A Formal Model of Provenance in Distributed Systems

Issam Souilah
University of Southampton, UK

Adrian Francalanza
University of Malta, Malta

Vladimiro Sassone
University of Southampton, UK

Abstract

We present a formalism for provenance in distributed systems based on the π-calculus. Its main feature is that all data products are annotated with metadata representing their provenance. The calculus is given a provenance tracking semantics, which ensures that data provenance is updated as the computation proceeds. The calculus also enjoys a pattern-restricted input primitive which allows processes to decide what data to receive and what branch of computation to proceed with based on the provenance information of data. We give examples to illustrate the use of the calculus and discuss some of the semantic properties of our provenance notion. We conclude by reviewing related work and discussing directions for future research.

1 Introduction

As a concept, provenance indicates the source and derivation of an object, and is also known as origin, lineage and pedigree. More concretely, provenance refers to a *record* of such source and derivation [19]. This record may include information about the ownership, influences, contributions and any other *historical* or *contextual* information which may be deemed relevant and useful. Provenance has many applications such as auditing, detecting errors and ensuring reproducibility of results. It is also central to the *trust* one places in data, since it can be used as an indicator of quality, especially in a setting where independent verification of other attributes may not be possible.

Provenance tracking is the problem of recording provenance information, and has been studied in a wide range of settings, including databases [4; 5; 7; 8; 11], scientific computing [12; 13; 21; 22] and file systems [20]. Initially, most of the work on provenance relied on intuitive and informal concepts such as *influences*, *contributes to*, and *depends on* in defining what provenance means. These notions were then used to provide mostly ad hoc implementations, with no formal guarantees as to their correctness or adequacy. However, lately there have been a surge of interest in underpinning the more theoretical principles of provenance [4; 7; 11]. Such body of work aims to establish a mathematical and semantic basis for provenance, which is important if we are to compare different notions of provenance and assess the correctness of their implementations. The present work falls in this latter line of research and aims to provide a formal study of provenance-based trust in concurrent and distributed systems.

We choose the asynchronous π-calculus [3; 15], a version of the π-calculus [18] where message output is non-blocking, as the formalism for modelling distributed systems. The choice of asynchrony is motivated mainly by the envisaged applications of our calculus and its impact on possible implementations, however, it has minimal effect on our results. In order to be able to refer to individual agents, we extend this basic formalism with explicit identities. These identities can be thought of as representing *units of trust*, and are merely labels for identifying processes with no effect on communication (cf. localities in the Dpi calculus [14]). With this extension we can express the essence of computation in a distributed setting, whereby multiple agents interact by exchanging messages. For instance the code:

$$a[n\langle v_1\rangle] \quad | \quad b[n\langle v_2\rangle] \, | \quad c[n(x).P]$$

denotes a system composed of three principals: a and b that are sending values v_1 and v_2 on the same channel n and c that is attempting to read a value from the channel n and use it in the continuation code P. Interestingly, from the point of view of c, there exists a level of *non-determinism*, that is a market of values on channel n from which c is free to choose. However, assuming c has no way of assessing the *quality* of the different values available on channel n, it cannot decide which of the two values v_1 or v_2 to consume. In such situations, *prove-*

nance may be used as a measure of the quality of data. For example, the three principals may agree on a convention whereby the senders (producers of data) attach a *provenance tag*. Thus the code may be modified to:

$$a[n\langle a, v_1\rangle] \mid b[n\langle b, v_2\rangle] \mid c[n(x, y).\mathsf{if}\ x = a\ \mathsf{then}\ P\ \mathsf{else}\ Q\ ]$$

Now, the consumer of the data, c, can determine where the data originated from and branch accordingly. Even though this encoding works, it carries two major disadvantages: (1) it is cumbersome and muddles the code carrying out the actual computation, and (2) principals have to adhere to these provenance conventions, which cannot be enforced. Crucially, the second point leads to circular reasoning with respect to trust. For instance, in the above code nothing stops principal b from forging a's identity using $b[n\langle a, v_2\rangle]$. This simple problem may be solved using a digital signature scheme for example, however, in general the picture can be far more complicated as messages may carry, not only data authored by the agent itself, but also data obtained from other agents. Moreover, agents may consult other agents about where to get data from and what to do with it. These may also be of interest to consumers of data in order to be able to judge the trustworthiness of data.

A distinguishing feature of our distributed π-calculus extension is the use of *provenance annotated data*. Every value, v, is annotated with its provenance, κ, and denoted as $v : \kappa$. The provenance κ is a sequence of ';'-delimited *events*. The events in a provenance sequence $e_1; \ldots; e_n$ are chronologically ordered with e_1 being the most recent event. An event e_i may be an output event $a!\kappa$, which says that the value has been sent by an agent a on a channel whose provenance is κ, or an input event $a?\kappa$, which says that the value has been received by an agent a on a channel whose provenance is κ; the rationale here is that in the π-calculus channels are data too.

We propose a two-tiered framework which automates the process of provenance tracking and separates it from the actual computation.[1] This two-tiered framework is manifested by the provenance tracking reduction semantics, which, in addition to describing the interaction between agents, keeps the provenance of values up-to-date. For instance, the rule for sending data on a particular channel has two facets:

$$a[m{:}\kappa_m\langle v : \kappa_v\rangle] \to m\langle\!\langle v : a!\kappa_m; \kappa_v\rangle\!\rangle$$

From the computational perspective, the rule represents the first of a two-step communication process: a principal a wanting to output a value $v : \kappa_v$ on a channel $m : \kappa_m$ generates a *packaged* message $m\langle\!\langle v : a!\kappa_m; \kappa_v\rangle\!\rangle$, where m represents the address and $v : a!\kappa_m; \kappa_v$ the annotated data content. From the provenance tracking perspective, the rule also describes how the provenance of the value v is updated to $a!\kappa_m; \kappa_v$ after the output operation. This tells us that the value has been most recently sent by agent a on a channel whose provenance is κ_m.

Dually, the rule for receiving values, the second step in communication, has two facets as well:

$$\frac{\kappa_v \models \pi}{b[m{:}\kappa_m(\pi\ \mathsf{as}\ x).P] \parallel m\langle\!\langle v : \kappa_v\rangle\!\rangle \to b[P\{^{v\,:\,b?\kappa_m;\kappa_v}/_x\}]}$$

On the one hand, the rule states that principal b can input a value from channel m and proceed as P if there exists a packaged value with destination m in the system; otherwise it blocks until such a packaged value becomes available. More importantly though, this rule also describes how the provenance information attached to the packaged data, κ_v, is first used for vetting purposes before consumption, and afterwards updated. More specifically, the data input on channel m only progresses if the provenance of the packaged data, κ_v, passes the input test π; this is denoted as $\kappa_v \models \pi$. Should this test succeed, the input is allowed to occur and, in addition, the provenance attached to value v is updated to $v : b?\kappa_m; \kappa_v$ in the continuation P, thereby recording the fact that the value was received by principal b on some channel with provenance κ_m.

In addition to proposing a provenance-based calculus, we also formalise a notion of *correctness* for our provenance annotations. We interpret the provenance κ of an annotated value $v : \kappa$ as a *partial order* of *assertions* about events that occurred in the system, relating *specifically* to value v. For example, the provenance of annotated value $v : a!\kappa_m; \kappa_v$ tells us that v has been sent by principal a on some channel with provenance κ_m, and that before this, the events described by κ_m and κ_v took place, without restricting the relative ordering between the events in κ_m and κ_v; the information in the recorded event $a!\kappa_m$ is also partial because it does not specify the channel on which v was sent. To prove correctness, we devise a technique whereby a decorated version of our reduction semantics constructs a global log which records a *total ordering* of past events relating to *all values*. The provenance of an annotated value is then correct if its partial order denotation is, in some sense, consistent with the total ordering of events specified by the global log.

The rest of the paper is structured as follows. In §2, we give an overview of the calculus, review its syntax and semantics, and provide examples to illustrate some of its features. We discuss what past information is exactly recorded by the provenance tracking reduction relation and study its correctness properties in §3. In §4, we review related work. In §5, we discuss possibilities for future work and conclude the paper.

[1] In a typical implementation of our language, we would assign the provenance tracking tier to a trusted underlying middleware.

2 The Provenance Calculus

The formalism we consider for studying provenance is a variant of the asynchronous π-calculus, which we extend with the following four main features:

1. **Explicit identities:** every process is *located* at a named *principal*, which is used for provenance and does not otherwise affect communication between processes.

2. **Annotated data:** every value is *annotated* with its *provenance*, which is updated as the computation proceeds to reflect what happened to the value.

3. **Provenance tracking:** we specify the meaning of the calculus by a *provenance tracking reduction semantics* which, in addition to describing the possible interactions between principals, also tracks the provenance of values as they are exchanged between principals.

4. **Pattern restricted input:** In order to allow principals to make use of the provenance information of values, we adopt a version of guarded choice that is restricted to inputs on the same channel with possibly differing patterns (similar in spirit to the one used in [6]). This allows principals to restrict the set of values they are willing to receive on a particular channel to those that satisfy the patterns specified. It also allows them to branch to different continuations based on the provenance information.

2.1 Syntax

The formal syntax of the calculus is given in Table 1. We assume a set $\mathcal{X}$ of *variables*, ranged over by $x, y, z, \ldots$, a set C of *channel names*, ranged over by $l, m, n, \ldots$, and a set $\mathcal{A}$ of *principal names*, ranged over by $a, b, c, \ldots$. We assume that all three sets are pair-wise disjoint and define the set $\mathcal{V}$ of *plain values* to be $C \cup \mathcal{A}$, and use the letters $u, v, \ldots$ to range over this set.

We represent provenance as a sequence of events. The events in a provenance sequence are assumed to be temporally ordered from left to right, where the left-most event (the head of the sequence) is the most recent event. We use $\mathcal{K}$ for the set of provenance sequences and $\mathcal{E}$ for the set of events and let $\kappa, \kappa', \ldots$ and $e, e', \ldots$ range over elements of each set respectively.

The set $\mathcal{D}$ of *annotated values* is then defined to be the set of terms of the form $v : \kappa$ where v is a plain value and κ its provenance. An output event $a!\kappa$ in the provenance of a value denotes that the value has been sent by principal a on a channel whose provenance is κ, while an input event $a?\kappa$ denotes that the value has been received by principal a on a channel whose provenance is κ. We also define the set $\mathcal{I}$ of identifiers to be $\mathcal{D} \cup \mathcal{X}$ and let $w, w', \ldots$ range over identifiers.

To allow principals to query the provenance of values we use patterns and pattern matching. Instead of defining a particular *pattern matching language*, we opt for a more general approach and make the calculus parametric on the choice of the pattern matching language. We do give a concrete language to use with the examples however.

Definition 1. A pattern matching language is a pair $(\Pi, \models)$ where Π is a set of patterns, ranged over by $\pi, \pi', \ldots$, and $\models \subseteq \mathcal{K} \times \Pi$ is the pattern satisfaction (or matching) relation, a relation between provenance sequences and patterns.

Processes in our calculus are based on those of the asynchronous π-calculus and are ranged over by $P, Q, \ldots$. The main differences with respect to the asynchronous π-calculus is our use of variants of the input and summation constructs. For input, we use a variant that takes two parameters in the form $\pi \,\mathsf{as}\, x$, where π is a pattern that is used to restrict the set of values to be received to those whose provenance matches the pattern π and x is the standard variable binder, a placeholder for the value to be received. We use the notation $w\,(\pi \,\mathsf{as}\, x).P$ for the process that is ready to receive, on channel w, a value whose provenance matches the pattern π and continue as P with the appropriate substitution. For summation, we use a restricted version that only allows input guarded choice on the same channel, denoted by $\Sigma_{i \in I} w\,(\pi_i \,\mathsf{as}\, x_i).P_i$ for some finite index set I. We use the notation 0 as syntactic sugar for the empty sum. The output form $w\langle w' \rangle$ denotes the process that is ready to send w' on channel w, while $\mathsf{if}\ w = w'\ \mathsf{then}\ P\ \mathsf{else}\ Q$ denotes the process that proceeds as P if w is equal to w' and as Q otherwise. Scope restriction of channel n to process P is denoted by $(\nu n)P$. Parallel composition of processes P and Q is denoted by $P \mid Q$ while $*P$ denotes replication of process P. It should be noted that all values in processes are annotated values, except channel names in restrictions. The reason for that is that we use restriction as usual to delimit scope, yet within one scope a channel name may have occurrences with possibly different provenance sequences.

A *system* is the composition of zero or more *located processes* and *messages*. We use $S, T, \ldots$ to range over systems. A located process, $a[P]$, is a process P that is running under the authority of a principal a. We overload the symbol 0 to denote the located process $a[0]$. As the structure of our systems is flat, it should be obvious from the context whether we mean by 0 the empty summation or the located process $a[0]$. A message is a value that has been sent but not yet received, and is denoted in the calculus by $n\langle\langle w' \rangle\rangle$. Restriction is denoted by $(\nu n)S$ while parallel composition of two systems is denoted by $S \parallel T$.

Table 1. *Syntax of the Provenance Calculus*

Processes		
$P ::= w\langle w\rangle$		Output
$\mid \quad \Sigma_{i\in I} w\,(\pi_i \text{ as } x_i).P_i$		input guarded sum
$\mid$	if $w = w$ then P else Q	Matching
$\mid \quad (\nu n)P$		Restriction
$\mid \quad P \mid Q$		Parallel composition
$\mid \quad * P$		Replication

Systems	
$S ::= a[P]$	Located process
$\mid \quad n\langle\!\langle w\rangle\!\rangle$	Message
$\mid \quad (\nu n)S$	Restriction
$\mid \quad S \parallel T$	Parallel composition

Provenance Sequences		
$\kappa ::= \epsilon$		Nil
$\mid \quad e$		Single event
$\mid \quad \kappa;\kappa$		Sequencing

Events	
$e ::= a!\kappa$	Output
$\mid \quad a?\kappa$	Input

Variables	$x, y, z \in \mathcal{X}$
Channels	$l, m, n \in \mathcal{C}$
Principals	$a, b, c \in \mathcal{A}$
Patterns	$\pi, \pi' \in \Pi$
Plain Values	$v, u \in \mathcal{V} \triangleq \mathcal{C} \cup \mathcal{A}$
Annotated Values	$v{:}\ \kappa \in \mathcal{D}$
Identifiers	$w, w' \in \mathcal{I} \triangleq \mathcal{D} \cup \mathcal{X}$

2.2 Provenance Tracking Semantics

The semantics of the calculus is defined by two relations, the *structural congruence relation*, $\equiv$, and the *provenance tracking reduction relation*, $\rightarrow$. Structural congruence allows us to make structural manipulations of systems which makes the definition of reduction simpler. The structural congruence relation is standard and is omitted due to space limitations.

Interaction between located processes is described by the reduction relation $\rightarrow$. As we mentioned previously, the reduction relation also tracks and updates the provenance of values as the system evolves. The reduction relation is defined on *closed* systems (i.e., those that contain no free variables) and is given in Table 2. The two main reduction rules are Red Snd and Red Rcv, which describe sending and receiving values respectively. The reason we split communication into the two steps of sending and receiving is to make the semantics simpler by only adding a single event to the provenance sequence at a time. In the rule Red Snd, a located process $a[m{:}\kappa_m\,\langle v{:}\ \kappa_v\rangle]$ may output the value $v{:}\ \kappa_v$ on the channel $m{:}\ \kappa_m$ which results in the message $m\langle\!\langle v{:}\ a!\kappa_m;\kappa_v\rangle\!\rangle$. What should be noted here is that the provenance of the value v changes from κ_v to $a!\kappa_m;\kappa_v$ after the output action to reflect the fact that the value has been most recently sent by principal a on a channel whose provenance is κ_m. In the rule Red Rcv, a message $m\langle\!\langle v{:}\ \kappa_v\rangle\!\rangle$ may be received by the located process $a[\Sigma_{i\in I} m{:}\ \kappa_m\,(\pi_i \text{ as } x_i).P_i]$ if the provenance of the value satisfies one of the patterns π_i. The process P_j whose pattern π_j is satisfied is chosen for the continuation. If more than one such pattern exists, one of them is chosen non-deterministically. Note that here too, the provenance of the value v is updated from κ_v to $a?\kappa_m;\kappa_v$ to reflect the fact that it has most recently been received by principal a on a channel with provenance κ_m. The value with the updated provenance is then substituted for the variable in the continuation chosen. The capture avoiding substitution of $w_1, \ldots, w_n$ for the free occurrences of variables $x_1, \ldots, x_n$ in P is denoted by $P\{^{w_1,\ldots,w_n}/_{x_1,\ldots,x_n}\}$ and its definition is standard. We let $\sigma, \sigma', \ldots$ range over substitutions. The rules Red IF$_t$ and Red IF$_f$ give the semantics of matching. It should be noted here that only the plain values are tested for equality while their provenance is ignored.[2] This means that if the two plain values are equal, irrespective of their provenances, the process in the then branch is chosen for the continuation as indicated by the rule Red IF$_t$. If the two plain values are not equal, then the process in the else branch is chosen for the continuation as indicated by the rule Red IF$_f$. The other three rules are standard and state that reduction is preserved under restriction, system composition as well as under structural congruence.

2.3 Examples

In this section, we will give a few examples to illustrate the use of the provenance calculus. First, however, we have to give a concrete pattern matching language.

2.3.1 A Sample Pattern Matching Language

As our provenance sequences have a structure that is similar to that of XML documents, we choose to base our sample pattern language on regular expression pattern matching [16]. The formal syntax and semantics of the language is given in Table 3.

The pattern Any matches any provenance sequence while the pattern ϵ is used to match the empty provenance sequence (denoted by ϵ as well). The two patterns $G!\pi$ and $G?\pi$ match send and receive events re-

[2] Depending on the intended application, the provenance of the two values tested could be useful and hence should be tracked in the continuation. This is not considered here however as it is not important for the aims of this paper.

Table 2. *Semantics of the Provenance Calculus - Reduction*

$$\textsc{Red Snd}\ \frac{}{a[m{:}\kappa_m\,\langle v{:}\ \kappa_v\rangle] \to m\langle\!\langle v{:}\ a!\kappa_m;\kappa_v\rangle\!\rangle} \qquad \textsc{Red Rcv}\ \frac{\kappa_v \models \pi_j}{a[\Sigma_{i\in I} m{:}\kappa_m\,(\pi_i \text{ as } x_i).P_i] \parallel m\langle\!\langle v{:}\ \kappa_v\rangle\!\rangle \to a[P_j\{^{v{:}\ a?k_m;\kappa_v}/_{x_j}\}]}$$

$$\textsc{Red If}_t\ \frac{}{a[\text{if } m{:}\ \kappa_m = m{:}\ \kappa'_m \text{ then } P \text{ else } Q\ ] \to a[P]} \qquad \textsc{Red If}_f\ \frac{}{a[\text{if } m{:}\ \kappa_m = n{:}\ \kappa_n \text{ then } P \text{ else } Q\ ] \to a[Q]}\ \text{if } m \neq n$$

$$\textsc{Red Res}\ \frac{S \to S'}{(\nu n)S \to (\nu n)S'} \qquad \textsc{Red Par}\ \frac{S \to T'}{S \parallel T \to T' \parallel T} \qquad \textsc{Red Str}\ \frac{S \equiv T \quad T \to T' \quad T' \equiv S'}{S \to S'}$$

Table 3. *The Sample Pattern Matching Language*

Syntax of patterns

$$\begin{array}{lll} \pi ::= \epsilon & \alpha ::= G!\pi \\ \quad\mid \alpha & \quad\mid G?\pi \\ \quad\mid \pi;\pi & G ::= a \\ \quad\mid \pi \vee \pi & \quad\mid \sim \\ \quad\mid \pi^* & \quad\mid G + G \\ \quad\mid \mathsf{Any} & \quad\mid G - G \end{array}$$

Satisfaction

$$\textsc{Sat Emp}\ \frac{}{\epsilon \models \epsilon} \qquad \textsc{Sat Snd}\ \frac{a \in [\![G]\!] \quad \kappa \models \pi}{a!\kappa \models G!\pi} \qquad \textsc{Sat Rcv}\ \frac{a \in [\![G]\!] \quad \kappa \models \pi}{a?\kappa \models G?\pi}$$

$$\textsc{Sat Cat}\ \frac{\kappa \models \pi \quad \kappa' \models \pi'}{\kappa;\kappa' \models \pi;\pi'} \qquad \textsc{Sat AltL}\ \frac{\kappa \models \pi}{\kappa \vee \kappa' \models \pi} \qquad \textsc{Sat AltR}\ \frac{\kappa' \models \pi}{\kappa \vee \kappa' \models \pi}$$

$$\textsc{Sat Rep}\ \frac{\forall i \in 1 \ldots n.\kappa_i \models \pi}{\kappa_1;\ldots;\kappa_n \models \pi^*} \qquad \textsc{Sat Any}\ \frac{}{\kappa \models \mathsf{Any}}$$

Definition of $[\![-]\!]$

$$[\![a]\!] = \{a\} \qquad [\![\sim]\!] = \mathcal{A} \qquad [\![G + G']\!] = [\![G]\!] \cup [\![G']\!] \qquad [\![G - G']\!] = [\![G]\!] \setminus [\![G']\!]$$

spectively. The use of *group expressions* G in these patterns allows us to perform more general tests against the principal that performed the event. The group expression a denotes the singleton set containing principal a only, while $\sim$ denotes the set of all principals. $G + G'$ and $G - G'$ denote union and difference of groups respectively. The denotation of group expressions is given by the function $[\![-]\!]$. The pattern $\pi;\pi'$ matches a provenance sequence that is composed of two parts that match π and π' respectively. The alternation of patterns π and π', denoted by $\pi \vee \pi'$, matches a sequence that matches either patterns, while the repetition of pattern π, denoted by π^*, matches any provenance sequence that is the composition of zero or more sub-sequences, each of which matches the pattern π.

2.3.2 Example Systems

Authentication. Provenance can be used to establish the authenticity of messages, for example by checking their immediate sender, their original sender or any principal in between. The following system illustrates this.

$$a[m\,(c!\mathsf{Any};\mathsf{Any}\text{ as }x).P] \parallel b[m\,(\mathsf{Any};d!\mathsf{Any}\text{ as }y).Q] \parallel S$$

In the above example, principal a wants to receive only data coming from c directly, no matter where it has been before, whereas principal b wants to receive data that originated at d, no matter what the intermediary links were.

Auditing. Provenance can also be used as an auditing and troubleshooting tool to establish who might have been responsible for an error. For example, in the following system:

$$S \triangleq a[m\,\langle v\rangle] \parallel s[m\,(x).n'\,\langle x\rangle] \parallel c[n'\,(x).P] \parallel b[n''\,(x).Q]$$

principal a is trying to send a value v to principal b, and this has to be done through an intermediary s (because a does not have a direct link to b for example). Because of faulty code at s, the value gets forwarded to c instead, as indicated by the following reduction:

$$S \to^* c[P\{^{v{:}\ c?\epsilon;s!\epsilon;s?\epsilon;a!\epsilon}/_x\}] \parallel b[n''\,(x).Q]$$

Now when c detects the error, perhaps due to the unexpected value, c can use the provenance $c?\epsilon; s!\epsilon; s?\epsilon; a!\epsilon$ to tell what principals were involved in making this error. In this case, a, s, and c itself. The three principals may be further investigated to determine who and what exactly caused the error.

Photography Competition. The following example describes a photography competition. Contestants submit their entries for the competition to the organiser of the competition, who forwards those entries to the appropriate judges. Each judge rates the entries allocated to them and returns the results to the organiser. The organiser then publishes the results and announces the winners. We consider a version of the competition with

three contestants: c_1, c_2 and c_3, one organiser: o, and two judges: j_1 and j_2. The contestants submit their entries to the organiser on channel *sub* and receive the published results on channel *pub*. The organiser forwards entries submitted by c_1 and c_3 to judge j_1 and the entry by c_2 to j_2. The judges return the entries together with their ratings to the organiser. Note that below we are using polyadic versions of the send and receive constructs, such an extension to the calculus being straightforward.

$$\begin{aligned}
\mathbf{C}(c, entry, P) &\triangleq c[sub\,\langle entry\rangle \mid \\
&\qquad pub\,(\mathsf{Any}; c!\mathsf{Any\ as}\ x\ ,\ \mathsf{Any\ as}\ y).P] \\
\mathbf{O} &\triangleq o[*\,(\Sigma_{i\in\{1,2\}} sub\,(\pi_i\ \mathsf{as}\ x).in_i\,\langle x\rangle \mid \\
&\qquad res\,(y,z).*\,pub\,\langle y,z\rangle)] \\
\mathbf{J}(j, inc) &\triangleq j[inc\,(x).res\,\langle x, rate(x)\rangle] \\
\mathbf{Comp} &\triangleq \mathbf{C}(c_1,e_1,P_1) \parallel \mathbf{C}(c_2,e_2,P_2) \parallel \mathbf{C}(c_3,e_3,P_3) \parallel \\
&\qquad \mathbf{O} \parallel \mathbf{J}(j_1, in_1) \parallel \mathbf{J}(j_2, in_2)
\end{aligned}$$

where $\pi_1 \triangleq (c_1 + c_3)!\mathsf{Any};\mathsf{Any}$ and $\pi_2 \triangleq c_2!\mathsf{Any};\mathsf{Any}$. The system above evolves as follows, where $\Pi_{i\in\{1,\ldots,n\}} S_i$ is used to denote $S_1 \parallel \ldots \parallel S_n$.

$$\begin{aligned}
\mathbf{Comp} \rightarrow^* \Pi_{i\in\{1,2,3\}} o[*\,pub\,\langle e_i\colon \kappa_{ei}, rate(e_i)\colon \kappa_{ri}\rangle] \parallel \\
\mathbf{O} \parallel \Pi_{i\in\{1,2,3\}} c_i[P_i\{^{e_i\colon \kappa'_{ei}, rate(e_i)\colon \kappa'_{ri}}/_{x,y}\}]
\end{aligned}$$

where:

$$\kappa_{ei} \triangleq \begin{cases} o?\epsilon; j_1!\epsilon; j_1?\epsilon; o!\epsilon; o?\epsilon; c_i!\epsilon & \text{if } i \in \{1,3\} \\ o?\epsilon; j_2!\epsilon; j_2?\epsilon; o!\epsilon; o?\epsilon; c_i!\epsilon & \text{if } i \in \{2\} \end{cases}$$

$$\kappa_{ri} \triangleq \begin{cases} o?\epsilon; j_1!\epsilon & \text{if } i \in \{1,3\} \\ o?\epsilon; j_2!\epsilon & \text{if } i \in \{2\} \end{cases}$$

$$\kappa'_{ei} \triangleq c_i?\epsilon; o!\epsilon; \kappa_{ei}$$

$$\kappa'_{ri} \triangleq c_i?\epsilon; o!\epsilon; \kappa_{ri}$$

After receiving the results from the judges, the organiser publishes them on channel *pub* as shown in the replicated output processes $*\,pub\,\langle e_i\colon \kappa_{ei}, rate(e_i)\colon \kappa_{ri}\rangle$ for $i \in \{1,2,3\}$. Every contestant listens on channel *pub* to receive their own results. This is achieved by the input construct $pub\,(\mathsf{Any}; c!\mathsf{Any\ as}\ x\ ,\ \mathsf{Any\ as}\ y)$. The pattern specifications allow the contestant to receive the pair containing the result for their own entry.

3 Properties of Provenance

In this section, we look at the properties of the provenance notion and the provenance tracking reduction relation we proposed.

3.1 Logs

We start by introducing *logs*. Logs are meant as a representation of the past behaviour of systems. They may be thought of as edge-labelled trees whose directed edges (from parent to child) are labelled with *actions* that occurred in a system at some point in the past. An edge leading out of a parent node represents an action that occurred more recently in time than those leading out of its children. Sibling subtrees are assumed to be *temporally independent*, in the sense that the order between their actions is unknown. The formal syntax of logs is as follows.

$$\begin{aligned}
&\phi ::= \varnothing \mid \alpha;\phi \mid \phi\,|\,\psi \\
&\alpha ::= a.\mathsf{snd}(V,V) \mid a.\mathsf{rcv}(V,V) \mid a.\mathsf{ift}(V,V) \mid a.\mathsf{iff}(V,V)
\end{aligned}$$

We use Φ for the set of logs, $\phi, \psi, \ldots$ to range over logs and $\alpha, \beta, \ldots$ to range over actions. We also define the set $\mathcal{D}_x$ to be $\mathcal{V} \cup \mathcal{X} \cup \{\star\}$ and use the metavariables $U, V, \ldots$ to range over its elements. Variables in logs are used to denote unknown values, so for example, action $a.\mathsf{snd}(x, v)$ says that principal a sent value v on some channel x, the identity of which is unknown. The special symbol $\star$ denotes an unknown private channel name, and its use will be demonstrated later. We use $\varnothing$ for the empty log, $\alpha;\phi$ for the log with edge labelled α leading to subtree ϕ, and $\phi\,|\,\psi$ for the composition of logs ϕ and ψ. Note that the composition of logs $\phi\,|\,\psi$ joins their roots and hence results in a well-formed tree. Operator ';' has higher precedence than '|' and parentheses may be used in the usual way. We also omit trailing $\varnothing$ to ease readability. We have four types of actions; (1) the *output* action, $a.\mathsf{snd}(V, V')$, which says that principal a sent V' on V, (2) the *input* action, $a.\mathsf{rcv}(V, V')$, which says that principal a received V' on V, (3) the *if true* action, $a.\mathsf{ift}(V, V')$, which says that principal a tested V and V' for equality and this returned true, and (4) the *if false* action, $a.\mathsf{iff}(V, V')$, which says that principal a tested V and V' for equality and this returned false. Note that, in the log $a.\mathsf{snd}(x, V);\phi$, the occurrence of variable x in action $a.\mathsf{snd}(x, V)$ binds occurrences of x in ϕ. The same holds for x in $a.\mathsf{rcv}(x, V);\phi$. All other occurrences of variables in logs are considered free. We use $fv(\phi)$ for the set of free variables in ϕ, $bv(\phi)$ for the set of bound variables in ϕ and define closed logs to be those containing no free variables. Furthermore, we consider indistinguishable those logs which only differ by alpha conversion (change of bound variables) or by application of the commutative monoid laws for the log composition operator | (with identity element $\varnothing$).

As we have already mentioned, logs are meant as records of the past behaviour of systems. It is natural then to ask: what is the relation between two logs ϕ and ψ that both record the past of the same system? To answer this question, we start by defining $\alpha \preceq \alpha'$ to mean

that there exists a (possibly empty) substitution σ of values for variables such that $\alpha' = \alpha\sigma$. Note that since we are requiring σ to strictly substitute variables with values, then α' must be just like α except that it has less variables. Now, since we are using variables to stand for unknown values, this simply means that α' contains as much or more information about the past as that contained in α. With this, we proceed to overload the symbol $\preceq$ and define the relation $\preceq$ on the set of closed logs. Intuitively, $\phi \preceq \psi$ can be taken to mean that log ψ tells us at least as much about the past as log ϕ does. Relation $\preceq$ is defined to be the smallest relation closed under the following inference rules, where σ and σ' denote *closing* substitutions.

$$\textsc{LNil}\ \frac{}{\varnothing \preceq \phi} \qquad \textsc{LPre}_1\ \frac{\alpha \preceq \alpha' \quad \phi\sigma \preceq \psi\sigma'}{\alpha;\phi \preceq \alpha';\psi} \qquad \textsc{LPre}_2\ \frac{\phi \preceq \psi}{\phi \preceq \alpha;\psi}$$

$$\textsc{LCmp}_1\ \frac{\phi \preceq \psi \quad \phi' \preceq \psi}{\phi \,|\, \phi' \preceq \psi} \qquad \textsc{LCmp}_2\ \frac{\phi \preceq \psi}{\phi \preceq \psi \,|\, \psi'}$$

The empty log $\varnothing$ tells us nothing about the past, and hence we have that $\varnothing \preceq \phi$ for every log ϕ as expressed by Rule LNil. Rule $\textsc{LPre}_1$ can be understood as follows: if we have two logs $\alpha;\phi$ and $\alpha';\psi$, then in order to prove that $\alpha;\phi \preceq \alpha';\psi$ holds, we need to prove that both $\alpha \preceq \alpha'$ and $\phi\sigma \preceq \psi\sigma'$ hold. Note that here we are using $\phi\sigma$ and $\psi\sigma'$ to denote the application of the (possibly empty) closing substitutions σ and σ' to logs ϕ and ψ respectively and that this is needed as $\preceq$ only applies to closed logs. Rule $\textsc{LPre}_2$ simply says that if $\phi \preceq \psi$, then adding an action at the beginning of ψ will only add more information to it, and hence, $\phi \preceq \alpha;\psi$ holds too. Rule $\textsc{LCmp}_1$ says that $\phi \,|\, \phi' \preceq \psi$ holds if it is the case that both $\phi \preceq \psi$ and $\phi' \preceq \psi$ hold. The definition of $\textsc{LCmp}_1$ implies that we are taking a nonlinear interpretation of logs, that is we allow ϕ and ϕ' in $\phi \,|\, \phi'$ to reference the same actions, in effect duplicating the information conveyed about the past. This interpretation is required as the calculus allows the copying of values and their provenance. Rule $\textsc{LCmp}_2$ says that in order to show that $\phi \preceq \psi \,|\, \psi'$, we need to show that $\phi \preceq \psi$ holds.

To illustrate the application of the above rules, let us consider the two logs $\phi \triangleq a.\mathsf{snd}(x,v);a.\mathsf{rcv}(n,x)$ and $\psi \triangleq a.\mathsf{snd}(m,v);a.\mathsf{rcv}(n,m)$. Log ϕ tells us that a sent v on some unknown channel x which it received on channel n while ψ tells us that a sent v on channel m which it received on channel n. Clearly ψ tells us more information about the past than ϕ and the unknown channel x in ϕ is actually channel m (inferred from ψ). To show this using the inference rules above, we first apply Rule $\textsc{LPre}_1$ as follows:

$$\frac{a.\mathsf{snd}(x,v) \preceq a.\mathsf{snd}(m,v) \qquad a.\mathsf{rcv}(n,x)\{^m/_x\} \preceq a.\mathsf{rcv}(n,m)}{a.\mathsf{snd}(x,v);a.\mathsf{rcv}(n,x) \preceq a.\mathsf{snd}(m,v);a.\mathsf{rcv}(n,m)}$$

Then, we show that $a.\mathsf{snd}(x,v) \preceq a.\mathsf{snd}(m,v)$ and that $a.\mathsf{rcv}(n,m) \preceq a.\mathsf{rcv}(n,m)$. The former holds since $a.\mathsf{snd}(x,v)\{^m/_x\} = a.\mathsf{snd}(m,v)$ whereas the latter may be shown to hold by further application of $\textsc{LPre}_1$ and LNil.

Proposition 1. *The relation $\preceq$ is a partial order.*

3.2 Denotation of Provenance

We interpret the provenance κ in an annotated value $v\colon \kappa$ to be a set of assertions about the past of the value v. These assertions tell us about events that took place in the system and that are relevant to the value v. For example, consider the value $v\colon a?\kappa; b!\kappa'; \kappa''$, its provenance tells us that (a) v was most recently received by a on a channel whose provenance was κ; (b) before that, it was sent by b on a channel with provenance κ'; (c) before that, it had provenance κ''; (d) κ and κ' in turn tell us about the past of the two channels used by a and b, while κ'' tells us about the past of v before it was sent by b . It is important here to note that the provenance of v does not reveal the identities of the channels used for communication, nor does it tell us about the ordering between events in κ and those in $b!\kappa', \kappa''$ or between events in κ' and those in κ''.

Assertions such as those above may be encoded in a "condensed form" as logs. We define the function $[\![-]\!]$ which maps $V\colon \kappa$ to a log ϕ that represents what the provenance sequence κ tells us about the past of V. The definition of $[\![-]\!]$ in Definition 2 below uses a different grammar for provenance sequences than that given in Table 1 in order to make the definition of $[\![-]\!]$ simpler. It is easy to check, however, that for our purposes the two grammars are equivalent.

Definition 2 (Denotation of provenance)**.** The function $[\![-]\!] : \mathcal{D}_x \to \Phi$ is defined inductively on the structure of provenance sequences as follow:

$$\begin{aligned}
[\![V\colon \epsilon]\!] &= \varnothing \\
[\![V\colon a!\kappa';\kappa]\!] &= a.\mathsf{snd}(x,V);([\![V\colon \kappa]\!] \mid [\![x\colon \kappa']\!]) \\
[\![V\colon a?\kappa';\kappa]\!] &= a.\mathsf{rcv}(x,V);([\![V\colon \kappa]\!] \mid [\![x\colon \kappa']\!])
\end{aligned}$$

The empty provenance sequence ϵ in an annotated value $V\colon \epsilon$ tells us that V originated here, and hence $[\![V\colon \epsilon]\!]$ is taken to be the empty log $\varnothing$. The annotated values $V\colon a!\kappa';\kappa$ and $V\colon a?\kappa';\kappa$ denote that V was sent (respectively received) on some unknown channel x, and that before that the logs denoted by $[\![V\colon \kappa]\!]$ and $[\![x\colon \kappa']\!]$ occurred in some unknown order. It is clear that a log $[\![V\colon \kappa]\!]$ is a partial record of what took place in a system. Indeed, it does not contain information about the identities of the channels used for communication, and it lacks information about the order in which some actions took place. This is to be expected of course as the provenance of a value is meant to only record information about actions that are relevant to the value itself.

3.3 Monitored Systems

In order to be able to assess the correctness and completeness of provenance, we introduce the notion of *monitored systems*. A monitored system is one where every action that takes place is recorded in a *global log*. The global log provides a repository where every action is logged and whose content is not accessible by principals. It is only meant as a proof tool against which the properties of provenance can be formulated and judged.

We use $M, N, \ldots$ to range over monitored systems and give their formal syntax below.

$$M, N ::= \phi \triangleright S \mid (\nu n)M \mid M \parallel S$$

The notation $\phi \triangleright S$ denotes the monitored system composed of global log ϕ and system S. Restriction $(\nu n)M$ and parallel composition $M \parallel S$ are needed to allow the global log to behave like other parts of the system with respect to scope extrusion and intrusion. More specifically, the form $(\nu n)M$ allows channel scopes to be extruded to include the log while the form $M \parallel S$ is needed to allow channels whose scope includes the log but not some part of the system S. Note that indeed the above syntax allows exactly one global log per monitored system. We also define versions of structural congruence and reduction for monitored systems and denote them by $\equiv_m$ and $\rightarrow_m$ respectively. These are given in Table 4. It should clear from the definition of $\rightarrow_m$ that the original provenance tracking semantics of systems is preserved, and that all $\rightarrow_m$ adds is the recording of actions in the global log. We formalise this in Proposition 2. This latter makes use of the *log erasure function*, $|-|$, which given a monitored system, removes the log and returns just the system part of it. The function $|-|$ is defined inductively on the structure of monitored systems as follows.

$$|\phi \triangleright S| = S \quad |(\nu n)M| = (\nu n)|M| \quad |M \parallel S| = |M| \parallel S$$

Proposition 2. *$M \rightarrow_m M'$ implies that $|M| \rightarrow |M'|$. Vice versa, $|M| \rightarrow S$ implies that $M \rightarrow_m M'$ for some M' such that $|M'| = S$.*

3.4 Correctness

The first provenance property we look at is correctness. A provenance sequence κ in an annotated value $v : \kappa$ is considered correct if what it tells us about the past of v agrees with what actually took place. This is defined relative to the global log which is assumed to be a correct and complete record of the past of a system. If every value in a system has correct provenance, then we say that the system as a whole has correct provenance. Theorem 1 states that provenance correctness is preserved by the reduction relation. That is, starting from a system with correct provenance, we are guaranteed to get a system with correct provenance after reduction. Definition 3 makes use of two auxiliary functions: $log(-)$, which returns the global log of a monitored system, and $values(-)$, which returns the set of "annotated values" of a monitored system. The definition of $log(-)$, by induction on the structure of monitored systems, is straightforward and hence omitted. This is mostly the case for $values(-)$ too and hence, we only discuss the most interesting cases below. The set of values in a monitored system is defined to be that in its system part (i.e., we ignore the global log and top level restrictions). This is expressed by the following three rules:

$$values(\phi \triangleright S) \triangleq values(S) \quad values((\nu n)M) \triangleq values(M)$$
$$values(M \parallel S) \triangleq values(M) \cup values(S)$$

For systems, we proceed simply by gathering annotated values, that is "subterms" of the form $v : \kappa$, and substituting $\star$ for any restricted channel names. So for example, we have that $values(a[P]) \triangleq values(P)$, $values(m\langle\!\langle v : \kappa \rangle\!\rangle) \triangleq \{v : \kappa\}$, $values(S \parallel S') \triangleq values(S) \cup values(S')$, and $values((\nu n)S) \triangleq values(S)\{^{\star}/_{n}\}$. Note that restriction here is treated differently from that at the top level of monitored systems. The rationale behind this discrepancy is that restricted names at the top level are known to the global log whereas those occurring here are not. The substitution is done to avoid any clashes with names appearing in the log and is inspired by [17]. Definition of $values(P)$ for processes P is similar.

Definition 3. A monitored system M *has correct provenance* if for all $V : \kappa$ in $values(M)$, we have that $[\![V : \kappa]\!] \preceq log(M)$.

Lemma 1. *M has correct provenance and $M \equiv_m M'$ implies that M' has correct provenance.*

Theorem 1. *(Provenance correctness). M has correct provenance and $M \rightarrow_m M'$ implies that M' has correct provenance.*

Proof. We assume that M has correct provenance and that $M \rightarrow_m M'$. We show that M' has correct provenance by induction on the derivation of $M \rightarrow_m M'$. The details of the proof are straightforward and therefore omitted. □

3.5 Towards a Notion of Completeness

We have already shown informally that the provenance of a value is not a complete record of what took place in the whole system. We formalise this below.

Definition 4. A monitored system M *has complete provenance* if for all $V : \kappa$ in $values(M)$, we have that $log(M) \preceq [\![V : \kappa]\!]$.

Proposition 3 (Provenance incompleteness)**.** *M has complete provenance and $M \rightarrow_m M'$ does not imply that M' has complete provenance.*

Table 4. *Semantics of Monitored Systems - Reduction*

Structural Congruence

$$S \equiv T \implies \phi \triangleright S \equiv_m \phi \triangleright T \qquad (\phi \triangleright S) \parallel S' \equiv_m \phi \triangleright (S \parallel S') \qquad M \equiv_\alpha M' \implies M \equiv_m M'$$

$$(\nu n)(\nu m)M \equiv_m (\nu m)(\nu n)M \qquad (\nu n)M \parallel S \equiv_m (\nu n)(M \parallel S) \text{ if } n \notin fn(S) \qquad M \parallel (\nu n)S \equiv_m (\nu n)(M \parallel S) \text{ if } n \notin fn(M)$$

Reduction Relation

$$\text{MRed Snd}\ \frac{}{\phi \triangleright a[m\colon \kappa_m\langle v\colon \kappa_v\rangle] \to_m a.\mathsf{snd}(m,v); \phi \triangleright m\langle\!\langle v\colon a!\kappa_m, \kappa_v\rangle\!\rangle}$$

$$\text{MRed Rcv}\ \frac{\kappa_v \models \pi_j}{\phi \triangleright a[\Sigma_{i\in I} m\colon \kappa_m(x_i,\pi_i).P_i] \parallel m\langle\!\langle v\colon \kappa_v\rangle\!\rangle \to_m a.\mathsf{rcv}(m,v); \phi \triangleright a[P_j\{^{v\colon a?\kappa_m,\kappa_v}/_x\}]}$$

$$\text{MRed If}_t\ \frac{}{\phi \triangleright a[\mathsf{if}\ m\colon k_m = m\colon k'_m\ \mathsf{then}\ P\ \mathsf{else}\ Q\ ] \to a.\mathsf{ift}(m,m); \phi \triangleright a[P]}$$

$$\text{MRed If}_f\ \frac{}{\phi \triangleright a[\mathsf{if}\ m\colon k_m = n\colon k_n\ \mathsf{then}\ P\ \mathsf{else}\ Q\ ] \to a.\mathsf{iff}(m,n); \phi \triangleright a[Q]}\ \text{if } m \neq n$$

$$\text{MRed Res}\ \frac{M \to_m M'}{(\nu n)M \to_m (\nu n)M'} \qquad \text{MRed Par}\ \frac{M \to_m M'}{M \parallel S \to_m M' \parallel S} \qquad \text{MRed Str}\ \frac{M \equiv_m N \quad N \to_m N' \quad N' \equiv_m M'}{M \to_m M'}$$

Proof. Let us consider the monitored system M defined as follows:

$$M \triangleq \varnothing \triangleright a[m\colon \epsilon\langle v\colon \epsilon\rangle] \parallel b[m\colon \epsilon(x).P]$$

Assuming P has complete provenance, then it is clear that M has too. Now, we have that $M \to_m M'$ where M' is defined as follows:

$$M' \triangleq a.\mathsf{snd}(m,v); \varnothing \triangleright m\langle\!\langle v\colon a!\kappa_m\rangle\!\rangle \parallel b[m\colon \epsilon(x).P]$$

M' does not have complete provenance, since $m\colon \epsilon$ does not. □

Completeness is too strong since we are requiring every value to have complete information about the past of the whole system. In fact, in general it is not even possible for a single value, as it can be easily demonstrated by considering a system where a value gets forgotten at some point in the evolution of the system, such as the following:

$$\phi \triangleright a[m\colon \kappa_m(x).0] \parallel m\langle\!\langle v\colon \kappa_v\rangle\!\rangle \parallel S$$

where, assuming that S does not contain copies of $v\colon \kappa_v$, the system will evolve to a state where $v\colon \kappa_v$ is completely forgotten.

4 Related Work

Provenance has been studied in a variety of settings, most prominently in databases and scientific computing. A major theme of provenance research in the database community has been that of identifying parts of the input of a query *relevant* to parts of its output. The definition of the provenance notion (i.e., what "relevant" means) as well as the granularity at which data is considered (i.e., what "parts" of the input or output constitute) varies depending on the intended application.

One of the earliest works that studied provenance for database management systems was by Wang and Madnick in their *Polygen model* [24], which aimed to resolve provenance issues for data composed from multiple sources. Woodruff and Stonebraker [25] studied a notion of provenance called *lineage*. Their aim was to provide fine-grained lineage information, which they argued was prohibitively expensive to achieve using metadata. Their approach instead relied on a small amount of information about the database operations performed and the analysis of base data. Buneman et al. [5] studied provenance in the setting of a data model that generalises both relational as well hierarchical databases. They proposed two different notions of provenance, which they termed "why" and "where" provenance, and gave algorithms to compute the two types of provenance information.

Motivated by the need for formal foundations of provenance, Cheney et al. [7] proposed a semantic characterisation of provenance based on dependency analysis. They showed that minimal dependency provenance is not computable and provided dynamic and static techniques to approximate it. Green et al. [11] introduced a model based on semiring valued annotations and showed that this model generalises several models of annotated relations including why provenance.

There is a big incentive to support provenance in scientific computing settings. This stems from the need to allow scientists to understand, analyse and replicate results of experiments. Some of the projects aimed at develop-

ing middleware to support provenance in this setting include Chimera [9] (physics and astronomy), myGrid [23] (biology), CMCS [21] (chemical sciences), and ESSW [10] (earth sciences). Bose and Frew [2] and Simmhan et al. [22] give surveys of the most important research efforts in this area.

5 Conclusion and Future Work

We presented a calculus for provenance in distributed systems. The semantics of the calculus tracks the provenance of values dynamically and allows principals to perform tests against it. These tests let principals decide what data to consume and what to do with it based on its provenance. We also discussed the semantics of provenance and studied some of its properties.

We proposed a characterisation of provenance correctness and completeness based on the notions of monitored systems and global logs. We showed that based on this characterisation, our provenance is correct but incomplete. However, as we have mentioned, completeness is too strong since we are requiring the provenance of every value to be a complete record of everything that has taken place within the system. Instead of asking for the provenance to be complete, what is needed is for provenance to be *adequate*. What we mean by this is that the provenance information recorded has to be enough for its intended application. In developing such an idea, we are interested in using information about the role each principal played in getting a piece of data to its current form (and the implied trust relations between principals), as a measure of how trustworthy a piece of data is likely to be. Hence, we only need to know about actions that are relevant to the value at hand, and we only need to know about the principals involved in the action. We are working on formalising adequacy at the moment. We also plan to investigate other ways of characterising provenance properties, for example, based on logic.

Provenance tracking is performed dynamically at the moment as part of the reduction relation. Although this yields accurate provenance information, it results in runtime overhead as provenance is computed, updated and tests are performed against it. We are working on a static analysis that would alleviate the need for dynamic provenance tracking. The idea behind it is to analyse the flow of data between principals and make sure that principals would only receive data with provenance that matches their expectations.

To keep the calculus simple, we only allowed statically defined patterns with no binding variables. This means that principals cannot use dynamic information for their provenance tests nor can they extract part (or all) of the provenance sequence and use it as data. This is one of the first extensions we aim to make to the calculus. In addition to this, provenance is currently tracked by the runtime environment, and principals are only given "read-only" access to it. Moreover, every principal is able to see the entire provenance sequence, regardless of the privacy and security policies of other principals. However, in many applications, principals may wish to control the disclosure of provenance information about them. We are working on extending the calculus with such features at the moment. We are also investigating how to extend the calculus to allow the communication of data structures. With this, we aim to provide support for scenarios where data is not only copied and communicated, but also broken down into pieces, combined with other data, and generally transformed to produce new data. In these scenarios, the ways in which we may manipulate data usually depend on its provenance and vice versa. This means that a transformation f of values v_1 and v_2 needs to take into account their provenances κ_1 and κ_2 as well. That is, the result of the transformation should be of the form $f(v_1, \kappa_1, v_2, \kappa_2)$ as opposed to $f(v_1, v_2)$. The same holds for transformations $\bar{f}$ that manipulate the provenance. To be able to express this, we aim to extend the calculus, along similar lines as the applied π-calculus [1], with data structures and operations on them.

References

[1] Abadi, M., and Fournet, C. Mobile values, new names, and secure communication. *ACM SIGPLAN Notices 36*, 3 (Mar. 2001), 104–115.

[2] Bose, R., and Frew, J. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys 37*, 1 (Mar. 2005), 1–28.

[3] Boudol, G. Asynchrony and the π-calculus. Tech. Rep. 1702, INRIA, Sophia-Antipolis, 1992.

[4] Buneman, P., Cheney, J., and Vansummeren, S. On the expressiveness of implicit provenance in query and update languages. In *ICDT* (2007), vol. 4353 of *LNCS*, Springer, pp. 209–223.

[5] Buneman, P., Khanna, S., and Tan, W. C. Why and where: A characterization of data provenance. In *ICDT* (2001), vol. 1973 of *LNCS*, Springer, pp. 316–330.

[6] Castagna, G., Nicola, R. D., and Varacca, D. Semantic subtyping for the pi-calculus. *Theoretical Computer Science 398*, 1-3 (2008), 217–242.

[7] Cheney, J., Ahmed, A., and Acar, U. A. Provenance as dependency analysis. In *Database Programming Languages* (2007), vol. 4797 of *LNCS*, Springer, pp. 138–152.

[8] Cui, Y., and Widom, J. Practical lineage tracing in data warehouses. In *ICDE* (Mar. 2000), IEEE Computer Society, pp. 367–378.

[9] Foster, I. T., Vöckler, J.-S., Wilde, M., and Zhao, Y. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Statistical and Scientific Database Management* (2002), IEEE Computer Society, pp. 37–46.

[10] Frew, J., and Bose, R. Earth system science workbench: A data management infrastructure for earth science products. In *Statistical and Scientific Database Management* (2001), IEEE Computer Society, pp. 180–189.

[11] Green, T. J., Karvounarakis, G., and Tannen, V. Provenance semirings. In *PODS* (2007), ACM, pp. 31–40.

[12] Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., and Oinn, T. Provenance of e-science experiments - experience from bioinformatics. Proceedings of the UK OST e-Science Second All Hands Meeting, Jan. 2003.

[13] Groth, P., Munroe, S., Miles, S., and Moreau, L. *In Lucio Grandinetti (ed.), HPC and Grids in Action*. IOS Press, Jan. 2008, ch. Applying the Provenance Data Model to a Bioinformatics Case.

[14] Hennessy, M., and Riely, J. Resource access control in systems of mobile agents. *Information and Computation 173*, 1 (Feb 2002), 82–120.

[15] Honda, K., and Tokoro, M. An object calculus for asynchronous communication. In *ECOOP* (1991), Springer-Verlag, pp. 133–147.

[16] Hosoya, H., and Pierce, B. C. Regular expression pattern matching. In *POPL* (2001). Full version in *Journal of Functional Programming*, 13(6), Nov. 2003, pp. 961–1004.

[17] Lhoussaine, C., and Sassone, V. A dependently typed ambient calculus. In *ESOP* (2004), vol. 2986 of *LNCS*, Springer, pp. 171–187.

[18] Milner, R. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.

[19] Moreau, L., Groth, P., Miles, S., Vazquez, J., Ibbotson, J., Jiang, S., Munroe, S., Rana, O., Schreiber, A., Tan, V., and Varga, L. The Provenance of Electronic Data. *Communications of the ACM* (Apr. 2008).

[20] Muniswamy-Reddy, K.-K., Holland, D. A., Braun, U., and Seltzer, M. I. Provenance-aware storage systems. In *USENIX Annual Technical Conference, General Track* (2006), USENIX, pp. 43–56.

[21] Pancerella, C., Myers, J., and Rahn, L. Data provenance in the collaboratory for multiscale chemical science (CMCS). *Workshop on Data Derivation and Provenance* (Jan 2002).

[22] Simmhan, Y. L., Plale, B., and Gannon, D. A survey of data provenance in e-science. *SIGMOD Record 34*, 3 (2005), 31–36.

[23] Stevens, R. D., Robinson, A. J., and Goble, C. A. myGrid: personalised bioinformatics on the information grid. In *ISMB (Supplement of Bioinformatics)* (2003), pp. 302–304.

[24] Wang, Y. R., and Madnick, S. E. A polygen model for heterogeneous database systems: The source tagging perspective. In *VLDB* (1990), Morgan Kaufmann Publishers Inc., pp. 519–538.

[25] Woodruff, A., and Stonebraker, M. Supporting fine-grained data lineage in a database visualization environment. In *Proceedings of the Thirteenth International Conference on Data Engineering* (1997), IEEE Computer Society, pp. 91–102.