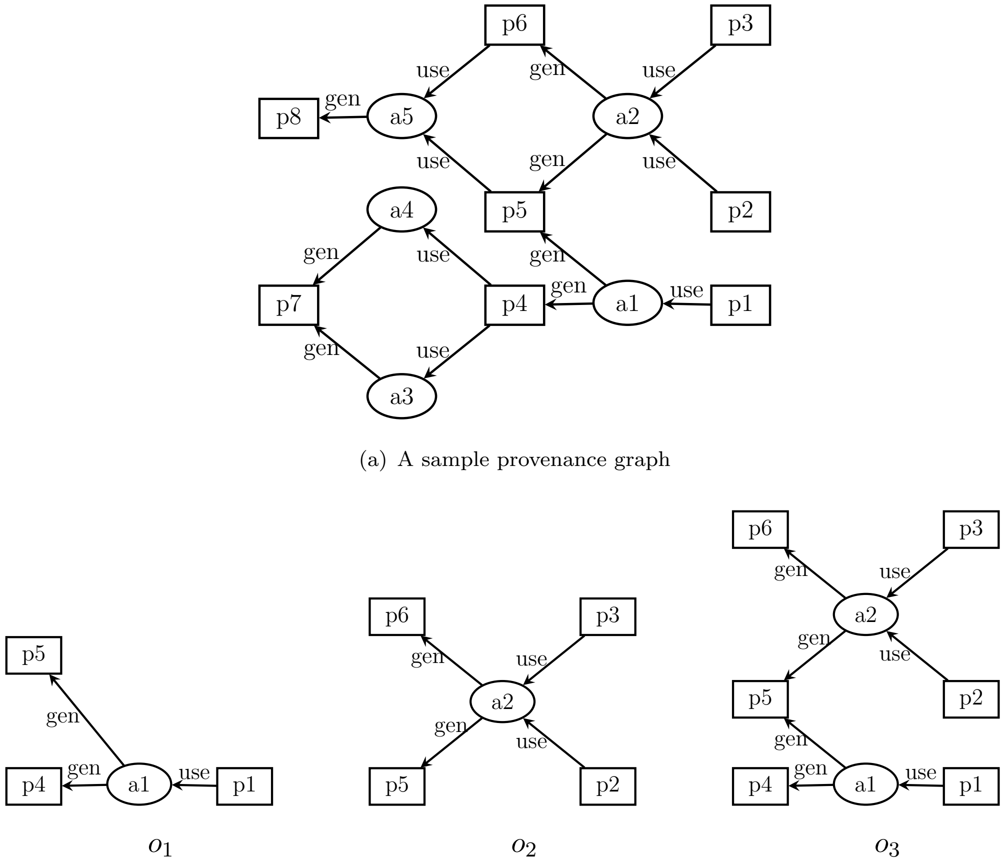


FIGURE 2.17: False dependency caused by TACLP when replacing the two entities e_1 and e_2 by an abstract entity which results in *wasInformedBy* edge between a_1 and a_2

and external causes of the set of nodes, which result in an *wasInformedBy (info)* edge as shown in Figure 2.17.

Access (Chen, Edwards, Nelson et al. 2015)

In this paper, an access control model is proposed that restricts access to a provenance graph by contracting subgraphs that contain objects to be protected. The subgraph is chosen in such a way that the process of retracting it results in a valid provenance graph by considering validity constraints associated with the graph when specifying certain regions for contraction. Conflict-free access control policies are implemented using a hierarchical structure that comes from dividing the whole (OPM) provenance graph into a set of protection objects. The contraction is done by replacing a subgraph with restricted nodes by a process node (activity) but not artifact (entity) which prevents producing an invalid OPM graphs. An example of a provenance graph and three different protection objects are shown in Figure 2.18. The subgraphs are chosen in a way that all boundary nodes of the region are process nodes (activity nodes). This guarantees that replacing the protection object (subgraph) by a process nodes will not result in invalid edges.



(a) A sample provenance graph

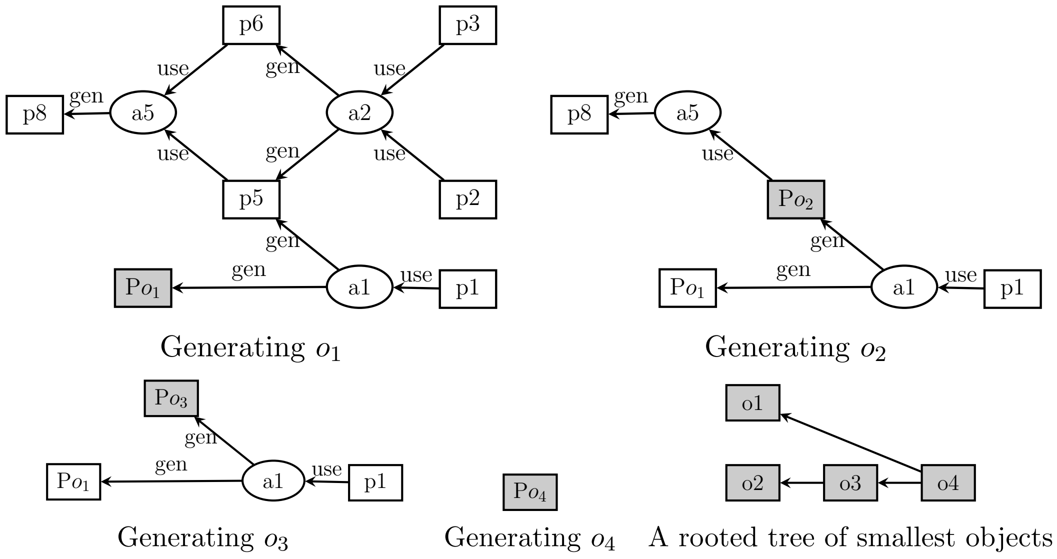
(b) Three redacted graphs where all boudary nodes are activities

FIGURE 2.18: A provenance graph and protection objects(Chen, Edwards, Nelson et al. 2015)

Selecting subgraphs as protection objects required expanding the region until all boundary edges are *process* nodes which leads to abstracting more nodes than the restricted nodes, i.e. excessiveness. In addition, the approach may result in false dependencies between the abstract process and other process nodes in the graph. For example, after the abstraction of the graph of Figure 2.18, an *wasInformedBy* edge can be inferred between some process nodes, for example between p_1 and p_{o_2} when generating o_2 as shown in Figure 2.19. Moreover, the algorithm may result in producing new paths that were not in the original graph.

2.8 The Outline of the Techniques Used by Reduction and Sanitization Approaches and the Relevant Issues

Table 2.1 shows the transformation techniques that have been used by the previous approaches and the proposed PROV-GTS system.



Generating o_1 Generating o_2

Generating o_3 Generating o_4 A rooted tree of smallest objects

FIGURE 2.19: Constructing a hierarchical structure of smallest objects for the graph in Figure 2.18 based on the graphs o_1, o_2 and o_3 (Chen, Edwards, Nelson et al. 2015)

TABLE 2.1: Provenance graph reduction and sanitization techniques that have been used by the graph rewriting approaches

Method	**Reference(s)**	**Reduction**	**Abstraction**	**Anonymization**
Workflow	(Chebotko, Chang, Lu et al. 2008; Chebotko, Lu, Chang et al. 2010)	✓	✓	✗
PROPUB	(Dey, Zinn and Ludäscher 2011a; Dey, Zinn and Ludäscher 2011b; Dey, Zinn and Ludäscher 2012; Dey and Ludäscher 2013)	✓	✓	✓
Framework	(Cadenhead, Kantarcioglu and Thuraisingham 2011)	✓	✗	✗
Language	(Cadenhead, Khadilkar, Kantarcioglu et al. 2011a; Cadenhead, Khadilkar, Kantarcioglu et al. 2011b)	✓	✗	✗
Surrogate	(Blaustein, Chapman, Seligman et al. 2011)	✗	✓	✗
ProvAbs	(Missier 2013; Missier, Bryans, Gamble et al. 2013; Missier, Bryans, Gamble et al. 2015)	✗	✓	✗
TACLP	(Danger, Curcin, Missier et al. 2015)	✗	✓	✗
Access	(Chen, Edwards, Nelson et al. 2015)	✗	✓	✗
PROV-GTS	The proposed	✓	✗	✓

In the previous section (Section 2.7), we justify this illustration by analysing each graph reduction and sanitization approach in details against the graph transformation properties. In Table 2.2, we summarize the graph transformation properties that have been preserved or violated by each graph reduction and sanitization approach. It is clear from the table that the proposed template-based PROV-GTS is the only approach that preserves all those properties.

TABLE 2.2: A comparison between the graph reduction approaches

Method	**Reference**	**NFD**	**NFI**	**PAP**	**NFP**	**NEX**	**VAP**
Workflow	(Chebotko, Chang, Lu et al. 2008; Chebotko, Lu, Chang et al. 2010)	✓	✗	✗	✓	✗	✓
ProPub	(Dey, Zinn and Ludäscher 2011a; Dey, Zinn and Ludäscher 2011b; Dey, Zinn and Ludäscher 2012; Dey and Ludäscher 2013)	✓	✓	✓	✓	✗	✓
Framework	(Cadenhead, Kantarcioglu and Thuraisingham 2011)	✗	✗	✗	✗	✓	✓
Language	(Cadenhead, Khadilkar, Kantarcioglu et al. 2011a; Cadenhead, Khadilkar, Kantarcioglu et al. 2011b)	✗	✓	✓	✗	✓	✗
Surrogate	(Blaustein, Chapman, Seligman et al. 2011)	✗	✗	✗	✓	✓	✓
ProvAbs	(Missier 2013; Missier, Bryans, Gamble et al. 2013; Missier, Bryans, Gamble et al. 2015)	✗	✓	✓	✗	✗	✓
TACLP	(Danger, Curcin, Missier et al. 2015)	✗	✓	✓	✗	✓	✓
Access	(Chen, Edwards, Nelson et al. 2015)	✗	✓	✓	✗	✗	✓
PROV-GTS	The proposed	✓	✓	✓	✓	✓	✓

NFD = No False Dependency NFI = No False Independency PAP = Path Preservation
NFP = No False Paths NEX = No Excessiveness VAP = Validity Preservation

2.9 Summary

In this chapter, the PROV data model and its inference rules are introduced. The basic concepts of category theory are defined and the notion of algebraic graph transformation is presented. The domains of graph reduction and sanitization approaches have been presented. In addition, the issues pertaining to provenance graph transformations have been determined. Finally, the most relevant previous approaches found in the literature are discussed. In the next chapter, two provenance graphs, which represent two different real situations, are provided and will be used as running examples throughout this thesis.

Chapter 3

Running Examples

3.1 Introduction

In the previous chapter, the areas of provenance graph transformation have been presented. The techniques used in variety of provenance rewriting systems along with their shortcomings have been described. In this chapter, two concrete provenance graphs are presented, which are used as running examples throughout this thesis. OnlinePost graph regarding an online post is described in Section 3.2 whereas RideShare graph shows ride plans created by the ride share application and presented in Section 3.3. The node anonymization process is depicted in Section 3.4.

The model of provenance we adopt is the W3C standardized PROV, aimed at sharing provenance information over the Web (Lebo, Sahoo, McGuinness et al. 2013; Moreau, Missier, Cheney et al. 2013; Cheney, Missier, Moreau et al. 2013). PROV defines a notion of graph consists of a set of nodes and a set of edges, as described in Section 2.2 of Chapter 2. In the proposed system, the sensitive parts of the provenance graph are specified by a set of *restricted* nodes, depicted as grey nodes ■, while non-restricted (*plain*) nodes are illustrated as white nodes □, and the nodes with hatched lines ▨ represent *anonymous* nodes. PROV-GTS deletes the edges connected to the restricted nodes in order to isolate those nodes from the provenance graph so they can be safely discarded. In addition, to preserve relations, PROV-GTS may create edges and (restricted) activities. Those created activities along with some other restricted nodes that cannot be fully isolated, since they are needed to preserve the integrity of the graph, will be anonymized. The anonymization is done by replacing the restricted nodes by less-sensitive anonymous nodes of the same type of the original nodes, as explained in Section 3.4.

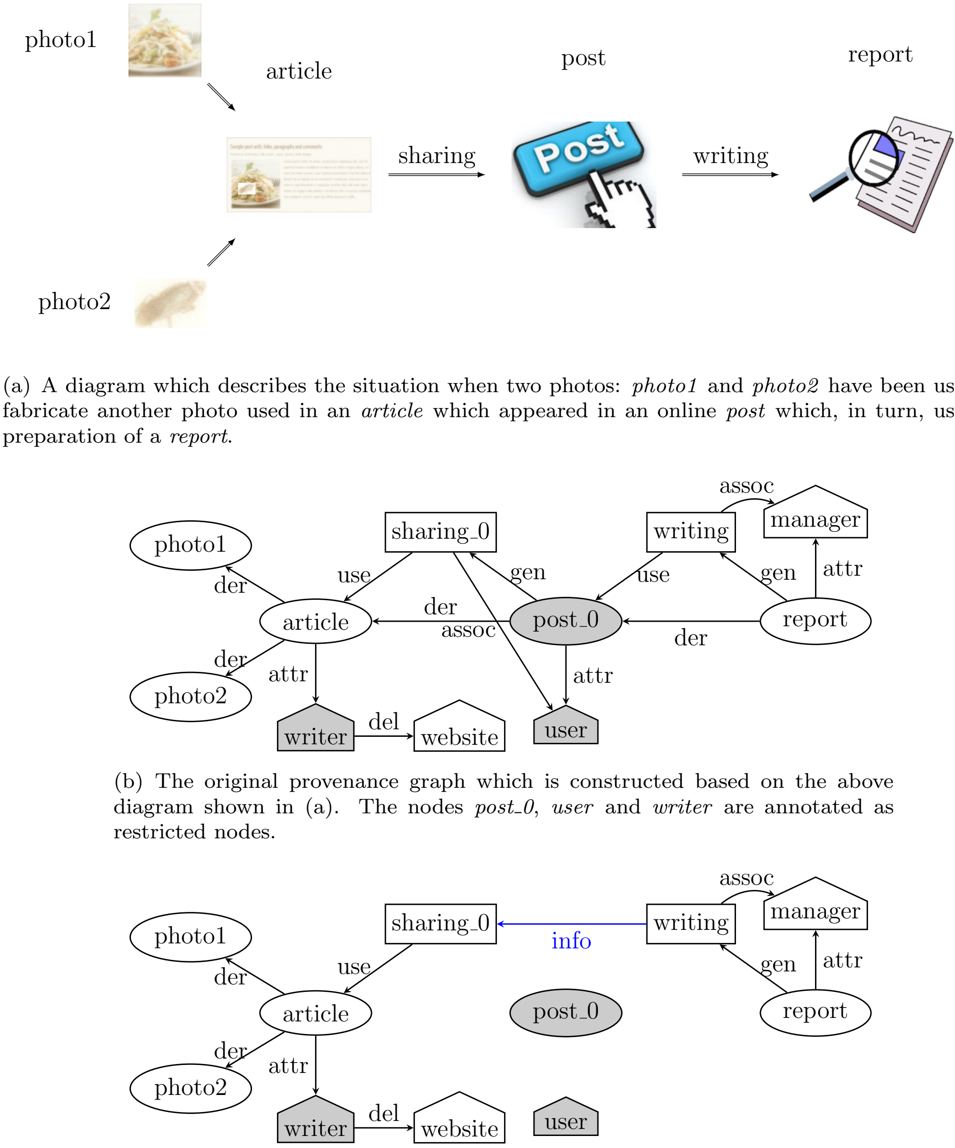
3.2 The Fake Online Post

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 is mainly used by rumour-mongers. Figure 3.1(a) shows a diagram that describes the participants that involved in generating the online post and the report.

The corresponding provenance graph that describes that situation is illustrated in Figure 3.1(b). 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* and the agent *user* have attributes which can be used to expose the identity of the individual who initially posted the photo on the social network website. In addition, the agent *writer* contains the personal information regarding the writer of the article. The template-based PROV-GTS is capable of isolating the nodes *post_0* and *user* without affecting the connectivity and integrity of the graph, i.e. no false dependency or false independency, while the *writer* cannot be isolated from the graph since it results in isolating the agent node *website* and any path leads to that node. In the transformed graph, as shown in Figure 3.1(c), to preserve the relation between the activities *writing* and *sharing_0* an *info* was created based on the PROV inference rule **Generation-Usage Inference** (see Section 2.2.2). The PROV-GTS managed to isolate the nodes *post_0* and *user* but not the agent *writer* since its incident edges are needed to preserve the paths that lead to the agent *website*. Step-by-step rule applications for the OnlinePost graph are shown in Section E.1 of Appendix E.

3.3 Ride Requests and Plans in Rideshare Application

The ride share application generates potential ride plans for commuters by automatically matching their ride requests to those offered by drivers (Packer and Moreau 2015). These ride plans are made by taking into account location of departures and arrivals, routes, car's capacity, and reputation. This application consists of a view, ride matching, reputation, and ProvStore (Huynh and Moreau 2014). The view provides the graphical interface that enables the user to enter ride requests and select potential ride plans (Packer, Dragan and Moreau 2014). These components interact with each other using REST APIs. ProvStore has been used by the other components to store the generated provenance data.



(a) A diagram which describes the situation when two photos: *photo1* and *photo2* have been used to fabricate another photo used in an *article* which appeared in an online *post* which, in turn, used in preparation of a *report*.

(b) The original provenance graph which is constructed based on the above diagram shown in (a). The nodes *post_0*, *user* and *writer* are annotated as restricted nodes.

(c) The graph transformed by PROV-GTS: An *info* edge is created between the activities *writing* and *sharing_0* before isolating the nodes *post_0* and *user* by deleting their edges, while the edges that are incident to the agent *writer* are kept since they are needed to preserve the paths that lead to the agent *website*.

FIGURE 3.1: A running example (the fake online post): (a) the online post diagram (b) the original provenance graph (c) the transformed provenance graph

The reputation service generates provenance data and stores it in the ProvStore when responding to the following actions made by ride share application:

- requesting reputation reports about a user.
- requesting all feedback reports about a subject.
- requesting feedback reports about a user.
- submitting feedback reports.
- submitting provenance information directly to the ProvStore.

In addition, users generates provenance data when:

- a driver creates a ride request.
- a commuter makes a ride request.
- using the two above requests and users' reputation from their reputation reports, the ride share application runs a ride matching algorithm and generates a ride plan.
- the driver views the commuter's reputation and accept the ride plan.
- the commuter views the driver's reputation and accept the ride plan.

For each of the above actions, a pre-defined provenance template can be used to generate provenance graphs for different requests and actions by binding values to the variables in those templates (Packer and Moreau 2015).

Figure 3.2 represents the RideShare graph which is a snapshot from a larger provenance graph generated by the rideshare application. This graph shows how a ride plan has been created from ride requests and intermediate ride plans. The initial ride request *'rideRequests/123'* has been generated by the activity *'receiving_request_123'* using the entity *'rideRequests/123/v/1'*. In addition, the initial ride request has been used to derive a new ride request entity (*'rideRequests/123/v/0'*) which in turn has been used to derive an invalid ride plan (the entity *'ridePlans/60/v/0'*). Both entities *'rideRequests/123/v/0'* and *'ridePlans/60/v/0'* have been involved in deriving the ultimate ride plan which is illustrated by the entity *'ridePlans/60'*. All the entities that represent the ride requests and the ride plans were attributed to the agent *'IMAGINARY-ABACAS'* which represents the Orchestrator Service.

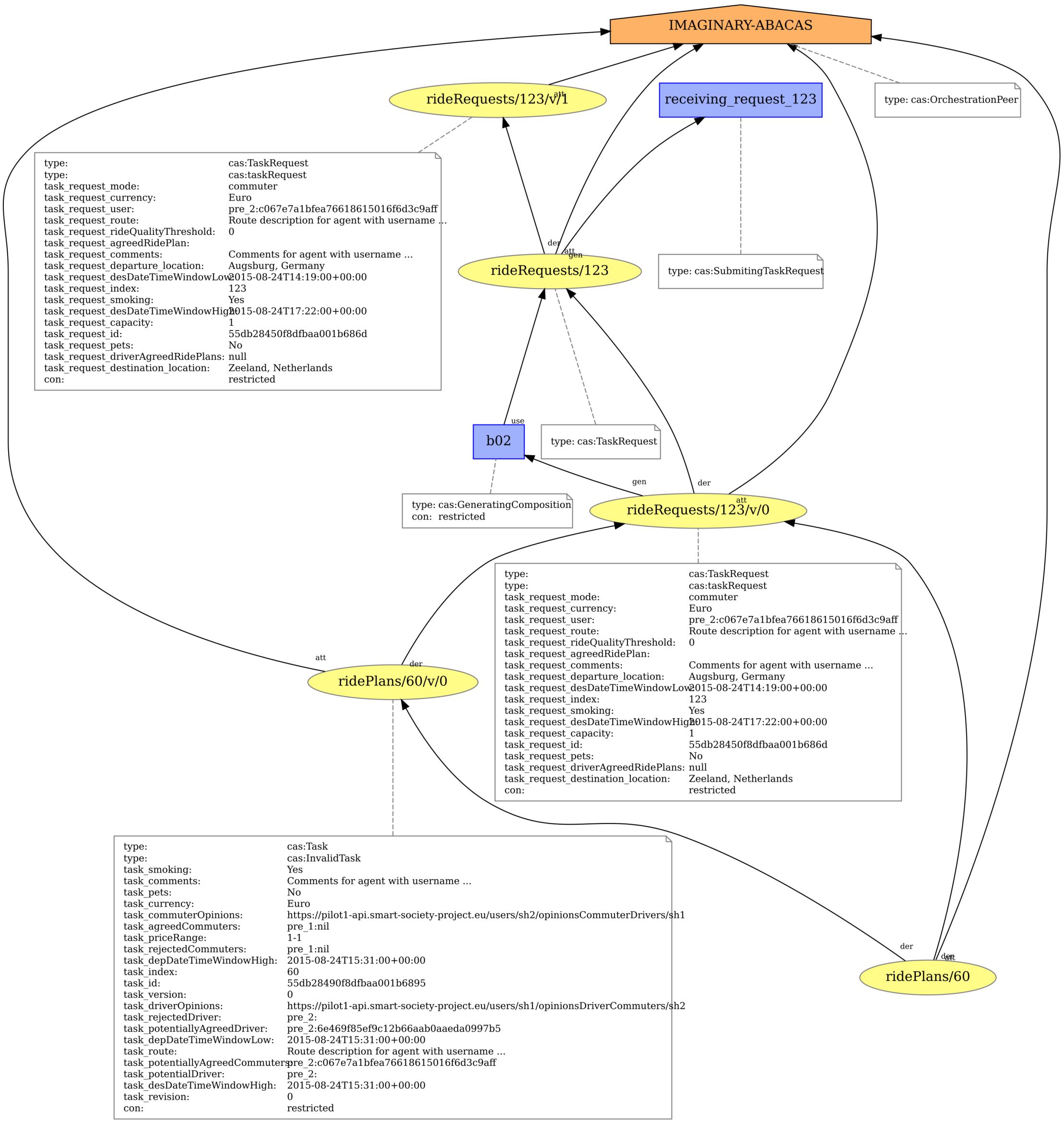

IMAGINARY-ABACAS
rideRequests/123/v/1
att
receiving_request_123
type: cas:OrchestrationPeer
type: cas:TaskRequest
type: cas:taskRequest
task_request_mode: commuter
task_request_currency: Euro
task_request_user: pre_2:c067e7a1bfea76618615016f6d3c9aff
task_request_route: Route description for agent with username ...
task_request_rideQualityThreshold: 0
task_request_agreedRidePlan:
task_request_comments: Comments for agent with username ...
task_request_departure_location: Augsburg, Germany
task_request_desDateTimeWindowLow: 2015-08-24T14:19:00+00:00
task_request_index: 123
task_request_smoking: Yes
task_request_desDateTimeWindowHigh: 2015-08-24T17:22:00+00:00
task_request_capacity: 1
task_request_id: 55db28450f8dfbaa001b686d
task_request_pets: No
task_request_driverAgreedRidePlans: null
task_request_destination_location: Zeeland, Netherlands
con: restricted
der
att
gen
rideRequests/123
type: cas:SubmitingTaskRequest
use
b02
type: cas:TaskRequest
type: cas:GeneratingComposition
con: restricted
gen
der
att
rideRequests/123/v/0
type: cas:TaskRequest
type: cas:taskRequest
task_request_mode: commuter
task_request_currency: Euro
task_request_user: pre_2:c067e7a1bfea76618615016f6d3c9aff
task_request_route: Route description for agent with username ...
task_request_rideQualityThreshold: 0
task_request_agreedRidePlan:
task_request_comments: Comments for agent with username ...
task_request_departure_location: Augsburg, Germany
task_request_desDateTimeWindowLow: 2015-08-24T14:19:00+00:00
task_request_index: 123
task_request_smoking: Yes
task_request_desDateTimeWindowHigh: 2015-08-24T17:22:00+00:00
task_request_capacity: 1
task_request_id: 55db28450f8dfbaa001b686d
task_request_pets: No
task_request_driverAgreedRidePlans: null
task_request_destination_location: Zeeland, Netherlands
con: restricted
att
der
ridePlans/60/v/0
type: cas:Task
type: cas:InvalidTask
task_smoking: Yes
task_comments: Comments for agent with username ...
task_pets: No
task_currency: Euro
task_commuterOpinions: https://pilot1-api.smart-society-project.eu/users/sh2/opinionsCommuterDrivers/sh1
task_agreedCommuters: pre_1:nil
task_priceRange: 1-1
task_rejectedCommuters: pre_1:nil
task_depDateTimeWindowHigh: 2015-08-24T15:31:00+00:00
task_index: 60
task_id: 55db28490f8dfbaa001b6895
task_version: 0
task_driverOpinions: https://pilot1-api.smart-society-project.eu/users/sh1/opinionsDriverCommuters/sh2
task_rejectedDriver: pre_2:
task_potentiallyAgreedDriver: pre_2:6e469f85ef9c12b66aab0aaeda0997b5
task_depDateTimeWindowLow: 2015-08-24T15:31:00+00:00
task_route: Route description for agent with username ...
task_potentiallyAgreedCommuters: pre_2:c067e7a1bfea76618615016f6d3c9aff
task_potentialDriver: pre_2:
task_desDateTimeWindowHigh: 2015-08-24T15:31:00+00:00
task_revision: 0
con: restricted
der
der
att
ridePlans/60

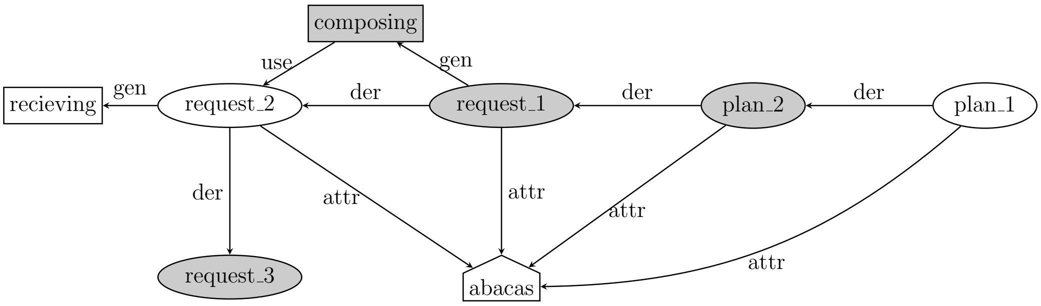
FIGURE 3.2: The original rideshare graph which represents the ride requests and ride plans that involved in generating the ride plan that represent by the entity *ridePlans/60*. The restricted nodes (*ridePlans/60/v/0*, *rideRequests/123/v/0*, *b02* and *rideRequests/123/v/1*) are annotated by inserting the attribute *con:restricted* to the end of each one of their list of attributes.

The rideshare graphs may expose many kinds of confidential and private information, such as personal information of user: name, address, etc., sensitive information regarding URLs, or location information. The users may want to prevent their location information to be exposed when the provenance graph RideShare is used as part of reputation report or published on store repositories. In our particular graph, the location information are represented by the attributes *'task_request_departure_location'* and *'task_request_departure_location'* of the entities *'rideRequests/123/v/0'*, *'rideRequests/123/v/1'* and *'ridePlans/60/v/0'*. These entities have been annotated as restricted nodes via the attribute *'con'* with the value *'restricted'*. In addition to the location information, the activity *'b02'*, which it had generated the restricted entity *'rideRequests/123/v/0'*, is annotated as a restricted node as well. We could hide the location attributes and keep the nodes, but that will make the ride requests represented by those nodes useless, since the departure and destination locations are unknown. That is why we prefer to hide those nodes from the graph rather than just hiding the attributes that expose confidential information.

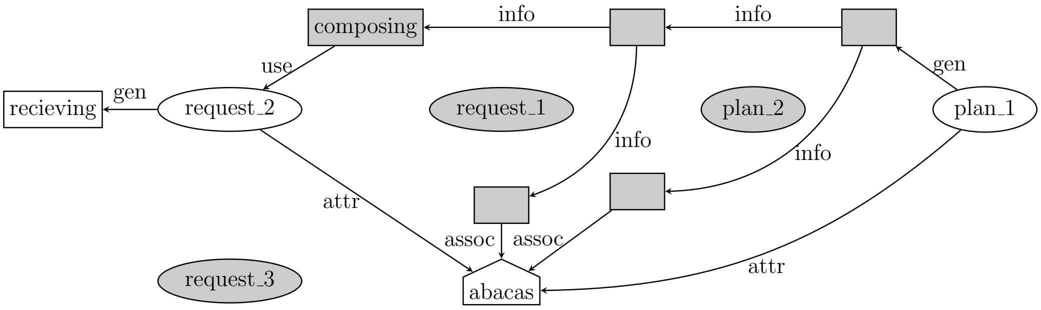
In Figure 3.3(a), for convenience, the nodes *'rideRequests/123'*, *'rideRequests/123/v/0'*, *'rideRequests/123/v/1'*, *'ridePlans/60'*, *'ridePlans/60/v/0'*, *'b02'*, *'receiving_request_123'*, and *'IMAGINARY-ABACAS'* are changed to *'request_1'*, *'request_2'*, *'request_3'*, *'plan_1'*, *'plan_2'*, *'composing'*, *'receiving'*, and *'abacas'* respectively. The transformed graph, which is created by PROV-GTS as a result of converting the RideShare graph, is shown in Figure 3.3(b). PROV-GTS was able to isolate the entities *'request_1'*, *'request_3'*, and *'plan_2'* from the graph. Many (restricted) activities have been added to the graph regarding the *der* and *attr* edges connected to the nodes *'request_1'* and *'plan_2'*, in order to preserve several relations and paths prior to deleting the edges that are incident to those nodes. The two edges connected to the activity *composing* have not been deleted since they are needed to preserve various paths in the transformed graph. The added restricted activities as well as the activity *composing* can then be further processed either by isolating them or replacing them with anonymous nodes. Step-by-step rule applications for the RideShare graph are shown in Section E.2 of Appendix E.

3.4 Node Anonymization

Clearly, not all nodes can be fully isolated from the provenance graphs, as some edges are required to keep some nodes connected together and hence to avoid false independency and preserve paths. For example, the edges *attr* and *del* connected to the agent *writer* in the transformed graph of Figure 3.1(c) cannot be deleted because they are required to maintain the paths that lead to the agent *website*. In addition, in Figure 3.3(b) the restricted activities (the *composing* activity and the new created activities) have not been completely isolated since keeping some edges is necessary in order to preserve the connectivity of the graph. In fact, the aim of adding the newly created activities is to fulfil such preservation. In this case, to avoid exposing confidential information regarding those nodes, we replace them by fictitious, anonymous nodes of the same type of the original node. The remaining restricted nodes of the provenance graphs shown in Figure 3.1(c) and Figure 3.3(b) are anonymized, after deleting the restricted nodes that were fully isolated, as shown in Figure 3.4(a) and Figure 3.4(b) respectively.

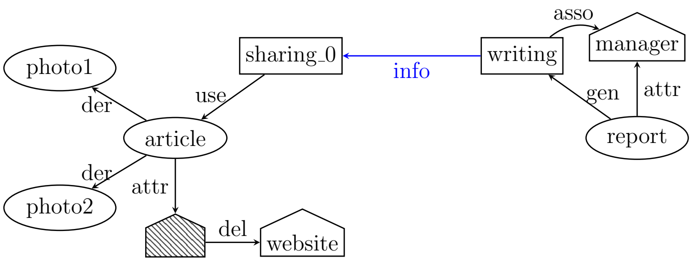


(a) The original RideShare provenance graph based on the graph shown in Figure 3.2 after abbreviating the nodes' labels *'rideRequests/123'*, *'rideRequests/123/v/0'*, *'rideRequests/123/v/1'*, *'ridePlans/60'*, *'ridePlans/60/v/0'*, *'b02'*, *'receiving_request_123'*, and *'IMAGINARY-ABACAS'* to *'request_1'*, *'request_2'*, *'request_3'*, *'plan_1'*, *'plan_2'*, *'composing'*, *'receiving'*, and *'abacas'* respectively.

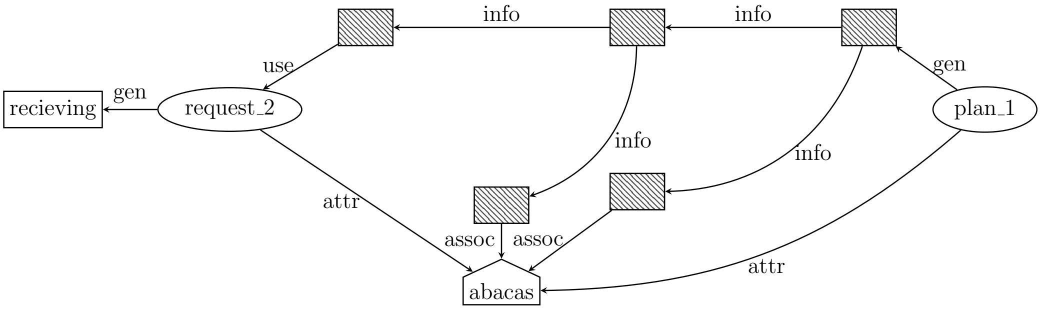


(b) The RideShare graph transformed by PROV-GTS. Four restricted activities and four *info* edges have been created before isolating the restricted entities *plan_2*, *request_1* and *request_3*. The restricted activity *composing* and the newly added activities are needed to preserve the relations and paths between the remaining nodes in the graph.

FIGURE 3.3: A running example from rideshare requests and plans: (a) the original provenance graph (b) the transformed provenance graph



(a) The transformed OnlinePost graph with anonymous nodes



(b) The transformed RideShare graph with anonymous nodes

FIGURE 3.4: The final transformed OnlinePost and RideShare graphs after deleting the restricted nodes that have been isolated and then anonymizing the remaining ones.

3.5 Summary

In this chapter, two provenance graphs from real life situations have been provided. The first one describes the entities and participants that influence the emerging of an online post contains fake information. While the second represents a part of a provenance graph generated by the rideshare application and shows how a ride plan has been generated. The two graphs will be used in the upcoming chapters to show how the graph transformation rules are constructed and used in transforming provenance graphs. In the next chapter, the construction of the graph transformation rules of PROV-GTS, which is based on the PROV data model, is described in detail.

Chapter 4

Construction of Provenance Transformation Rules

4.1 Introduction

In the previous chapter, two provenance graphs represent two real situations have been presented and will be used as running examples. In this chapter, we formally describe the core PROV data model graphs. Then, this formal description of provenance graphs is used in defining the conditional graph transformation rules of PROV-GTS. Finally, a set of graph transformation rules is proposed. The rules are gradually constructed based on properties from the PROV data model taking into account the graph integrity and determinism of the rule applications. The constructed conditional rules are capable of obscuring restricted provenance information while maintaining graph integrity by preserving paths and avoiding false independencies and false dependencies.

The PROV data model presents provenance information as graphs with a set of interrelated nodes connected via a set of edges. Provenance graphs can be validated over sets of constraints and inference rules (Cheney, Missier, Moreau et al. 2013). In the previous chapter, we provide two examples that represent real situations with privacy issues and we show that the proposed approach is capable of converting those graphs while preserving their integrity. The aim of the proposed template-based PROV-GTS is to construct a set of transformation rules that delete the edges connected to restricted nodes in order to isolate those nodes from the provenance graph. In this chapter, based on properties from PROV data model, we construct a set of graph transformation rules. First, we start with the basic rules (consisting of left-hand and right-hand sides) and then we move to adding conditions to the rules as necessary to ensure graph integrity and also to guarantee consistency between the transformation rule applications. For each type of edges, the system requires at least two transformation rules: one when the source node of the edge is restricted and the other when the target node is restricted.

A principled definition of PROV graph transformation rules, which are capable of obscuring restricted provenance information while maintaining graph integrity by preserving paths and avoiding false independencies and false dependencies, is presented.

In Section 4.2, graph patterns are used to illustrate the confidentiality level values and then PROV type graph with confidentiality level is defined. The Definition of graphs with reverse edges is presented in Section 4.3. In Section 4.4, the PROV type graph is defined. The definition of typed PROV graph with inheritance is provided in Section 4.5. Properties from PROV data model are presented in Section 4.6 and used to construct graph transformation rules in Section 4.7. How graph integrity is dealt with is discussed in Section 4.8. Section 4.9 used universal-existential conditions to tackle the issue of multiple similar events that involved the same node in PROV data model. Path preservation is presented in Section 4.10. Section 4.11 shows how non-determinism of rule applications is avoided by expanding LHS, adding extra negative application conditions, or changing the confidentiality level values.

4.2 Graphs with Confidentiality Level Values

In general, the definition of directed graphs consists of a set of nodes, a set of edges, and the functions that assign to each edge its source and target nodes. In the proposed system, we expand this general definition of directed graphs by adding a set of values presented as data nodes called confidentiality level values and used to determine which nodes are sensitive and which are not. In addition, proper functions that maps between the graph nodes and confidentiality level data nodes are added.

Definition 4.1 (Directed graph with confidentiality levels)**.** A directed graph with confidentiality levels is $CG = (N_{CG}, E_{CG}, C_{CG}, B_{CG}, s_{CG}, t_{CG}, c_{CG}, d_{CG})$ where N_{CG} is the set of nodes, $E_{CG} \subseteq N_{CG} \times N_{CG}$ is the set of edges, and C_{CG} is the set of confidentiality level nodes and $B_{CG} \subseteq N_{CG} \times C_{CG}$ is the set of confidentiality edges. The functions $s_{CG}, t_{CG} : E_{CG} \rightarrow N_{CG}$ assign to the graph edges their source and target nodes respectively, while $c_{CG} : B_{CG} \rightarrow N_{CG}$ and $d_{CG} : B_{CG} \rightarrow C_{CG}$ assign respectively the graph nodes and the graph confidentiality level nodes to the confidentiality level edges. □

A graph morphism is a mapping between two graphs respecting their structures. That is, it maps the sets of nodes and edges in the first graph to their corresponding sets in the second graph in such a way that preserves the adjacency of the edges. This kind of morphisms are called homomorphisms. In case of typed graphs, in addition to the above preservation, morphisms must preserve the types of nodes and edges. In our system, this mapping must be extended to preserve the confidentiality level values of the graph nodes as well.

Definition 4.2 (Directed graph with confidentiality levels morphism)**.** Directed graph with confidentiality levels morphism $f : G \longrightarrow H$ consists of four functions $f_N : N_G \rightarrow E_G$, $f_E : E_G \rightarrow E_N$, $f_C : C_G \rightarrow C_H$ and $f_B : B_G \rightarrow B_H$ such that $f_N \circ s_G = s_H \circ f_E$, $f_N \circ t_G = t_H \circ f_E$, $f_N \circ c_G = c_H \circ f_B$ and $f_C \circ d_G = d_H \circ f_B$. □

The above definition represents the total morphisms which map each node and edge in graph G to a node and an edge in graph H. If the morphism maps a sub-graph of G to H then it is a partial morphism.

Definition 4.3 (Directed graph with confidentiality levels partial morphism)**.** A partial graph morphism $g : G \longrightarrow H$ is a total graph morphism from some sub-graph K of G to H, where $N_K \subseteq N_G$ and $E_K \subseteq E_G$.

□

4.3 Graphs with Reverse Edges

When grouping the transformation rules in templates for the rules that share the same graph topologies, some rules have the same topologies but the edges are reversed. To group this kind of rules in templates, we need to consider graphs with reverse edges and graph morphisms regarding those reverse graphs.

The standardized PROV Ontology (PROV-O) recommended an inverse name for each statement or property in PROV model so that modellers could use one set instead of the other (Lebo, Sahoo, McGuinness et al. 2013). We attach to each provenance graph a reverse indicator $d \in \{1, -1\}$, where 1 and -1 indicate a graph with direct and reverse edges respectively. First, we define the reverse PROV graph (Definition 4.4), reverse graph with reverse indicator (Definition 4.5), and then the morphisms (total and partial) between PROV graphs with reverse indicator (Definition 4.6). The graphs can have either direct edges or reverse edges but not a mixture of them.

Definition 4.4 (Reverse directed graph)**.** The reverse of a directed graph with confidentiality levels G is G^{-1} where:

$$N_{G^{-1}} = N_G$$

$$E_{G^{-1}} = \{(t, s) \mid (s, t) \in E_G\}$$

$$C_{G^{-1}} = C_G$$

$$B_{G^{-1}} = B_G$$

□

Definition 4.5 (Directed graph with confidentiality levels and reverse indicator)**.** A graph with reverse indicator is $G = (\mathcal{G}, d)$ where $\mathcal{G}$ is directed graph with confidentiality levels with forward or reverse graph edges specified by $d \in \{1, -1\}$. The variable d holds either the value of 1 or -1 indicating the direction of edges in graph $\mathcal{G}$, direct or reverse respectively, where $\mathcal{G}^1 = \mathcal{G}$ and $\mathcal{G}^{-1}$ is the reverse of graph $\mathcal{G}$. □

The morphisms between graphs with reverse indicator are the morphisms defined in Definition 4.6 after ensuring that the edges of a graph with $d = -1$ have been converted to direct edges by reversing their source and target nodes.

Definition 4.6 (Directed graph with confidentiality levels and reverse indicator morphism)**.** A directed graph with confidentiality levels morphism $f : G \longrightarrow H$ between two graphs $G = (\mathcal{G}, d_1)$ and $H = (\mathcal{H}, d_2)$ with reverse indicators is the (total or partial) morphism $f^{'} : \mathcal{G}^{d_1} \longrightarrow \mathcal{H}^{d_2}$.

□

Reversing the direction of the edges in a graph with reverse edges results in the same graph but with direct edges, that is $\left(G^{-1}\right)^{-1} = G$.

The category of graphs with reverse indicator consists of a set of directed graphs with reverse indicator (as defined in Definition 4.5) as objects and a set of morphisms between them (as defined in Definition 4.6) as arrows.

Definition 4.7 (graph category)**.** Let $Graph_p$ be the category of directed graphs with reverse indicator and partial graph morphisms between them. We use *Graph* to denote the category of directed graphs with reverse indicator and their (total) graph morphisms. □

4.4 PROV Data Model Graphs

The graph representation of the PROV data model consists of nodes and edges labelled with identifiers and optionally accompanied by a set of attributes (Moreau, Missier, Cheney et al. 2013). A type graph consists of a set of types used to assign types to nodes and edges via typing morphisms between a graph and the type graph (Ehrig, Ehrig, Prange et al. 2006c). The type graph of the core PROV data model, as illustrated in Figure 4.1, is a directed graph which consists of the set of nodes and edges defined in Section 2.2 of Chapter 2, in addition to a set of confidentiality level nodes. As there is a unique edge type between each pair of node types in the core PROV data model, we ignore edge types; instead, the types of the edges will be determined by the types of their source and target nodes. Therefore, in the upcoming definitions, edges are represented by pairs as (s, t) where s and t are the source node and the target node of the edge respectively. For instance, *(entity, activity)* represents the edge *wasGeneratedBy* between the two PROV nodes: *entity* and *activity*. In addition, the (node and edge) labels are used for the purpose of clarification, they are not part of the system's formal description.

Graphically, the graph patterns □, ■ and ▨ are used to annotate the plain, restricted and anonymous nodes respectively. These patterns represent the confidentiality level of the graph nodes and illustrated in Figure 4.1 as data nodes with (*dashed*) edges from the graph nodes. In order to avoid adding a confidentiality level node for each graph node when drawing provenance graphs, we draw the graph nodes with the graph patterns that represent their confidentiality levels. For example, the node ⬬ represents a restricted entity and ⌂ is used to depict a plain agent node.

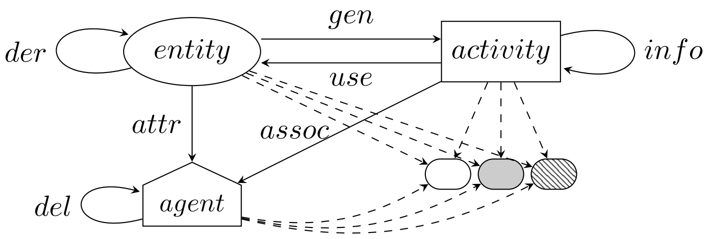


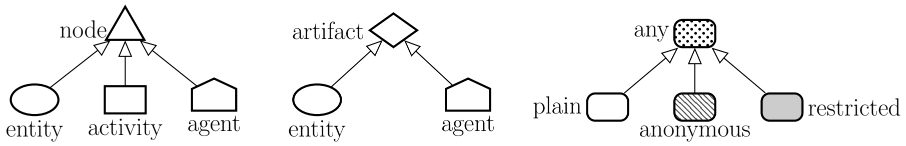
FIGURE 4.1: Core PROV data model type graph with confidentiality levels

Definition 4.8 (Type graph)**.** A type graph is a distinguished graph $(TG, 1)$ where $TG = (N_{TG}, E_{TG}, C_{TG}, B_{TG}, s_{TG}, t_{TG}, c_{TG}, d_{TG})$ is a directed graph with confidentiality levels and direct graph edges. □

As shown in Figure 4.1, the PROV type graph consists of {*entity, activity, agent*} as a set of node types, {*(entity, entity), (entity, activity), (activity, entity), (activity, activity), (entity, agent), (activity, agent), (agent, agent)*} as a set of edges, {*plain, restricted, anonymous*} as set of confidentiality level values, and a set of edges from each graph node type to every confidentiality level value.

4.5 Typed Graphs with Inheritance

The PROV model defines provenance graphs using two sets of typed nodes and edges which characterise various relations between the nodes as described in Section 4.4. In the construction of the conditional rules of the 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. universal-existential and negative application conditions, respectively. In both cases, abstract node types for PROV nodes and confidentiality level nodes are essential to avoid having many graph patterns to satisfy a particular condition. This 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 data model to have abstract graph nodes and confidentiality level nodes via the inheritance graphs shown in Figure 4.2. The type graph cannot be used to define the provenance graphs with abstract nodes that are required by PROV-GTS. In this section, a type graph with abstract nodes and inheritance are described and then used to define the typed graphs and their morphisms.



(a) Node inheritance graph (b) Confidentiality inheritance graph

Figure 4.2: Inheritance graphs in PROV-GTS

Definition 4.9 (Abstract nodes)**.** Let AN be a set of abstract graph nodes and AC be a set of abstract confidentiality level nodes. □

In PROV-GTS , $AN = \{node,\ artifact\}$ is a set of abstract graph nodes and $AC = \{any\}$ is a set of abstract confidentiality level nodes.

Now we expand the definition of type graph with confidentiality levels by adding the abstract graph nodes and abstract confidentiality level nodes to it resulting in the following definition (Definition 4.10).

Definition 4.10 (Type graph with abstract elements)**.** A type graph with abstract elements is $(AG, 1)$ where $AG = (N_{AG}, E_{AG},\ C_{AG}, B_{AG}, s_{AG}, t_{AG}, c_{AG}, d_{AG})$ is a directed graph with confidentiality levels and direct edges. Given a type graph $TG = (N_{TG}, E_{TG}, C_{TG}, B_{TG}, s_{TG}, t_{TG}, c_{TG}, d_{TG})$, the sets $N_{AG} = N_{TG} \cup AN$ and $C_{AG} = C_{TG} \cup AC$ are the graph nodes and confidentiality level nodes with abstract elements respectively, and $E_{AG} = E_{TG}$ and $B_{AG} = B_{TG}$ are the graph edges and confidentiality level edges respectively. The functions $s_{AG}, t_{AG} : E_{AG} \rightarrow N_{AG}$ assign to graph edges their source and target graph nodes and $c_{AG} : B_{AG} \rightarrow N_{AG}$ and $d_{AG} : B_{AG} \rightarrow C_{AG}$ assign graph nodes and confidentiality level nodes to confidentiality level edges respectively. □

When mapping between graphs that consist of abstract nodes, each one of the abstract nodes can be mapped either to itself or to one of its subtypes. For example, the node *node* can be mapped to itself or any of PROV model nodes: *entity*, *activity* and *agent*. In addition, the subtypes of the abstract node *artifact* are *entity* and *agent* while the subtypes of the confidentiality level abstract node *any* are *plain*, *anonymous* and *restricted*.

Definition 4.11 describes the node inheritance graph where the set of graph nodes is same as the set of nodes in the type graph with abstract elements, while the set of edges consists of all the subtype edges of the node inheritance graph.

Definition 4.11 (Node inheritance graph)**.** A directed graph $IN = (N_{IN}, E_{IN}, s_{IN}, t_{IN})$ is the node inheritance graph where: $N_{IN} = N_{AG} = N_{TG} \cup AN$ is the set of nodes, $E_{IN} = (N_{TG} \times AN)$ is the set of edges, and $s_{IN}, t_{IN} : E_{IN} \rightarrow N_{IN}$ are source and target functions respectively. □

The PROV node inheritance graph consists of all the nodes and the subtype edges shown in Figure 4.2(a). The PROV node inheritance graph does not have the edge (activity, artifact), since *activity* is not a subtype of the abstract node *artifact*.

Similarly, Definition 4.12 represents the confidentiality inheritance graph where the set of nodes is same as the set of confidentiality level nodes in the type graph with abstract elements and the set of edges represents the subtype edges of the confidentiality inheritance graph.

Definition 4.12 (Confidentiality inheritance graph)**.** A directed graph $IC = (N_{IC}, E_{IC}, s_{IC}, t_{IC})$ is the inheritance graph of confidentiality level nodes where: $N_{IC} = C_{AG} = C_{TG} \cup AN$ is the set of nodes, $E_{IN} = C_{TG} \times AC$ is the set of edges, and $s_{IC}, t_{IC} : E_{IC} \rightarrow N_{IC}$ are source and target functions respectively. □

The PROV confidentiality inheritance graph consists of the set of nodes and subtype edges that have been illustrated in Figure 4.2(b).

We adopt the definition of inheritance type graphs in (Bardohl, Ehrig, De Lara et al. 2003). Combining the type graph with abstract elements and the inheritance graphs results in a type graph with inheritance, where each graph node and confidentiality level node has an inheritance clan as shown below (Definition 4.13). The inheritance clan of a node is the set of all the sub-nodes of that node.

Definition 4.13 (Type graph with inheritance)**.** A type graph with inheritance is $ITG = ((AG, 1), IN, IC, AN, AC)$ consists of a type graph with abstract elements and direct graph edges AG, the inheritance graphs IN and IC with the same set of graph nodes and confidentiality level nodes as AG, and the sets AN and AC which represent the abstract graph nodes and the abstract confidentiality level nodes respectively.

The inheritance clans are defined by:

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

□

In PROV-GTS, regarding the type graph of the PROV data model and according the above definition, we can assign a clan to each graph node and confidentiality level node as following:

- $clan_{IN}(node) = \{node,\ artifact,\ entity,\ activity,\ agent\}$
- $clan_{IN}(artifact) = \{artifact,\ entity,\ agent\}$
- $clan_{IN}(entity) = \{entity\}$
- $clan_{IN}(activity) = \{activity\}$
- $clan_{IN}(agent) = \{agent\}$
- $clan_{IC}(any) = \{any, plain, anonymous, restricted\}$
- $clan_{IC}(plain) = \{plain\}$
- $clan_{IC}(anonymous) = \{anonymous\}$
- $clan_{IC}(restricted) = \{restricted\}$

The type graphs with inheritance can be flattened to ordinary graphs, called closure of type graph with inheritance (Bardohl, Ehrig, De Lara et al. 2003), by adding normal graph edges to the type graph implied by the subtype edges in the inheritance graphs. In our system, the abstract graph nodes *node* and *artifact* can have an edge to each node in their inheritance clans.

Definition 4.14 (Closure of type graph with inheritance)**.** Given a type graph with inheritance $ITG = ((AG, 1), IN, IC, AN, AC)$ with a type graph with abstract elements $(AG, 1)$ where $AG = (N_{AG}, E_{AG}, C_{AG}, B_{AG}, s_{AG}, t_{AG}, c_{AG}, d_{AG})$, the closure of ITG is the graph $(\overline{AG}, 1)$ where $\overline{AG} = (N_{AG}, \overline{E_{AG}}, C_{AG}, \overline{B_{AG}}, \overline{s_{AG}}, \overline{t_{AG}}, \overline{c_{AG}}, \overline{d_{AG}})$ such that

- $\overline{E_{AG}} = E_{AG} \cup \{(n_1, n_2) \mid n_1 \in AN, n_2 \in clan_{IN}(n_1)\}$
- $\overline{s_{AG}}((n_1, n_2)) = n_1$
- $\overline{t_{AG}}((n_1, n_2)) = n_2$
- $\overline{B_{AG}} = B_{AG} \cup \{(r_1, r_2), r_1 \in AC, r_2 \in clan_{IC}(r_1)\}$
- $\overline{c_{AG}}((r_1, r_2)) = r_1$
- $\overline{d_{AG}}((r_1, r_2)) = r_2$

□

Now we can define typed graphs and the corresponding morphisms with respect to the type graph $\overline{AG}$.

Definition 4.15 (Typed graph with inheritance)**.** A typed graph $((G,d),type)$ over $ITG = ((AG,1),IN,IC,AN,AC)$ is an instance of $\overline{AG}$, i.e. $((G,d),type : G \longrightarrow \overline{AG})$ where $G = (N_G,E_G,C_G,B_G,s_G,t_G,c_G,d_G)$, $AG = (N_{AG},E_{AG},C_{AG},B_{AG},s_{AG},t_{AG},c_{AG},d_{AG})$ and $type' : G^d \rightarrow \overline{AG}$ consists of four morphisms $type'_N : N_{G^d} \rightarrow N_{AG}$, $type'_E : E_{G^d} \rightarrow \overline{E_{AG}}$, $type'_C : C_{G^d} \rightarrow C_{AG}$ and $type'_B : B_{G^d} \rightarrow \overline{B_{AG}}$ such that $type'_N \circ s_{G^d} = \overline{s_{AG}} \circ type'_E$, $type'_N \circ t_{G^d} = \overline{t_{AG}} \circ type'_E$, $type'_N \circ c_{G^d} = \overline{c_{AG}} \circ type'_B$ and $type'_C \circ d_{G^d} = \overline{d_{AG}} \circ type'_B$.

$$\begin{tikzcd}
E_{G^d} \arrow[r, "s_{G_d}"] \arrow[d, "type'_E"'] \arrow[dr, phantom, "="] & N_{G^d} \arrow[d, "type'_N"] \\
\overline{E_{AG}} \arrow[r, "\overline{s_{AG}}"] & N_{AG}
\end{tikzcd}
\qquad
\begin{tikzcd}
E_{G_d} \arrow[r, "t_{G_d}"] \arrow[d, "type'_E"'] \arrow[dr, phantom, "="] & N_{G_d} \arrow[d, "type'_N"] \\
\overline{E_{AG}} \arrow[r, "\overline{t_{AG}}"] & N_{AG}
\end{tikzcd}$$

$$\begin{tikzcd}
B_{G_d} \arrow[r, "c_G"] \arrow[d, "type'_B"'] \arrow[dr, phantom, "="] & N_{G_d} \arrow[d, "type'_N"] \\
\overline{B_{AG}} \arrow[r, "\overline{c_{AG}}"] & N_{AG}
\end{tikzcd}
\qquad
\begin{tikzcd}
B_{G_d} \arrow[r, "d_G"] \arrow[d, "type'_B"'] \arrow[dr, phantom, "="] & C_{G_d} \arrow[d, "type'_C"] \\
\overline{B_{AG}} \arrow[r, "\overline{d_{AG}}"] & C_{AG}
\end{tikzcd}$$

□

A morphism $G \longrightarrow H$ over the type graph with inheritance is the homomorphism that maps each node x in graph G to a node y in graph H as long as type of y is in the clan of type of x.

Definition 4.16 (Typed graph with inheritance morphism)**.** The morphism $f : G \longrightarrow H$ between two typed graphs with inheritance $G = (\mathcal{G}, d_1)$ and $H = (\mathcal{H}, d_2)$ is $f' : \mathcal{G}^{d_1} \rightarrow \mathcal{H}^{d_2}$ which consists of four functions: $f'_N : N_{\mathcal{G}^{d_1}} \rightarrow N_{\mathcal{H}^{d_2}}$, $f'_E : E_{\mathcal{G}^{d_1}} \rightarrow E_{\mathcal{H}^{d_2}}$, $f'_C : C_{\mathcal{G}^{d_1}} \rightarrow C_{\mathcal{H}^{d_2}}$ and $f'_B : B_{\mathcal{G}^{d_1}} \rightarrow B_{\mathcal{H}^{d_2}}$ such that $f'_N \circ s_{\mathcal{G}^{d_1}} = s_{\mathcal{H}^{d_2}} \circ f'_E$, $f'_N \circ t_{\mathcal{G}^{d_1}} = t_{\mathcal{H}^{d_2}} \circ f'_E$, $f'_N \circ c_{\mathcal{G}^{d_1}} = c_{\mathcal{H}^{d_2}} \circ f'_B$, $f'_C \circ d_{\mathcal{G}^{d_1}} = d_{\mathcal{H}^{d_2}} \circ f'_B$, for each $x \in N_{\mathcal{G}^{d_1}}$: $(type'_{N_{\mathcal{H}^{d_2}}} \circ f'_N)(x) \in clan_{IN}(type'_{N_{\mathcal{G}^{d_1}}}(x))$ and for each $y \in C_{\mathcal{G}^{d_1}}$: $(type'_{C_{\mathcal{H}^{d_2}}} \circ f'_C)(y) \in clan_{IC}(type'_{C_{\mathcal{G}^{d_1}}}(y))$.

$$\begin{tikzcd}
E_{\mathcal{G}^{d_1}} \arrow[r, "s_{\mathcal{G}^{d_1}}"] \arrow[dd, "f'_E"'] \arrow[ddr, phantom, "="] & N_{\mathcal{G}^{d_1}} \arrow[dd, "f'_N"] \arrow[dr, "type'_{N_{\mathcal{G}^{d_1}}}"] & \\
 & & N_{AG} \\
E_{\mathcal{H}^{d_2}} \arrow[r, "s_{\mathcal{H}^{d_2}}"] & N_{\mathcal{H}^{d_2}} \arrow[ur, "type'_{N_{\mathcal{H}^{d_1}}}"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
E_{\mathcal{G}^{d_1}} \arrow[r, "t_{\mathcal{G}^{d_1}}"] \arrow[dd, "f'_E"'] \arrow[ddr, phantom, "="] & N_{\mathcal{G}^{d_1}} \arrow[dd, "f'_N"] \arrow[dr, "type'_{C_{\mathcal{G}^{d_1}}}"] & \\
 & & C_{AG} \\
E_{\mathcal{H}^{d_2}} \arrow[r, "t_{\mathcal{H}^{d_2}}"] & N_{\mathcal{H}^{d_2}} \arrow[ur, "type'_{C_{\mathcal{H}^{d_1}}}"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
B_{\mathcal{G}^{d_1}} \arrow[r, "c_{\mathcal{G}^{d_1}}"] \arrow[dd, "f'_B"'] \arrow[ddr, phantom, "="] & N_{\mathcal{G}^{d_1}} \arrow[dd, "f'_N"] \arrow[dr, "type'_{N_{\mathcal{G}^{d_1}}}"] & \\
 & & N_{AG} \\
B_{\mathcal{H}^{d_2}} \arrow[r, "c_{\mathcal{H}^{d_2}}"] & N_{\mathcal{H}^{d_2}} \arrow[ur, "type'_{N_{\mathcal{H}^{d_1}}}"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
B_{\mathcal{G}^{d_1}} \arrow[r, "d_{\mathcal{G}^{d_1}}"] \arrow[dd, "f'_B"'] \arrow[ddr, phantom, "="] & C_{\mathcal{G}^{d_1}} \arrow[dd, "f'_C"] \arrow[dr, "type'_{C_{\mathcal{G}^{d_1}}}"] & \\
 & & C_{AG} \\
B_{\mathcal{H}^{d_2}} \arrow[r, "d_{\mathcal{H}^{d_2}}"] & C_{\mathcal{H}^{d_2}} \arrow[ur, "type'_{C_{\mathcal{H}^{d_1}}}"'] &
\end{tikzcd}$$

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

Regarding the two categories $Graph_p$ and $Graph$ of provenance graphs as objects and respectively partial and total graph morphisms as arrows and by using the notion of slice categories, we can define the category of typed provenance graphs over the type graph ITG.

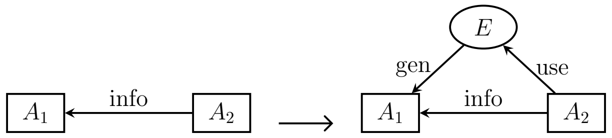
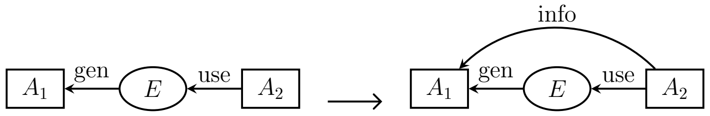
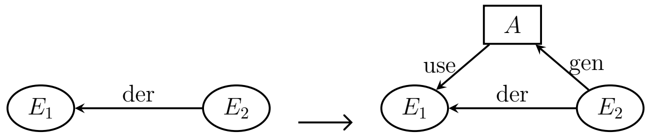
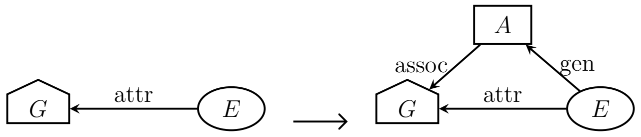
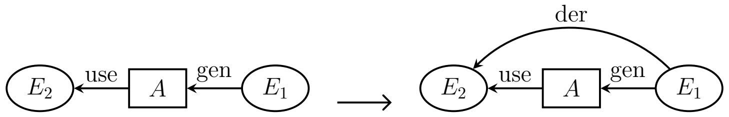
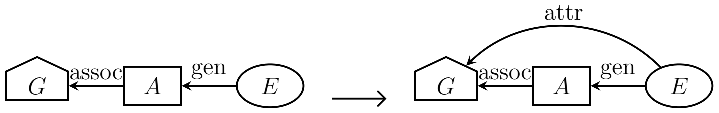
Definition 4.17 (Category of typed graphs with inheritance)**.** Let $Graph_{pITG} = Graph_p/ITG$ and $Graph_{ITG} = Graph/ITG$ be two slice categories represent the categories of typed graphs with inheritance with partial and total morphisms, respectively. □

4.6 PROV Data Model Properties

The graph transformation rules in PROV-GTS are of two types: creation rules and deletion rules. The creation rules insert nodes and/or edges to the graph as necessary in order to preserve the graph integrity, while each deletion rule removes an edge connected to a restricted node from the graph. 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 (prov1-6, see Table 4.1) from the PROV data model. These properties represent genuine implications of PROV, they are not transformation rules. But they represent the building blocks of the graph transformation rules in PROV-GTS. Each of prov1,2,4 is constructed from the PROV model inference rules Inference 5, Inference 6 and Inference 13 respectively (Cheney, Missier, Moreau et al. 2013) while prov3 came from the fact that the derivation (der) edge implies the existence of an activity that connects the generated and used entities (Moreau and Missier 2013). In addition, (prov1-4) represent what can be inferred from the provenance graphs based on PROV model properties, i.e. for the PROV property $C_i \longrightarrow E_i$, the nodes and edges that are in E_i but not in C_i (E_i/C_i) can be created when C_i exists. These properties can be used to construct creation graph transformation rules. While prov5 and prov6, inspired by prov3 and prov4, are only used to construct additional conditions that will be attached to some of the deletion rules, as we will see in the next section. They cannot be used as creation rules as there is no inference rules in PROV model for inferring *der* edges from *gen-use* patterns or *attr* edges from *gen-assoc* patterns. So, the properties prov1-6 are used to build the required conditions for the deletion rules while only the properties prov1-4 can be used to construct the creation rules.

Definition 4.18 (PROV property)**.** A PROV property provi is $p_i : C_i \longrightarrow E_i$ in $Graph_{ITG}$ for $i = 1..6$ where C_i and E_i are respectively premise and conclusion of PROV properties, and p_i is the obvious inclusion morphism. □

TABLE 4.1: PROV data model properties

Property	**Graph Patterns** ($C_i \longrightarrow E_i$)
prov1 (Cheney, Missier, Moreau et al. 2013, Inference 5)	 A_1 info A_2 E gen use A_1 info A_2
prov2 (Cheney, Missier, Moreau et al. 2013, Inference 6)	 A_1 gen E use A_2 info A_1 gen E use A_2
prov3 (Moreau and Missier 2013)	 E_1 der E_2 A use gen E_1 der E_2
prov4 (Cheney, Missier, Moreau et al. 2013, Inference 13)	 G attr E A assoc gen G attr E
prov5 (Moreau and Missier 2013)	 E_2 use A gen E_1 der E_2 use A gen E_1
prov6 (Cheney, Missier, Moreau et al. 2013, Inference 13)	 G assoc A gen E attr G assoc A gen E

4.7 Conditional Graph Transformation Rules

Algebraic approaches can be extended by additional application conditions such as existence and non-existence of certain nodes and edges (Habel, Heckel and Taentzer 1996) as well as conditions that repeated frequently in the host graph known as nested conditions (Habel and Pennemann 2005). An application condition matching requires the existence of a certain sub-structure in the target graph. The elements of this matching do not need to be distinct to each other and to be unrelated to the other elements in the graph (Rensink 2004).

An application conditions is a positive or negative constraint that can be used to ensure the existence or non-existence of elements and relations in the graph before applying the rule. The rule is applicable if all its application conditions match (Heckel 2006). In Figure 4.3, the left-hand side L may have many application conditions K_i, when the matching of L is computed; this matching should be expanded to all the application conditions connected to L. The positive conditions specify what should be matched while the negative conditions specify what have to be mismatched. In addition, there are nested conditions which must be matched universally rather than existentially. The rule is applied on a graph when its match in that graph satisfied all the application conditions of the rule.

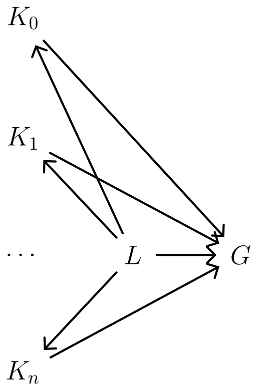


Figure 4.3: Application conditions: the left-hand side (L) is connected to a set of application conditions ($K_0, K_1, \dots K_n$). Before the rule is applied, the match of L in graph G must satisfy all the rule's application conditions

Some graph properties, such as the universally qualified conditions, cannot be expressed using the above positive and negative application conditions. For example, a rule to be applicable may require the host graph to satisfy several occurrences of the same graph pattern such as the multiple events in PROV data model that are described in Section 1.3 of Chapter 1.

The creation rules require negative application conditions to avoid infinite number of rule applications. Similarly, the deletion rules need universal-existential conditions to deal with nodes that have multiple incoming and/or outgoing edges of the same type.

To this end, the simple rules, which consist of LHS and RHS in the form of $r : L \longrightarrow R$, require to be extended by additional application conditions (Habel and Pennemann 2005; Ehrig, Ehrig, Prange et al. 2006b).

Definition 4.19 (Application conditions)**.** Given a simple rule $r : L \longrightarrow R$, an application condition is either a total morphism $c : u \longrightarrow e$ in $Graph_{ITG}$ representing a universal-existential condition or a single graph (nac) representing a negative application condition (NAC) where $L \rightarrow u$ and $L \rightarrow nac$ are total morphisms. □

Now, a simple rule and sets of application conditions can be combined to construct a conditional rule.

Definition 4.20 (Conditional rule)**.** A conditional rule $\nabla = (r, C, T, M, N)$ consists of a simple rule $r : L \longrightarrow R$ where L and R are left- and right-hand sides of the rule respectively, a set of universal-existential application conditions $C = \{c_0, c_1 \ldots\}$, a set of negative application conditions $T = \{nac_0, nac_1, \ldots\}$, and two sets of total morphisms $M = \{m_0, m_1, \ldots\}$ and $N = \{n_0, n_1 \ldots\}$ where $c_i = u_i \longrightarrow e_i$, $m_i = L \longrightarrow u_i$ and $n_j = L \longrightarrow nac_j$ for $i = 0 \ldots |C| - 1$ and $j = 0 \ldots |T| - 1$. All the morphisms r, c_i, m_i and n_j are monomorphisms. □

The conditional rule $\nabla = (r : L \rightarrow R, C, T, M, N)$ is applicable on a PROV graph G if the match $f : L \longrightarrow G$ satisfies the sets of conditions C and T.

Definition 4.21 (Conditional rule satisfaction)**.** Let $\nabla = (r : L \longrightarrow R, C, T, M, N)$ be a conditional rule and G be a PROV graph, then the match $f : L \longrightarrow G$ in the diagrams below satisfies each $(c : \mathrm{u} \longrightarrow \mathrm{e}) \in C$ if for each $g : u \longrightarrow G$ there exist at least one $h : e \longrightarrow G$ such that $g = h \circ c$ and satisfies each $(n : \mathrm{L} \longrightarrow \mathrm{nac}) \in N$ if there does not exist a total morphism $p : nac \longrightarrow G$ with $p \circ n = f$.

$$\begin{tikzcd}
L \arrow[rr, "f"] \arrow[dd, "m"'] \arrow[dr, phantom, "="] & & G \\
& (\forall) \arrow[dr, phantom, "="] & \\
u \arrow[rr, "c"] \arrow[uurr, "g"] & & e \arrow[uu, "h", "(\exists)"']
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[r, "f"] \arrow[d, "n"'] & G \\
nac \arrow[ur, "p"', "/" marking] &
\end{tikzcd}$$

□

4.8 Graph Integrity and Rule Construction

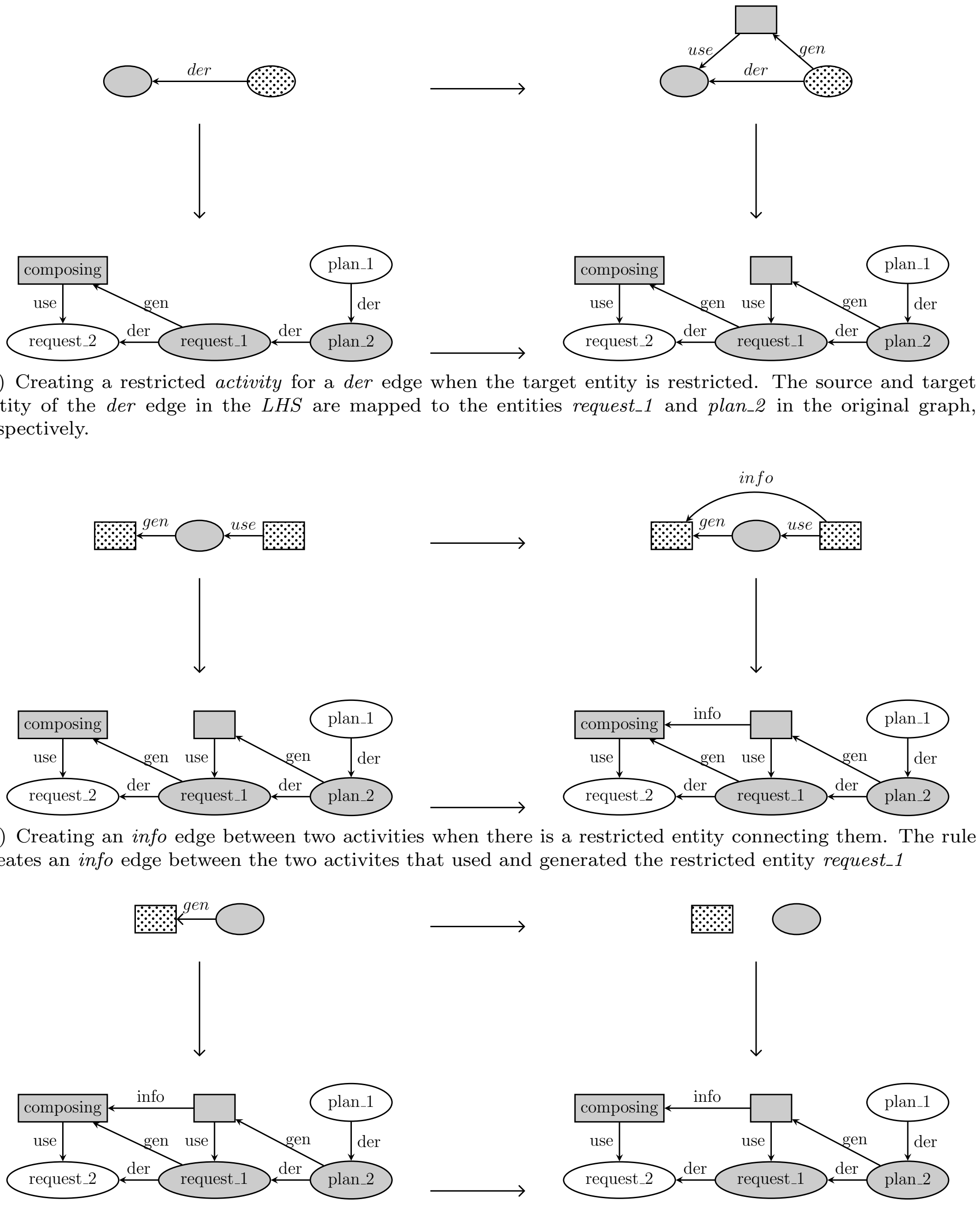
Deleting the edges connected to restricted nodes may result in omitting non-relevant information, i.e. false independency. In the graph of Figure 4.4, we use a subgraph of the RideShare graph to show why creating some nodes and edges are important before deleting the edges connected to restricted nodes. For example, if we apply the *gen* deletion rule in Figure 4.4(c) before the *activity* creation rule in Figure 4.4(a) or the *info* creation rule in Figure 4.4(b), then we will no longer be able to infer the *info* edge, 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.

Regarding the situation above, the deletion rules should have enough graph constraints as conditions so that edges are not deleted before ensuring that all relevant dependencies have been preserved. For example, the deletion rule in Figure 4.4(c) can be rewritten such that it deletes the *gen* edge only if it is part of a *use-gen* pattern with an *info* edge, as shown in Figure 4.5. In this case, the edge *gen* is deleted only when the *info* edge exists. Since this *info* edge, if it does not explicitly exist, can be inferred from the *use-gen* sequence according to prov2, so a graph transformation rule can be defined to create the *info* edge if it does not exist, as shown Figure 4.4(b). The rule requires a negative application condition (NAC) in the form of $\neg A$ where A in our example is the edge *info*, which tells the system to create the edge only when it does not exist.

4.8.1 Creation rules and negative application conditions

The creation rules generate the edges and nodes that do not exist in the graph and are required to satisfy all the conditions of the deletion rules. This creation process guarantees that the relations between the nodes adjacent to the restricted nodes and the paths that they are part of have been preserved. But it is also important to avoid adding the same nodes and edges again and again to the provenance graphs. To do so, we need an appropriate negative application condition NAC for each creation rule to prevent non-deterministic rule applications. The LHS and RHS of each creation rule shown in Figure 4.4(a) and Figure 4.4(b) represents monomorphisms (monos) to premise and conclusion of the properties prov3 and prov2, respectively. A morphism between two graphs is mono if the functions that map the set of nodes and the set of edges from the source graph to the corresponding sets in the target graph are injective functions. Each creation rule is accompanied by a negative application condition NAC to avoid infinite number of rule applications, as shown in the two creation rules in Figure 4.6.

We will show how the rules are constructed gradually starting from simple rules, which consist of LHS and RHS only, and then we move on for more sophisticated rules containing negative application conditions and universal-existential conditions. Since the rules



(a) Creating a restricted *activity* for a *der* edge when the target entity is restricted. The source and target entity of the *der* edge in the *LHS* are mapped to the entities *request_1* and *plan_2* in the original graph, respectively.

(b) Creating an *info* edge between two activities when there is a restricted entity connecting them. The rule creates an *info* edge between the two activites that used and generated the restricted entity *request_1*

(c) Deleting a *gen* edge when the generated entity is restricted. The *gen* edge in the LHS of the rule matches the *gen* edge between the restricted entity *request_1* and the activity *composing*.

FIGURE 4.4: Preserving relations before deleting edges: generating an activity (a) following by creating of an *info* edge (b) before deleting the *gen* edge (c)

will be constructed gradually, only the final rules will be given distinguished names. For example, the *info* creation rule that is illustrated in Figure 4.6(b) represents a final rule called ④***crtInfUseGen***, as shown in the label of the figure, where ④ and ***crtInfUseGen*** represent the rule number and the rule name respectively. But the *activity* creation rule shown in Figure 4.6(a) is not a final rule, it needs further construction as we will see in the upcoming sections.

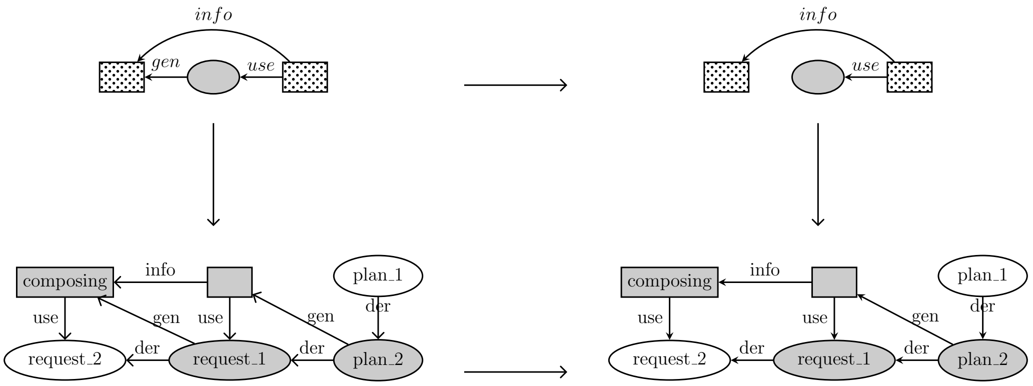
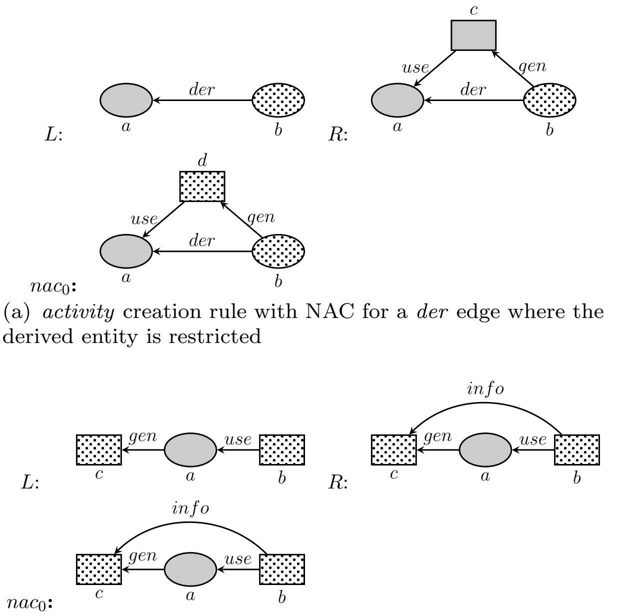


FIGURE 4.5: Checking for *info* edge before deleting *gen* edge: a *gen* edge connected to a restricted entity can only be deleted with rule when there is an incoming *use* edge and an *info* edge connected the activity that used the restricted entity and the activity that generated it.



(a) *activity* creation rule with NAC for a *der* edge where the derived entity is restricted

(b) *info* creation rule with NAC between two activities where there exist a restricted entity that has been used by the first activity and generated by the second ((4)***crtInfUseGen***).

FIGURE 4.6: Creation rules with negative application conditions (NACs)

4.9 Dealing with Nodes with Multiple Similar Events in the PROV Data Model

The PROV data model allows multiple incoming and outgoing edges of the same type to and from a node (Cheney, Missier, Moreau et al. 2013). The simple transformation rules that consist only of LHS and RHS are not enough to check for multiple edges. Regarding the OnlinePost graph, suppose another user shared the same post on a social media website which resulted in another entity *post_1* as shown in Figure 4.7. The edge *gen* between the entity *post_0* and the activity *sharing_0* can be deleted since the *info*

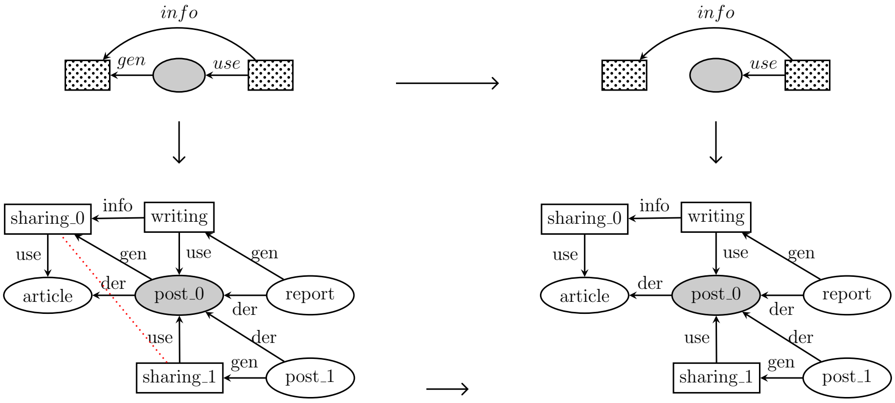
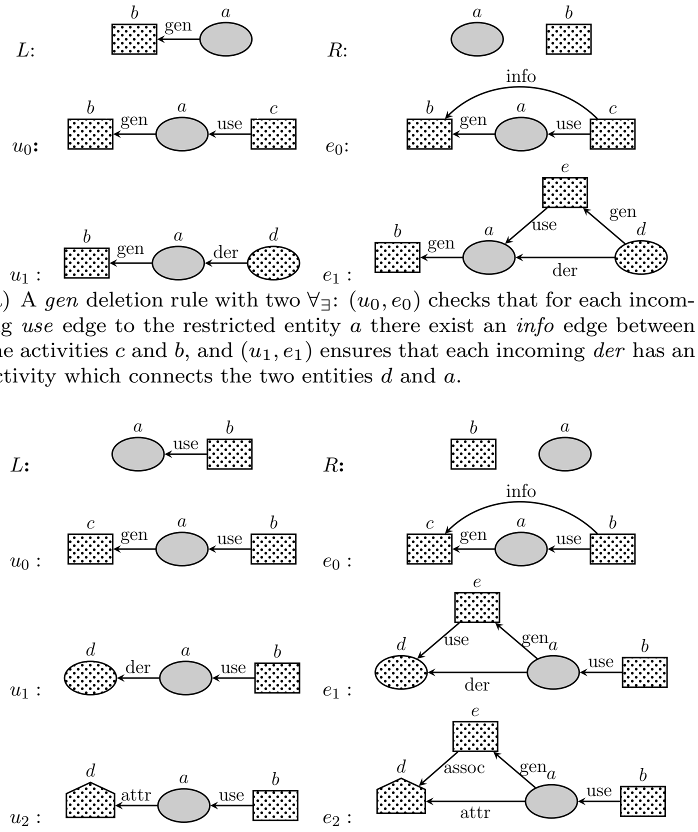


FIGURE 4.7: Transforming a provenance graph with multiple edges: there is two incoming *use* edges to the restricted entity *post_0* and one outgoing *gen*, deleting this *gen* will prevent us from inferring the *info* between the activities *sharing_1* and *sharing_0* which causes false independency

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* (shown as a red dotted line). To tackle this issue, we need to check this kind of patterns universally rather than existentially using universal-existential conditions of the form $\forall A : \exists B$. In our case, A and B represent the *use-gen* pattern and the corresponding *info* edge respectively. It states that "for each *use-gen* pattern, there exists an *info* edge between the informed and informant activities", as shown in the transformation rules of Figure 4.8, where u_i and e_i respectively represent the universal and existential parts of the i's condition for $(i = 0, 1, \dots)$. In Figure 4.7, the condition that ensures the existence of an *info* edge if the restricted entity has incoming *use* edge has been attached to the LHS and RHS of the basic *gen* deletion rule, which will be satisfied existentially. To represent this kind of constraints as universal-existential conditions, we must attach them to the rule as conditions of the form (u_i, e_i). As shown in Figure 4.8(a), the previous condition has been attached to the *gen* deletion rule as the condition (u_0, e_0) which has been constructed based on prov2 and checks that for each incoming *use* edge to the restricted entity there exists an *info* edge. In addition, another condition (u_1, e_1) is required to ensure that for each incoming *der* there is an activity node according to the PROV property prov3. Similarly, we construct another conditional rule to delete the *use* edge as illustrated in Figure 4.8(b). The rule is accompanied by the condition (u_0, e_0) to ensure that for each outgoing *gen* edge from the restricted entity there is an *info*. In addition, there are two conditions (u_1, e_1) and (u_2, e_2), which respectively check for existence of activities for outgoing *der* and *attr* edges.



(a) A *gen* deletion rule with two $\forall_\exists$: (u_0, e_0) checks that for each incoming *use* edge to the restricted entity a there exist an *info* edge between the activities c and b, and (u_1, e_1) ensures that each incoming *der* has an activity which connects the two entities d and a.

(b) A *use* deletion rule with three $\forall_\exists$ conditions: (u_0, e_0) checks that for each outgoing *gen* edge from the restricted entity a there exist an *info* edge between the activities c and b, (u_1, e_1) ensures that each outgoing *der* has an activity which connects the two entities a and d, and (u_2, e_2) confirms the existence of an activity for each outgoing *attr* which connects the restricted entity a to an agent d.

FIGURE 4.8: Two graph transformation rules with universal-existential conditions ($\forall_\exists$)

4.10 Path Preservation

The aim of PROV-GTS rules is to isolate the restricted nodes from the graph by deleting their incident edges one by one up to isolation. The graph transformation rules must have enough conditions that ensure preserving relations between the nodes adjacent to the source and target of the deleting edge. This preservation is important to maintain the integrity of provenance information. In addition, since provenance represents the lineage of an artifact or thing, the paths that lead to a specific state is important when querying over paths, such that deleting an edge will not affect the retrieval of the nodes adjacent to the source and the target of the deleted edge via path traversal queries (Blaustein, Chapman, Seligman et al. 2011). In PROV-GTS, creating nodes and edges as well as node anonymization lead to path preservation. It guarantees that for each path between two plain nodes in the original graph, there will be still a path connecting them in the transformed graph.

However, not all paths can be maintained based on this preservation. For example, an

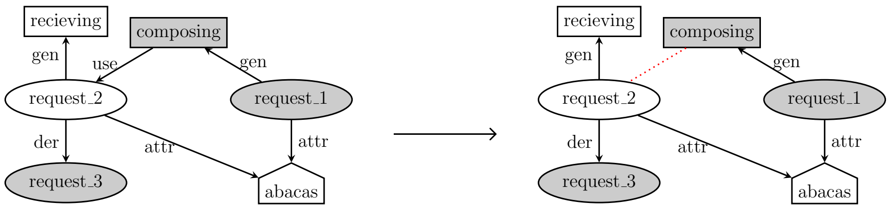


FIGURE 4.9: Deleting an edge results in cutting paths: deleting the *use* edge between the activity *composing* and the entity *request_2* results in cutting the paths that connect each of *request_1* and *composing* to the nodes *request_2*, *request_3* and *receiving*.

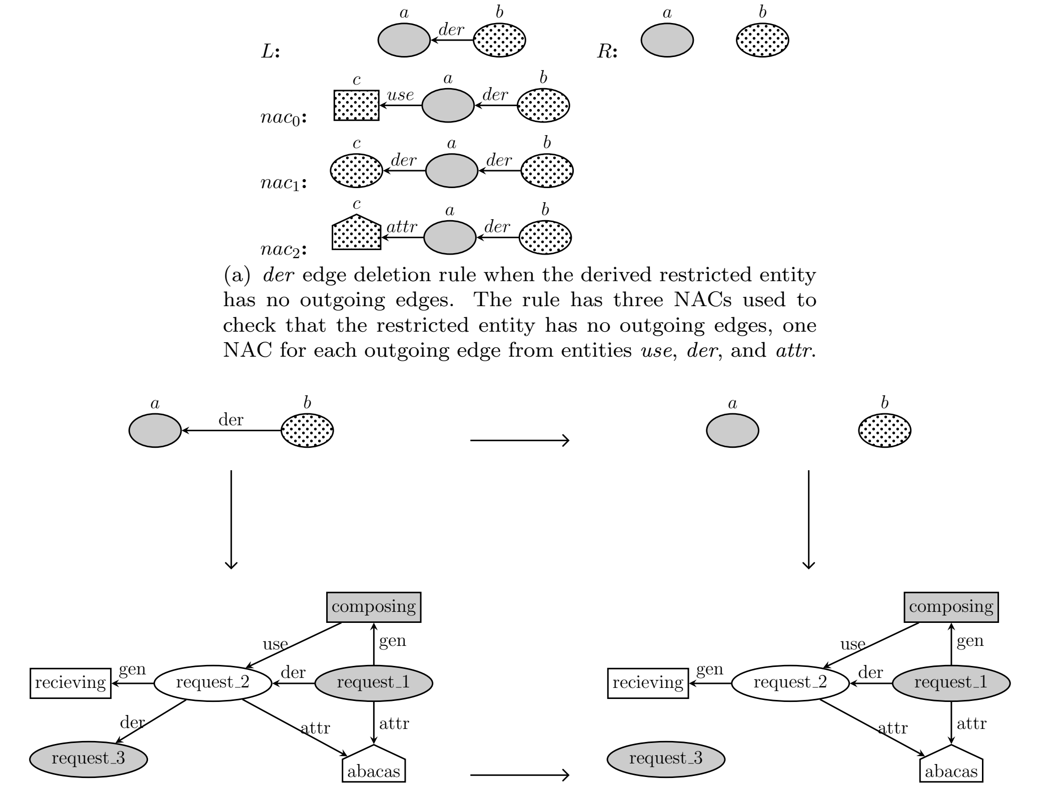
activity can be inferred from the *der* edge (prov4) but the opposite is not true, that is, the *der* edge cannot be inferred from the *gen-use* pattern (prov5). In this case, deleting the edges in *gen-use* pattern, when the activity is restricted, will cut the path that linked the two entities and hence any path that includes the restricted activity. In Figure 4.9, which shows a subgraph of the RideShare graph, deleting *use* edge between *composing* and *request_2* nodes will cut all the paths from *request_1* to each of the nodes *request_2*, *request_3* and *receiving*. So, preventing deleting the *gen* and *use* edges is important to maintain any path that consists of these two edges, and as a result the restricted activity cannot be fully isolated from the graph. In this case, the activity *composing* will be anonymized, as described in Section 3.4 of Chapter 3.

4.11 Non-determinism

In order to avoid non-determinism in PROV-GTS, we must ensure that rules are independent of each other, and that the output of a transformation is not dependent on the order in which rules are applied. With the rules as presented so far, these properties do not always hold. The key to ensure PROV-GTS's determinism is by embedding appropriate positive conditions in the left-hand side of the rules to prevent unnecessary and redundant rule applications, adding negative conditions to the transformation rules to curb infinite numbers of rule applications, or changing nodes' confidentiality level values to avoid inconsistency between edge deletion rule applications. By embedding positive conditions, we mean expanding the left-hand side of the rule with extra graph constraints. The same graph constraints will be added to the right-hand side of the rule as well such that the result of graph transformation remains the same.

4.11.1 Expanding the LHS

The edges incident to restricted nodes that have no incoming or outgoing edges can immediately be deleted since no false independency is caused and no path is broken. For instance, the restricted entity *request_3* has no outgoing edges so it is safe to delete the incoming *der* connected to it. In Figure 4.10(a), a *der* deletion rule is defined and



(a) *der* edge deletion rule when the derived restricted entity has no outgoing edges. The rule has three NACs used to check that the restricted entity has no outgoing edges, one NAC for each outgoing edge from entities *use*, *der*, and *attr*.

(b) Deleting *der* edge between the entity *request_2* and the restricted entity *request_3* which has no outgoing edges

FIGURE 4.10: Deleting *der* edge when the *entity* has no outgoing edges

applied to the graph resulting in deleting the *der* that connected *request_2* to *request_3* as shown in Figure 4.10(b). Certainly, this deletion rule requires three negative application conditions to ensure that the rule is applied only when the restricted entity has no outgoing *info*, *der* and *attr* edges. These conditions are depicted as the negative application conditions nac_0, nac_1, and nac_2.

However, this edge deletion rule application may be preceded by an activity creation rule application for the *der* edge, as shown in Figure 4.11, which creates an activity connecting the entities *request_2* and *request_3*. This creation rule application is unnecessary since *request_3* does not have any outgoing edge hence no information will be lost. As a result, since the created activity is also a restricted node, a number of redundant rule applications are needed in order to process it.

To avoid redundant activity creation rule applications, the LHS of that rule is expanded by adding an outgoing edge to the restricted entity such that the activity is created only when the restricted entity has an outgoing edge. Since entities can have three types of outgoing edges, *gen*, *der* and *attr*, three transformation rules are needed, one for each outgoing edge type. These three creation rules with appropriate negative application conditions are shown in Figure 4.12(a), Figure 4.12(b) and Figure 4.12(c) respectively.

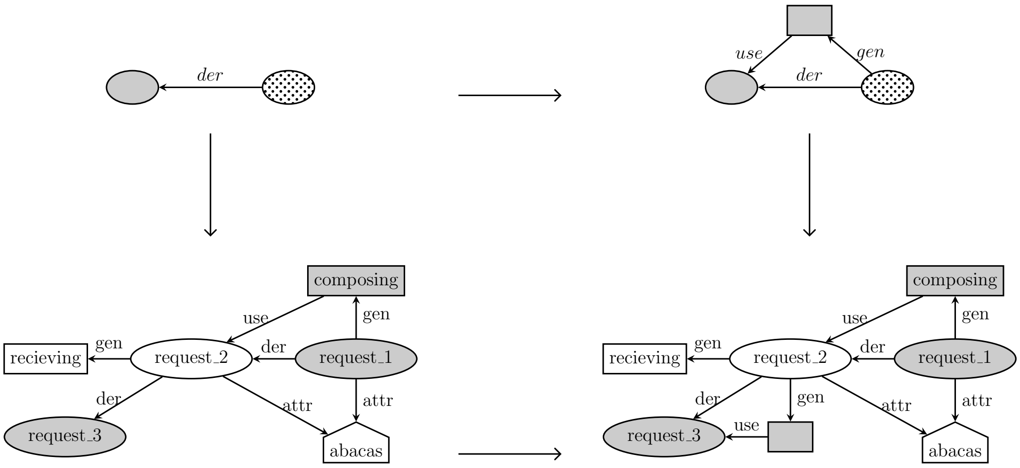


FIGURE 4.11: Unnecessary activity creation rule application: a *der* edges can be deleted by the deletion rule presented in Figure 4.10(a) if the target entity is restricted and has no outgoing edges, there is no need to create an activity by the creation rule that shown in Figure 4.6(a).

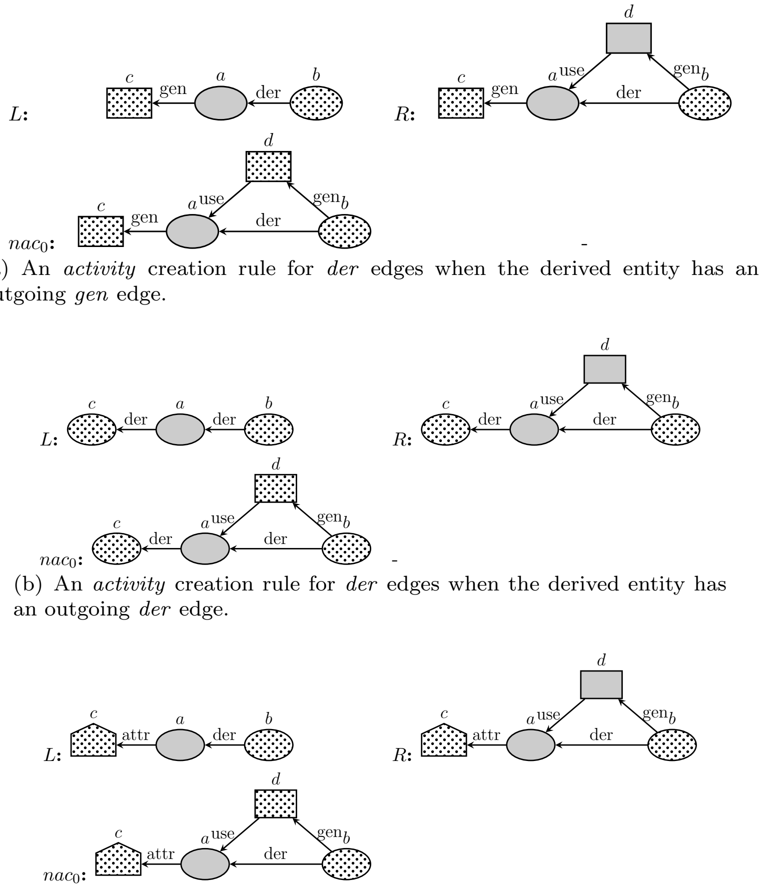
4.11.2 Negative application conditions for deletion rules

In addition to the above *der* deletion when it is located at the boundary of the graph, we can delete the *der* edge when it is in the middle of the graph. In this case we need to avoid false independency and preserve the paths between the nodes adjacent to the source and target of the *der* edge. To do so, we need to ensure that the activity node which connects the two entities exists. For example, the *der* edge between *post_0* and *article* entity nodes from OnlinePost graph can be deleted since the activity *sharing_0* presents, as illustrated in Figure 4.13.

However, if the *gen* edge from *post_0* to *sharing_0* is deleted, as shown in Figure 4.14, then we will no longer be able to delete the *der* edge until another activity between the entities *post_0* and *article* is created. Subsequent deletions of *gen* and creations of the activity result in an inconsistent situation. To prevent this, we need to avoid deleting the *gen* edge if it is part of an outgoing *der* or *attr* edge by adding negative application conditions to the *gen* deletion rule, as shown in Figure 4.15 with the conditions nac_0 and nac_1.

4.11.3 Changing nodes' confidentiality levels

We can construct another *gen* deletion rule but this time when the activity is restricted, as shown in Figure 4.16(a). By applying this rule on the running example RideShare, we can delete the *gen* edge between the entity *request_1* and the activity *composing* as shown in Figure 4.16(b). Since the entity *request_1* is restricted, the same issue above emerges, i.e. preventing the *der* edge between *request_1* and *request_2* to be deleted. To tackle this issue, we avoid deleting the *gen* edge this way when the source entity is



(a) An *activity* creation rule for *der* edges when the derived entity has an outgoing *gen* edge.

(b) An *activity* creation rule for *der* edges when the derived entity has an outgoing *der* edge.

(c) An *activity* creation rule for *der* edges when the derived entity is restricted and has an outgoing *attr* edge.

FIGURE 4.12: Solving inconsistency between deletion and creation rules by creating an activity for *der* edge only when the target entity is restricted and has outgoing edges. Three rules are provided one for each outgoing edge types *gen*, *der*, and *attr*.

restricted, so that the new rule deletes the *gen* edge only when the entity is plain as shown in the modified rule in Figure 4.17.

4.12 Merging Rules and Conditions Using Abstract nodes

As mentioned before in defining PROV type graph with inheritance, abstract node types will be used to merge conditions and rules. In Figure 4.18, four rules are presented which came from merging the conditions and the rules that have previously been defined. Figure 4.18(a) represent the rule in Figure 4.10(a) after merging the three negative conditions of the rule in one conditions using the abstract node *node*. The three activity creation rules defined in Figures 4.12(a), 4.12(b), 4.12(c) have been combined in one single rule as shown in Figure 4.18(b). The two negative application conditions nac_0 and nac_1 of the *gen* deletion rule shown in Figure 4.15 are merged using the abstract node *artifact* as illustrated in Figure 4.18(c). Similarly, the two universal-existential conditions (u_0, e_0) and (u_1, e_1) of the rule in Figure 4.17 are merged in one condition as

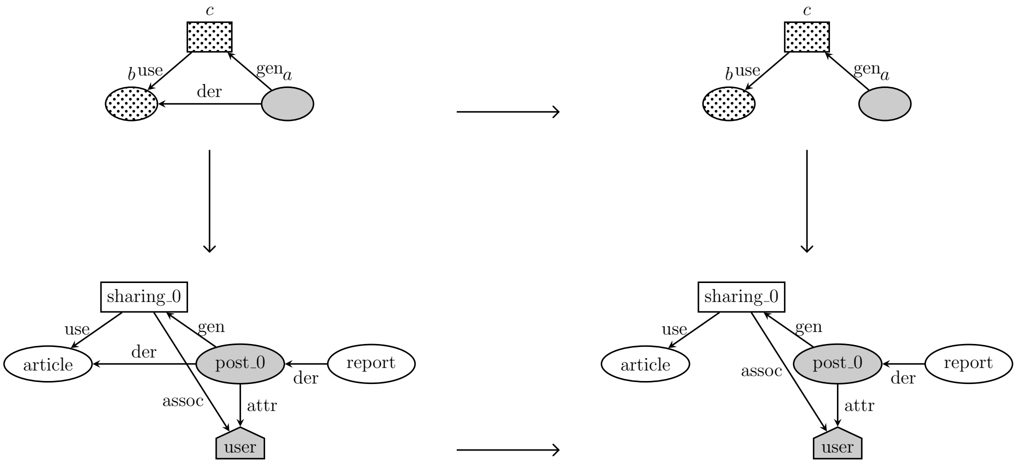


FIGURE 4.13: Deleting a *der* edge in the middle of the graph when the activity that connecting the source and target entities exists

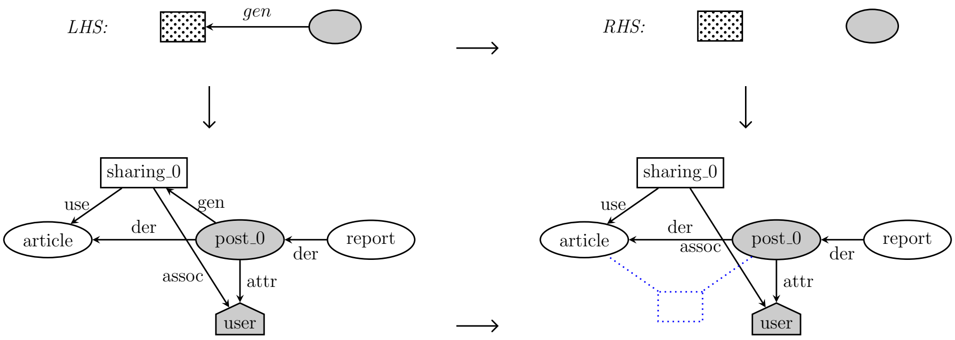


FIGURE 4.14: Inconsistent situation after deleting the *gen* edge which will results in recreating an activity between the entities *post_0* and *article*.

illustrated in Figure 4.18(d). All these rules are final constructed rules with designated names in the figures' labels.

A total of twenty-four graph transformation rules have been constructed based of the construction methodology presented in this chapter. The complete set of the constructed graph transformation rules is presented in Appendix B with a brief description for each part of each rule.

4.13 Summary

In this chapter, the formal description of the PROV data model graphs and graph morphisms is presented. The basic graph transformation rules are defined followed by the conditional rules with negative and universal-existential conditions. Based on the formal description of the PROV data model graphs and conditional rules, a construction methodology is defined and used to construct the graph transformation rules of PROV-GTS which are capable of obscuring restricted provenance information while maintaining

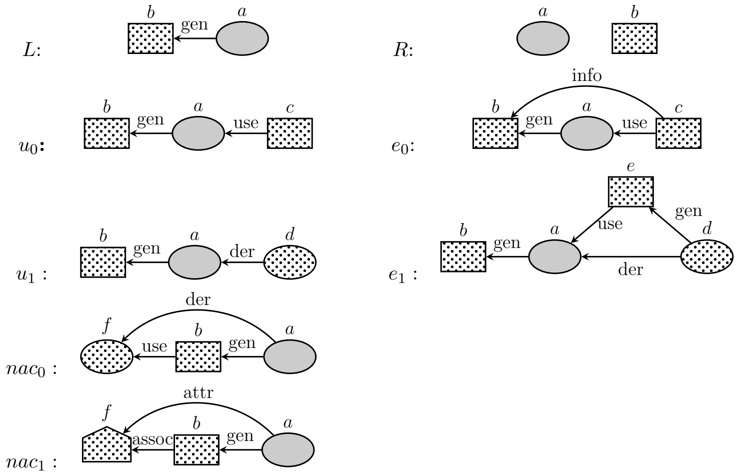
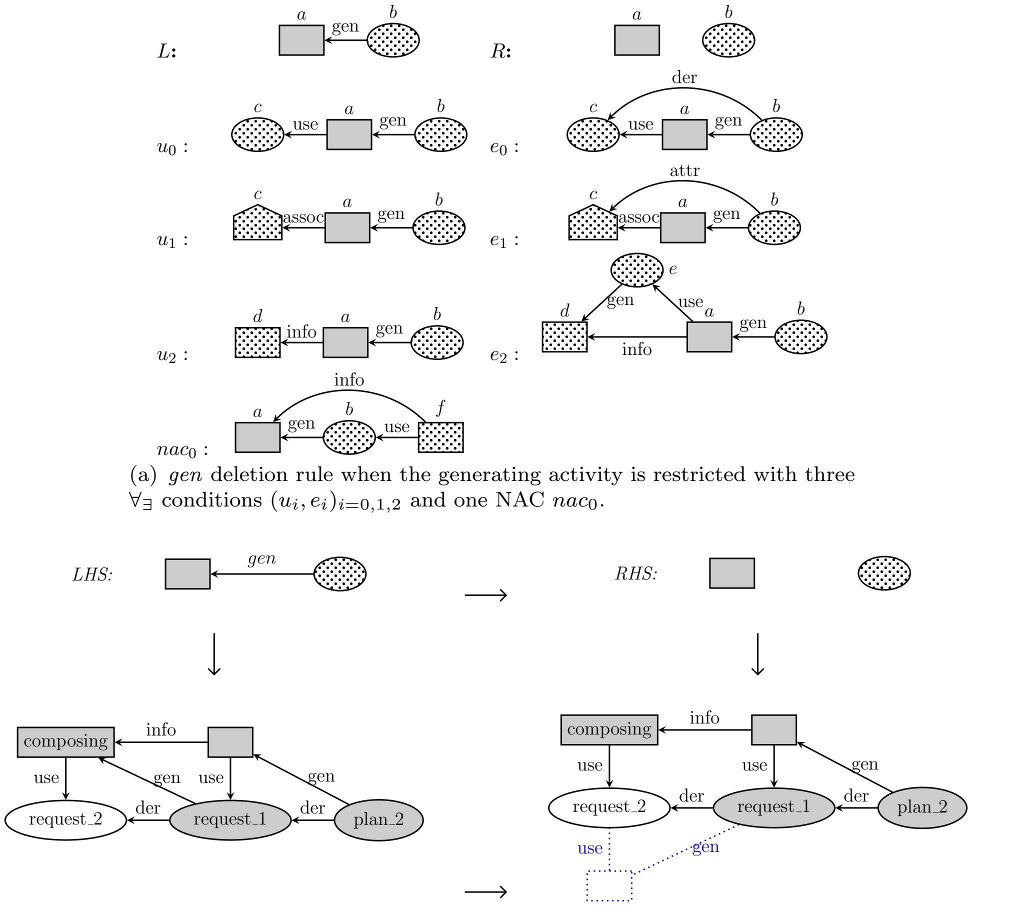


Figure 4.15: Updating the *gen* deletion rule depicted in Figure 4.8(a) by attaching NACs: nac_0 to ensure that there is no entity with a *der* edge from the restricted entity a and a *use* from the activity b, likewise, nac_1 checks for non-existence of an agent that has the incoming edges *attr* and *assocc* from the restricted entity a and the activity b, respectively.



(a) *gen* deletion rule when the generating activity is restricted with three $\forall_\exists$ conditions $(u_i, e_i)_{i=0,1,2}$ and one NAC nac_0.

(b) Applying the *gen* deletion rule from (a) when the generated entity is also restricted results in an inconsistent situation when the activity between the nodes *request_1* and *request_2* needs to be recreated.

Figure 4.16: Deleting *gen* edge when both the activity and the entity are restricted may result in an inconsistent situation

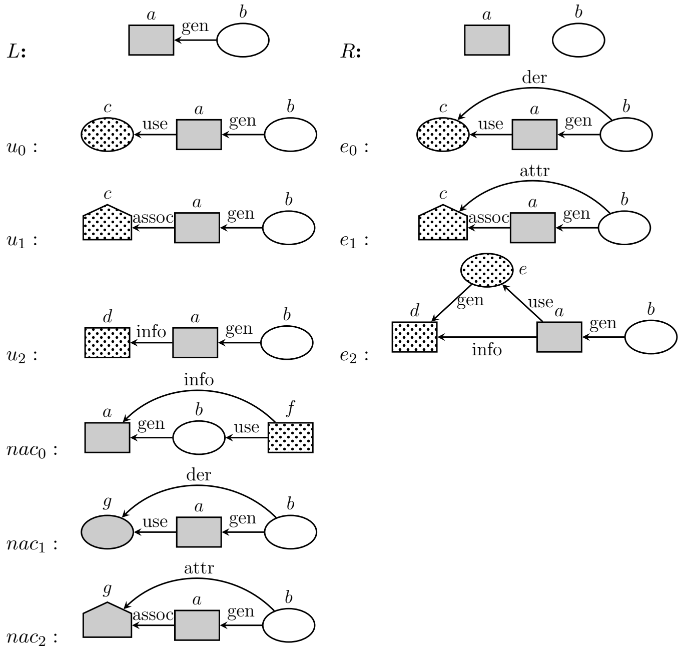
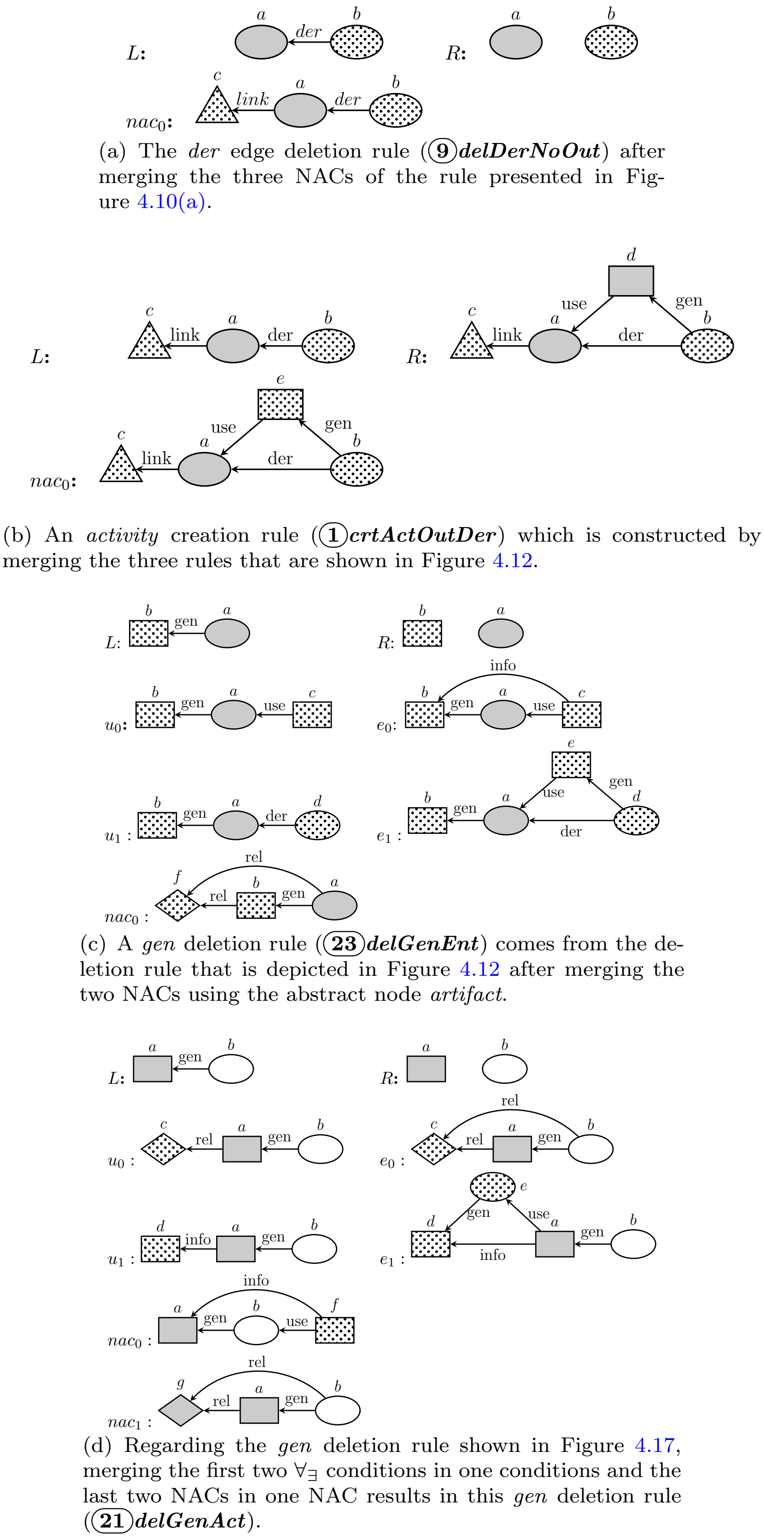


FIGURE 4.17: Updating the rule in Figure 4.16(a) by changing the confidentiality level value for the entity *b* from *any* to *plain* so that the *gen* edge is deleted when the generated entity is plain. In addition, to avoid inconsistency between the *der* and *attr* deletion rules, two NACs (nac_1 and nac_2) are added to the rule to ensure that the *gen* edge is not part of a *der* edge when the derived entity is restricted or an *attr* edge when the agent is restricted.

graph integrity. This rule construction process result in numerous graph transformation rules (twenty-four rules). This large number of rules makes it difficult to follow each rule individually. It also complicates many proofs such as termination, parallelism and confluence. Therefore, in the next chapter, the constructed rules that are presented in this chapter along with new defined rules are grouped together by defining the notion of rule templates.



(a) The *der* edge deletion rule (⑨***delDerNoOut***) after merging the three NACs of the rule presented in Figure 4.10(a).

(b) An *activity* creation rule (①***crtActOutDer***) which is constructed by merging the three rules that are shown in Figure 4.12.

(c) A *gen* deletion rule (㉓***delGenEnt***) comes from the deletion rule that is depicted in Figure 4.12 after merging the two NACs using the abstract node *artifact*.

(d) Regarding the *gen* deletion rule shown in Figure 4.17, merging the first two $\forall_{\exists}$ conditions in one conditions and the last two NACs in one NAC results in this *gen* deletion rule (㉑***delGenAct***).

FIGURE 4.18: Merging rules and conditions using the abstract nodes: *node* and *artifact*.

Chapter 5

Template-based Graph Transformation System

5.1 Introduction

In the previous chapter, the core PROV data model is formally described. Simple graph transformation rules, which only consist of LHS and RHS, are expanded to conditional rules with negative and universal-existential conditions. A methodology is proposed and used to construct the graph transformation rules of PROV-GTS gradually. This rule construction results in twenty-four graph transformation rules. This large number of graph transformation rules presents challenges in understanding the overall system and makes analysing and proving various graph transformation properties, such as termination, parallelism and confluence, complicated. To tackle these issues and overcome the challenges, in this chapter, a new notion of rule templates is used to group the parallel independent rules that are composed of similar graph patterns. To conduct this grouping process, some rules that are constructed in Chapter 4 as well as new ones demonstrated in this chapter are used. This grouping process results in only eight templates. The rule templates enhance the readability of the rules and allow a more concise definition of the transformation system and a tractable way of proving its properties. Formally, the description of the system, which is constructed from the PROV data model, is established based on a variety of concepts from category theory such as functors, slice categories, and natural transformations. The basic concepts of category, such as functors, diagrams and pushout, are provided in Appendix A. Section 5.2 presents the approach used in constructing the rule templates. In Section 5.3, the notion of rule templates of PROV-GTS is formally defined. Finally, this chapter is summarized in Section 5.4.

5.2 Grouping the Conditional Rules Using Templates

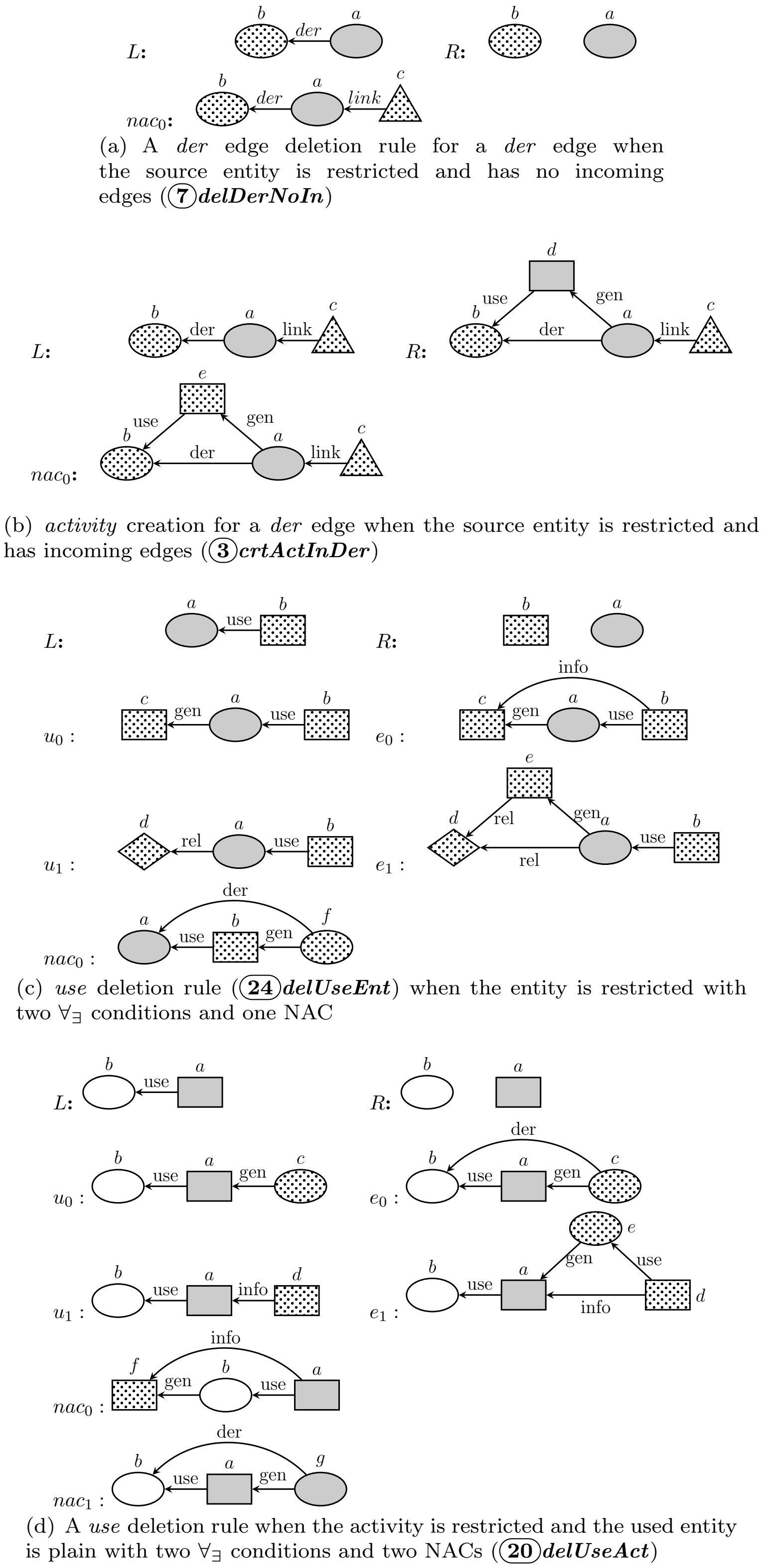
Using the same methodology described in Chapter 4, we have constructed the following transformation rules as shown in Figure 5.1:

- A deletion rule is shown in Figure 5.1(a) capable of removing a *der* edge when the derived entity is restricted and has no incoming edges.
- A rule for creating an activity for a *der* edge when its source node is a restricted entity is defined, as shown in Figure 5.1(b).
- Another rule for deleting the *use* edge connected to a restricted entity is depicted in Figure 5.1(c). This rule consists of three conditions, (u_0, e_0) for checking that for each outgoing *gen* edge there is an *info* edge, (u_1, e_1) to ensure that for each outgoing *der* or *attr* there is an activity, and nac_0 for ensuring that the restricted entity has no incoming *der* edge from an entity generated by the activity b.
- In Figure 5.1(d) a *use* deletion rule is shown when the activity is restricted and the used entity is plain.

The above rules consist of graphs of the same graph topology as the rules defined in Figure 4.18 but after reversing the edges of all the graphs that form those rules. That is, the graphs in Figures 5.1(a), 5.1(b), 5.1(c) and 5.1(d) represent graph transformation rules which consist of LHS, RHS and sets of conditions with similar graph topologies with the rules presented in Figures 4.18(a), 4.18(b), 4.18(c) and 4.18(d) respectively, after the edges in all the graphs were reversed. We can use the notion of directed graphs with reverse indicator defined in Section 4.3 of Chapter 4 to group rules with direct and reverse edges together in the same template.

We define the notion of rule templates to group together the rules that have similar graph topologies in left- and right-hand sides, as well as the sets of conditions. For example, if we consider removing an *assoc* edge when the activity is *restricted*, there will be another deletion rule ($\textcircled{\textbf{22}}$***delAssocAct***), as depicted in Figure 5.2(a), with the following conditions:

- Two universal-existential conditions to ensure that for each incoming *gen* edge to the restricted activity there exists an *attr* edge (u_0, e_0) and for each incoming *info* edge there is an entity (u_1, e_1).
- A negative application condition to check that there is not a restricted entity with an incoming *gen* edge to the restricted activity and an *attr* edge to the agent nac_0.



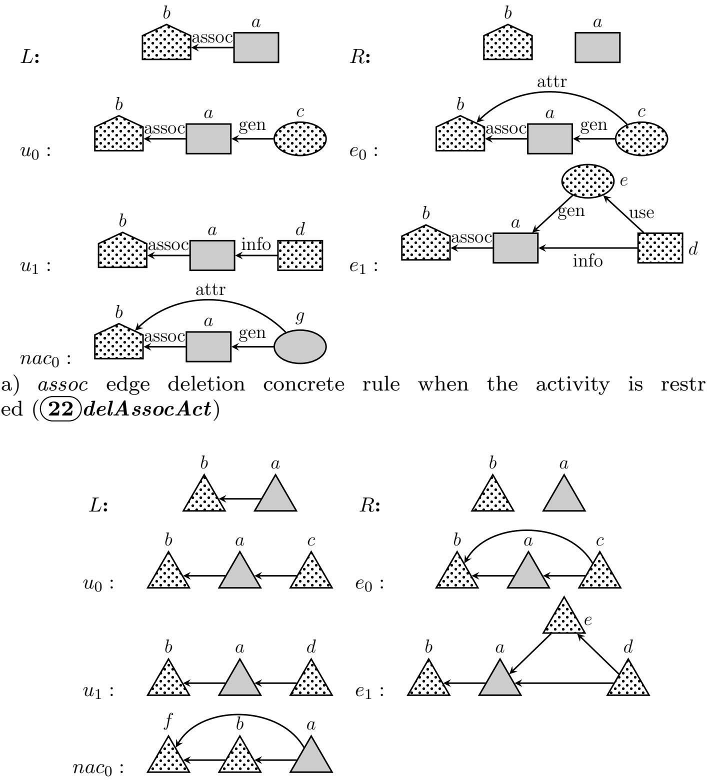
(a) A *der* edge deletion rule for a *der* edge when the source entity is restricted and has no incoming edges (⑦***delDerNoIn***)

(b) *activity* creation for a *der* edge when the source entity is restricted and has incoming edges (③***crtActInDer***)

(c) *use* deletion rule (㉔***delUseEnt***) when the entity is restricted with two $\forall_\exists$ conditions and one NAC

(d) A *use* deletion rule when the activity is restricted and the used entity is plain with two $\forall_\exists$ conditions and two NACs (⑳***delUseAct***)

FIGURE 5.1: Additional rules with universal-existential and negative application conditions



(a) *assoc* edge deletion concrete rule when the activity is restricted (㉒***delAssocAct***)

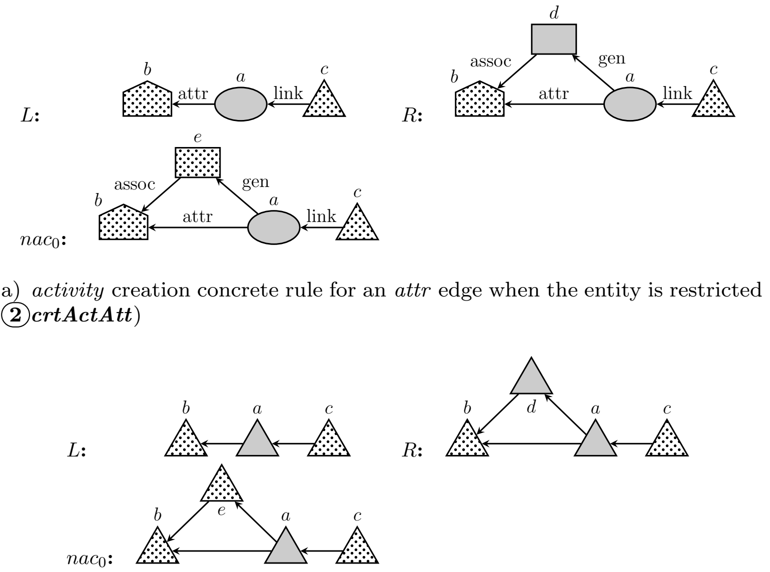
(b) *der-assoc* deletion abstract rule (**template8**) constructed form the concrete rules ㉓***delGenEnt*** (Figure 4.18(c)) and ㉒***delAssocAct*** (Figure 5.2(a)).

FIGURE 5.2: Constructing a new concrete provenance graph transformation rules for *assoc* edge and the corresponding abstract rule

The rule ㉒***delAssocAct*** consists of graphs of the same topology as the *gen* deletion rule (㉓***delGenEnt***) shown in Figure 4.18(c) but with different nodes and edges. We define a template with the same structure as the two rules by using the abstract node *node*, as shown in Figure 5.2(b). The *use* deletion rule when the entity is restricted, as illustrated in 5.1(c), has the same structure as the two rules above but with reverse edges. By using the definitions of graphs with reverse indicator and their corresponding morphisms, we can merge this *use* deletion rule with the other two rules as well.

Similarly, regarding the two activity creation rules for *der* edges, ①***crtActOutDer*** and ③***crtActInDer***, presented in Figures 4.18(b) and 5.1(b) respectively, we can define another rule for creating activity nodes for the *attr* edge (②***crtActAtt***) when the entity is restricted, as illustrated in Figure 5.3(a). Then we can group those three creation rules in one rule template as shown in Figure 5.3(b).

Using the same approach of grouping rules that have similar structures and consists of graphs with similar topologies, a set of rule templates, as shown in Figure 5.4, are constructed. Each template has an abstract rule representing the structure of the instance rules with direct and reverse edges. Each template is accompanied by two tables: the first table represents the set of graph nodes with confidentiality level graph patterns



(a) *activity* creation concrete rule for an *attr* edge when the entity is restricted (②***crtActAtt***)

(b) An *activity* creation abstract rule (template1) based on the three rules ①***crtActOutDer*** (Figure 4.18(b)), ③***crtActInDer*** (Figure 5.1(b)) and ②***crtActAtt*** Figure 5.3(a).

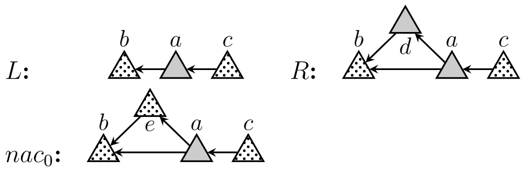
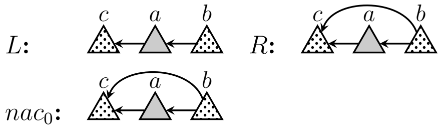
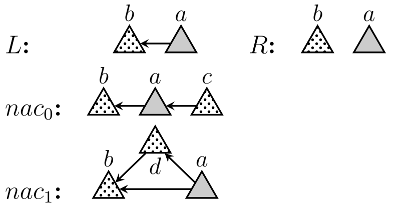
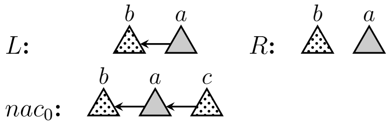
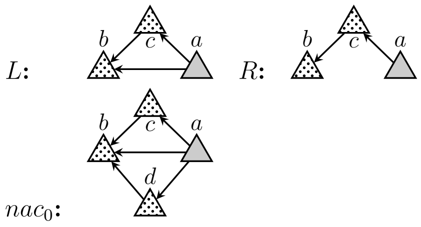
FIGURE 5.3: Constructing a new concrete provenance graph transformation rules for creating *activity* node and the corresponding abstract rule

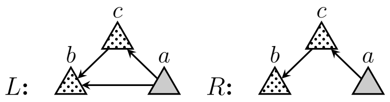
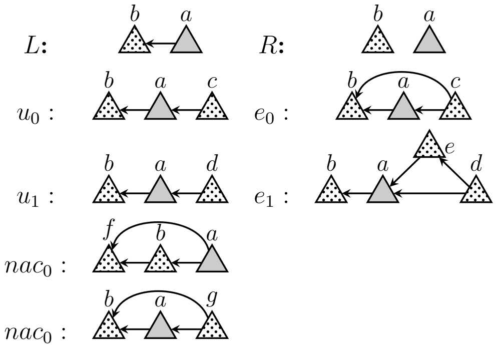
for each rule instance and the second table represents their edges. Some rule instances have direct edges while others have graphs with reverse edges written as $edge^{-1}$ where *edge* is one of the edges in the closure of PROV type graph with inheritance $\overline{AG}$ (Definition 4.14). The label *link* is used for any edge that has the abstract node *node* as its source or target, while the label *rel* is used for the edges connected to the abstract node *artifact*. Each column in the first table with its corresponding column in the second table represent one of the rule instances of that template. Each rule instance is represented by two numbers $\boxed{\text{m-n}}$, where 'm' is the template number and 'n' is the rule instance number.

5.3 Formal Description of Rule Templates

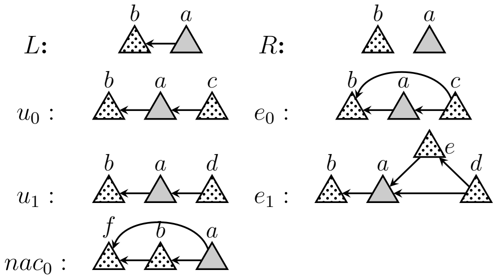
Each rule template has an abstract conditional rule, which consist only of abstract nodes, with the same structure as the concrete rules that represents the set of instance rules of that template.

Definition 5.1 (Abstract conditional rule)**.** an abstract conditional rule is $\nabla_A = (r_A : L_A \rightarrow R_A , C_A, T_A, M_A, N_A)$ which is constructed from graphs that consist only of abstract graph nodes of type *node* $\in AN$ (△). □

<table>
<tr><th>Template</th><th>Abstract rule</th><th>Nodes and edges tables</th></tr>
<tr><td>template1</td><td> </td><td>
<table>
<tr><th></th><th>1-1</th><th>1-2</th><th>1-3</th></tr>
<tr><td>a</td><td>⬬</td><td>⬬</td><td>⬬</td></tr>
<tr><td>b</td><td>⬭</td><td>⬠</td><td>⬭</td></tr>
<tr><td>c</td><td>△</td><td>△</td><td>△</td></tr>
<tr><td>d</td><td>■</td><td>■</td><td>■</td></tr>
<tr><td>e</td><td>▨</td><td>▨</td><td>▨</td></tr>
</table>
<table>
<tr><th></th><th>1-1</th><th>1-2</th><th>1-3</th></tr>
<tr><td>ab</td><td><i>der</i></td><td><i>attr</i></td><td>der^{-1}</td></tr>
<tr><td>ca</td><td><i>link</i></td><td><i>link</i></td><td>$link^{-1}$</td></tr>
<tr><td>ad</td><td><i>gen</i></td><td><i>gen</i></td><td>use^{-1}</td></tr>
<tr><td>db</td><td><i>use</i></td><td><i>assoc</i></td><td>gen^{-1}</td></tr>
<tr><td>ae</td><td><i>gen</i></td><td><i>gen</i></td><td>use^{-1}</td></tr>
<tr><td>eb</td><td><i>use</i></td><td><i>assoc</i></td><td>gen^{-1}</td></tr>
</table></td></tr>
<tr><td>template2</td><td> </td><td>
<table>
<tr><th></th><th>2-4</th></tr>
<tr><td>a</td><td>⬬</td></tr>
<tr><td>b</td><td>▨</td></tr>
<tr><td>c</td><td>▨</td></tr>
</table>
<table>
<tr><th></th><th>2-4</th></tr>
<tr><td>ac</td><td><i>gen</i></td></tr>
<tr><td>ba</td><td><i>use</i></td></tr>
<tr><td>bc</td><td><i>info</i></td></tr>
</table></td></tr>
<tr><td>template3</td><td> </td><td>
<table>
<tr><th></th><th>3-5</th><th>3-6</th></tr>
<tr><td>a</td><td>■</td><td>■</td></tr>
<tr><td>b</td><td>▨</td><td>▨</td></tr>
<tr><td>c</td><td>△</td><td>△</td></tr>
<tr><td>d</td><td>⬬</td><td>⬬</td></tr>
</table>
<table>
<tr><th></th><th>3-5</th><th>3-6</th></tr>
<tr><td>ab</td><td><i>info</i></td><td>$info^{-1}$</td></tr>
<tr><td>ca</td><td><i>link</i></td><td>$link^{-1}$</td></tr>
<tr><td>ad</td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>db</td><td><i>gen</i></td><td>use^{-1}</td></tr>
</table></td></tr>
<tr><td>template4</td><td> </td><td>
<table>
<tr><th></th><th>4-7</th><th>4-8</th><th>4-9</th><th>4-10</th></tr>
<tr><td>a</td><td>⬬</td><td>⬬</td><td>⬬</td><td>⬟</td></tr>
<tr><td>b</td><td>⬭</td><td>⬠</td><td>⬭</td><td>⬭</td></tr>
<tr><td>c</td><td>△</td><td>△</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th></th><th>4-11</th><th>4-12</th><th>4-13</th></tr>
<tr><td>a</td><td>⬟</td><td>⬟</td><td>⬟</td></tr>
<tr><td>b</td><td>⬠</td><td>⬠</td><td>▨</td></tr>
<tr><td>c</td><td>△</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th></th><th>4-7</th><th>4-8</th><th>4-9</th><th>4-10</th></tr>
<tr><td>ab</td><td><i>der</i></td><td><i>attr</i></td><td>der^{-1}</td><td>$attr^{-1}$</td></tr>
<tr><td>ca</td><td><i>link</i></td><td><i>link</i></td><td>$link^{-1}$</td><td>$link^{-1}$</td></tr>
</table>
<table>
<tr><th></th><th>4-11</th><th>4-12</th><th>4-13</th></tr>
<tr><td>ab</td><td>del^{-1}</td><td><i>del</i></td><td>$assoc^{-1}$</td></tr>
<tr><td>ca</td><td>$link^{-1}$</td><td><i>link</i></td><td>$link^{-1}$</td></tr>
</table></td></tr>
<tr><td>template5</td><td> </td><td>
<table>
<tr><th></th><th>5-14</th><th>5-15</th></tr>
<tr><td>a</td><td>■</td><td>■</td></tr>
<tr><td>b</td><td>▨</td><td>▨</td></tr>
<tr><td>c</td><td>⬭</td><td>⬭</td></tr>
<tr><td>d</td><td>⬬</td><td>⬬</td></tr>
</table>
<table>
<tr><th></th><th>5-14</th><th>5-15</th></tr>
<tr><td>ab</td><td><i>info</i></td><td>$info^{-1}$</td></tr>
<tr><td>ac</td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>cb</td><td><i>gen</i></td><td>use^{-1}</td></tr>
<tr><td>ad</td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>db</td><td><i>gen</i></td><td>use^{-1}</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Template</th><th>Abstract rule</th><th>Nodes and edges tables</th></tr>
<tr><td>template6</td><td> </td><td>
<table>
<tr><th></th><th>6-16</th><th>6-17</th><th>6-18</th><th>6-19</th></tr>
<tr><td>a</td><td>⬬</td><td>⬬</td><td>⬬</td><td>⬟</td></tr>
<tr><td>b</td><td>◍</td><td>⬠</td><td>◍</td><td>◍</td></tr>
<tr><td>c</td><td>▦</td><td>▦</td><td>▦</td><td>▦</td></tr>
</table>
<table>
<tr><th></th><th>6-16</th><th>6-17</th><th>6-18</th><th>6-19</th></tr>
<tr><td>ab</td><td><i>der</i></td><td><i>attr</i></td><td>der^{-1}</td><td>$attr^{-1}$</td></tr>
<tr><td>ac</td><td><i>gen</i></td><td><i>gen</i></td><td>use^{-1}</td><td>$assoc^{-1}$</td></tr>
<tr><td>cb</td><td><i>use</i></td><td><i>assoc</i></td><td>gen^{-1}</td><td>gen^{-1}</td></tr>
</table>
</td></tr>
<tr><td>template7</td><td>

L:
R:
u_0 :
e_0 :
u_1 :
e_1 :
nac_0 :
nac_0 :
a
b
c
d
e
f
g

</td><td>
<table>
<tr><th></th><th>7-20</th><th>7-21</th></tr>
<tr><td>a</td><td>▬</td><td>▬</td></tr>
<tr><td>b</td><td>⬭</td><td>⬭</td></tr>
<tr><td>c</td><td>◍</td><td>◈</td></tr>
<tr><td>d</td><td>▦</td><td>▦</td></tr>
<tr><td>e</td><td>◍</td><td>◍</td></tr>
<tr><td>f</td><td>▦</td><td>▦</td></tr>
<tr><td>g</td><td>⬬</td><td>◆</td></tr>
</table>
<table>
<tr><th></th><th>7-20</th><th>7-21</th></tr>
<tr><td>ab</td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>ca</td><td><i>gen</i></td><td>rel^{-1}</td></tr>
<tr><td>cb</td><td><i>der</i></td><td>rel^{-1}</td></tr>
<tr><td>da</td><td><i>info</i></td><td>$info^{-1}$</td></tr>
<tr><td>de</td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>ea</td><td><i>gen</i></td><td>use^{-1}</td></tr>
<tr><td>af</td><td><i>info</i></td><td>$info^{-1}$</td></tr>
<tr><td>bf</td><td><i>gen</i></td><td>use^{-1}</td></tr>
<tr><td>ga</td><td><i>gen</i></td><td>rel^{-1}</td></tr>
<tr><td>gb</td><td><i>der</i></td><td>rel^{-1}</td></tr>
</table>
</td></tr>
<tr><td>template8</td><td>

L:
R:
u_0 :
e_0 :
u_1 :
e_1 :
nac_0 :
a
b
c
d
e
f

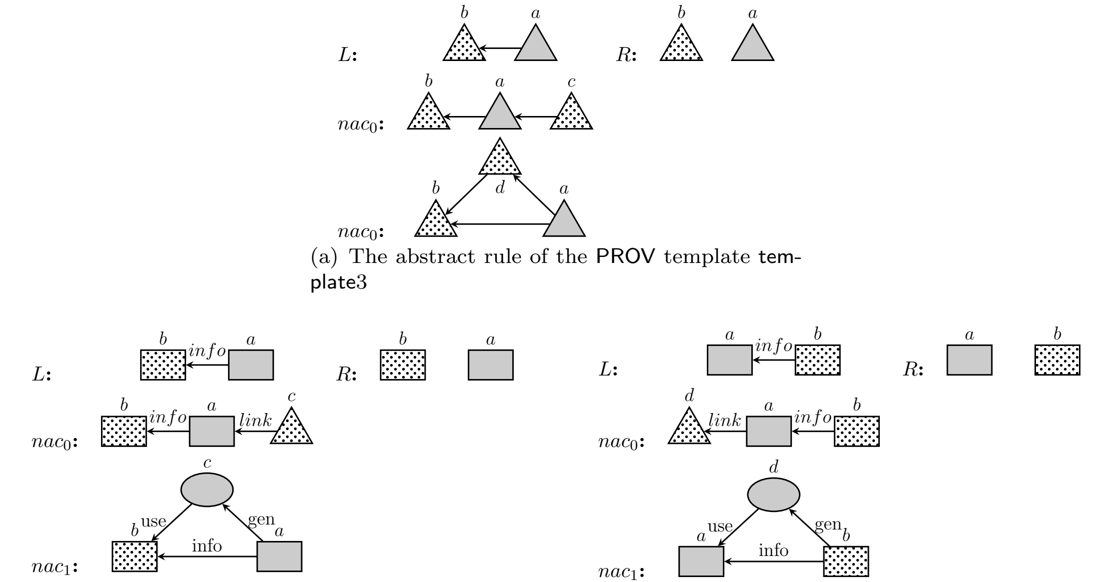
</td><td>
<table>
<tr><th></th><th>8-22</th><th>8-23</th><th>8-24</th></tr>
<tr><td>a</td><td>▬</td><td>⬬</td><td>⬬</td></tr>
<tr><td>b</td><td>⬠</td><td>▦</td><td>▦</td></tr>
<tr><td>c</td><td>◍</td><td>▦</td><td>▦</td></tr>
<tr><td>d</td><td>▦</td><td>◍</td><td>◈</td></tr>
<tr><td>e</td><td>◍</td><td>▦</td><td>▦</td></tr>
<tr><td>f</td><td>⬠</td><td>◈</td><td>◍</td></tr>
</table>
<table>
<tr><th></th><th>8-22</th><th>8-23</th><th>8-24</th></tr>
<tr><td>ab</td><td><i>assoc</i></td><td><i>gen</i></td><td>use^{-1}</td></tr>
<tr><td>ca</td><td><i>gen</i></td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>cb</td><td><i>attr</i></td><td><i>info</i></td><td>$info^{-1}$</td></tr>
<tr><td>da</td><td><i>info</i></td><td><i>der</i></td><td>rel^{-1}</td></tr>
<tr><td>de</td><td><i>use</i></td><td><i>gen</i></td><td>rel^{-1}</td></tr>
<tr><td>ea</td><td><i>gen</i></td><td><i>use</i></td><td>gen^{-1}</td></tr>
<tr><td>af</td><td><i>assoc</i></td><td><i>rel</i></td><td>der^{-1}</td></tr>
<tr><td>bf</td><td><i>del</i></td><td><i>rel</i></td><td>gen^{-1}</td></tr>
</table>
</td></tr>
</table>

Figure 5.4: Set of templates

The template is instantiated using the nodes and edges from one of its rule instances to construct that rule using the following three steps:

1. node replacement: replace each node of the abstract rule with its corresponding node of the instance rule from the table of nodes.
2. edge labelling: label each edge with its corresponding label from the table of edges.
3. edge redirection: if the edge labels have (-1) as superscript then reverse the direction of all of the edges.

For instance, as shown in Figure 5.5, the two instances of template3 can be instantiated using the abstract rule of the template and the nodes and edges tables. To construct the abstract rule **3-5** ***delInfNoIn*** we replace each node of the abstract rule of the template with the nodes in the first column which represents the rule instance. Then we label each edge with the corresponding label from the table of edges. In case of the second rule instance **3-6** ***delInfNoOut***, the same node replacement and edge relabelling as above is applied, but since the direction of edges is reverse, we apply the edge redirection as well.



(a) The abstract rule of the PROV template template3

(b) **3-5** ***delInfNoIn***: The first instance rule of template3 with direct edges

(c) **3-6** ***delInfNoOut***: The second instance rule of template3 with reverse edges

FIGURE 5.5: Instantiation of the two rule instances **3-5** ***delInfNoIn*** and **3-6** ***delInfNoOut*** of the template template3

The rule templates are defined based on categorical constructions such as functors, natural transformation, and pushouts. Diagrams are homomorphisms from shape graphs to categories (Barr and Wells 1995; Awodey 2010). Since each rule template has an abstract rule with the same graph topologies as the template's rule instances, a shape graph of the same structure of the abstract rule can be defined.

Definition 5.2 (Shape graph of ∇_A)**.** The shape graph of $\nabla_A = (\ r_A : L_A \longrightarrow R_A, C_A, T_A, M_A, N_A)$ is the directed graph γ consists of $N_\gamma = \{L_A, R_A\} \cup \{u|\ (u \longrightarrow e\ )\in C_A\} \cup \{e|\ (u \longrightarrow e)\ \in C_A\} \cup T_A$ as a set of nodes and $E_\gamma = \{r_A\} \cup C_A \cup M_A \cup N_A$ as a set of edges. □

Two diagram functors can be defined from the free category of the shape graph to $Graph_{pITG}$, one over the abstract rule and the other over one of the rule instances of the template with an obvious natural transformation between the two functors. Then the rule template can be represented as a set of such natural transformations, one for each rule instance.

Definition 5.3 (Rule template)**.** A rule template is $\tau = (\nabla_A, \mathcal{R}, \mathcal{E})$ consists of an abstract conditional rule ∇_A, a set of concrete instance rules $\mathcal{R}$ and the set $\mathcal{E}$ of natural transformations. Let γ be the shape graph of ∇_A, and $F(\gamma)$ be the free category of γ. Given the diagram functors $D : F(\gamma) \longrightarrow Graph_{pITG}$ on ∇_A and $t : F(\gamma) \longrightarrow Graph_{pITG}$ on $\nabla \in \mathcal{R}$ then for each ∇ there is a natural transformations $(\psi : D \rightsquigarrow t)\in \mathcal{E}$ such that for each $(f : X \longrightarrow Y) \in F(\gamma)$ the diagram below commutes.

$$\begin{tikzcd}
F(\gamma) \arrow[r, bend left=50, "D"{name=U}] \arrow[r, bend right=50, "t"'{name=L}] \arrow[from=U, to=L, squiggly, "\psi"'] & Graph_p^r
\end{tikzcd}
\qquad
\begin{tikzcd}
D(X) \arrow[r, "D(f)"] \arrow[d, "\psi_X"'] \arrow[dr, phantom, "="] & D(Y) \arrow[d, "\psi_Y"] \\
t(X) \arrow[r, "t(f)"'] & t(Y)
\end{tikzcd}$$

$$\psi_Y \circ D(f) = t(f) \circ \psi_X$$

Ignoring the typing, the morphism $\psi : D \rightsquigarrow t$ is a natural isomorphism over the set of nodes and edges.

□

To apply a concrete rule via its template, we need to find a matching of the abstract rule's LHS of the template and check if it matches the LHS of any of the concrete instance rules. This matching must satisfy all the application conditions of the concrete rule via the template, as described in Definition 5.4 below.

Definition 5.4 (Template satisfaction)**.** Given an abstract rule $\nabla_A = (\ r_A : L_A \longrightarrow R_A, C_A, T_A, M_A, N_A)$ and a concrete rule $\nabla = (\ r : L \longrightarrow R\ , C, T, M, N)$ of the rule template $\tau = (\nabla_A, \mathcal{R}, \mathcal{E})$, let $D : F(\gamma) \longrightarrow Graph_{pITG}$ on ∇_A and $t : F(\gamma) \longrightarrow Graph_{pITG}$ on $\nabla \in \mathcal{R}$ be two diagram functors, $\psi : D \rightsquigarrow t$ a natural transformation, G a graph, and

$f : L \longrightarrow G$ a match which satisfies C and N. Given that $f^{'} = f \circ \psi_L$, $g^{'} = g \circ \psi_u$, and $h^{'} = h \circ \psi_e$ then the match $f^{'} : L_A \longrightarrow G$ in the diagrams below satisfies C_A if for each $g^{'} : u_A \longrightarrow G$ there exist at least one $h^{'} : e_A \longrightarrow G$ such that $g^{'} = h^{'} \circ c_A$ and satisfies T_A if for each $(n : L_A \longrightarrow nac_A) \in T_A$ there does not exist a total morphism $p^{'} : nac_A \longrightarrow G$ with $p^{'} \circ n_A = f$.

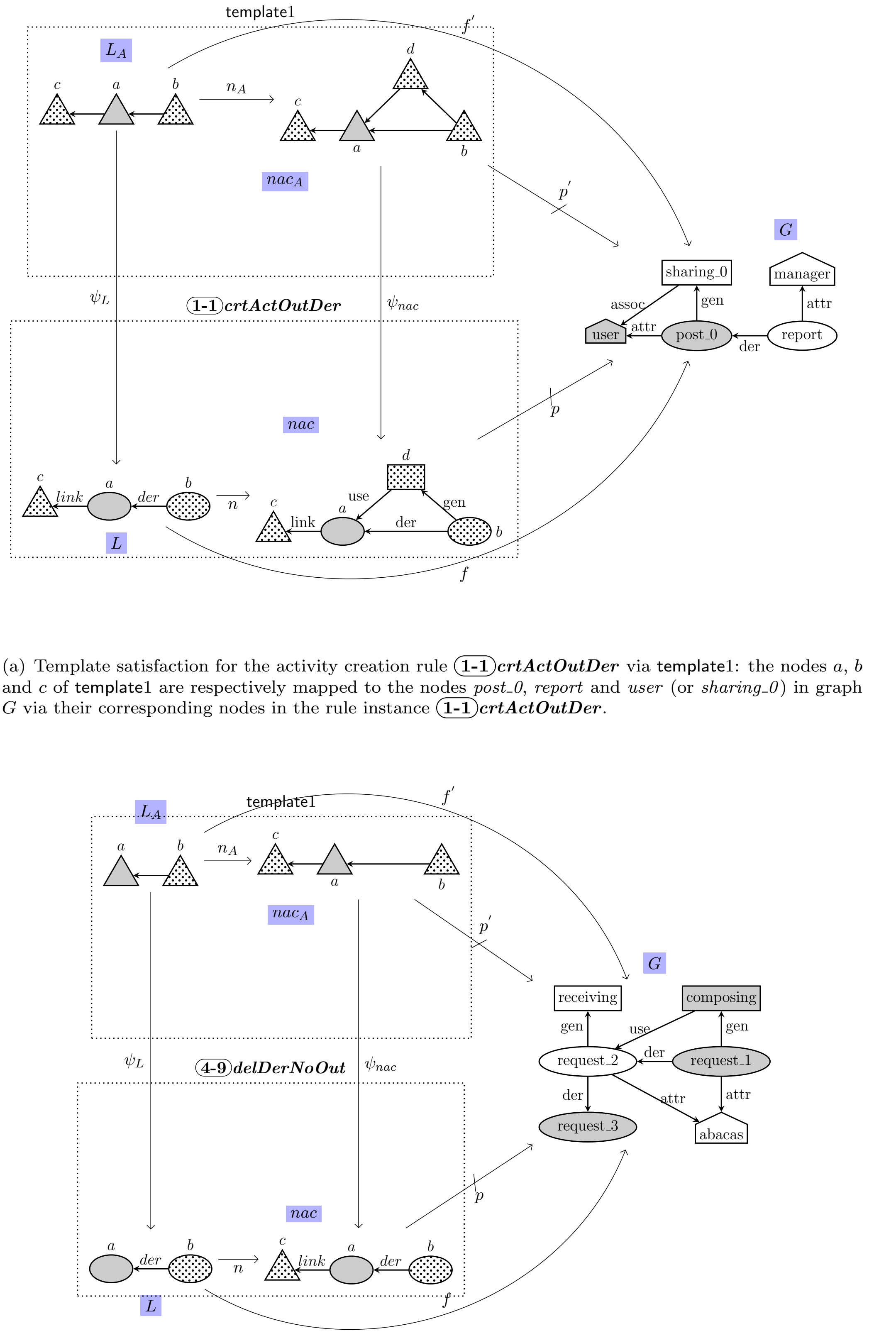
$$\begin{tikzcd}
L_A \arrow[r, "m_A"] \arrow[dr, "n_A"'] \arrow[dddd, "\psi_L"'] \arrow[ddrr, bend left, "f^{'}"] & u_A \arrow[d, "c_A"] \arrow[ddr, "g^{'}"] \arrow[dddd, bend right, "\psi_u"'] & \\
 & e_A \arrow[dr, "h^{'}"] \arrow[dd, "\psi_e"] & \\
 & & G \\
 & e \arrow[ur, "h"] & \\
L \arrow[r, "m"'] \arrow[ur, "n"] \arrow[uurr, bend right, "f"'] & u \arrow[u, "c"'] \arrow[uur, "g"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
L_A \arrow[r, "n_A"] \arrow[dd, "\psi_L"'] \arrow[dr, bend left=60, "f^{'}"] & nac_A \arrow[dr, "p^{'}", "/"{marking}] \arrow[dd, bend right, "\psi_{nac}"'] & \\
 & & G \\
L \arrow[r, "n"'] \arrow[urr, bend right, "f"'] & nac \arrow[ur, "p"', "/"{marking}] &
\end{tikzcd}$$

□

Figure 5.6 illustrates two examples of using rule templates for finding a match of a rule instance in a provenance graph via its template. In Figure 5.6(a), the *activity* creation template (template1) is used to find a match of the concrete rule **(1-1)*crtActOutDer*** in a subgraph of the OnlinePost graph. Figure 5.6(b) illustrates another example of template satisfaction by finding the match of the concrete rule **(4-9)*delDerNoOut*** in a subgraph of the RideShare graph using template4.

To apply the rule, we use the single-pushout approach but instead of direct modification, the rule application is performed via the template that represents the rule instance. We replace the match of LHS found by the template satisfaction with the RHS of the concrete rule.

Definition 5.5 (Template application)**.** Given an abstract rule $\nabla_A = (r_A : L_A \longrightarrow R_A, C_A, T_A, M_A, N_A)$ and a concrete rule $\nabla = (r : L \longrightarrow R, C, T, M, N)$ of the rule template $\tau = (\nabla_A, \mathcal{R}, \mathcal{E})$, a provenance graph G, the match $f : L \longrightarrow G$ which satisfies C and T, and the match $h : L_A \longrightarrow G$ which satisfies C_A and T_A then the rule template is applied by replacing the match $h(L_A)$ by the right-hand side R of the concrete rule resulting in the graph H such that $h = f \circ \psi_L$ and the following diagram is a pushout:



(a) Template satisfaction for the activity creation rule **1-1** ***crtActOutDer*** via template1: the nodes a, b and c of template1 are respectively mapped to the nodes *post_0*, *report* and *user* (or *sharing_0*) in graph G via their corresponding nodes in the rule instance **1-1** ***crtActOutDer***.

(b) Template satisfaction for *der* deletion rule **4-9** ***delDerNoOut*** via template4: the nodes a and b of template4 are respectively mapped to the nodes *request_2* and *request_3* in graph G via their corresponding nodes in the rule instance **4-9** ***delDerNoOut***.

Figure 5.6: Two examples of template satisfactions

$$\begin{tikzcd}
L_A \arrow[r, "r_A"] \arrow[d, "\psi_L"'] \arrow[dd, bend right=60, "h"'] & R_A \arrow[d, "\psi_R"] \arrow[dd, bend left=60, "h'"] \\
L \arrow[r, "r"] \arrow[d, "f"'] & R \arrow[d, "f'"] \\
G \arrow[r, "r'"] & H
\end{tikzcd}$$

$$h = f \circ \psi_L$$

$$h' = f' \circ \psi_R$$

$$r' \circ h = h' \circ r_A$$

□

Regarding the matches $f : L \longrightarrow G$ and $h : L_A \longrightarrow G$ and the composition $h = f \circ \psi_L$, since $\psi : D \longrightarrow t$ is isomorphic then $f = h \circ \psi_L^{-1}$ which means that applying a concrete rule using its template is same as applying the rule directly. This indicates that the same set of transformation steps must be applied to convert a provenance graph whether by using template-based transformation or directly applying the concrete graph transformation rules.

Two examples of using template in transformation rule application are shown in Figure 5.7. In Figure 5.7(a), the *activity* creation template (template1) is used for applying the concrete rule **1-1** ***crtActOutDer*** on a subgraph of the OnlinePost graph. Figure 5.6(b) used a subgraph of the RideShare graph for applying the concrete rule **4-9** ***delDerNoOut*** via its template (template4). Both template applications use the matches found in the template satisfactions depicted in Figure 5.6.

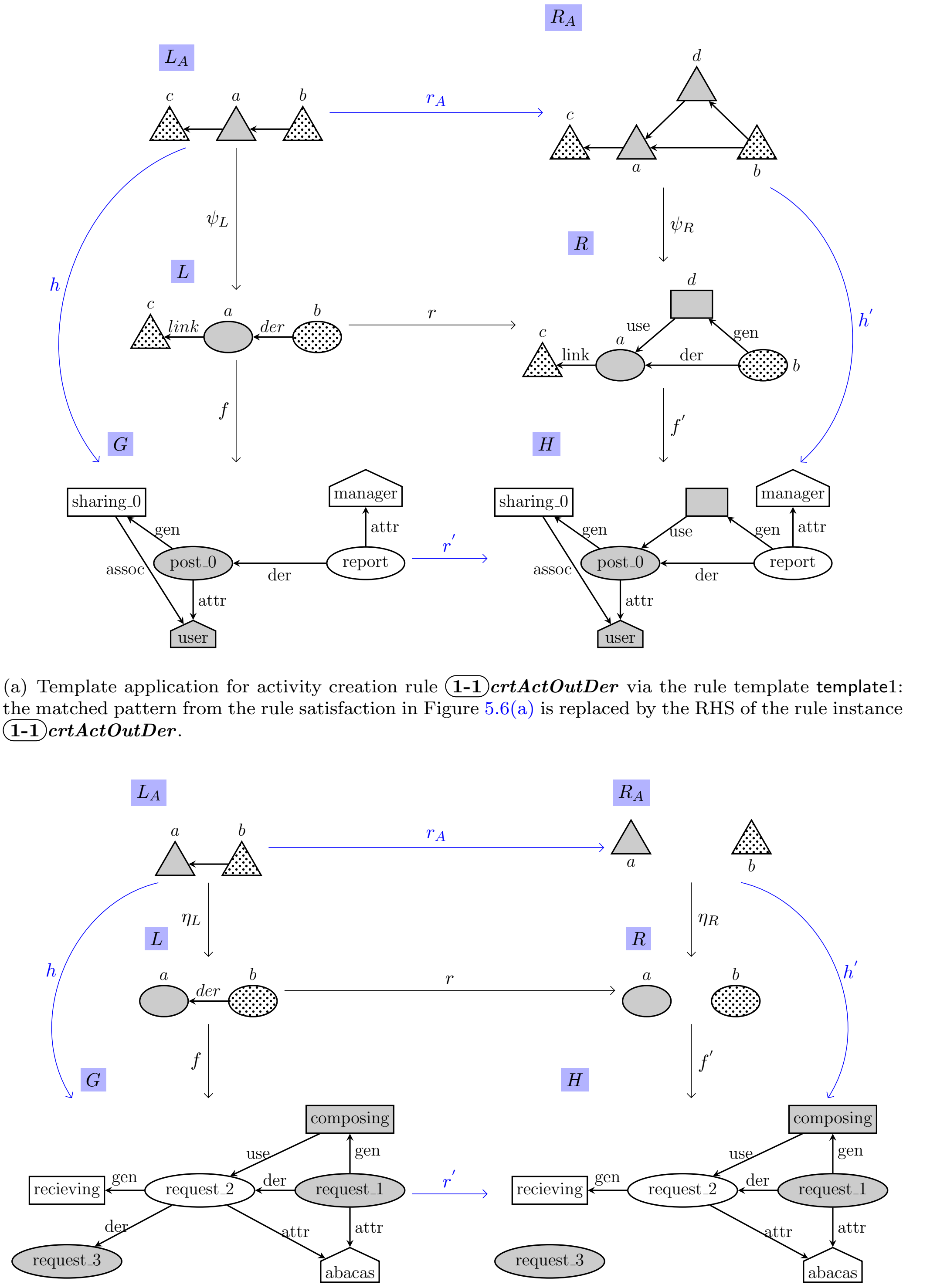
Step by step rule applications regarding the graphs OnlinePost and RideShare are illustrated in Appendix E.

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

Definition 5.6 (Provenance Graph Transformation System)**.** A template-based provenance graph transformation system PROV-GTS = $(ITG, TEMP, INS)$ consists of a type graph with inheritance ITG, a set of rule templates $TEMP = \{\tau_{i=1...8}\}$ and a collection of rule instances INS where $I_i \in INS$ is a set of conditional rules which represents the rule instances of the template τ_i. □

5.4 Summary

In this chapter, the formal definitions required for constructing the proposed template-based graph transformation system (PROV-GTS) are presented. Rule templates are



(a) Template application for activity creation rule **1-1** ***crtActOutDer*** via the rule template `template1`: the matched pattern from the rule satisfaction in Figure 5.6(a) is replaced by the RHS of the rule instance **1-1** ***crtActOutDer***.

(b) Template application for *der* deletion rule **4-9** ***delDerNoOut*** via the rule template `template4`: the matched pattern from the rule satisfaction in Figure 5.6(b) is replaced by the RHS of the rule instance **4-9** ***delDerNoOut***.

FIGURE 5.7: An Example on template application using `template1` and the graph $\mathcal{V}$.

used to group the rules that are composed of similar graph patterns. Finally, the rule satisfaction and application are presented on template basis. This template construction reduces the number of rules from twenty-four down to only eight rule templates allowing a more concise definition of the transformation system and a tractable way of proving its properties. In the next chapter, the rule templates will be used to investigate and analysis various properties of PROV-GTS such as termination, parallelism and confluence.

Chapter 6

Termination, Parallelism, and Confluence of PROV-GTS

6.1 Introduction

The template construction of PROV-GTS, by grouping the set of concrete graph transformation rules, followed by the demonstration of template-based rule satisfaction and application, have been presented in the previous chapter. In this chapter, various proofs and analyses of graph transformation system's properties such as termination, parallelism, and confluence are presented on template basis. These proofs show that the template-based PROV-GTS is terminating and confluent with deterministic and consistent rule applications. The system's termination using a layered-based rule application approach, which distributes the rules over different layers, is proven in Section 6.2. Section 6.3 provides an analysis for the parallel independence and sequential independence of the graph transformation rules in PROV-GTS. Critical pairs, local confluence, and confluence are presented in Section 6.4. The complexity of the system is analysed in Section 6.5. Finally, this chapter is summarized in Section 6.6.

6.2 Termination

To prove the termination of our proposed system, the layered termination criteria defined in (Ehrig, Ehrig, Lara et al. 2005), are used. This approach is based on layered graph grammars by distributing the transformation rules over different layers. In each layer, the rules are applied for as long as possible before moving to the next layer. Each layer is either a deletion or a non-deletion (creation) layer. Each rule in a deletion layer deletes at least one item, while rules in creation layers create items with NACs to avoid an infinite number of rule applications. Each layer (deletion or creation) has a number of conditions which must be satisfied by the rules. We use this approach to define a layered graph grammar for PROV-GTS on template basis, that is distributing the eight rule templates (templates1–8) in PROV-GTS, instead of individual rule instances, over a number of layers.

Definition 6.1 (PROV-GTS graph grammar)**.** A PROV-GTS graph grammar $PGG =$ (PROV-GTS$, G_0)$ consists of a provenance graph transformation system PROV-GTS and a start graph G_0. We assume that each type found in the PROV type graph TG has an instance in the start graph G_0. □

6.2.1 Termination of PROV-GTS Graph Grammar

The graph grammar PGG with injective matches (monomorphisms) terminates if there is a finite transformation sequence starting from G_0 via the set of rule templates $TEMP = \{\tau_{i=1...8}\}$, and it is layered in the following sense:

1. For each template $\tau \in TEMP$, there is a template layer $tl(\tau)$ with $0 \leq tl(\tau) \leq k_0 (k_0 \in \mathbb{N})$ where $k_0 + 1$ is the number of layers.
2. Each template $\tau \in TEMP$ has a set of negative application conditions T_A with a $n : L_A \longrightarrow nac$ for each $nac \in T_A$.
3. A type set $TYPE = N_{TG} \cup E_{TG}$ where TG is the PROV type graph (see Definition 4.8).

   Note: The edge types in TG are determined by their source and target nodes, for example, the pair of nodes *(entity, activity)* is used for *wasGeneratedBy (gen)* edge. For convenience, we will use the edge names instead of the node pairs.
4. For each $t \in TYPE$, there is a creation layer $cl(t) \in \mathbb{N}$ and a deletion layer $dl(t) \in \mathbb{N}$.
5. Each layer is either a deletion or a creation layer, satisfying the following conditions for all $\tau \in TEMP_k$:

Deletion layer conditions	Creation layer conditions
1. τ deletes at least one item x with $type(x) \in TYPE$. 2. $0 \leq cl(t) \leq dl(t) \leq k_0$ for all $t \in TYPE$. 3. if τ deletes an item of type t then $dl(t) \leq tl(\tau)$. 4. if τ creates an item of type t then $dl(t) > tl(\tau)$.	1. $r : L \longrightarrow R$ of τ is injective (monomorphism). 2. τ has a nac with a monomorphism $n : L \longrightarrow nac$ and there is an isomorphic $\acute{n} : nac \longrightarrow R$ with $\acute{n} \circ n = r$. 3. $x \in L, type(x) \in TYPE$ then $cl(type(x)) \leq tl(\tau)$. 4. if τ creates t then $cl(t) > tl(\tau)$.

The detailed proof of the termination of the layered graph grammar PGG from (Ehrig, Ehrig, Lara et al. 2005) is provided in Appendix C.

Definition 6.2 (Layer Assignment)**.** Given a provenance graph grammar PGG with a start graph G_0 and the types $TYPE$ then for each $t \in TYPE$ the creation and deletion layers are defined as follows:

$$cl(t) = \textit{if } t \textit{ is created by some } \tau \textit{ then } max\{tl(\tau)|\tau \textit{ creates } t\} + 1 \textit{ else } 0,$$

$$dl(t) = \textit{if } t \textit{ is deleted by some } \tau \textit{ then } min\{tl(\tau)|\tau \textit{ deletes } t\} \textit{ else } k_0 + 1.$$

□

6.2.2 Termination Analysis of Provenance Graph Transformation System PROV-GTS

To apply the termination criteria defined above on PROV-GTS, we need first to decide how many layers are required. Since each template represents a set of deletion or a set of creation rules but not both, therefore we can define layers for templates rather than rules.

Templates can be classified to four groups according to the tasks they perform, activity creation template (template1), *info* edge creation template (template2), the templates that delete (*der, attr, info*) edges (templates3-6), and the templates that delete (*use, gen, assoc*) edges (templates7,8). Thus, only four layers are sufficient to accommodate all those templates, as shown in Table 6.1(a). The deletion and creation layers for each element in TG are determined in Table 6.1(b).

Table 6.1: Template layers and layer assignment

(a) Template layers

Layer	**Templates**	**Type**	**Elements affected**
0	template1	creation	activity, use, gen, asso
1	template2	creation	info
2	templates3-6	deletion	info, der, attr
3	template7,8	deletion	use, gen, asso

(b) Layer assignment

Elements (t)	$cl(t)$	$dl(t)$
activity	1	4
info	2	2
der, attr	0	2
use, gen, asso	1	3
entity, agent, del	0	4

6.2.2.1 Checking Conditions of Termination

After successfully distributing the templates over different layers and assigning layers to each element in *TYPE*, in this subsection, we provide the proof of termination of PROV-GTS based on the conditions presented previously in Subsection 6.2.1. We show how the sufficient conditions of deletion and creation layers are fulfilled by the previous layer assignments shown in Table 6.1 as follows:

- *Creation Layer Conditions.*
  The satisfaction of Condition 1 and Condition 2 is straightforward from the templates' construction as there exist some provi=(C_i, E_i) with monomorphisms from C_i and E_i to respectively LHS and RHS of the template and RHS is isomorphic with the NAC of the template. To validate Condition 3, we can see that for each item $x \in LHS$ of template1 $cl(type(x)) = 0$ and $tl(\tau) = 0$, while for template2, $tl(\tau) = 2$ and $cl(type(x)) = 0$ for *entity* and $cl(type(x)) = 1$ for each of *activity*, *gen* and *use*. The validity of Condition 4 can be justified by $cl(t) = 1$ and $tl(\tau) = 0$ for items of type t created by template1 and $cl(info) = 2$ and $tl(\tau) = 1$ for *info* edge created by template2.

- *Deletion Layer Conditions.*
  First, Condition 1 is obvious for the deletion templates of Layer 2 and Layer 3. The validity of Condition 2 can be justified by Table 6.1(b). Condition 3 holds directly, since all deletion templates are included in the last two layers and each element in TG is deleted only in one layer. Finally, the fact that these deletion templates do not create new elements implies Condition 4.

6.3 Parallelism in PROV-GTS

Two rule applications are parallel independent if they produce the same result regardless of the order of their occurrence, that is the two rule applications are independent. Regarding two rule applications, the second is weakly parallel independent to the first application if the first of the applications does not delete from the graph what is required by the second rule to be applicable. This means that the part deleted by the first rule application is not in the match of the LHS of the second rule. The two rule applications are parallel independent if the first rule application is weakly parallel independent to the second as well (Ehrig, Heckel, Korff et al. 1997). We denote a rule application as $d = G \overset{\nabla,m}{\Longrightarrow} H$ where G and H are the original and transformed graphs respectively, $\nabla = (r : L \rightarrow R, C, T, M, N)$ is a conditional rule, and $m : L \rightarrow G$ is a match of L in the graph G which satisfies C and T.

Definition 6.3 (Parallel Independence)**.** Given two alternative rule applications $d_1 = G \overset{\nabla_1,m_1}{\Longrightarrow} H_1$ and $d_2 = G \overset{\nabla_2,m_2}{\Longrightarrow} H_2$ using conditional rules $\nabla_1 = (r_1, C_1, T_1, M_1, N_1)$

and $\nabla_2 = (r_2, C_2, T_2, M_2, N_2)$ and the matches m_1 and m_2 for the simple rules r_1 and r_2 respectively, regarding the diagrams below, d_2 is weakly parallel independent of d_1 if $\acute{m}_2 = r_1^* \circ m_2 : L_2 \longrightarrow H_1$ is a match for r_2 satisfies C_2 and T_2 of ∇_2. The rule applications d_1 and d_2 are parallel independent if they are mutually weakly parallel independent.

$$\begin{tikzcd}
R_2 \arrow[d, "m_2^*"'] & L_2 \arrow[l, "r_2"'] \arrow[dr, "m_2"] \arrow[drrr, "\acute{m}_2"] & & L_1 \arrow[r, "r_1"] \arrow[dl, "m_1"'] & R_1 \arrow[d, "m_1^*"] \\
H_2 & & G \arrow[ll, "r_2^*"'] \arrow[rr, "r_1^*"] & & H_1
\end{tikzcd}$$

$$d_1 = G \xRightarrow{\nabla_1, m_1} H_1 \text{ and } d_2 = G \xRightarrow{\nabla_2, m_2} H_2$$

are two parallel independent rule applications

□

Two rule applications are weakly sequentially independent if the first application does not create elements that are required by the match of the second rule application. The two rule applications are sequentially independent if the second is also weakly sequentially independent to the first (Ehrig, Heckel, Korff et al. 1997).

Definition 6.4 (Sequential Independence). Let $d_1 = G \xRightarrow{\nabla_1, m_1} H_1$ and $\acute{d}_2 = h_1 \xRightarrow{\nabla_2, \acute{m}_2} X$ be two rule applications using conditional rules $\nabla_1 = (r_1, C_1, T_1, M_1, N_1)$ and $\nabla_2 = (r_2, C_2, T_2, M_2, N_2)$ and the matches m_1 and m_2 for the simple rules r_1 and r_2 respectively. The rule application $\acute{d}_2$ is weakly sequentially independent of d_1 if $m_2 = (r_1^*)^{-1} \circ \acute{m}_2 : L_2 \longrightarrow G$ is a match for r_2 satisfies C_2 and T_2. The rule applications d_1 and d_2 are sequentially independent if d_1 is also weakly sequentially independent of the corresponding rule application $d_2 = G \xRightarrow{\nabla_2, m_2} H_2$.

$$\begin{tikzcd}
L_1 \arrow[r, "r_1"] \arrow[d, "m_1"] & R_1 \arrow[dr, "m_1^*"] & & L_2 \arrow[r, "r_2"] \arrow[dlll, "m_2"'] \arrow[dl, "\acute{m}_2"'] & R_2 \arrow[d, "\acute{m}_2{}^*"] \\
G \arrow[rr, "r_1^*"] & & H_1 \arrow[rr, "\acute{r}_2{}^*"] & & X
\end{tikzcd}$$

$$d_1 = G \xRightarrow{\nabla_1, m_1} H_1 \text{ and } d_2 = G \xRightarrow{\nabla_2, m_2} H_2$$

are two sequencially independent rule applications

□

The proof of categorical description from (Ehrig, Heckel, Korff et al. 1997) regarding many parallelism concepts such as parallel independence and sequential independence are provided in Appendix D.

This independence of rule applications can be used to formulate the notion of Local Church-Rosser in Theorem 6.5 below (Church and Rosser 1936).

Theorem 6.5 (Local Church-Rosser)**.** *Let* $G \xRightarrow{\nabla_1, m_1} H_1$ *and* $G \xRightarrow{\nabla_2, m_2} H_2$ *be two parallel independent conditional rule applications. Then there exist a graph* X *and graph transformations* $H_1 \xRightarrow{\nabla_1, \acute{m}_1} X$ *and* $H_2 \xRightarrow{\nabla_2, \acute{m}_2} X$ *such that* $G \xRightarrow{\nabla_1, m_1} H_1 \xRightarrow{\nabla_2, \acute{m}_2} X$ *and* $G \xRightarrow{\nabla_2, m_2} H_2 \xRightarrow{\nabla_1, \acute{m}_1} X$ *are sequentially independent conditional rule applications. Let* $G \xRightarrow{\nabla_1, m_1} H_1 \xRightarrow{\nabla_2, \acute{m}_2} X$ *be two sequentially independent conditional rule applications then there exit a graph* H_2 *and conditional rule applications* $G \xRightarrow{\nabla_2, m_2} H_2 \xRightarrow{\nabla_1, \acute{m}_1} X$ *such that* $G \xRightarrow{\nabla_1, m_1} H_1$ *and* $G \xRightarrow{\nabla_2, m_2} H_2$ *are parallel independent.*

$$\begin{tikzcd}
 & G \arrow[dl, Rightarrow, "{r_1, m_1}"'] \arrow[dr, Rightarrow, "{r_2, m_2}"] \arrow[dd, Rightarrow, "{r_1 + r_2, m}" description] & \\
H_1 \arrow[dr, Rightarrow, "{r_1, \acute{m}_1}"'] & & H_2 \arrow[dl, Rightarrow, "{r_2, \acute{m}_2}"] \\
 & X &
\end{tikzcd}$$

Local Church Rosser. For the proof see Appendix D. □

The rule templates in PROV-GTS are constructed from graph transformation rules that are always parallel independent in their applications and sharing similar graph topologies. For instance, the rules $\boxed{\textbf{8-23}}$***delGenEnt*** and $\boxed{\textbf{8-24}}$***delUseEnt*** based on template8 delete the edges *gen* and *use* connected to a restricted activity, respectively. The matches of these two rules are never overlapped in their existential parts; they delete two different edges, *gen* and *use*. In addition, the *use* edge in the existential part of the universal-existential condition (e_1, u_1) in rule $\boxed{\textbf{8-23}}$***delGenEnt*** cannot be deleted by $\boxed{\textbf{8-24}}$***delUseEnt*** since the condition requires the existence of an *info* edge. For the same reason, the *gen* edge in the existential part of (e_1, u_1) in rule $\boxed{\textbf{8-24}}$***delUseEnt*** cannot be deleted by $\boxed{\textbf{8-23}}$***delGenEnt***. Figure 6.1 depicts two parallel independent rule applications and shows how the same rule applications can be applied sequentially with the same transformed graph.

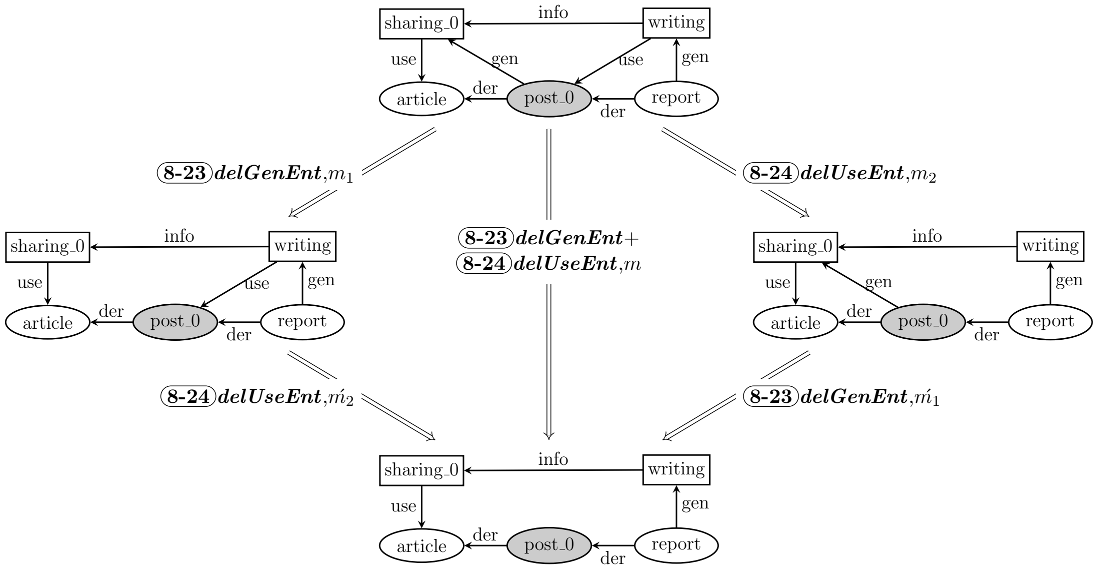


Figure 6.1: An example on local Church-Rosser theorem using two rule based on template8

6.4 Critical Pairs, Local Confluence and 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 are applied, i.e. the system is confluent. Confluence means that for each pair of terminating rule applications $G \overset{*}{\Rightarrow} H_1$ and $G \overset{*}{\Rightarrow} H_2$ the resulting graphs H_1 and H_2 are equal or isomorphic. In Figure 6.2(a) graph G is used as an initial graph, when transformation rules are applied in different orders, it results in the intermediate graphs H_1 and H_2. A terminating system is confluent if the transformation always finishes with the same final graph X. The system is locally confluent if the same condition holds for all pairs of rule applications, as shown in Figure 6.2(b).

The Local-Church Rosser shows that for any two parallel rule applications $G \xRightarrow{\nabla_1,m_1} H_1$ and $G \xRightarrow{\nabla_2,m_2} H_2$ there are two sequentially independent rule applications $H2 \xRightarrow{\nabla_1,\acute{m}_1} X$ and $H_1 \xRightarrow{\nabla_2,\acute{m}_2} X$ such that the two rules r_1 and r_2 can be applied in arbitrary order (Plump 1995; Prange and Ehrig 2007).

Definition 6.6 (Confluence)**.** A graph transformation system is called confluent if for all graph transformations $G \overset{*}{\Rightarrow} H_1$ and $G \overset{*}{\Rightarrow} H_2$ there is a graph X and graph transformations $H_1 \overset{*}{\Rightarrow} X$ and $H_2 \overset{*}{\Rightarrow} X$. The system is locally confluent if the same condition holds for all pairs of rule applications $G \Rightarrow H_1$ and $G \Rightarrow H_2$. □

A graph transformation system is confluent if the system is terminating and any pair of rule applications are locally confluent. We already showed the termination of PROV-GTS in Section 6.2. To prove that any pair of rule applications are locally confluent,

$$\begin{tikzcd}
& G \arrow[dl, Rightarrow, "*"'] \arrow[dr, Rightarrow, "*"] & \\
H_1 \arrow[dr, Rightarrow, "*"'] & & H_2 \arrow[dl, Rightarrow, "*"] \\
& X &
\end{tikzcd}$$

(a) Confluence

$$\begin{tikzcd}
& G \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
H_1 \arrow[dr, Rightarrow, "*"'] & & H_2 \arrow[dl, Rightarrow, "*"] \\
& X &
\end{tikzcd}$$

(b) Local confluence

FIGURE 6.2: Confluence in graph transformation

we need to show that all critical pairs of rule applications are strictly confluent (Prange and Ehrig 2007).

Any two conflicting rule applications represent a critical pair. Two rule applications are in conflict if at least one of the following reasons fulfilled (Ehrig, Ehrig, Prange et al. 2006b):

- *delete-use conflict*: when one of the rule applications delete from the graph what is required by the other rule application.
- *produce-forbid conflict*: when one rule application creates an element that is prohibited by a negative application condition of the other rule.
- *change-attribute conflict*: one rule application changes attributes that are required by the match of the other rule application.

Since the graph transformation rules in PROV-GTS deletes or creates graph elements (nodes and edges), only the two first reasons are relevant. To show that the system is locally confluent, all critical pairs have to be strictly confluent. This means that for the critical pair $H_1 \overset{*}{\Leftarrow} G \overset{*}{\Rightarrow} H_2$, there is graph $\acute{G}$ together with rule applications $H_1 \overset{*}{\Rightarrow} \acute{G}$ and $H_2 \overset{*}{\Rightarrow} \acute{G}$.

In Section 4.11, all the potential conflicts between the rule applications have been solved by expanding the LHS, adding NACs, or changing the confidentiality level values of certain nodes. There are still some other conflicts that represent trivial critical pairs. Two rule applications are a trivial critical pair if they produce the same resulting graph. Since the conflicting rule applications are all trivial critical pairs, only the overlapping between the LHSs of the rules are considered. Some graph transformation rules in PROV-GTS delete or create similar graph elements but with different application conditions. For example, the two rules $\boxed{\textbf{4-10}}$***delAttNoOut*** and $\boxed{\textbf{6-17}}$***delInAtt*** delete an *attr* edge when the target agent node of the edge is restricted with two different sets of conditions. The former deletes the *attr* edge when the agent has no outgoing edges, while the latter deletes it when there exists an activity which connects between the source entity and the target agent of the *attr* edge. Both rules are applicable on the same *attr* edge if their matches share that *attr* edge, the restricted agent has no outgoing edges and the activity connects the entity and the agent exists. In this case, either rules can be applied and the same resulting graph is produced, as shown in Figure 6.3 by using a subgraph of OnlinePost example. This trivial critical pair of rule applications is safe since the resulting graph is the same for both transformation steps. The set of trivial critical pairs in PROV-GTS is shown in Table 6.2.

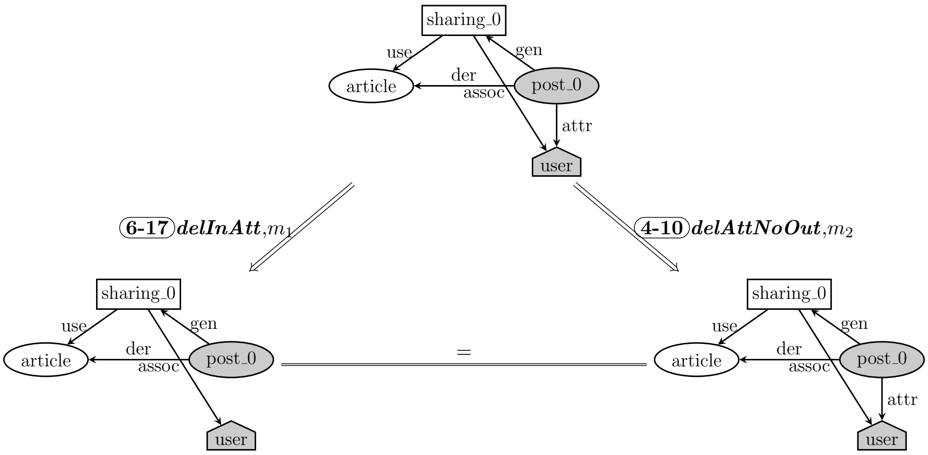


FIGURE 6.3: Trivial critical pairs in PROV-GTS - applications of two different rules **(6-17)*delInAtt*** and **(4-10)*delAttNoOut*** result in same transformed graph

In addition, the rule **(6-19)*delOutAtt*** deletes an *attr* edge between a restricted entity and an agent when the activity that connects the entity to the agent exists. The *attr* edge in Figure 6.4 between the nodes *post_0* and *user* can be deleted by any of the *attr* deletion rules (**(4-10)*delAttNoOut***, **(6-17)*delInAtt*** or **(6-19)*delOutAtt***). In addition, regarding the restricted entity *post_0*, an *info* edge can be added between the activities *writing* and *sharing_0* using the rule **(2-4)*crtInfUseGen***. After deleting the *attr* edge and creating the *info* edge, the two rules **(8-23)*delGenEnt*** and **(8-24)*delUseEnt*** can be applied in parallel for deleting the *gen* edge between *post_0* and *sharing_0* and the *use* edge between *writing* and *post_0*, respectively. These two deletion rule applications isolate the restricted entity *post_0* and result in the final transformed graph. We can conclude from the graph transformation steps shown in Figure 6.4 that any sequence of applicable transformation rules, which applied on the same start provenance graph, is always end up with the same final transformed graph. This is true for all critical pairs in PROV-GTS which proves that all critical pairs are strictly confluent. The proof of termination and strictness of local confluence of all the critical pairs in PROV-GTS indicate that the system is confluent.

Table 6.2: The set of trivial critical pairs.

Critical pairs	**Action**	**Reason**
1-1 ***crtActOutDer*** **1-3** ***crtActInDer***	create an activity for a der edge	if both entities are restricted.
3-5 ***delInfNoIn*** **3-6** ***delInfNoOut***	delete an *info* edge.	if both activities are restricted, the informed activity has no incoming edges, and the informant activity has no outgoing edges.
5-14 ***delOutInf*** **5-15** ***delInInf***	delete an *info* edge.	if both activities are restricted.
3-5 ***delInfNoIn*** **3-6** ***delInfNoOut*** **5-14** ***delOutInf*** **5-15** ***delInInf***	delete an *info* edge.	if both activities are restricted, the informed activity has no incoming edges, the informant activity has no outgoing edges, and there is a non-restricted entity connecting the two activities.
4-7 ***delDerNoIn*** **4-9** ***delDerNoOut***	delete a *der* edge.	if both entities are restricted, the source entity has no incoming edges, and the target entity has no outgoing edges.
6-16 ***delInDer*** **6-18** ***delOutDer***	delete a *der* edge.	if both entities are restricted.
4-7 ***delDerNoIn*** **4-9** ***delDerNoOut*** **6-16** ***delInDer*** **6-18** ***delOutDer***	delete a *der* edge.	if both entities are restricted, the source entity has no incoming edges, the target entity has no outgoing edges, and there is an activity connecting the two entities.
4-8 ***delAttNoIn*** **4-10** ***delAttNoOut***	delete an *attr* edge.	if the entity has no incoming edge, the agent has no outgoing edges, and both nodes are restricted.
6-17 ***delInAtt*** **6-19** ***delOutAtt***	delete an *attr* edge.	if the entity and the agent are restricted.
4-8 ***delAttNoIn*** **4-10** ***delAttNoOut*** **6-17** ***delInAtt*** **6-19** ***delOutAtt***	delete an *attr* edge.	if the entity has no incoming edges, the agent has no outgoing edges, both nodes are restricted, and there exists an activity that connects between them.
4-11 ***delDelNoIn*** **4-12** ***delDelNoOut***	delete a *del* edge.	if the source agent has no incoming edges, the target agent has no outgoing edges, and both nodes are restricted.
4-13 ***delAssocNoOut*** **8-22** ***delAssocAct***	delete an *assoc* edge.	if the agent has no outgoing edges and both nodes are restricted.

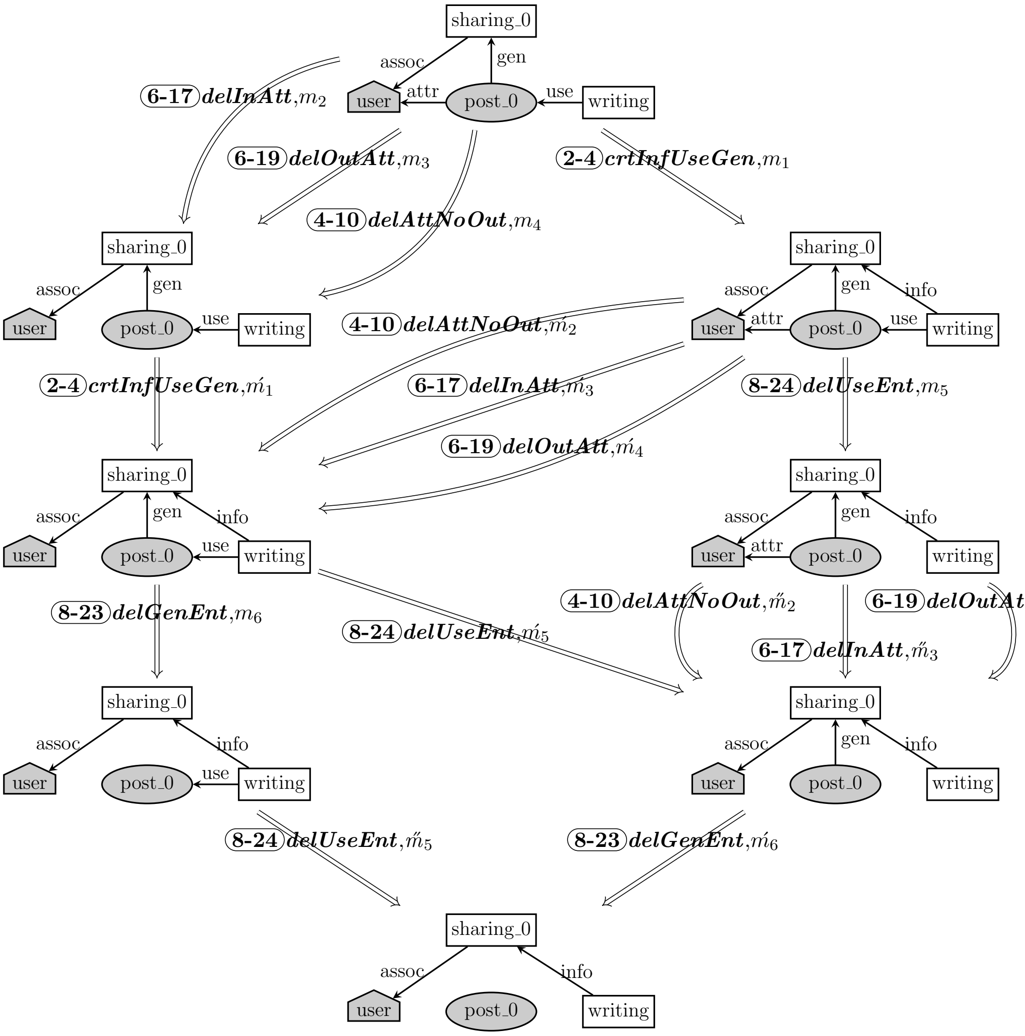

sharing_0
assoc
gen
attr
use
user
post_0
writing
info
6-17 *delInAtt*,m_2
6-19 *delOutAtt*,m_3
4-10 *delAttNoOut*,m_4
2-4 *crtInfUseGen*,m_1
4-10 *delAttNoOut*,$\acute{m}_2$
2-4 *crtInfUseGen*,$\acute{m}_1$
6-17 *delInAtt*,$\acute{m}_3$
8-24 *delUseEnt*,m_5
6-19 *delOutAtt*,$\acute{m}_4$
8-23 *delGenEnt*,m_6
8-24 *delUseEnt*,$\acute{m}_5$
4-10 *delAttNoOut*,$\H{m}_2$
6-19 *delOutAtt*,$\H{m}_4$
6-17 *delInAtt*,$\H{m}_3$
8-24 *delUseEnt*,$\H{m}_5$
8-23 *delGenEnt*,$\acute{m}_6$

FIGURE 6.4: Different sequences of critical pairs produce the same final transformed graph despite the order in which the rules have been applied

6.5 Complexity Analysis

In this section, we analyse the complexity of PROV-GTS in terms of the number of restricted nodes regarding the best and worst cases. In the best case graph scenario, each edge connected to a restricted entity requires only one rule application, there is no need to create activities using template1 or *info* edges using template2 to enable the rules that delete the edges connected to restricted nodes. This means that the time complexity is linear and equals the number of edges incident to the restricted nodes, that is $\sum_{i=1}^{N}(i-1) = N(N-1)/2$ where N is the number of restricted nodes. For the worst case, we need to consider a graph where all nodes are restricted entities with maximum connectivity between the nodes, i.e. each entity is connected to each node in the graph as long as the graph is acyclic. In other words, the worst case graph is the valid graph in which each two entities are linked to each other by a *der* edge, as illustrated in Fig. 6.5.

We suppose that the transformation rules are applied on this graph using the layered-based graph grammar that has been used in proving the termination of PROV-GTS as shown in Section 6.2. This guarantees the worst case scenario by creating the required nodes and edges before starting deleting edges. The number of added activities and *info* edges are directly proportional to the number of edges connected to the *restricted* nodes.

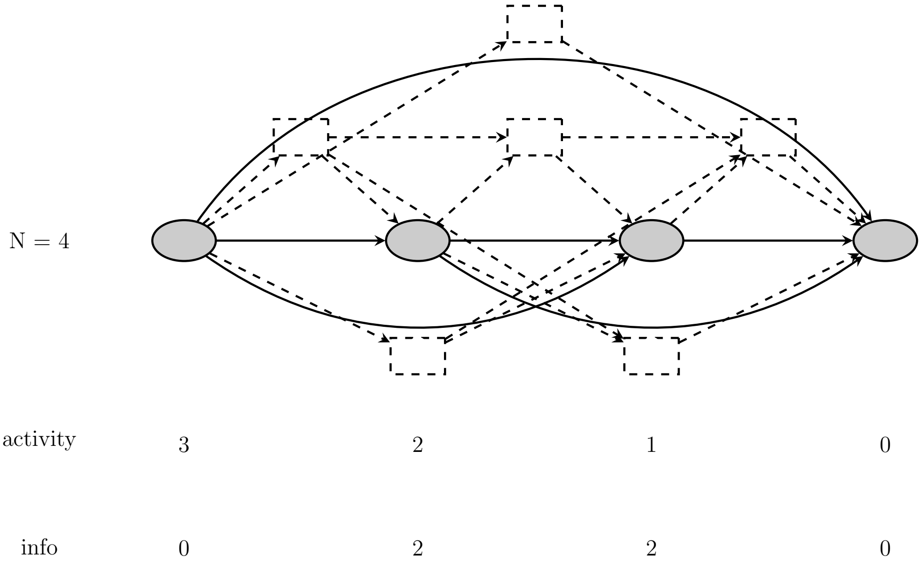


Figure 6.5: Worst case in PROV-GTS ($W = 4$) – original graph (solid lines), added activities and *info* edges (dashed lines)

There will be a number of activity creation rule applications equal to the number of *der* edges $\sum_{i=1}^{N}(i-1) = N(N-1)/2$ and an *info* edge between each two activities that used and generated the same entity. The total number of *info* edges that must be created equals to $(\sum_{i=1}^{N}(N-i)(i-1))$. There are two edges for each created activity, in total the number of created *use* edges as well as the created *gen* edges is equal to the number of the *der* edges. This indicates that the time complexity for the worst case graph is cubic N^3, where N is the number of restricted nodes.

6.6 Summary

In this chapter, many concepts relevant to algebraic graph transformation systems are investigated including termination, parallelism and confluence. We prove that PROV-GTS is terminating using layer-based graph grammars. The parallel independence and sequential independence are analysed followed by the proof of critical pair strictness. The proofs of termination and strictness of critical pairs indicates that the system is also (locally) confluent. In addition, the time complexity of the system is analysed regarding the best and worst case scenarios. In the next chapter, we continue proving various properties of PROV-GTS such as graph integrity, path preservation, and graph connectivity.

Chapter 7

Provenance Graph Integrity and Connectivity

7.1 Introduction

So far, we have presented the construction of the PROV graph transformation rules and rule templates. We provide sufficient proofs for many general graph transformation properties such as termination, parallelism and confluence on template basis. In this chapter, we investigate other properties relevant to the semantics of the PROV data model such as graph integrity by avoiding false dependencies and false independencies, graph validity by creating nodes and edges according to properties from PROV data model, and path preservation. 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. We start with graph integrity in Section 7.2 followed by proofs of path preservation in Section 7.3. Section 7.4 provides the proof of system's non-excessiveness.

7.2 Graph Integrity

As provenance is crucial in validating the origin and quality of data and enabling its reusability, it is important to preserve the integrity of provenance graphs when hiding restricted nodes that expose confidential information. Such provenance graph integrity has been guaranteed by constructing the template-based PROV-GTS based on the PROV data model so that it avoids false dependencies and false independencies and it preserves paths. Suppose G is the original graph, H is the graph transformed by template-based PROV-GTS rules and $H\backslash G$ represents the elements (nodes and edges) in H that are not in G.

Theorem 7.1 (No False Dependency)**.** *The elements in $H \backslash G$ can be inferred from graph G based on the properties of the* PROV *data model, i.e. there are no false dependencies.*

Proof. Given any concrete creation rule ($L \longrightarrow R$) with NAC, which is constructed from a PROV property $C \longrightarrow E$ and belongs to a template with the simple abstract rule $L_A \longrightarrow R_A$, as shown in the diagram below, and also given that both morphisms $c : C \longrightarrow L$ and $e : E \longrightarrow R$ are monos (inclusion) after ignoring the confidentiality level nodes and edges, we need to prove that the followings are true in order to prove that no false dependency are caused by the creation rules:

1. $f : L_A \longrightarrow C$ and $h : R_A \longrightarrow E$ for each creation rule $L \longrightarrow R$ are monos, i.e. $\psi_L = l \circ f$ and $\psi_R = r \circ h$.
2. $p \circ f = h \circ r'_A$.

$$\begin{tikzcd}
C \arrow[r, "p"] \arrow[d, "l"'] \arrow[dr, phantom, "="] & E \arrow[d, "r"] \\
L \arrow[r, "r'"] \arrow[dr, phantom, "="] & R \\
L_A \arrow[u, "\psi_L"] \arrow[r, "r'_A"] \arrow[uu, bend left=60, "f"] & R_A \arrow[u, "\psi_R"'] \arrow[uu, bend right=60, "h"']
\end{tikzcd}$$

The morphisms ψ_L and ψ_R are monos if and only if the morphisms f and h are monos. Since the morphisms ψ_L and ψ_R are monos by definition, at least f and h need to be monos such that $\psi_L = l \circ f$ and $\psi_R = r \circ h$. Since p and r'_A are monos by definition, then the second requirement $p \circ f = h \circ r'_A$ is fulfilled, and the diagram above commutes. These two requirements are satisfied by all the rule instances of the creation templates in PROV-GTS. □

For example, the instance rule **1-3** ***crtActInDer*** of template1 is initially constructed from the PROV property prov3 and then an extra condition was attached to left-hand side (LHS) and right-hand side (RHS) of of the rule which indicates that the rule can be applied only when the restricted entity has an incoming edge. The abstract rule of the template has the same structure as the instance rule but consists only of abstract nodes of type *node*. The diagram that shows the relations between the PROV property prov3, the instance rule **1-3** ***crtActInDer***, and the template template1 is depicted in Figure 7.1. All nodes in the PROV property, the instance rule, and the abstract rule are considered as *plain* nodes since only graph nodes and edges are relevant to the false dependency prevention.

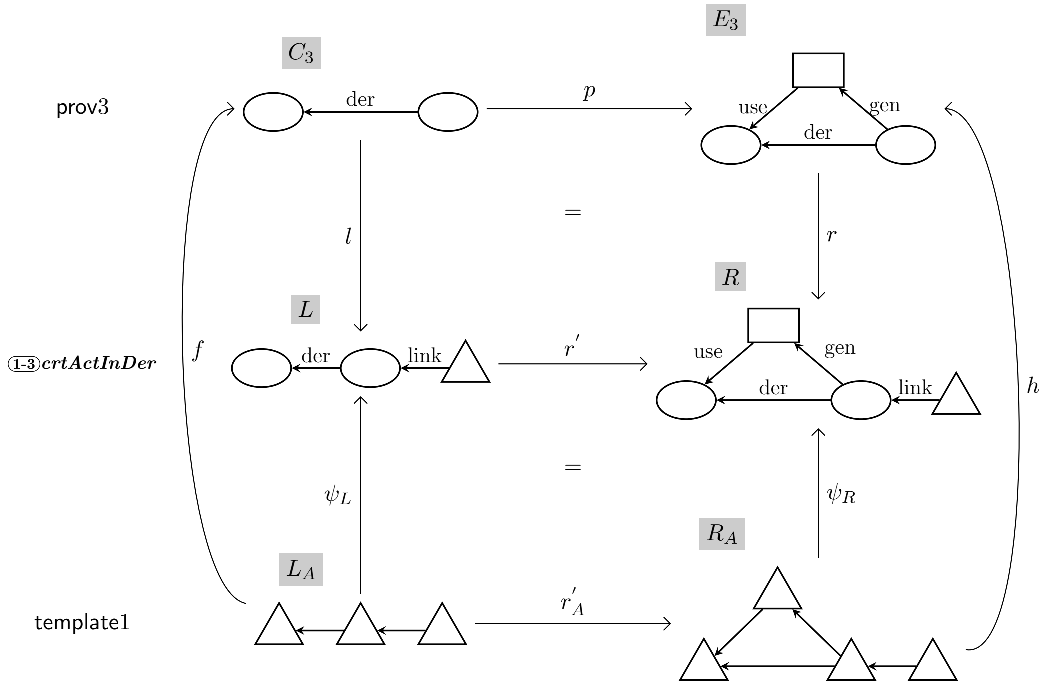


FIGURE 7.1: No false dependency: the relation between the PROV property, the instance rule (1-3)***crtActInDer***, and the template template1 after ignoring the confidentiality level nodes and edges

Theorem 7.2 (No False Independency)**.** *Any two plain nodes in G are preserved in H along with all explicit and inferred edges connecting them.*

Proof. Since the creation templates create *activity*, *use*, *gen*, *assoc*, and *info* regarding the restricted entities, in the following, we examine each rule instance of the deletion templates regarding restricted entities and show how they avoid false independencies by providing adequate universal-existential conditions:

- template4: This template deletes *der* or *attr* only when the restricted entity has no incoming or outgoing edges.
- template6: This template deletes *der* or *attr* only when there is an activity connecting between the two entities or the entity and the agent. This activity can be created by template1.
- template8: The condition (u_0, e_0) checks the existence of an *info* edge for each incoming *use* when deleting a *gen* edge and for each outgoing *gen* when deleting a *use* edge. This *info* can be created by template2. Similarly, the condition (u_1, e_1) ensures that for each incoming or outgoing *der* or *attr* there exists an activity. This activity can be created by template1. These two universal-existential conditions are sufficient to check all possible graph patterns that include the deleted edge.

Similarly, adequate conditions are provided for the edge deletion rules regarding activities and agents.

□

It is crucial for graph transformation systems to produce valid graphs. By valid graph we mean the graph was defined over the PROV type graph TG. In PROV-GTS the transformed graph must still be a valid PROV graph over the type graph TG. To this end, the creation rules in PROV-GTS have been constructed based on PROV model properties such that only valid PROV model elements are created by the system.

Theorem 7.3 (Graph Validity Preservation)**.** *If the original graph G is a valid graph over the type graph TG then the transformed graph H is a valid graph over the same type graph TG.*

Proof. Follow from the proof of Theorem 7.1 where all nodes and edges created by PROV-GTS are valid components of the type graph TG justified by the properties of PROV data model. □

7.3 Path Preservation

In provenance graphs, paths consist of elements and relations that led to the current state of an object. So, preserving those paths is important when transforming provenance graphs such that same paths, after obscuring any private information, can be reached when querying over paths in provenance graphs.

Theorem 7.4 (Path Preservation)**.** *Any path between two plain nodes in G is preserved in H.*

Proof. The proof of *No False Independency* (Theorem 7.2) shows that the relation between the nodes adjacent to restricted entities are preserved before deletion and path preservation is dependency preservation along paths. So, avoiding false independencies along paths result in preserving those paths. Preventing false independencies is a subset of the overall process of path preservation. Now we examine the situation when the restricted node is an activity or an agent.

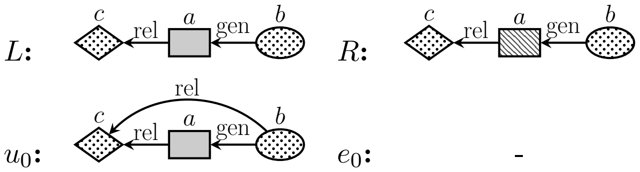
- For activities:
  - template3: This template deletes an *info* edge when the restricted activity has no incoming or outgoing edges.
  - template5: This template deletes an *info* edge when there exists an entity used by the informed activity and generated by the informant activity.

  - templates7, 8: The condition (u_0, e_0) checks the existence of an *der* 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. Both conditions are sufficient to check that the paths between all the nodes adjacent to the source and the target of the deleted edge are preserved.

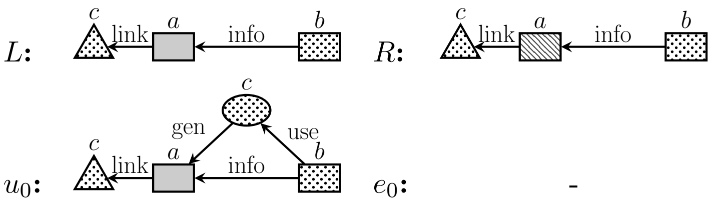
- For agents:
  - template4: This template deletes *attr* and *assoc* edges when the agent is restricted and has no outgoing edges.
  - template6: This template deletes an *attr* edge when there exists an activity generated the entity and was associated with the agent.

In addition, not all edges incident to restricted activities or agents can be deleted without violating the connectivity of the graph. Sometimes deleting some edges result in cutting paths between the nodes adjacent to the source and target nodes of the deleted edge. If this is the case, then those edges are preserved and the activities and agents connected to them are anonymized according the following graph patterns which can be used for constructing transformation rules that anonymized restricted nodes. These rules could have been added to PROV-GTS, but rather we apply the edge deletion rules, eliminate the isolated nodes, and then anonymize the remaining restricted nodes. We provide these rules for the purpose of explanation only.

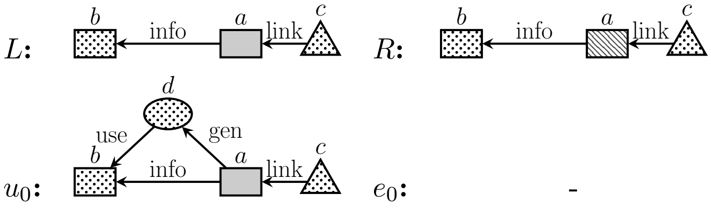
- For activities:
  - Regarding a restricted activity that has incoming *gen* edge from an entity and an outgoing edge (*use* or *assoc*) to an artifact (entity or agent), these incoming and outgoing edges cannot be deleted if there is not an edge connecting the entity to the artifact.



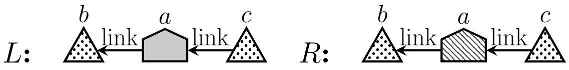
  - An incoming *info* edge to a restricted activity that has outgoing edges cannot be deleted if the entity connects the informed and informant activities does not exist in the graph. As a result, any outgoing *use* or *assoc* edges cannot be deleted from the graph.



- An outgoing *info* edge from a restricted activity that has incoming edges cannot be deleted if the entity connects the informed and informant activities does not exist in the graph. As a result, any incoming *gen* edge cannot be deleted from the graph.



- For agents:
  - Agents in PROV-GTS can only be isolated when they have no incoming and/or outgoing edges. Some edges connected to the agents that have both incoming and outgoing edges, cannot be deleted from the graph, as shown in the following graph pattern:



□

As part of the preservation of provenance graph integrity, it is important to ensure that no false paths emerge as a consequence of the graph transformation process performed by PROV-GTS.

Theorem 7.5 (No False Paths)**.** *There is a path connecting two plain nodes in H if and only if there exists a path between them in G.*

Proof. The proofs of *No False Dependency* (Theorem 7.1) and *Path Preservation* (Theorem 7.4) can be used to prove that no false paths are created by PROV-GTS, as follows:

- From the proof of *No False Dependency* (Theorem 7.1), only the elements that can be inferred from graph G are created in graph H by template1 and template2 based on the PROV properties prov2 and prov3.

- Based on the proof of *Path Preservation* (Theorem 7.4), paths between the plain nodes in G are preserved in H.

□

7.4 Information Preservation

The trade-off between privacy protection and provenance information preservation is essential for any provenance graph transformation system. In PROV-GTS, the restricted nodes are isolated from provenance graphs as long as the integrity and connectivity of the graph are not violated. This balance between obscuring restricted information and providing useful provenance graphs is achieved by avoiding false independencies and preserving paths.

Theorem 7.6 (No Excessiveness). *No additional edges and nodes will be removed from provenance graphs by* PROV-GTS *beyond the edges connected to restricted nodes.*

Proof. PROV-GTS deletes only the edges connected to the restricted nodes in the following sense:

- Each deletion template deletes a single edge only when connected to a restricted node.
- From the proof of *No False Independency* (Theorem 7.2) the relations between nodes adjacent to the source and target of the deleted edges are preserved.

□

7.5 Summary

In this chapter, the formal proofs for graph integrity, path preservation, graph information preservation are presented. The proofs show the capability of PROV-GTS in maintaining graph integrity and preserving paths by avoiding false dependency and false independency. These preservations are done by adding to the graph only the nodes and the edges that can be inferred from it based on properties from the PROV data model. Creating nodes and edges to prevent false independency, attaching application conditions to the deletion rules, and anonymizing the nodes that cannot be isolated from the graph guarantee path preservation. In addition, this strict graph reduction, by avoiding false dependencies and false independencies and preserving paths, prohibits excessive information to be discarded from provenance graphs. In the next chapter, we evaluate the performance of the system's implementation using data sets from variety of applications that produce provenance information.

Chapter 8

Evaluation and Performance

8.1 Introduction

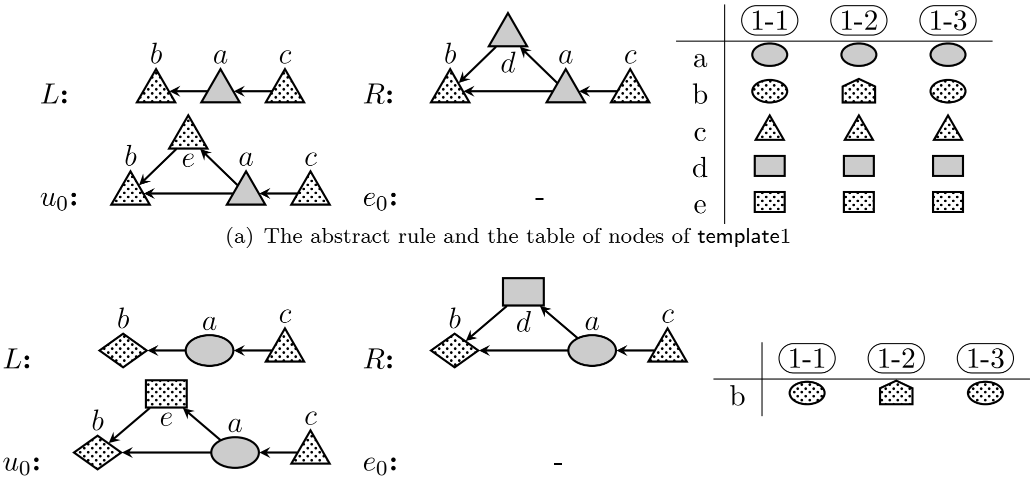
In Chapter 7, the proofs of graph integrity, path preservation, and provenance information preservation are presented. In order to evaluate the performance of PROV-GTS, it is important to integrate it with other systems that provide access to provenance information which may contain confidential data like private personal details. The confidentiality of provenance information can be determined according to privileges defined by access control systems or based on users' preferences. The transformation rules can be used to hide or anonymize this information. In this chapter, variety of measurements such as graph connectivity, number of transformation steps, and average degree are used to evaluate PROV-GTS's performance. The evaluation process are applied on datasets from three different applications that process provenance information, namely, AtomicOrchid, Ride share, and W3C working group email archive. This performance evaluation demonstrates a high connectivity preservation with minimum graph reduction. For example, PROV-GTS preserves 90% graph connectivity for a provenance graph with 10% restricted nodes. In Section 8.2, the implementation of PROV-GTS in Prolog language is demonstrated. Prerequisites to the evaluation process are presented in Section 8.3. In Section 8.4, the data sets used in this analysis are described. Finally, Section 8.5 provides a set of hypotheses used in conducting the evaluation process.

8.2 Implementation

In order to simplify the implementation of the template-based PROV-GTS, we replace each node in the abstract conditional rule of each template with the lowest common ancestor (LCA) (Aho, Hopcroft and Ullman 1973) of the corresponding nodes in the concrete instance rules of the template based on the node inheritance graphs. Similarly, we define the lowest common ancestor for the confidentiality level of each node from the confidentiality inheritance graph.

Definition 8.1 (Lowest Common Anscestor LCA)**.** Regarding two nodes n and m in a tree or directed acyclic graph (DAG), the lowest common ancestor (LCA) is the lowest node that has both n and m as descendants, where each node is a descendant of itself. □

For example, regarding template1, the node a in the abstract rule is instantiated to restricted entities in all the rule instances while the node b has either entities or agents, that is the *artifact* abstract nodes, as corresponding nodes in the rule instances. The reduced abstract rule of template1 (Figure 8.1(a)) with its reduced node table are shown in Figure 8.1(b).



(a) The abstract rule and the table of nodes of template1

(b) The abstract rule and the reduced table of nodes of template1 after replacing each node with its LCA based on the inheritance graphs

FIGURE 8.1: Constructing the lowest common anscestor LCA for template1 using the inheritance graphs

8.2.1 Representing the PROV model graphs in Prolog

To apply the rule templates in Prolog language, first we need to represent provenance graphs as Prolog predicates. We define a PROV-N like syntax to express the graph components of the PROV data model suitable and readable by Prolog language which enables us to query over provenance graphs. To this end, we parse the PROV documents using the Definite Clause Grammar DCG package deployed as part of SWI-Prolog distribution. As shown in Figure 8.2, the PROV-N representation has been converted to appropriate Prolog predicates. For instance, the PROV-N statement '`entity(ex:report)`' represented as '`entity(ex:report).`', where the only difference is the dot (.) at the end of the Prolog predicate. The restricted nodes are indicated by using a predicate used to define attributes of the form of `Name=Value %% Type` as follows:

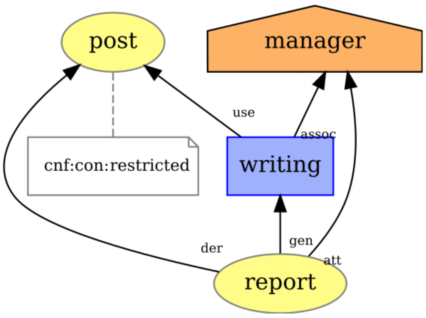
```
attributed(Id, attribute(Name, Value, Type)).
```

Where `Id` is the identifier of the node and `Name`, `Value`, and `Type` indicate the name, the value and the datatype of the attribute, respectively. For example, `attributed(ex:post, attribute("cnf:con", "cnf:restricted", "prov:QUALIFIED_NAME"))` used to show that the entity `ex:post` is a restricted node.

```
document
prefix ex <http://example/>
prefix cnf <cnf:>

entity(ex:report)
entity(ex:post, [cnf:con="restricted"])
activity(ex:writing, -, -)
agent(ex:manager)

wasDerivedFrom(ex:report, ex:post)
wasGeneratedBy(ex:report, ex:writing,-)
used(ex:writing, ex:post, -)
wasAttributedTo(ex:report, ex:manager)
wasAssociatedWith(ex:writing, ex:manager, -)
endDocument
```



(a) PROV-N and graph representations of a provenance graph

```
prefix("ex", "<http://example/>").
prefix("cnf", "<cnf:>").

entity("ex:report").
entity("ex:post").
activity("ex:writing").
agent("ex:manager").

wasDerivedFrom("ex:report", "ex:post").
wasGeneratedBy("ex:report", "ex:writing").
used("ex:writing", "ex:post").
wasAttributedTo("ex:report", "ex:manager").
wasAssociatedWith("ex:writing", "ex:manager").
attributed("ex:post", attribute("cnf:con", "cnf:restricted",
    "prov:QUALIFIED_NAME")).
```

(b) Prolog representations of a provenance graph

FIGURE 8.2: PROV-N representation vs Prolog representation

8.2.2 Implementing the template based PROV-GTS in Prolog

Each reduced template, when each node in the abstract rule was replaced with its LCA, is implemented by writing appropriate Prolog rules for the reduced abstract rule of the template and each one of its rule instances taking into account the direct and reverse edges. The following example shows the implementation of the reduced version of template1 and its instance rules. First, the Prolog rule `template1` finds an entity node `A` which has an outgoing edge to a node `B`. Then the `template1_direct_or_reverse(A, B)` is used to indicate which instance rule must be called, the one with direct edges or the one with reverse edges. This has been done by checking which of the two nodes is restricted, A or B, indicating the direction of edges direct or reverse respectively.

```
template1 :-
  entity(_, A),
  has_edge(A, B),
  template1_direct_or_reverse(A, B).

template1_direct_or_reverse(A, B) :-
  restricted(A),
  has_edge(_, A), % A has incoming edges
  !,
  template1_instances(A, B).

template1_direct_or_reverse(A, B) :-
  restricted(B),
  has_edge(B, _), % B has outgoing edges
  template1_instances(A, B).
```

In either situation, the Prolog rule `template1_instances` with A and B as arguments is called, which checks whether the node B is an entity or an agent (i.e. the edge $A \rightarrow B$ is a *der* or an *attr* edge. Before creating an activity between the two nodes, we need to check that there is not such an activity, i.e. implementing the negative application condition of template1 using the negation operator of the Prolog language (`\+`) followed by the predicate `has_activity`.

```
template1_instances(A, B):-
  entity(B),
  !,
  \+ has_activity(A, B), % NAC: there is no activity node connecting A and B
  create_activity_der(A, B).

template1_instances(A, B) :-
  agent(B),
  \+ has_activity(A, B), % NAC: there is no activity node connecting A and B
  create_activity_attr(A, B).
```

In the following, we provide a brief description for some of the Prolog rules used in PROV-GTS implementation to perform various tasks.

- Restricted nodes: A node is restricted if it has an attribute `"cnf:con"` with the value `"cnf:restricted"`.

  ```
  restricted(A) :-
    attributed(A, attribute("cnf:con", "cnf:restricted", _).
  ```

- Finding edges: The predicate `has_edge` is used to find an edge between two nodes. The edge may be any edge from the set of edge types of core PROV data model.

  ```
    has_edge(A, B) :-
      wasDerivedFrom(A, B); % ; is the or operator in Prolog
      wasAttributedTo(A, B);
      ...
  ```

- Finding activities: The predicate `has_activity` checks for the existence of an activity connecting two entity nodes or an entity and an agent.

```
has_activity(A, B) :-
  entity(A),
  entity(B),
  !,
  has_edge(A, C),
  has_edge(C, B),
  activity(C).

has_activity(A, B) :-
  entity(A),
  agent(B),
  has_edge(A, C),
  has_edge(C, B),
  activity(C).
```

- Creating an activity: First we need to generate an identifier for the new activity, which has been done by using the clause `random_codes(Id1, 10)` which generates a string of length 10. After that we use the predicate `assertz` to add the required statements to the Prolog knowledge base.

```
create_activity_der(A, B) :-
  random_codes(Id1, 10),
  atom_codes("cnf:", CNF),
  append(CNF, Id1, Id),
  assertz(activity(Id)),
  assertz(wasGeneratedBy(-, A, Id)),
  assertz(used(-, Id, B)),
  assertz(attributed(Id, attribute("cnf:con", "cnf:restricted",
           "prov:QUALIFIED_NAME"))).

create_activity_attr(A, B) :-
  random_codes(Id1, 10),
  atom_codes("cnf:", CNF),
  append(CNF, Id1, Id),
  assertz(activity(C, Id)),
  assertz(wasGeneratedBy(-, A, Id)),
  assertz(wasAssociatedWith(-, Id, B)),
  assertz(attributed(Id, attribute("cnf:con", "cnf:restricted",
           "prov:QUALIFIED_NAME"))).
```

- Universal-existential conditions: SWI-Prolog has the built-in predicate `forall(U, E)` for checking that each occurrence of the pattern `U` satisfies the condition `E`. The following code checks that for each `used(_, A1, E1)` and `wasGeneratedBy(_, E1, A2)` edges there exist another edge `wasInformedBy(_, A1, A2)` between the two activities `A1` and `A2`.

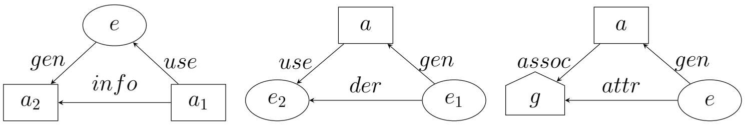
```
forall(
  ( used(_, A1, E1), wasGeneratedBy(_, E1, A2) ),
  wasInformedBy(_, A1, A2)
)
```

8.3 Preparation

Let G be the original graph and H be the transformed graph that has been generated by PROV-GTS using graph G as the initial graph. In the following, we provide some metrics from both graphs. These measurements are necessary in evaluating the performance of PROV-GTS by computing statistical information based on them.

The edges {*der, info, attr, del*} are related to extra information in provenance graphs. For example, the edge *der* are (optionally) related to an activity which generated the source entity using the target entity of the *der* edge. The activity along with the *gen* and *use* edges can be created if they do not already exist in the graph. We used this technique to bridge the gaps in transformed graphs that might be generated by deleting the edges connected to restricted nodes avoiding, by this, losing information and breaking paths. This process was aimed at saturating the graph by creating the required edges and nodes in order to preserve relations. The graph is fully-saturated if no nodes and edges can be inferred from the existing nodes and edges of the graph. These edges with their relevant edges and nodes formulate triangle-shaped graph patterns (Kwasnikowska, Moreau and Bussche 2015). In our proposed system, we consider only the subset {*der, info, attr*} of these edges, since we do not have any creation rules regarding *del* edges.

Definition 8.2 (Graph triangles)**.** The triangles $info_{tri}$, $attr_{tri}$, and der_{tri} represent derivation, attribution and communication triangles respectively.



$info_{tri}$ der_{tri} $attr_{tri}$

□

In case of *der* and *attr*, the optional activities in their definition are ignored by the above calculation, instead any existing activity connects the two entities or the entity and the agent are used.

8.3.1 Normalization degree

The normalization degree is the value that shows to what extent the graph is in normal form by comparing the number of triangles with the number of graph's {*der*} and {*attr*} edges plus the graph patterns that can be saturated, i.e. *use-gen* patterns. These edges and patterns are those that appear in the left-hand sides of the creation rules in PROV-GTS, while the triangles are in their right-hand sides. The normalization degree is ranging between 0 and 1, where 1 indicates that the graph is fully saturated while 0 shows that the graph has no triangles.

$$\text{normalization degree} = \frac{\text{number of triangles}}{\text{number of der} + \text{number of attr} + \text{number of (use-gen)}}$$

We avoid division by zero by calculating normalization degree only if the graph has *der*, *attr*, and/or *use-gen* arrows.

8.3.2 The average degree

In graph terminology, the degree of a node in a directed graph is the number of nodes connected to it by incoming or outgoing edges. However, for our system, the edges that can be saturated, i.e. *der* and *attr*, have different weights. Since those edges hold more information than others and deleting them result in hiding more information. In addition, in our system, deleting those edges may require more rule applications and condition satisfactions. In particular, for triggering the rules that delete the *der* and *attr* edges, we may need several creation rule applications. So, we give the edges *der* and *attr* the weight of two while the weight of the remaining edges will be one. Figure 8.3 depicts the degree of each node in OnlinePost graph using different weights for the edges. The degree of a node n is:

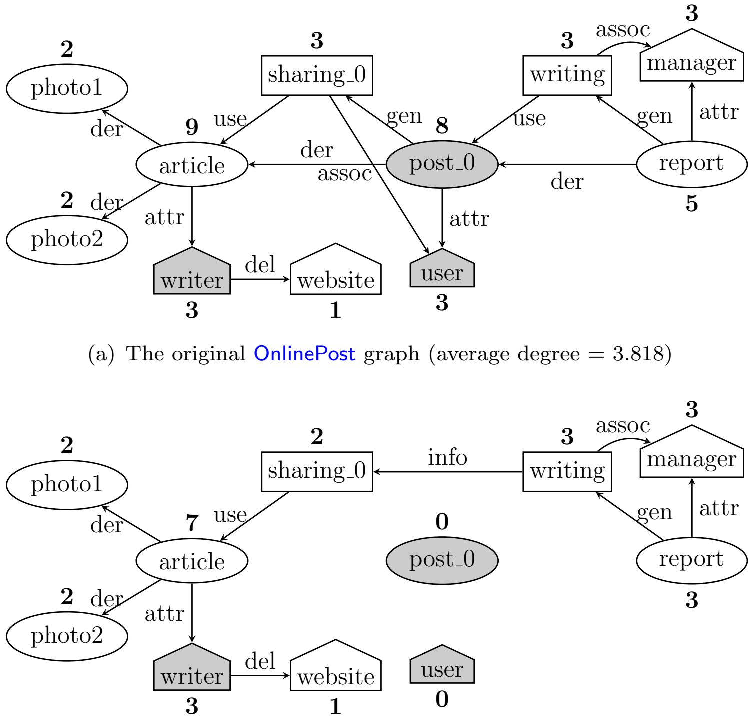
$$degree(n) = 2 * (der(n) + attr(n)) + info(n) + del(n) + use(n) + gen(n) + assoc(n)$$

where $der(n), attr(n), info(n), del(n), ...$ represent the number of those edges connected to the node n and the average degree of graph G is

$$\frac{1}{|N_G|} \sum_{i=1}^{|N_G|} degree(n_i \in N_G)$$

Regarding the OnlinePost graph as shown in Figure 8.3(a), the degree of the entity *report* is

$$\begin{aligned} degree(report) &= 2 * (der(report) + attr(report)) + use(report) + \\ &\quad gen(report) + assoc(report) \\ &= 2 * 2 + 0 + 1 + 0 \\ &= 5 \end{aligned}$$



(a) The original OnlinePost graph (average degree = 3.818)

(b) The transformed OnlinePost graph (average degree = 2.364)

Figure 8.3: Graph weights

and the average degree of the original graph is

$$\frac{5+3+3+8+3+3+9+3+1+2+2}{11} = \frac{42}{11} = 3.818$$

The average degree of transformed OnlinePost graph shown in Figure 8.3(b) is

$$\frac{3+3+3+0+0+2+7+3+1+2+2}{11} = \frac{26}{11} = 2.364$$

8.3.3 Connectivity measure

The ratio of node's degree in the transformed graph to its degree in the original graph can be is used to show the amount of information preserved regarding that node, that is node connectivity measure. The summation of all node connectivity measures divided by the size of the original graph indicates the graph's connectivity measure. Using the node utility measure defined in (Blaustein, Chapman, Seligman et al. 2011) and node degrees as described in the previous section, we can measure the amount of preserved information in the transformed graph using the following graph connectivity measure:

$$Connectivity\ Measure = \frac{1}{|N_G|}\sum_{0}^{|N_H|} sim(n, n^{'})$$

For each $n \in N_G$ and its corresponding node $n^{'} \in N_H$ where $sim(n) = \dfrac{degree(n^{'})}{degree(n)}$ and G and H are the original and transformed graphs respectively.

Using the above formula, the connectivity measure regrading the original and transformed OnlinePost graphs, as shown in Figure 8.3, can be computed as follows:

$$\begin{aligned} Connectivity\ Measure &= \frac{1}{11} * (\frac{3}{5} + \frac{3}{3} + \frac{3}{3} + \frac{0}{8} + \frac{0}{3} + \frac{2}{3} + \frac{7}{9} + \frac{3}{3} + \frac{1}{1} + \frac{2}{2} + \frac{2}{2}) \\ &= 0.731 \end{aligned}$$

8.3.4 Confidence Interval

Due to variability of degree distribution of the data sets, the result of the analysis of transformation process will not be always the same. We use a suitable confidence interval for each data set and for each plot. The confidence intervals are determined by repeating each test defined in the hypotheses below several times, computing the metrics, and then finding statistical data such as mean, median, and standard deviation of degree distribution of the results of those tests. Using these statistics, we can determine confidence interval for each graph in the plots.

8.4 Data sets

In order to evaluate the performance of PROV-GTS, data sets from three different applications are used. The applications are the **AtomicOrchid** (Jiang, Fischer, Greenhalgh et al. 2014; Fischer, Jiang and Moran 2012), the **Ride Share** (Packer and Moreau 2015), and PROV **Working Group email archive** (Moreau, Groth, Herman et al. 2011).

8.4.1 AtomicOrchid (AO) (degree = 1085, size = 2004)

AtomicOrchid is a mixed reality location-based game designed to coordinate teams and making decisions in response to natural or nuclear disasters (Jiang, Fischer, Greenhalgh et al. 2014; Fischer, Jiang and Moran 2012). In AtomicOrchid, provenance is used to facilitate coordination and decision making such that graph reduction, by removing non-relevant information such as the old massages that may distract the Headquarters (HQ), could be helpful in improving the HQs response time (Fischer, Jiang, Kerne et al. 2014). By transforming the provenance graph, we may improve the rescue operation, help the HQ in making better decisions and team coordination.

Selection criteria:

- In order to analyse the graph regarding particular participants we may need to:
  - delete some other parties (soldiers, medics, fire-fighters, . . .).
  - Remove the nodes that represent old messages, or the participants that may mislead the HQ.

8.4.2 Ride share (RS) (degree = 374, size = 1085)

The ride share application generates potential ride plans for commuters by automatically matching their ride requests to those offered by drivers (Packer and Moreau 2015).

Selection criteria:

- Specific nodes that belong to certain namespaces or share some attributes are chosen to be annotated as restricted nodes.

8.4.3 W3C PROV Working Group email archive (EA) (degree = 17875, size = 59260)

This archive represents the emails exchanged between W3C Working Group members. The archive shows how people participate and respond to other's emails and what are the most influenced emails and topics (Moreau, Groth, Herman et al. 2011).

Selection criteria:

- Eliminating some information from the emailing list using PROV-GTS may help data analysis.
- Annotating agents, which represent particular users, with all the relevant entities that represent the emails they have sent or received, as well as the sending and receiving activities.

8.5 Evaluation

Each dataset has been annotated, based on the above criteria, by selecting a specific amount of nodes, namely %5, %10, %20, %35, and %50, resulting in five annotated graphs for each dataset. These datasets are used in experimental evaluation of PROV-GTS as follows:

1. Hypothesis 1 (Connectivity): PROV-GTS removes from the provenance graph the edges that are connected to restricted nodes while preserving the relations between the remaining nodes.

   - Method: The annotated graphs will be transformed and then the connectivity measure is computed for each transformed graph H in comparison with the original graph G (Figure 8.4).

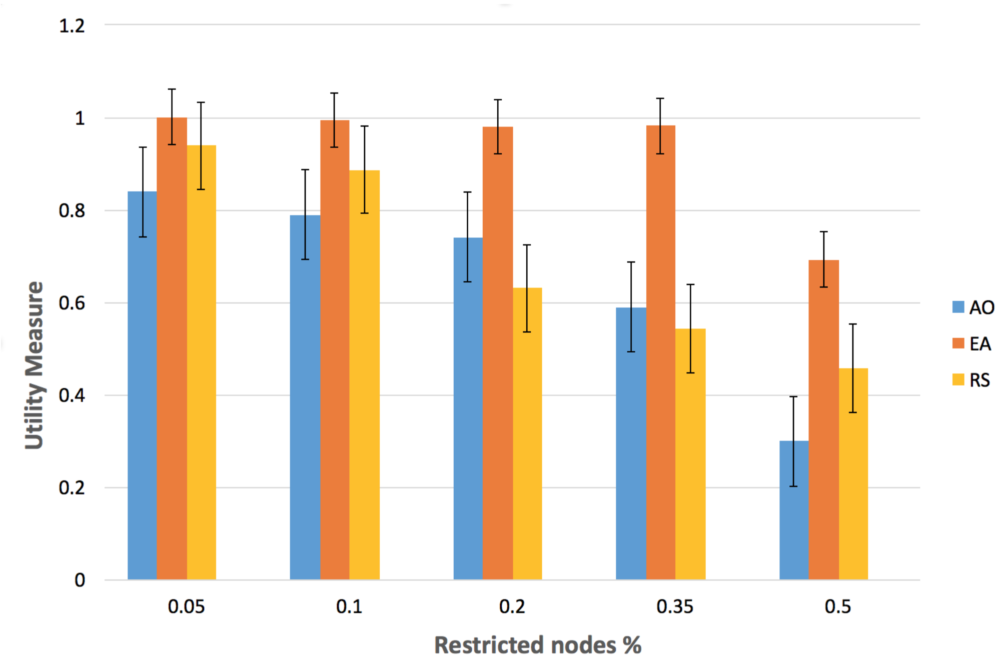


FIGURE 8.4: Node connectivity measures in comparison with the number of restricted nodes in the original graph

   - Analysis: Connectivity measure is used to show the amount of information in the original graph that is retained in the transformed graph, for example, the value 0.80 of graph connectivity measure means that 80 per cent of the provenance information of the original graph is preserved in the transformed graph. The plot shows the amount of information preserved for each data set. The results indicate that the amount of information retained in the transformed graph is, dramatically, affected by the number of restricted nodes in the original graph. What is lost is only the edges originated at the restricted nodes.

2. Hypothesis 2 (Transformation steps): The number of the transformation steps required to convert the graph is mostly affected by the number of restricted nodes in the provenance graph.

   - Method: The graphs are transformed and the number of the transformation steps are computed. The plot below shows the transformation steps for the datasets with different number of rule applications (Figure 8.5).

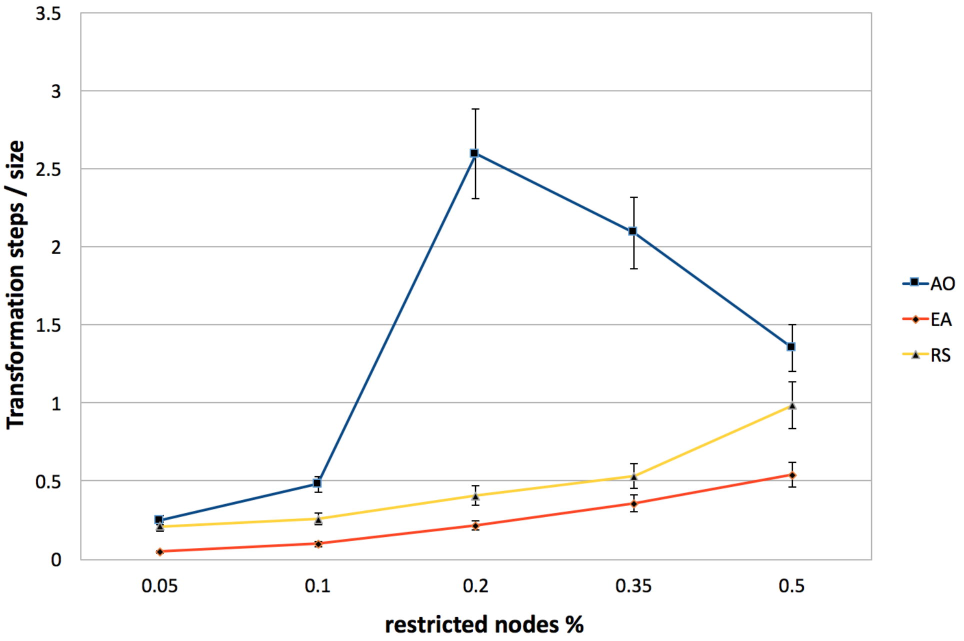


Figure 8.5: Transformation steps for different numbers of restricted nodes

- Analysis: The results show the effect of the number of restricted nodes on determining the number of transformation steps. However, increasing the number of restricted nodes does not always result in more transformation steps, since it also depend on the degree of the restricted nodes.

  Nodes with high connectivity require more transformation steps and vice versa. It is obvious from the plot that the transformation steps for the annotated graphs of AO dataset after %20 started going downhill drastically. The reason behind this is that the restricted nodes for both %35 and %50 graphs have less connectivity than the nodes of %20 graph.

3. Hypothesis 3 (Average degree): Nodes with higher degree need more deletion rule applications, since more edges must be deleted in order to isolate those nodes from the graph.

   - Method: the annotated graphs are transformed and the number of deletion steps is computed for each data set. The plot below shows the number of deletion rule applications for each data set with different average degrees (see Figure 8.6). The average degree for the original graph of each dataset is shown in parentheses next to the dataset's name to the right of the plot.

   - Analysis: The graphs with high average degrees required more deletion rule applications than creation rules.

4. Hypothesis 4 (Normalization): To what extent the graph is normal? The graphs with high normalization degree need less creation rule applications and vice versa.

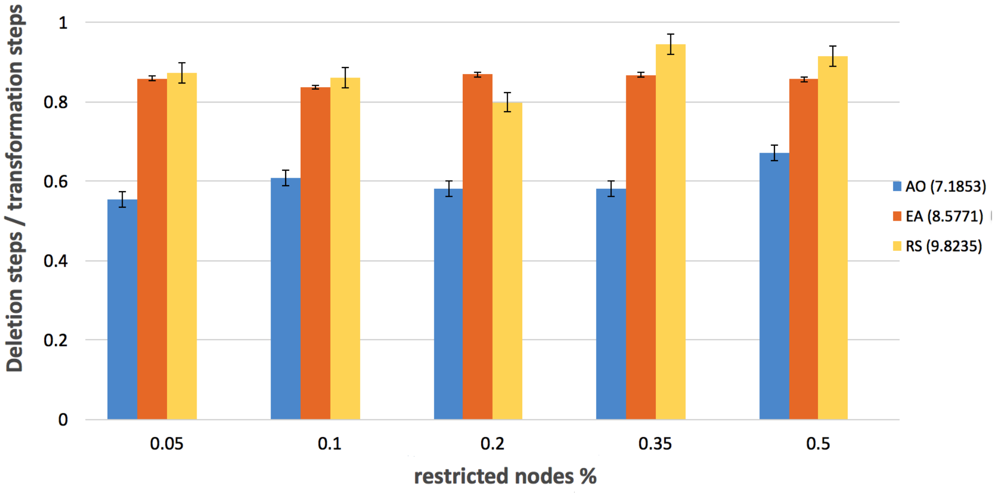


Figure 8.6: Average degree and deletion steps

- Method: The normalization degree for each original graph of the datasets are computed. The annotated graphs are transformed and the number of creation rules is counted. The number of creation steps for each annotated graph are shown in the plot of (Figure 8.7).

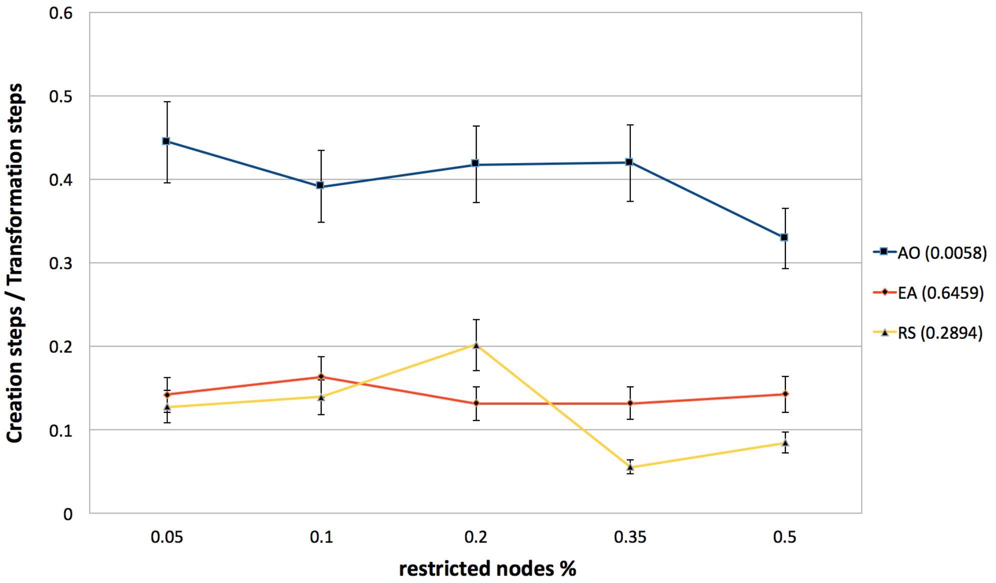


Figure 8.7: Normalization and creation steps

- Analysis: The plot in Figure 8.7 shows how the normalization degree of the original graphs determine the number of creation rules. The higher the normalization degree the less the creation steps. For example, the graph AO with the lowest normalization degree required more creation transformation steps. While the graphs EA and RS, which have higher normalization degrees, record less creation steps.

8.6 Comparison with Other Approaches

The main properties that affect connectivity are ‘no false independency’, ‘path preservation’, and ‘no excessiveness’. All the provenance graph reduction and sanitization approaches failed to preserve graph connectivity, except Language (Cadenhead, Khadilkar, Kantarcioglu et al. 2011b) and TACLP (Missier, Bryans, Gamble et al. 2015) but not without causing false dependencies and false paths.

In the other approaches, the transformation steps and the average degree are not only affected by the number of restricted nodes, but also by the redaction policies (Cadenhead, Kantarcioglu and Thuraisingham 2011; Chebotko, Chang, Lu et al. 2008) and merging techniques (Missier, Bryans, Gamble et al. 2015; Chen, Edwards, Nelson et al. 2015) used by those approaches.

Finally, none of the previous approaches used any kind of normalization by creating nodes and edges to preserve relations in provenance graphs without causing false dependencies or excessiveness (Chebotko, Lu, Chang et al. 2010; Dey and Ludäscher 2013).

8.7 Summary

In this chapter, the implementation of PROV-GTS is presented. A set of hypotheses are used to perform system performance evaluation. This performance evaluation has been conducted using three different datasets. The results show the ability of the system to preserve as much information as possible with reasonable number of creation and deletion transformation steps. Next chapter provides the conclusion and plans for further development of the proposed system in the future.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this research work, a template-based graph transformation system (PROV-GTS) has been proposed. The system consists of a set of 24 graph transformation rules derived from the PROV data model. These rules are summarized into only 8 rule templates. Each template has one or more rule instances. Rule application happens by using a template via one of the rule instances. Termination, parallelism and confluence of the system are analysed based on templates rather than rule instances. Using provenance graphs from real-world applications, the performance of PROV-GTS has been evaluated using variety of measurements such as graph connectivity, transformation steps and normalization.

The restricted nodes are processed by PROV-GTS by deleting their connected edges one by one until the nodes are isolated from the graph. If the system manages to isolate a node, then the node can be safely discarded from the graph. When the node cannot be completely isolated, since removing some edges leads to violating graph connectivity, it will be replaced by an anonymous node of the same type of the original node.

PROV-GTS is able to hide restricted nodes that expose private and confidential information from provenance graphs as long as the integrity of the graph is not violated by causing false independencies and false dependencies and breaking paths. The graph transformation rules are of two types: creation and deletion rules. Creation rules, with negative application conditions (NACs), are constructed from properties of the PROV data model such as its inference rules. This construction guarantees that no false dependencies are caused by those creation rules. In addition, a set of deletion rules are defined. Each deletion rule is capable of removing a single edge connected to a restricted node. These deletion rules incorporate a set of universal-existential graph conditions, constructed from the properties of PROV data model such that no false independencies ensue as a result of graph reduction performed by PROV-GTS.

The rule templates allow a more compact definition of PROV-GTS and make proving its properties manageable. The constructed PROV graph transformation rules that consist of similar graph topologies are grouped in abstract rules, called rule templates, defined over a PROV type graph with inheritance. This template-based PROV-GTS is formalized using a variety of concepts from category theory, such as functors, diagrams, natural transformations and pushouts. This categorical machinery provides an accurate and precise way for representing the templates, relating them to their rule instances, finding matches and applying the templates on a provenance graph via their rule instances. The rule templates make the graph transformation rules much more readable and make it easier to prove various properties of the system such as termination, which can be done on template basis.

The termination and parallelism analysis demonstrate that the system is terminating and confluent with deterministic rule applications. Inconsistency between rule applications has been avoided and parallelism of the constructed graph transformation rules has been guaranteed by attaching appropriate negative application conditions (NACs) to the rules or by expanding the left-hand sides (LHS) of the rules with positive application conditions. Analysing the rule applications shows that all the critical pairs are safe which means that the system is locally confluent. The local confluence and termination of PROV-GTS lead us to conclude that the system is also confluent.

The graph integrity and path preservation, as well as the information preservation regarding the template-based PROV-GTS, are investigated and proved to be correct. Avoiding false independencies and false dependencies guarantees that every path between the remaining nodes in the graph transformed by PROV-GTS is preserved and no false paths are created. Thus, PROV-GTS preserves as much information as possible by limiting the process of edge deletion to the edges connected to restricted nodes and preserving the graph integrity. This transformation results in reasonable connectivity preservation of provenance graphs after obscuring the private and confidential information.

Three different datasets from real applications that generate and/or consume provenance information are used to analyse the properties of PROV-GTS such as graph connectivity and the number of transformation steps, in comparison with the number of restricted nodes in the graph. To analyse those properties, measurements such as average degree, normalization degree, and connectivity measure are used. Different selection criteria are used for each dataset which result in graphs with different number of annotated nodes. The results of this evaluation show that the number of restricted nodes and their incident edges as well as the normalization degree present the main factors that determine the number of transformation steps required for converting provenance graphs. In addition, the results show the ability of the system to preserve as much information as possible while protecting the provenance confidential information.

9.2 Future Work

In this section we discuss potential future work. First, we discuss extending PROV-GTS to the full version of the PROV data model and then we look at further reducing the size of a transformed graph by merging the restricted nodes that cannot be fully isolated from the graph. Second, we show our plan for looking at other properties of the PROV data model such as preserving time constraints and processing the agent nodes. Finally, we show how PROV-GTS can be used with other models that require graph rewriting techniques, such as graph summarization and template expansion.

9.2.1 Adopting the Full Set of PROV Data Model Terms

Any future development must cover the extended terms of the PROV model that have not been involved in this research work. These extended terms classified to subclasses, superclasses, and superproperty as shown in Figure 9.1 (Lebo, Sahoo, McGuinness et al. 2013). For example, subclasses of *Agent* are *Person*, *Organization*, and *SoftwareAgent* and for *Entity* are *Collection*, *Bundle* and *Plan*. The relationship *wasInfluencedBy* is the superproperty that can represent any relationship between any two provenance components.

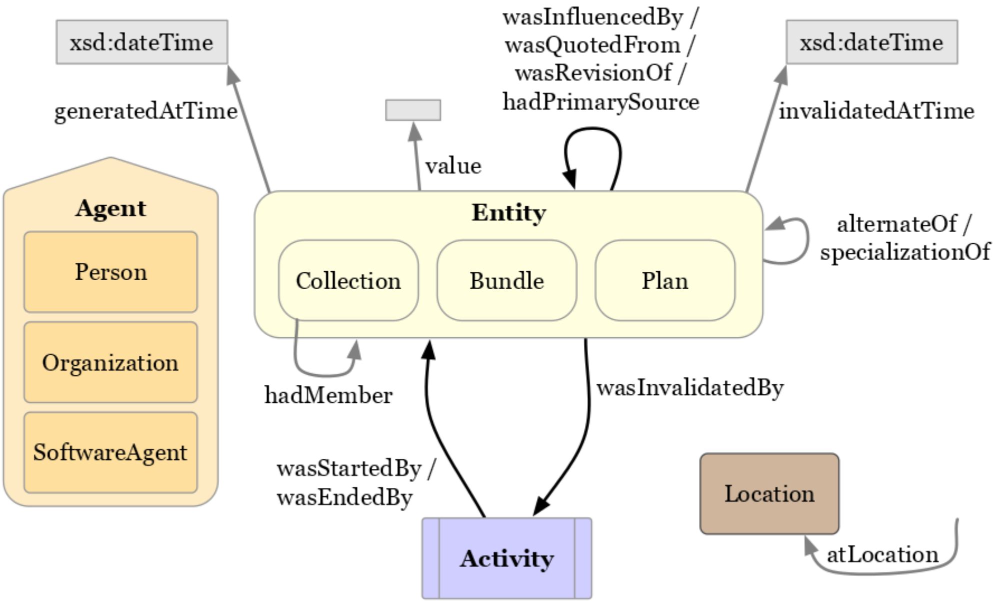
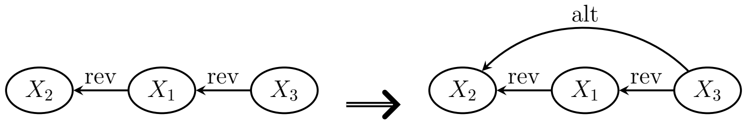
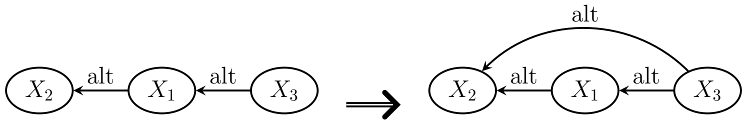
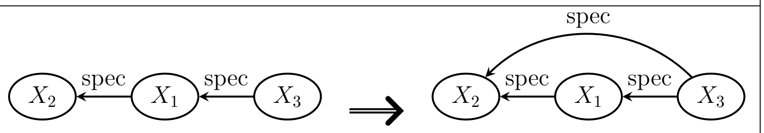


Figure 9.1: Expanded terms of PROV model (Lebo, Sahoo, McGuinness et al. 2013)

As for the core PROV model terms, there are some inference rules regarding the extended terms as well. These inference rules can be used to define new creation rules and also to construct universal-existential conditions for the deletion rules. Table 9.1 illustrates the following three inference rules regarding the relations *wasRevisionOf*, *alternateOf*, and *specializationOf*:

Table 9.1: PROV model properties for the expanded terms

Inference	**Graph Patterns** ($C_i \rightarrow E_i$)
prov5 (Cheney, Missier, Moreau et al. 2013, Inference 12, Inference 17)	 X_2 rev X_1 rev X_3 $\Longrightarrow$ alt X_2 rev X_1 rev X_3
prov6 (Cheney, Missier, Moreau et al. 2013, Inference 17)	 X_2 alt X_1 alt X_3 $\Longrightarrow$ alt X_2 alt X_1 alt X_3
prov7 (Cheney, Missier, Moreau et al. 2013, Inference 19)	 X_2 spec X_1 spec X_3 $\Longrightarrow$ spec X_2 spec X_1 spec X_3

- prov5: If there is a derivation of type revision (*rev*) between two entities then the first entity is an alternate of the second (Cheney, Missier, Moreau et al. 2013, Inference 12). Based of the fact that the *alternateOf (alt)* edges are transitive (Cheney, Missier, Moreau et al. 2013, Inference 17), we can infer an *alternateOf* (*alt*) edge between the source and target of the first and the second of two consecutive revision edges.

- prov6: The *alternateOf* relation (*alt*) is transitive (Cheney, Missier, Moreau et al. 2013, Inference 17).

- prov7: The *specializationOf* relation (*spec*) is transitive (Cheney, Missier, Moreau et al. 2013, Inference 19).

9.2.2 Merging Technique

An improvement to the system may be beneficial by merging the remaining restricted nodes that cannot be fully isolated from the graph. For instance, the restricted activities of the transformed graph of RideShare example, as shown in Figure 3.3, can be merged in one activity without affecting the graph integrity. This merging process could be useful in reducing the size of the transformed graph and further obscuring the provenance graph by making it difficult for a hacker to infer the topology of the original provenance graph from the transformed one. When merging nodes, it is crucial to preserve the integrity of provenance graphs by preserving dependencies and paths, and avoiding false paths.

9.2.3 Further Properties of PROV data model

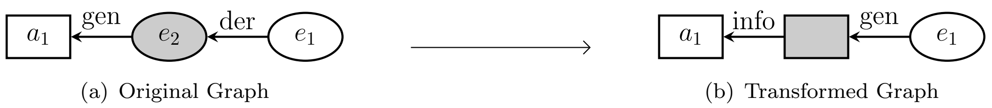
In addition to the extended terms of PROV data model, any other properties of PROV model that have not been covered by PROV-GTS should be taken into account in any upcoming development. In the following we present some of these properties and terms briefly:

9.2.3.1 Ordering constraints in PROV data model

PROV-GTS preserves the relations and paths connecting the nodes in the transformed graph. However, in some cases deleting nodes may result in violating the time sequence of some events in provenance graph such as the order in which generation of entities or starting of activities occurred (Kwasnikowska, Moreau and Bussche 2015). For example, the two relations (derivation and generation) in Figure 9.2(a) imply that the entity e_1 has been generated after the activity a_1 started according to the PROV constraint *derivation-generation-generation-ordering* (Cheney, Missier, Moreau et al. 2013, Constraint 42) below:

```
IF      wasDerivedFrom ( _d ; e2 , e1 , _a, _g, _u, attr)    and
        wasGeneratedBy (gen1 ; e1 , _a1 , _t1 , attrs1 )    and
        wasGeneratedBy (gen2 ; e2 , _a2 , _t2 , attrs2 )
THEN    gen1 strictly precedes gen2
```

However, this order of generation and start events cannot be confirmed from the transformed graph in Figure 9.2(b) since the generation of e_1 could be before the start of a_1. It is crucial for the transformed graph to satisfy the same ordering constraints that are satisfied by the original graph.



(a) Original Graph

(b) Transformed Graph

FIGURE 9.2: The issue of ordering constraints

9.2.3.2 Dual Behaviour of Agents

Agents in the PROV model, in spite being an agent, could behave as an activity or an entity as well (Moreau, Groth, Cheney et al. 2015), so it should be handled accordingly. For example an agent may represent an entity derived from another entity, attributed to another agent, and used or generated by an activity. Investigating this dual behaviour of agents and incorporating it in our system is important to integrate all PROV aspects and taking into account all its properties.

9.2.4 Graph abstraction using PROV-GTS

PROV-GTS is a general graph transformation system that was initially designed to provide effective means to obscure private provenance information while preserving graph integrity. Since PROV-GTS is a domain-agnostic algebraic graph transformation system that is constructed based on properties of PROV data model, it can also be used with other models that require provenance graph rewriting such as summarization and template expansion.

9.2.4.1 Summarization

Provenance is useful for the applications that collect and manipulate large amounts of data. However, maintaining and explaining large and complex provenance graphs are impracticable for such applications (Ainy, Davidson, Deutch et al. 2014; Ainy, Bourhis, Davidson et al. 2015). Provenance graph aggregation and summarization are helpful in understanding and viewing large amounts of provenance data.

A summarization approach presented in (Moreau 2014) which generates provenance summaries by converting provenance paths up to some length k to attributes referred to as provenance types and then grouping the nodes that have the same provenance types. Summaries have set of nodes and edges based on the PROV data model. The same construction methodology of PROV-GTS can be used to build a set of graph transformation rules capable of summarising provenance graphs.

9.2.4.2 Template expansion

Provenance graphs can be generated according to pre-defined templates[1]. Templates are provenance graphs that follow PROV data model with variables in any position in the graph. For example, variables could represent node identifiers, attribute names or attribute values. Templates represent frequent patterns and variables act as placeholders which can be instantiated to create an instance of that template through a set of bindings (Michaelides, Huynh and Moreau 2014; Moreau, Batlajery, Huynh et al. 2016). Multiple values are allowed for variables which leads to template expansion. Templates are used in some systems such as the online repository system PICASO (Huynh, Michaelides and Moreau 2016).

Template expansion via variables and their bindings represents some sort of graph rewriting. Which means that the PROV-GTS is suitable in this case to provide a formal graph transformation system for template expansion.

[1] We must not confuse between this template and our proposed rule template, they represent two different aspects, former is a template that is based on variables and bindings while latter is constructed from similar-shape rules using abstract nodes.

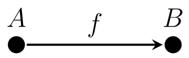
Appendix A

Category Theory

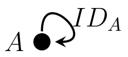
In PROV-GTS, the similar-shaped transformation rules are grouped in rule templates which improve readability and facilitate the proofs of many properties such as termination. In this section, we describe a variety of concepts from category theory which are used in the formal description of the template-based PROV-GTS, as presented in Chapter 5. A category is a mathematical structure consisting of a collection of objects and composable arrows (morphisms) that describes the abstract structure of algebra formulas (Awodey 2010). Category theory is first defined by Samuel Eilenberg and Saunders Mac Lane in 1945 (Eilenberg and MacLane 1945).

A category C consists of a class of objects C_0 and a class of morphisms C_1 (also called arrows or maps). Each morphism $f : A \to B$ in C_1 has a domain object A and a codomain object B representing the source and target of the morphism respectively. There is an identity morphism ID_A for each object A which has A as its domain (dom) and codomain (cod). Given a pair of morphisms $f : A \to B$ and $g : B \to C$ such that $cod(f)$=$dom(g)$, the pair f and g can be composed as $g \circ f$: $A \to C$ with $dom(f)$ and $cod(g)$ as its domain and codomain respectively. In addition, each category obeys the following two rules:

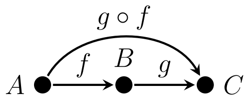
1. Identity composition: for each morphism $f : A \to B$ there is $f \circ ID_A = ID_B \circ f$
2. Associativity: for each set of morphisms $f : A \to B$, $g : B \to C$, and $h : C \to D$ there is $(h \circ g) \circ f = h \circ (g \circ f)$.



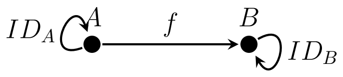
morphism



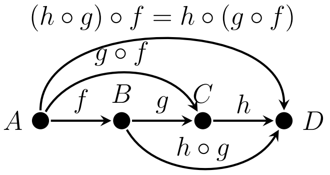
identity



composition



identity composition



associativity

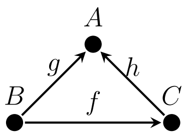
For example, category *Graph* has graphs as nodes and graph homomorphisms as arrows, while *Set* is the category of sets as objects and the functions between sets as morphisms.

A.1 Dual Category

Each category consists of a set of objects, a set of morphisms, and four operations, namely, domain, codomain, identity, and composition. By reversing the domain and codomains and arrow compositions of category $\mathcal{C}$ we end up with the dual category $\mathcal{C}^{-1}$. The identities are the same in $\mathcal{C}$ and $\mathcal{C}^{-1}$.

A.2 Slice categories

A slice category is $\mathcal{C}/A$ (a special case of comma category founded by F. W. Lawvere in 1963 (Lawvere 1963)), when $\mathcal{C}$ is a category and A is an object of $\mathcal{C}$, such that the objects of $\mathcal{C}/A$ are the set of arrows from any object in $\mathcal{C}$ to A and the set of arrows consists of $f : B \to C$ from $g : B \to A$ to $h : C \to A$ as shown in the following diagram.



Slice category in algebraic graph transformation is used to define a category of graphs (C) determined by a specific type graph (A).

A.3 Terminal and initial objects

An object T of a category $\mathcal{C}$ is called terminal if for each object A in $\mathcal{C}$ there is exactly one arrow $A \rightarrow T$. An initial object of category $\mathcal{C}$ is the dual of terminal object, i.e. an object of a category that has a unique arrow to each object in the category.

A.4 Properties of objects and arrows in categories

- **Monomorphism (*mono*):**
  An arrow $f : A \rightarrow B$ in category $\mathcal{C}$ is *mono* (monomorphic) if for any object C and any pair of arrows $g, h : C \rightarrow A$, $f \circ g = f \circ h$ implies $g = h$. In *Set*, a morphism is *mono* iff it is an injective function (one-to-one).

$$\begin{tikzcd} C \arrow[r, bend left, "g"] \arrow[r, bend right, "h"'] & A \arrow[r, "f"] & B \end{tikzcd}$$

- **Epimorphism (*epi*):**
  An arrow $f : A \rightarrow B$ in category $\mathcal{C}$ is *epi* (epimorphic) if for any object C and any pair of arrows $g, h : B \rightarrow C$, $g \circ f = h \circ f$ implies $g = h$. In *Set*, f is *epi* iff f is a surjective function (onto).

$$\begin{tikzcd} A \arrow[r, "f"] & B \arrow[r, bend left, "g"] \arrow[r, bend right, "h"'] & C \end{tikzcd}$$

- **Isomorphism (*iso*):**
  An arrow $f : A \rightarrow B$ in category $\mathcal{C}$ is *iso* (isomorphic) if there exist the inverse arrow $f^{-1} : B \rightarrow A$ in $\mathcal{C}$ such that $f^{-1} \circ f = ID_A$ and $f \circ f^{-1} = ID_B$. In *Set*, every morphism which is both *epi* and *mono* is an isomorphic (bijection).

$$\begin{tikzcd} A \arrow[loop left, "ID_A"] \arrow[r, bend left, "f"] & B \arrow[l, bend left, "f^{-1}"] \arrow[loop right, "ID_B"] \end{tikzcd}$$

A.5 Pullback

Given two arrows f and g in any category $\mathcal{C}$ with $cod(f) = cod(g)$ then the pullback of f and g is given by the arrows $f^{'}$ and $g^{'}$ such that the following diagram commutes:

$$\begin{tikzcd}
P \arrow[r, "f^{'}"] \arrow[d, "g^{'}"'] & A \arrow[d, "g"] \\
B \arrow[r, "f"] & C
\end{tikzcd}$$

This commutative diagram is universal, that is for any two arrows $h_1 : Z \to A$ and $h_2 : Z \to B$ with $f \circ h_2 = g \circ h_1$ there exists a unique arrow $u : Z \to P$ with $h_1 = f^{'} \circ u$ and $h_2 = g^{'} \circ u$.

$$\begin{tikzcd}
Z \arrow[dr, dotted, "u"] \arrow[drr, "h_1"] \arrow[ddr, "h_2"'] & & \\
& P \arrow[r, "f^{'}"] \arrow[d, "g^{'}"'] & A \arrow[d, "g"] \\
& B \arrow[r, "f"] & C
\end{tikzcd}$$

A.6 Pushout

The dual construction of pullback is pushout. The pushout of two arrows $f : A \to B$ and $g : A \to C$ in any category $\mathcal{C}$ is given by the arrows $g^{'} : B \to P$ and $f^{'} : C \to P$ such that following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "g^{'}"] \\
C \arrow[r, "f^{'}"] & P
\end{tikzcd}$$

This commutative diagram is universal, that is for any two arrows $h_1 : C \to Z$ and $h_2 : B \to Z$ with $h_2 \circ f = h_1 \circ g$ there exists a unique arrow $u : P \to Z$ with $h_1 = u \circ f^{'}$ and $h_2 = u \circ g^{'}$.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "g^{'}"] \arrow[ddr, "h_2"] & \\
C \arrow[r, "f^{'}"] \arrow[drr, "h_1"'] & P \arrow[dr, dotted, "u"] & \\
& & Z
\end{tikzcd}$$

A.7 Functors and Constant Functors

A functor $F : \mathcal{C} \to \mathcal{D}$ between the categories $\mathcal{C}$ and $\mathcal{D}$ consists of two operations $F_0 : C_0 \to D_0$ and $F_1 : C_1 \to D_1$, such that for each $f : A \to B$ in C_1 there is $F_1(f) : F_0(A) \to F_0(B)$ in D_1, for each composition $g{\circ}f$ there is $F_1(g{\circ}f) = F_1(f){\circ}F_1(g)$ and $F_1(ID_A) = ID_{F_0(A)}$ for each $A \in C_0$. A constant functor $F : \mathcal{C} \to \mathcal{D}$ sends each object $C \in \mathcal{C}_0$ to an object $D \in \mathcal{D}_0$ and each morphism $f \in \mathcal{C}_1$ to the identity morphism $ID_D \in \mathcal{D}_1$.

The category Cat has categories as objects and functors as morphisms with functor compositions $G \circ F : \mathcal{C} \to \mathcal{E}$ for the two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{E}$ and the identity functor $ID_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$.

A.8 Free Category and Diagrams

Directed graphs, where each edge has a source and target node, are just like categories except that they have no identities and composition. For any given directed graph G there is a free category $F(G)$ with the nodes of G as objects and the paths in G as arrows. The identity of each object A is the empty path ID_A and composition is the concatenation of two paths:

$$(h_1, h_2, \ldots, h_n) \circ (h_{n+1}, h_{n+2}, \ldots, h_m) = (h_1, h_2, \ldots, h_m)$$

A diagram in category $\mathcal{C}$ of shape J is a functor $D : F(J) \to \mathcal{C}$ where $F(J)$ is the free category on the graph J.

A.9 Natural Transformation

As functors are morphisms between categories, natural transformation are morphisms between functors respecting the internal structure of the categories, i.e. compositions of morphisms. For two categories $\mathcal{C}$ and $\mathcal{D}$ the functors $F, G : \mathcal{C} \to \mathcal{D}$ and the natural transformation between them $\psi : F \Rightarrow G$ form a category denotes as $\mathcal{C}^{\mathcal{D}}$ or $[\mathcal{C}, \mathcal{D}]$.

$$\begin{tikzcd}
\mathcal{C} \arrow[r, bend left=50, "F"{name=U}] \arrow[r, bend right=50, "G"'{name=D}] & \mathcal{D} \arrow[Rightarrow, from=U, to=D, "\psi"']
\end{tikzcd}$$

Natural transformation associate to every object X of $\mathcal{C}$ a morphism $\psi_X : F(X) \to G(X)$ between objects of $\mathcal{D}$ called the component of ψ at X such that for every morphism $f : X \to Y$ in $\mathcal{C}$ the following diagram commutes:

$$\begin{tikzcd}
F(X) \arrow[r, "F(f)"] \arrow[d, "\psi_X"'] \arrow[dr, phantom, "="] & F(Y) \arrow[d, "\psi_Y"] \\
G(X) \arrow[r, "G(f)"'] & G(Y)
\end{tikzcd}$$

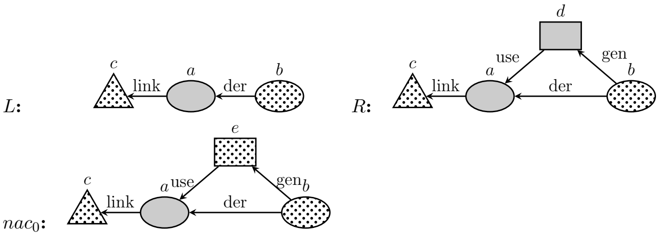
$$\psi_Y \circ F(f) = G(f) \circ \psi_X$$

Appendix B

Set of Rules

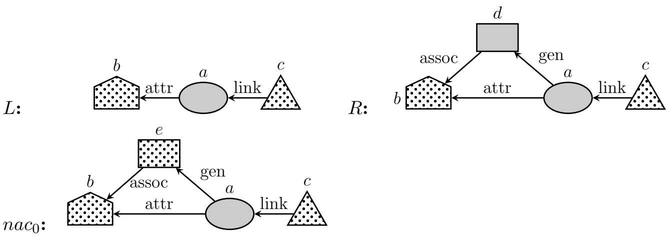
In this Appendix, the complete set of graph transformation rules of PROV-GTS is presented. The list starts with the four creation rules that belong to template1 and template2 followed by the set of deletion rules that are instances of templates3-8. For each rule, we have briefly provided a description for the (L, R) and each one of the conditions of that rule.

(1-1)*crtActOutDer*:



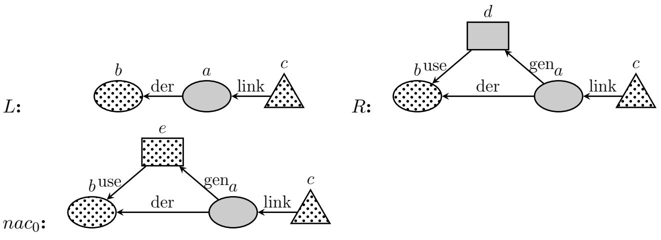
(L, R):	creates a restricted activity for incoming *der* edges if the target entity is restricted and has no outgoing edges
nac_0:	ensures that the *der* has no activity.

(1-2)*crtActAtt*:



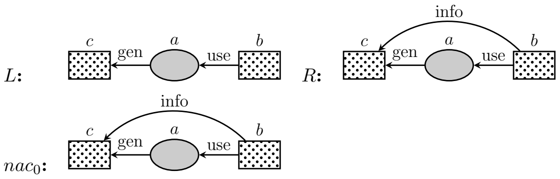
(L, R):	creates a restricted activity for *attr* edges if the entity is restricted and has an incoming edges
nac_0:	ensures that the *attr* edge has no activity.

(1-3)*crtActInDer*:



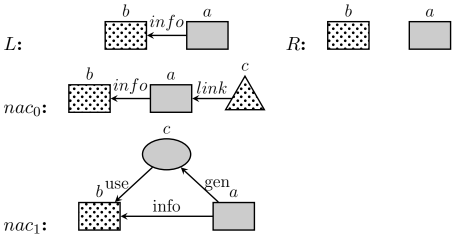
(L, R):	creates a restricted activity for outgoing *der* edges if the source entity is restricted and has an incoming edge.
nac_0:	ensures that the *der* edge has no activity.

2-4 ***crtInfUseGen*:**



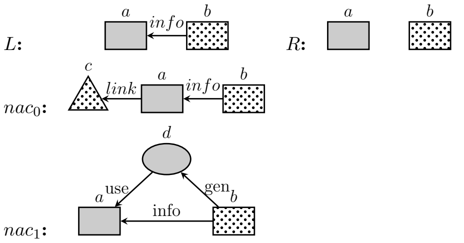
(L, R):	creates an $info$ edge for $use-gen$ pattern if the entity is restricted.
nac_0:	ensures that there is no $info$ edge.

3-5 ***delInfNoIn*:**



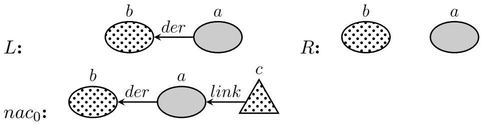
(L, R):	deletes an outgoing $info$ edge if the informant activity is restricted
nac_0:	ensures that the informant activity has no incoming edges.
nac_1:	prevents deleting $info$ edge if there is a restricted entity generated by the informant activity and used by the informed one.

3-6 ***delInfNoOut*:**



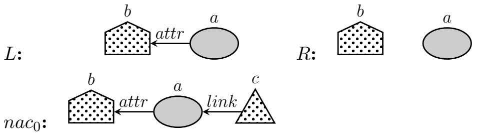
(L, R):	deletes an outgoing $info$ edge if the informed activity is restricted
nac_0:	ensures that the informed activity has no outgoing edges.
nac_1:	prevents deleting the $info$ edge if there is a restricted entity generated by the informant activity and used by the informed one.

4-7 ***delDerNoIn*:**



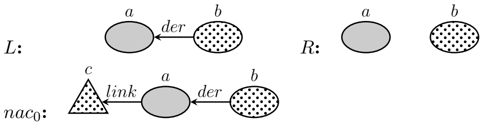
(L, R):	deletes an *der* edge if the derived entity is restricted.
nac_0:	ensures that the derived entity has no incoming edges

4-8 ***delAttNoIn*:**



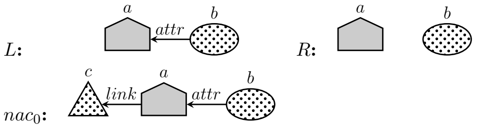
(L, R):	deletes an *attr* edge if the entity is restricted.
nac_0:	ensures that the entity has no incoming edges

4-9 ***delDerNoOut*:**



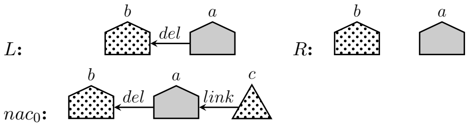
(L, R):	deletes a *der* edge if the target entity is restricted.
nac_0:	ensures that the restricted entity has no outgoing edges.

4-10 ***delAttNoOut*:**



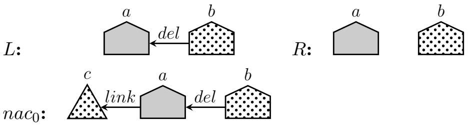
(L, R):	deletes an *attr* edge if the agent is restricted.
nac_0:	ensures that the restricted agent has no outgoing edges

4-11 ***delDelNoIn*:**



(L, R):	deletes a *del* edge if the source agent is restricted.
nac_0:	ensures that the the restricted agent has no incoming edges

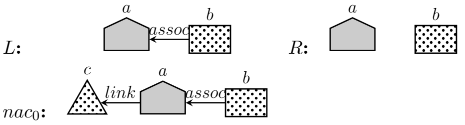
(4-12) ***delDelNoOut*:**



(L, R): deletes a *del* edge if the target agent is restricted.

nac_0: ensures that the the restricted agent has no outgoing edges

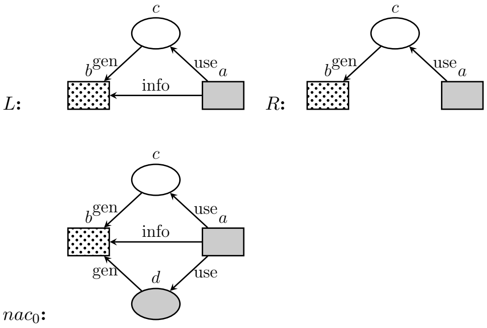
(4-13) ***delAssocNoOut*:**



(L, R): deletes an *assoc* edge if the agent is restricted.

nac_0: ensures that the restricted agent has no outgoing edges.

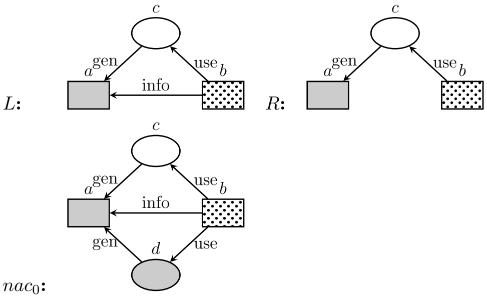
(5-14) ***delOutInf*:**



(L, R): deletes an *info* edge when the informant activity is restricted and there is an entity generated by informed activity and used by the restricted informant one. The entity must be *plain* so that any conflict is avoided in case if the entity was restricted.

nac_0: prevents deleting the *info* edge if there is a *restricted* entity between the two activities.

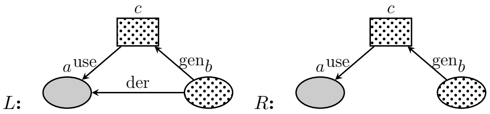
(5-15) ***delInInf*:**



(L, R): deletes an *info* edge when the informed activity is restricted and there is an entity that was generated by restricted informed activity and used by the informant one. The entity must be *plain* so that any conflict is avoided in case if the entity was restricted.

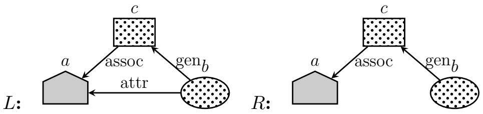
nac_0: prevents deleting the *info* edge if there is a *restricted* entity between the two activities.

(6-16) ***delInDer*:**



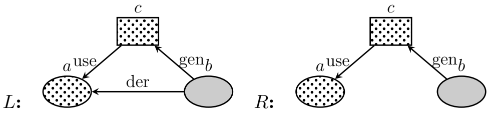
(L, R): deletes a *der* edge when the target entity is restricted and there is an activity generated the source entity by using the restricted one.

(6-17) ***delInAtt*:**



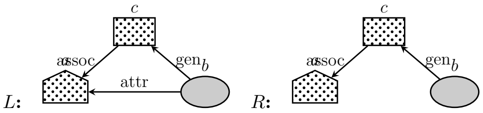
(L, R): deletes a *der* edge when the target entity is restricted and there is an activity generated the source entity by using the restricted one.

(6-18) ***delOutDer*:**



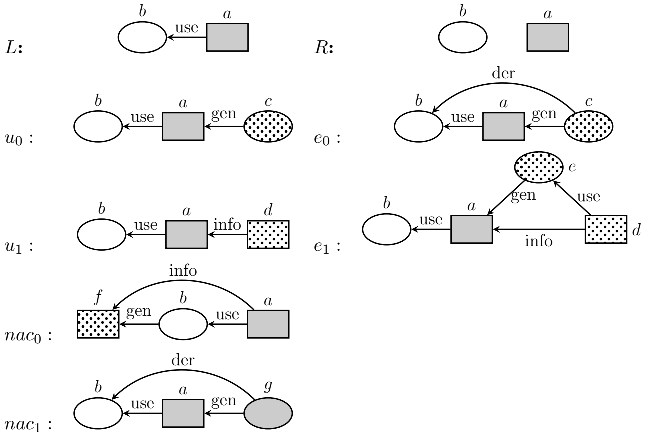
(L, R): deletes a *der* edge when the source entity is restricted and there is an activity generated the restricted entity by using the target one.

(6-19) ***delOutAtt*:**



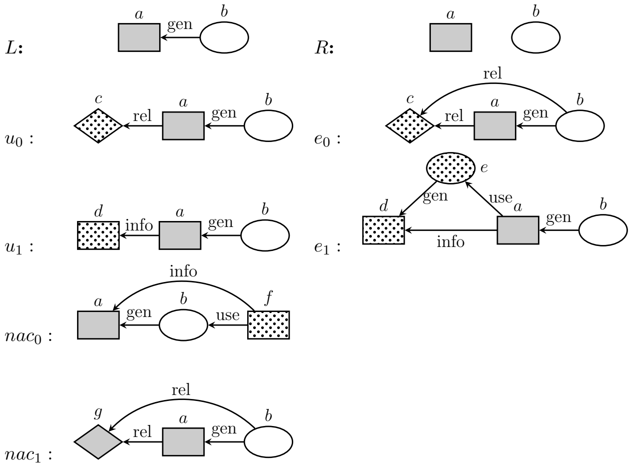
(L, R): deletes an *attr* edge when the source entity is restricted and there is an activity generated the restricted entity by associated to the agent.

(7-20) ***delUseAct*:**



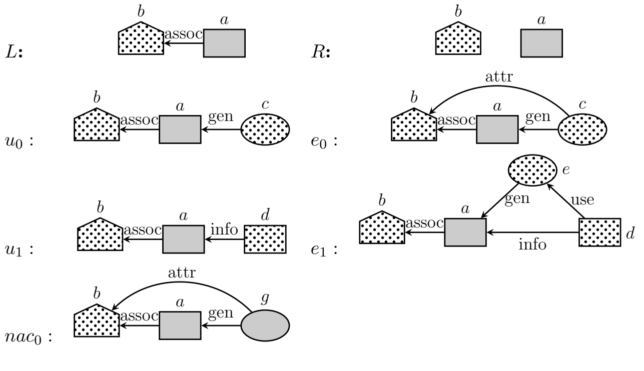
(L, R):	deletes a *use* edge when the activity is restricted. the used entity is plain so that any inconsistency with the deletion rules regarding the restricted entities is avoided.
(u_0, e_0):	for each incoming *gen* to the restricted activity there exists a *der* edge to the plain node.
(u_1, e_1):	for each incoming $info$ there exists an entity.
nac_0:	prevents deleting the *use* edge if it is part of an $info$ edge.
nac_1:	prevents deleting the *use* edge if it is part of a *der* edge its source entity is restricted.

(7-21) ***delGenAct*:**



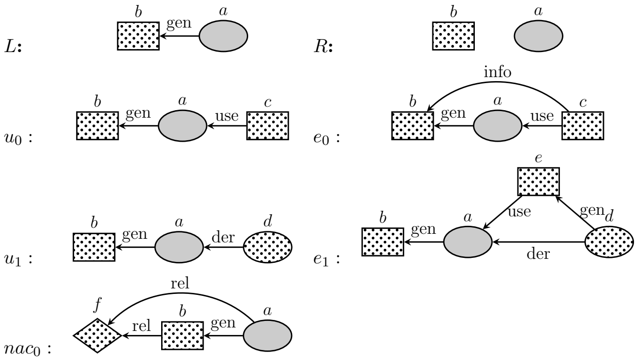
(L, R):	deletes a *gen* edge when the activity is restricted. The generated entity is plain so that any inconsistency with the deletion rules regarding the restricted entities is avoided.
(u_0, e_0):	for each outgoing *use* or *assoc* from the restricted activity there exists a *der* or an *attr* edge respectively from the plain entity.
(u_1, e_1):	for each outgoing $info$ there exists an entity.
nac_0:	prevents deleting the *gen* edge if it is part of an $info$ edge.
nac_1:	prevents deleting the *gen* edge if it is part of a *der* or an *attr* edge its target node is restricted.

(8-22) ***delAssocAct*:**



(L, R):	delete an *assoc* edge connected to a restricted activity.
(u_0, e_0):	ensures that for each incoming *gen* there exists an *attr*.
(u_1, e_1):	ensures that for each incoming $info$ there exists an entity.
nac_0:	prevents deleting the *assoc* edge if it is part of an *attr* edge its source entity is restricted.

(8-23) ***delGenEnt*:**



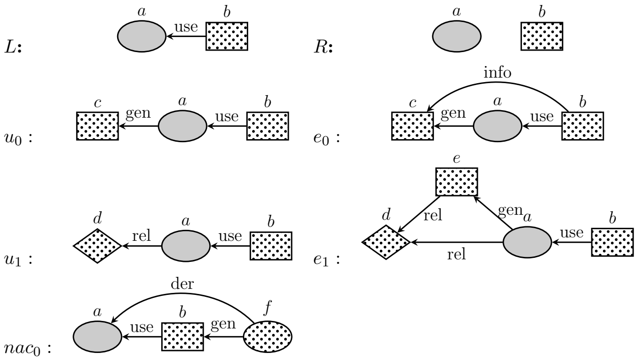
(L, R): deletes a *gen* edge when the entity is restricted.

(u_0, e_0): for each incoming *use* there exists an *info* edge.

(u_1, e_1): for each incoming *der* there exists an activity.

nac_0: prevents deleting the *gen* edge if it is part of a *der* edge or *attr* edge.

(8-24) ***delUseEnt*:**



(L, R): deletes a *use* edge when the entity is restricted.

(u_0, e_0): for each outgoing *gen* there exists an *info* edge.

(u_1, e_1): for each outgoing *der* or *attr* there exists an activity.

nac_0: prevents deleting the *gen* edge if it is part of a *der* edge.

Appendix C

Proof of Termination

Lemma C.1 (Termination of Layered Graph Grammars with Deletion)**.** *Each layered graph grammar with deletion terminates.*

Proof. ***Step 0:***

Let $c_0 = card\{x \in G_0 | dl(type(x)) = 0\}$.

By deletion layer Conditions 1 and 3 the application of a rule template τ to G_0 with $tl(\tau) = 0$ deletes at least one item $x \in G_0$ with type $t = type(x)$ and $dl(t) = 0$. Moreover by deletion layer Condition 4 each of the rule templates τ can only create items x with $type(x) = t$, where $cl(t) > 0$. This means by using deletion layer Condition 2 that only items x with $type(x) = t$ and $dl(t) \geq cl(t) > 0$ can be created. Hence, at most c_0 applications of rule templates $\tau \in TEMP_0$ are possible in layer 0 leading to $G_0 \Rightarrow *G_1$ via $TEMP_0$.

Step k:

Given graph G_k as result of step $(k-1)$ for $1 \leq k \leq k_0$ then define $c_k = card\{x \in G_k | dl(type(x)) \leq k\}$. Using rule templates τ with $tl(\tau) = k$ each $\tau \in TEMP_k$ deletes at least one item $x \in G_k$ with $dl(type(x)) \leq k$ by deletion layer Conditions 1 and 3 and creates at most items x with $cl(type(x)) > k$ by deletion layer condition 4 which implies $dl(type(x)) \geq cl(type(x)) > k$ by deletion layer Condition 2. Hence, at most c_k applications of rule templates $\tau \in TEMP_k$ are possible in layer k leading to $G_k \Rightarrow *G_{k+1}$ via $TEMP_k$.

After step n we have at most $c = \sum_{k=0}^{k_0} c_k$ applications of rule templates $\tau \in TEMP$ leading to $G_0 \Rightarrow *G_{k_0+1}$, which implies termination.

□

For proving termination of creation layers, essential matches need to be defined. An essential match of a match $m_1 : L \longrightarrow H_1$ is $m_0 : L \longrightarrow G_0$ for a transformation $G_0 \Rightarrow *H_1$ with $G_0 \subseteq H_1$. In the following definition, we consider the set RUL of the basic concrete rules of the form $r : L \longrightarrow R$ which are instances of the rule templates in $TEMP$ and RUL_k is the set of rule instances of layer k.

Definition C.2 (Essential match)**.** Given a creation graph grammar with injective matches, a creation rule $r : L \longrightarrow R$ and a match i$m : L \longrightarrow G$ are injective morphisms leading to a transformation step $G \xRightarrow{(r,m)} H$ defined by the pushout (1) of r and m, where $d : G \longrightarrow H$ is called tracking morphism of $G \xRightarrow{(r,m)} H$ via (r, m).

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & R \arrow[d, "\acute{m}"] \\
G \arrow[r, "d"'] & H
\end{tikzcd}$$

Since r and m are injective morphisms, pushout properties (1) imply that also d and $\acute{m}$ are injective. Given a transformation $G_0 \Rightarrow *H_1$ i.e. a sequence of transformation steps with induced injective tracking morphism $d_1 : G_0 \longrightarrow H_1$ a match $m_1 : L \longrightarrow H_1$ of L in H_1 has an essential match $m_0 : L \longrightarrow G_0$ of L in G_0 if we have $d_1 \circ m_0 = m_1$. Note, that there is at most one essential match m_0 for m_1, because d_1 is injective.

□

Lemma C.3 (Rule application with essential matches)**.** *In each transformation sequence starting from G_0 of a creation layered graph grammar with injective matches, each rule $r : L \longrightarrow R$ can be applied at most once with the same essential match $m_0 : L \longrightarrow G_0$ and $m_0 \models NAC$.*

Proof. Assume that in $G_0 \Rightarrow *H_1$ rule r has been applied with the same 'essential match' m_0 already. This means we can decompose $G_0 \Rightarrow *H_1$ into $G_0 \Rightarrow *G \Rightarrow H \Rightarrow *H_1$ with pushout(1) and injective morphisms $G_0 \xrightarrow{g} *G \xrightarrow{d} H \xrightarrow{h_1} *H_1$ satisfying $d_1 = h_1 \circ d \circ g$ and $d_1 \circ m_0 = m_1$ in Fig. **??**.

$$\begin{tikzcd}
 & L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dl, "m_0"'] \arrow[dr, phantom, "(1)"] & R \arrow[d, "\acute{m}"] & L \arrow[r, "r"] \arrow[d, "m_1"'] \arrow[dr, "n"] \arrow[dlll, bend right, "m_0"'] & R \\
G_0 \arrow[r, "g"'] \arrow[rrr, bend right, "d_1"'] & G \arrow[r, "d"'] & H \arrow[r, "h_1"'] & H_1 & N \arrow[l, dashed, "q_1"] \arrow[u, "\acute{n}"']
\end{tikzcd}$$

In order to prove the lemma now it is sufficient to show that $m_1 : L \longrightarrow H_1$ does not satisfy the NAC of r, i.e. $m_1 \not\models NAC$, where the NAC is given by an injective morphism $n : L \longrightarrow N$ with $n : N \longrightarrow R$ injective satisfying $n \circ n = r$ by condition 2. In fact we are able to construct an injective $q_1 : N \longrightarrow H_1$ with $q_1 \circ n = m_1$.

Let $q1 = h1 \circ \acute{m} \circ \acute{n}$, then q_1 is injective because n, $\acute{m}$ and h_1 are injective and injectivity of $\acute{m}$ follows from injectivity of match m. Moreover, we have: $q_1 \circ n = h1 \circ \acute{m} \circ n \circ n = h_1 \circ \acute{m} \circ r = h1 \circ d \circ m = h1 \circ d \circ g \circ m_0 = d_1 \circ m_0 = m_1$ This completes the proof of Lemma C.3.

□

Lemma C.4 (Termination of Creation Layered Graph Grammar)**.** *Each creation layered graph grammar with injective matches terminates.*

Proof. ***Step 0:***

Given the start Graph G_0 we count for each $r \in RUL_0$ with $r : L \longrightarrow R$ and NAC the number of possible matches $m : L \longrightarrow G_0$ with $m \models NAC$

$c_r^0 = card\{m_0 | m_0 : L \longrightarrow G_0\}$ match with $m_0 \models NAC\}$

We will show the following:

The application of rules $r \in RUL_0$ creates by creation layer Condition 4 only new items x with $cl(type(x)) > tl(r) = 0$, while each item $x \in L$ for any rule $r \in RUL_0$ has $cl(type(x)) \leq tl(r) = 0$ by creation layer condition 3. This means that for each transformation sequence $G_0 \Rightarrow *H_1$ via RUL_0 with injective matches and injective morphism $d_1 : G_0 \longrightarrow H_1$ (induced from $G_0 \Rightarrow *H_1$ by creation layer Condition 1) each match $m_1 : L \longrightarrow H_1$ of some $r \in RUL_0$ must have an 'essential match' $m_0 : L \longrightarrow G_0$ with $d_1 \circ m_0 = m_1$.

From Lemma C.3 we conclude that in step 0 we have at most

$$c_0 = \sum_{r \in RUL_0} c_r^0$$

application of rules $r \in RUL_0$ leading to $G_0 \Rightarrow *G_1$ via RUL_0.

Step k:

Given graph G_k as result of step $(k-1)$ for $1 \leq k \leq k_0$ then define for each $r \in RUL_k$ with $r : L \longrightarrow R$ and NAC

$c_r^k = card\{m_k | m_k : L \longrightarrow G_k$ match with $m_k \models NAC\}$

Similar to step 0 each $r \in RUL_k$ creates only new items x with $cl(type(x)) > tl(r) = k$, while each item $x \in L$ has $cl(type(x)) \leq tl(r) = k$. Now we can apply Lemma C.3 for G_k, RUL_k, and m_k instead of G_0, RUL_k, and m_0 and can conclude to have at most

$$c_k = \sum_{r \in RUL_k} c_r^k$$

application of rules leading to $G_k \Rightarrow *G_{k+1}$, via RUL_k.

After step n we have at most

$$c = \sum_{k=0}^{k_0} c^k$$

applications of rules $r \in RUL$ leading to $G_0 \Rightarrow *G_{k_0+1}$, which implies termination.

□

Theorem C.5 (Termination of Layered Provenance Graph Grammar)**.** *Each layered graph grammar with injective matches terminates.*

Proof. Starting with $k = 0$ we can apply for each deletion layer the deletion layer conditions (see Lemma C.1) and for each creation layer the creation layer conditions (see Lemma C.4). □

Appendix D

Parallelism

Two graph transformations can be concurrent if they can be applied on the same graph in any order and it will always result in the same transformed graph. In this appendix, we investigate the parallel and sequential independences as shown in (Ehrig, Heckel, Korff et al. 1997) and (Ehrig, Ehrig, Prange et al. 2006b). We provide the basic definition and proofs of parallel and sequential independence and their specifications that needed to show PROV-GTS's characteristics, detailed proofs can be found in the references above. In the following theorems, we refer to the domain (source) of any morphism mor as $dom(mor)$, for example the domain of $mor : d \longrightarrow c$ is $dom(mor) = d$. In addition, the notation G_1/G_2 refers to the elements (edges and nodes) that are in G_1 but not in G_2.

Theorem D.1 (Parallel Independence)**.** *Let $d_1 = G \overset{m_1}{\underset{r_1}{\Longrightarrow}} H_1$ and $d_2 = G \overset{m_2}{\underset{r_2}{\Longrightarrow}} H_2$ be two graph transformations. Then d_2 is weakly parallel independent of d_1 if and only if $r_1^* \circ m_2 : L_2 \longrightarrow H_1$ is a match for r_2.*

$$\begin{tikzcd}
R_2 \arrow[d, "m_2^*"'] & L_2 \arrow[l, "r_2"'] \arrow[dr, "m_2"] & & L_1 \arrow[r, "r_1"] \arrow[dl, "m_1"'] & R_1 \arrow[d, "m_1^*"] \\
H_2 & & G \arrow[ll, "r_2^*"] \arrow[rr, "r_1^*"'] & & H_1
\end{tikzcd}$$

Proof. Let there be a match $\acute{m}_2 = r_1^* \circ m_2$. Then $m_2(L_2) \subseteq dom(r_1^*)$ by definition of composition. The commutativity of pushouts provides $m_1(L_1/dom(r_1)) \cap dom(r_1^*) = \emptyset$ which is the desired weak parallel independence.

Vice versa, we must show that $\acute{m}_2 = r_1^* \circ m_2$ is total. According to the construction of a transformation, each node which is not explicitly deleted by r_1 is preserved, i.e., $v \in G/m_1(L_1/dom(r_1))$ implies $v \in dom(r_1^*)$. This implies that each preimage of v under m_2 has also an image in H_1. Each edge which is not explicitly deleted by r_1 is either (i) preserved or (ii) implicitly deleted by the deletion of an incident node i.e., for each edge

$e \in G/m_1(L_1/dom(r_1))$ we either have (i) $e \in dom(r_1^*)$ which inherits the arguments for nodes; Otherwise, in case (ii), $s_G(e) \in m_1(L_1/dom(r_1))$ or $t_G(e) \in m_1(L_1/dom(r_1))$. By definition of parallel independence this implies $s_G(e) \notin m_2(L_2)$ or $t_G(e) \notin m_2(L_2)$ which, by definition of graph morphisms excludes $e \in m_2(L_2)$. In other words $e \in m_2(L_2)$ implies $e \in dom(r_1^*)$.

□

Theorem D.2 (Sequential Independence). *Assume two graph transformations* $d_1 = G \overset{m_1}{\underset{r_1}{\Longrightarrow}} H_1$ *and* $\acute{d}_2 = H_1 \overset{\acute{m}_2}{\underset{r_2}{\Longrightarrow}} X$ *to be given. Then* $\acute{d}_2$ *is weakly sequentially independent of* d_1 *if and only if there is a match* $m_2 : L_2 \longrightarrow G$ *for* r_2 *such that* $\acute{m}_2 = r_1^* \circ m_2$*. The transformation* $\acute{d}_2$ *is sequentially independent of* d_1 *if and only if* d_2^* *is weakly sequentially independent of* d_1 *and* d_1 *is weakly parallel independent of the correspondingly existing transformation* $d_2 = G \overset{m_2}{\underset{r_2}{\Longrightarrow}} H_2$*.*

$$\begin{tikzcd}
L_1 \arrow[r, "r_1"] \arrow[d, "m_1"'] & R_1 \arrow[dr, "m_1^*"'] & & L_2 \arrow[r, "r_2"] \arrow[dl, "\acute{m}_2"'] & R_2 \arrow[d, "\acute{m}_2^*"] \\
G \arrow[rr, "r_1^*"'] & & H_1 \arrow[rr, "r_2^*"'] & & X
\end{tikzcd}$$

Proof. the proof of this theorem is analogous to the case of weak parallel independence.

□

Definition D.3 (Pushout Properties). Regarding the pushout diagram below where $r^* : G \longrightarrow H$ and $m^* : R \longrightarrow H$ is the pushout of $r : L \longrightarrow R$ and $m : G \longrightarrow H$.

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "m"'] & R \arrow[d, "m^*"] \\
G \arrow[r, "r^*"] & H
\end{tikzcd}$$

Then the following properties are fulfilled:

- Pushouts preserve surjectivity, i.e. r surjective implies r^* surjective.
- Pushouts preserve injectivity, i.e. r injective implies r^* injective.
- r^* and m^* are jointly surjective.
- m conflict-free[1] implies m^* total.

[1] m is called conflict-free if $m(x) = m(y)$ implies $x, y \in dom(r)$ or $x, y \notin dom(r)$.

□

$$\begin{tikzcd}
 & L_1 \arrow[r, "r_1"] \arrow[d, "m_1"'] \arrow[dr, phantom, "(1)"] & R_1 \arrow[d, "m_1^*"] \\
L_2 \arrow[r, "m_2"] \arrow[d, "r_2"'] \arrow[dr, phantom, "(2)"] & G \arrow[r, "r_1^*"] \arrow[d, "r_2^*"] \arrow[dr, phantom, "(3)"] & H_1 \arrow[d, "m_1^{**}"] \\
R_2 \arrow[r, "m_2^*"'] & H_2 \arrow[r, "m_1^{**}"'] & X
\end{tikzcd}$$

Local-Church Rosser

Lemma D.4 (Weak Independence)**.** *Given a graph transformation* $d_1 = (G \xRightarrow[m_1]{r_1} H_1)$ *the following statements are equivalent:*

- *There is a graph transformation* $d_2 = (G \xRightarrow[m_2]{r_2} H_2)$ *which is weakly parallel independent of* d_1*.*
- *There is a graph transformation* $\acute{d_2} = (G \xRightarrow[\acute{m_2}]{r_2} X)$ *which is weakly sequential independent of* d_1*.*

Up to isomorphism, a bijective correspondence between 1. and 2. above is given by $\acute{m_2} = r_1^* \circ m_2$ *and* $m_2 = (\acute{r_2})^{-1} \circ m_2^*$*.*

Proof. As a direct consequence of the parallel independence and sequential independence proofs and the fact that injectivity of r implies the injectivity of r^* according to Definition D.3, then $m_2 = (r_1^*)^{-1} \circ r_1^* \circ m_2$ and $\acute{m_2} = r_1^* \circ (r_1^*)^{-1} \circ \acute{m_2}$. □

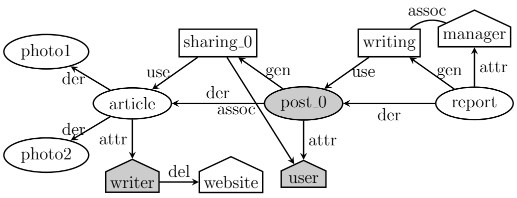
Proof of Local Church Rosser. Consider the diagram in above. Subdiagrams (1) and (2) depict the transformations d_l and d_2, respectively. Subdiagram (3) represents the pushout of r_1^* and r_2^*. The composition of pushout diagrams (2) and (3) yields a pushout diagram (2)+(3) which is a rule application $H_1 \overset{r_2,\acute{m_2}}{\Longrightarrow} X$ provided that $\acute{m_2} = r_1^* \circ m_2$ is a match, i.e., total. But this is ensured since d_l and d_2 have been required to be parallel independent. Analogously we obtain a rule application $H_2 \overset{r_1,\acute{m_1}}{\Longrightarrow} X$ by composing pushout diagrams (1) and (3). The stated sequential independence and the bijective correspondence follow from Lemma D.4.

□

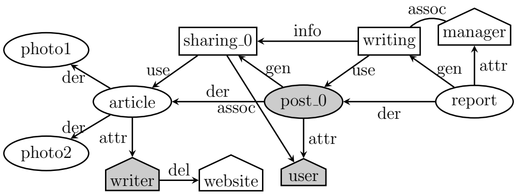
Appendix E

Step by step rule application

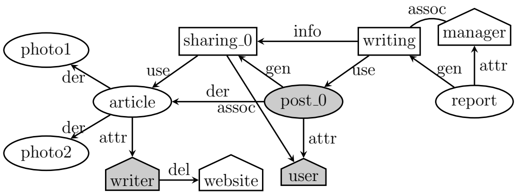
E.1 OnlinePost graph



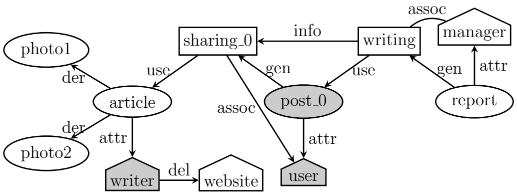
2-4 *crtInfUseGen*	use(writing, post_0), gen(post_0, sharing_0)
6-18 *delOutDer*	der(post_0,article)
6-16 *delInDer*	der(report, post_0)
4-10 *delAttNoOut*	attr(post_0,user)
6-17 *delInAtt*	
6-19 *delOutAtt*	
4-13 *delAssocNoOut*	assoc(sharing_0,user)



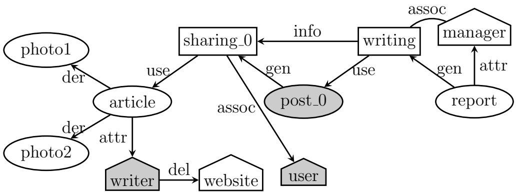
6-18 *delOutDer*	der(post_0,article)
6-16 *delInDer*	der(report, post_0)
4-10 *delAttNoOut*	attr(post_0,user)
6-17 *delInAtt*	
6-19 *delOutAtt*	
4-13 *delAssocNoOut*	assoc(sharing_0,user)



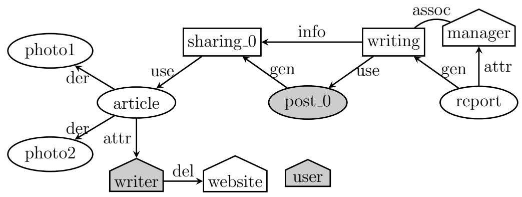
6-16 *delInDer*	der(report, post_0)
4-10 *delAttNoOut*	attr(post_0,user)
6-17 *delInAtt*	
6-19 *delOutAtt*	
4-13 *delAssocNoOut*	{assoc(sharing_0,user)}
8-24 *delUseEnt*	use(writing, post_0)



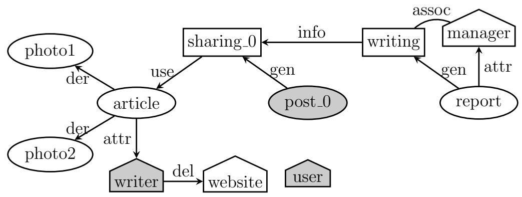
4-10 *delAttNoOut*	attr(post_0,user)
6-17 *delInAtt*	
6-19 *delOutAtt*	
4-13 *delAssocNoOut*	assoc(sharing_0,user)
8-24 *delUseEnt*	use(writing, post_0)



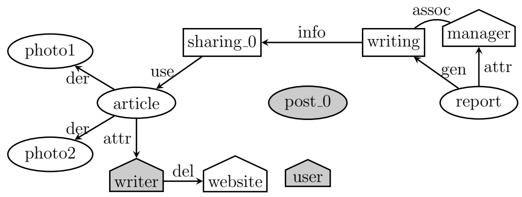
4-13 *delAssocNoOut*	assoc(sharing_0,user)
8-24 *delUseEnt*	use(writing, post_0)
8-23 *delGenEnt*	gen(post_0, sharing_0)



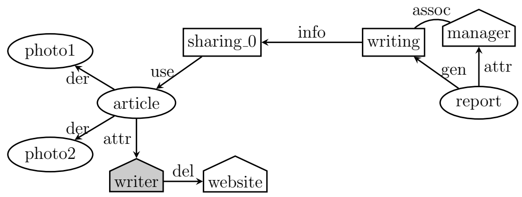
8-24 ***delUseEnt***	use(writing, post_0)
8-23 ***delGenEnt***	gen(post_0, sharing_0)



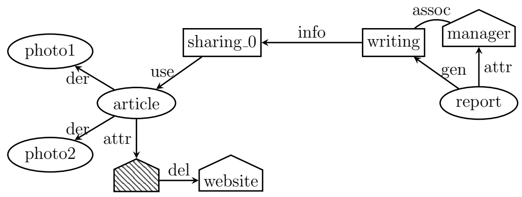
8-23 ***delGenEnt***	gen(post_0, sharing_0)



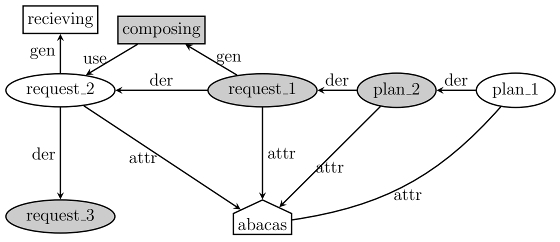
Deleting the isolated nodes



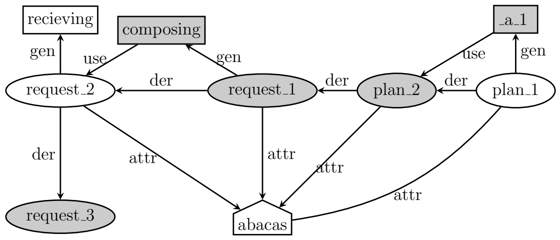
Anonymizing the remaining restricted nodes



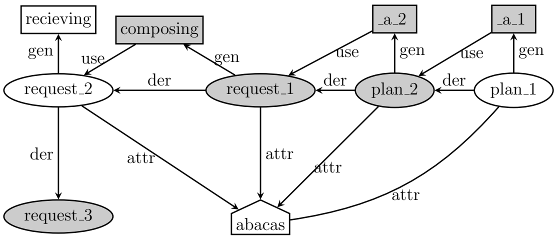
E.2 RideShare graph



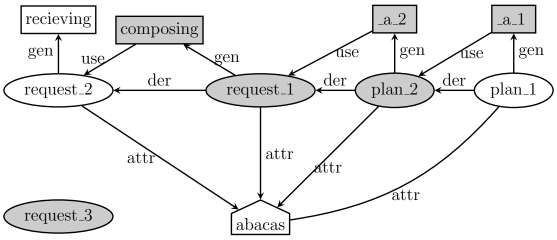
1-1 ***crtActOutDer***	der(plan_1, plan_2)
1-3 ***crtActInDer***	der(plan_2, request_2)
1-1 ***crtActOutDer***	
4-9 ***delDerNoOut***	der(request_2,request_3)
1-2 ***crtActAtt***	attr(request_1, abacas)
1-2 ***crtActAtt***	attr(plan_2, abacas)
6-18 ***delOutDer***	der(request_1, request_2)



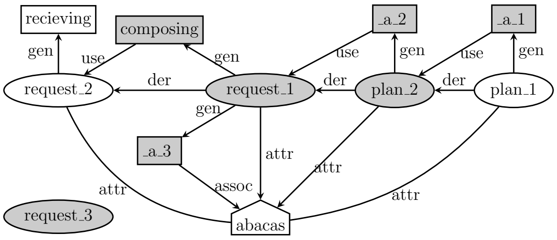
1-3 ***crtActInDer***	der(plan_2, request_2)
1-1 ***crtActOutDer***	
4-9 ***delDerNoOut***	der(request_2,request_3)
1-2 ***crtActAtt***	attr(request_1, abacas)
1-2 ***crtActAtt***	attr(plan_2, abacas)
6-18 ***delOutDer***	der(request_1, request_2)
6-16 ***delInDer***	der(plan_1, plan_2)



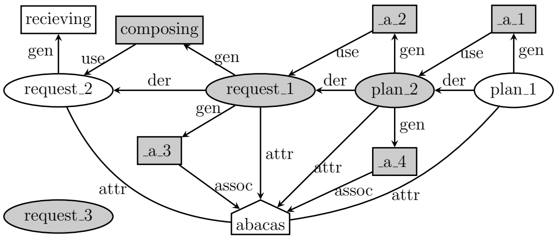
4-9 ***delDerNoOut***	der(request_2,request_3)
1-1 ***crtActOutDer***	
1-2 *crtActAtt*	attr(request_1, abacas)
1-2 *crtActAtt*	attr(plan_2, abacas)
6-18 *delOutDer*	der(request_1, request_2)
6-16 *delInDer*	der(plan_1, plan_2)
6-18 *delOutDer*	der(plan_2, request_1)
2-4 *crtInfUseGen*	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)



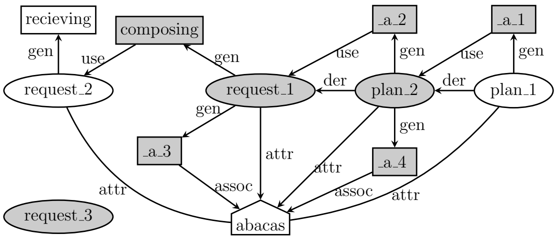
1-2 ***crtActAtt***	attr(request_1, abacas)
1-2 *crtActAtt*	attr(plan_2, abacas)
6-18 *delOutDer*	der(request_1, request_2)
6-16 *delInDer*	der(plan_1, plan_2)
6-18 *delOutDer*	der(plan_2, request_1)
2-4 *crtInfUseGen*	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)



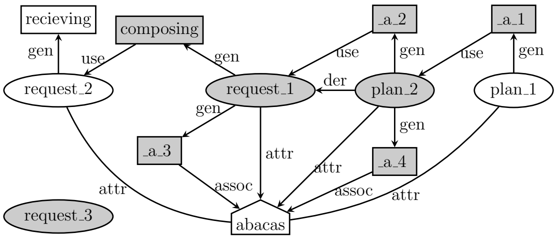
1-2 ***crtActAtt***	attr(plan_2, abacas)
6-18 *delOutDer*	der(request_1, request_2)
6-16 *delInDer*	der(plan_1, plan_2)
6-18 *delOutDer*	der(plan_2, request_1)
2-4 *crtInfUseGen*	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)
6-17 *delInAtt*	attr(request_1, abacas)
2-4 *crtInfUseGen*	info(_a_2, _a_3)



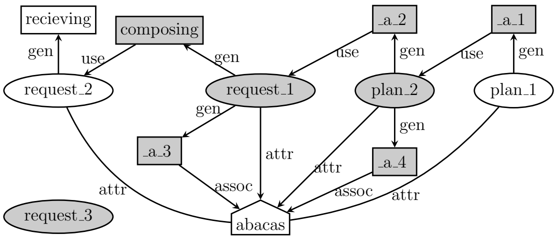
6-18 ***delOutDer***	der(request_1, request_2)
6-16 *delInDer*	der(plan_1, plan_2)
6-18 *delOutDer*	der(plan_2, request_1)
2-4 *crtInfUseGen*	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)
6-17 *delInAtt*	attr(request_1, abacas)
2-4 *crtInfUseGen*	info(_a_2, _a_3)
6-17 *delInAtt*	attr(plan_2, abacas)
2-4 *crtInfUseGen*	info(_a_1, _a_4)



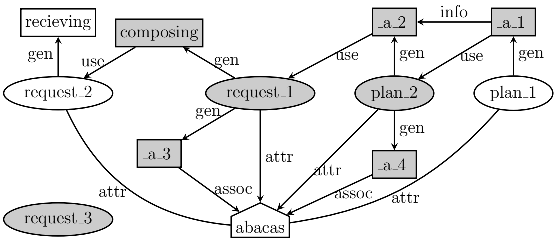
6-18 ***delOutDer***	der(plan_1, plan_2)
6-18 *delOutDer*	der(plan_2, request_1)
2-4 *crtInfUseGen*	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)
6-17 *delInAtt*	attr(request_1, abacas)
2-4 *crtInfUseGen*	info(_a_2, _a_3)
6-17 *delInAtt*	attr(plan_2, abacas)
2-4 *crtInfUseGen*	info(_a_1, _a_4)



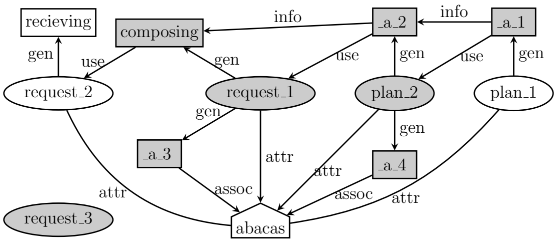
6-18 ***delOutDer***	der(plan_2, request_1)
2-4 *crtInfUseGen*	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)
6-17 *delInAtt*	attr(request_1, abacas)
2-4 *crtInfUseGen*	info(_a_2, _a_3)
6-17 *delInAtt*	attr(plan_2, abacas)
2-4 *crtInfUseGen*	info(_a_1, _a_4)



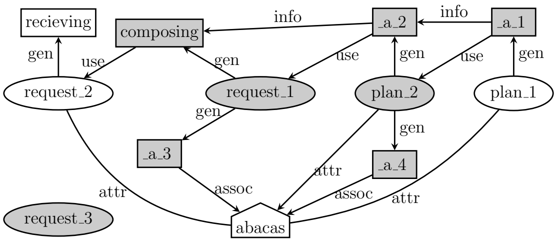
2-4 ***crtInfUseGen***	info(_a_1, _a_2)
2-4 *crtInfUseGen*	info(_a_2, composing)
6-17 *delInAtt*	attr(request_1, abacas)
2-4 *crtInfUseGen*	info(_a_2, _a_3)
6-17 *delInAtt*	attr(plan_2, abacas)
2-4 *crtInfUseGen*	info(_a_1, _a_4)



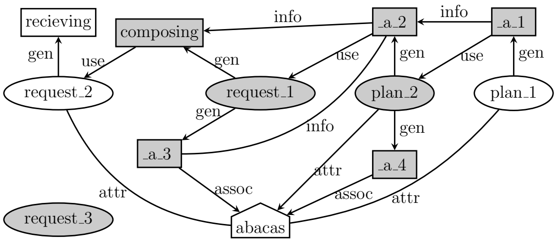
2-4 ***crtInfUseGen***	info(_a_2, composing)
6-17 ***delInAtt***	attr(request_1, abacas)
2-4 ***crtInfUseGen***	info(_a_2, _a_3)
6-17 ***delInAtt***	attr(plan_2, abacas)
2-4 ***crtInfUseGen***	info(_a_1, _a_4)



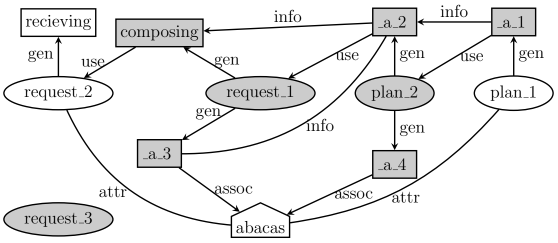
6-17 ***delInAtt***	attr(request_1, abacas)
2-4 ***crtInfUseGen***	info(_a_2, _a_3)
6-17 ***delInAtt***	attr(plan_2, abacas)
2-4 ***crtInfUseGen***	info(_a_1, _a_4)
8-23 ***delGenEnt***	gen(r_1, c)



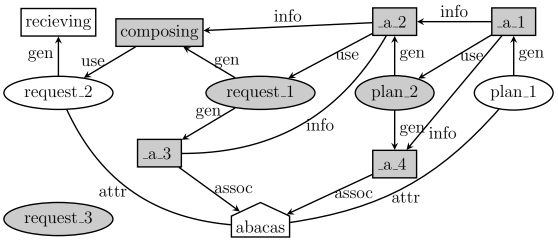
2-4 ***crtInfUseGen***	info(_a_2, _a_3)
6-17 ***delInAtt***	attr(plan_2, abacas)
2-4 ***crtInfUseGen***	info(_a_1, _a_4)
8-23 ***delGenEnt***	gen(r_1, c)
8-23 ***delGenEnt***	gen(p_2, _a_2)



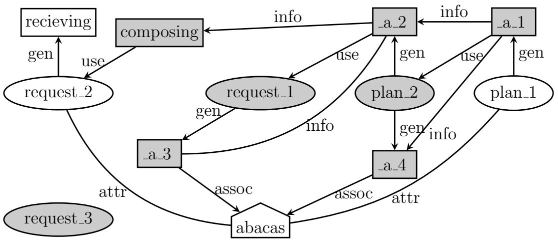
6-17 ***delInAtt***	attr(plan_2, abacas)
2-4 ***crtInfUseGen***	info(_a_1, _a_4)
8-23 ***delGenEnt***	gen(r_1, c)
8-23 ***delGenEnt***	gen(p_2, _a_2)
8-23 ***delGenEnt***	gen(r_1, _a_3)
8-24 ***delUseEnt***	use(_a_2, r_1)



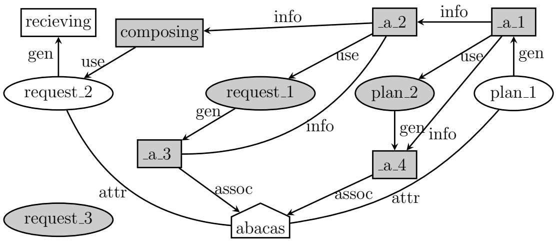
2-4 ***crtInfUseGen***	info(_a_1, _a_4)
8-23 ***delGenEnt***	gen(r_1, c)
8-23 ***delGenEnt***	gen(p_2, _a_2)
8-23 ***delGenEnt***	gen(r_1, _a_3)
8-24 ***delUseEnt***	use(_a_2, r_1)



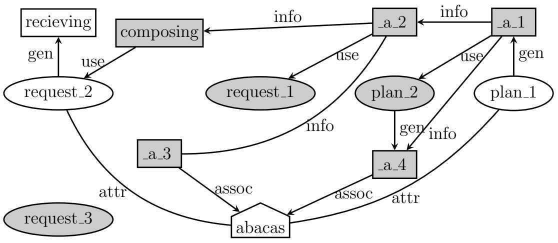
8-23 ***delGenEnt***	gen(r_1, c)
8-23 ***delGenEnt***	gen(p_2, _a_2)
8-23 ***delGenEnt***	gen(r_1, _a_3)
8-24 ***delUseEnt***	use(_a_2, r_1)
8-23 ***delGenEnt***	gen(p_2, _a_4)
8-24 ***delUseEnt***	use(_a_1, p_2)



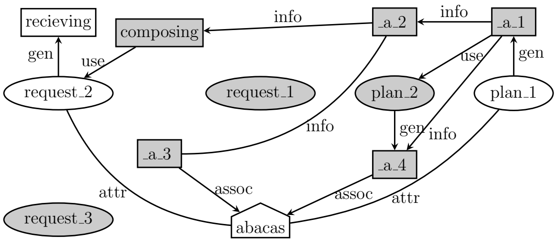
8-23 ***delGenEnt***	gen(p_2, _a_2)
8-23 ***delGenEnt***	gen(r_1, _a_3)
8-24 ***delUseEnt***	use(_a_2, r_1)
8-23 ***delGenEnt***	gen(p_2, _a_4)
8-24 ***delUseEnt***	use(_a_1, p_2)



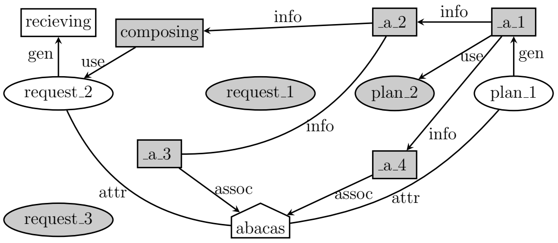
8-23 ***delGenEnt***	gen(r_1, _a_3)
8-24 ***delUseEnt***	use(_a_2, r_1)
8-23 ***delGenEnt***	gen(p_2, _a_4)
8-24 ***delUseEnt***	use(_a_1, p_2)



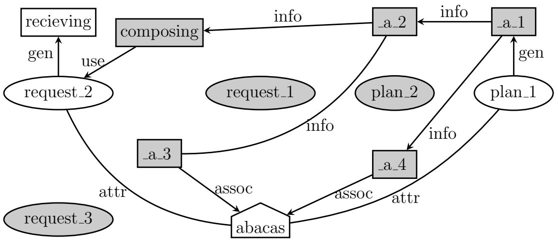
8-24 ***delUseEnt***	use(_a_2, r_1)
8-23 ***delGenEnt***	gen(p_2, _a_4)
8-24 ***delUseEnt***	use(_a_1, p_2)



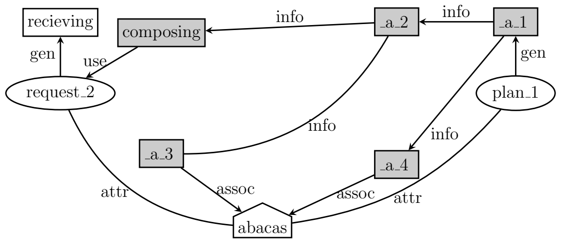
8-23 ***delGenEnt***	gen(p_2, _a_4)
8-24 ***delUseEnt***	use(_a_1, p_2)



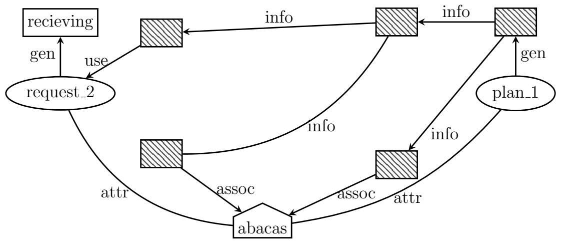
8-24 ***delUseEnt***	use(_a_1, p_2)



Deleting the isolated nodes



Anonymizing the remaining restricted nodes



Bibliography

Aho, A. V., J. E. Hopcroft and J. D. Ullman (1973). "On Finding Lowest Common Ancestors in Trees". In: *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. STOC '73. Austin, Texas, USA: ACM, pp. 253–265. DOI: 10.1145/800125.804056. URL: http://doi.acm.org/10.1145/800125.804056.

Ainy, Eleanor, Pierre Bourhis, Susan B. Davidson, Daniel Deutch and Tova Milo (2015). "Approximated Summarization of Data Provenance". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: ACM, pp. 483–492. ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806429. URL: http://doi.acm.org/10.1145/2806416.2806429.

Ainy, Eleanor, Susan B Davidson, Daniel Deutch and Tova Milo (2014). "Approximated Provenance for Complex Applications". In: *6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. Cologne: USENIX Association. URL: https://www.usenix.org/conference/tapp2014/agenda/presentation/ainy.

Anand, Manish Kumar, Shawn Bowers and Bertram Ludäscher (2012). "Database Support for Exploring Scientific Workflow Provenance Graphs". In: *Scientific and Statistical Database Management: 24th International Conference, SSDBM 2012, Chania, Crete, Greece, June 25-27, 2012. Proceedings*. Ed. by Anastasia Ailamaki and Shawn Bowers. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 343–360. ISBN: 978-3-642-31235-9. DOI: 10.1007/978-3-642-31235-9_23. URL: http://dx.doi.org/10.1007/978-3-642-31235-9_23.

Andries, Marc, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jrg Kreowski, Sabine Kuske, Detlef Plump, Andy Schrr and Gabriele Taentzer (1999). "Graph transformation for specification and programming". In: *Science of Computer Programming* 34.1, pp. 1 –54. ISSN: 0167-6423. DOI: http://dx.doi.org/10.1016/S0167-6423(98)00023-9. URL: http://www.sciencedirect.com/science/article/pii/S0167642398000239.

Awodey, Steve (2010). *Category Theory*. 2nd. New York, NY, USA: Oxford University Press, Inc. ISBN: 0199237182, 9780199237180.

Backstrom, Lars, Cynthia Dwork and Jon Kleinberg (2007). "Wherefore Art Thou R3579x?: Anonymized Social Networks, Hidden Patterns, and Structural Steganography". In: *Proceedings of the 16th International Conference on World Wide Web*. WWW '07. Banff, Alberta, Canada: ACM, pp. 181–190. ISBN: 978-1-59593-654-7.

DOI: 10.1145/1242572.1242598. URL: http://doi.acm.org/10.1145/1242572.1242598.

Barbier, Geoffrey, Zhuo Feng, Pritam Gundecha and Huan Liu (2013). *Provenance data in social media*. Vol. 4. 1. Morgan & Claypool Publishers, pp. 1–84. ISBN: 1608457834, 9781608457830.

Bardohl, Roswitha, Hartmut Ehrig, Juan De Lara, Olga Runge, Gabi Taentzer and Ingo Weinhold (2003). *Node type inheritance concept for typed graph transformation*. Tech. rep.

Baresi, Luciano and Reiko Heckel (2002). “Tutorial introduction to graph transformation: A software engineering perspective”. In: *International Conference on Graph Transformation*. Springer Berlin Heidelberg, pp. 402–429. ISBN: 978-3-540-45832-6. DOI: 10.1007/3-540-45832-8_30. URL: http://dx.doi.org/10.1007/3-540-45832-8_30.

Barr, Michael and Charles Wells, eds. (1995). *Category Theory for Computing Science, 2Nd Ed.* Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd. ISBN: 0-13-323809-1.

Blaustein, Barbara, Adriane Chapman, Len Seligman, M. David Allen and Arnon Rosenthal (2011). “Surrogate Parenthood: Protected and Informative Graphs”. In: *Proceedings of the VLDB Endowment* 4.8, pp. 518–525. ISSN: 2150-8097. DOI: 10.14778/2002974.2002979. URL: http://dx.doi.org/10.14778/2002974.2002979.

Braun, Uri, Avraham Shinnar and Margo Seltzer (2008). “Securing Provenance”. In: *Proceedings of the 3rd Conference on Hot Topics in Security*. HOTSEC’08. San Jose, CA: USENIX Association, 4:1–4:5. URL: http://dl.acm.org/citation.cfm?id=1496671.1496675.

Brill, Eric (1995). “Transformation-based Error-driven Learning and Natural Language Processing: A Case Study in Part-of-speech Tagging”. In: *Computational linguistics* 21.4, pp. 543–565. ISSN: 0891-2017. URL: http://dl.acm.org/citation.cfm?id=218355.218367.

Broek, PM Van den (1991). “Algebraic graph rewriting using a single pushout”. In: *TAPSOFT ’91: Proceedings of the International Joint Conference on Theory and Practice of Software Development Brighton, UK, April 8–12, 1991*. Ed. by S. Abramsky and T. S. E. Maibaum. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 90–102. ISBN: 978-3-540-46563-8. DOI: 10.1007/3-540-53982-4_6. URL: http://dx.doi.org/10.1007/3-540-53982-4_6.

Buneman, Peter, Adriane Chapman and James Cheney (2006). “Provenance Management in Curated Databases”. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’06. Chicago, IL, USA: ACM, pp. 539–550. ISBN: 1-59593-434-0. DOI: 10.1145/1142473.1142534. URL: http://doi.acm.org/10.1145/1142473.1142534.

Cadenhead, Tyrone, Murat Kantarcioglu and Bhavani Thuraisingham (2011). “A Framework for Policies over Provenance”. In: vol. 501, p. 1.

Cadenhead, Tyrone, Vaibhav Khadilkar, Murat Kantarcioglu and Bhavani Thuraisingham (2011a). "A Language for Provenance Access Control". In: *Proceedings of the First ACM Conference on Data and Application Security and Privacy*. CODASPY '11. San Antonio, TX, USA: ACM, pp. 133–144. ISBN: 978-1-4503-0466-5. DOI: 10.1145/1943513.1943532. URL: http://doi.acm.org/10.1145/1943513.1943532.

– (2011b). "Transforming Provenance Using Redaction". In: *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*. SACMAT '11. Innsbruck, Austria: ACM, pp. 93–102. ISBN: 978-1-4503-0688-1. DOI: 10.1145/1998441.1998456. URL: http://doi.acm.org/10.1145/1998441.1998456.

Cellan-Jones, BBC Rory (2016). *Facebook, fake news and the meaning of truth*. URL: http://www.bbc.co.uk/news/technology-38106131 (visited on 27/11/2016).

Chapman, Adriane P., H. V. Jagadish and Prakash Ramanan (2008). "Efficient Provenance Storage". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, pp. 993–1006. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376715. URL: http://doi.acm.org/10.1145/1376616.1376715.

Cheah, You-Wei, Beth Plale, Joey Kendall-Morwick, David Leake and Lavanya Ramakrishnan (2012). "A Noisy 10GB Provenance Database". In: *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part II*. Ed. by Florian Daniel, Kamel Barkaoui and Schahram Dustdar. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 370–381. ISBN: 978-3-642-28115-0. DOI: 10.1007/978-3-642-28115-0_35. URL: http://dx.doi.org/10.1007/978-3-642-28115-0_35.

Chebotko, Artem, Seunghan Chang, Shiyong Lu, Farshad Fotouhi and Ping Yang (2008). "Scientific Workflow Provenance Querying with Security Views". In: *Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management*. WAIM '08. IEEE Computer Society, pp. 349–356. ISBN: 978-0-7695-3185-4. DOI: 10.1109/WAIM.2008.41. URL: http://dx.doi.org/10.1109/WAIM.2008.41.

Chebotko, Artem, Shiyong Lu, Seunghan Chang, Farshad Fotouhi and Ping Yang (2010). "Secure abstraction views for scientific workflow provenance querying". In: *IEEE Transactions on Services Computing* 3.4, pp. 322–337.

Chen, L., P. Edwards, J. D. Nelson and T. J. Norman (2015). "An access control model for protecting provenance graphs". In: *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, pp. 125–132. DOI: 10.1109/PST.2015.7232963.

Cheney, James, Paolo Missier, Luc Moreau and Tom DeNies (2013). *Constraints of the PROV Data Model*. Tech. rep. W3C Recommendation, W3C, http://www.w3.org/TR/prov-constraints/.

Cheney, James and Roly Perera (2014). "An Analytical Survey of Provenance Sanitization". In: *Provenance and Annotation of Data and Processes : 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany, June 9-13, 2014*. Springer International Publishing, pp. 113–126. ISBN: 978-3-319-16462-5. DOI:

10.1007/978-3-319-16462-5_9. URL: http://dx.doi.org/10.1007/978-3-319-16462-5_9.

Church, Alonzo and J. B. Rosser (1936). "Some Properties of Conversion". In: *Transactions of the American Mathematical Society* 39.3, pp. 472–482. ISSN: 00029947. URL: http://www.jstor.org/stable/1989762.

Danger, Roxana, Vasa Curcin, Paolo Missier and Jeremy Bryans (2015). "Access Control and View Generation for Provenance Graphs". In: *Future Generation Computer Systems* 49.C, pp. 8–27. ISSN: 0167-739X. DOI: 10.1016/j.future.2015.01.014. URL: http://dx.doi.org/10.1016/j.future.2015.01.014.

Davidson, Susan, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen, Yi Chen and Tova Milo (2011a). "Enabling privacy in provenance-aware workflow systems". In: *The biennial Conference on Innovative Data Systems Research (CIDR)*.

Davidson, Susan B. and Juliana Freire (2008). "Provenance and Scientific Workflows: Challenges and Opportunities". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, pp. 1345–1350. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376772. URL: http://doi.acm.org/10.1145/1376616.1376772.

Davidson, Susan B., Sanjeev Khanna, Sudeepa Roy and Sarah Cohen Boulakia (2010). "Privacy Issues in Scientific Workflow Provenance". In: *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*. Wands '10. Indianapolis, Indiana: ACM, 3:1–3:6. ISBN: 978-1-4503-0188-6. DOI: 10.1145/1833398.1833401. URL: http://doi.acm.org/10.1145/1833398.1833401.

Davidson, Susan B., Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen and Yi Chen (2011b). "On Provenance and Privacy". In: *Proceedings of the 14th International Conference on Database Theory*. ICDT '11. Uppsala, Sweden: ACM, pp. 3–10. ISBN: 978-1-4503-0529-7. DOI: 10.1145/1938551.1938554. URL: http://doi.acm.org/10.1145/1938551.1938554.

Dey, Saumen and Bertram Ludäscher (2013). "A Declarative Approach to Customize Workflow Provenance". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. EDBT '13. Genoa, Italy: ACM, pp. 9–16. ISBN: 978-1-4503-1599-9. DOI: 10.1145/2457317.2457320. URL: http://doi.acm.org/10.1145/2457317.2457320.

Dey, Saumen, Daniel Zinn and Bertram Ludäscher (2011a). "ProPub: Towards a Declarative Approach for Publishing Customized, Policy-Aware Provenance". English. In: *Scientific and Statistical Database Management*. Ed. by Judith Bayard Cushing, James French and Shawn Bowers. Vol. 6809. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 225–243. ISBN: 978-3-642-22350-1. DOI: 10.1007/978-3-642-22351-8_13. URL: http://dx.doi.org/10.1007/978-3-642-22351-8_13.

– (2011b). "Repairing provenance policy violations by inventing non-functional nodes". In: vol. 737. ii, pp. 109–122.

– (2012). "Reconciling Provenance Policy Conflicts by Inventing Anonymous Nodes". In: *The Semantic Web: ESWC 2011 Workshops: ESWC 2011 Workshops, Heraklion,* *Greece, May 29-30, 2011, Revised Selected Papers*. Ed. by Raúl García-Castro, Dieter Fensel and Grigoris Antoniou. Springer Berlin Heidelberg, pp. 172–185. ISBN: 978-3-642-25953-1. DOI: 10.1007/978-3-642-25953-1_14. URL: http://dx.doi.org/10.1007/978-3-642-25953-1_14.

Ehrig, H., G. Engels, H.-J. Kreowski and G. Rozenberg, eds. (1999). *Handbook of Graph Grammars and Computing by Graph Transformation: Vol. 2: Applications, Languages, and Tools*. World Scientific Publishing Co., Inc. ISBN: 981-02-4020-1.

Ehrig, H., R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner and A. Corradini (1997). "Handbook of Graph Grammars and Computing by Graph Transformation". In: ed. by Grzegorz Rozenberg. World Scientific Publishing Co., Inc. Chap. Algebraic Approaches to Graph Transformation. Part II: Single Pushout Approach and Comparison with Double Pushout Approach, pp. 247–312. ISBN: 98-102288-48. URL: http://dl.acm.org/citation.cfm?id=278918.278930.

Ehrig, H., M. Pfender and H. J. Schneider (1973). "Graph-grammars: An algebraic approach". In: *14th Annual Symposium on Switching and Automata Theory (swat 1973)*. IEEE, pp. 167–180. DOI: 10.1109/SWAT.1973.11.

Ehrig, Hartmut, Karsten Ehrig, Juan de Lara, Gabriele Taentzer, Daniel Varro and Szilvia Varro-Gyapay (2005). "Termination Criteria for Model Transformation". English. In: *Fundamental Approaches to Software Engineering*. Ed. by Maura Cerioli. Vol. 3442. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 49–63. ISBN: 978-3-540-25420-1. DOI: 10.1007/978-3-540-31984-9_5. URL: http://dx.doi.org/10.1007/978-3-540-31984-9_5.

Ehrig, Hartmut, Karsten Ehrig, Ulrike Prange and Gabriele Taentzer (2006a). "Fundamental Theory for Typed Attributed Graphs and Graph Transformation Based on Adhesive HLR Categories". In: *Fundam. Inf.* 74.1, pp. 31–61. ISSN: 0169-2968. URL: http://dl.acm.org/citation.cfm?id=2369448.2369451.

– (2006b). *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series). Secaucus*. Vol. 373. NJ, USA: Springer-Verlag New York, Inc.

– (2006c). "Graphs, Typed Graphs, and the Gluing Construction". In: *Fundamentals of Algebraic Graph Transformation*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 21–35. ISBN: 978-3-540-31188-1. DOI: 10.1007/3-540-31188-2_2. URL: http://dx.doi.org/10.1007/3-540-31188-2_2.

Ehrig, Hartmut, Ulrike Golas, Annegret Habel, Leen Lambers and Fernando Orejas (2014). "M-Adhesive transformation systems with nested application conditions. Part 1: parallelism, concurrency and amalgamation". In: 24.4. DOI: 10.1017/S0960129512000357.

Ehrig, Hartmut, Ulrike Prange and Gabriele Taentzer (2004). "Fundamental Theory for Typed Attributed Graph Transformation". In: *Graph Transformations: Second International Conference, ICGT 2004, Rome, Italy, September 28–October 1, 2004. Proceedings*. Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce and

Grzegorz Rozenberg. Springer Berlin Heidelberg, pp. 161–177. ISBN: 978-3-540-30203-2. DOI: 10.1007/978-3-540-30203-2_13. URL: http://dx.doi.org/10.1007/978-3-540-30203-2_13.

Eilenberg, Samuel and Saunders MacLane (1945). "General Theory of Natural Equivalences". In: *Transactions of the American Mathematical Society* 58.2, pp. 231–294. ISSN: 00029947. DOI: 10.2307/1990284. URL: http://www.jstor.org/stable/1990284.

Fischer, Joel E., Wenchao Jiang and Stuart Moran (2012). "AtomicOrchid: A Mixed Reality Game to Investigate Coordination in Disaster Response". In: *Entertainment Computing - ICEC 2012: 11th International Conference, ICEC 2012, Bremen, Germany, September 26-29, 2012. Proceedings*. Ed. by Marc Herrlich, Rainer Malaka and Maic Masuch. Springer Berlin Heidelberg, pp. 572–577. ISBN: 978-3-642-33542-6. DOI: 10.1007/978-3-642-33542-6_75. URL: http://dx.doi.org/10.1007/978-3-642-33542-6_75.

Fischer, JoelE., Wenchao Jiang, Andruid Kerne, Chris Greenhalgh, SarvapaliD. Ramchurn, Steven Reece, Nadia Pantidi and Tom Rodden (2014). "Supporting Team Coordination on the Ground: Requirements from a Mixed Reality Game". English. In: *COOP 2014 - Proceedings of the 11th International Conference on the Design of Cooperative Systems, 27-30 May 2014, Nice (France)*. Ed. by Chiara Rossitto, Luigina Ciolfi, David Martin and Bernard Conein. Springer International Publishing, pp. 49–67. ISBN: 978-3-319-06497-0. DOI: 10.1007/978-3-319-06498-7_4. URL: http://dx.doi.org/10.1007/978-3-319-06498-7_4.

Freire, Juliana, David Koop, Emanuele Santos and Cláudio T. Silva (2008). "Provenance for Computational Tasks: A Survey". In: *Computing in Science and Engineering* 10.3, pp. 11–21. DOI: http://dx.doi.org/10.1109/MCSE.2008.79. URL: http://scitation.aip.org/content/aip/journal/cise/10/3/10.1109/MCSE.2008.79.

Garijo, Daniel, Oscar Corcho and Yolanda Gil (2013). "Detecting Common Scientific Workflow Fragments Using Templates and Execution Provenance". In: *Proceedings of the Seventh International Conference on Knowledge Capture*. K-CAP '13. Banff, Canada: ACM, pp. 33–40. ISBN: 978-1-4503-2102-0. DOI: 10.1145/2479832.2479848. URL: http://doi.acm.org/10.1145/2479832.2479848.

Gil, Yolanda, Simon Miles, Khalid Belhajjame, Helena Deus, Daniel Garijo, Graham Klyne, Paolo Missier, Stian Soiland-Reyes and Stephen Zednik (2013). *PROV model primer*. Tech. rep. Technical Report, W3C Working Group, 2013. URL: http://www.w3.org/TR/prov-primer/.

Groth, Paul, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou and Luc Moreau (2006). *An architecture for provenance systems*. Technical Report. University of Southampton. URL: http://eprints.soton.ac.uk/263216/.

Groth, Paul and Luc Moreau (2013). "PROV verview". In: *W3C Working Group Note*. URL: http://www.w3.org/TR/prov-overview/.

Habel, Annegret, Reiko Heckel and Gabriele Taentzer (1996). “Graph grammars with negative application conditions”. In: *Fundamenta Informaticae* 26.3,4, pp. 287–313. ISSN: 0169-2968. URL: http://dl.acm.org/citation.cfm?id=2379538.2379542.

Habel, Annegret and Karl-Heinz Pennemann (2005). “Nested Constraints and Application Conditions for High-Level Structures”. In: *Formal Methods in Software and Systems Modeling*. Ed. by Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg and Gabriele Taentzer. Springer Berlin Heidelberg, pp. 293–308. ISBN: 978-3-540-31847-7. DOI: 10.1007/978-3-540-31847-7_17. URL: http://dx.doi.org/10.1007/978-3-540-31847-7_17.

Habel, Annegret and Detlef Plump (2001). “Computational completeness of programming languages based on graph transformation”. In: *International Conference on Foundations of Software Science and Computation Structures*. Springer Berlin Heidelberg, pp. 230–245. ISBN: 978-3-540-45315-4. DOI: 10.1007/3-540-45315-6_15. URL: http://dx.doi.org/10.1007/3-540-45315-6_15.

Hasan, Ragib, Radu Sion and Marianne Winslett (2007). “Introducing secure provenance: problems and challenges”. In: *Proceedings of the 2007 ACM workshop on Storage security and survivability*. Alexandria, Virginia, USA: ACM, pp. 13–18. ISBN: 978-1-59593-891-6. DOI: 10.1145/1314313.1314318. URL: http://doi.acm.org/10.1145/1314313.1314318.

Heckel, Reiko (2006). “Graph Transformation in a Nutshell”. In: *Electron. Notes Theor. Comput. Sci.* 148.1, pp. 187–198. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2005.12.018. URL: http://dx.doi.org/10.1016/j.entcs.2005.12.018.

Hussein, Jamal, Luc Moreau and Vladimiro Sassone (2015). “Obscuring Provenance Confidential Information via Graph Transformation”. In: *Trust Management IX: 9th IFIP WG 11.11 International Conference, IFIPTM 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*. Vol. 454. IFIP Advances in Information and Communication Technology. Springer International Publishing, pp. 109–125. ISBN: 978-3-319-18491-3. DOI: 10.1007/978-3-319-18491-3_8. URL: http://link.springer.com/chapter/10.1007/978-3-319-18491-3_8.

Hussein, Jamal, Vladimiro Sassone and Luc Moreau (2016). “A template-based graph transformation system for the PROV data model”. In: *Seventh International Workshop on Graph Computation Models*. URL: http://eprints.soton.ac.uk/397032/.

Huynh, Trung Dong, Mark Ebden, Sarvapali Ramchurn, Stephen Roberts and Luc Moreau (2014). “Data quality assessment from provenance graphs”. URL: http://eprints.soton.ac.uk/365510/.

Huynh, Trung Dong, Danius T Michaelides and Luc Moreau (2016). “PICASO: Provenance Interlinking and Collective Authoring for Scientific Objects”. In:

Huynh, Trung Dong and Luc Moreau (2014). “ProvStore: A Public Provenance Repository”. In: *Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany, June 9-13, 2014. Revised Selected Papers*. Ed. by Bertram Ludäscher and Beth Plale. Springer

International Publishing, pp. 275–277. ISBN: 978-3-319-16462-5. DOI: 10.1007/978-3-319-16462-5_32. URL: http://dx.doi.org/10.1007/978-3-319-16462-5_32.

Jiang, Wenchao, J.E. Fischer, C. Greenhalgh, S.D. Ramchurn, Feng Wu, N.R. Jennings and T. Rodden (2014). “Social implications of agent-based planning support for human teams”. In: *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, pp. 310–317. DOI: 10.1109/CTS.2014.6867582.

Karvounarakis, Grigoris, Zachary G. Ives and Val Tannen (2010). “Querying Data Provenance”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’10. Indianapolis, Indiana, USA: ACM, pp. 951–962. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807269. URL: http://doi.acm.org/10.1145/1807167.1807269.

Keller, S. E., J. A. Perkins, T. F. Payton and S. P. Mardinly (1984). “Tree Transformation Techniques and Experiences”. In: *Proceedings of the 1984 SIGPLAN Symposium on Compiler Construction*. SIGPLAN ’84. Montreal, Canada: ACM, pp. 190–201. ISBN: 0-89791-139-3. DOI: 10.1145/502874.502893. URL: http://doi.acm.org/10.1145/502874.502893.

Kifor, Tamás, László Z Varga, Sergio Álvarez, Javier Vázquez-Salceda and Steven Willmott (2006). “Privacy issues of provenance in electronic healthcare record systems”. In: *Proceedings of the 1st Int. Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE 2006)*. URL: http://twiki.pasoa.ecs.soton.ac.uk/pub/Provenance/ProjectPublications/EHCR-Prov-Privacy.pdf.

Kifor, Tamas, LS Varga, Javier Vazquez-Salceda, Sergio Alvarez, Steven Willmott, Simon Miles and Luc Moreau (2006). “Provenance in agent-mediated healthcare systems”. In: *Intelligent Systems, IEEE* 21.6, pp. 38–46. ISSN: 1541-1672. DOI: 10.1109/MIS.2006.119.

König, Barbara and Jan Stückrath (2012). “Well-Structured Graph Transformation Systems with Negative Application Conditions”. In: *Graph Transformations: 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings*. Ed. by Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski and Grzegorz Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 81–95. ISBN: 978-3-642-33654-6. DOI: 10.1007/978-3-642-33654-6_6. URL: http://dx.doi.org/10.1007/978-3-642-33654-6_6.

Kwasnikowska, Natalia, Luc Moreau and Jan Van Den Bussche (2015). “A Formal Account of the Open Provenance Model”. In: *ACM Trans. Web* 9.2, 10:1–10:44. ISSN: 1559-1131. DOI: 10.1145/2734116. URL: http://doi.acm.org/10.1145/2734116.

Lawvere, F. William (1963). “Functorial Semantics of Algebraic Theories”. In: *Proceedings of the National Academy of Sciences of the United States of America* 50.5, pp. 869–872. ISSN: 00278424. URL: http://www.jstor.org/stable/71935.

Lebo, Timothy, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik and Jun Zhao

(2013). *PROV-O: The PROV Ontology*. Tech. rep. W3C Recommendation, W3C, http://www.w3.org/TR/prov-o/.

Lim, Hyo-Sang, Yang-Sae Moon and Elisa Bertino (2010). "Provenance-based Trustworthiness Assessment in Sensor Networks". In: *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks*. DMSN '10. Singapore: ACM, pp. 2–7. ISBN: 978-1-4503-0416-0. DOI: 10.1145/1858158.1858162. URL: http://doi.acm.org/10.1145/1858158.1858162.

Löwe, Michael (1993). "Algebraic Approach to Single-pushout Graph Transformation". In: *Theor. Comput. Sci.* 109.1-2, pp. 181–224. ISSN: 0304-3975. DOI: 10.1016/0304-3975(93)90068-5. URL: http://dx.doi.org/10.1016/0304-3975(93)90068-5.

Löwe, Michael and Hartmut Ehrig (1991). "Algebraic approach to graph transformation based on single pushout derivations". In: *Graph-theoretic concepts in computer science*. Springer, pp. 338–353.

Mens, Tom and Pieter Van Gorp (2006). "A Taxonomy of Model Transformation". In: *Electronic Notes in Theoretical Computer Science* 152, pp. 125–142. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2005.10.021. URL: http://dx.doi.org/10.1016/j.entcs.2005.10.021.

Michaelides, Danius, Dong Huynh and Luc Moreau (2014). *PROV-TEMPLATE: A Template System for PROV Documents*. URL: https://provenance.ecs.soton.ac.uk/prov-template/.

Missier, Paolo (2013). *Preserving privacy in shared provenance data*. Computing Science, Newcastle University.

Missier, Paolo, Jeremy Bryans, Carl Gamble, Vasa Curcin and Roxana Danger (2013). *Provenance graph abstraction by node grouping*. Computing Science, Newcastle University.

– (2015). "ProvAbs: Model, Policy, and Tooling for Abstracting PROV Graphs". In: *Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop, IPAW 2014, Cologne, Germany, June 9-13, 2014. Revised Selected Papers*. Ed. by Bertram Ludäscher and Beth Plale. Springer International Publishing, pp. 3–15. ISBN: 978-3-319-16462-5. DOI: 10.1007/978-3-319-16462-5_1. URL: http://dx.doi.org/10.1007/978-3-319-16462-5_1.

Moreau, L, B Batlajery, T Huynh, D Michaelides and H Packer (2016). *prov-template evaluation dataset*. URL: http://eprints.soton.ac.uk/390436/.

Moreau, Luc (2010). "The Foundations for Provenance on the Web". In: *Foundations and Trends in Web Science* 2.2&#8211;3, pp. 99–241. ISSN: 1555-077X. DOI: 10.1561/1800000010. URL: http://dx.doi.org/10.1561/1800000010.

– (2014). "Aggregation by Provenance Types: A Technique for Summarising Provenance Graphs". In: *GRAPHS AS MODELS 2015*. Vol. 181. Electronic Proceedings in Theoretical Computer Science, pp. 129–144. DOI: 10.4204/EPTCS.181.9. URL: https://arxiv.org/abs/1504.02616.

Moreau, Luc, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan and Jan Van den Bussche (2011). "The Open Provenance Model Core Specification (V1.1)". In: *Future Generation Computer Systems* 27.6, pp. 743–756. ISSN: 0167-739X. DOI: 10.1016/j.future.2010.07.005. URL: http://dx.doi.org/10.1016/j.future.2010.07.005.

Moreau, Luc, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers and Patrick Paulson (2008). "The Open Provenance Model: An Overview". In: *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers.* Ed. by Juliana Freire, David Koop and Luc Moreau. Springer Berlin Heidelberg, pp. 323–326. ISBN: 978-3-540-89965-5. DOI: 10.1007/978-3-540-89965-5_31. URL: http://dx.doi.org/10.1007/978-3-540-89965-5_31.

Moreau, Luc and Paul Groth (2013). "Provenance: An Introduction to PROV". In: *Synthesis Lectures on the Semantic Web: Theory and Technology* 3.4, pp. 1–129.

Moreau, Luc, Paul Groth, James Cheney, Timothy Lebo and Simon Miles (2015). "The Rationale of PROV". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 35.P4, pp. 235–257. ISSN: 1570-8268. DOI: 10.1016/j.websem.2015.04.001. URL: http://dx.doi.org/10.1016/j.websem.2015.04.001.

Moreau, Luc, Paul Groth, Ivan Herman and Sandro Hawke (2011). *W3C PROV Working Group.*

Moreau, Luc, Trung Dong Huynh and Danius Michaelides (2014). "An online validator for provenance: algorithmic design, testing, and API". In: *International Conference on Fundamental Approaches to Software Engineering (FASE'14).* Springer. Springer Berlin Heidelberg, pp. 291–305. ISBN: 978-3-642-54804-8. DOI: 10.1007/978-3-642-54804-8_20. URL: http://dx.doi.org/10.1007/978-3-642-54804-8_20.

Moreau, Luc, Bertram Ludäscher, Ilkay Altintas, Roger S. Barga, Shawn Bowers, Steven Callahan, George Chin, Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, Susan Davidson, Ewa Deelman, Luciano Digiampietri, Ian Foster, Juliana Freire, James Frew, Joe Futrelle, Tara Gibson, Yolanda Gil, Carole Goble, Jennifer Golbeck, Paul Groth, David A. Holland, Sheng Jiang, Jihie Kim, David Koop, Ales Krenek, Timothy McPhillips, Gaurang Mehta, Simon Miles, Dominic Metzger, Steve Munroe, Jim Myers, Beth Plale, Norbert Podhorszki, Varun Ratnakar, Emanuele Santos, Carlos Scheidegger, Karen Schuchardt, Margo Seltzer, Yogesh L. Simmhan, Claudio Silva, Peter Slaughter, Eric Stephan, Robert Stevens, Daniele Turi, Huy Vo, Mike Wilde, Jun Zhao and Yong Zhao (2008). "Special Issue: The First Provenance Challenge". In: *Concurrency and Computation: Practice and Experience* 20.5, pp. 409–418. ISSN: 1532-0634. DOI: 10.1002/cpe.1233. URL: http://dx.doi.org/10.1002/cpe.1233.

Moreau, Luc and Paolo Missier (2013). *PROV-DM: The PROV Data Model.* Tech. rep. W3C Recommendation, W3C, http://www.w3.org/TR/prov-dm/.

Moreau, Luc, Paolo Missier, James Cheney and Stian Soiland-Reyes (2013). *PROV-N: The Provenance Notation.* Tech. rep. W3C Recommendation, W3C, http://www.w3.org/TR/prov-n/.

Morgan, Kevin, Terry Marsden and Jonathan Murdoch (2008). *Worlds of food: Place, power, and provenance in the food chain.* Oxford University Press on Demand.

Nagappan, Meiyappan and Mladen A. Vouk (2008). "A Model for Sharing of Confidential Provenance Information in a Query Based System". In: *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008.* Ed. by Juliana Freire, David Koop and Luc Moreau. Springer Berlin Heidelberg, pp. 62–69. ISBN: 978-3-540-89965-5. DOI: 10.1007/978-3-540-89965-5_8. URL: http://dx.doi.org/10.1007/978-3-540-89965-5_8.

Packer, Heather and Luc Moreau (2015). "Sentence Templating for Explaining Provenance". English. In: *Provenance and Annotation of Data and Processes.* Ed. by Bertram Ludäscher and Beth Plale. Vol. 8628. Lecture Notes in Computer Science. Springer International Publishing, pp. 278–280. ISBN: 978-3-319-16461-8. DOI: 10.1007/978-3-319-16462-5_33. URL: http://dx.doi.org/10.1007/978-3-319-16462-5_33.

Packer, Heather S., Laura Dragan and Luc Moreau (2014). "An auditable reputation service for collective adaptive systems". In: *Social Collective Intelligence: Combining the Powers of Humans and Machines to Build a Smarter Society.* Ed. by Daniele Miorandi, Vincenzo Maltese, Michael Rovatsos, Anton Nijholt and James Stewart. Springer, pp. 159–184. ISBN: 978-3-319-08681-1. DOI: 10.1007/978-3-319-08681-1_8. URL: http://dx.doi.org/10.1007/978-3-319-08681-1_8.

Plump, Detlef (1995). "On termination of graph rewriting". In: *Graph-Theoretic Concepts in Computer Science: 21st International Workshop, WG '95 Aachen, Germany, June 20–22, 1995 Proceedings.* Ed. by Manfred Nagl. Springer Berlin Heidelberg, pp. 88–100. ISBN: 978-3-540-48487-5. DOI: 10.1007/3-540-60618-1_68. URL: http://dx.doi.org/10.1007/3-540-60618-1_68.

Prange, Ulrike and Hartmut Ehrig (2007). "From Algebraic Graph Transformation to Adhesive HLR Categories and Systems". In: *Algebraic Informatics: Second International Conference, CAI 2007, Thessaloniki, Greece, May 21-25, 2007.* Ed. by Symeon Bozapalidis and George Rahonis. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 122–146. ISBN: 978-3-540-75414-5. DOI: 10.1007/978-3-540-75414-5_8. URL: http://dx.doi.org/10.1007/978-3-540-75414-5_8.

Rensink, Arend (2004). "Representing First-Order Logic Using Graphs". In: *Graph Transformations: Second International Conference, ICGT 2004, Rome, Italy, September 28–October 1, 2004. Proceedings.* Ed. by Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce and Grzegorz Rozenberg. Springer Berlin Heidelberg, pp. 319–335. ISBN: 978-3-540-30203-2. DOI: 10.1007/978-3-540-30203-2_23. URL: http://dx.doi.org/10.1007/978-3-540-30203-2_23.

Rensink, Arend, Iovka Boneva, Harmen Kastenberg and Tom Staijen (2010). “User manual for the groove tool set”. In: *Deptt. of Computer Science, University of Twente*.

Rozsnyai, S., A. Slominski and Y. Doganata (2011). “Large-Scale Distributed Storage System for Business Provenance”. In: *2011 IEEE 4th International Conference on Cloud Computing*. CLOUD ’11. IEEE. IEEE Computer Society, pp. 516–524. ISBN: 978-0-7695-4460-1. DOI: 10.1109/CLOUD.2011.28. URL: http://dx.doi.org/10.1109/CLOUD.2011.28.

Sassone, Vladimiro and Pawel Sobocinski (2004). “Congruences for Contextual Graph-Rewriting”. In: *BRICS Report Series* 11.11. ISSN: 1601-5355. DOI: 10.7146/brics.v11i11.21836. URL: http://ojs.statsbiblioteket.dk/index.php/brics/article/view/21836.

Satta, Giorgio and John C. Henderson (1997). “String Transformation Learning”. In: *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*. ACL ’98. Madrid, Spain: Association for Computational Linguistics, pp. 444–451. DOI: 10.3115/976909.979674. URL: http://dx.doi.org/10.3115/976909.979674.

Simmhan, Yogesh, Paul Groth and Luc Moreau (2011). “Special Section: The Third Provenance Challenge on Using the Open Provenance Model for Interoperability”. In: *Future Generation Computer Systems* 27.6, pp. 737–742. ISSN: 0167-739X. DOI: 10.1016/j.future.2010.11.020. URL: http://dx.doi.org/10.1016/j.future.2010.11.020.

Taentzer, Gabriele (2004). “AGG: A graph transformation environment for modeling and validation of software”. In: *Applications of Graph Transformations with Industrial Relevance*. Springer, pp. 446–453.

Taentzer, Gabriele, Michael Goedicke and Torsten Meyer (2000). “Dynamic Change Management by Distributed Graph Transformation: Towards Configurable Distributed Systems”. In: *Theory and Application of Graph Transformations: 6th International Workshop, TAGT’98, Paderborn, Germany, November 16-20, 1998. Selected Papers*. Ed. by Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski and Grzegorz Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 179–193. ISBN: 978-3-540-46464-8. DOI: 10.1007/978-3-540-46464-8_13. URL: http://dx.doi.org/10.1007/978-3-540-46464-8_13.

Varró, Dániel, Márk Asztalos, Dénes Bisztray, Artur Boronat, Duc-Hanh Dang, Rubino Geiß, Joel Greenyer, Pieter Van Gorp, Ole Kniemeyer, Anantha Narayanan, Edgars Rencis and Erhard Weinell (2008). “Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools”. In: *Applications of Graph Transformations with Industrial Relevance: Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers*. Ed. by Andy Schürr, Manfred Nagl and Albert Zündorf. Springer Berlin Heidelberg, pp. 540–565.

ISBN: 978-3-540-89020-1. DOI: 10.1007/978-3-540-89020-1_36. URL: http://dx.doi.org/10.1007/978-3-540-89020-1_36.

Varró, Dániel, Gergely Varró and András Pataricza (2002). "Designing the Automatic Transformation of Visual Languages". In: *Sci. Comput. Program.* 44.2, pp. 205–227. ISSN: 0167-6423. DOI: 10.1016/S0167-6423(02)00039-4. URL: http://dx.doi.org/10.1016/S0167-6423(02)00039-4.

Vázquez-Salceda, Javier, Sergio Álvarez, Tamás Kifor, László Z. Varga, Simon Miles, Luc Moreau and Steven Willmott (2008). "EU PROVENANCE Project: An Open Provenance Architecture for Distributed Applications". In: *Agent Technology and e-Health*. Ed. by Roberta Annicchiarico, Ulises Cortés and Cristina Urdiales. Birkhäuser Basel, pp. 45–63. ISBN: 978-3-7643-8547-7. DOI: 10.1007/978-3-7643-8547-7_4. URL: http://dx.doi.org/10.1007/978-3-7643-8547-7_4.

Zanibbi, Richard, Dorothea Blostein and James R. Cordy (2002). "Recognizing mathematical expressions using tree transformation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.11, pp. 1455–1467. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2002.1046157. URL: http://dx.doi.org/10.1109/TPAMI.2002.1046157.

Zhao, Yong and Shiyong Lu (2008). "A Logic Programming Approach to Scientific Workflow Provenance Querying". In: *Provenance and Annotation of Data and Processes*. Ed. by Juliana Freire, David Koop and Luc Moreau. Berlin, Heidelberg: Springer-Verlag, pp. 31–44. ISBN: 978-3-540-89964-8. DOI: 10.1007/978-3-540-89965-5_5. URL: http://dx.doi.org/10.1007/978-3-540-89965-5_5.