A framework for provenance-preserving history distribution and incremental reduction

Lluch Lafuente, Alberto

Published in:
Models, Languages, and Tools for Concurrent and Distributed Programming

Link to article, DOI:
10.1007/978-3-030-21485-2_26

Publication date:
2019

Document Version
Peer reviewed version

A Framework for Provenance-preserving History Distribution and Incremental Reduction*

Alberto Lluch Lafuente

Technical University of Denmark
{albl}@dtu.dk

Abstract. Provenance properties help asses the level of trust on the integrity of resources and events. One of the problems of interest is to find the right balance between the expressive power of the provenance specification language and the amount of historical information that needs to be remembered for each resource or event. This gives rise to possibly conflicting objectives relevant to integrity, privacy, and performance. Related problems are how to reduce historical information in a way that the provenance properties of interest are preserved, that is suitable for a distributed setting, and that relies on an incremental construction. We investigate these problems in a simple model of computation where resources/events and their dependencies form an acyclic directed graph, and computation steps consist of addition of new resources and of provenance-based queries. The model is agnostic with respect to the actual provenance specification language. We present then a framework, parametric on such language, for distributing, and incrementally constructing reduced histories in a sound and complete way. In the resulting model of computation, reduced histories are computed incrementally and queries are tested locally on reduced histories. We study different choices for instantiating the framework with concrete provenance specification languages, and their corresponding provenance-preserving history reduction techniques.

Keywords: Provenance · Integrity · Concurrency Theory · Temporal Logics · Minimisation

1 Introduction

Integrity is one of the key security goals in information security. Integrity techniques aim at assessing and controlling the level of trust of resources and events. Information about the history of a resource or event, e.g. who created it, how was it derived from other resources or events, etc., is often called *provenance* or *lineage*, and can be of great support for integrity mechanisms. This paper is motivated by the need to efficiently check integrity properties of resources or events based on their history. A typical scenario of interest are distributed

* This work has been supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu).

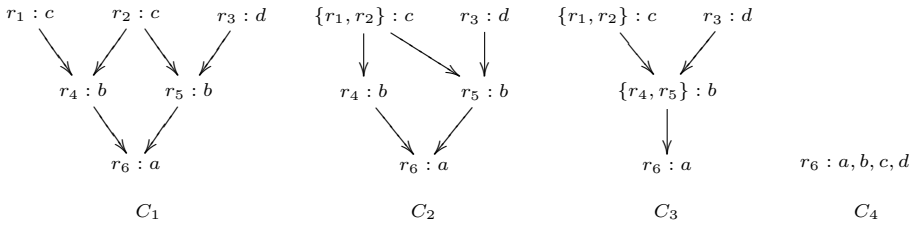

Fig. 1. Four configurations: C_1, C_2, C_3 and C_4

ledgers where transactions and their casual dependencies form an acyclic graph, and where one would need to assess the integrity of a transaction t based on the transactions it depends on: *Is there any suspicious transaction t' in the history of t? Is there a trustworthy transaction between t and t' that would provide some guarantees on the solidity of t?* Another typical scenario of interest are repositories of linked data: *Was data item x freshly created? Or was x produced by combining trustworthy data items y and z? Are y and z related, e.g. derived from disjoint sources?*

Provenance and expressivity: examples. Several languages can be used to express properties of the history of resources or events. In the area of databases, for example, provenance query languages have been studied for a long time, giving rise to variations of SQL and query languages for graphs (see e.g. [16,5,3,6] and the references therein). In this paper we take a different perspective based on models and techniques from the area of concurrency theory. From now on, we will talk about resources without the implication that we focus on data items or that we rule out events. We will start illustrating some basic choices for a provenance query language based on well-known models from concurrency theory with the examples of Figure 1. The figure illustrates four different configurations, where resources $r_1, r_2, \{r_1, r_2\} \dots$ are annotated with their atomic observable properties $(a, b, c, \dots)$, and are related to each other with arrows ($r \to r'$ denotes that r is a direct ancestor of r').

Four natural options for a provenance query language arise if one considers histories as *sets*, *strings*, *trees*, or *graphs* over the observable properties. Typically, those options would provide an increasing observation power: a string language would allow to observe the *order* of properties in the history, a tree language would in addition allow to observe the *merging* of resources, and a graph-based language would in addition allow to observe the *forking* of resources, and in general the entire structure of the history. To illustrate this, assume a scenario where the observable properties $a, b, c \dots$ of resources denote the agents that created the resources, respectively Alice, Bob, Charlie, ... In such scenario, examples of provenance properties in the above mentioned cases could be:

(P1) *Was the resource of interest influenced by Alice, Bob, Charlie, and Dave?*

(P2) *Was the resource of interest created by Alice and influenced by Charlie through Bob or by Dave through Bob?*

(P3) *Was the resource of interest created by Alice and influenced by resources created by Charlie and Dave that then Bob combined?*

(P4) *Was the resource of interest created by Alice and influenced by resources created by Bob, which in turn depended on the same resource created by Charlie?*

Let us now illustrate the effect of the expressive power of the provenance language on a concrete example, namely the history of r_6 in the configurations in Figure 1. An example of a set-based language would be a specification language to check atomic observables of r_6 and its entire genealogy (its direct and indirect ancestors). In this case, the observable history of r_6 would be the same in all four configurations. Indeed we could see C_4 as the minimised history of r_6 in C_1–C_3, where all ancestors have been collapsed into r_6. Examples of string-based languages could be based on any language to check the linear histories of resources (from the resource backwards through ancestors), for example based on regular expressions or linear-time logic (LTL) over sets of observables. In this case, the observable history of r_6 would be the same in C_1–C_3 (i.e. the set of strings $\{\{a\}\{b\}\{c\}, \{a\}\{b\}\{d\}\}$) but different in C_4 (i.e. the set of strings $\{\{a, b, c\}\}$). Similarly as before we could see C_3 as the minimised history of r_6 in C_1–C_2, where resources r_1, r_2 and r_4, r_5 have been collapsed. An example of a tree language could be computation-tree logic (CTL), interpreted on the transition system obtained from the (backwards) reachable subgraph starting from the resource of interest. In this case, C_1–C_2 would be indistiguishable but distinguishable from C_3, essentially since r_4 and r_5 have different dependencies. Similarly as before we could see C_2 as the minimised history of r_6 in C_1, where resources r_1, r_2 have been collapsed. Finally, we could decide to observe the full graphical structure of the history. In that case, all four configurations would be distinguishable and there would be no room for reductions.

Reductions, distribution and incrementality. As we have seen, the choice of the provenance language implicitly induces a notion of abstract history. Such abstraction can be exploited to reduce concrete histories, i.e. represent them in a more compact way. For example, C_1 in Figure 1 can be reduced to C_2 if we consider computation trees up to bisimulation, to C_3 if we consider linear histories (trace equivalence), and C_4 if we are interested just in the set of reachable properties. In many scenarios such configurations are continuously growing as new resources or events are added. Being able to build new reduced histories *incrementally* is desirable for the sake of scalability. As an orthogonal aspect, it may not be feasible or desirable in some situations to implement configurations as *global* structures. It may indeed be convenient or even mandatory to implement them as a *distributed* structure, which, on the other hand, may pose challenges in terms of communication or performance when checking provenance properties. Distributing local histories can help address some of those issues. Figures 2–4 illustrate distributed representations of the local histories (based on trees, strings

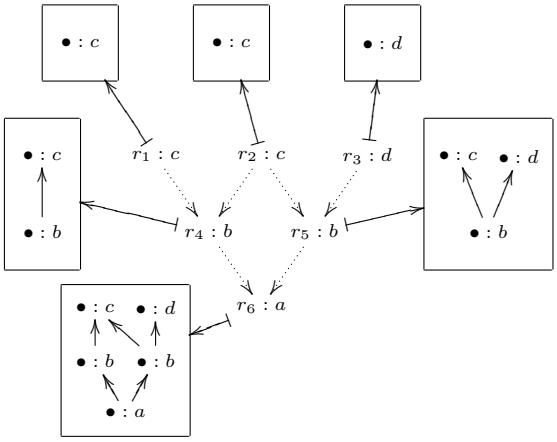

Fig. 2. Distributed configurations with minimised tree-based histories

and sets, respectively) for configuration C_1 of Figure 1. In the figures, each resource is mapped via $\mapsto$ arrows to its local history (enclosed in a square box).

Research questions and contribution. The main research question under consideration in this paper is: *Can we define a framework that is parametric with respect to the provenance-specification language $\mathcal{L}$ and that allows us to distribute and compress the history of resources in way that is (1) sound and complete w.r.t. to $\mathcal{L}$; and (2) incremental, i.e. possibly exploiting the already computed compressed histories?* We answer to the above research question by investigating this problem in a simple model of computation where new resources (with their dependencies) can be added, and can be subject to provenance queries (Section 2). We present in Section 3 a framework for history distribution and incremental reduction that is parametric with respect to the actual provenance specification language $\mathcal{L}$, and we study in Section 4 several choices for $\mathcal{L}$, and the corresponding provenance-preserving history reduction techniques. Section 5 concludes the paper.

A tribute to Rocco De Nicola. This paper is a contribution to the Festschrift that celebrates Rocco De Nicola's 65th birthday. I have had the pleasure and the honour to work closely with Rocco for several years, an experience that has inspired both my professional career and my personal life. This paper contains several elements related to Rocco's scientific contribution to models, languages and applications of concurrency theory. First, the model of programmable configurations in Section 2 has been inspired by coordination models investigated by Rocco (see [9] for a survey of Rocco's work in this field), in particular locality-centred process calculi [8] and attribute-based interactions [11,2]. Such works have motivated me to look for an unconventional model of coordination, based on provenance. Second, the consideration of modal logics as provenance languages, and history reductions based on semantic equivalences relate to Rocco's

contributions to those fields [10,13,7,14,12]. Even if Section 4 does not directly apply Rocco's own contributions, his search for generalisations and establishing relations between semantic equivalences and logics have motivated the search for a general framework (Section 3) where different semantic notions of provenance could fit. Last, the application domain of information security is very dear to Rocco and has been present in his work in several occasions, from the access and information flow control features of KLAIM [8] to his white paper on the future of Cybersecurity in Italy [18].

2 A Simple Model of programmable Global Configurations

Configurations. Let $\mathcal{R}$ denote a universe of resource identifiers (resources, for short) that we range over by $r, r_1, \ldots$. A *(global) configuration* in our model is essentially a directed acyclic graph with resources as nodes. Resources have associated values in some domain $\mathcal{V}$ and are labelled over a set Σ of atomic observable properties on those values, through a function $\pi : \mathcal{V} \to 2^{\Sigma}$.

Definition 1 (configuration). *A* configuration *is a tuple $\langle R, v, L, < \rangle$ where R is the set of nodes/resources, $v : R \to \mathcal{V}$ is a mapping of resources to their actual values, $L : R \to 2^{\Sigma}$ is a mapping of resources to observable properties, $<\subseteq R \times R$ is the set of edges/dependencies, and such that $<$ is acyclic.*

Figure 1 illustrates four configurations, where values are not included. We denote the set of all configurations by $\mathcal{C}$. For a configuration C we will often use R, v, L, and $<$, assuming implicitly that $C = \langle R, v, L, < \rangle$ when clear from the context. Note that $r < r'$ is illustrated as an edge $r \to r'$. Function $pre^*_C : R \to 2^R$ denotes the genealogy of a resource in C (its direct and indirect ancestors, including itself), i.e. $pre^*_C(r) = \{r' \in R \mid r' = r \text{ or } \exists r'' \in R. r'' < r \wedge r' \in pre^*_C(r'')\}$. With $C_{|r}$ we will denote the sub-configuration of C formed just by r and its predecessors (i.e. the subgraph of $<$ backwards-reachable from r), formally $C_{|r} = \langle pre^*_C(r), v_{|pre^*_C(r)}, L_{|pre^*_C(r)}, <_{|r} \rangle$ such that $r' <_{|r} r''$ iff $r' < r''$ and $\{r', r''\} \subseteq pre^*_C(r)$.

Provenance properties. We use $\mathcal{L}$ to denote the provenance specification language, and $\phi \in \mathcal{L}$ to range over provenance properties. Given a configuration $C = \langle R, v, L, < \rangle$ and a resource r, we use $r, C \models \phi$ to denote that r satisfies ϕ in C. The actual semantics of $\models$ depends on the choice of $\mathcal{L}$. For a query language $\mathcal{L}$ we will consider satisfaction relations $\models \subseteq \mathcal{M} \times \mathcal{L}$ for several model domains beyond $\mathcal{R} \times \mathcal{C}$. For two model domains $\mathcal{M}_1$ and $\mathcal{M}_2$, the relation $\equiv_{\mathcal{L}} \subseteq \mathcal{M}_1 \times \mathcal{M}_2$ is defined in the usual way, i.e. $M_1 \equiv_{\mathcal{L}} M_2$ whenever for all $\phi \in \mathcal{L}$ we have that $M_1 \models \phi$ iff $M_2 \models \phi$. We restrict to our attention to languages $\mathcal{L}$ that are able to observe atomic properties, and not actual values. The actual information observable on resources is defined by function π, that can be understood as a privacy-protecting function, hiding actual information about resources and just exposing information needed to assess their integrity.

Global Computations. Computations in our model of *programmable configurations* are of the form

$$C \xrightarrow{\lambda \triangleright \rho} C'$$

denoting that configuration C reacts to action λ by transforming into C' and producing ρ.

We consider first a simple model of *internal* computations with two actions out and rd, respectively used to add fresh resources (and their explicit dependencies), and to retrieve them with provenance-based queries.

More in detail, a computation based on action $\mathsf{out}(R' < u)$ adds a fresh resource r with value u, and dependencies on a subset $R' \subseteq R$ of the current set of resources R.

$$\frac{r \notin R \qquad R' \subseteq R}{\langle R, v, L, < \rangle \xrightarrow{\mathsf{out}(R' < u) \triangleright r} \langle R \cup \{r\}, v \cup \{v \mapsto u\}, L \cup \{r \mapsto \pi(u)\}, < \bigcup_{r' \in R'} \{r' < r\} \rangle}$$

Computations based on rd actions return a resource r whose history satisfies the provenance property ϕ, provided one such resource exists:

$$\frac{r \in R \qquad r, \langle R, v, L, < \rangle \models \phi}{\langle R, v, L, < \rangle \xrightarrow{\mathsf{rd}(\phi) \triangleright r} \langle R, v, L, < \rangle}$$

Internal computations expose actual resources as part of the interactions and demand causal dependencies to be specified *explicitly*. In many situations resources need to be hidden, and causal dependencies need to be *implicitly* computed. To deal with this we introduce *external* computations in our model of programmable configurations with two actions: put and get.

In particular, $\mathsf{put}(f(\phi_1, .., \phi_n))$ is an action that, given a function $f : \mathcal{V}^* \to \mathcal{V}$ and n provenance formulas $\phi_1, .., \phi_n$ adds a new resource obtained by applying f to the values of resources $r_1, .., r_n$, respectively satisfying $\phi_1, .., \phi_n$ and storing the corresponding dependencies. Such operation can be easily defined as a combination of internal computations:

$$\frac{C \xrightarrow{\mathsf{rd}(\phi_i) \triangleright r_i} C \text{ for } i \in \{1, .., n\} \qquad C \xrightarrow{\mathsf{out}(\{r_1, .., r_n\} < f(v(r_i), .., v(r_n))) \triangleright r} C'}{C \xrightarrow{\mathsf{put}(f(\phi_1, .., \phi_n)) \triangleright f(v(r_i), .., v(r_n)))} C'}$$

The query operation is very much like rd but it returns the value of the resource instead of the resource identifier itself:

$$\frac{C \xrightarrow{\mathsf{rd}(\phi) \triangleright r} C}{C \xrightarrow{\mathsf{query}(\phi) \triangleright v(r)} C}$$

3 Distributed Configurations with Abstract Histories

We now introduce *distributed configurations* where the history of each resource is localised and possibly reduced into some abstract history, from a domain $\mathcal{H}$ of abstract histories, for which the satisfaction relation $\models \subseteq \mathcal{H} \times \mathcal{L}$ is defined.

Definition 2 (distributed configurations). *A* distributed configuration *is a tuple* $\langle R, v, L, h\rangle$ *where* R *is a set of resources,* $v : R \to \mathcal{V}$ *is a mapping of resources to their actual values,* $L : R \to 2^{\Sigma}$ *maps resources into their observable properties, and* $h : R \to \mathcal{H}$ *is a function that maps each resource to its own abstract history.*

Figures 2–4 illustrate three examples of distributed configurations where, respectively, the domain of abstract histories are (implicitly) trees, strings, and sets of atomic properties, as we shall explain in detail in Section 4. Abstract histories are enclosed in square boxes, and the mapping h is denoted with $\mapsto$ arrows. Dotted arrows are used to represent the original dependencies. We denote the set of all distributed configurations by $\mathcal{D}$.

Let $\alpha : \mathcal{R} \times \mathcal{C} \to \mathcal{H}$ be a function that maps concrete histories (i.e. pairs of resources and configurations) into abstract histories. We say that α is $\mathcal{L}$-*preserving* if and only if for all configurations $C = \langle R, v, L, <\rangle \in \mathcal{C}$, all resources $r \in R$ it holds $r, C \equiv_{\mathcal{L}} \alpha(r, C)$. We will assume that all languages $\mathcal{L}$ of interest are such that $r, C \equiv_{\mathcal{L}} r, C_{|r}$, i.e. their formulas allow to observe just the history of events.

History abstraction functions can be lifted to a mapping from global configurations to distributed configurations.

Definition 3 (history distribution function). *Let* $C = \langle R, v, L, <\rangle$ *be a configuration and* $\alpha : 2^{\mathcal{R}\times\mathcal{R}} \to \mathcal{H}$ *be a history abstraction function. Then,* $\alpha(C)$ *is the distributed configuration* $\langle R, v, L, \{r \mapsto \alpha(r, C_{|r}) \mid r \in R\}\rangle$.

Note that for the trivial case when $\mathcal{H}$ is $\mathcal{R} \times \mathcal{C}$, there is a trivial choice for α, namely the function $\lambda r, C = (\{r\}, C_{|r})$ that would map each resource to just own history. Other, more interesting cases could be mapping each resource into a Σ-labelled transition system, finite-state automaton over Σ or just a subset of Σ, as suggested in Figures 2–4, respectively. These cases will be discussed in detail in Section 4.

Another component of our framework are *history update functions* of the form $\oplus : \mathcal{R} \times 2^{\Sigma} \times 2^{\mathcal{H}} \to \mathcal{H}$. These are functions that, given a resource r, its observations A, and a set H of abstract histories (typically, the direct ancestors of r), produce an abstract history for r.

We say that $\oplus$ *is* $\mathcal{L}$*-preserving for* α if and only if for all configurations $C = \langle R, c, L, <\rangle \in \mathcal{C}$, all resources $r \in R$ and all provenance formulas $\phi \in \mathcal{L}$ it holds

$$\alpha(r, C_{|r}) \equiv_{\mathcal{L}} (r, L(r)) \oplus \{\alpha(r', C_{|r'}) \mid r' < r\}$$

that is, abstract histories obtained globally are $\mathcal{L}$-equivalent to abstract histories computed incrementally with $\oplus$.

Distributed Computations. The semantics for distributed configurations is similar to that of global configurations, but based on local histories. In particular, a adding resource with out is now defined by rule

$$\frac{r \notin R \qquad R' \subseteq R}{\langle R, v, L, h\rangle \xrightarrow{\mathsf{out}(R' < u) \triangleright r} \langle R \cup \{r\}, v \cup \{r \mapsto u\}, L \cup \{r \mapsto \pi(u)\}, h \cup \{r \mapsto (r, \pi(u)) \oplus \bigcup_{r' \in R'} h(r')\}\rangle}$$

In words: a configuration is enriched by adding a fresh resource r and computing its local, abstract history incrementally, based on the local, abstract history of its ancestors. Note that the computation is not entirely decentralised, as the preconditions still require global information, but part of that information (set R') is in any case required to compute the history of r incrementally.

Internal queries, instead, rely only on local information:

$$\frac{r \in R \qquad h(r) \models \phi}{\langle R, v, L, h\rangle \xrightarrow{\mathsf{rd}(\phi) \triangleright r} \langle R, v, L, h\rangle}$$

External computations with put and query are defined using the same combinations of rd and out operations with rules

$$\frac{D \xrightarrow{\mathsf{rd}(\phi_i) \triangleright r_i} D \text{ for } i \in \{1, .., n\} \qquad D \xrightarrow{\mathsf{out}(r_1, .., r_n < f(v(r_i), .., v(r_n))) \triangleright r} D'}{D \xrightarrow{\mathsf{put}(f(\phi_1, .., \phi_n)) \triangleright f(v(r_i), .., v(r_n))} D'}$$

and

$$\frac{D \xrightarrow{\mathsf{rd}(\phi) \triangleright r} D}{D \xrightarrow{\mathsf{query}(\phi) \triangleright v(r)} D}$$

respectively.

Provenance-preserving result. We define now a relation $\approx \subseteq \mathcal{C} \times \mathcal{D}$ that relates global configurations and distributed configurations over the same set of resources such that the concrete and abstract history of each resource is $\mathcal{L}$-equivalent.

Definition 4 (distribution relation). *The relation $\approx \subseteq \mathcal{C} \times \mathcal{D}$ is the set of all pairs $(\langle R, v, L, <\rangle, \langle R, v, L, h\rangle)$ such that $\forall r \in R.\ r, C \equiv_{\mathcal{L}} h(r)$.*

Note that if α is $\mathcal{L}$-preserving then, trivially, $C \approx \alpha(C)$ for all configurations $C \in \mathcal{C}$. The main property of our framework is that, if α is $\mathcal{L}$-preserving and $\oplus$ is $\mathcal{L}$-preserving for α, then $\approx$ is a bisimulation. As a consequence a global configuration C and its distributed version $\alpha(C)$ behave equivalently.

Theorem 1 (bisimilar distribution). *Let α be a history reduction function and $\oplus$ be a history update function such that (i) α is $\mathcal{L}$-preserving and (ii) $\oplus$ is $\mathcal{L}$-preserving for α, then $\approx$ is a bisimulation.*

Proof sketch. To prove that $\approx$ is a bisimulation we have to show that for every $C \in \mathcal{C}$ and every $D \in \mathcal{D}$ such that $C \approx D$ the following properties hold:

1. if $C \xrightarrow{\lambda \triangleright r} C'$ then $\exists D' \in \mathcal{D}. D \xrightarrow{\lambda \triangleright r} D'$ and $C' \approx D'$.
2. if $D \xrightarrow{\lambda \triangleright r} D'$ then $\exists C' \in \mathcal{C}. C \xrightarrow{\lambda \triangleright r} C'$ and $C' \approx D'$.

We prove property (1) by considering the cases for λ separately:

$[\lambda = \mathsf{out}(r_1, .., r_n < u)]$ Let $C = \langle R, v, L, < \rangle$. By definition of $\approx$, we know that $D = \langle R, v, L, h \rangle$ for some h such that $\forall r \in R.\ r, C \equiv_{\mathcal{L}} h(r)$.
The new global configuration

$$C' = \langle R \cup \{r\}, v \cup \{r \mapsto u\}, L \cup \{r \mapsto \pi(u)\}, < \bigcup_{i \in \{1..n\}} \{r_i < r\} \rangle$$

differs from C only in the addition of r, its properties and its dependencies. Clearly D can make the same choice for λ by selecting the same r, thus resulting in

$$D' = \langle R \cup \{r\}, v \cup \{r \mapsto u\}, L \cup \{r \mapsto \pi(u)\}, h \cup \{r \mapsto r \oplus h(r_1), .., h(r_n)\} \rangle$$

To show that $C' \approx D'$ we just need to prove that for the newly added resource r it holds

$$r, C' \equiv_{\mathcal{L}} (r, L(r)) \oplus h(r_1), .., h(r_n)$$

since the above property holds for all other resources as mentioned above. We proceed as follows:

$$\begin{array}{rl}
r, C' & \\
\equiv_{\mathcal{L}} & (\mathcal{L} \text{ observes histories only}) \\
r, C'_{|r} & \\
\equiv_{\mathcal{L}} & (\alpha \text{ is } \mathcal{L}\text{-preserving}) \\
\alpha(r, C'_{|r}) & \\
\equiv_{\mathcal{L}} & (\oplus \text{ is } \mathcal{L}\text{-preserving for } \alpha) \\
(r, L(r)) \oplus \alpha(r_1, C'_{|r_1}), .., \alpha(r_1, C'_{|r_1}) & \\
\equiv_{\mathcal{L}} & (C'_{|r_i} = C_{|r_i} \text{ for } i = 1..n) \\
(r, L(r)) \oplus \alpha(r_1, C_{|r_1}), .., \alpha(r_1, C_{|r_1}) & \\
\equiv_{\mathcal{L}} & (\oplus \text{ is } \mathcal{L}\text{-preserving for } \alpha) \\
(r, L(r)) \oplus h(r_1), .., h(r_n) &
\end{array}$$

We can conclude that $C' \approx D'$.

$[\lambda = \mathsf{rd}(\phi)]$ Note that in this case $C' = C$. From $C \xrightarrow{\mathsf{rd}(\phi) \triangleright r} C$ we know that $r, C \models \phi$. Since $C \approx D$ we know that $r, C \equiv_{\mathcal{L}} h(r)$ and hence it holds $h(r) \models \phi$. We have thus the computation step $D \xrightarrow{\mathsf{rd}(\phi) \triangleright r} D$. Trivially choosing D' to be D concludes our argument.

$[\lambda = \mathsf{put}(f(\phi_1, .., \phi_n))$ and $\lambda = \mathsf{query}(\phi)]$ These are actually derived operations, obtained by composition of the above two ones.

To prove property (2) we proceed similarly.

□

4 Instantiating the framework

We instantiate the framework in a set of examples, respectively based on interpreting histories as trees (thus observing merging, order, and atomic properties of resources), strings (thus observing order and atomic properties of resources only) and sets (thus just observing atomic properties of resources). Each instance consists of a specific choice for the domain $\mathcal{H}$ of abstract histories, the provenance language $\mathcal{L}$, the history reduction function α, and the incremental history update function $\oplus$. It is not the aim of this section to show the most efficient option for each case or to provide a comprehensive overview of all possibilities. Instead, we aim at illustrating the framework with well-known models computation from the area of concurrency theory and related ones.

4.1 Tree-based Provenance

As abstract histories we will consider finite trees over Σ, compactly represented as transition systems. We recall that a state-labelled transition system (a Kripke structure) is a tuple $\langle S, I, \rightarrow, AP, M\rangle$ such that S is a set of states, $I \subseteq S$ is set of initial states, $\rightarrow \subseteq S \times S$ is a transition relation, AP is a set of atomic propositions and $M : S \rightarrow AP$ is a mapping from states into subsets of atomic propositions. Transition systems compactly represent trees in the same manner of graphs: the trees of a transition system are obtained by unfolding the transition system.

Let $\mathcal{H}$ be $\mathcal{T}$, the set of all transition systems with resources as states, dependencies as transitions, Σ as atomic propositions and L as state labelling function. In particular, $TS(C, r) = \langle R, \{r\}, <^{-1}, \Sigma, L\rangle$ denotes the transition system representing the history of r in C, where r is the initial state and the transition relation is the inverse of the predecessor relation.[1]

A natural choice for $\mathcal{L}$ is then any logic to predicate over trees in a transition system. We choose CTL for our illustration purposes. For $\phi \in \mathsf{CTL}$ we define the satisfaction relation for resources indirectly via a transformation of the history into its transition system representation:

$$r, C \models \phi \text{ iff } TS(C, r) \models \phi$$

In our example, the property (P3) *Was the resource of interest created by Alice and influenced by resources created by Charlie and Dave that then Bob combined?* can be expressed in CTL as $a \wedge \mathbf{EF}(b \wedge \mathbf{EF}c \wedge \mathbf{EF}d)$.

[1] Note that we do not introduce self-loop transitions in leaf states/resources. This allows formulae to observe whether a resource has dependencies. Alternatively, loops could have been introduced as usual in CTL semantics. In that case, the ability to observe absence of dependencies can be obtained with a dedicated predicate.

Let $bmin : \mathcal{T} \to \mathcal{T}$ be a bisimulation-minimisation algorithm that transforms a transition system T into a smallest bisimilar one $bmin(T)$. Let α be defined as $\alpha(r, C) = bmin(TS(r, C_{|r}))$, that is the function that first transforms a history into a transition system and then minimises it. Clearly, α is $\mathcal{L}$-preserving since bisimilar transition systems are CTL-equivalent.

For incremental history construction we define the following function $\oplus$

$$
\begin{aligned}
&(r, A) \oplus \uplus_{i\in\{1..n\}}\langle R_i, r_i, \to_i, \Sigma, L_i\rangle \\
&= bmin(\langle\{r\} \uplus_{i\in\{1..n\}} R_i, \{r\}, \uplus_{i\in\{1..n\}}\{r \to r_i\} \uplus \to_i, \Sigma, \{r \mapsto A\} \uplus_{i\in\{1..n\}} L_i\rangle)
\end{aligned}
$$

In words: we first build a new transition system that has r as initial state, and has a transition from r to the initial state r_i of each of the n transitions systems that represent abstract histories (that we assume disjoint to avoid the cumbersome notation of introducing fresh renamings). Such transition system is then minimised. We need to prove that $\oplus$ is CTL-preserving for $bmin$.

Lemma 1. *The incremental history construction function $\oplus$ is CTL-preserving for bmin.*

Proof sketch. We need to prove

$$bmin(TS(r, C_{|r})) \equiv_{\mathsf{CTL}} (r, L(r)) \oplus \{bmin(TS(r', C_{|r'})) \mid r' < r\}$$

we proceed as follows

$$
\begin{aligned}
& bmin(TS(r, C_{|r})) \\
\equiv_{\mathsf{CTL}} & \text{ bisimulation preserves CTL} \\
& TS(r, C_{|r}) \\
\equiv_{\mathsf{CTL}} & \text{ (*)} \\
& (r, L(r)) \oplus \{TS(r', C_{|r'}) \mid r' < r\} \\
\equiv_{\mathsf{CTL}} & \text{ bisimulation preserves CTL} \\
& (r, L(r)) \oplus \{bmin(TS(r', C_{|r'})) \mid r' < r\}
\end{aligned}
$$

The main idea behind (*) is that $(r, L(r)) \oplus \{TS(r', C_{|r'}) \mid r' < r\}$ essentially corresponds to an unfolding of $TS(r, C_{|r})$ into a tree-shaped transition system, and that unfolding of transitions systems preserves bisimulation and hence CTL. □

Instantiating Theorem 1 in the above described setting provides the desired result: if we choose CTL as provenance query language we can distribute and incrementally compute histories without loosing information by resorting to bisimulation minimisation algorithms. Figure 2 illustrates a distributed representation of the transition system based local histories for configuration C_1 of Figure 1. In the figure, the initial state of each transition system is its unique root.

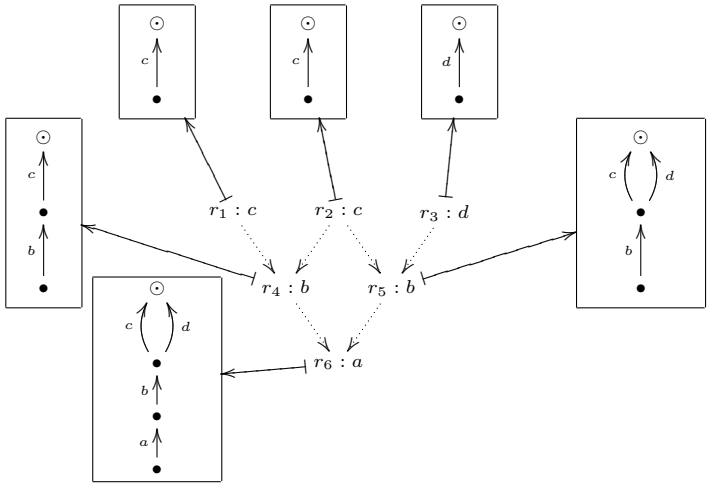

Fig. 3. Distributed configurations with minimised string-based histories

4.2 Linear Provenance

We consider now abstract histories as sets of finite strings over 2^{Σ}. A natural way to compactly represent a set of strings is with a non-deterministic finite automaton (NFA). Recall that an NFA is a tuple $\langle Q, q_0, F, A, \rightarrow \rangle$, where Q is a set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a final set of states, A is an alphabet of symbols, and $\rightarrow \subseteq 2^{S \times \Sigma \times S}$ is a transition relation. Let us denote with $\mathfrak{L}(B)$ the language of the NFA B.

In particular, for a configuration C and a resource r, its abstract history can be defined as the automaton $\mathit{NFA}(C, r) = \langle R \uplus \circ, r, \{\circ\}, 2^{\Sigma}, \rightarrow \rangle$ where $\rightarrow$ is the set of transitions $\{r' \xrightarrow{L(r')} r'' \mid r'' < r'\} \cup \{r' \xrightarrow{L(r')} \circ \mid \neg \exists r''.r'' < r'\}$. In words, the set of states of the automaton are the resources of C plus the unique final state $\circ$, and transitions correspond to the observable properties in the departing resource of the transition.

Let $\mathcal{A}$ denote the set of all such NFAs and let $\mathcal{H}$ be $\mathcal{A}$. A natural choice for $\mathcal{L}$ is then any language for finite strings, for example regular expressions. For a regular expression ϕ we define the satisfaction relation for resources indirectly via a transformation of the history into its NFA representation:

$$r, C \models \phi \text{ iff } \mathfrak{L}(\mathit{NFA}(C, r)) \subseteq \mathfrak{L}(\phi)$$

in words, the history of a resource r in C satisfies the property ϕ if, for each original ancestor r', the path from r to r' is accepted by ϕ.

In our example, the property (P2) *Was the resource of interest created by Alice and influenced by Charlie through Bob, or by Dave through Bob?* can be formalised by the regular expression $\{a\}(.^*\{b\}.^*\{c\} + .^*\{b\}.^*\{d\})$.

Let $\mathit{min} : \mathcal{A} \rightarrow \mathcal{A}$ be an NFA minimisation algorithm that transforms an NFA into the smallest NFA accepting the same language. Let α be defined as

$\alpha(r, C) = min(NFA(r, C_{|r}))$, that is the function that first translates a concrete history into an NFA and then minimises it. Clearly, α is $\mathcal{L}$-preserving since NFA minimisation preserve regular language equivalence.

For incremental history construction we define the following function $\oplus$.

$$r \oplus \uplus_{i\in\{1..n\}}\langle R_i, r_i, \rightarrow_i, \Sigma, L\rangle$$
$$= min(\langle\{r\} \uplus_{i\in\{1..n\}} R_i, r_i, \uplus_{i\in\{1..n\}}\{r \rightarrow r_i\}\uplus \rightarrow_i, \Sigma, L\rangle)$$

In words: we first build an NFA that has r as initial state, and has a transition from r to the initial state r_i of each of the n NFAs that represent abstract histories. The resulting NFA is then minimised with *min*. Proving that $\oplus$ is $\mathcal{L}$-preserving for α can be done along the lines of 1.

Instantiating Theorem 1 allows us to use a provenance query language based NFA so to distribute and incrementally compute reduced histories based on NFA minimisation, without loosing information. Figure 3 illustrates a distributed representation of the NFA-based local history for configuration C_1 of Figure 1. In the figure, the initial state of each automaton is its unique root.

4.3 Set-based Provenance

Let $\mathcal{H}$ be 2^{Σ}, i.e. abstract histories are just subsets of Σ representing which properties the resource has or has been influenced by. For a configuration C and a resource r we can interpret the history of r with the following function *inf* to compute the influencing properties of r:

$$inf(r, C) = L(r) \odot \{inf(r') \mid r' < r\}$$

The function is parametric on a function $\odot$ to combine properties of resources with the influencing properties of its ancestors. Taking $h \odot H$ to be $h \cup \bigcup H$ would provide all properties that r *or any* of its ancestors has. If instead we take $h \odot H$ to be $h \cap \bigcap H$ we would consider the influencing properties as those that r *and all* of its ancestors have.

A natural choice for the provenance language $\mathcal{L}$ could be just propositional logic over Σ

$$r, C \models \phi \text{ iff } inf(r) \models \phi$$

The example property (P1) *Was the resource of interest influenced by Alice, Bob, Charlie, and Dave?* would be just the proposition $a \wedge b \wedge c \wedge d$.

We can now do two trivial choices for α and $\otimes$. First, let α be the function *inf*, which is trivially $\mathcal{L}$-preserving. Second, the incremental history construction function $\oplus$ is defined based on $\otimes$:

$$r \oplus \{A_1, .., A_n\} = L(r) \cdot \{A_1, .., A_n\}$$

which can easily be show to be $\mathcal{L}$-preserving for α. We can then instantiate Theorem 1 to obtain the desired result: we can distribute and incrementally

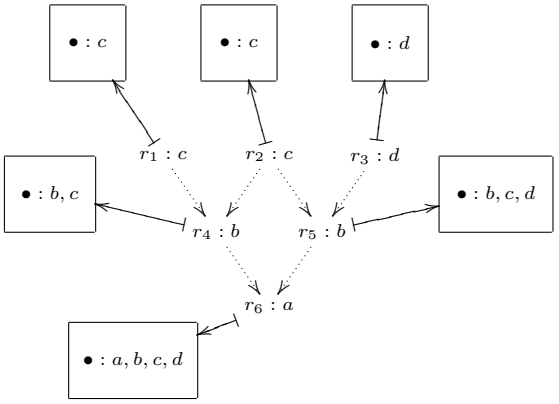

Fig. 4. Distributed configurations set-based histories

compute histories on the above described mechanism without loosing information. Figure 4 illustrates a distributed representation of the local histories for configuration C_1 of Figure 1, where $\cdot$ is based on set union.

5 Conclusion

We have presented a basic framework to support provenance-based computations, with incremental history distribution and reduction. The main components of the framework are an abstract domain of histories $\mathcal{H}$, a provenance specification language $\mathcal{L}$, and two functions: α for history reduction, and $\oplus$ for incremental history construction, that are required to be sound and complete with respect to $\mathcal{L}$. We have illustrated the framework with a basic model of programmable configurations, and notions of abstract histories based on well-understood models of computation and information flow.

The approach we have presented can lead to deeper investigations in the future. First, in the examples we have used to illustrate our approach we have not considered efficiency as an issue. Efficient algorithms for minimisation of histories that exploit their acyclic structure could be to be identified. For example, in the case of bisimulations a partition refinement algorithm based on [17] could easily exploit the acyclic structure in the strategy to select the candidate partitions subject to a partition check. The basic model of computation we have used to illustrate the approach could lead to novel coordination languages, taking inspiration from locality-centred [8] and provenance-tracking [4] coordination languages, one could design a provenance-oriented coordination language where interactions depended on provenance properties. An interesting variant of the instances we have seen could be to consider histories up to stuttering. In the case of computation trees and bisimulations we could consider the corresponding notions of stuttering bisimulation minimisation algorithms [14,15]. For linear provenance, it would be also interesting to investigate provenance queries based on testing equivalences [10,1] or LTL variants.

References

1. Aceto, L., De Nicola, R., Fantechi, A.: Testing equivalences for event structures. In: Zilli, M.V. (ed.) Mathematical Models for the Semantics of Parallelism, Advanced School, Rome, Italy, September 24 - October 1, 1986, Proceedings. Lecture Notes in Computer Science, vol. 280, pp. 1–20. Springer (1986). https://doi.org/10.1007/3-540-18419-8_9, https://doi.org/10.1007/3-540-18419-8_9
2. Alrahman, Y.A., De Nicola, R., Loreti, M.: On the power of attribute-based communication. In: Albert, E., Lanese, I. (eds.) Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9688, pp. 1–18. Springer (2016). https://doi.org/10.1007/978-3-319-39570-8_1, https://doi.org/10.1007/978-3-319-39570-8_1
3. Altintas, I., Wang, J., Crawl, D., Li, W.: Challenges and approaches for distributed workflow-driven analysis of large-scale biological data: vision paper. In: Srivastava, D., Ari, I. (eds.) Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012. pp. 73–78. ACM (2012). https://doi.org/10.1145/2320765.2320791, https://doi.org/10.1145/2320765.2320791
4. Bodei, C., Degano, P., Ferrari, G.L., Galletta, L.: Tracing where iot data are collected and aggregated. Logical Methods in Computer Science **13**(3) (2017). https://doi.org/10.23638/LMCS-13(3:5)2017, https://doi.org/10.23638/LMCS-13(3:5)2017
5. Chavan, A., Huang, S., Deshpande, A., Elmore, A.J., Madden, S., Parameswaran, A.G.: Towards a unified query language for provenance and versioning. In: Missier, P., Zhao, J. (eds.) 7th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2015, Edinburgh, Scotland, UK, July 8-9, 2015. USENIX Association (2015), https://www.usenix.org/conference/tapp15/workshop-program/presentation/chavan
6. Davidson, S.B., Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: Wang, J.T. (ed.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008. pp. 1345–1350. ACM (2008). https://doi.org/10.1145/1376616.1376772, https://doi.org/10.1145/1376616.1376772
7. De Nicola, R.: Extensional equivalences for transition systems. Acta Inf. **24**(2), 211–237 (1987). https://doi.org/10.1007/BF00264365, https://doi.org/10.1007/BF00264365
8. De Nicola, R., Ferrari, G.L., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. **24**(5), 315–330 (1998). https://doi.org/10.1109/32.685256, https://doi.org/10.1109/32.685256
9. De Nicola, R., Ferrari, G.L., Pugliese, R., Tiezzi, F.: A formal approach to the engineering of domain-specific distributed systems. In: Serugendo, G.D.M., Loreti, M. (eds.) Coordination Models and Languages - 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings. Lecture Notes in Computer Science, vol. 10852, pp. 110–141. Springer (2018). https://doi.org/10.1007/978-3-319-92408-3_5, https://doi.org/10.1007/978-3-319-92408-3_5

10. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. **34**, 83–133 (1984). https://doi.org/10.1016/0304-3975(84)90113-0, https://doi.org/10.1016/0304-3975(84)90113-0
11. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. TAAS **9**(2), 7:1–7:29 (2014). https://doi.org/10.1145/2619998, https://doi.org/10.1145/2619998
12. De Nicola, R., Montanari, U., Vaandrager, F.W.: Back and forth bisimulations. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings. Lecture Notes in Computer Science, vol. 458, pp. 152–165. Springer (1990). https://doi.org/10.1007/BFb0039058, https://doi.org/10.1007/BFb0039058
13. De Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings. Lecture Notes in Computer Science, vol. 469, pp. 407–419. Springer (1990). https://doi.org/10.1007/3-540-53479-2_17, https://doi.org/10.1007/3-540-53479-2_17
14. De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation. J. ACM **42**(2), 458–487 (1995). https://doi.org/10.1145/201019.201032, https://doi.org/10.1145/201019.201032
15. Groote, J.F., Vaandrager, F.W.: An efficient algorithm for branching bisimulation and stuttering equivalence. In: Paterson, M. (ed.) Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings. Lecture Notes in Computer Science, vol. 443, pp. 626–638. Springer (1990). https://doi.org/10.1007/BFb0032063, https://doi.org/10.1007/BFb0032063
16. Holland, D.A., Braun, U.J., Maclean, D., Muniswamy-Reddy, K.K., Seltzer, M.I.: Choosing a data model and query language for provenance. In: Provenance and Annotation of Data and Processes: Proceedings of the 2nd International Provenance and Annotation Workshop (IPAW '08). Lecture Notes in Computer Sciene, vol. 5272. Springer (2008)
17. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987). https://doi.org/10.1137/0216062, https://doi.org/10.1137/0216062
18. Roberto Baldoni, Rocco De Nicola, P.P.: The future of Cybersecurity in Italy: Strategic focus areas. Laboratorio Nazionale di Cybersecurity, CINI - Consorzio Interuniversitario Nazionale per l'Informatica (2018), https://www.consorzio-cini.it/index.php/it/labcs-home/libro-bianco