

Collaborative data-driven workflows: think global, act local

Serge Abiteboul, Victor Vianu

▶ **To cite this version:**

Serge Abiteboul, Victor Vianu. Collaborative data-driven workflows: think global, act local. Proceedings of the 32nd symposium on Principles of database systems, Jun 2013, New York, NY, USA, United States. pp.91–102, 10.1145/2463664.2463672 . hal-00840306

HAL Id: hal-00840306
https://inria.hal.science/hal-00840306

Submitted on 2 Jul 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Collaborative Data-Driven Workflows: Think Global, Act Local

Serge Abiteboul
INRIA Saclay & ENS Cachan
serge.abiteboul@inria.fr

Victor Vianu
U.C. San Diego
vianu@cs.ucsd.edu

Abstract

We introduce and study a model of *collaborative data-driven workflows*. In a local-as-view style, each peer has a partial view of a global instance that remains purely virtual. Local updates have side effects on other peers' data, defined via the global instance. We also assume that the peers provide (an abstraction of) their specifications, so that each peer can actually see and reason on the specification of the entire system.

We study the ability of a peer to carry out runtime reasoning about the global run of the system, and in particular about actions of other peers, based on its own local observations. A main contribution is to show that, under a reasonable restriction (namely, *key-visibility*), one can construct a finite symbolic representation of the infinite set of global runs consistent with given local observations. Using the symbolic representation, we show that we can evaluate in PSPACE a large class of properties over global runs, expressed in an extension of first-order logic with past linear-time temporal operators, PLTL-FO. We also provide a variant of the algorithm allowing to *incrementally* monitor a statically defined property, and then develop an extension allowing to monitor an infinite class of properties sharing the same temporal structure, defined dynamically as the run unfolds. Finally, we consider an extension of the language, augmeting workflow control with PLTL-FO formulas. We prove that this does not increase the power of the workflow specification language, thereby showing that the language is closed under such introspective reasoning.

1 Introduction

Process-centric workflows focus on control flow, often abstracting away data almost entirely. In contrast, recently proposed data-driven workflows treat data as first-class citizens, e.g., the *business artifact model* pioneered in [20] and deployed by IBM in commercial products. Data-driven workflows have become ubiquitous in a wide array of application domains. Their system architecture may range from totally centralized to fully distributed. While multiple-peer workflows have been extensively studied in the process-centric case using finite-state models, little formal research has been done on collaborative workflows centered around a database, which have infinitely many states (see related work). In this paper, we introduce a simple model for collaborative data-driven workflows and provide techniques that enable a peer to reason about runs of the global workflow based on its local observations.

In our model, peers modify local data using condition/update actions. The connection between the data at different peers is specified using a local-as-view approach, in which the data at each peer is an exact view of a virtual global database. We impose restrictions (using the presence of keys) to guarantee that peer updates can be propagated in an unambiguous manner to other peers. We assume that update propagation is instantaneous, i.e., we assume some underlying synchronization mechanism to support update propagation.

Our goal is to enable peers to reason, based on local observations, about the global state of the system and about actions occurring at other peers. This can serve as the basis for a wealth of runtime tools for monitoring the global run, detecting and diagnosing anomalous behavior, balancing load to improve efficiency, or analyzing the current run to derive competitive advantage over other peers.

Consider a peer p in such a system. We assume p knows the specification of all the other peers. (In fact, p is likely to only be given an *abstraction* of these specifications, hiding details and confidential behavior of the peers.) Peer p only sees a local view of the global run. Note that there are generally infinitely many global runs that are consistent with p's observations. Based on this local view, one would like to evaluate queries over the global run, specified by an extension of first-order logic (FO) with temporal operators (PLTL-FO), referring to the entire history of the run. In particular, we would like to decide whether a formula in this language is *possibly* or *certainly* true in the global runs that correspond to what p sees locally. Deciding such properties is at the heart of the paper.

More precisely, our main contributions are the following:

- developing a finite symbolic representation system for the infinite set of global runs consistent with local observations;

- using the representation system to provide a PSPACE algorithm for evaluating PLTL-FO properties of the global runs consistent with the local observations, with respect to both possible and certain world semantics;
- developing an incremental variant of the algorithm suitable for monitoring some properties specified beforehand; and extending this variant to monitor an infinite class of properties sharing the same temporal structure, so that properties can be chosen in this class while the run unfolds.

Finally, we consider the effect of integrating the reasoning previously described into the control of the workflow itself. This allows a peer to guide its actions based on properties of the global run that can be monitored, detecting some other peer actions that are not visible locally. We show, somewhat surprisingly, that adding such control features does not increase the expressiveness of the workflow specification language. Intuitively, this shows that the workflow specification language is closed under such introspective reasoning.

Related work Although not focused explicitly on workflows, Dedalus [8, 15] and Webdamlog [4, 2] are systems supporting distributed data processing based on condition/action rules. Local-as-view approaches are considered in a number of P2P data management systems, e.g., Piazza [21] that also consider richer mappings to specify views. Update propagation between views is considered in a number of systems, e.g., based on ECA rules in Hyperion [9].

Finite-state workflows with multiple peers have been formalized and extensively studied using communicating finite-state systems (called CFSMs in [1, 10], and *e-compositions* in the context of Web services, as surveyed in [16, 17]). Formal research on infinite-state, data-driven collaborative workflows is still in an early stage. The business artifact model [20] has pioneered data-driven workflows, but formal studies have focused on the single-user scenario. Compositions of data-driven web services are studied in [12], focusing on automatic verification. Active XML [3] provides distributed data-driven workflows manipulating XML data.

A collaborative system for distributed data sharing geared towards life sciences applications is provided by the Orchestra project [14, 19]. The underlying update propagation model among peers is based on schema mappings and is similar to our local-as-view approach. However, Orchestra does not address the kind of analysis problems studied here.

Organization After some preliminaries, we introduce the model of collaborative workflows. We then develop in Section 3 the representation system for the infinite set of global runs consistent with given peer observations. In Section 4, we show how the representation system can be used to evaluate PLTL-FO properties of global runs. We also consider incremental and preemptive evaluation, and discuss the expressiveness of introspection in workflow control. An appendix contains a few detailed definitions and some proofs.

2 The Model

In this section, we introduce the model of collaborative workflows. We begin with some preliminaries, then introduce collaborative workflows.

Preliminaries We assume an infinite data domain **dom** with one distinguished element $\perp$ (representing undefined data values). We also assume an infinite countable domain of variables $\mathbf{var} = \{x_n\}_{n\geq 1}$ disjoint from **dom**. We denote variables by x, y, z, possibly with subscripts. A *relation schema* is a relation symbol together with a sequence of distinct attributes (whose length is the *arity* of the relation). We denote the set of attributes of R by $att(R)$. A *database schema* is a finite set of relation schemas. An *instance* of a database schema is a mapping I associating to each relation schema R a finite relation $I(R)$ over **dom**, of the same arity as R. An instance (or tuple) containing $\perp$ is called *partial*, and otherwise *total*.

We assume that each relation schema R is equipped with a unique key K, consisting of a non-empty subset of its attributes. We say that an instance I over R is *valid* if I satisfies the key and all tuples in I are total on the key attributes.

We recall the notion of *conjunctive query with safe negation* ($\text{CQ}^\neg$ query for short). A *term* is a variable or a constant. A *literal* is of the form $R(\bar{x})$, $\neg R(\bar{x})$, $x = y$, $x \neq y$, where $\bar{x}$ is a sequence of terms of appropriate arity, x is a variable, and y a term. A $\text{CQ}^\neg$ query is an expression $A_1 \wedge ... \wedge A_n$ (for $n \geq 0$) where each A_i is a literal and each variable x occurs in a positive relational literal or in an equality $x = c$ where $c \in \mathbf{dom}$ (i.e., x is *bound*).

Collaborative schema and instance A *collaborative schema* $\mathcal{S}$ consists of:

1. A database schema $\mathcal{D}$, the *global schema*, in which each relation is equipped with a key.
2. A finite set of peer names $\{p_i \mid 1 \leq i \leq m\}$.
3. For each peer p_i, the *local schema* $\mathcal{D}_i$ consisting of a set of relation schemas $R@p_i$, where $R \in \mathcal{D}$, $att(R@p_i) \subseteq att(R)$, and $att(R@p_i)$ contains the key of R.
4. For each $R \in \mathcal{D}$, $att(R) = \cup\{att(R@p_i) \mid R@p_i \in \mathcal{D}_i, 1 \leq i \leq m\}$.

The main motivation for item (4) is to guarantee that the global instance (which is purely virtual) can be computed from the peer instances. Consider for instance some relation R in the global schema. Note that R may be "invisible" from some particular p_i, i.e., $R@p_i$ is not

in $\mathcal{D}_i$. However because of (4) and the key constraints, $I(R)$ can be reconstructed from its projections on the peer schemas.

Let $\mathcal{S}$ be a collaborative schema with global schema $\mathcal{D}$ and peers $\{p_i \mid 1 \leq i \leq m\}$. A *global instance* of $\mathcal{S}$ is a valid instance I over $\mathcal{D}$. The *peer view* of I at p_i, denoted $I@p_i$, is the instance over $\mathcal{D}_i$ defined by: for each $R@p_i \in \mathcal{D}_i$, $I@p_i(R@p_i) = \pi_{att(R@p_i)}(I(R))$. Observe that this introduces a constraint on the instances $I@p_i$: they are projections of the same global instance. Note also that the peer views of an instance I uniquely determine the global instance because of the key constraints and condition (4). More precisely, for each R in $\mathcal{D}$, $I(R) = \bowtie \{I@p_i(R) \mid R@p_i \in \mathcal{D}_i\}$. In particular, this induces a connection between the local instances $\{I@p_i\}_{1 \leq i \leq m}$ that can be stated without reference to the global instance I (which is purely virtual and never materialized):

for each j and $R@p_j \in \mathcal{D}_j$,$I@p_j(R@p_j) =$
$\pi_{att(R@p_j)}(\bowtie \{I@p_i(R@p_i) \mid R@p_i \in \mathcal{D}_i, 1 \leq i \leq m\})$

Remark 2.1 *The views we consider are limited to simple projections. However, more complex views can be provided using actions performed by peers. For instance, consider a selection query σ over a relation R. A peer p_i that sees a relation R can maintain in another relation, say R_σ, the result of $\sigma(R)$. Then any peer p_j that sees R_σ will see the result of that selection even if p_j does not have access to R.*

Example 2.2 We use as a running example a very simplified workflow to process travel expenses in a research institute. The workflow involves the following peers: researchers, e.g., Alice, who can initiate trip requests; a travel agency that provides expense estimates; and admin services that approve or deny trip expenses. The global schema has 3 relations (each with key Id): Submitted(Id, Person, Date, Location), Processing(Id, Person, Expense, Comment, Status) and Web(Id, Person, Date, Conference, Domain). If Domain="inter", the information is published on the Internet. If Domain="intra", it is only published on the Intranet of the institute. The peers' schemas (with the obvious associated view definitions) are the following (as noted in Remark 2.1, selections in view definitions can be simulated and are used for convenience):

Alice (and similarly for all other researchers):
Submitted(Id, "Alice", Date, Location)
Processing(Id, "Alice")
Web(Id, Person, Date, Conference, Domain)

Travel agency schema:
Submitted(Id, Person, Date, Location)
Processing(Id, Person, Expense)
Web(Id, Person, Date, Conference, "inter")

Admin services schema: same as global schema. □

An update to a peer's local data can be propagated to the other peers so that the local instances remain the views of a valid global instance. We assume here that propagation of updates is instantaneous, which can be ensured by the underlying system by a protocol involving asynchronous communication. We do not address this aspect here.

Formally, we define the effect on a global instance I of performing a tuple insertion and deletion at peer p_i. The semantics will guarantee that the resulting global instance remains valid.

Consider the deletion of a tuple t from $I(R@p_i)$. The resulting global instance J is obtained by deleting from $I(R)$ the tuple whose projection on $att(R@p_i)$ equals t, if such exists (note that there is at most one such tuple per relation).

Now consider the insertion of a tuple t in $I(R@p_i)$ (the more interesting case). Let $\bar{t}$ be the tuple over $att(R)$ extending t with $\perp$ for all attributes in $att(R) - att(R@p_i)$. Let J be the result of inserting into I the tuple $\bar{t}$, then chasing with respect to the key K of R. Specifically, the chase consists of the following. For each pair of tuples u, v agreeing on K for which $u(A) = \perp$ and $v(A) \in \mathbf{dom} - \{\perp\}$, replace $u(A)$ by $v(A)$. The insertion is said to be *consistent* if J is valid (the update is rejected otherwise).

We next illustrate the semantics of updates.

Example 2.3 Suppose we have a relation R over $ABCD$ with key A, $R@p1$ is over ABD and $R@p_2$ over ACD. The insertion of $(0,0,0)$ and $(1,1,1)$ in $R@p1$ propagates to the insertion of $(0,\perp,0)$ and $(1,\perp,1)$ in $R@p_2$. Then the deletion of $(0,\perp,0)$ from $R@p_2$ propagates to the deletion of $(0,0,0)$ from $R@p1$. And the insertion of $(1,2,2)$ in $R@p_2$ is refused. A subtlety is that we cannot consistently *modify* attributes of tuples with a given key across peers without losing information. For instance, suppose we wish to modify the D column of the tuple $(1,\perp,1)$ in $R@p_2$ from 1 to 2. This is done by deleting $(1,\perp,1)$ and inserting $(1,\perp,2)$. However, this does not propagate to a modification of D from 1 to 2 in $R@p_1$. Indeed, the previous deletion and insertion first delete $(1,1,1)$ from $R@p_1$, then insert the tuple $(1,\perp,2)$. Thus, the B column was lost as a side effect. It is not hard to extend the model with explicit modifications circumventing this problem. □

Collaborative workflow A *collaborative workflow specification* (in short *workflow spec*) $\mathcal{W}$ consists of a collaborative schema $\mathcal{S}$ and a finite set of actions for each peer p_i of $\mathcal{W}$. An *action* at peer p_i is an expression
Update :- *Condition* where:

- *Condition* is a $\text{CQ}^\neg$ query over $\mathcal{D}_i$.

- *Update* is a non-empty sequence of positive and negative relational literals over $\mathcal{D}_i$ such that each variable occurring in a negative literal also occurs in *Condition*.

Intuitively, positive literals in the update are interpreted as insertions, and negative literals as deletions.

Example 2.4 Continuing Example 2.2, we next show some of the actions of the travel expense processing workflow. For readability, we use attribute names rather than variables, and underline those occurring only in insertions of actions, generating new values. The workflow proceeds as follows.

1. *Alice initiates a new trip request*
   Submitted(Id, "Alice", Date, Location) :-
2. *Alice publishes the trip on the Intranet*
   Web(Id, "Alice", Date, Conference, "intra") :-
   Submitted(Id, "Alice", Date, Location)
3. *Travel agency inserts an estimate of the cost*
   Processing(Id, Person, Expense) :-
   Submitted(Id, Person, Date, Location)
4. *Admin inserts comments*
   Processing(Id, Expense, Comment, $\perp$) :-
   Processing(Id, Expense, Comment, $\perp$)
5. *Admin approves or rejects*
   Processing(Id, Expense, Comment,
   "approve"/"reject") :-
   Processing(Id, Expense, Comment, $\perp$)
6. *Admin deletes rejected trip from the Intranet*
   $\neg$ Web(Id, Person, Date, Conference, "intra") :-
   Web(Id, Person, Date, Conference, "intra"),
   Processing(Id, Expense, Comment, "reject")
7. *Admin publishes approved trip on the Internet*
   Web(Id, Person, Date, Conference, "inter") :-
   Web(Id, Person, Date, Conference, "intra"),
   Processing(Id, Expense, Comment, "approve")

Note that the workflow imposes a number of constraints on the actions of participants. For instance, an admin can modify a comment as many times as wished before a decision is made, but once a trip has been approved or rejected, the comment cannot be modified. Rules (6,7) are internal computations of peer *Admin*: deletion of a rejected trip from the Intranet, and posting of an approved trip on the Internet. We may prefer that these rules be triggered automatically once a decision is made. We could easily extend the model with immediate triggers without affecting the results. Observe the underlined variables in Rules (1-4), not bound in the body. Such unbound values have to be supplied either by the user or by the system; in which case, we will assume the system chooses new values outside the active domain. To simplify the presentation, we will ignore in the paper the differences between user and system actions, and assume that unbound variables are always assigned values outside the active domain. □

Workflow runs Intuitively, the semantics of a workflow spec consists of runs of consecutive global instances. (Clearly, one could also consider trees of runs.) Note that this also determines the runs of the corresponding peer views. Each transition is caused by one application of one instantiation of one action at one peer.

A run starts at an *initial global instance* of $\mathcal{W}$, i.e. a valid instance over $\mathcal{D}$. In practice, one may wish to impose some conditions on initial global instances. For instance, it may make sense to require that some relations be initially total, or initially empty (for relations recording tasks to be performed). To simplify, we ignore here this aspect, which does not affect the results.

The transition relation $\vdash$ is defined using the auxiliary notion of *instantiation* of an action at peer p_i for a global instance I. We use the notion of *active domain*. First, the active domain of $\mathcal{W}$, denoted $adom(\mathcal{W})$, consists of the constants used in $\mathcal{W}$, and $\perp$. The active domain of an instance I, denoted $adom(I)$, is the set of constants occurring in I together with $adom(\mathcal{W})$. Let $\alpha = Update(\bar{x}, \bar{y})$:- $Condition(\bar{x})$ be an action at p_i where $\bar{x}$ are the variables occurring in *Condition* and $\bar{y}$ are the variables in *Update* other than $\bar{x}$. Let ν be a valuation of $\bar{x}$ in **dom** such that $I@p_i \models Condition(\nu(\bar{x}))$. Let $\bar{\nu}$ be an extension of ν mapping variables in $\bar{y}$ to distinct values in **dom** outside the active domain of I. Then $\bar{\nu}\alpha$ is an instantiation of this action at peer p_i for the global instance I.

For two global instances I and J over $\mathcal{D}$, $I \vdash_e J$ if the following holds:

> (†) There is a peer p_i, an instantiation $\bar{\nu}\alpha$ of an action at peer p_i for I such that J is obtained from I by applying the sequence of insertions and deletions in $Update(\bar{\nu}(\bar{x}, \bar{y}))$, in the specified order, and all insertions are consistent.

The label e, referred to as the *event* causing the transition, consists of the triple $(peer(e), action(e), val(e))$ where $peer(e) = p_i$, $action(e) = \alpha$ and $val(e) = \bar{\nu}$. We denote by a special symbol *init* the vacuous event creating the initial instance in a run, needed for technical reasons. From the definition, it follows that if I is valid and $I \vdash_e J$, then J is valid.

Note a subtlety in the active domain semantics we use. In the definition, the active domain refers to the current snapshot I. However, in some applications, it is desirable for new values to be outside the active domain of the *entire run* leading to I. For instance, new values may represent task IDs, and we may wish for them to be

unique in each run. Such a semantics can be easily simulated with the one adopted here, simply by keeping in a designated relation the values that may not be reused.

We next definite runs of workflow specs.

A *run* of $\mathcal{W}$ is a finite sequence $\{(I_i, e_i)\}_{0 \leq i \leq n}$, such that:

- $e_0 = init$ and I_0 is a valid instance over $\mathcal{D}$,
- for each $0 < i \leq n$, $I_{i-1} \vdash_{e_i} I_i$

Note that the sequence $\{I_i\}_{0 \leq i \leq n}$ of instances in a run does *not* generally determine the events causing each transition. However, if desired, the actions of $\mathcal{W}$ can be modified so that events are explicitly recorded in designated relations. When this is the case, the sequence of instances is sufficient to uniquely identify the events.

Remark 2.5 *Although left implicit, it is easy to see that our collaborative workflows provide an expressive model that can simulate the execution of sets of tasks and can capture hierarchical tasks of arbitrary depth, making use of keys and invented values. In particular, our collaborative workflows subsume the popular business artifact model [20]. This can be formalized using the framework developed in [5] for comparing the expressiveness of workflow languages.*

3 Symbolic representation of runs

We next develop a symbolic representation for the set of global runs consistent with given local observations at a peer. This will be used in the next section to carry out reasoning about the global runs, given such local observations. As we will see, it will be necessary to impose some simple restrictions on workflow specifications in order to render such reasoning feasible.

Consider a global run of a workflow spec $\mathcal{W}$. Let p be a peer of $\mathcal{W}$. The information about the run as observed by p is captured by the notion of *p-trace*, defined next. Intuitively, a p-trace retains only transitions caused by actions of p, or by actions of other peers that have visible side effects at p. In this latter case, p does not know which action actually took place. We use the symbol $\star$ to denote such an unknown action. Also, some transitions are completely invisible to p, so do not participate to the p-trace. Formally:

Definition 3.1 *Let $\rho = \{(I_i, e_i)\}_{0 \leq i \leq n}$ be a run of some workflow spec $\mathcal{W}$, and p be a peer of $\mathcal{W}$. Let $\rho@p = \{(I_i@p, f_i)\}_{0 \leq i \leq n}$ where $f_i = e_i$ if* peer$(e_i) = p$ *and $f_i = \star$ otherwise, where $\star$ is a new symbol. The* p-trace *of ρ, denoted $\nu_p(\rho)$, is the sequence obtained from $\rho@p$ by recursively deleting all $(I_j@p, f_j)$ such that $I_j@p = I_{j-1}@p$ and $f_j = \star$.*

Suppose that p observes a p-trace τ in the course of the run of $\mathcal{W}$. We would like to describe and reason about the set of all runs ρ of $\mathcal{W}$ that are consistent with τ, i.e., such that $\nu_p(\rho) = \tau$. We denote this set by $\nu_p^{-1}(\tau)$. Note that, because of silent transitions, the set $\nu_p^{-1}(\tau)$ may contain runs of unbounded length and is generally infinite. Unfortunately, even basic properties of such runs are generally undecidable. To illustrate, we mention a few such properties.

Theorem 3.2 *The following are undecidable, for a workflow spec $\mathcal{W}$ and a p-trace τ:*

current *Is it possible/certain that the current local instance at some peer satisfies some first-order (FO) property φ?*

past *Is it possible/certain that some local instance at a peer satisfied some FO property φ during the run?*

event *Is it possible/certain that some peer q performed some particular action α during the run?*

validation *Is a sequence $\{(I_i@p, f_i)\}_{0 \leq i \leq n}$ that is syntactically a p-trace an actual p-trace of a global run?*

The proofs are by reduction from the undecidability of FO satisfiability (see [6]), using the fact that workflow computations can produce the answer to an FO query.

The above undecidability results are not surprising. A main contribution of the paper is to demonstrate decidability of a wide range of properties (including the previous ones) for a large class of workflow specs. The restriction we impose, called *key visibility*, is often reasonable in practice and is an acceptable price to pay for the ability to perform useful reasoning tasks. Key visibility requires that peer p sees at least *some* projection view of *each* global relation (which by definition includes its key). Formally (with $\mathcal{D}_p$ denoting the schema of peer p):

Definition 3.3 *A workflow spec $\mathcal{W}$ with schema $\mathcal{D}$ is* key-visible *at p if $R@p \in \mathcal{D}_p$ for each relation $R \in \mathcal{D}$.*

For instance, the workflow of Example 2.2 is key-visible at all peers. While key visibility is a strong restriction for arbitrary specifications, it is reasonable in the likely event that the specification available to p is an *abstraction* of the actual specification, provided to p as a surrogate (or explanation) for it. In actual specifications, peers q will generally use relations not revealed to p, that determine their precise behavior. The abstraction available to p can be expected to provide an approximation of the actual behavior of other peers on relations they share, in some sense a contract between p and such peers. This enables reasoning by p while ignoring the full details of other peers' specification.

Even for a workflow that is key-visible at p, the set of global runs consistent with a given p-trace is infinite. However, we are able to provide a symbolic representation for runs of key-visible workflows given a trace. We do this next. The representation is based on a variant of the classic conditional tables, a formalism introduced

to capture incomplete information [18]. Intuitively, we capture a set of possible global instances of the system using a table. We then consider "transitions" between such tables to represent possible moves. So the set of global runs consistent with a p-trace can be described by a transition system over a set of tables.

Incomplete instances We use the following auxiliary notions. An *atomic constraint* is an expression $x = (\neq)\ t$ where $x \in \mathbf{var}$ and $t \in \mathbf{var} \cup \mathbf{dom}$. An atomic constraint is *trivial* if it is $x = x$ for some $x \in \mathbf{var}$. A *constraint* is a Boolean combination of atomic constraints and a *conjunctive constraint* is a conjunction of atomic constraints, with no repetition of the same atom. As a shorthand, if $\bar{x}$ and $\bar{y}$ are tuples of the same arity, we denote by $\bar{x} = \bar{y}$ the conjunction of the componentwise equalities, and by $\bar{x} \neq \bar{y}$ the disjunction of the componentwise inequalities. The *closure* φ^*_V of a conjunctive constraint φ on a subset V of it variables is the conjunction of all non-trivial atomic constraints implied by φ, whose variables are in V. If V consists of all variables in φ, we simply write φ^* instead of φ^*_V.

We can now define the notion of *incomplete instance*, *I-instance* for short. Intuitively, it includes some unknown values (not to be confused with the $\perp$ values) denoted by variables, and a global constraint on these variables.

An *I-instance* over $\mathcal{D}$ is a pair $(\mathcal{I}, \varphi)$, where:

- $\mathcal{I}$ is a mapping associating to each $R \in \mathcal{D}$ a finite relation over R using values in $\mathbf{dom} \cup \mathbf{var}$.
- φ is a satisfiable conjunctive constraint using variables in $\mathcal{I}$ and a finite set of constants.
- $\varphi \models \varphi_{key}$ where φ_{key} is a constraint stating that no distinct tuples in $\mathcal{I}(R)$ agree on the key attributes of R, for every $R \in \mathcal{D}$.

An I-instance represents a set of possible instances as follows. For an I-instance $(\mathcal{I}, \varphi)$, we denote by $var(\mathcal{I})$ the set of variables occurring in tuples of $\mathcal{I}$. Given an I-instance $(\mathcal{I}, \varphi)$ over $\mathcal{D}$, the set of instances over $\mathcal{D}$ represented by $(\mathcal{I}, \varphi)$ is

$$rep(\mathcal{I}, \varphi) = \{v(\mathcal{I}) \mid v \text{ is a valuation of } var(\mathcal{I}) \text{ into } \mathbf{dom} \text{ satisfying } \varphi\}$$

It is clear that, by definition, every $I \in rep(\mathcal{I}, \varphi)$ is a valid instance. Note also that (because of the completeness of the keys) the number of rows in $\mathcal{I}(R)$ is the same as the number of rows in $I(R)$ for each $I \in rep(\mathcal{I}, \varphi)$ and $R \in \mathcal{D}$.

Symbolic transitions As noted earlier, given a p-trace, there are infinitely many corresponding runs, which renders the analysis nontrivial. However, we will see that we can represent such runs by "symbolic runs", essentially by considering I-instances and abstract actions on such I-instances. Intuitively, when applying an abstract action to an I-instance, we obtain another I-instance by applying symbolically the peer action to the original I-instance. Such a transition from one I-instance to another generates additional constraints on the original I-instance, akin to preconditions, and transitions are labeled by these constraints. We next describe these transitions.

Intuitively, a symbolic transition (S-transition) $(\mathcal{I}, \varphi) \vdash_{f,\gamma} (\mathcal{J}, \psi)$ captures how an action f updates instances in $rep(\mathcal{I}, \varphi)$ to instances in $rep(\mathcal{J}, \psi)$ assuming that the *transition constraint* γ (to be defined) is satisfied. We will define S-transitions and prove that they provide a complete representation for actual transitions (Lemma 3.4).

We describe symbolic transitions informally. It will be useful to consider a normal form for actions $Update(\bar{x}, \bar{y})$:- $Condition(\bar{x})$. The normal form requires that each variable occurs at most once in the relational atoms of the rule. It is easy to see that all specifications can be rewritten in normal form by introducing additional variables and equalities between variables resulting from repeated occurrences. In the following, we assume the actions are all in normal form.

Consider an I-instance $(\mathcal{I}, \varphi)$. Let q be a peer. We define the local I-instance at peer q by $(\mathcal{I}, \varphi)@q = (\mathcal{I}@q, \varphi@q)$ where $\mathcal{I}@q$ is the projection view of $\mathcal{I}$ at peer q, and $\varphi@q$ is the closure of φ on the variables in $\mathcal{I}@q$.

Consider an action $Update(\bar{x}, \bar{y})$:- $Condition(\bar{x})$ at peer q (assumed to be in normal form). Intuitively, the action is applied to a local I-instance in two stages: first find a valuation v of $\bar{x}$ into the I-instance. The valuation transfers to $v(\bar{x})$ the constraints from $Condition(\bar{x})$, and imposes "new value" constraints on $\bar{y}$. These become part of the transition constraints. Next, the updates in $Update(v(\bar{x}), v(\bar{y}))$ are applied for the valuation v. When a tuple is inserted, this may yield several transitions, depending on agreement with already existing tuples on the key. In each case, the resulting I-instance is obtained by chasing with the key. When a tuple is deleted, the result depends once again on the possible equalities of the deleted tuple with existing tuples in the instance. Each such equality is captured by a constraint and generates a separate transition. If the final transition constraint is γ, the resulting I-instance is $(\mathcal{J}, \psi)$ where $\mathcal{J}$ is obtained by applying a sequence of updates to $\mathcal{I}$ corresponding to $Update(v(\bar{x}), v(\bar{y}))$, and ψ is the closure of $\varphi \wedge \gamma$ on the variables of $\mathcal{J}$. The transition constraint γ involves variables from both $\mathcal{I}$ and $\mathcal{J}$, so cannot be absorbed into the static I-instance constraints. The detailed construction of S-transitions is presented in the appendix.

If $(\mathcal{J}, \psi)$ is obtained from $(\mathcal{I}, \varphi)$ by an S-transition with transition constraint γ, action α at peer q and valuation v, we say that $e = (q, \alpha, v)$ is the *event* of the transition. If furthermore $\psi \wedge \gamma$ is satisfiable, we write $(\mathcal{I}, \varphi) \vdash_{e,\gamma} (\mathcal{J}, \psi)$. It is easy to see that, by construction, $\psi \models \varphi_{key}$, so $(\mathcal{J}, \psi)$ is an I-instance. This defines the S-transition

relation among I-instances over $\mathcal{D}$.

Similarly to I-instances, the purpose of S-transitions is to represent a set of actual transitions among global instances. Let $(\mathcal{I},\varphi) \vdash_{e,\gamma} (\mathcal{J},\psi)$ be an S-transition, where e is the event (q,α,v). The set of transitions represented by the above S-transition is

$$rep((\mathcal{I},\varphi) \vdash_{e,\gamma} (\mathcal{J},\psi)) = \{(\nu(\mathcal{I}) \vdash_{\nu(e)} \nu(\mathcal{J})) \mid \nu \text{ is a valuation of the variables in } \mathcal{I}\cup\mathcal{J} \text{ into } \mathbf{dom} \text{ satisfying } \varphi\wedge\gamma\wedge\psi \text{ and } \nu(e) = (q,\alpha,v\circ\nu)\}$$

The following key lemma says that, starting from some I-instance, the S-transitions capture all possible actual transitions from instances represented by the I-instance. Thus, S-transitions are a complete representation of actual transitions.

Lemma 3.4 *For each I-instance $(\mathcal{I},\varphi)$,*

$$\begin{array}{ll}\{(I \vdash_e J) \mid & I \in rep(\mathcal{I},\varphi), e \text{ is an event}\} = \\ \{(I \vdash_e J) \mid & \text{there exists } (\mathcal{I},\varphi) \vdash_{f,\gamma} (\mathcal{J},\psi) \text{ such that} \\ & (I \vdash_e J) \in rep((\mathcal{I},\varphi) \vdash_{f,\gamma} (\mathcal{J},\psi))\}\end{array}$$

Lemma 3.4 follows from the construction of S-transitions. The fact that I-instances satisfy φ_{key} is critical, because it guarantees that no distinct tuples in $(\mathcal{I},\varphi)$ may represent the same tuple in some $I \in rep(\mathcal{I},\varphi)$. The construction would not be correct otherwise.

Symbolic runs We now turn to the notion of symbolic run, and to the connection between symbolic runs and actual runs. A *symbolic run* (S-run) of $\mathcal{W}$ is a sequence $\{((\mathcal{I}_i,\varphi_i),(e_i,\gamma_i))\}_{0\le i\le n}$ such that

- $e_0 = \mathit{init}$ and $\gamma_0 = \mathit{true}$
- for each $0 < i \le n$, $(\mathcal{I}_{i-1},\varphi_{i-1}) \vdash_{e_i,\gamma_i} (\mathcal{I}_i,\varphi_i)$

Thus, an S-run is a finite sequence of consecutive symbolic transitions. Let $\mathbf{s}$ be an S-run $\{((\mathcal{I}_i,\varphi_i),(e_i,\gamma_i))\}_{0\le i\le n}$. The set of actual runs represented by $\mathbf{s}$, denoted $rep(\mathbf{s})$, consists of all runs $\{(I_i,g_i)\}_{0\le i\le n}$ for which $(I_{i-1} \vdash_{g_i} I_i) \in rep((\mathcal{I}_{i-1},\varphi_{i-1}) \vdash_{e_i,\gamma_i} (\mathcal{I}_i,\varphi_i))$ for all $0 < i \le n$.

As a consequence of Lemma 3.4, S-runs provide a complete representation of actual runs.

Symbolic runs constrained by traces Next, consider a p-trace τ. We wish to use S-runs to represent *precisely* the global runs in $\nu^{-1}(\tau)$. To this end, we need to constrain symbolic runs by p's observations as given by τ. Since all relations in $\mathcal{D}$ are key-visible at p, we need to only consider I-instances that are fully instantiated on the attributes visible at p (which include all key attributes). Therefore, we need to compute specializations of transitions limited to such instances.

Let I_p be an instance over $\mathcal{D}_p$ (at peer p). We say that an I-instance $(\mathcal{I},\varphi)$ is I_p-instantiated if $I@p = I_p$ for every $I \in rep(\mathcal{I},\varphi)$. Now consider an I_p-instantiated I-instance $(\mathcal{I},\varphi)$ and let J_p be another instance of $\mathcal{D}_p$ (which may equal I_p, as allowed in a p-trace). We wish to find representations of transitions from $(\mathcal{I},\varphi)$ constrained to produce J_p-instantiated instances. Such constrained transition define a new relation among I-instances, that we call J_p-constrained transition relation, denoted $\vdash^{J_p}$. The relation $\vdash^{J_p}$ is obtained by specializing the unrestricted transition relation $\vdash$ as follows. Consider an I-transition $(\mathcal{I},\varphi) \vdash_{e,\gamma} (\mathcal{J},\psi)$. Let $\mathcal{V}$ be the set of valuations ν mapping variables in $\mathcal{J}@p$ into values in J_p such that ν satisfies ψ and $\nu(\mathcal{J}@p) = J_p$. Let θ_ν be the constraint consisting of the conjunction of all equalities $x = \nu(x)$. The set of J_p-constrained transitions generated by $(\mathcal{I},\varphi) \vdash_{e,\gamma} (\mathcal{J},\psi)$ consists of all expressions $(\mathcal{I},\varphi) \vdash^{J_p}_{e,(\gamma\wedge\theta_\nu)} (\mathcal{J},\psi\wedge\theta_\nu)$ for $\nu \in \mathcal{V}$. The semantics of J_p-constrained transitions is the same as for unconstrained transitions. More precisely, $rep((\mathcal{I},\varphi) \vdash^{J_p}_{e,\pi} (\mathcal{J},\xi)) = \{\nu(\mathcal{I}) \vdash_{\nu(e)} \nu(\mathcal{J}) \mid \nu \text{ is a valuation from the variables of } \mathcal{I}\cup\mathcal{J} \text{ into } \mathbf{dom} \text{ satisfying } \varphi\wedge\pi\wedge\xi\}$.

The next result follows easily by construction.

Lemma 3.5 *There is a* PTIME *nondeterministic algorithm that, given $(\mathcal{I},\varphi)$ and J_p, outputs each J_p-constrained transition from $(\mathcal{I},\varphi)$.*

Next, consider a p-trace $\tau = \{(P_i,f_i)\}_{0\le i\le k}$. Let us first ignore the order of the local instances and the operations f_i. So, let $\mathcal{P}_\tau = \{P_i \mid 0 \le i \le k\}$. Recall that each instance in $\nu^{-1}(\tau)$ is P_j-instantiated for some $P_j \in \mathcal{P}_\tau$. We are therefore interested in runs in which each transition is P_j-constrained for some $P_j \in \mathcal{P}_\tau$. We call such runs $\mathcal{P}_\tau$-constrained.

Definition 3.6 *A $\mathcal{P}_\tau$-constrained run is a finite sequence $\{((\mathcal{I}_i,\varphi_i),(e_i,\gamma_i,P_{j_i}))\}_{0\le i\le n}$ such that*

(i) $e_0 = \text{init}$, $\tau_0 = \text{true}$, $\mathcal{J}_0@p = P_{j_0}$, *and* $var(\mathcal{J}_0) = \{x_1,\ldots,x_m\}$ *for some* $m \ge 0$.
(ii) for each $0 < i \le n$, $(\mathcal{I}_{i-1},\varphi_{i-1}) \vdash^{P_{j_i}}_{e_i,\gamma_i} (\mathcal{I}_i,\varphi_i)$

Note that, in the initial instance of a $\mathcal{P}_\tau$-constrained run, there are no variables occurring in the attributes visible at p. Moreover, the variables occurring in $\mathcal{J}_0$ are picked among those of smallest index. This is a harmless assumption useful for technical reasons. In particular, we can show the following (see Appendix for proof).

Lemma 3.7 *For each finite $\mathcal{P}_\tau$, the set of I-instances reachable by $\mathcal{P}_\tau$-constrained runs is finite.*

We are close to our goal. The $\mathcal{P}_\tau$-constrained runs we defined produce p-traces using only instances in the p-trace $\tau = \{(P_i,f_i)\}_{0\le i\le k}$, but not necessarily in the correct order nor with proper f_i. In order to define precisely $\nu^{-1}(\tau)$ we need to further constrain the runs. We do this using a nondeterministic finite-state automaton A_τ defined as follows:

the set of states of A_τ is $\{p_0\} \cup \{q_i \mid 0 \leq i \leq k\}$, with initial state p_0 and final state q_k.

the alphabet consists of the finite set of all $((\mathcal{I}, \varphi), (e, \gamma, P))$ occurring in $\mathcal{P}_\tau$-constrained runs of $\mathcal{W}$.

the transition mapping δ is defined as follows:

- **start** $\delta(p_0, ((\mathcal{I}, \varphi), (e, \gamma, P))) = q_0$ if $((\mathcal{I}, \varphi), (e, \gamma, P))$ is the initial instance of a $\mathcal{P}_\tau$-constrained run and $P = P_0$,
- **visible** for $0 \leq i < k$, $\delta(q_i, ((\mathcal{I}, \varphi), (e, \gamma, P))) = q_{i+1}$ if $P = P_{i+1}$, and (nonlocal) $f_{i+1} = \star$ and $peer(e) \neq p$, or (local) $f_{i+1} = p$ and $peer(e) = p$.
- **silent** for $0 \leq i \leq k$, $\delta(q_i, ((\mathcal{I}, \varphi), (e, \gamma, P))) = q_i$ if $P = P_i$ and $peer(e) \neq p$,

Let $A_\tau(\mathcal{P}_\tau)$ denote the set of $\mathcal{P}_\tau$-constrained runs accepted by A_τ. We have the following (see Appendix for proof).

Theorem 3.8 *Let τ be a p-trace for a peer p of $\mathcal{W}$. Then $\nu^{-1}(\tau) = \cup\{rep(\mathbf{s}) \mid \mathbf{s} \in A_\tau(\mathcal{P}_\tau)\}$.*

Thus, A_τ together with our transition system on $\mathcal{P}_\tau$-constrained instances provide a finite representation of the infinite set of runs in $\nu^{-1}(\tau)$.

Remark 3.9 *It is easy to see that the size of $A_\tau(\mathcal{P}_\tau)$ is exponential in τ. However, the evaluation algorithm of the next section never materializes the full $A_\tau(\mathcal{P}_\tau)$. Instead, the S-runs in $A_\tau(\mathcal{P}_\tau)$ are explored lazily, one transition at a time. As we shall see, this yields an algorithm of complexity* PSPACE *in τ.*

4 Peer reasoning

We next formalize the properties of global runs that we focus on, and show how they can be evaluated using the representation system developed in the previous section.

Temporal properties of runs Recall that we are interested in verifying and monitoring properties of global runs based on local observations at a given peer. We specify the properties of interest in an extension of Past Linear-Time Temporal Logic (PLTL). The language, denoted PLTL-FO, is obtained from propositional PLTL with past operators (e.g., see [13]) by interpreting each proposition as an FO formula.

We first recall the language PLTL that is obtained by augmenting propositional logic with: past temporal operators Z (initially), X^{-1} (previously), S (since) and G^{-1} (always previously) as follows. If ϕ and ϕ' are formulas, then so are $Z\phi$, $X^{-1}\phi$, $\phi\ S\ \phi'$ and $G^{-1}\phi$. A *PLTL formula* is evaluated on finite sequences $\sigma_0 \ldots \sigma_n$ of truth assignments to its propositions. The semantics is defined as follows (we omit the standard definition of $\wedge$ and $\neg$).

- $\sigma_0 \ldots \sigma_n \models r$ for a proposition r if $\sigma_n(r) = 1$.
- $\sigma_0 \ldots \sigma_n \models Z\phi$ if $n = 0$ and $\sigma_0 \models \phi$.
- $\sigma_0 \ldots \sigma_n \models X^{-1}\phi$ iff $n > 0$ and $\sigma_0 \ldots \sigma_{n-1} \models \phi$.
- $\sigma_0 \ldots \sigma_n \models \phi\ S\ \phi'$ iff $\sigma_0 \ldots \sigma_j \models \phi'$ for some $j \leq n$ and $\sigma_0 \ldots \sigma_k \models \phi$ for every h, $j < h \leq n$.
- $\sigma_0 \ldots \sigma_n \models G^{-1}\phi$ iff $\sigma_0 \ldots \sigma_j \models \phi$ for each $j \in [0, n]$.

Consider a PLTL formula ϕ, the set P of propositions occurring in ϕ and the set of sequences of truth assignments over P satisfying ϕ. It is straightforward to construct a finite-state alternating automaton with alphabet 2^P that accepts precisely this set of sequences, with a number of states linear in ϕ. This alternating automaton can then be converted to a nondeterministic automaton A_ϕ with a number of states exponential in ϕ. Moreover, there is a nondeterministic PSPACE algorithm (w.r.t. ϕ) that, given a state q of A_ϕ and a truth assignment σ, outputs the successors of q under input σ (see [22, 11]).

We next define the extension PLTL-FO. A *PLTL-FO formula* over $\mathcal{W}$ is an expression $\phi_f = (\phi, f)$ where ϕ is a propositional PLTL formula and f maps each proposition r of ϕ to an FO formula $f(r)$. Each FO formula $f(r)$ is called an FO *component* of ϕ_f. FO components are formulas over the global schema $\mathcal{D}$, extended as follows: for each action $\alpha = Update(\bar{x}, \bar{y})$:-$Condition(\bar{x})$ at peer q, we add to $\mathcal{D}$ an action-relation α_q of arity $|\bar{x}| + |\bar{y}|$ (with the semantics that $\alpha_q(\bar{a}, \bar{b})$ holds at some step if the corresponding action is taken with valuation $\nu(\bar{x}) = \bar{a}$ and $\nu(\bar{y}) = \bar{b}$).

In addition, FO components may use constants in $adom(\mathcal{W})$. (It is always possible, if desired, to introduce any fixed set of constants considered significant in the active domain).

In a run $\{(I_i, e_i)\}_{0 \leq i \leq n}$, an FO component $f(r)$ with no free variables holds in (I_i, e_i), denoted $(I_i, e_i) \models f(r)$, if $f(r)$ is true in the structure I_i extended to the action relations as above.

The semantics of ϕ_f is defined as follows. Consider a run $\rho = \{(I_i, e_i)\}_{0 \leq i \leq n}$ of $\mathcal{W}$. For each i, let σ_i be the truth assignment to propositions in ϕ defined by $\sigma_i(r) = 1$ iff $(I_i, e_i) \models f(r)$. The run ρ satisfies ϕ_f iff $\sigma_0 \ldots \sigma_n \models \phi$. Clearly, checking that $\rho \models \phi_f$ can be done in PSPACE by nondeterministically running the automaton A_ϕ on the sequence of truth assignments $\sigma_0 \ldots \sigma_n$ computed on ρ.

In the presence of incomplete information on runs, we are interested in giving possible and certain world semantics to PLTL-FO formulas. Let $\phi_f(\bar{x})$ be a PLTL-FO formula and $\mathcal{R}$ a set of runs of $\mathcal{W}$. We say that $poss(\phi_f(\bar{x}))$ holds in $\mathcal{R}$ if there exists a run $\rho \in \mathcal{R}$ and there exists a valuation ν for $\bar{x}$ in the active domain of ρ, such that ρ satisfies $\phi_f(\nu(\bar{x}))$. Likewise, $cert(\phi_f(\bar{x}))$ holds in $\mathcal{R}$ if $\phi_f(\nu(\bar{x})$ holds for each run $\rho \in \mathcal{R}$ and each valuation ν of $\bar{x}$ into the active domain of ρ. Thus, the free variables are quantified existentially in possible world semantics and universally in certain world semantics.

Example 4.1 Consider the rules in Example 2.4. Suppose that a researcher, say Bob, would like to know if Alice's trip Id455 has been rejected. Bob does not have direct access to this information. However, he does see the trips that are inserted and deleted from the Intranet and Internet. Based on these local observations, he can *infer*, once Alice's trip is posted on the Internet, that the trip has been approved; and, if the trip is first posted on the Intranet and then deleted, Bob can infer that it has been rejected. On the other hand, if the trip is posted on the Intranet but not (yet) deleted, the trip may or may not have been rejected. Clearly, the acceptance/rejection of Alice's trip can be expressed in PLTL-FO (with certain or possible semantics). We will see next how such properties can be evaluated using the local observations. □

Evaluating PLTL-FO properties Given a p-trace τ, we are interested in evaluating $poss(\phi_f(\bar{x}))$ and $cert(\phi_f(\bar{x}))$ on the set of global runs of $\mathcal{W}$ compatible with τ, that is, $\nu^{-1}(\tau)$. We now show how this can be done using the framework developed earlier. To simplify the presentation, we assume without loss of generality that FO components of PLTL-FO formulas are over the schema $\mathcal{D}$, without the extension to action relations defined above. (Intuitively, one can simulate the reasoning in the extended global schema by considering a schema with additional "normal" relations carrying the extra information.)

We next show how to use this to evaluate and monitor temporal properties of runs in $\nu^{-1}(\tau)$.

Let us fix a PLTL-FO property ϕ_f we wish to evaluate under possible and certain semantics on $\nu^{-1}(\tau)$. Suppose for the moment that ϕ_f has no free variables. In order to evaluate FO components of ϕ_f we will use I-instances in which the equality type of all variables and constants is completely specified. More precisely, let $(\mathcal{I}, \varphi)$ be an I-instance. We call $(\mathcal{I}, \varphi)$ *complete* if for each $x \in var(\mathcal{I})$ and $t \in var(\mathcal{I}) \cup adom(\mathcal{I}, \varphi)$, $\varphi \models x = t$ or $\varphi \models x \neq t$. A $\mathcal{P}_\tau$-constrained run is *complete* if each of its I-instances is complete.

Observe the following (see Appendix).

Lemma 4.2 *(i) Let $(\mathcal{I}, \varphi)$ be a complete I-instance and $f(r)$ an FO component of ϕ_f. Then $f(r)$ has the same truth value in every $I \in \text{rep}(\mathcal{I}, \varphi)$. (ii) If $\{((\mathcal{I}_i, \varphi_i), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ is a $\mathcal{P}_\tau$-constrained run and $(\mathcal{I}_0, \varphi_0)$ is a complete I-instance, then $(\mathcal{I}_i, \varphi_i)$ is a complete I-instance for every $i > 0$.*

Because of (i), complete runs are convenient in order to evaluate ϕ_f, because the truth value of each FO component is well defined on each I-instance of the run. More precisely, given a complete $\mathcal{P}_\tau$-constrained run $\mathbf{s} = \{((\mathcal{I}_i, \varphi_i), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$, the truth value of an FO component $f(r)$ at $(\mathcal{I}_i, \varphi_i)$ can be defined as its truth value on *any* instance $I_i \in rep(\mathcal{I}_i, \varphi_i)$, and can clearly be computed in PSPACE.

We are now ready to show the following main result.

Theorem 4.3 *Let $\mathcal{W}$ be a workflow spec, p a peer of $\mathcal{W}$, τ a p-trace of $\mathcal{W}$ and $\phi_f(\bar{x})$ a PLTL-FO property over $\mathcal{W}$. Then* $\text{poss}(\phi_f(\bar{x}))$ *and* $\text{cert}(\phi_f(\bar{x}))$ *can be evaluated in* PSPACE *with respect to ϕ_f and τ.*

Proof: Since $cert(\phi_f(\bar{x}))$ is equivalent to $\neg poss(\neg\phi_f(\bar{x}))$, it is enough to consider the possible world semantics. We outline a nondeterministic algorithm for evaluating $poss(\phi_f(\bar{x}))$ given a p-trace τ, of complexity PSPACE wrt ϕ_f and τ. Consider first the case when ϕ_f has no free variables $\bar{x}$. We need to check whether there exists a run $\rho \in \nu^{-1}(\tau)$ such that $\rho \models \phi_f$. The algorithm consists of nondeterministically generating a complete $\mathcal{P}_\tau$-constrained run together with computations of $A_\tau(\mathcal{P}_\tau)$ and A_ϕ on the run. The algorithm outputs YES if both automata accept. To make sure the $\mathcal{P}_\tau$-constrained run is complete, it is enough, as noted in Lemma 4.2, that its initial I-instance be complete. Note that the size of each generated I-instance in the run is polynomial in the number of constants occurring in previous I-instances in the run or in $\mathcal{W}$. By Lemma 3.5, the $\mathcal{P}_\tau$-constrained transitions from an I-instance $(\mathcal{I}, \varphi)$ can be computed nondeterministically in PTIME wrt $(\mathcal{I}, \varphi)$ and $\mathcal{P}_\tau$. Also recall that each transition of A_ϕ can be computed nondeterministically in PSPACE wrt ϕ, and each transition of $A_\tau(\mathcal{P}_\tau)$ can clearly be computed in PTIME with respect to τ. Thus, the algorithm has complexity PSPACE wrt ϕ_f and τ, for fixed $\mathcal{W}$. If $\mathcal{W}$ is not fixed, then the algorithm is EXPSPACE (with the maximum arity of relations in $\mathcal{D}$ in the exponent). The correctness of the algorithm follows from Theorem 3.8 and Lemma 4.2.

Now consider the case when ϕ_f has free variables $\bar{x}$. We need to check whether there exists a run $\rho \in \nu^{-1}(\tau)$ *and* a valuation v of $\bar{x}$ into the active domain of ρ such that $\rho \models \phi_f(v(\bar{x}))$. To verify this, we augment the previous algorithm generating a complete $\mathcal{P}_\tau$-constrained run accepted by A_ϕ and $A_\tau(\mathcal{P}_\tau)$ by *guessing* a consistent connection between the variables in $\bar{x}$ and the variables or constants in the I-instances in the run, and evaluating the FO components of $\phi_f(\bar{x})$ according to that guess. More precisely, this is done as follows. Let $\mathbf{s} = \{((\mathcal{I}_i, \varphi_i), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ be a complete $\mathcal{P}_\tau$-constrained run generated as in the earlier algorithm. As the run is generated, an additional conjunctive constraint $\psi_i(\bar{x})$ over $\bar{x}$ is computed nondeterministically for every i. The formula $\psi_i(\bar{x})$ is of the form $\beta_i(\bar{x}) \wedge \gamma_i(\bar{x})$. Intuitively, $\beta_i(\bar{x})$ guesses the connection of $\bar{x}$ with variables and constants in the current I-instance, and $\gamma_i(\bar{x})$ consists of the constraints on $\bar{x}$ inherited from previous guesses. Specifically, $\psi_i(\bar{x}) = \beta_i(\bar{x}) \wedge \gamma_i(\bar{x})$ is defined inductively as follows. For $i = 0$, $\beta_0(\bar{x})$ consists, for each $z \in \bar{x}$, of an equality $z = t$ for some $t \in var(\mathcal{I}_0) \cup adom(\mathcal{I}_0, \varphi_0)$,

or the conjunction of all inequalities $z \neq t$ for all such t. The constraint $\gamma_0(\bar{x}) = \mathit{true}$. For $i > 0$, $\gamma_i(\bar{x}) = (\varphi_{i-1} \wedge \psi_{i-1}(\bar{x}))^*_{\bar{x}}$ and $\beta_i(\bar{x})$ consists, as for the base case, of a nondeterministically chosen conjunction consisting, for each $z \in \bar{x}$, of an equality $z = t$ for some $t \in \mathit{var}(\mathcal{I}_i) \cup \mathit{adom}(\mathcal{I}_i, \varphi_i)$, or the conjunction of all inequalities $z \neq t$ for all such t, such that $\varphi_i \wedge \psi_i(\bar{x})$ is satisfiable.

We can show the following:

(†) Let $\mathbf{s} = \{((\mathcal{I}_i, \varphi_i), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ be a $\mathcal{P}_\tau$-constrained run. There exists a sequence $\{\psi_i(\bar{x})\}_{0 \leq i \leq n}$ computed as above for $\mathbf{s}$ iff there exists a run $\rho = \{(I_i, g_i)\}_{0 \leq i \leq n}$ in $\mathit{rep}(\mathbf{s})$, and a valuation v of $\bar{x}$ into $\mathit{adom}(\rho)$ such that the following holds for every i $(0 \leq i \leq n)$, $z \in \bar{x}$, $t \in \mathit{var}(\mathcal{I}_i) \cup \mathit{adom}(\mathcal{I}_i, \varphi_i)$, and the unique valuation v_i such that $I_i = v_i(\mathcal{I}_i)$: $\beta_i(\bar{x}) \models z = t$ iff $v(z) = v_i(t)$.

Intuitively, (†) says that each equality type induced by $\beta_i(\bar{x})$ wrt the constants and variables in $\mathcal{I}_i$ is realizable in a run $\rho \in \mathit{rep}(\mathbf{s})$ for some *fixed* valuation of $\bar{x}$ in the $\mathit{adom}(\rho)$. Furthemore, the sequence of formulas $\{\psi_i(\bar{x})\}_{0 \leq i \leq n}$ can be computed successfully for $\mathbf{s}$ iff such a run ρ and valuation v exists. We define the following extension of our notion of $\mathcal{P}_\tau$-constrained run. A *parameterized* $\mathcal{P}_\tau$-constrained run is a sequence $\mathbf{s}(\bar{x}) = \{((\mathcal{I}_i, \varphi_i, \psi_i(\bar{x})), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ where $\{((\mathcal{I}_i, \varphi_i), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ is a $\mathcal{P}_\tau$-constrained run and the sequence $\{\psi_i(\bar{x})\}_{0 \leq i \leq n}$ is computed as above. Also, we refer to each $(\mathcal{I}_i, \varphi_i, \psi_i(\bar{x}))$ as a *parameterized* I-instance.

We will use the following notion of isomorphic parameterized I-instances. Given $(\mathcal{I}, \varphi, \psi(\bar{x}))$, let $\sim$ be the equivalence relation on variables and constants in $\mathit{var}(\mathcal{I}) \cup \{\bar{x}\} \cup \mathit{adom}(\mathcal{I}, \varphi)$ defined by $z \sim t$ iff $\varphi \wedge \psi(\bar{x}) \models z = t$, and let $[z]$ be the equivalence class of z wrt $\sim$. Let $\mathcal{I}/_\sim$ be obtained by replacing in $\mathcal{I}$ each variable z by the unique constant in $[z]$, if it exists, or otherwise by the variable of smallest index in $[z]$. We say that $(\mathcal{I}_1, \varphi_1, \psi_1(\bar{x}))$ and $(\mathcal{I}_2, \varphi_2, \psi_2(\bar{x}))$ are isomorphic if $\mathcal{I}_1/_\sim$ and $\mathcal{I}_2/_\sim$ are isomorphic when variables are frozen as distinct constants. The *isomorphism type* of $h = (\mathcal{I}, \varphi, \psi(\bar{x}))$ is its equivalence class under isomorphism, denoted $\hat{h}$.

Let $\mathbf{s}(\bar{x}) = \{((\mathcal{I}_i, \varphi_i, \psi_i(\bar{x})), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ be a $\mathcal{P}_\tau$-constrained parameterized run. Consider the evaluation of $\phi_f(\bar{x})$. Clearly, for each $i \geq 0$, $\varphi_i \wedge \psi_i(\bar{x})$ completely determines the isomorphism type of $(\mathcal{I}_i, \varphi_i, \psi_i(\bar{x}))$. Thus, the truth value of each FO component $f(r)(\bar{x})$ of $\phi_f(\bar{x})$ is well defined and can be evaluated at each $(\mathcal{I}_i, \varphi_i, \psi_i(\bar{x}))$. As before, the algorithm outputs YES if a $\mathcal{P}_\tau$-constrained parameterized run $\mathbf{s}(\bar{x})$ can be generated that is accepted by both A_ϕ and $A(\mathcal{P}_\tau)$. The complexity remains PSPACE wrt $\phi_f(\bar{x})$ and τ. □

Remark 4.4 *As stated in Theorem 4.3, the algorithm described above has complexity* PSPACE *wrt $\phi_f(\bar{x})$ and τ. It is of interest to note the impact of the* length *of τ on complexity. It is easy to see that, if* adom(τ) *and $\varphi_f(\bar{x})$ are fixed, the algorithm is in NL (nondeterministic logarithmic space) in the length of τ.*

Remark 4.5 *Theorem 3.2 provided examples of useful properties that are undedecidable without the key-visible restriction. As a consequence of Theorem 4.3, all questions of Theorem 3.2 become decidable for key-visible specs.*

Incremental monitoring We next adapt the algorithm described in the proof of Theorem 4.3 in order to incrementally monitor PLTL-FO properties. The goal is to avoid re-evaluating the formula after each move. We will present an incremental algorithm that avoids computations that depend on the entire trace. However, as we will see, this is at the cost of maintaining a possibly very large auxiliary structure.

Consider a PLTL-FO property $\phi_f(\bar{x})$ to be monitored. An incremental algorithm for evaluating $\mathit{poss}(\phi_f(\bar{x}))$ on a p-trace τ uses two functions, *aux* and inc_{aux}. As we shall see, $\mathit{aux}(\tau)$ provides enough information to answer $\mathit{poss}(\phi_f(\bar{x}))$, and provides additional information needed to incrementally maintain its own value using the second function inc_{aux}. More precisely, for a new observation (J, f) at peer p, $\mathit{aux}(\tau \cdot (J, f)) = \mathit{inc}_{aux}(\mathit{aux}(\tau), (J, f))$.

The functions *aux* and inc_{aux} are defined as follows. Consider first *aux*. Intuitively, $\mathit{aux}(\tau)$ consists of all I-instances $(\mathcal{I}, \varphi)$ with associated formula $\psi(\bar{x})$ reachable by complete runs in $\nu^{-1}(\tau)$, together with the set of states of A_ϕ reachable on such runs. More precisely, $\mathit{aux}(\tau)$ consists of the set of tuples $(\mathcal{I}, \varphi, \psi(\bar{x}), Q)$ where:

- there exists a complete $\mathcal{P}_\tau$-constrained parameterized run $\mathbf{s}(\bar{x}) = \{((\mathcal{I}_i, \varphi_i, \psi_i(\bar{x})), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ accepted by A_τ, defined as in the proof of Theorem 4.3, for which $(\mathcal{I}, \varphi) = (\mathcal{I}_n, \varphi_n)$, and $\psi(\bar{x}) = \psi_n(\bar{x})$,
- Q is the set of states of A_ϕ reachable from the initial state on some run $\mathbf{s}$ as above.

Clearly, $\mathit{poss}(\phi_f(\bar{x}))$ is true on τ iff there exists $(\mathcal{I}, \varphi, \psi(\bar{x}), Q)$ in $\mathit{aux}(\tau)$ for which Q contains an accepting state of A_ϕ.

Next, consider the function inc_{aux}. Given $\mathit{aux}(\tau)$ as above, and a new observation (J, f) at peer p, $\mathit{inc}_{aux}(\mathit{aux}(\tau), (J, f))$ consists of all $(\mathcal{I}', \varphi', \psi'(\bar{x}), Q')$ such that, for some $(\mathcal{I}, \varphi, \psi(\bar{x}), Q)$ in $\mathit{aux}(\tau)$:

- there exists a J-constrained parameterized run suffix $\mathbf{s}(\bar{x}) = \{((\mathcal{I}_i, \varphi_i, \psi_i(\bar{x})), (e_i, \gamma_i, J))\}_{0 \leq i \leq n}$, where:
  - (i) $(\mathcal{I}', \varphi') = (\mathcal{I}_n, \varphi_n)$ and $\psi'(\bar{x}) = \psi_n(\bar{x})$,
  - (ii) $(\mathcal{I}, \varphi) \vdash^J_{e_0, \gamma_0} (\mathcal{I}_0, \varphi_0)$, $\mathit{peer}(e_0) = p$ if $f = p$ and $\mathit{peer}(e_0) \neq p$ if $f = \star$, and $\mathit{peer}(e_i) \neq p$ for $i > 0$,

(iii) $\psi_0(\bar{x})$ is computed from $\psi(\bar{x})$ and the initial transition $(\mathfrak{I}, \varphi) \vdash^J_{e_0,\gamma_0} (\mathfrak{I}_0, \varphi_0)$

- Q' is the set of states of A_ϕ reachable from some $q \in Q$ on runs $\mathbf{s}$ as above.

Clearly, $inc_{aux}(aux(\tau), (J, f)) = aux(\tau \cdot (J, f))$, as desired.

Since $cert(\phi_f(\bar{x})) = \neg poss(\neg\phi_f(\bar{x}))$, the incremental evaluation algorithm for $poss(\neg\phi_f(\bar{x}))$ also provides an incremental evaluation algorithm for $cert(\phi_f(\bar{x}))$.

Clearly, the size of $aux(\tau)$ is exponential in $adom(\tau)$ and ϕ (for $\mathcal{W}$ fixed). The function inc_{aux} can be computed in EXPTIME wrt $adom(\tau)$ and ϕ. In terms of complexity, the main advantage of incremental evaluation over re-evaluation on the entire run is that the complexity wrt τ depends only on the size $adom(\tau)$ and not on the length of τ. However, this has to be balanced against the need to create intermediate results of exponential size wrt $adom(\tau)$ and ϕ.

Pre-emptive monitoring We have so far considered the incremental monitoring of statically specified properties. Suppose that the properties to be monitored are not known ahead of time but instead may be specified dynamically as the run unfolds. Is some form of incremental evaluation still possible? We provide here a partially affirmative answer. Indeed, we show that large classes of properties can be preemptively monitored, as long as partial information is available on the *type* of temporal property they specify. More precisely, the *temporal type* of a PLTL-FO property $\phi_f(\bar{x})$ is the propositional formula ϕ. For example, commonly arising types include $G^{-1}r$, or $F^{-1}r$, or $G^{-1}(r_1 \rightarrow F^{-1}r_2)$. In addition to the temporal type, we also need to know the maximum number of free variables $|\bar{x}|$.

Definition 4.6 *A PLTL-FO property type is a pair (Φ, m), where Φ is a finite set of PLTL formulas and $m \geq 0$. A PLTL-FO formula $\phi_f(\bar{x})$ for $\mathcal{W}$ is of type (Φ, m) if $\phi \in \Phi$ and $|\bar{x}| \leq m$.*

For example, $(\{G^{-1}r,\ F^{-1}r,\ G^{-1}(r_1 \rightarrow F^{-1}r_2)\},\ 10)$ is a PLTL-FO type.

We next outline an incremental algorithm that allows to evaluate *all* formulas of a given type (Φ, m). Note that there are infinitely many such formulas. Let P_Φ be the set of propositions occuring in Φ. The main idea of the algorithm is to modify the incremental algorithm for monitoring $\phi_f(\bar{x})$ described in the previous section as follows. Recall that the algorithm generates constrained parameterized runs and produces the tuples $(\mathfrak{I}, \varphi, \psi(\bar{x}), Q)$ of reachable I-instances, constraint $\psi(\bar{x})$ on the free variables $\bar{x}$, and the set Q of corresponding states reachable in the automaton A_ϕ. The input of A_ϕ at each transition consists of the truth value to the propositions of ϕ induced by the FO components $f(r)$. In our case, the FO components are unknown. Instead of evaluating each $f(r)$, the new algorithm simply *guesses* the truth assignments σ for the propositions in ϕ, for the isomorphism types of all reachable I-instances and free variables $\bar{x}$.

Let τ be a p-trace. Let $\mathcal{S}(\tau)$ be the set of all isomorphism types[1] of $(\mathfrak{I}, \varphi, \psi(\bar{x}))$ such that there is a $\mathcal{P}_\tau$-constrained parameterized run $\mathbf{s}(\bar{x}) = \{((\mathfrak{I}_i, \varphi_i, \psi_i(\bar{x})), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ accepted by A_τ, with $\mathfrak{I} = \mathfrak{I}_j, \varphi = \varphi_j, \psi(\bar{x}) = \psi_j(\bar{x})$ for some $j \in [0, n]$. A truth assignment mapping for $\mathcal{S}(\tau)$ is a mapping Σ from $\mathcal{S}(\tau)$ to truth assignments of P_Φ.

The auxiliary information $aux(\tau)$ computed by the incremental algorithm now consists of the set of all pairs $(\Sigma, \mathcal{H})$ where Σ is a truth assignment mapping for $\mathcal{S}(\tau)$ and $\mathcal{H}$ is the set of tuples $(\mathfrak{I}, \varphi, \psi(\bar{x}), \{Q_\pi \mid \pi \in \Phi\})$ where:

- there exists a complete $\mathcal{P}_\tau$-constrained parameterized run $\mathbf{s}(\bar{x}) = \{((\mathfrak{I}_i, \varphi_i, \psi_i(\bar{x})), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$ accepted by A_τ, for which $(\mathfrak{I}, \varphi) = (\mathfrak{I}_n, \varphi_n)$, and $\psi(\bar{x}) = \psi_n(\bar{x})$,
- for each $\pi \in \Phi$, Q_π is the set of states of A_π reachable from the initial state on some run $\mathbf{s}(\bar{x})$ as above, where the truth assignment for P_Φ at the i-th transition is $\Sigma(\hat{h})$, for $h = (\mathfrak{I}_i, \varphi_i, \psi_i(\bar{x}))$.

The function $aux(\tau)$ can be maintained incrementally by a function inc_{aux} similar to the previous section. The set Σ of truth assignment mappings is maintained by augmenting it with truth assignments for isomorphism types of newly reached instances in the run suffixes generated when a new observation (J, f) is added (we ommit the straightforward details).

Now suppose that we wish evaluate $poss(\phi_f(\bar{x}))$ for a PLTL-FO formula $\phi_f(\bar{x})$ of type (Φ, m), for the p-trace τ. Let $aux(\tau)$ be as defined above. Let Σ be such that for each proposition r of ϕ and every $\hat{h} \in \mathcal{S}(\tau)$, $f(r)$ holds in $\hat{h}$ iff $\Sigma(\hat{h})(r) = 1$. Let $\mathcal{H}$ be such that $(\Sigma, \mathcal{H}) \in aux(\tau)$. Then $poss(\phi_f(\bar{x}))$ holds iff there exists $(\mathfrak{I}, \varphi, \psi(\bar{x}), \{Q_\pi \mid \pi \in \Phi\}) \in \mathcal{H}$ such that Q_ϕ contains an accepting state of A_ϕ.

To evaluate the size of $aux(\tau)$, note that the number of isomorphism types in $\mathcal{S}(\tau)$ is exponential in the maximum size of an I-instance in τ (and independent of its active domain). Thus, the number of truth assignment mappings Σ is double exponential in the same (single exponential for fixed type (Φ, m)). For each Σ, the size of $\mathcal{H}$ is exponential in (Φ, m) and $adom(\tau)$. Finally, the evaluation of a PLTL-FO property $\phi_f(\bar{x})$ of type (Φ, m) on $aux(\tau)$ is in PSPACE.

Clearly, the use of preemptive incremental monitoring becomes beneficial compared to direct evaluation over the entire p-trace τ only under certain conditions, including the following: (i) $adom(\tau)$ is small relative to the length of τ, (ii) the number of isomorphism types of parameterized I-instances in runs of $A_\tau(\mathcal{P}_\tau)$ is small relative to

[1] Recall the definition in the proof of Theorem 4.3.

$adom(\tau)$, and (iii) the number of formulas of type (Φ, m) to be evaluated is large.

Introspective closure We showed how a peer can reason about temporal properties of global runs based on its local observations. In many cases, it would be desirable for a peer to be able to use the information gained by such reasoning to make decisions on the actions it takes in the workflow. A natural question is whether the specification language we defined would need to be extended or whether it is already closed under such introspective reasoning. We next show that it is closed under introspective reasoning, for a natural definition of simulation.

We can straightforwardly define an extension of workflow specs allowing the use in conditions of atoms of the form $poss(\phi_f(\bar{x}))$ and $cert(\phi_f(\bar{x}))$, that we refer to as *introspective atoms*. The semantics of these atoms (that refer to the global run) is as previously defined. Specifically, $poss(\phi_f(\bar{x}))$ is evaluated on the p-trace of the run leading to the current application of the action. We refer to specs that allow introspective atoms in the actions of peers p for which the spec is key-visible, as *introspective* specs.

In order to compare the expressiveness of introspective and regular specs, we define a natural notion of simulation. Intuitively, a spec simulating $\mathcal{W}$ is allowed to use additional relations and actions, but its restriction to the relations and actions of $\mathcal{W}$ must yield exactly the runs of $\mathcal{W}$. We make this more precise. First, consider a spec $\mathcal{W}$, let $\mathcal{D}_0$ be a subset of its schema and $\mathcal{A}_0$ a subset of its actions. For each run ρ of $\mathcal{W}$, the projection of $\rho = \{(I_i, e_i)\}_{0 \leq i \leq n}$ on $\mathcal{D}_0$ and $\mathcal{A}_0$, denoted $\pi_{\mathcal{D}_0, \mathcal{A}_0}(\rho)$, is the sequence obtained by removing from ρ all terms (I_i, e_i) for which $action(e_i) \notin \mathcal{A}_0$ and restricting each instance in the remaining sequence to $\mathcal{D}_0$.

Let $\mathcal{W}_1$ and $\mathcal{W}_2$ be specs with the same set of peers, both key-visible at p. We denote by $\mathcal{D}_i$ the schema of $\mathcal{W}_i$. We say that $\mathcal{W}_2$ *simulates* $\mathcal{W}_1$ if: (i) $\mathcal{D}_1 \subseteq \mathcal{D}_2$, (ii) each action α of $\mathcal{W}_1$ has a corresponding action $\bar{\alpha}$ in $\mathcal{W}_2$ at the same peer (we denote $\bar{\mathcal{A}}_1 = \{\bar{\alpha} \mid \alpha \in \mathcal{A}_1\}$), and

$$
\begin{aligned}
(iii)\quad & \{\pi_{\mathcal{D}_1, \bar{\mathcal{A}}_1}(\rho_2) \mid \rho_2 \text{ is a run of } \mathcal{W}_2\} = \\
& \{\{(I_i, \bar{e}_i)\}_{0 \leq i \leq n} \mid \{(I_i, e_i)\}_{0 \leq i \leq n} \text{ is a run of } \mathcal{W}_1, \\
& peer(e_i) = peer(\bar{e}_i), action(\bar{e}_i) = \overline{action}(e_i), \\
& val(\bar{e}_i) = val(e_i), i \in [0, n]\}
\end{aligned}
$$

We can show the following (see Appendix).

Theorem 4.7 *For every introspective workflow spec $\mathcal{W}_1$ there exists a workflow spec $\mathcal{W}_2$ that simulates $\mathcal{W}_1$.*

Acknowledgement We wish to thank Val Tannen for useful discussions on the update propagation model of collaborative workflows, inspired by the Orchestra approach to data sharing.

References

[1] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Inf. and Comp.*, 127(2), 1996.

[2] S. Abiteboul, E. Antoine, and J. Stoyanovich. Viewing the web as a distributed knowledge base. In *ICDE*, 2012.

[3] S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *VLDB J.*, 17(5), 2008.

[4] S. Abiteboul, M. Bienvenu, A. Galland, and E. Antoine. A rule-based language for web data management. In *PODS*, pages 293–304, 2011.

[5] S. Abiteboul, P. Bourhis, and V. Vianu. Comparing workflow specification languages: A matter of views. *TODS*, 37(2), 2012.

[6] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[7] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *JCSS*, 43(1), 1991.

[8] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. Sears. Dedalus: Datalog in time and space. In *Datalog*, pages 262–281, 2010.

[9] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The hyperion project: from data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003.

[10] D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2), 1983.

[11] J. Brzozwski and E. Leiss. Finite automata and sequential networks. *Theoretical Computer Science*, 10, 1980.

[12] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *PODS*, 2006.

[13] E. A. Emerson. Temporal and modal logic. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*. North-Holland Pub. Co./MIT Press, 1990.

[14] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. Orchestra: facilitating collaborative data sharing. In *SIGMOD*, 2007.

[15] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1), 2010.

[16] R. Hull. Web services composition: A story of models, automata, and logics. In *ICSOC*, 2005.

[17] R. Hull and J. Su. Tools for composite web services: a short overview. *SIGMOD Record*, 34(2):86–95, 2005.

[18] T. Imieliński and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[19] Z. G. Ives, T. J. Green, G. Karvounarakis, N. E.

Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira. The orchestra collaborative data sharing system. *SIGMOD Record*, 37(3), 2008.

[20] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[21] I. Tatarinov, Z. G. Ives, J. Madhavan, A. Y. Halevy, D. Suciu, N. N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. *SIGMOD Record*, 32(3):47–52, 2003.

[22] M. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, 1995.

Appendix

A Symbolic transitions

We describe the symbolic transitions of Section 3 in detail. For convenience, we first define transition constraints that are *not* necessarily conjunctive. Subsequently, each such transition is replaced with a set of transitions, one for each disjunct in the disjunctive normal form (DNF) of the constraint, yielding conjunctive transition constraints.

We will need the notion of *active domain* of $(\mathcal{I}, \varphi)$, denoted $adom(\mathcal{I}, \varphi)$. This consists of the set of constants c in **dom** that

- occur explicitly in some tuple of $\mathcal{I}$; or
- occur in a conjunct $x = c$ of φ; or
- occur in $\mathcal{W}$ or $\{\perp\}$.

Consider, as above, an I-instance $(\mathcal{I}, \varphi)$, a peer q and an action $Update(\bar{x}, \bar{y})$:- $Condition(\bar{x})$ at peer q. A *valuation* for the variables of the action into $\mathcal{I}@q$ is a mapping v from $\bar{x} \cup \bar{y}$ (extended with the identity on constants) such that:

- v maps $\bar{x}$ to variables and constants in $\mathcal{I}@q$, and $\bar{y}$ to the first $|\bar{y}|$ distinct variables in $\mathbf{var} - var(\mathcal{I})$ with the smallest index[2].
- for each $R@q(\bar{z})$ of $Condition(\bar{x})$, $R@q(v(\bar{z}))$ is a tuple in $\mathcal{I}@q$.

The transition constraint γ_v induced by v is the conjunction of the following:

- $v(x) = (\neq)\ v(y)$ where $x = (\neq)\ y$ is an (in)equality in $Condition(\bar{x})$
- for each $\neg R@q(\bar{z})$ in $Condition(\bar{x})$ and tuple $R@q(\bar{w})$ in $\mathcal{I}@q$, the constraint $v(\bar{z}) \neq \bar{w}$.
- $v(y) \neq t$ where $y \in \bar{y}$ and t is a variable in $\mathcal{I}$ or a constant in the active domain of $(\mathcal{I}, \varphi)$.

Note that the above is *not* a conjunctive constraint because of the tuple inequality in the second item.

Next, fix a valuation v as above and consider $Update(v(\bar{x}), v(\bar{y}))$. We describe the effect of tuple insertions and deletions, with the associated transition constraints. Consider first a tuple insertion $R@q(v(\bar{z}))$. Let $R(v(\bar{z})\ \perp^*)$ be the extension of $R@q(v(\bar{z}))$ to $att(R)$ obtained by padding the missing attributes with $\perp$. For each tuple $R(\bar{w})$, denote by $\bar{w}_K$ the subsequence of $\bar{w}$ corresponding to the key K of R. Similarly, let $\bar{z}_K$ consist of the subsequence of $\bar{z}$ correponding to K. If $\bar{z}_K$ contains some variable in $\bar{y}$ then the result of the insertion consists of adding $R(v(\bar{z})\ \perp^*)$ to R. Otherwise, the result depends on whether $R(v(\bar{z})\ \perp^*)$ agrees with an

[2]This is done to use variables economically, which is needed for technical reasons explained further.

existing tuple on the key. More precisely, the transitions generated by the insertion are as follows:

- For each tuple $R(\bar{w})$ in $\mathcal{I}$, the result of the insertion under the transition constraint $v(\bar{z}_K) = \bar{w}_K$ is obtained by chasing $R(\bar{w})$ as follows. Let A be an non-key attribute of R and z_A, w_A be the values of $R(v(\bar{z})\ \perp^*)$ and $R(\bar{w})$ for attribute A. If $z_A, w_A \in \mathbf{dom} - \{\perp\}$, the chase fails and there is no transition. If $w_A = \perp$ then it is replaced by z_A. If w_A and z_A are both variables, then $w_A = z_A$ is added to the transition constraint.
- Finally, one transition occurs for each disjunct in the DNF of the constraint consisting of the conjunction of $\bar{w}_K \neq v(\bar{z}_K)$ for all tuples $R(\bar{w})$ in $\mathcal{I}$, yielding the instance obtained by inserting the tuple $R(v(\bar{z})\ \perp^*)$ into $\mathcal{I}$.

Consider now a tuple deletion $\neg R@q(v(\bar{z}))$. The result depends again on agreement with existing tuples on the key attributes. Recall that deleted tuples contain no "new" variables among $\bar{y}$. There is one possible transition for each tuple $R(\bar{w})$ in $\mathcal{I}$, consisting of deleting the tuple under the transition constraint $\bar{w}_K = \bar{z}_K$. In addition there are transitions leaving $\mathcal{I}$ unchanged, for the constraint consisting of the conjunction of all inequalities $\bar{z}_K \neq \bar{w}_K$ for all tuples $R(\bar{w})$ in $\mathcal{I}$. As earlier, each disjunct in the DNF of the constraint generates a separate transition.

Finally, the transitions caused by the sequence of updates in $Update(\bar{x}, \bar{y})$ are the compositions of the transitions for each update. Each transition constraint is the conjunction of the constraints for the composed transitions. Note that, by construction, these are conjunctive constraints. The local constraint ψ for each resulting I-instance $(\mathcal{J}, \psi)$ consists of the closure of $\varphi \wedge \gamma$ on the variables of $\mathcal{J}$, where γ is the corresponding transition constraint. Note that this again yields a conjunctive constraint.

B Some proofs

Proof of Lemma 3.7 First note that there exists $M > 0$ so that for every $\mathcal{P}_\tau$-constrained run $\{((\mathcal{I}_i, \varphi_i), (e_i, \gamma_i, P_{j_i}))\}_{0 \leq i \leq n}$, the set of variables occurring in $\mathcal{I}_j$ is included in $\{x_1, \ldots, x_M\}$ for each j. This is due to the following:

- there is a fixed bound on the number of tuples (and therefore variables) in a P-instantiated I-instance for $P \in \mathcal{P}_\tau$,
- the variables in $\mathcal{J}_0$ are $\{x_1, \ldots, x_m\}$ for some $m \geq 0$, and
- by construction of S-transitions, new variables introduced by transitions are picked among those of smallest index that are currently unused.

Finally, there are finitely many conjunctive constraints using the variables $\{x_1, \ldots, x_M\}$ and constants occurring in $\mathcal{P}_\tau$, $\mathcal{W}$, or $\{\perp\}$.

Proof of Theorem 3.8 We use the following property, that considers partial instantiations of S-transitions. Let $(\mathcal{I}, \varphi) \vdash_{e,\gamma} (\mathcal{J}, \psi)$ be an S-transition and ν a partial valuation of the variables of $\mathcal{I} \cup \mathcal{J}$ into $\mathbf{dom}$. For a constraint β, let $\nu(\beta)$ denote the constraint obtained by replacing in β each variable $x \in dom(\nu)$ by $\nu(x)$. For an event $e = (p, \alpha, v)$ we denote by $\nu(e)$ the event $(p, \alpha, v \circ \nu)$. The following is shown similarly to Lemma 3.4.

(†) Let $(\mathcal{I}, \varphi) \vdash_{e,\gamma} (\mathcal{J}, \psi)$ be an S-transition and ν a partial valuation of the variables of $\mathcal{I} \cup \mathcal{J}$ into $\mathbf{dom}$ such that $\nu(\varphi \wedge \gamma \wedge \psi)$ is satisfiable. Then $(\nu(\mathcal{I}), \nu(\varphi)) \vdash_{\nu(e),\nu(\gamma)} (\nu(\mathcal{J}), \nu(\psi))$ is also an S-transition.

Lemma 3.4 together with (†) shows the following completeness result.

(‡) For each I-instance $(\mathcal{I}, \varphi)$ and instance J_p at peer p,

$$\begin{aligned}\{(I \vdash_e J) \mid\ & I \in rep(\mathcal{I}, \varphi), e \text{ is an event}, , J@p = J_p\} = \\ \{(I \vdash_e J) \mid\ & \text{there exists } (\mathcal{I}, \varphi) \vdash^{J_p}_{e,\gamma} (\mathcal{J}, \psi) \text{ such that} \\ & (I \vdash_e J) \in rep((\mathcal{I}, \varphi) \vdash^{J_p}_{e,\gamma} (\mathcal{J}, \psi))\}\end{aligned}$$

Theorem 3.8 now follows from (‡) and the construction of A_τ.

Proof of Lemma 4.2 (i) Consider $J_i \in rep(\mathcal{I}, \varphi)$, such that $J_i = \nu_i(\mathcal{I})$, $i = 1, 2$. Define the mapping h from J_1 to J_2 by $h(\nu_1(t)) = \nu_2(t)$ for $t \in var(\mathcal{I}) \cup adom(\mathcal{I}, \varphi)$. It is easy to see that, because of completeness of $(\mathcal{I}, \varphi)$, h is well defined and an isomorphism from J_1 to J_2 fixing $adom(\mathcal{W})$. Since $f(r)$ uses only constants in $adom(\mathcal{W})$, it has the same truth value on J_1 and J_2. (ii) The preservation of completeness by transitions is due to the fact that all newly introduced variables in a transition are constrained to differ from all variables and constants in the active domain of the current I-instance.

Proof of Theorem 4.7 Let $\mathcal{W}_1$ be an introspective workflow. Let p be a peer such as $\mathcal{W}_1$ is key-visible at p. The simulation by $\mathcal{W}_2$ of introspective atoms used in actions of p has two main aspects. First, $\mathcal{W}_2$ uses additional relations to store the p-trace of the current run. This is done by copying, at each transition caused by p or with side-effects at p, the corresponding observation in the p-trace. Moreover, each copy is timestamped by a new value created using a variable occurring only in the updates of an action, and the timestamps are ordered. Doing this at each transition requires additional control, which is enforced using additonal propositions. Second, peer p must evaluate introspective atoms $poss(\phi_f(\bar{x}))$ or

$cert(\phi_f(\bar{x}))$ on the currently stored p-trace. This can be done because sets of actions at p, with appropriate control provided by propositions, are computationally complete. Once again, this is due to the ability to create new values using variables occurring only in the updates of actions. The proof is similar to the query completeness of nondeterministic Datalog$^\neg$ with value invention (using variables occurring only in heads of rules), see [7].