Composition and Substitution in Provenance and Workflows

Peter Buneman
University of Edinburgh
opb@inf.ed.ac.uk

Adrià Gascón
University of Edinburgh
agascon@inf.ed.ac.uk

Dave Murray-Rust
University of Edinburgh
drust@inf.ed.ac.uk

Abstract

It is generally accepted that any comprehensive provenance model must allow one to describe provenance at various levels of granularity. For example, if we have a provenance graph of a process which has nodes to describe subprocesses, we need a method of expanding these nodes to obtain a more detailed provenance graph. To date, most of the work that has attempted to formalize this notion has adopted a *descriptive* approach to this issue: for example, given two provenance graphs under what conditions is one "finer grained" than another.

In this paper we take an *operational* approach. For example, given two provenance graphs of interacting processes, what does it mean to compose those graphs? Also, given a provenance graph of a process and a provenance graph of one of its subprocesses, what is the operation of substitution that allows us to expand the graph into a finer-grained graph? As well as provenance graphs, these questions also apply to workflow graphs and other process models that occur in computer science. We propose a model and operations that addresses these problems. While it is only one of a number of possible solutions, it does indicate that a basic adjustment to provenance models is needed if they are properly to accommodate such an operational approach to composition and substitution.

Keywords provenance, composition, substitution, abstraction

1. Introduction

One of the long-standing issues in the study of provenance is what it means to view provenance at various levels of granularity. From a practical perspective, it is important to have a systematic way of representing data at an appropriate level of abstraction. For example, in (Missier et al. 2014), an abstraction mechanism called *abstraction by grouping*, is proposed for PROV (Moreau and Missier 2013) graphs. Broadly speaking, it consists of replacing a set of nodes in a PROV graph by a new node while preserving certain type constraints that guarantee that the result of the abstraction is again a PROV graph. The motivation is twofold: simplifying provenance data for an intended audience, and hiding sensitive data. From another perspective, provenance data abstraction shows up in attempts to reconcile workflow provenance and data provenance (Amsterdamer et al. 2011).

Now one of the advantages of the PROV model (Moreau and Missier 2013; Moreau and Groth 2013) is its claimed support for various levels of granularity or for "subactivities"[1], and while the cited research formalizes what it means for one graph to be a coarser-grained representation of another graph, we would like this form of abstraction to be *invertible*: how do we synthesize a finer-grained version of provenance graph from a coarser-grained one? Figure 1 loosely illustrates this. There is a graph G_1 with three similar subgraphs which differ only by a label at one of the nodes. One could imagine generating these subgraphs with a function $f(x)$ where x is the label, and then representing the whole graph by G_2 with nodes – activities in PROV parlance– represented by applications of f. However, when it comes to *inverting* this transformation, we immediately see that there is a "plumbing" problem. How do we establish the relationship between the inputs to $f(x)$ and the edges in the graph G_2? We are asking for some graph-theoretic model of abstraction that could apply, for example, both to workflow and to provenance graphs. One can also see this as a form of compression that works, say, on PROV graphs in such a way that the compressed graphs are also PROV graphs. In that case, one would certainly want such an inversion. Even if we ignore issues concerning granularity or compression, there are certainly non-commutative operations for which we should surely record the order of inputs, such as in the confusing use of subtraction in Figure 1.

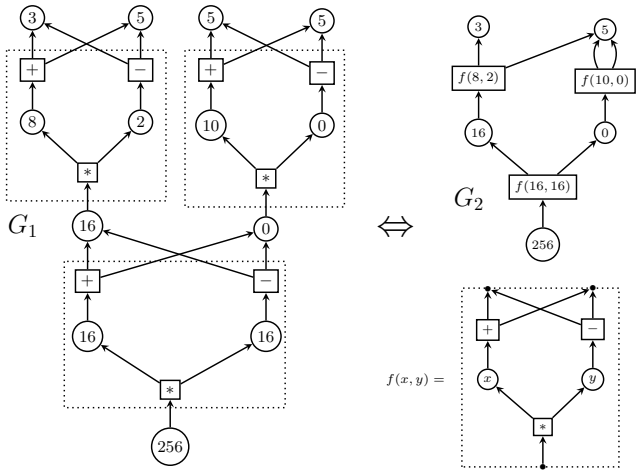

Figure 1. An attempt to specify graph abstraction

There is a related problem of provenance *composition*: suppose we have recorded provenance for two processes that are somehow related – perhaps they are processes that have communicated with each other. How do we record this communication and how do we compose the two graphs into a unified provenance record? This issue has been addressed in systems that support decentralized prove-

[1] `https://www.w3.org/2001/sw/wiki/PROV-FAQ`, 2016-03-07.

TaPP 2016, June 8–9, 2016, Washington, DC.

nance (Gehani et al. 2010). We are interested in a formal approach to the problem, which requires identifying the key properties of a suitable data model, query language, and, more concretely, a notion of composition. We believe that our model presented in Section 2 is a step forward towards that goal.

Graph substitution and composition do not look like particularly new problems to computer science and there is certainly related work. Although graph rewriting (Rozenberg 1997) does not directly address this problem, (Drewes et al. 1997; Engelfriet 1997) use hypergraphs to address an issue similar to the one we are discussing. The hypergraph edges contained *labeled sets* of nodes. If one interchanges the rôles of edges and nodes, one arrives at a model with some similarity to the model we exploit. Recursive state machines (Alur et al. 2005) require a model in which a state of one automaton can be expanded into another automaton. Although finite state machines are very different beasts from workflows and provenance models, *op cit* asks questions about reachability, which might be relevant to people interested in provenance. Closely related to this and to what we propose is the use of graph grammars (Bao et al. 2012) to describe substitution in the context of hierarchical workflow models. The model is used to good effect in a system that describes how much provenance can be revealed without compromising privacy, (Davidson et al. 2011) use a model in which nodes have labeled input and output ports.

Following these ideas, we investigate a model in which the basic components are boxes with labeled input and output ports, and the operations that combine networks built out of such boxes. We show some simple properties of the operations, describe a real scenario in which one needs to combine provenance. In contrast to (Bao et al. 2012) we investigate some more basic operations from which substitution can be derived. These operations also apply to non-disjoint representations, which we believe to be important for manipulating provenance graphs. We briefly discuss the relationship between this work and PROV.

2. The model

Before going into formalism, and in order to motivate it, consider Figure 2 which illustrates the substitution of the contents of the dotted box a in view 2 of network N, for the box f in network N'. This is a simple operation in which we connect the edges pointing at the two ports at the bottom edge of a to the outgoing edges from the corresponding ports of f. Symmetrically, the edges emanating from the boundary port at the top of a to the single input edge to f. The boxes a and f disappear leaving the result $N'[f \to N]$ of the substitution.

Unfortunately this process requires two kinds of box (dotted and not dotted). What happens if we assume only one kind of box? We can achieve this by "everting" dotted box a in view 2 of network N by moving its contents to the outside. What one might have termed the output ports of the dotted box have become the input ports of the solid box and *vice versa* but if we take "input to" to mean "pointing at the boundary of" the diagrams are the same. Our substitution operation now becomes a *symmetrical* operation: join edges to the input ports of f to the corresponding output ports of a and *vice versa*. Let us formulate this operation and look at some of its properties.

In this and subsequent diagrams we use the convention that for any non-dotted box: the input ports are on top, the output ports are on the bottom, and the ports are numbered from left to right. Also the symbols $a, b, c, \ldots$ in Figure 2 are *box identifiers* (members of $\mathcal{B}$) not labels. For example, the various boxes with c in them are all pictures of the same member of $\mathcal{B}$.

We shall assume a context of a set $\mathcal{B}$ of boxes together with functions $\mathcal{O} : \mathcal{B} \to \mathbb{N}$ and $\mathcal{I} : \mathcal{B} \to \mathbb{N}$, which give the number of output and input ports for each box. We will be at liberty to add new boxes to $\mathcal{B}$ but when we do so we must fix the values of $\mathcal{O}$ and $\mathcal{I}$ on that box.

If b is a box with m inputs and n outputs, i.e., $b \in \mathcal{N}$ with $\mathcal{O}(b) = m$ and $\mathcal{I}(b) = n$, we shall refer to b as an (m, n)-box. We shall write the i^{th} output port of b as $b^{\to i}$, for $1 \leq i \leq \mathcal{O}(b)$, and its j^{th} input port as $b^{\leftarrow j}$, for $1 \leq j \leq \mathcal{I}(b)$. If $B \subseteq \mathcal{B}$ is a set of boxes, we shall use $B^{\to} = \{b^{\to i} \mid b \in B \wedge (1 \leq i \leq \mathcal{O}(b)\}$ for the set of all output ports of B , and similarly use $B^{\leftarrow}$ for all of its input ports.

Edges connect output ports to input ports, so we can define a *network* as a pair (B, E) where $B \subseteq \mathcal{B}$ and $E \subseteq B^{\to} \times B^{\leftarrow}$. For reasons that will emerge we can usually define a network simply by its edges and assume the boxes are those that have at least one port with an incident edge in that set.

2.1 Network join

We start by defining an operation which, given a set of edges and a pair of output and input ports (p, q), produces a new set of edges where p and q are not connected and containing the edge (t, r), for every pair of ports r, t such that r is a successor of p and t is a predecessor of q. This has the effect of "bypassing" q and p in the resulting set of edges.

Defn 1. Let E be a set of edges with output port p and an input port q . The *excision* of (p, q) from E, $\mathsf{exc}(E, p, q)$ is the set of edges:

$$E \setminus (\{(p, r) \mid (p, r) \in E\} \cup \{(t, q) \mid (t, q) \in E\}) \\ \cup \{(t, r) \mid (p, r) \in E \wedge (t, q) \in E\}$$

If (p, q) and (p', q') are both pairs of input and output ports in a network N with $p \neq p'$ and $q \neq q'$ then $\mathsf{exc}(\mathsf{exc}(N, p, q), p', q') = \mathsf{exc}(\mathsf{exc}(N, p', q'), p, q)$. This commutativity means that we can extend the definition of excision to a set S of pairwise disjoint pairs of output and input ports as follows: $\mathsf{exc}(N, S) = N$ if $S = \emptyset$, and $\mathsf{exc}(N, S) = \mathsf{exc}(\mathsf{exc}(N, p, q), S')$ if $S = S' \uplus \{(p, q)\}$.

We are now in a position to define a join.

Defn 2. If $N_1 = (B_1, E_1)$ and $N_2 = (B_2, E_2)$ are networks with b_1 a (n, m)-box in B_1 and b_2 a (m, n)-box in B_2, define the *join* $N_1[b_1; b_2]N_2$ as the network $(B_1 \cup B_2, E)$ where E is given by:

$$\mathsf{exc}(E_1 \cup E_2, \\ \{(p, q) \mid (p, b_1^{\leftarrow i}) \in E_1 \wedge (b_2^{\to i}, q) \in E_2 \wedge 1 \leq i \leq n\} \cup \\ \{(p, q) \mid (p, b_2^{\leftarrow i}) \in E_2 \wedge (b_1^{\to i}, q) \in E_1 \wedge 1 \leq i \leq m\})$$

This formalizes the intuition from Figure 2 that join connects the inputs of b_1 to the corresponding outputs of b_2 and *vice versa*. Note that the networks N_1, N_2 do not have to be disjoint, and this definition deals with self-loops, i.e. edges of the form $(b^{\to i}, b^{\leftarrow j})$, on boxes.

One of the immediate consequences of this definition is that it allows us to group any collection of boxes into a single box. If $N = (B, E)$ and $B_1 \subseteq B$, define an *induced subnetwork* $N \mid B_1$ of N on B_1 in the usual way.

Prop 1. Let $N = (B, E)$ and $B_1 \subseteq B$. We can construct boxes b_1, b_2 and edge sets E_1, E_2 such that $N_1 = (B_1 \cup \{b_1\}, E_1)$ and $N_2 = ((B \setminus B_1) \cup \{b_2\}, E_2)$ are networks, $N = N_1[b_1; b_2]N_2$, $N_1 = N \mid B_1$ and $N_2 = N \mid (B \setminus B_1)$. □

In other words the subnetwork N_1 can be represented as a single box, b_2 within N_2, and the whole network recovered with a join. We believe this to be important for provenance graphs where we understand that it should be possible to treat an arbitrary set of activities as a single activity. We remark that there is a more general version of this result in which subnetwork N_1 is not required to be induced.

The join operation enjoys several other properties.

Defn 3. A network N is:

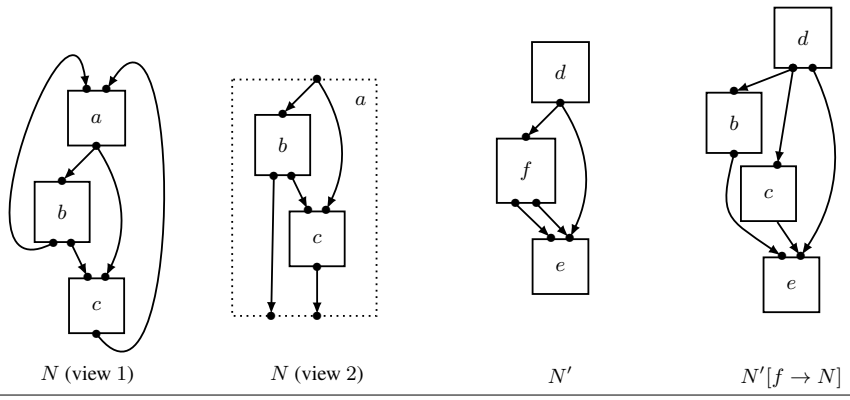

Figure 2. A join of two networks

- *full* if every port on every box in N has at least one incident edge,
- n-*input-bounded* if every box in N has at least n inputs,
- n-*output-bounded* if every box in N has at least n outputs, and

Prop 2. All properties in Definition 3 are preserved by join. □

The importance of this result is that we usually want to work with full networks; and in such networks, the boxes are prescribed by the set of edges. Networks that are not 1-output-bounded have redundant boxes, and we probably do not want to create such networks; however, $(n, 0)$- and $(0, n)$-boxes can be useful in the *construction* of networks from simpler components.

2.2 Substitution

In order to define substitution following Figure 2, we need first to designate which of the boxes in the network N is going to serve as the dotted box. We therefore define an *interfaced network* as a triple (B, E, a), where (B, E) is a network and $a \in B$. Given $I = (B, E, a)$ and a network N', it one naturally defines substitution via join, $N'[f \to I] = N'[f; a]I$. However this is not correct as it would substitute the same network everywhere; we need to create copies of B. In addition, to deal with the kind of substitution we want to achieve in Figure 1, we need to add labels to boxes. These can be achieved, using the hierarchical identifier scheme used in (Alur et al. 2005). We do not have the space to describe details.

It is also tempting to define N' as *finer-grained* than N if N' can be obtained from N by a series of substitutions. Consider the interfaced network $I = (\{a\}, \{(a^{\to 1}, a^{\leftarrow 1}), (a^{\to 1}, a^{\leftarrow 2})\}, a)$. the substitution $N'[f \to I]$, which should be finer-grained than N'contains *fewer* boxes than N'. One might accept this; one might ban boxes with loops such as $(p^{\to 1}, p^{\leftarrow 1})$ or one might require substitution to leave some other kind of trace (provenance of provenance). Again our discussion is limited by space.

Although we have not given a complete description of granularity, we believe that substitution and join provide the machinery for constructing a hierarchy of provenance representations. Relevant problems for provenance, such as reachability, have efficient algorithms in this setting (Alur et al. 2005; Bao et al. 2012).

3. Provenance in distributed architectures

There is what appears to be a common situation which the authors have encountered in connection with research analyses of biological and clinical data when one has distributed processing and one wants to maintain an accurate provenance record. It is of interest because it calls for the use of join rather than substitution. Consider two long-running processes A and B communicating with each other while recording individual provenance. Hence, processes A and B maintain local provenance graphs, and A represents B as an (external) activity V_B and B represents A as V_A, as illustrated in Figure 3.

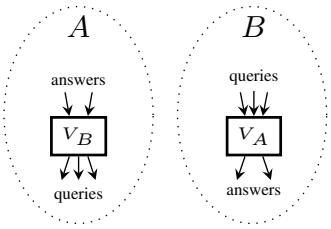

Figure 3. Provenance of communicating processes

One such situation is where we have a scientific research team, that requires, for example, clinical data to be fed into some workflow. The provider of such data could be a hospital, which performs patient anonymization, data cleaning and preliminary aggregation before delivering it to the research team. Both parties keep a provenance record: A for the research team and B for the provider, but for obvious reasons, neither side wants to reveal their provenance record to the other party so they respectively represent each other as "black box" activities V_B and V_A. Now the research team detects that there may be a patient at risk. To determine that patient someone needs to combine the provenance records of A and B.

In another application one could think of A as a database server and B as a client program. To be able to diagnose a problem found in the results of a series of queries from B by analysing its provenance, one should be able to query the provenance of the server side A, which requires a representation of the joint provenance of A and B.

From both these examples, it is clear that the operation we need for forming the combined provenance is the join $A[V_B; V_A]B$. Moreover we do not want (as in substitution) to copy any boxes. Suppose A also includes a box W_C that represents its interaction with a provenance graph C which includes a corresponding box W_A. It is entirely possible that B and C share common substructures and we would want this to be represented in the combined provenance $(A[V_B; V_A]B)[W_C, W_A]C$.

From a high-level perspective, we believe that a minimal solution for representing the provenance of communicating processes should provide: (a) a representation of messages to which each party has access – together with the relationship between the message and its content; (b) a representation, within each individual provenance graph, of the processes with which it communicates; (c) a way of attributing ownership of each component of the result to the parties involved in the communication; and (d) formal guaran-

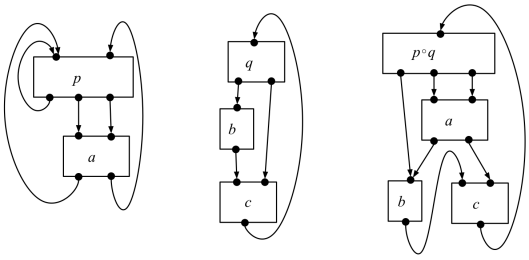

Figure 4. Serial composition

tees that the result of provenance composition accurately captures the full provenance of the distributed system. The representation we have proposed goes some way towards satisfying these.

4. Parallel and serial composition

Although we motivated the technical development with the practical need for both a substitution and a join operation, we should now confess that we do not believe that join is a fundamental operation. There are more fundamental operations of *serial* and *parallel composition* of boxes. To summarise: serial composition takes an (m, n)-box and a (l, m)-box and combines them into a (l, n)-box; serial composition combines a (m_1, n_1)-box and a (m_2, n_2)-box to produce a $(m_1 + m_2, n_1 + n_2)$-box. Intuitively, parallel and serial composition respectively stack boxes vertically and horizontally. Figure 4 shows the serial composition of p, a $(2, 3)$-box, with q a $(1, 2)$-box to produce a $(1, 3)$-box, which we have identfied by $p \circ q$.

These definitions have some intriguing consequences. First, join can be implemented "on the nose" (preserving box identities) using serial composition. Second any network can be constructed, to within isomorphism, using parallel and serial composition and the basic building blocks of the form $(\{p, q\}, \{(p^{\rightarrow 1}, q^{\leftarrow 1})\})$. Compare this with series-parallel *graphs* which are highly restricted. It has been observed (Bao et al. 2009; Cohen-Boulakia et al. 2014) that series-parallel graphs are useful in the practical analysis of workflows. It is an intriguing possibility that this research could be applied, with greater effect, to more general networks as we have described them. We hope to report on both of these in the future.

Finally, while it is a simple mattter to introduce box labels, edge labels may also be needed, and they are more challenging. If edge labels represent data, as they do in (Davidson et al. 2011) then maybe we want a condition in Defn. 1 that the excision fails if the labels do not agree. However, if as is suggested by some provenance work, edge labels may be composed, we may need some kind of algebra in determining the edge labels introduced by excision.

5. Conclusions

We have put forward a model of networks that allows for the synthesis of provenance and workflow representations from simpler components. In addition to investigating the use of serial and parallel composition, as we have just indicated, there is the obvious question of how we can incorporate these ideas into PROV. This is something we are currently investigating in the hope that we can produce a simple recipe for combining the PROV graphs as we have suggested in Section 3. The problem is non-trivial because our current proposal is "bare bones" and, to relate it to schema of PROV requires, among other things, encoding port identifiers. Although PROV nodes do not have ports, we should note that the PROV model allows for arbitrary key-value pairs to be attached to edge relations, so it is possible to represent this information in PROV, though this may not be the most natural solution.

Representation on PROV is but one of a number of open issues. Nevertheless we hope that we have made a convincing argument that in order fully to understand granularity and in order to make better use of provenance records, we need combinatory operations on provenance graphs.

Acknowledgements. These ideas originated from discussions with Paolo Missier on some work he had been doing with provenance abstraction (Missier et al. 2014). We are grateful to Paolo, Sebastian Maneth, Thomas Heinis, Kousha Etessami, Luc Moreau and to the referees for very useful input. This work was funded in part by the UK EPSRC SOCIAM project.

References

R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818, 2005.

Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on pig: Enabling database-style workflow provenance. *Proceedings of the VLDB Endowment*, 5(4):346–357, 2011.

Z. Bao, S. Cohen-Boulakia, S. B. Davidson, A. Eyal, and S. Khanna. Differencing provenance in scientific workflows. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 808–819. IEEE, 2009.

Z. Bao, S. B. Davidson, and T. Milo. Labeling workflow views with fine-grained dependencies. *Proc. VLDB Endow.*, 5(11):1208–1219, July 2012. ISSN 2150-8097. doi: 10.14778/2350229.2350240.

S. Cohen-Boulakia, J. Chen, P. Missier, C. Goble, A. R. Williams, and C. Froidevaux. Distilling structure in taverna scientific workflows: a refactoring approach. *BMC bioinformatics*, 15(1):1, 2014.

S. B. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. In *Proceedings of the 30th ACM PODS 2011*, pages 175–186, 2011.

F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement, graph grammars. In *Handbook of Graph Grammars*, pages 95–162, 1997.

J. Engelfriet. Handbook of formal languages, vol. 3. chapter Context-free Graph Grammars, pages 125–213. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

A. Gehani, M. Kim, and T. Malik. Efficient querying of distributed provenance stores. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 613–621, 2010.

P. Missier, J. Bryans, C. Gamble, V. Curcin, and R. D. Mercaderes. ProvAbs: model, policy, and tooling for abstracting PROV graphs. *CoRR*, abs/1406.1998, 2014.

L. Moreau and P. T. Groth. *Provenance: An Introduction to PROV*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2013.

L. Moreau and P. Missier. PROV-DM: The PROV Data Model. `http://www.w3.org/TR/prov-dm/`, April 2013.

G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, 1997. World Scientific. ISBN 9810228848.