Abstracting PROV provenance graphs: A validity-preserving approach

Missier, P., Bryans, J., Gamble, C. & Curcin, V.

Original citation & hyperlink:
Missier, P, Bryans, J, Gamble, C & Curcin, V 2020, 'Abstracting PROV provenance graphs: A validity-preserving approach', Future Generation Computer Systems, vol. 111, pp. 352-367.
https://dx.doi.org/10.1016/j.future.2020.05.015

DOI 10.1016/j.future.2020.05.015
ISSN 0167-739X

Publisher: Elsevier

Abstracting PROV provenance graphs: a validity-preserving approach

P. Missier[a,*], J. Bryans[b,*], C. Gamble[a], V. Curcin[c]

[a]*School of Computing, Newcastle University, UK*
[b]*Institute for Future Transport and Cities, Coventry University, UK*
[c]*Kings College, London, UK*

Abstract

Data provenance is a structured form of metadata designed to record the activities and datasets involved in data production, as well as their dependency relationships. The PROV data model, released by the W3C in 2013, defines a schema and constraints that together provide a structural and semantic foundation for provenance. This enables the interoperable exchange of provenance between data producers and consumers. When the provenance content is sensitive and subject to disclosure restrictions, however, a way of hiding parts of the provenance in a principled way before communicating it to certain parties is required. In this paper we present a provenance abstraction operator that achieves this goal. It maps a graphical representation of a PROV document PG_1 to a new abstract version PG_2, ensuring that (i) PG_2 is a valid PROV graph, and (ii) the dependencies that appear in PG_2 are *justified* by those that appear in PG_1. These two properties ensure that further abstraction of abstract PROV graphs is possible. A guiding principle of the work is that of minimum damage: the resultant graph is altered as little as possible, while ensuring that the two properties are maintained. The operator developed is implemented as part of a user tool, described in a separate paper, that lets owners of sensitive provenance information control the abstraction by specifying an abstraction policy.

Keywords: Provenance, Provenance metadata, provenance abstraction

1. Introduction

The provenance of data is a form of structured metadata that records the processes involved in data production. In addition to containing references to data generation or transformation processes, a *provenance trace* typically includes input or intermediate data products as well as references to *agents*, that

[*] Corresponding Author

Email addresses: Paolo.Missier@newcastle.ac.uk (P. Missier), Jeremy.Bryans@coventry.ac.uk (J. Bryans), Carl.Gamble@newcastle.ac.uk (C. Gamble), Vasa.Curcin@kcl.ac.uk (V. Curcin)

is the humans or software systems who were responsible for those processes. In multi-party collaboration settings that involve data sharing, as well as in third party auditing of data and processes, there is a broad expectation that shipping the available provenance to collaborators, or more generally publishing it along with the data, may help data consumers, including auditors, form judgements regarding the reliability of the data itself.

Offering to disclose the provenance of data as evidential basis for establishing data quality and reliability is particularly important when data products are exchanged as part of transactions that involve parties with limited mutual trust. This is the case for instance of *dynamic coalitions* [7]. These are ad hoc collaborative partnerships that are created to pursue a common goal, in scenarios such as multi-agency emergency/threat responses, as well as the exchange of intelligence information. Despite the need to share data of a possibly sensitive nature, these coalitions are characterized by a lack of established interaction protocols and by limited trust amongst the partners. This situation creates a tension between data providers and consumers, when it comes to negotiating the level of detail of the provenance that providers are prepared to offer to consumers. On the one hand, consumers will require as much provenance detail as possible, to use as a basis for establishing data credibility. Data providers, on the other hand, will be reticent to offer detailed provenance traces, because those may contain sensitive information regarding their own internal processes as well as any proprietary data used by those processes. In fact, the provenance of a data product will typically contain more sensitive information than the data product itself.

In this paper we propose to resolve such tension by introducing an operator to achieve selective disclosure of provenance information.

1.1. Motivating scenario: provenance of intelligence information

To appreciate how such tension may arise, consider a scenario where a public agency PA wants to buy intelligence reports, say about potential threats to the public, from an intelligence provider, IP. The trust model is governed by a desire on the side of the PA for risk mitigation, and the environment is such that PA is not prepared to fully trust and act upon the information provided by the IP without performing some risk assessment on the information. The key issue here is not that the PA does not trust the IP (it is assumed that sufficient trust exists to contemplate purchasing IP's intelligence reports), but that it must come to an independent assessment of the trustworthiness of the intelligence reports which have been provided. Under the assumption of limited trust by PA in IP's information, PA must find some way to mitigate the risk of acting upon information provided by IP, which is potentially unreliable. At the same time, IP has a business incentive to supply PA with additional evidence that facilitates PA's risk assessment and thus increases the chance of a successful transaction.

The key assumption that motivates our work is that the provenance of each intelligence report is relevant in contributing, at least in part, the required evidence. However, simply disclosing all provenance may not be in IP's business interests, and so IP must find a way to strike a balance between supporting PA's risk assessment and protecting its own business interests.

We illustrate this with a fictional but realistic example of provenance for an intelligence report, shown in Fig. 1. Provenance can be visually depicted as a digraph whose nodes represent either *entities* (ovals in the figure), i.e., data, documents, etc., *activities* (rectangles), which represent the execution of some process over a period of time, or *agents* (pentagons), which represent humans or computing systems. The edges represent various types of directed relationships, the most common being "activity a used entity e", "entity e was generated by activity a", "activity a was associated with agent ag" (i.e., ag was responsible for a), and more. It is also possible to annotate each of the nodes using properties, for instance to qualify the kind of data and activities involved in the process execution. We omit annotations from our examples for readability, and because, within this work, they are not handled in any special way.

Formally, such a graph is a depiction of a PROV document, which in turn conforms to the W3C PROV data model [23], introduced in Section 3.1. We will use the graph representation of provenance throughout the paper, as it facilitates reasoning about the mechanisms for provenance abstraction, which are at the core of our work. The graph layout in Figs. 1, 2 and 3 is such that the process execution flows from top to bottom, i.e, the top entities represent initial inputs, while the outputs are at the bottom.

This provenance graph depicts a process of intelligence report generation, which is initiated by a request by PA. The process identifies target users from the request and acquires further information about those users, both on Twitter (`Twitter_query`) and from a proprietary database, `IP_users_profile_DB`. The results are fed to two analytics sub-processes, each of which generates a report. Note that `analytics_1` only uses Twitter data, while `analytics_2` also uses query results of the proprietary database. A `consolidate` step follows, which produces a master `IP_report`. This is checked, to validate and possibly also to remove sensitive information, before the final `PA_report` is generated for the customer. Notice that various agents are specified as being responsible for some of the steps, along with their chain of responsibility, i.e., `Alice` acts as `Bob`'s delegate, who in turn reports to `Karen`, along with `Charlie`.

As we can see, the full-fledged provenance graph contains information about IP's internal business processes, including the use of a proprietary database, which IP may consider privileged. It is therefore realistic to imagine that IP may want to hide some of those elements. Using our selective disclosure model, IP marks the sensitive elements, in this case the `redact` activity as well as the references to the two analytics processes. Note that doing so does not require any knowledge of the graph topology, rather only of the nodes (either activities, entities, or agents) that are to be abstracted out.

The PROV document represented in Fig. 1 is *valid*, in the sense that it satisfies all the constraints specified as part of the PROV standard [10]. As we will see in the rest of the paper, replacing the three selected nodes with a single abstract node (an activity in this case) while preserving the validity

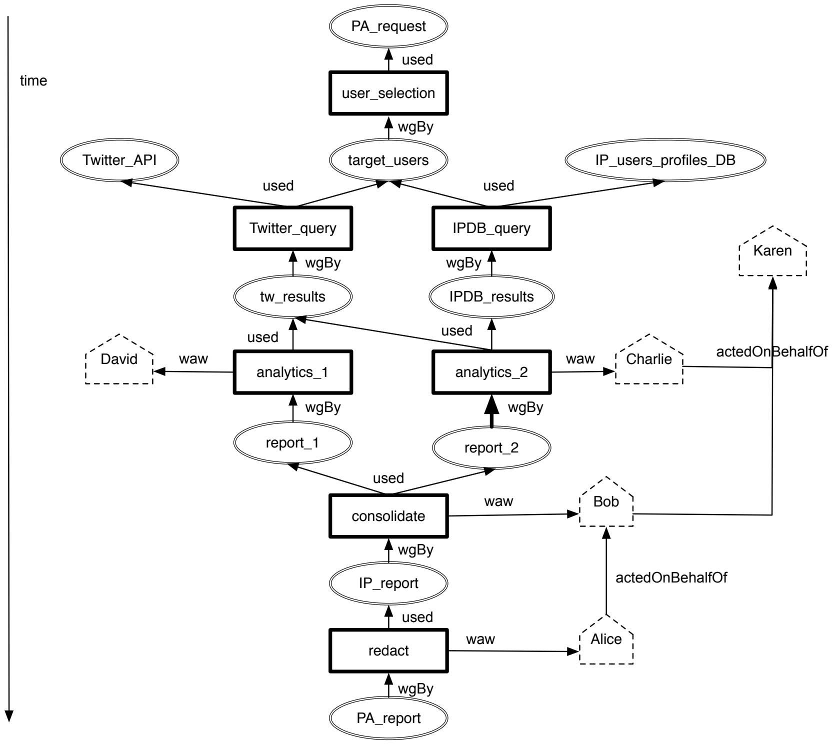


Figure 1: Example provenance graph depicting the generation of an intelligence report.

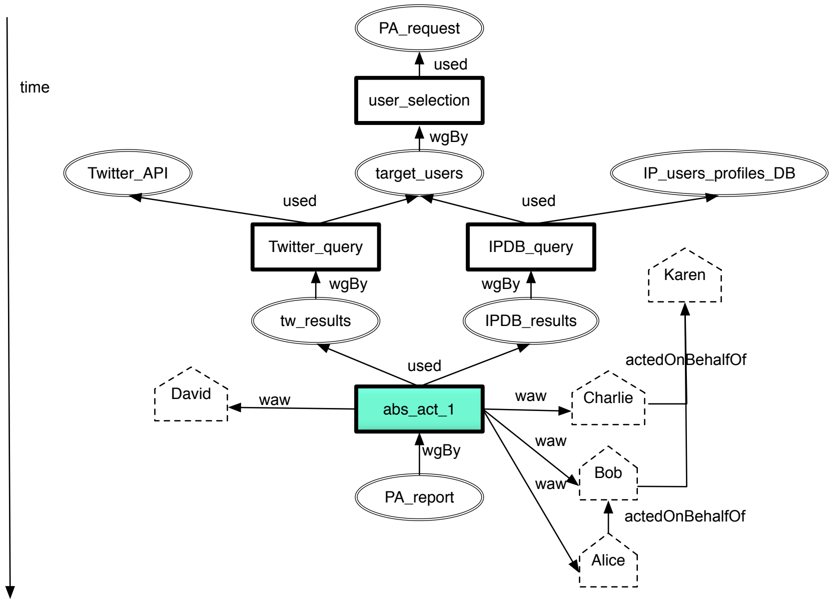


Figure 2: The result of abstracting out selected nodes `redact`, `analytics_1`, and `analytics_2` from the graph in Fig. 1.

of the document, requires that all other nodes that lie on the directed paths that connect these nodes are also removed. In this example, the result of such abstraction operation is shown in Fig. 2.

By construction, this is a new valid PROV graph. It is therefore possible to further abstract out some of its nodes. For example, IP may also decide that mentioning the use of a proprietary data source is inappropriate. Nodes `IP_users_profiles_DB` and `IPDB_query` are therefore abstracted out. As these are both entities, the new abstract node is also an entity, as shown in Fig. 3. Note that, to PA, while still informative, the report now appears as if it had been generated from its initial request using Twitter as a data source, and without reference to specific analytics algorithms.

The mechanisms by which the data and provenance owner selects the nodes to be abstracted are not discussed in this paper, however a policy-based model is described in detail in our previous work [20]. Briefly, the idea, which makes use of the Bell-Lapadula model [2], is that the owner assigns a sensitivity value to each node, and nodes are selected to be abstracted out based on a specific recipient's clearance level. Thus, different recipients will potentially receive different abstract versions of the same graph. Note also that forcefully removing nodes that were not marked for abstraction has implications, too, as some of those non-sensitive nodes may have had evidential value that is now lost. To model this problem we associate a utility value to each node, and then compute the *residual utility* of the abstracted graph. The paper cited above [20] provides

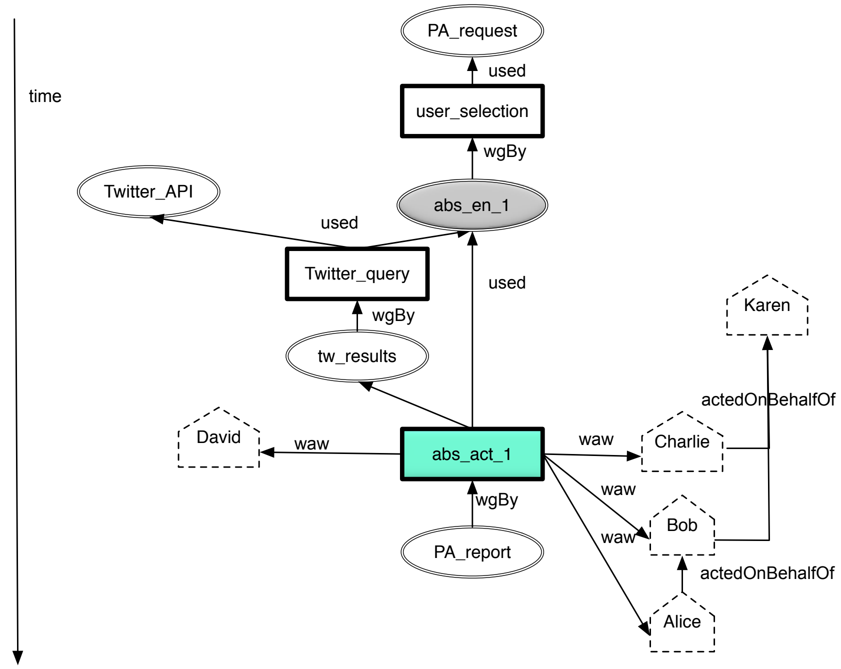


Figure 3: The result of further abstracting out IP_users_profiles_DB and IPDB_query after the first abstraction step (Fig. 2).

further details.

We observe that removal of information from a provenance graph could be achieved in a number of other ways. For example, one could simply remove the labels as well as the annotations from individual nodes and relationships, i.e., anonymize part of the graph. Doing so, however, does not hide any of the structure of the process of data production. One could further remove nodes and relationships or indeed entire sub-graphs. The new graph will be disconnected, however, making it difficult to reconstruct the lineage of the end data product, that is, the sequence of data derivations from the initial inputs to the outcome of the process.

Instead, in our approach the selected nodes are replaced with a new abstract node, which is then "re-wired" to the remaining original graph. This has the effect of hiding parts of the process structure as it was represented in the original provenance, while maintaining connectivity. One can still query the lineage, but some of the provenance elements returned by the query will now be an abstraction of the actual data production process.

The main challenge addressed in this paper is to guarantee that abstraction produces PROV-compliant graphs, maintaining the interoperability guarantees provided from having standardized PROV and ensuring that the results can be consumed by standard PROV tools.

1.2. Contributions

In this paper we develop a model and algorithm for performing abstraction over PROV graphs, providing the theoretical underpinning to ensure that the abstraction process satisfies a number of properties. For the model development we restrict our attention almost completely to provenance graphs that contain only activities and entities, and the relevant relationships *wgBy* and *used*. Work on extending the model to agents is less mature, and we have chosen to exclude consideration of agents in this paper. The *wasInfluencedBy* relation is a super-property of both *wgBy* and *used* and we explore the advantages of incorporating it in Section 4.5.

Our main contribution is the formal functional definition of a provenance abstraction operator (*Group*) that rewrites a PROV graph PG into a new graph PG', by (a) mapping a set V_{gr} of nodes (for "vertex in a group") in PG to a new abstract node v_{new}, and (b) mapping each relationship involving elements of V_{gr} (nodes) to a new relationship involving v_{new} in PG'. The set V_{gr} is chosen by the user of the abstraction operator as the set of nodes she wishes to hide, and the graph rewriting operator *Group* is defined in three steps using three subordinate operators, detailed in Section 4.

Our grouping operator ensures two formal properties of PG': firstly, validity: if PG is a valid PROV graph, that is, it conforms to the PROV data model [23], then PG' is also a valid PROV graph. Secondly, no unjustified dependencies are introduced into PG': a relationship involving v_{new} is only created as a result of a mapping from an existing relationship involving elements of V_{gr}. Strictly, if two nodes are not *directly* related in PG, we guarantee that they are not directly related in PG'. Note that new indirect dependencies between two nodes in PG', manifested as new paths in the graph, may be introduced, however we argue that these are always justified by the topology of the underlying graph PG.

A guiding principle throughout is that of minimal damage. We require that the grouping operator inflicts minimum unnecessary damage on the graph, while meeting the first two constraints.

It is important to observe here that the task that we set ourselves is to develop this abstraction operator given an *arbitrary* set of nodes in the graph that must be abstracted. For example, we do not allow ourselves the luxury of partitioning the set before the *Group* operator is applied. Here we wish to develop a pure PROV graph abstraction operator that can be applied independently of, or in combination with, other solutions.

Furthermore, by making the abstraction operator closed with respect to the set of valid PROV graphs, abstraction can be naturally composed, i.e., using the *Group* operator one can abstract PG' into some PG'' as we have shown in the earlier example.

Finally, note also that PG' itself has also an associated provenance graph, that is, a record of the provenance abstraction process as it was applied to PG. PROV provides a syntactic facility to maintain the association between a provenance graph and its own provenance, namely using the "provenance of provenance" mechanism (i.e., bundles [23]).

2. Related Work

Multiple strands of research relate to our work. These include creating views over a provenance graph to reduce its complexity, redacting a graph (including non-provenance graphs, such as a social media network) to obscure or remove some its sensitive elements, and summarising a collection of graphs. We also mention graph access control and anonymisation techniques, which are more peripheral to our work. A more comprehensive recent survey on approaches for *provenance sanitisation* is available [11].

2.1. Provenance views and graph redaction

Graph redaction is used in [5] to rewrite a graph where particular nodes, node properties, or relationship instances are sensitive. The technique relies on the notion of *surrogates*, which are less sensitive versions of the graph where some information has been either removed or replaced, depending on the clearance level of the user who has access to the graph. If a measure of utility is associated with elements of the graph, removing or redacting graph elements may reduce the residual utility, and may also result in loss of connectivity. The paper describes techniques for generating surrogates that achieve a desired protection level, while maximizing graph connectivity and minimising utility loss. The technique applies to generic graphs, for instance a social media network, and is also demonstrated on a provenance graph.

Similar elements, namely a technique for graph editing, a user clearance policy based on nodes sensitivity and user clearance levels, and a quantitative measure of utility, are indeed at the core of our own ProvAbs system [20]. The current paper extends ProvAbs, but policies and utility maximisation are not discussed as the focus is on proving validity-preservation properties of the redaction operator (i.e., grouping). This is indeed the main distinctive feature that sets our work apart from [5], where there is no such notion of a *valid* graph.

Also close to our abstraction model, both in motivation and in its technical approach, is the ProPub system [15], which computes views over provenance graphs that are suitable for publication by meeting certain privacy requirements. In ProPub, users specify edit operations on a graph, such as anonymizing, abstracting, and hiding certain parts of it. The operations are specified as logic rules, and are interpreted natively by the Datalog-based prototype implementation. ProPub adopts an "apply–detect–repair" approach, whereby user rules are applied to the graph first, then consistency violations that may occur in the resulting new graph are detected, and a final set of edits are applied to the graph in order to repair such violations. In some cases, this causes nodes that the user wanted removed to be reintroduced, and it is not always possible to satisfy all rules. In contrast, our grouping involves more simply a set of nodes to be abstracted (but note that anonymization is a particular case, when the group contains a single element). In return for this simplicity in the specification of the nodes to be grouped, our method always produces a valid abstract graph while ensuring that the nodes specified in the policy are removed.

Techniques for *provenance redaction* that are based on graph grammars in combination with a redaction policy language are discussed in [9], where they are deployed to edit provenance that is expressed using the Open Provenance Model [22] (a precursor to PROV). Although the authors claim that the redaction operators ensure that specific relationships are preserved, this critical issue is not addressed formally in the paper, i.e., with reference to the OPM semantics. In contrast, the formal schema and set of constraints that come with PROV [23, 10] provide the necessary grounding for reasoning about the validity-preservation properties of the editing operations.

The concept of *sound workflow views* proposed in [19] is closely related to our notion of *justified relations* (cf. Section 4.3), by which we impose that only certain new relations can be added to a provenance graph as a consequence of abstraction by grouping. A view over a workflow that consists of multiple interconnected tasks, is obtained by forming groups of such tasks. The groups are connected based on the underlying original dependencies, to form a new, higher-level workflow. By doing so, however, one may create new paths between tasks that were previously not reachable from each other. As a result, lineage queries that operate on the workflow view, and essentially compute node reachability by transitive closure, return spurious results. A *sound* view is one where the groupings do not produce any spurious lineage results. As the number of possible groupings is combinatorial, the paper explores heuristics for detecting and generating sound views.

The problem does not directly apply to our approach to abstraction, essentially because in our case the abstraction is performed *on the provenance graph itself*, rather than on the workflow whose execution the provenance graph represents. It is computed over a graph using a user-chosen set of nodes, with the purpose of obscuring information. In particular, while it is true that new paths are created, it is easy to see that these always involve both a concrete and an abstract node either at the source or at the destination, that is, they involve new relations (see for instance the example in Fig. 8 on page 22). As these relations on the abstract graph are justified (see Section 4.3), we conclude, intuitively, that the new paths encode dependencies that are similarly justified by the underlying graph.

Related to [19] is Zoom [4], from some of the same authors. In Zoom, the main assumption is that the graph is a trace that specifically represents the execution of a dataflow. This is a common occurrence in e-science, where workflows that follow the dataflow model are a popular high level programming paradigm. In this setting views over provenance are effectively a form of abstraction and are computed based on the user's indication of which workflow modules (tasks) are relevant, or perhaps based on which modules the user has access to. Thus, key to this approach is knowledge of the underlying workflow structure, which is used to specify the nodes in the graphs to be abstracted. This sets Zoom apart from our work, which instead investigates the properties of a grouping operator *independently of the origins of the trace to which it is applied.*

Also specific to workflow-generated provenance, and thus too narrow in scope for our purposes, is a strand of research that investigates the problem of preserving the privacy of functions used in workflows, when a large number of input/output pairs for those functions is revealed through the provenance traces of multiple workflow executions. This work on *module privacy* [14, 13, 12] is concerned with protecting the semantics of workflow modules. It applies anonymization techniques specifically to provenance graphs and is again centred around a workflow-specific form of provenance and is thus also peripheral to our interest.

2.2. Provenance Access Control

Most of the work on protecting access to sensitive provenance includes policy models that extend traditional data Role-Based Access Control (RBAC), with a distinction made between PBAC (Provenance-Based Access Control) and PAC (Provenance Access Control). PBAC is about policy to specify access rights to data objects based on their provenance. An example, from [24], is a rule of the form "only the student submitter can access the graded homework object". This rule can be enforced by looking for a dependency path in a provenance graph, whereby a given homework is attributed to a specific student (i.e., relation *IsAuthoredBy* in the Open Provenance Model). This assumes that the object's attribution is explicit in the provenance graph. It is less clear how such a rule would be evaluated when the provenance is incomplete with respect to such attribution dependency, however.

PAC, or how to enforce access control on parts of a provenance graph, is more directly relevant to our work. An analysis of some of the challenges associated with secure provenance exchange can be found in [6], where examples are presented that show how the provenance of data can be more sensitive than the data itself. Another position paper [17] describes the challenges associated with the exchange of provenance across multiple partners, in a setting where forgery of provenance by malicious users is a possibility, and where users may collude to reveal sensitive provenance to others. These are all common and complex security problems. Unfortunately, the paper stops short of providing any hints at technical solutions, and indeed it is not clear how these problems are specific to provenance, as opposed to data sharing in general.

A concrete specification of an access control system or provenance [8] consists of a XACML-based policy language, in which path queries are used to specify target elements of the graph, as well as an implementation architecture and a prototype.

2.3. Summarisation of provenance graphs

A loosely related strand of research in this area aims at summarising a collection of provenance graphs by constructing a "super-graph" that captures the common features across a collection of similar graphs, such as those that are produced by repeating execution of a process with different inputs and parameters. This has been addressed with an aim to improve provenance queries [16], as well as to provide a compact but approximate representation of provenance at the possible cost of information loss [1]. This work is only peripherally relevant here, as our approach only operates on one graph at a time.

In [21] a mechanism is proposed to automatically construct aggregations from a single PROV graph. This relies on the concept of *provenance types*, which are fixed-length paths in the graph that occur more than once. The aggregation is defined as a mapping from provenance nodes to the provenance types, and there is a way to connect these types into a new PROV-like graph, by similarly mapping the graph edges to new weighted edges. The result is a new graph that is meant to capture the "essence" of a fine-grained set of provenance statements by observing regularities in the original graph. This is substantially different from our approach, namely (i) the choice of nodes to aggregate is driven by the discovery of provenance types, which is entirely driven by graph topology and not by a user choice, and (ii) there is no intent to generate *valid* PROV graphs, which is instead the main goal of our transformation. Thus, the approach is not suitable to support policy-driven (or other user-oriented) selective disclosure, and the aggregation operation produces a graph that may violate PROV constraints.

2.4. General graph anonymization

For completeness, we briefly mention more general techniques for graph editing, largely motivated by the need to preserve privacy in social network data. This body of work, which is not specific to provenance, extends the well-known data anonymization framework developed for relational data to graph data structures [25, 3, 18]. The main idea is to randomly remove arcs between two nodes and replace them with new ones. As arcs in PROV graphs represent

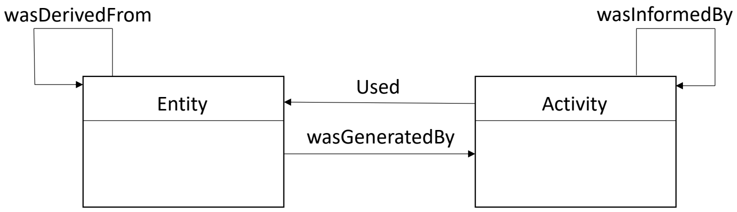


Figure 4: Core elements of the PROV model, adapted from [23].

relationships with a given semantics, this approach generally results in false dependencies being created in the edited graph, and is therefore not viable. The main value of this body of work in this setting, as summarised in [26], is to ensure that various forms of anonymization are provably robust to attacks from adversaries who can potentially leverage their partial information about fragments of the graph, to infer additional knowledge. In this paper we do not discuss the robustness of abstraction by grouping, indeed we do not consider any specific threats, and so the challenge of preventing the reconstruction of the abstracted fragments of provenance graphs is left for future work.

3. Background

3.1. Core PROV model

We now introduce the core elements of the PROV model, which forms the basis for the grouping operator. We maintain a dual view of provenance, both as a relational model (with binary relations) and as a graph model. Viewed as a relational model, PROV includes the three types of elements: Entities (En), Activities (Act), and Agents. However in this paper we restrict our attention to entities and activities and the relations between them. Agents have proved difficult to incorporate into our framework, but the results we have with entities and activities are worth recording. In line with the description in [23] (Section 2), PROV is defined by the following core relations, with common abbreviations in brackets.

$$\begin{array}{rcl} \mathit{Used}\ (\mathit{used}) & \subseteq & \mathit{Act} \times \mathit{En} \\ \mathit{WasGeneratedBy}\ (\mathit{wgBy}) & \subseteq & \mathit{En} \times \mathit{Act} \\ \mathit{WasDerivedFrom}\ (\mathit{wasDerivedFrom}) & \subseteq & \mathit{En} \times \mathit{En} \\ \mathit{WasInformedBy}\ (\mathit{wasInformedBy}) & \subseteq & \mathit{Act} \times \mathit{Act} \end{array}$$

These are summarized in Fig. 4.

We note that, when considering graphs containing the relations *wasInformedBy* and *wasDerivedFrom*, we can replace those relations with patterns involving only

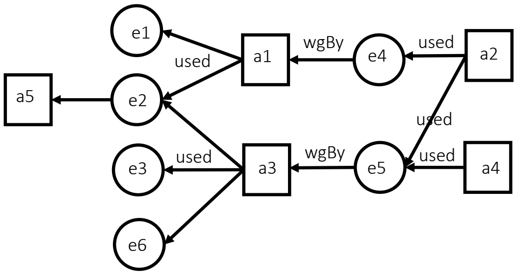


Figure 5: $PG_{gu/ea}$ provenance graph used as a running example to illustrate abstraction by grouping.

used and *wgBy* using Inference 5 (communication-generation-use-inference)[1] and Inference 11 (derivation-generation-use-inference)[2] in the PROV-CONSTRAINTS document [10]. This replacement has to be handled with care, however. In the case of *wasInformedBy* and Inference 5, PROV-CONSTRAINTS contains a corresponding inference (Inference 6, generation-communication-use-inference)[3] that allows the reverse replacement, but there is no such reverse replacement for *wasDerivedFrom* and Inference 11, meaning that the original graph cannot be inferred back again, and in general the use of Inference 11 loses information.

As the relations that we consider (*used* and *wgBy*) are binary, we can view a provenance graph D as a bipartite digraph $G = (V, E)$, where $V = En \cup Act$, and each relation instance maps to a labelled directed edge. By convention, we orient these edges from right to left, to denote that the relation "points back to the past". Thus: $a \xleftarrow{wgBy} e \in E$ iff $wgBy(e, a) \in D$, and $e \xleftarrow{used} a \in E$ iff $used(a, e) \in D$. We denote the label associated to edge (v_i, v_j) as $label(v_i, v_j)$.

We denote a generic such graph by $PG_{gu/ea}$, to indicate that it only contains En and Act nodes, and *wgBy* and *used* edges. In the rest of this paper we will equate provenance documents with their graphical models. Fig. 5 portrays a simple $PG_{gu/ea}$ graph that we will be using as a running example.

3.2. Events in $PG_{gu/ea}$

Central to PROV is the notion that provenance is marked by events. A partial order is defined over events, so that it may or may not be possible to establish whether or not one event precedes another. Events occur instantaneously, and they mark the lifetime boundaries of Entities (generation, invalidation), Activities (start, end), and Agents (start, end), as well as some of the interactions amongst those elements. These include the generation and usage of an entity

[1] https://www.w3.org/TR/prov-constraints/#communication-generation-use-inference

[2] https://www.w3.org/TR/prov-constraints/#derivation-generation-use-inference

[3] https://www.w3.org/TR/prov-constraints/#generation-communication-use-inference

by an activity, attribution of an entity to an agent, and more. More specifically, the PROV-CONSTRAINTS document [10] defines the following types of events (quoted verbatim from Section 2.2):

- **An activity start event** is the instantaneous event that marks the instant an activity starts.
- **An activity end event** is the instantaneous event that marks the instant an activity ends.
- **An entity generation event** is the instantaneous event that marks the final instant of an entity's creation timespan, after which it is available for use. The entity did not exist before this event.
- **An entity usage event** is the instantaneous event that marks the first instant of an entity's consumption timespan by an activity. The described usage had not started before this instant, although the activity could potentially have used the same entity at a different time.

We denote the start and end events of an activity a as $start(a)$, $end(a)$, respectively, and we write $ev(wgBy(e,a))$ and $ev(used(a,e))$ to refer to events associated to instances $wgBy(e,a)$ and $used(a,e))$ of entity generation and usage, respectively.

As an example, in the graph of Fig. 5 the generation relation $wgBy(e_4, a_1)$ has an associated generation event $ev(wgBy(e_4, a_1))$, whilst a_1 has start and / or end events, written $start(a_1)$ and $end(a_1)$, respectively. Similarly, usage of e_4 by a_2 is marked by event $ev(used(a_2, e_4))$.

3.3. Constraints and valid $PG_{gu/ea}$ graphs

Validity of a PROV document is defined in terms of a set of constraints, as stated in the PROV-CONSTRAINTS document [10]. For instance, Constraint 55 ("entity-activity-disjoint") states that the Entities and Activities are disjoint:

$$En \cap Act = \emptyset$$

Thus, a provenance graph PG in which both (1) $a_1 \xleftarrow{used} e_1$ and (2) $e_1 \xleftarrow{used} a_1$ cannot be valid, because by definition (1) entails $e_1 \in En$, $a_1 \in Act$, while (2) entails $a_1 \in En$, $e_1 \in Act$, violating the constraint. We refer to this constraint in the sequel as C1.

In this paper we are mainly concerned with temporal constraints which apply to $PG_{gu/ea}$ instances and determine admissible partial orderings over events. More precisely, let $\preceq \subset Ev \times Ev$ denote a pre-order relation[4] on the set Ev of events associated to instances of activities and relations as defined above.

[4] Recall that a pre-order is a binary relation with reflexivity and transitivity, but no symmetry or anti-symmetry.

For a PROV document to be valid, $\preceq$ is required to satisfy the following set of constraints[5](using the original numbering in [10]):

- **C2: generation-generation-ordering (Constraint 39):** If an entity is generated by more than one activity, then the generation events must all be simultaneous.

  Let $gen_1 = ev(wgBy(e, a_1))$, $gen_2 = ev(wgBy(e, a_2)) \in PG$. Then

  $$gen_1 \preceq gen_2, \quad gen_2 \preceq gen_1$$

  must hold.

- **C3: generation-precedes-usage(Constraint 37):** A generation event for an entity must precede any usage event for that entity. For any $a \in Act$ such that $used(a, e) \in PG$,

  $$ev(wgBy(e, a)) \preceq ev(used(a, e))$$

  must hold.

- **C4: usage-within-activity (Constraint 33):** Any usage of e by a cannot precede the start of a and must precede the end of a. For any $e \in En, a \in Act$ such that $used(a, e) \in PG$:

  $$start(a) \preceq ev(used(a, e)) \preceq end(a)$$

- **C5: generation-within-activity (Constraint 34):** The generation of e by a cannot precede the start of a and must precede the end of a. Let $wgBy(e, a) \in PG$:

  $$start(a) \preceq ev(wgBy(e, a)) \preceq end(a)$$

Additional relevant constraints state that multiple start (resp. end) events must all be simultaneous, and that the start event of an activity must precede the end event for that activity.

Definition 1 (Validity). *A graph $G \in PG_{gu/ea}$ is valid iff it satisfies constraints C1-C5.*

In the next section we present the *Group* operator. We will return to the constraints above in Section 5, where we propose a new set of *abstract events* that are derived from these and apply to the abstract graphs.

[5] For simplicity, *entity invalidation constraints* are not considered.

4. Grouping Provenance graph nodes

As mentioned in Section 1.2, our goal is to define a graph editing operator that selectively removes information from a graph $G \in PG_{gu/ea}$, yielding a new graph $G' \in PG_{gu/ea}$. We require the final operator to remove all the chosen nodes, and replace them by a single abstract node. As discussed in Section 1.1, this may be to avoid inadvertently revealing any internal business processes or details about the chain of command. Furthermore, we wish the modified graph to be a valid PROV graph, in the sense that the relations and constraints from Section 3 are respected. This will allow the output to be consumed by any system capable of reading the PROV language.

In this section, we focus exclusively on the definition of the *Group* graph transformation operator as the prime way to achieve abstraction over provenance graphs. *Group* takes a graph $G \in PG_{gu/ea}$ with nodes V and edges E and a subset $V_{gr} \subset V$ of its nodes that the user wishes to hide and produces a modified graph $G' \in PG_{gu/ea}$. The nodes in V_{gr} are "grouped" together and replaced by a new single node. The *Group* operator has the following signature, where $\mathbb{P}(V)$ is the powerset of the nodes V of the graph G :

$$Group : PG_{gu/ea} \times \mathbb{P}(V) \rightarrow PG_{gu/ea} \tag{1}$$

As the operator is closed under composition, further abstraction can be achieved by repeated grouping, either on multiple disjoint sets V_{gr}, or on sets that include abstract nodes (abstraction of abstraction).

We take a functional approach to the definition of *Group*, by defining three subordinate functions and then defining *Group* as the functional composition of these subordinate functions.

In Section 4.1 we make the simplifying assumption that the set of nodes to be removed are all of the same type (either all *En* or all *Act*). We refer to this as *homogeneous grouping* and the replacement node is implicitly of the same type as the nodes selected to be removed. In Section 4.2 we remove this assumption, so the group of nodes initially selected to be removed can contain both entities and activities. A consequence of this is that the type of the replacement node is no longer implicit, and must be explicitly given. This leads to two variants of the operator, depending on which type (*En* or *Act*) is chosen for the final replacement node. In Section 4.2 we also identify an issue arising from simultaneous generation, and show how it may be circumvented, leading to a further variant of *Group*.

In Section 4.3 we show that the relations in the abstract graph produced by any of the *Group* operators are *justified* by the original graph. In Section 4.4 we discuss the complexity of the operator, and in Section 4.5 we explore alternative approaches to abstraction.

4.1. Closure and homogeneous grouping

In this subsection we present the homogeneous version of the *Group* operator that will replace a selected set of nodes with an abstracted one, working under the simplifying assumption that all the nodes selected to be removed are of the same type. As stated previously, we also wish to ensure that the final abstracted graph is a valid PROV graph.

Let the set of nodes to be replaced in a graph G be V_{gr}. An issue, pointed out in the description of the ProPub system [15], mentioned earlier, is that removing V_{gr} and replacing it with a single node can lead to cycles in the modified graph. Intuitively, this occurs when V_{gr} is not "convex", that is, there are paths in the graph that lead out of V_{gr} and then back in again.

This suggests the introduction of a preliminary closure operation $\mathit{pclos}()$, which ensures acyclicity by capturing and including all the nodes on paths between nodes in the initial group. It is defined as follows.

Definition 2 (Path Closure). *Let $G = (V, E) \in PG_{gu/ea}$ be a provenance graph, and let $V_{gr} \subset V$. For each pair $v_i, v_j \in V_{gr}$ such that there are one or more directed paths $v_i \rightsquigarrow v_j$ in G, let $V_{ij} \subset V$ be the set of all nodes in all paths $v_i \rightsquigarrow v_j$. The Path Closure of V_{gr} in G is*

$$\mathit{pclos}(V_{gr}, G) = \bigcup_{v_i, v_j \in V_{gr}} V_{ij}$$

Fig. 6 illustrates $\mathit{pclos}()$ in the transition from Fig. 6(a) to Fig. 6(b). $\mathit{pclos}()$ is performed on the set $\{e_1, e_3, e_4, e_5\}, G)$, resulting in the set $\{e_1, e_3, e_4, e_5, a_1, a_3\}$ in Fig. 6(b). We assume, for the moment, that nodes in $\mathit{pclos}(V_{gr}, G)$ induce a single connected subgraph under G.

However, while the application of $\mathit{pclos}()$ ensures that the group to be abstracted is free from cycles, simply replacing the shaded nodes in Fig. 6(b) with a single node e' is not sufficient. This is because the resulting graph would no longer be bipartite, since the new edges $e' \rightarrow e_2$ and $e' \rightarrow e_6$ would connect nodes of the same type.

To ensure that the eventual node replacement preserves the type-consistency of graph, we also require all the set boundary nodes (nodes in the defined set connected to nodes outside the set) to be of the same type. In the example in Fig. 6 we must extend the shaded set in Fig. 6(b) to include e-nodes e_2, e_6, as shown in Fig. 6(c). We define a second operator $\mathit{extend}()$ to do this.

Formally, the application of $\mathit{extend}()$ to a set $V_{gr} \subset V$ relative to type $t \in \{En, Act\}$ will be V_{gr} augmented with all its adjacent nodes, in either direction, of type t. In this way all boundary nodes of our group will have type t.

Definition 3 (*extend*). *Let $G = (V, E) \in PG_{gu/ea}$, $t \in \{En, Act\}$. v_s and v_d are the source and destination nodes of a relationship.*

$$\begin{aligned}
&\mathit{extend}(V_{gr}, G, t) = \\
&\quad V_{gr} \cup \\
&\quad \{v_d \mid (v_d \leftarrow v_s) \in E \wedge v_s \in V_{gr} \wedge v_d \notin V_{gr} \wedge \mathit{type}(v_d) = t\} \cup \\
&\quad \{v_s \mid (v_d \leftarrow v_s) \in E \wedge v_s \notin V_{gr} \wedge v_d \in V_{gr} \wedge \mathit{type}(v_s) = t\}
\end{aligned}$$

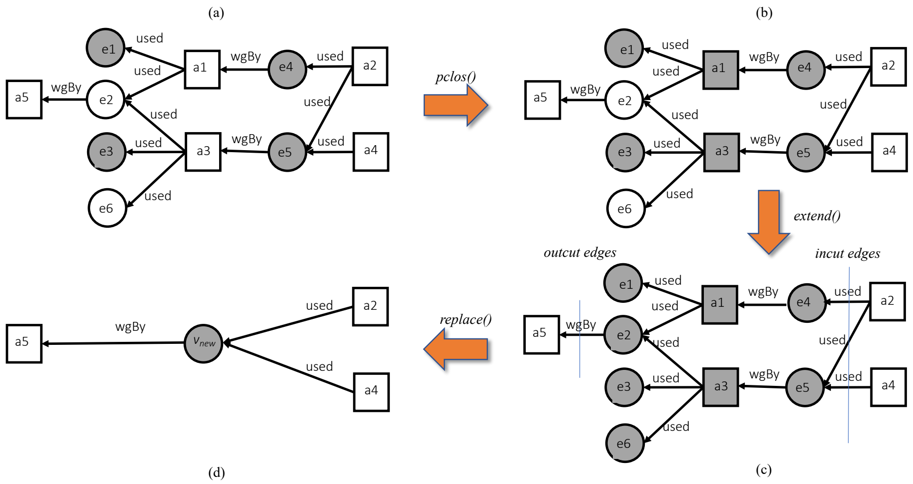


Figure 6: Path closure and replacement with extension to a set of entity nodes.

In the example in Fig. 6 nodes e_2 and e_6 are now included, and

$$extend(\{e_1, e_3, e_4, e_5, a_1, a_3\}, G, En) = \{e_1, e_3, e_4, e_5, a_1, a_3, e_2, e_6\}$$

as shaded in Fig. 6(c). To see that this is a minimal extension, in the sense that nodes are only included if necessary, observe that a new node is only included if i) it is of type t and ii) it is adjacent to a node already in the set. Finally, we can replace the collected nodes with a new abstract node, as shown in Fig. 6(d). The function *replace*() is defined to do this.

Let $V^* \subset V$ be obtained using *pclos*() then *extend*(), as outlined above, and let v_{new} be a new node that does not appear in V. Function *replace*() replaces V^* with v_{new} in V, and connects v_{new} to the rest of the graph. To aid us in the definition, we begin by defining the *outcut*, *incut* and the *internal* edges of V^*. The *incut* and *outcut* of the group of shaded nodes in Fig. 6(c) are marked.

Definition 4. *Let $\vartheta_{out}(V^*)$ denote the outcut of G associated with V^*, defined as the set of arcs of G pointing out of V^*, let $\vartheta_{in}(V^*)$ denote the incut of G associated with V^*, i.e., the set of arcs of G leading into V^*, and let $\vartheta_{int}(V^*)$ denote internal edges, that connect two nodes inside V^*. $\vartheta_{out}(V^*)$, $\vartheta_{in}(V^*)$ and $\vartheta_{int}(V^*)$ are given by:*

$$\vartheta_{out}(V^*) = \{(v_d \leftarrow v_s) \mid v_s \in V^*, v_d \in V \setminus V^*\}$$

$$\vartheta_{in}(V^*) = \{(v_d \leftarrow v_s) \mid v_d \in V^*, v_s \in V \setminus V^*\}$$

$$\vartheta_{int}(V^*) = \{(v_d \leftarrow v_s) \mid v_d, v_s \in V^*\}$$

Function *replace*() replaces each arc $(v_d \leftarrow v_s) \in \vartheta_{out}(V^*)$ with a new arc $(v_d \leftarrow v_{new})$ of the same type, and replaces each arc $(v_d \leftarrow v_s) \in \vartheta_{in}(V^*)$ with a new arc $(v_{new} \leftarrow v_d)$ of the same type. Arcs in $\vartheta_{int}(V^*)$ simply disappear along with the nodes in V^*.

The definitions of $\vartheta'_{out}(V^*)$ and $\vartheta'_{in}(V^*)$ below define the final part of the "rewiring" carried out by *replace*().

Definition 5. *Let $ty \in \{used, wgBy\}$. Then:*

$$\vartheta'_{out}(V^*) = \{(v \xleftarrow{ty} v_{new}) \mid v \xleftarrow{ty} v' \in \vartheta_{out}(V^*)$$
$$\vartheta'_{in}(V^*) = \{(v_{new} \xleftarrow{ty} v) \mid v' \xleftarrow{ty} v \in \vartheta_{in}(V^*)$$

And the full definition of *replace*() is

Definition 6 (replace).

$replace(V^*, v_{new}, G) = (V', E')$, where:

$$\begin{aligned} V' &= V \setminus V^* \cup \{v_{new}\} \\ E' &= E \setminus (\vartheta_{out}(V^*) \cup \vartheta_{in}(V^*) \cup \vartheta_{int}(V^*))\} \\ &\quad \cup \vartheta'_{out}(V^*) \cup \vartheta'_{in}(V^*)\} \end{aligned}$$

It is easy to verify that the resulting graph is type-correct. All boundary nodes in V^* are of the same type $t \in \{En, Act\}$, as noted above, and v_{new} is of type t by construction. Since the arcs have the same type as those they replace, it follows that *replace*() preserves type correctness.

We now provide an initial definition of our *Group*() operator, under the simplifying assumption that all nodes in V_{gr} are of the same type before closure. We denote this type by $type(V_{gr})$ (with a slight abuse of notation), and denote the initial definition of *Group* as $Group_{hom}$. Definitions 8 and 9 in Section 4.2 remove the assumption of type-homogeneity.

Under assumption of type homogeneity, the grouping operator is a functional composition of *pclos*(), *extend*(), and *replace*() functions, defined as follows.

Definition 7 (Homogeneous Grouping). *Let $G = (V, E) \in PG_{gu/ea}$, $V_{gr} \subseteq V$ be a type-homogeneous set, and let v_{new} be a new node with $type(v_{new}) = type(V_{gr})$.*

$$\begin{aligned} & Group_{hom}(G, V_{gr}, v_{new}) = \\ & \qquad replace(\\ & \qquad extend(\\ & \qquad pclos(V_{gr}, G), V, type(V_{gr})), v_{new}, G) \end{aligned}$$

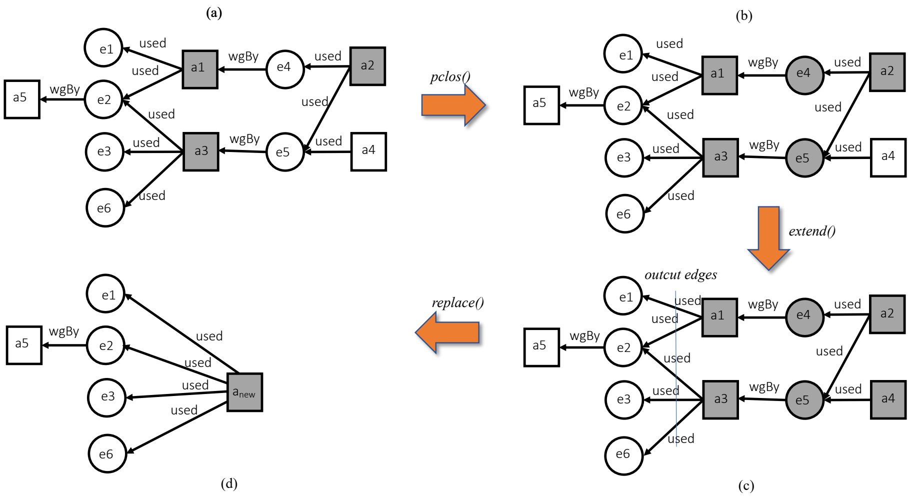


Figure 7: Path closure and replacement with extension to a set of activity nodes.

Fig. 7 shows the application of $Group_{hom}$ to a set of Activity nodes. The progression is similar to that of Fig. 6. This time $type(v) = Act$ for each $v \in V_{gr} = \{a_1, a_2, a_3\}$, and V_{gr} is replaced by another activity node, a_{new}. The *pclos* operator ensures that the nodes e_4 and e_5 are included (Fig. 7(b)), and the *extend* operator includes the activity node a_4 in Fig. 7(c). Note that there are no incut edges: $\vartheta_{in}(V^*) = \emptyset$. All shaded nodes are replaced with a_{new} and the graph is rebuilt by the operator $replace()$. In the next section we remove the assumption that all the nodes initially selected are of the same type.

4.2. Generalization to e-grouping and a-grouping

So far we have described the grouping operator in terms of the component functional parts. We have been operating under the assumption made in Definition 2: that there is only one subgraph induced by $pclos(V_{gr}, G)$. In the case where we have two or more subgraphs, the $extend()$ operator and the $replace()$ operator would iterate over the set of subgraphs produced, and be applied to each subgraph separately.

We have also been operating under the assumption of group homogeneity: that all nodes in V_{gr} are of the same type. Additional care must be taken if we allow V_{gr} to include both node types. The type of the replacement node must now be specified, as it is no longer implied from the type of the nodes in V_{gr}.

Indeed, the choice of the type can lead to different abstracted graphs. Thus, we will now refer to grouping as *t-grouping*, where $t \in \{En, Act\}$, i.e., *e-grouping* or *a-grouping*.

Consider Fig. 8. Fig. 8(a) is a subgraph of our running example. Fig. 8(a-1, a-2) illustrates the application of the $Group_{hom}$ operator (Def. 7), assuming *a-grouping* and $V_{gr} = \{e_4, a_2\}$. Although *pclos* has no effect, the extension incorporates activity node a_1 because the boundary nodes for the set to be replaced must be of type *Act*. In Fig. 8(a-2) *replace* replaces all these nodes with a_{new} and edits the edges of the graph accordingly.

Consider now the case of *e-grouping* in Fig. 8(e-1, e-2). Again, *pclos* has no effect, but the extension leads to the incorporation of e_5, which in turn leads to the pattern shown in Fig. 8(e-2), involving two generation events for the new entity e_{new}. Although this is a valid pattern, the two generation events must be simultaneous (this is one of the temporal constraints defined in [10]):

$$ev(wgBy(e_{new}, a_1)) \preceq ev(wgBy(e_{new}, a_3)) \quad \wedge \tag{2}$$

$$ev(wgBy(e_{new}, a_3)) \preceq ev(wgBy(e_{new}, a_1)) \tag{3}$$

The intuitive interpretation for this pattern is that each of the two activities generated one entity in the group represented by e_{new}, and that the abstraction makes these two events indistinguishable. Formally, nothing further needs to be done to the graph. We will explore the implications of the event ordering rules further in Section 5. However it is more natural for a PROV graph to record an entity as having been generated by a single activity, and to record a single event of generation. We can restore, if desired, the more natural pattern whereby one single event is recorded as having generated e_{new}. This is achieved by forming a single generating activity, by merging the set of generating activities together as a new (abstract) node a_{new}. In the example, this leads to the graph in Fig. 8(e-3).

We now formalize these considerations by introducing two definitions for *Group*. The first, which we call *t-grouping* where $t \in \{En, Act\}$, is agnostic of multiple generation patterns, while the second (*strict e-grouping*) applies a further step to e-grouping to ensure that the new graph is free from multiple generation patterns. Note that a-grouping does not need a similar strict version, since the new node, an activity, does not have the possibility of multiple generation.

Definition 8 (t-Grouping). *Let $G = (V, E) \in PG_{gu/ea}$, $V_{gr} \in V$, $t \in \{En, Act\}$, and let v_{new} be a new node with $type(v_{new}) = t$. Then:*

$$\begin{aligned}&Group(G, V_{gr}, v_{new}, t) = \\ &\qquad replace(extend(pclos(V_{gr}, G), V, t), v_{new}, G)\end{aligned}$$

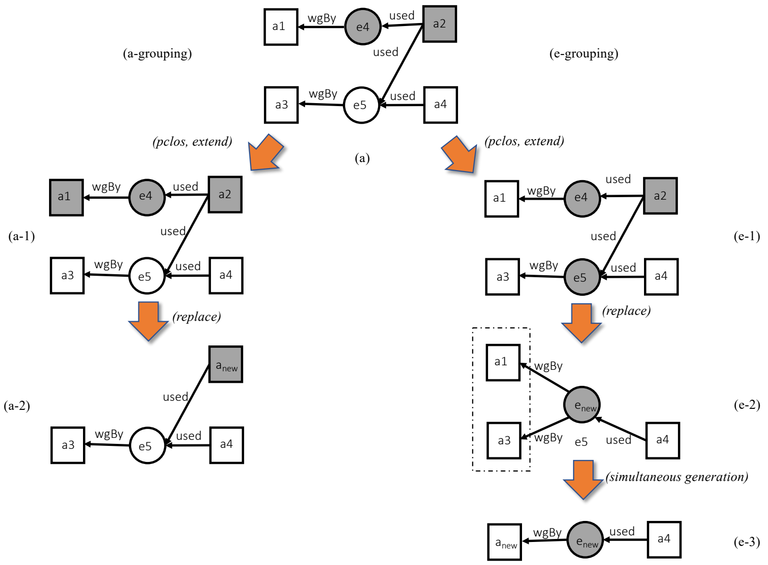


Figure 8: e-grouping and a-grouping on mixed type nodes

Note that the assumption that boundary nodes in the closure are homogeneous still holds in this case.

Strict e-grouping performs the further step illustrated in the transition from Fig. 8(e-2) to Fig. 8(e-3). Note that the *Group* operator can only produce a multiple generation pattern if $type(v_{new}) = En$.

Definition 9 (Strict e-Grouping). *Given $G = (V, E) \in PG_{gu/ea}$, $V_{gr} \in V$, $t \in \{En, Act\}$, and a new node v_{new} with $type(v_{new}) = En$, let*

$$G' = (V', E') = Group(G, V_{gr}, v_{new}, En).$$

Let $V_{gen} = \{a \in V' \mid a \xleftarrow{wgBy} v_{new} \in E'\}$ be the set of activity nodes that generate v_{new} according to G', and let a_{new} be a new activity node. Then:

$$Group_{str}(G, V_{gr}, v_{new}, t) = \begin{cases} G' & \text{if } |V_{gen}| \leq 1 \\ replace(V_{gen}, a_{new}, G') & \text{otherwise} \end{cases}$$

4.3. Justifying relations

In this section, we clarify what it means to say that relations in the abstract graph are justified, and show that the relations produced by the *Group* operator (more properly, the family of *Group* operators) are justified.

Definition 10. *An abstract node is justified by the concrete graph if (i) it appears unchanged in the concrete graph, or (ii) it is a new node representing a* *set of concrete nodes, and one of the concrete nodes has the same type as the abstract node. An abstract relation between two nodes is justified if (i) the nodes are justified (in either sense (i) or (ii) above) and (ii) the type of the abstract relation is unchanged. An abstract graph is justified if the nodes and relations in it are justified.*

To see that the *Group* operator produces justified abstract graphs, consider two graphs: a concrete one, C, represented by the graph $G_C = (V_C, E_C)$, and an abstract one A, represented by the graph $G_A = (V_A, E_A)$. If $(v_d^A \xleftarrow{t} v_s^A)$ is a relation in E_A of type t, we need to show that there is a justifying relation $(v_{d'}^C \xleftarrow{t'} v_{s'}^C)$ in E_C. Observe too that the *Group* operator removes some nodes from G_C but produces only one new node v_{new} in G_A, so there are three cases to consider. Either (i) the relation is unchanged, so $v_d^A = v_{d'}^C$, $t = t'$ and $v_s^A = v_{s'}^C$; or (ii) the destination or (iii) the source node is a new (and therefore abstract) node which the *Group* operator has inserted as a replacement for a set of nodes, and the other node is unchanged. In all cases the type of the relation must be unchanged, so $t = t'$.

To see that relations are justified in this sense, consider how the nodes v_d^A and v_s^A in $(v_d^A \xleftarrow{t} v_s^A)$ were identified.

Considering case (i) first, we need to show that the type of the relation remains unchanged during the abstraction operation. But since v_s^A and v_d^A have not changed from the concrete graph, we know that $v_s^C, v_d^C \in V \setminus V^*$. From this, and observation of Definition 4, it follows that none of $\vartheta_{out}(V^*)$, $\vartheta_{in}(V^*)$, and $\vartheta_{int}(V^*)$ apply and so, by Definition 6, neither v_s^A or v_d^A is removed from V_C in the transformation by *Group* to V_A. Thus the relationship is maintained in E_A, and the type of the relation in E_A does not change, $t = t'$.

Consider next case (ii), in which v_d^A is the new node v_{new} and v_s^A is unchanged. In this case, to justify the new relation, we need to show that there is a relation $(v_{d'}^C \xleftarrow{t'} v_{s'}^C)$ in E, and that the operation of *Group* produces the relation $(v_{new} \xleftarrow{t} v_s^A)$ in E_A, where $V_s^A = V_{s'}^C$ and $t = t'$. We see this by considering the definition of $\vartheta_{in}(V^*)$ in Definition 4. The source of the relation is unchanged, since $v_s \in V \setminus V^*$, so it is not in the set V^* that has been chosen to be abstracted. The type of v_{new} is given by the type of the boundary nodes in V^*, and the replacement relation is given by the definition of $\vartheta'_{in}(V^*)$ in Definition 5, from which we see that the source and type of the relation are unchanged in E_A.

Case (iii) is similar, except that v_s^A is the new node v_{new} and v_d^A is unchanged, and follows from inspection of $\vartheta_{out}(V^*)$ and $\vartheta'_{out}(V^*)$.

To see that this reasoning applies across all group operators ($group_{hom}$, t-grouping, e-grouping and strict e-grouping), observe that the definitions called upon in the reasoning above are the definitions of $\vartheta_{in}, \vartheta_{out}, \vartheta_{int}, \vartheta'_{in}$, and ϑ'_{out} in Definitions 4 and 5, and the definition of *replace* in Definition 6, and that these do not change across any of the grouping operators.

4.4. Complexity of grouping operators

The grouping operator ensures that the validity of a PROV graph is preserved, that is, the abstracted graph that results from a valid PROV graph does not violate PROV constraints. Such a guarantee, however, comes at a cost which is defined by the complexity of the $pclos()$ and $extend()$ operators. Here we analyse their worst-case complexity. Firstly, observe that the closure operator can be reduced to a special case of the node reachability problem for acyclic digraphs. Specifically, given two nodes $v_1, v_2 \in V_{gr}$ such that v_1 is reachable from v_2, we need to collect all nodes along *all* paths that connect v_1 to v_2. This can be accomplished by enumerating all nodes x that are reachable from each $v \in V_{gr}$ while keeping track of the corresponding paths. Whenever $x \in V_{gr}$, we collect all nodes along the recorded path from v to x.

It is easy to see that the worst-case scenario occurs when the nodes in V_{gr} are located at the two ends of the graph, i.e., they are either source or sink nodes. In such case, the reachability algorithm needs to visit *all* nodes V and *all* edges E in the graph, and it must additionally keep track of all edges it traverses.

Using a simple BFS approach, we can solve the reachability problem in $O(|V| + |E|)$ steps, which in the worst-case is $O(|V|^2)$, with $O(|E|)$ space complexity for recording all edges. Note that the many algorithms that exist to address the problem aim to strike a balance between the cost of pre-processing the graph in order to efficiently answer multiple reachability queries, and the complexity of each individual query. In our case, however, there is little advantage in pre-processing as we expect the closure over V_{gr} to be computed only once.[6]

An experimental evaluation of the actual cost of computing closures in practice is beyond the scope of this paper, which is focused on the theoretical underpinnnings of the abstraction operations. However, two factors suggest that the practical complexity will be considerably less than the worst-case. Firstly, we can stop the graph traversal as soon as we have visited all V_{gr} nodes. Unless one of those is a sink node, this results in pruning part of the graph. Note also that in this case the abstraction will consist of one single abstract node that represents the entire graph, because the closure will include all nodes, which is unlikely to be a desired outcome. And secondly, in a $PG_{gu/ea}$ graph not all edges are allowed, in fact $PG_{gu/ea}$ graphs are bipartite with respect to the nodes types (entities and activities). Furthermore, there is at most one generating activity per entity. These factors greatly reduce the expected number of edges to much less than the theoretical maximum $|V|^2$.

Regarding the $extend()$ operator, note that this requires all nodes in $pclos(V_{gr}, G)$ to be visited in order to check their type and possibly extend the closure to their immediate successors. This is a linear problem in the worst case, namely when the closure contains all nodes in the graph.

[6] Note however that, when consecutive abstraction rounds are envisioned, i.e., abstraction over abstracted graphs, pre-processing may be appropriate, but that needs to be balanced against the cost of updating the data structures after each abstraction round.

4.5. Alternative approaches

An alternative weaker approach to abstraction. Our entire approach is based on the premise that we aim to preserve the original *used* and *wgBy* relationships, possibly at the expense of incorporating additional nodes into the abstraction, i.e., by means of the *extend*() operator. For completeness, we mention here one alternative approach which replaces affected *wgBy* and *used* relationships with a weaker generic relationship. The PROV *influence* relationship is defined as *An influence relation between two objects o_2 and o_1 is a generic dependency of o_2 on o_1 that signifies some form of influence of o_1 on o_2*[7].

The generic nature of the influence relationship, denoted *wasInfluencedBy*, is captured formally in the PROV-CONSTRAINTS document [10], specifically Inference 15[8] which states that $wasInfluencedBy(id;\ e, a, attrs)$ follows from both $used(id;\ a, e, t, attrs)$ and from $wgBy(id;\ e, a, t, attrs)$, and that $wasInfluencedBy(id;\ a2, a1, attrs)$ follows from $wasInformedBy(id;\ a2, a1, attrs)$.

This inference rule justifies replacing *used* and *wgBy* relationships with the weaker *wasInfluencedBy* as needed. Consider Fig. 9, where the starting point is the same as in Fig. 8 and the goal is to abstract the set $\{e_2, a_2\}$. As we have seen, this can be accomplished through either e-grouping or a-grouping. Considering first a-grouping (on the left hand side of Fig. 9), first we generalise e_4 *wgBy* a_1 to e_4 *wasInfluencedBy* a_1 as shown on the left hand side. At this point, a_2 and e_4 are collapsed into the new a_{new} node, but notice that there is no need to expand the grouping set, i.e., to include a_1, because the new PROV graph at the bottom left is already type-correct. In particular, a_{new2} *used* e_5 is a legal relationship, and is justified according to the argument in the previous section.

Similarly, we can easily construct a valid abstract PROV graph for e-grouping without using *extend*(), by first generalising a_2 *used* e_5 to a_2 *wasInfluencedBy* e_5, then creating the abstract entity node e_{new}, and connecting the node to the rest of the graph as shown in the bottom right part of the figure.

With this approach we relax the requirement that the original relationships be preserved, accepting to use the more generic *wasInfluencedBy* relationship in return for not having to expand the grouping set to include “boundary” nodes to ensure type-correctness.

Note on subdivision of input set. A second alternative approach is to subdivide the initial set chosen by the user into multiple smaller sets, thereby reducing the amount of extra information hidden. This subdivision could be done manually, at the cost of a greater cognitive load for the user, but ideally would be done automatically, with the algorithm searching for an “ideal” subdivision according to some optimisation function. Care would be required to develop this algorithm. For example the subdivision could be taken to its logical limit by abstracting individual nodes, but this would have to be balanced against

[7] https://www.w3.org/TR/prov-dm/#term-influence

[8] https://www.w3.org/TR/2013/REC-prov-constraints-20130430/#influence-inference

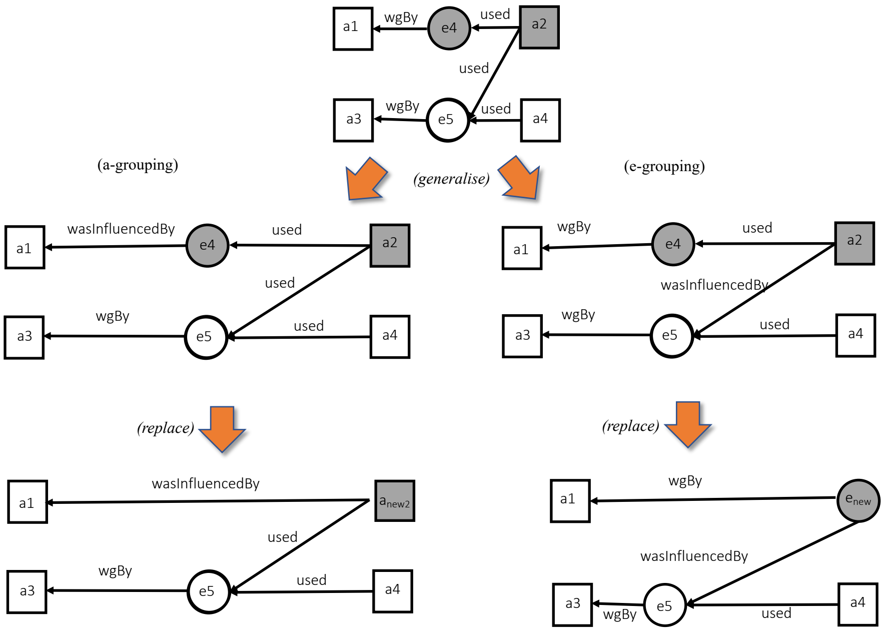


Figure 9: Abstraction by relationship generalisation

the increasing revelation of the structure of the graph. Understanding these trade-offs to develop an automatic approach would require substantial testing and evaluation.

This approach would also require a measure of evaluating the "damage" to a graph caused by an application of an abstraction operator. We address the issue of damage evaluation in [20] where we present a simple policy model and language for controlling abstraction, in the context where provenance owners want to control the disclosure of their provenance graphs. There, the owner defines a policy which results in a sensitivity value being associated with nodes, which gives us a means of evaluating the "damage" to a graph caused by the abstraction operator. In [20] we do this by means of defining a property *utility* as a counterpart to sensitivity. It is used to indicate the interest of the provenance owner in ensuring that a node be retained as part of the graph, as it represents important evidence which is not sensitive. The utility values associated to different nodes are used to quantify any loss of utility as a result of the application of *group* though a measure of *residual utility.* If we write the utility of a node n as $u(n)$, and $V_{ret} = V / V_{gr}$ is the set of nodes *not* intended to be hidden, and $V'_{ret} \subset V_{ret}$ the nodes which were in fact retained after grouping, the residual utility is simply

$$RU_V = \frac{\Sigma_{n \in V'_{ret}} u(n)}{\Sigma_{n \in V_{ret}} u(n)} \tag{4}$$

which is a measure of the proportion of the graph utility not selected by the grouping operator.

We believe that subdivision of the initial user input will be valuable in situations where the users requires a further dimension of control over provenance disclosure, but that the costs and benefits would be explored with respect to a particular user context. Thus, we regard further investigation of this approach as out of scope for the present paper.

4.6. Summary

We have presented operators that abstract information in a provenance graph. We first presented *homogeneous grouping* (in Section 4.1), in which the user selects a set of nodes of the same type, and for which the new, abstract node retains that type. Section 4.2 extended this work to allow the user to select any nodes, at the expense of having to chose the type of the final, abstract, node, and gave a further extension to deal with a problem arising from simultaneous generation of events. Section 4.3 showed that the newly created relations were justified by the original graph. Section 4.4 discussed the complexity of the new operator, and Section 4.5 briefly discussed two alternative approaches: making use of the *wasInfluencedBy* operator and subdivision of the input set of nodes.

The *Group* operator preserves schema validity: the *extend* operator ensures that all the nodes to be replaced have the same type and so the *replace* operator maintains type consistency. This is true because of the restricted focus of this work: we are considering $PG_{gu/ea}$ graphs which contain only entities and activities. Section 5 shows how event validity is ensured by identifying an order-preserving mapping between from abstract to concrete events.

It is clear that, in order to meet our initial requirement of maintaining type-correctness of the abstracted graph, in general more nodes than just the original ones selected will have to be hidden. This has implications for the use of this operator, especially given that hidden information may be a critical part of the graph. The choice that we make in this paper is to develop the abstraction operator ensuring first of all that abstract graph produced is valid and justified. Minimality remains a guiding principle, enforced by the fact that the *extend* operator only extends the set to be abstracted where necessary. This has the advantage that it allows us to ensure that the operator maintains the PROV-compliance of the new graph, but the disadvantage that the loss of information cannot be controlled.

The two alternative approaches in Section 4.5 both present valuable further lines of enquiry.

5. Abstraction over events

In Section 3.3 we recalled the definition of PROV ordering constraints C2-C5, given in the PROV-CONSTRAINT document, which must be satisfied by any valid PROV graph. We now extend the notion of validity to events

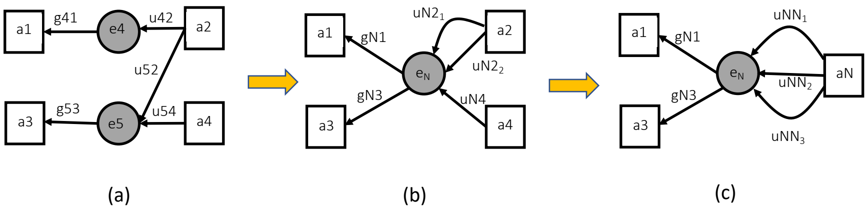


Figure 10: Abstraction over a document content, and associated abstracted events

defined on abstract graphs G^A. In general these are not the same events as G^C's, because when G^A is obtained using either e-grouping or a-grouping over some concrete base graph G^C, both entities and relationships may have changed. Specifically, when a_{new} is created through a-grouping, G^A's events include $start(a_{new})$, $end(a_{new})$, as well as $ev(used(e, a_{new}))$, $ev(wgBy(a_{new}, e))$ for all e that are generated by or used by a_{new}. For e-grouping, the new events are $ev(wgBy(e_{new}, a))$ and $ev(used(e_{new}, a))$ for any a that has generated (resp. used) e_{new}. We are going to refer to the two sets of events in G^C and G^A as EV^C, EV^A, respectively.

Note that until this point we have been using single edges between nodes. Abstraction, however, can create single relations from more than one concrete relation. PROV allows multiple edges between the same nodes to be distinguished using identifiers, and in this section we introduce identifiers on relations to allow us to develop the abstract event preorder on the abstract graph that is consistent with the preorder on the concrete graph. We use symbols such as g_{41}, as in Fig. 10, to indicate relationships like $wgBy(e_4, a_1)$. With slight abuse of notation, but in the interest of simplicity, in the following we are going to use g_{41} to also denote $ev(wgBy(e_4, a_1))$ when it is clear from the context that we refer to the event rather than to the relationship itself.

To fix ideas, consider G^C in Fig. 10(a), where two sections of a document are independently generated by two editing activities, and then they are independently used by two more activities. Note that this is a slight extension of the abstract pattern of Fig. 8, where the document sections are e_4, e_5. The e-grouping set $V_{gr} = \{e_4, e_5\}$ represents the whole document.

The argument in this section is focused on e-grouping and generation-usage constraints, and omits similar logic which applies to a-grouping and to other temporal constraints. Let G^A be the result of (non-strict) e-grouping, as depicted in Fig. 10(b), where the abstract generation and usage events are given new names, namely g_{Ni} as a shorthand for $wgBy(e_N, a_i)$, and u_{Ni_j} for each usage j of the form $used(a_i, e_N)$. The form u_{Ni_j} is only needed when multiple *used* relationships link the same two nodes. Thus, $EV^C = \{g_{41}, g_{53}, u_{42}, u_{52}, u_{54}\}$ and $EV^A = \{g_{N1}, g_{N3}, u_{N2_1}, u_{N2_2}, u_{N4}\}$. If G^C is valid, by constraint C3 the following must hold:

$$g_{41} \preceq u_{42}, \quad g_{53} \preceq u_{52}, \quad g_{53} \preceq u_{54} \tag{5}$$

where $\preceq$ is the preorder relationship introduced in Section 3.2. Importantly, note however that $g_{41} \preceq u_{54}$ does not hold. Similarly, for G^A to be valid we must have:

$$g_{N1} \preceq u_{N2_1}, \quad g_{N1} \preceq u_{N2_2}, \quad g_{N1} \preceq u_{N4} \tag{6}$$

$$g_{N3} \preceq u_{N2_1}, \quad g_{N3} \preceq u_{N2_2}, \quad g_{N3} \preceq u_{N4} \tag{7}$$

To understand multiple generation and usage events associated with the same entity, recall that PROV-DM [23] defines those events as follows:

- **Generation** *is the completion of production of a new entity by an activity* (Section 5.1.3 of [23])
- **Usage** *is the beginning of utilizing an entity by an activity* (Section 5.1.4 of [23])

Thus, e_N is generated when both generation events g_{N1}, g_{N3} have occurred (by Constraint C2 from PROV-CONSTRAINTS, this implies that these abstract events must be simultaneous), and it starts being used when the "earliest" of the usage events takes place, keeping in mind that no ordering relationships amongst the usage events is necessarily defined.

We aim to establish a formal relationship between concrete and abstract events, and between temporal constraints amongst those events. Specifically, we will see that some constraints over events in EV^C map directly to constraints in EV^A, but also that, to be valid, G^A may require additional constraints that are not in G^C. Thus, intuitively, we aim to show that validity of G^A follows from validity of G^C, but only in part.

5.1. Constraint C3 (generation precedes usage)

For simplicity, consider initially the single temporal constraint C3 (generation precedes usage), i.e., assume that the preorder on EV^C satisfies C3. We observe that Def. 5 (pg 19) effectively defines a mapping between relationships (graph arcs) in G^C that cross the boundaries of a "convex" subgraph V^* of G^C obtained by closure *pclos*() and expansion (*extend*(), for type consistency), and the new arcs in G^A. Specifically, given V^*, Def. 5 provides the basis for the *replace*() operator, by mapping:

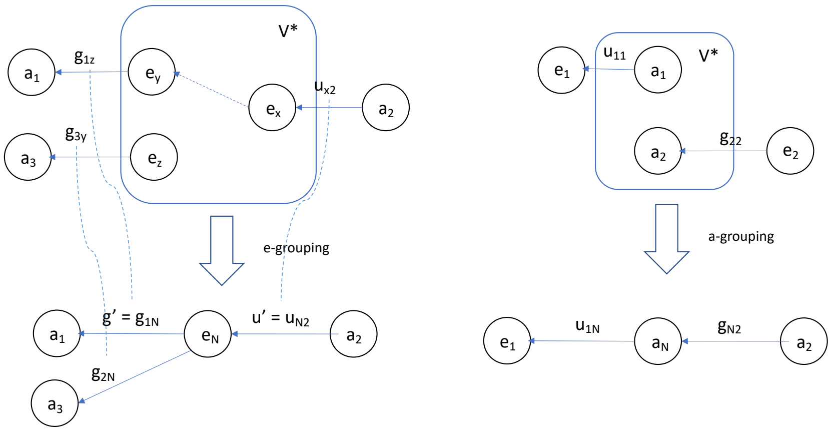
$$\text{arcs: } v \xleftarrow{ty} v' \in \vartheta_{out}(V^*) \text{ to arcs } v \xleftarrow{ty} v_{new}; \tag{8}$$

$$\text{arcs: } v' \xleftarrow{ty} v \in \vartheta_{in}(V^*) \text{ to arcs } v_{new} \xleftarrow{ty} v; \tag{9}$$

$$\text{arcs: in } \vartheta_{int}(V^*) \text{ to themselves.} \tag{10}$$

In turn, this defines a mapping between concrete and abstract events corresponding to the arcs (*wgBy* and *used*). We denote such mapping by:

$$\psi : EV^C \to EV^A \tag{11}$$



(a) e-grouping

(b) a-grouping

Figure 11: Paths within V^* determine temporal constraints in the abstract graph.

For example, for the events in Fig. 10(a) and (b) the mappings are: $\psi(g_{41}) = g_{N1}$, $\psi(u_{42}) = u_{N2_1}$, $\psi(u_{54}) = u_{N4}$, etc. By the nature of *replace*(), the only interesting constraints covered by $\psi()$ relate to events that occur on the boundary of V^*.

In this example, note that $g_{41} \preceq u_{42}$ holds in G^C, and $\psi(g_{41}) = g_{N1} \preceq u_{N2_1} = \psi(u_{42})$ holds in G^A, that is, $\psi()$ preserves the order in G^C. But for instance, for G^A to be valid, $g_{N1} \preceq u_{N4}$ must also hold, which however does not follow from a corresponding constraint in G^C, in fact g_{41} and u_{54} are unrelated. Thus, intuitively we expect to be able to "explain" only some of the constraints in G^A in terms of constraints on G^C.

To formalise this idea, and observing that $\psi()$ is injective and total, thus invertible, we define the *support* of a constraint in G^A as follows.

Definition 11 (Support of constraint C3 in G^A). *Let $g' = (a_1 \leftarrow e_N), u' = (e_N \leftarrow a_2) \in EV^A$ be two generation and usage events, respectively, associated with abstract entity node e_N. For G^A to be valid relative to C3, $g' \preceq u'$ must hold. The support of $g' \preceq u'$ in G^C is defined as the constraint $\psi^{-1}(g') \preceq \psi^{-1}(a')$.*

Proposition 1 (Condition for support). *Let $g' = (a_1 \leftarrow e_N), u' = (e_N \leftarrow a_2) \in EV^A$, $g' \preceq u'$ be a constraint as in Def. 11, and let $\psi^{-1}(g') = (a'_1 \leftarrow e_y), \psi^{-1}(a') = (e_x \leftarrow a'_2) \in EV^C$. The support of $g' \preceq a'$ is defined in G^C only if there a path from e_x to e_y in V^*.*

Proof. Firstly, observe that $\psi^{-1}(g') \in \vartheta_{out}(V^*)$ and $\psi^{-1}(u') \in \vartheta_{in}(V^*)$. These both follow directly from our definition of $\psi()$. Also, $a_1 = a_1'$ and $a_2 = a_2'$. Thus, the situation is as depicted in Fig. 11(a), showing the generation and usage events on the boundaries of V^*. If there is a path from e_x to e_y, then this must form a chain of generation-usage relationships, and by transitivity of the preorder, this entails $g_{1y} = (a_1 \leftarrow e_y) \preceq (e_x \leftarrow a_2) = u_{x2}$, which is the support of $g' \preceq a'$.

On the other hand, consider the constraint $(a_3 \leftarrow e_N) \preceq (e_N \leftarrow a_2)$. Its support is $(a_3 \leftarrow e_z) \preceq (e_x \leftarrow a_2)$, however if there is no path from e_x to e_z, we cannot entail this constraint through transitivity, and in fact there is no such constraint in G^C.

The example in Fig. 10 shows a similar scenario, as noted above. Thus, the dependencies within V^* determine which additional constraints must hold in the abstract graph, that do not hold in the concrete graph. We can provide a simple interpretation of this situation, by considering the set of all possible total orderings amongst events in EV^C, or *unfoldings* of G^C, that are consistent with the temporal constraints in G^C. It should be clear that adding constraints reduces the number of unfoldings. Thus, we have shown that ensuring validity of G^A may require constraints that correspond to *new* constraints that did not need to hold in G^C. We could say that G^A represents only a fragment of the unfoldings of G^C. In the example, these are only the unfoldings where $g_{41} \preceq u_{54}$ is true.

5.2. Constraints C2, C4, C5

Regarding C2 (generation-generation ordering), we have again that new constraints on abstract events need to hold in G^A, even when those are mapped from concrete events that are unrelated. For example, consider again Fig. 10 where g_{41} and g_{53} are unrelated, yet in G^A, the corresponding abstract events must be simultaneous, because both are associated wth the generation of e_N:$\psi(g_{41}) \preceq \psi(g_{53})$ and $\psi(g_{53}) \preceq \psi(g_{41})$. Generalising from the example, it is straighforward to see that for any pair of generation events $g, g' \in \vartheta_{out}(V^*)$ in G^A the following must hold:

$$\psi(g) \preceq \psi(g'), \quad \psi(g') \preceq \psi(g)$$

Regarding C4 and C5, intuitively these constraints state that any entity generation and usage events must "lie within" the interval defined by the start and end events for the relevant activities. Consider Fig. 11(b), with constraints:

$$start(a_N) \preceq u_{1N} \preceq end(a_N) \tag{12}$$

$$start(a_N) \preceq g_{N2} \preceq end(a_N) \tag{13}$$

We would like to define $start(a_N)$, $end(a_N)$ in terms of corresponding concrete events, i.e., $start(a_1)$, $end(a_1)$ and $start(a_2)$, $end(a_2)$, so that we can determine the relationship between the two. Our construction of $\psi()$ does not help here, however, because $\psi()$ is defined on constraints (usage, generation) that are associated with edges in the abstract graph, whereas these activity constraints are associated to (activity) nodes. The problem is that, to the best of our knowledge, PROV-CONSTRAINTS does not offer an intution to define such mappings. Unlike for "composite" entities, where usage and generation can be defined as we have done at the start of this section, there seems to be no corresponding notion of "compound activity" where the start and end events of the composite are defined in terms of those of its components.[9] Thus, we regard further investigations involving constraints C4, C5 as out of scope, and as an open problem.

6. Summary and further research

We have proposed a model for the principled hiding of provenance based on a formal definition of abstraction, in which elements of a provenance graph are grouped together and replaced by a single abstract node. A guiding principle throughout is that we avoid the introduction of *unjustified dependencies*: our abstraction will reduce the information content of a provenance graph, but it will not introduce information that is not justified by the combination of the abstraction and the information in the original graph. The abstraction acts on and results in provenance graphs which are PROV compliant. A separate paper [20] presents the tool implementing this model in detail.

Our grouping operators ensure the validity of the resulting abstracted graph: if PG is a PROV graph, that is, it conforms to the PROV data model, meeting the constraints outlined in [23]. Also, no unjustified dependencies are introduced into PG': a relationship involving v_{new} is only created as a result of a mapping from an existing relationship involving elements of V_{gr}. Strictly, if a' and e' are *directly* related in PG', we guarantee that for each of a' and e', either they or an element in their abstracted set are directly related in PG.

The work described in this paper is progressing in two main directions. First, we are aware that the fragment of PROV to which *Group* applies does not cover all relation types. Nevertheless, the method described in the paper for reasoning about PROV graph transformation can be used as a guideline to extend the work to the missing parts of PROV. A first extension would consider agents, possibly beginning with an initial simplifying assumption that agents are disjoint from entities and activities.

Secondly, although in this work we have predominantly chosen not to allow the grouping operators to change types of edges, in Section 4.5 we have explored the use of the *wasInfluencedBy* relation, which is a supertype of both *wgBy* and *used*. Allowing the operator to change the node types has the advantage of retaining more information from the original PROV graph, as we point out,

[9] There is a notion of one activity starting or ending other activities, but that does not seem to help with the mapping we are seeking.

and we should explore more completely the implications of allowing these types of changes more generally.

Acknowledgements

The support of the ONRG and the EPSRC in funding this research is gratefully acknowledged. We are also grateful to the anonymous reviewers for the insightful suggestions that helped us improve the paper.

References

[1] Eleanor Ainy, Pierre Bourhis, Susan B. Davidson, Daniel Deutch, and Tova Milo. Approximated summarization of data provenance. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 483–492, New York, NY, USA, 2015. ACM.

[2] D Bell. The Bell-LaPadula model. *Journal of computer security*, 4(2):3, 1996.

[3] Smriti Bhagat, Graham Cormode, Balachander Krishnamurthy, and Divesh Srivastava. Class-based graph anonymization for social network data. *Proc. VLDB Endow.*, 2(1):766–777, August 2009.

[4] O Biton, S Cohen Boulakia, S B Davidson, and C S Hara. Querying and Managing Provenance through User Views in Scientific Workflows. In *ICDE*, pages 1072–1081, 2008.

[5] Barbara Blaustein, Adriane Chapman, Len Seligman, M. David Allen, and Arnon Rosenthal. Surrogate parenthood: Protected and informative graphs. *Proc. VLDB Endow.*, 4(8):518–525, May 2011.

[6] Uri Braun, Avraham Shinnar, and Margo Seltzer. Securing provenance. In *Proceedings of the 3rd conference on Hot topics in security*, pages 4:1—-4:5, Berkeley, CA, USA, 2008. USENIX Association.

[7] Jeremy Bryans, John S. Fitzgerald, Cliff B. Jones, and Igor Mozolevsky. Formal modelling of dynamic coalitions, with an application in chemical engineering. In *ISoLA*, pages 91–98. IEEE, 2006.

[8] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani Thuraisingham. A language for provenance access control. In *Proceedings of the first ACM conference on Data and application security and privacy*, CODASPY '11, pages 133–144, New York, NY, USA, 2011. ACM.

[9] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani Thuraisingham. Transforming provenance using redaction. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, SACMAT '11, pages 93–102, New York, NY, USA, 2011. ACM.

[10] James Cheney, Paolo Missier, and Luc Moreau. Constraints of the Provenance Data Model. Technical report, World Wide Web Consortium, 2012. `http://www.w3.org/TR/prov-constraints/`.

[11] James Cheney and Roly Perera. An analytical survey of provenance sanitization. In Bertram Ludäscher and Beth Plale, editors, *Provenance and Annotation of Data and Processes*, pages 113–126, Cham, 2015. Springer International Publishing.

[12] Susan B Davidson, Sanjeev Khanna, Tova Milo, Debmalya Panigrahi, and Sudeepa Roy. Provenance views for module privacy. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 175–186, New York, NY, USA, 2011. ACM.

[13] Susan B. Davidson, Sanjeev Khanna, Sudeepa Roy, and Sarah Cohen Boulakia. Privacy issues in scientific workflow provenance. In Paolo Missier, Vasa Curcin, and Susan Dadvidson, editors, *First International Workshop on Workflow Approaches to New Data-centric Science (WANDS'10)*, Indianapolis, 2010. ACM.

[14] Susan B Davidson, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen, and Yi Chen. On provenance and privacy. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 3–10, New York, NY, USA, 2011. ACM.

[15] Saumen Dey, Daniel Zinn, and Bertram Ludäscher. ProPub: Towards a Declarative Approach for Publishing Customized, Policy-Aware Provenance. In Judith Bayard Cushing, James French, and Shawn Bowers, editors, *Scientific and Statistical Database Management*, volume 6809 of *Lecture Notes in Computer Science*, pages 225–243. Springer Berlin / Heidelberg, 2011.

[16] Daniele El-Jaick, Marta Mattoso, and Alexandre A B Lima. SGProv: Summarization Mechanism for Multiple Provenance Graphs. *JIDM*, 5(1):16–27, 2014.

[17] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS '07, pages 13–18, New York, NY, USA, 2007. ACM.

[18] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 93–106, New York, NY, USA, 2008. ACM.

[19] Ziyang Liu, Susan B. Davidson, and Yi Chen. Generating sound workflow views for correct provenance analysis. *ACM Trans. Database Syst.*, 36(1):6:1–6:35, March 2011.

[20] Paolo Missier, Jeremy Bryans, Carl Gamble, Vasa Curcin, and Roxana Dánger Mercaderes. Provabs: model, policy, and tooling for abstracting PROV graphs. In *IPAW 2014: Provenance and Annotation of Data and Processes*, volume 8628 of *Lecture Notes in Computer Science*. Springer, 2014.

[21] Luc Moreau. Aggregation by provenance types: A technique for summarising provenance graphs. *arXiv preprint arXiv:1504.02616*, 2015.

[22] Luc Moreau, 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. The Open Provenance Model — Core Specification (v1.1). *Future Generation Computer Systems*, 7(21):743–756, 2011.

[23] Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes. PROV-DM: The PROV Data Model. Technical report, World Wide Web Consortium, 2012.

[24] Dang Nguyen, Jaehong Park, and Ravi Sandhu. Dependency path patterns as the foundation of access control in provenance-aware systems. In *4th USENIX Workshop on the Theory and Practice of Provenance*, 2012.

[25] Elena Zheleva and Lise Getoor. Preserving the Privacy of Sensitive Relationships in Graph Data. In Francesco Bonchi, Elena Ferrari, Bradley Malin, and Yücel Saygin, editors, *Privacy, Security, and Trust in KDD*, volume 4890 of *Lecture Notes in Computer Science*, pages 153–171. Springer Berlin / Heidelberg, 2008.

[26] Bin Zhou, Jian Pei, and WoShun Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, 10(2):12–22, December 2008.