A Provenance Tracking Model for Data Updates

Ciobanu, Gabriel; Horne, Ross

2012

Ciobanu, G., & Horne, R. (2012). A Provenance Tracking Model for Data Updates. Proceedings 11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA 2012), Electronic Proceedings in Theoretical Computer Science, 91, 31-44.

https://hdl.handle.net/10356/80955

https://doi.org/10.4204/EPTCS.91.3

A Provenance Tracking Model for Data Updates

Gabriel Ciobanu Ross Horne
Romanian Academy, Institute of Computer Science,
Blvd. Carol I, no. 8, 700505 Iaşi, Romania
gabriel@info.uaic.ro ross.horne@gmail.com

For data-centric systems, provenance tracking is particularly important when the system is open and decentralised, such as the Web of Linked Data. In this paper, a concise but expressive calculus which models data updates is presented. The calculus is used to provide an operational semantics for a system where data and updates interact concurrently. The operational semantics of the calculus also tracks the provenance of data with respect to updates. This provides a new formal semantics extending provenance diagrams which takes into account the execution of processes in a concurrent setting. Moreover, a sound and complete model for the calculus based on ideals of series-parallel DAGs is provided. The notion of provenance introduced can be used as a subjective indicator of the quality of data in concurrent interacting systems.

1 Introduction

There is a growing trend to publish data openly on the Web. This movement is gaining significant momentum as the governments of several countries and numerous other organisations adopt common principles for publishing data [2]. Data published according to these principles is referred to as Linked Data, due to the use of URIs to establish links between published data. By establishing links between arbitrary data sets, significant problems emerge that are of a different flavour to those associated with traditional closed databases.

Many of the new problems which emerge in this scenario are due to the the decentralised nature of the published data. Some significant challenging problems include: the efficient execution of distributed queries and processes which exploit multiple data sources; the impossibility of enforcing a global schema on data; the lack of boundaries for data ensuring the impossibility of complete results; and establishing global standards for data formats and protocols.

This work considers another essential problem, which reflects the diversity of published data. The challenge considered here is that each piece of data published has a varying degree of trust or relevance. A user may consider data published by the BBC to be more trustworthy than data published on a personal blog. However, if the blog is run by a political activist that the consumer of data approves of, then the blog may be more relevant. Thus data should not be associated with a specific trust measure. Instead, some extra information about the data should be tracked, i.e. the provenance of data. From the extra information provided by the provenance of data, the consumer may judge the quality of the associated data according to their own policy.

Provenance can track several characteristics of the origin of data. Characteristics include "who" has influenced the data, "where" the data has been located, and "how" the data is produced [7]. For Linked Data, a basic notion of "where" provenance called a named graph, which indicates where the data is located now, is the recognised standard [5]. In related work, a model extending named graphs is used to track more comprehensive "where" and "who" provenance [8]. The related work associates trees of identifiers for agents and locations with data. This allows a history of where the data has been published and who published it to be tracked.

Submitted to:
FOCLASA 2012

This work focuses on a form of "how" provenance. This form of "how" provenance tracks causal relationships between stored data and data that was used to produce the data [6]. For instance, due to a change in usage of a building, data about the building may be updated. The updates may replace or extract information from the original data. Thus "how" provenance can be used to determine how old data influenced the new data with respect to an update.

The notion of "how" provenance investigated is strongly related to event based models of causality [3, 16]. This model clarifies, for the first time, the relationship between concurrent process and the provenance diagrams that they produce. An operational semantics formalises the operational behaviour of processes while recording the provenance associated with the resulting data. The model presented provides insight that may be used to refine the definition of provenance diagrams. Provenance diagrams that arise from concurrent updates are guaranteed to be in a particular (series-parallel) form. This insight is a contribution to the effort to establish a common notion of a provenance diagram [15]. Furthermore, the model presented is proven to be sound and complete. The formal model provides a foundation for investigating problems associated with tracking and exploiting the provenance of data, including querying provenance [4, 1], and employing trust metrics [10].

2 Causal Dependencies in Provenance Diagrams

This work focusses on a particular aspect of provenance tracking. The aspect considered is a form of "how" provenance, which indicates how old data contributed to producing new data. The consensus in the provenance community is that provenance diagrams which record this information form a directed acyclic graph (DAG), where the edges are transitively closed. A standard format for representing provenance, called the Open Provenance Model [15], encompasses this notion of provenance. The informal definitions provided by the standard are as follows.

For this work, *artefacts* are data tuples. The *was derived from* relation between artefacts is such that if there is an edge from artefact $\boxed{d_2}$ to artefact $\boxed{d_1}$, then there is a causal relationship that indicates that $\boxed{d_1}$ needs to have been generated to enable $\boxed{d_2}$ to be generated. The standard defines a multi-step was derived from relation. This is simply the transitive closure of the *was derived from* relation, indicating that an artefact had an influence on another artefact.

A provenance diagram that indicates the provenance of two stored pieces of data, where the stored data is indicated by an over line, is presented in Fig. 1. The example is used throughout this work and concerns monuments adjacent to the venue of the workshop.

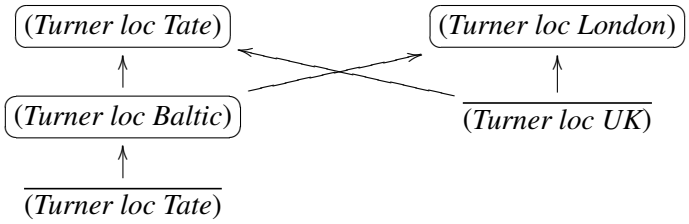


Figure 1: The Turner Prize is held at the Tate Britain in London. However, in 2011 it was held in The Baltic Gallery in Gateshead, but returned to the Tate Britain in 2012. The data in the above diagram is about the location of the Turner Prize. Edges are causal relationships indicating the data consumed to produce new data as the location of the Turner Prize is updated.

3 A Syntax and Semantics for Provenance Tracking Data Updates

This section introduces the syntax and semantics for a concurrent interacting system that tracks provenance. The provenance community have introduced provenance structures based on DAGs; therefore a process model which gives rise to DAGs is considered [15]. Unfortunately, many models of concurrency are based on traces or trees rather than DAGs, such as in the calculus and provenance structures introduced in [18]. This limitation is addressed by providing a non-interleaving semantics, inspired by [9].

3.1 An Abstract Syntax for Processes

The grammar for processes is provided in Fig. 2. The concepts are summarised and made precise by the operational and denotational semantics presented in this work.

a name $\quad x$ variable

$\lambda ::= x \mid a$ variable or name

$d ::= \lambda \mid \lambda\lambda \mid \lambda\lambda\lambda \mid \ldots$ data tuple

$$
\begin{array}{rll}
P ::= & \mathrm{I} & \text{skip} \\
\mid & d & \text{consume data} \\
\mid & \overline{d} & \text{stored data} \\
\mid & \boxed{d} & \text{artefact} \\
\mid & P\,;P & \text{seq} \\
\mid & P \mid P & \text{par} \\
\mid & P \oplus P & \text{choose} \\
\mid & \exists x.P & \text{exists}
\end{array}
$$

Figure 2: The syntax of processes.

The data tuples. The basic unit of information considered in this work is a tuple of names. Tuples are commonly used to convey data. Linked Data is based on RDF which involves triples of names [2, 13]. RDF makes use of URIs for names, since URIs provide a globally recognised naming system. In Linked Data, often RDF triples are extended to quadruples of URIs where the extra URI indicates where the triple is located [5]. This provides a basic notion of "where" provenance. This notion of "where" provenance is extended in [8].

The artefacts. A data tuple can be stored, represented as $\overline{d}$. Stored data can then be consumed in an interaction with the process d. The result is an artefact $\boxed{d}$ used to explicitly track interactions which have occurred. An artefact is used to record a data tuple involved in an interaction. Artefacts are used to capture "how" provenance.

The multiplicatives. There are two multiplicative operators. The "par" multiplicative represents the parallel composition of processes where interactions between processes are permitted. The "seq" multiplicative represents the strict sequential composition of processes, where the first process must terminate before the second process begins, hence the second process is causally dependent on the first process. There is one unit for the multiplicatives, namely skip, which represents a successful action with no side effects.

$$\mathrm{I} \,;\, Q \equiv Q \,;\, \mathrm{I} \equiv Q \mid \mathrm{I} \equiv Q \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P \,;\, (Q \,;\, R) \equiv (P \,;\, Q) \,;\, R \qquad Q \mid R \equiv R \mid Q$$

$$(P \oplus Q) \oplus R \equiv P \oplus (Q \oplus R) \qquad P \oplus Q \equiv Q \oplus P \qquad P \oplus P \equiv P$$

$$(P \oplus Q) \,;\, R \equiv (P \,;\, R) \oplus (Q \,;\, R) \qquad P \,;\, (Q \oplus R) \equiv (P \,;\, Q) \oplus (P \,;\, R) \qquad (P \oplus Q) \mid R \equiv (P \mid R) \oplus (Q \mid R)$$

$$\exists x.(P \oplus Q) \equiv \exists x.P \oplus \exists x.Q \qquad \exists x.\mathrm{I} \equiv \mathrm{I}$$

$$\exists x.(P \mid S) \equiv \exists x.P \mid S \qquad \exists x.(S \,;\, Q) \equiv S \,;\, \exists x.Q \qquad \exists x.(P \,;\, S) \equiv \exists x.P \,;\, S$$

where S is a process where x does not appear free

Figure 3: The structural congruence, which can be applied at any point in a derivation.

The additives. There are two additives: $\oplus$ represents a choice between two branches; $\exists$ represents a choice between all possible name substitutions for the bound variable.

3.2 Operational Semantics of Processes

Deductive systems are typically presented using inference rules applied at the base of a syntax tree, as in the sequent calculus. However, such systems are unsuited to systems which mix commutative and non-commutative operators [11]. For this reason, a deep inference style of presentation is adopted, where inference rules can be applied at an arbitrary depth in a formula.

A structural congruence which extends α-conversion is introduced, in Fig. 3, which is used to rearrange processes. The structural congruence ensures that the order of composition matters for sequential composition, but does not matter for parallel composition. For simplicity, both parallel composition and sequential composition share the same unit. The structural congruence handles contraction for choice, using idempotency. The other rules of the structural congruence determine how operators distribute over each other. Distributivity properties are used in related models of concurrency [9, 12]. Note that this selection of rules is not minimal; however they are used in Sec. 5 to rewrite processes into normal forms, thereby simplifying the completeness proof.

A deductive system is presented in Fig. 4. Deductions may be applied at any depth in a process, as with the structural congruence. Deductions are presented with the premise above the line and the conclusion below the line.

$$\frac{\boxed{d}}{\overline{d} \mid d}\ \textit{interact} \qquad \frac{(P \mid Q) \,;\, (P' \mid Q')}{(P \,;\, P') \mid (Q \,;\, Q')}\ \textit{sequence} \qquad \frac{P}{P \oplus Q}\ \textit{choice} \qquad \frac{P\{^{a}/_{x}\}}{\exists x P}\ \textit{exists}$$

Figure 4: The deductive system for processes. All deductions can be applied in any context.

The interact rule. The interact rule only applies to tuples. The rule indicates that a stored tuple is consumed by the process which deletes that triple. The result of the interaction is an artefact that records

the consumed tuple.

The sequence rule. The sequence rule reorders processes composed in parallel. The premise is more deterministic than the conclusion. The premise decides which part of the process will execute first; whereas the conclusion leave open several other opportunities. This rule allows parts of a process to travel to the intended location where they will interact. This rule appears in related models of concurrency [9, 11, 17].

The additives. The premises of the additives indicate the branch that is chosen. For choice, either the left or the right branch is chosen. For exists, any name may be substituted for the bound variable. This kind of choice is known as external choice in process calculi, where exists is an infinite external choice.

4 A Process Calculus for Provenance Tracking Updates

This section identifies a sub-grammar of processes that model certain systems. The systems modelled are those which involve stored data composed in parallel with updates. The updates involve the removal of some stored data satisfying a query, followed by the insertion of some new stored data.

The operational semantics for processes are provided by the rules of the system in the previous section. A system can evolve to a given state if and only if the new state entails the original state. Notice that implication is in the opposite direction to the evolution of the system. The direction of implication is in line with related approaches to operational semantics [14].

$$
\begin{array}{rcl}
\mathit{Data} & ::= & \mathrm{I} \\
& | & \overline{d} \\
& | & \mathit{Data} \mid \mathit{Data}
\end{array}
\qquad
\begin{array}{rcl}
\mathit{Update} & ::= & \mathit{Query} \,;\, \mathit{Data} \\
& | & \mathit{Update} \oplus \mathit{Update} \\
& | & \exists x.\mathit{Update}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{Query} & ::= & \mathrm{I} \\
& | & d \\
& | & \mathit{Query} \mid \mathit{Query} \\
& | & \mathit{Query} \oplus \mathit{Query} \\
& | & \exists x.\mathit{Query}
\end{array}
\qquad
\begin{array}{rcl}
\mathit{System} & ::= & \mathrm{I} \\
& | & \boxed{d} \\
& | & \mathit{Update} \\
& | & \overline{d} \\
& | & \mathit{System} \,;\, \mathit{System} \\
& | & \mathit{System} \mid \mathit{System}
\end{array}
$$

Figure 5: Sub-algebras of processes for data, queries, updates and systems.

Data. Data simply represents zero or more stored data atoms. The following presents two stored triples in RDF format, which consist of three URIs: the subject, property, and object.

$$\overline{(\mathit{Sage\ rdf{:}type\ Concert\_Hall})} \mid \overline{(\mathit{Baltic\ rdf{:}type\ Art\_Gallery})}$$

Note that all names are active URIs which link to real published Linked Data. The reader is invited to follow the URIs to witness the examples in a real context.

Queries. Parallel composition | and choice $\oplus$ are exploited to model the following queries. As in [13], the existential quantifier is used to select URIs which occur in data. The following pattern uses choice to select between two objects. This example discovers a concert hall located in either Newcastle or Gateshead.

$$\exists x.((x \ \mathit{rdf{:}type}\ \mathit{Concert\_Hall}) \mid ((x \ \mathit{loc}\ \mathit{Newcastle}) \oplus (x \ \mathit{loc}\ \mathit{Gateshead})))$$

Note that a tighter operational semantics could be provided by using a tensor product to join queries [13]. A tensor product ensures both parts of a query are answered atomically. Unfortunately, the calculus for Linked Data in [13] has an interleaving semantics, which would give rise to trees of provenance diagrams as in [18]. Future work would be to combine the strengths of both calculi.

Updates. The following is an example of an update which applies to some stored data. The existential quantification discovers a name which is used in the delete statement and the data stored after the delete. The Baltic Art Gallery is a converted flour mill. The update turns a depiction of the old flour mill into a depiction of the new art gallery.

$$\overline{(\mathit{Mill\ depiction\ photo})} \mid \exists x.\Big((\mathit{Mill\ depiction}\ x)\ ;\ \overline{(\mathit{Baltic\ depiction}\ x)}\Big)$$

The above process is provable from the following process, using the exists, sequence and interact rules. This means that the system above can evolve to the system below.

$$\boxed{(\mathit{Mill\ depiction\ photo})}\ ;\ \overline{(\mathit{Baltic\ depiction\ photo})}$$

Notice that the original stored triple appears as an artefact, which the new triple is dependent on. This provides "how" provenance that indicates the old triple used to create the new triple.

Distinctions between execution paths. There are multiple ways of evaluating processes. Different methods of evaluation can give rise a different provenance. Here three distinct executions of the same process are presented to demonstrate the complexity of provenance tracking in a concurrent setting.

An example which involves two updates executed in parallel is presented below. It is a common misconception that The Sage and Baltic Art Gallery are prominent monuments in Newcastle. In reality they are located in Gateshead on the opposite bank of the river Tyne[1]. The updates transform the location of these monuments from Newcastle to Gateshead.

$$\begin{array}{l}\overline{(\mathit{Sage\ loc\ Gateshead})} \mid \Big((\mathit{Sage\ loc\ Gateshead})\ ;\ \overline{(\mathit{Sage\ loc\ Newcastle})}\Big) \mid \\ \overline{(\mathit{Baltic\ loc\ Gateshead})} \mid \Big((\mathit{Baltic\ loc\ Gateshead})\ ;\ \overline{(\mathit{Baltic\ loc\ Newcastle})}\Big)\end{array}$$

The process below yields the process above, using the sequence rule. The two updates occur independently, hence each provenance is independent.

$$\begin{array}{l}\Big(\boxed{(\mathit{Sage\ loc\ Gateshead})}\ ;\ \overline{(\mathit{Sage\ loc\ Newcastle})}\Big) \mid \\ \Big(\boxed{(\mathit{Baltic\ loc\ Gateshead})}\ ;\ \overline{(\mathit{Baltic\ loc\ Newcastle})}\Big)\end{array}$$

The process below yields both process above. This suggest that the two updates were combined before they were applied, hence data produced by each update is dependent on the artefact of the other update. Therefore the process below has stronger dependencies than the process above.

$$\Big(\boxed{(\mathit{Sage\ loc\ Gateshead})} \mid \boxed{(\mathit{Baltic\ loc\ Gateshead})}\Big)\ ;\ \Big(\overline{(\mathit{Sage\ loc\ Newcastle})} \mid \overline{(\mathit{Baltic\ loc\ Newcastle})}\Big)$$

Indeed the above process can be refined further to impose a sequential dependency on the artefacts. Thus the execution of the concurrent processes greatly affects the form of "how" provenance.

[1] Indeed the venue of FOCLASA 2012 is also in Gateshead, rather than in Newcastle.

The Turner Prize revisited. The operational behaviour which gives rise to the provenance diagram in the introduction can now be expressed. The initial configuration is expressed below. It shows two stored triples, an update that moves the exhibition from the Tate Britain to the Baltic and broadens London to the UK, and an update which moves the exhibition back from the Baltic to the Tate Britain.

$$\begin{array}{l}\overline{(\mathit{Turner\ loc\ London})} \mid \overline{(\mathit{Turner\ loc\ Tate})} \mid \\ ((\mathit{Turner\ loc\ Tate}) \mid (\mathit{Turner\ loc\ London})) \,;\, \Big(\overline{(\mathit{Turner\ loc\ Baltic})} \mid \overline{(\mathit{Turner\ loc\ UK})}\Big) \mid \\ \Big((\mathit{Turner\ loc\ Baltic}) \,;\, \overline{(\mathit{Turner\ loc\ Tate})}\Big)\end{array}$$

By applying the sequence rule several times the processes can be rearranged as follows.

$$\begin{array}{l}\Big((\mathit{Turner\ loc\ Tate}) \mid \overline{(\mathit{Turner\ loc\ Tate})} \mid \overline{(\mathit{Turner\ loc\ London})} \mid (\mathit{Turner\ loc\ London})\Big) \,; \\ \Big(\Big(\overline{(\mathit{Turner\ loc\ Baltic})} \mid (\mathit{Turner\ loc\ Baltic})\Big) \,;\, \overline{(\mathit{Turner\ loc\ Tate})}\Big) \mid \overline{(\mathit{Turner\ loc\ UK})}\Big)\end{array}$$

Finally, by applying the interact rule the delete operations and stored data cancel each other out. The interaction produce the artefacts that record the provenance of the data.

$$\begin{array}{l}\Big(\boxed{(\mathit{Turner\ loc\ Tate})} \mid \boxed{(\mathit{Turner\ loc\ London})}\Big) \,; \\ \Big(\Big(\boxed{(\mathit{Turner\ loc\ Baltic})} \,;\, \overline{(\mathit{Turner\ loc\ Tate})}\Big) \mid \overline{(\mathit{Turner\ loc\ UK})}\Big)\end{array}$$

The next section provides a denotational semantics where the denotation of above process is exactly the provenance diagram in the introduction.

5 A Denotational Semantics for the Provenance Tracking Calculus

This section provides a denotational semantics for the calculus. A denotational semantics provides a sound and complete model which increases confidence in the definition of the calculus. In this case, the semantics of the calculus fulfils an additional purpose. It also makes explicit the connection between certain terms of the calculus and provenance diagrams. Furthermore, a restriction on provenance diagrams that track series-parallel computations is highlighted.

The denotational semantics, similarly to provenance diagrams, is based on directed acyclic graphs (DAGs). The denotation relies on some technical apparatus. Firstly, DAGs are restricted by a forbidden minor property, which guarantees that each DAG arises from applying series and parallel composition to smaller DAGs. Secondly, homomorphisms between DAGs are defined such that the inference rules of the calculus hold. By taking ideals of series-parallel DAGs with respect to these homomorphism, a sound and complete model is obtained.

5.1 Series-Parallel DAGs and the N-free Condition

This section recalls some standard definitions which are used to build a denotational semantics. The definition of a DAG is standard, as are the definitions of the transitive closure of a graph and the notion of a graph homomorphism. Transitive DAGs are used because provenance diagrams are transitive, and graph homomorphism are used to compare the structure of such diagrams.

Definition 5.1. *A DAG $D = (V, E)$ is a digraph with no directed cycles. Let $A = (V, E)$ and $B = (V', E')$ be graphs. A graph homomorphism is given by a function on vertices $f : V \to V'$ such that if $(u, v) \in E$ then $(f(u), f(v)) \in E'$. Two graphs are isomorphic iff there exists a bijective homomorphism whose inverse*

function is also a homomorphism. A transitive digraph is such that if there exists a path from u to v, then there exists an edge from u to v. A transitive closure of a digraph (V,E) *is a minimal transitive digraph* (V,E') *such that there exists an injective graph homomorphism from* (V,E) *to* (V,E')*. A graph* (V,E) *is a sub-graph* (V',E') *if and only if* $V \subseteq V'$ *and* $E = E' \cap V \times V$*.*

Several series-parallel digraphs are studied in [19]. Here transitive series-parallel DAGs are defined. The series-parallel restriction on transitive DAGs is required because this work considers provenance diagrams which arise from the execution of series-parallel processes.

Definition 5.2. *The trivial DAG with no vertices is a series-parallel DAG, and the DAG with a single vertex and no edges is a series-parallel DAG. If* $G_0 = (V_0, E_0)$ *and* $G_1 = (V_1, E_1)$ *are series-parallel graphs with disjoint vertices, then the following are series-parallel DAGs.*

- $G_0 \| G_1$ *defined by* $(V_0 \cup V_1, E_0 \cup E_1)$*.*
- $G_0 \,;\, G_1$ *defined as the transitive closure of* $(V_0 \cup V_1, E_0 \cup E_1 \cup (L \times R))$*, where* L *is the source nodes of* G_0 *and* R *is the sink nodes of* G_1*.*

In structural graph theory it is studied how graph classes either can be defined by forbidden minors, or by being glued together from simple starting graphs (as in the definition above). A forbidden minor is a sub-graph with a particular form; the forbidden minor for series-parallel DAGs has an N-shape, as proven in [19].

Theorem 5.3 (Forbidden minor)**.** *A transitive DAG is series-parallel if and only if it does not have a sub-graph isomorphic to* $N = (\{v_0, v_1, v_2, v_3\}, \{(v_2, v_0), (v_3, v_0), (v_3, v_1)\})$*.*

Notice that use of transitive DAGs is motivated, by provenance diagrams; while the series-parallel restriction is motivated by concurrent processes. Thus the model studies structures which respect both provenance and the processes which track the provenance.

5.2 Interacting Series-Parallel DAGs Labelled with Data

The notion of a series-parallel DAG is extended with labels. The labels allow data to be accommodated in the model. Also the notion of a homomorphism is extended to allow interactions between data and operations on data which give rise to artefacts.

The definition of a labelled graph is standard. A special kind of homomorphism is defined on labelled DAGs. This smoothing homomorphism is bijective, but does not define an isomorphism. Thus vertices are preserved, but extra edges may appear.

Definition 5.4. *Fix* Σ *as the set of labels which are either tuples* d*, stored tuples* $\overline{d}$ *or artefacts* $\boxed{d}$*. A labelled graph* (V,E,μ) *is such that* (V,E) *is a graph and* $\mu : V \to \Sigma$ *is a labelling function from vertices to labels. Let* $A = (V,E,\mu)$ *and* $B = (V',E',\mu')$ *be labelled DAGs. A labelled homomorphism* f *from* A *to* B *is such that* f *is a graph homomorphism from* (V,E) *to* (V',E') *and for all vertices* u*,* $\mu(u) = \mu'(f(u))$*. A smoothing homomorphism is a bijective labelled homomorphism.*

The notation of a smoothing homomorphism defined above is used to characterise the sequence rule. To capture both the sequence rule and the interact rule, interaction homomorphisms are introduced. The definition involves a coherence condition which captures the conditions under which an interaction may occur. Two vertices can interact if they have complementary labels and they are in parallel with each other. This leads to the following definition.

Definition 5.5. *For a labelled graph $A = (V, E, \mu)$, define $u \frown_d v$ in A such that there is no directed path between u and v, and either $\overline{d} = \mu(u) = \overline{\mu(v)}$ or $\overline{\mu(u)} = \mu(v) = \overline{d}$. Let $A = (V, E, \mu)$ and $B = (V', E', \mu')$ be labelled DAGs. An interaction homomorphism f from A to B is a labelled graph homomorphism such that f is onto and, if $f(u) = f(v)$, one of the following hold: either $u \frown_d v$ in A and $\mu'(f(u)) = \boxed{d}$; or $u = v$ and $\mu(u) = \mu'(f(u))$.*

The following example demonstrates two compatible vertices mapped to the same vertex by an interaction homomorphism.

Note that the diagrams in examples represent equivalence classes of labelled graphs up to labelled graph isomorphism. Thus only the labels and not the underlying vertices are indicated. The same practice is followed when presenting provenance diagrams.

The homomorphisms defined over labelled DAGs are used to generate ideals. Ideals are sets of labelled series-parallel DAGs closed with respect to either smoothing or interacting homomorphisms.

Definition 5.6. *A smoothing/interacting ideal I is a set of labelled series-parallel DAGs such that if $A \in I$ and there exists a smoothing/interacting homomorphism $f : A \to B$, then $B \in I$. For any set of labelled series-parallel DAGs P the smoothing/interacting ideal closure of P, denoted $\iota_s P / \iota_i P$, is the least ideal containing P, defined as the intersection of all smoothing/interacting ideals I such that $P \subseteq I$.*

These ideals are employed to denote processes in the next section. Ideal closure is essential for the denotation of parallel composition.

5.3 Correctness of the Denotational Semantics

The denotational semantics for processes is defined using the ideals introduced in the previous section. Most operations on ideals are the obvious point-wise extension of the corresponding operator. The main subtlety is that parallel composition introduces new possibilities for both smoothing and interaction, which are not represented by the point-wise parallel composition of ideals. Thus the ideal closure is employed to denote parallel composition. Valuations are used to represent substitutions, which are required to denote existential quantification.

Definition 5.7 (denotation)**.** *A valuation v is a mapping from variables to names. Let $v[x \mapsto a]$ be the valuation which is the same as v except at x where it maps to a. The effect of a valuation on a label is defined as follows.*

$$\boxed{d}^v = \boxed{d^v} \qquad \left(\overline{d}\right)^v = \overline{d^v} \qquad (\lambda_0..\lambda_n)^v = \lambda_0^v..\lambda_n^v \qquad a^v = a \qquad x^v = v(x)$$

The denotation of a process with respect to a valuation v satisfies the following, where $h \in \{s, i\}$, ϵ is the set containing the empty labelled graph, and $e(l, v)$ is the equivalence class of labelled graph with one

vertex labelled with l^v with respect to labelling isomorphism.

$$[\![v, \mathrm{I}]\!]_h = \epsilon \qquad [\![v, l]\!]_h = e(l, v) \qquad [\![v, \exists x P]\!]_h = \bigcup_{a \in \mathtt{Names}} [\![v[x \mapsto a], P]\!]_h$$

$$[\![v, P \oplus Q]\!]_h = [\![v, P]\!]_h \cup [\![v, Q]\!]_h \qquad [\![v, P \,;\, Q]\!]_h = \{A \,;\, B \mid (A, B) \in [\![v, P]\!]_h \times [\![v, Q]\!]_h\}$$

$$[\![v, P \,||\, Q]\!]_h = \iota_h \{A \,||\, B \mid (A, B) \in [\![v, P]\!]_h \times [\![v, Q]\!]_h\}$$

All the operations used in the denotational semantics preserve ideals, as verified by the following proposition. Therefore the denotational semantics is a well defined mapping from processes to ideals.

Proposition 5.8. *The following are ideals: ϵ, $e(l, v)$, the union and intersection of sets of ideals, and the point-wise sequential composition of ideals.*

Soundness of the calculus defined in Sec. 3 with respect to the denotation is straight forward. The proof follows from checking that all equations of the structural congruence hold as set equality of ideals, and that all deductive rules hold as set inclusions of ideals.

Theorem 5.9 (soundness)**.** *If P yields Q, then, $[\![v, P]\!]_i \subseteq [\![v, Q]\!]_i$ for all valuations v.*

Completeness of the calculus with respect to the denotation is more challenging. The proof follows from interpolation lemmas. An interpolation lemma establishes that if there is a strict inclusion between the denotation of processes then there must be a finite sequence of deductions that can be applied to transform one process into the other process. The trick is to rewrite processes into a normal form and deal with each deductive rule one by one.

Firstly consider series-parallel terms, which are processes which does not feature any choice or exists. Two interpolation lemmas apply to series-parallel terms. The first interpolation lemma, stated below, deals only with the sequence rule. This lemma is closely related to the interpolation lemma established in [9], where a similar calculus without interactions is considered. Thus only smoothing ideals are treated.

Lemma 5.10 (sequence interpolation)**.** *Given two series-parallel terms P and Q, if $[\![v, P]\!]_s \subseteq [\![v, Q]\!]_s$ for all valuations v, then either: $[\![v, P]\!]_s = [\![v, Q]\!]_s$ for all valuations v; or there exists R such that $[\![v, P]\!]_s \subset [\![v, R]\!]_s \subseteq [\![v, Q]\!]_s$ for all valuations v, and P yields R is provable using only the sequence rule.*

The above result is extended to interacting homomorphism in the following interpolation lemma. The proof of this lemma is an important technical contribution of this work. It shows that, for any strict inclusion between the denotation of series-parallel process, either the sequence rule or the interact rule can be applied.

Lemma 5.11 (interaction interpolation)**.** *Given two series-parallel terms P and Q, if $[\![v, P]\!]_i \subseteq [\![v, Q]\!]_i$ for all valuations v, then: either $[\![v, P]\!]_s \subseteq [\![v, Q]\!]_s$ for all valuations v; or there exists R such that $[\![v, P]\!]_i \subset [\![v, R]\!]_i \subseteq [\![v, Q]\!]_i$ for all valuations v and P yields R is provable using only the interact rule.*

Proof. Assume that P and Q are series-parallel terms such that $[\![v, P]\!]_i \subset [\![v, Q]\!]_i$ for all valuations v. Also assume that $[\![v, P]\!]_s \not\subset [\![v, Q]\!]_s$ for some valuation v. Since P, Q are series-parallel terms, there exist series-parallel DAGs $D_0 = (V_0, E_0, \mu_0)$, $D_1 = (V_1, E_1, \mu_1)$ such that $\iota_i D_0 = [\![v, P]\!]_i$ and $\iota_i D_1 = [\![v, Q]\!]_i$. Also, since $[\![v, P]\!]_i \subset [\![v, Q]\!]_i$, there exists an interacting homomorphism $f \colon D_1 \to D_0$.

There must be at least one interaction in the homomorphism f exhibited above, i.e. there exists $m, n \in V_1$ such that $f(m) = f(n)$, $m \frown_d n$ and $f(m) = w \in V_0$ such that $\mu_0(w) = \boxed{d}$. Suppose otherwise,

then for all $m,n \in V_1$ if $f(m) = f(n)$ then $m = n$, and so f is bijective, since interacting homomorphisms are surjective. Hence f is a smoothing homomorphism from D_1 to D_0, so $[\![v, P]\!]_s \subset [\![v, Q]\!]_s$ contradicting the above assumption.

A DAG $D_2 = (V_2, E_2, \mu_2)$ is constructed to differ from D_0 only by the interaction exhibited by f. Firstly, take V_0, remove vertex w and include vertices m and n, so $V_2 = V_0 \setminus \{w\} \cup \{m,n\}$. Let $E_0 \setminus w$ be the set of edges in E_0 without the vertex w and define $E_2 = (E_0 \setminus w) \cup \{(x,m) \mid (x,w) \in E_0\} \cup \{(m,x) \mid (w,x) \in E_0\} \cup \{(x,n) \mid (x,w) \in E_0\} \cup \{(n,x) \mid (w,x) \in E_0\}$. Retain all the labels of μ_0 except at m and n, so if $x = m$ or $x = n$ then $\mu_2(x) = \mu_1(x)$ and otherwise $\mu_2(x) = \mu_0(x)$.

Construct two homomorphisms from $g\colon D_2 \to D_0$ and $h\colon D_1 \to D_2$ as follows.

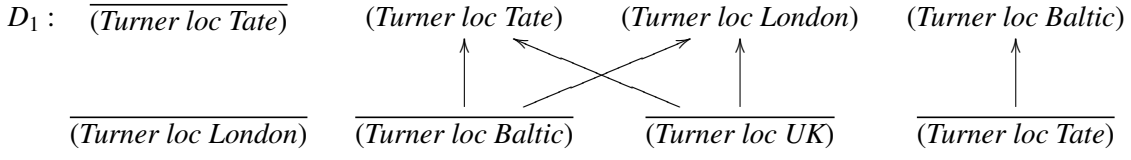
$$g(x) = \begin{cases} f(x) & \text{if } x = m \text{ or } x = n \\ x & \text{otherwise} \end{cases} \qquad h(x) = \begin{cases} x & \text{if } x = m \text{ or } x = n \\ f(x) & \text{otherwise} \end{cases}$$

Clearly $f = g \circ h$. Furthermore, both g and h are interacting homomorphisms by the following arguments. Check that g is a graph homomorphism, by case analysis. Only one case is presented. By definition of E_2, if $(m,x) \in E_2$ then $(m,x) \in \{(m,x) \mid (w,x) \in E_0\}$, thus $(g(m), g(x)) = (w,x) \in E_0$. Also check that g is an interaction homomorphism, as follows: If $g(x) = g(y)$ then either $x = y$, or $x = m$ and $y = n$. Clearly, m and n are not connected in E_2 and both $\mu_2(m) = \mu_1(m)$ and $\mu_2(n) = \mu_1(n)$ hold, so $m \frown_d n$ in D_2 and $\mu_0(f(m)) = \mu_0(w) = \boxed{d}$. Check that h is a graph homomorphism. Only one case is presented. If $(m,x) \in E_1$ then $(w, f(x)) \in E_0$ since f is a graph homomorphism, thus $(m, f(x)) \in \{(m,x) \mid (w,x) \in E_0\}$ so $(m, f(x)) \in E_2$, by definition. Now consider when $h(x) = h(y)$ either $x = y$ or $x, y \notin \{m,n\}$, hence $f(x) = f(y)$, thus $x \frown_d y$ since f is an interacting homomorphism. Suppose, without loss of generality, that $x = m$ and $y \notin \{m,n\}$, so $m = f(x)$, but $m \notin V_0$ contradicting the definition of f. Thus h is an interaction homomorphism.

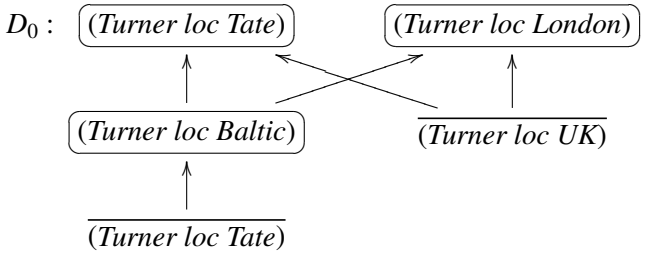
Furthermore, the constructed DAG, D_2, is series-parallel. Suppose otherwise, then there exists an N-shape isomorphic to a sub graph of D_2. Now consider the image of the N-shape under g. Either zero or one nodes in the N-shape are m or n so the image of the N shape is an N-shape in D_0. By Theorem 5.3, this contradicts the fact that D_0 is series-parallel. Now, suppose that both m and n are in the N-shape. Since $m \frown_d n$ in D_2, m and n are not connected, so an N-shape must be of the form $\{(m,x),(n,x),(n,y)\}$ or $\{(x,m),(x,n),(y,n)\}$. However $(m,x) \in D_2$ iff $(w,x) \in D_0$ iff $(n,x) \in D_2$ and $(x,m) \in D_2$ iff $(x,w) \in D_0$ iff $(x,n) \in D_2$, so neither shapes are sub-graphs of D_2. Thus D_2 is N-free, hence by Theorem 5.3, D_2 is a series-parallel DAG.

Since, D_2 is a series-parallel DAG, there exists a series-parallel term R such that $[\![v,R]\!]_i = \iota_i D_2$. Since $g : D_2 \to D_0$ exhibiting an interaction and $h\colon D_1 \to D_2$, the following inequalities hold $[\![v,P]\!]_i \subset [\![v,R]\!]_i$ and $[\![v,R]\!]_i \subseteq [\![v,Q]\!]_i$. Since $\mu_2(w) = \boxed{d}$, the sub-term $\boxed{d}$ must appear in the process $P = \mathcal{S}\{\ \boxed{d}\ \}$, for some context $\mathcal{S}\{\ \}$. Also, since $m \frown_d n$ and, the edges of D_0 differs from those of D_2 only in that the edges connected to w in D_0 are instead connect to both m and n in D_2, the following holds $R \equiv \mathcal{S}\{\ d \,\|\, \overline{d}\ \}$. Thus the interact rule proves that P yields R, as required. □

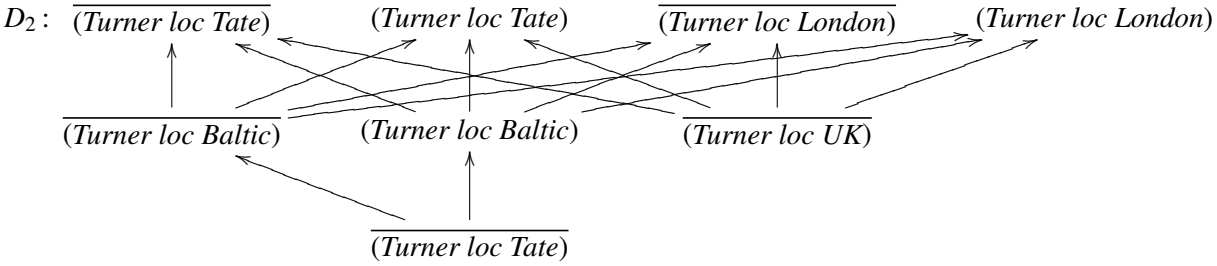
To clarify the significance of the interpolation lemmas consider the running example. The initial configuration of processes is denoted by the following DAG (D_1).



There exists an interaction homomorphism from D_1 to the DAG D_0 below, which appears also in Sec. 2.



Now, by applying Lemma 5.11 three times, we can construct a series of DAGs cumulating in D_2 presented below, such that the following properties hold. There exist interaction homomorphisms from D_1 to D_2 and from D_2 to D_0, and the process denoted by D_0 yields the process denoted by D_2 using the interact rule three times. Furthermore, the homomorphism from D_1 to D_2 is a smoothing homomorphism. Hence, by Lemma 5.10, the process denoted by D_2 can be transformed using the sequence rule applied a finite number of times into the process denoted by D_1.



Thereby the existence of the interaction homomorphism between D_1 and D_0 guarantees the existence of a deduction from the process denoted by D_0 to the process denoted by D_1 using the interact and sequence rules. Indeed the processes and deductions are presented in the example at the end of Sec. 4, where the first process in the example is denoted by D_1, the second by D_2 and the third by D_0.

Every process can be written in a normal form, using the structural congruence, as a sum of series-parallel process with all the existential quantification moved to the front of the process, i.e. for all P there exist series-parallel processes A_i such that $P \equiv \exists \vec{x}.\Sigma_{i\in I} A_i$. It is then easy to show that a finite number of choice and exists rules can be applied to prove any inequality between ideals. This establishes the completeness of the calculus with respect to the denotation, stated as follows.

Theorem 5.12 (completeness)**.** *If* $[\![v, P]\!]_i \subseteq [\![v, Q]\!]_i$ *for all valuations v, then P yields Q.*

Thus the model based on ideals of labelled series-parallel DAGs is a sound and complete model of processes. The labelled DAGs are inspired by the guidelines provided for provenance diagrams [15]; while, the series-parallel processes are motivated by calculi which model systems which produce provenance diagrams. Hence a formal connection between series-parallel DAGs and processes is established. Specifically, provenance diagrams are the denotation of series-parallel processes consisting of only artefacts and stored data. Hence provenance diagrams are contained within a denotation for a provenance tracking calculus. Due to soundness and completeness of the calculus with respect to the denotation, provenance diagrams can be considered in a new operational language based setting.

6 Conclusion

Provenance is a key problem in processing data which is particularly important in systems that publish data on the Web, such as the Web of Linked Data [8, 13]. Already certain aspects of provenance are gifted with deep theoretical results [10]. However, there is no sound and complete model for the aspects of provenance tracking considered in this work: specifically "how" provenance which indicates causal relationships; and a provenance tracking calculus which produces such diagrams by recording interactions between processes and stored data. The relationship between the diagrams and the calculus is exhibited by providing a sound and complete denotational semantics which contains such provenance diagrams.

The examples presented in this paper illustrate that tracking provenance is particularly challenging in a concurrent setting. The causal aspects of data provenance are closely related to the operational semantics of the systems involved. Hence when considering concurrent systems, models of concurrency provide insight into problems associated with provenance in a concurrent setting. For instance, this work demonstrates that provenance diagrams that arise from concurrent interactions form series-parallel DAGs. Consequently, certain graph homomorphism problems, which can be employed to query provenance diagrams, can be solved more efficiently for series-parallel digraphs [19]. This model is proposed as a foundation for "how" provenance, which can be applied as a subjective measure of the quality of data.

Acknowledgements. We thank the reviewers for feedback that improved the exposition. The work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0919.

References

[1] Manish Kumar Anand, Shawn Bowers & Bertram Ludäscher (2010): *Techniques for efficiently querying scientific workflow provenance graphs*. In: *EDBT '10*, ACM, pp. 287–298, doi:10.1145/1739041.1739078.

[2] Christian Bizer, Tom Heath & Tim Berners-Lee (2009): *Linked Data — The Story So Far*. *International Journal on Semantic Web and Information Systems* 5(3), pp. 1–22, doi:10.4018/jswis.2009081901.

[3] Grard Boudol & Ilaria Castellani (1989): *Permutation of transitions: An event structure semantics for CCS and SCCS*. In: *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, *Lecture Notes in Computer Science* 354, Springer, pp. 411–427, doi:10.1007/BFb0013028.

[4] Peter Buneman, Adriane Chapman & James Cheney (2006): *Provenance management in curated databases.* In: *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 539–550, doi:10.1145/1142473.1142534.

[5] Jeremy J. Carroll, Christian Bizer, Pat Hayes & Patrick Stickler (2005): *Named graphs*. *Web Semantics: Science, Services and Agents on the World Wide Web* 3(4), pp. 247–267, doi:10.1016/j.websem.2005.09.001.

[6] James Cheney (2010): *Causality and the Semantics of Provenance*. In: *Developments in Computational Models*, pp. 63–74, doi:10.4204/EPTCS.26.6.

[7] James Cheney, Laura Chiticariu & Wang-Chiew Tan (2009): *Provenance in Databases: Why, How, and Where*. *Found. Trends databases* 1(4), pp. 379–474, doi:10.1561/1900000006.

[8] Mariangiola Dezani, Ross Horne & Vladimiro Sassone (2012): *Tracing where and who provenance in Linked Data: a calculus*. *Theoretical Computer Science*, doi:10.1016/j.tcs.2012.06.020.

[9] Jay L. Gischer (1988): *The equational theory of pomsets*. *Theoretical Computer Science* 61(2-3), pp. 199–224, doi:10.1016/0304-3975(88)90124-7.

[10] Todd J. Green, Grigoris Karvounarakis & Val Tannen (2007): *Provenance semirings*. In: *PODS '07*, ACM, pp. 31–40, doi:10.1145/1265530.1265535.

[11] Alessio Guglielmi (2007): *A system of interaction and structure*. *ACM Transactions on Compututational Logic* 8, doi:10.1145/1182613.1182614.

[12] Tony Hoare, Bernhard Möller, Georg Struth & Ian Wehrman (2011): *Concurrent Kleene Algebra and its Foundations*. *Journal of Logic and Algebraic Programming* 80(6), pp. 266–296, doi:10.1016/j.jlap.2011.04.005.

[13] Ross Horne & Vladimiro Sassone (2011): *A Verified Algebra for Linked Data*. In: *FOCLASA*, pp. 20–33, doi:10.4204/EPTCS.58.2.

[14] Naoki Kobayashi & Akinori Yonezawa (1993): *ACL – A Concurrent Linear Logic Programming Paradigm*. In: *Proceedings of the 1993 International Logic Programming Symposium*, MIT Press, pp. 279–294. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.2776&rep=rep1&type=pdf`.

[15] 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 & Jan Van den Bussche (2011): *The Open Provenance Model core specification (v1.1)*. *Future Gener. Comput. Syst.* 27(6), pp. 743–756, doi:10.1016/j.future.2010.07.005.

[16] Vaughan Pratt (1986): *Modeling concurrency with partial orders*. *International Journal of Parallel Programming* 15(1), pp. 33–71, doi:10.1007/BF01379149.

[17] Cristian Prisacariu (2010): *Synchronous Kleene algebra*. *Journal of Logic and Algebraic Programming* 79(7), pp. 608–635, doi:10.1016/j.jlap.2010.07.009.

[18] Issam Souilah, Adrian Francalanza & Vladimiro Sassone (2009): *A Formal Model of Provenance in Distributed Systems*. In: *Workshop on the Theory and Practice of Provenance*, pp. 1–11. Available at `http://static.usenix.org/events/tapp09/tech/full_papers/souilah/souilah.pdf`.

[19] Jacobo Valdes, Robert E. Tarjan & Eugene L. Lawler (1979): *The recognition of Series Parallel digraphs*. In: *STOC '79*, ACM, pp. 1–12, doi:10.1145/800135.804393.