Provenance Analyzer: Exploring Provenance Semantics with Logic Rules

*Saumen Dey** *Sean Riddle** *Bertram Ludäscher**

1 Introduction

Consider the snippet of an OPM (Open Provenance Model [8]) graph shown in Fig. 1. It consists of a single `wasGeneratedBy` edge with two nodes, data artifact A and process P.

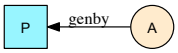

Fig. 1: Data artifact A `wasGeneratedBy` process P

What exactly can we say about the dependency between artifact A and process P? Can we say that A was in fact generated by P, or was A generated by another process Q which used an artifact B that was generated by P? In fact, we cannot be sure. In particular, the given provenance information is insufficient to describe unambiguously in which order some basic OPM events (i.e., start of P, creation of A, end of P) occurred.

The authors of [4] introduced *precise edges* (in addition to the default imprecise edges) to help narrow such ambiguities, provided a formal temporal OPM semantics via partial orders, and gave an efficient deductive procedure to compute all constraints implied by the axioms. Precise edges are labeled with a *role*, and represent a direct rather than transitive relationship, whereas an imprecise edge can represent a direct or a transitive relationship.

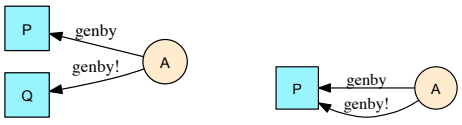

Fig. 2: Two different possibilities in the trace file

Let us examine different scenarios in which the edge shown in Fig. 1 could exist: (i) the imprecise `wasGeneratedBy` edge $A \overset{genby}{\rightarrow} P$ in Fig. 1 is all we have, or (ii) P generated an artifact B (not shown) and Q used B to create A as in Fig. 2(a), indicated by a precise edge[1] $A \overset{genby!}{\rightarrow} Q$, or (iii) P did in fact create A and there is a precise edge as in Fig. 2(b) to indicate this. In the latter case, the imprecise edge is redundant.[2]

Note that even though Fig. 2(b) specifies an unambiguous total order of events (assuming that events cannot occur simultaneously), Fig. 1 and Fig. 2(a) do not. The situation only gets more complex when additional edge types are introduced.

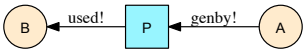

Fig. 3: P generated A, but did it use B for that?

In Fig. 3, there are five temporal events: create(B), use(B), begin(P), end(P), and create(A). One obvious order is create(B)≤begin(P)≤use(B)≤create(A)≤end(P). But also legal is one in which begin(P)≤create(B). There are more dramatic ones as well, e.g., (a) where all events occur at the same time (e.g., via a sufficiently coarse-grained clock [4]) or (b) data artifact A was generated first and *then* data artifact B was used! The many different ways these events might have occurred can make it difficult to understand and use provenance. In addition, these ambiguities may make system to system provenance data sharing impossible.

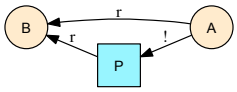

Fig. 4: Refinement of Fig. 3 via a *generate-use-derive* triangle [4]

Fig. 4 refines and clarifies the situation via a *generate-use-derive* triangle [4]: The r-labeled `wasDerivedFrom` edge between A and B, together with the r-labeled `use` edge from P to B and the third precise edge `wasGeneratedBy` from A to P imply that A was in fact generated by P and that P used B in the creation of A. The additional knowledge captured in this way eliminates certain possible worlds as create(A) cannot precede use(B) anymore. In fact, the order of events implied by Fig. 4 is almost completely determined, the only excep-

*{scdey,swriddle,ludaesch}@ucdavis.edu. Dept. of Computer Science, UC Davis. The first and second authors contributed equally.

[1] Similar to [4], we use "!" to indicate precise relations.

[2] Axiom 2 in [4] states for a precise `wasGeneratedBy` edge we have: begin(P)≤create(A)≤end(P), while for an imprecise `wasGeneratedBy` edge, Axiom 5 only states: begin(P)≤create(A), so no additional constraints are inferred from the imprecise edge.

```
data(D).     % data identifier D
invoc(I).    % process invocation id I
usd(I,D).    % process invocation I used data D
gen(D,I).    % data D was generated by invocation I
der(D2,D1). % data D2 was derived form D1
inf(I2,I1). % invocation I1 was informed by I2
```

Fig. 5: Datalog schema used in Provenance Analyzer.

tion that the events create(B) and begin(P) can be in either order.

We are developing *Provenance Analyzer*, a framework and system prototype that allows users to compute all *possible worlds*, i.e., all orders in which the events of a given OPM graph may have occured and then analyze these alternative orders to better understand the provenance data and their respective dependencies. In particular, our system allows enumeration of the (often very large number of) possible orders, as well as queries about which dependencies are true in all (or just some) of the possible world. The main idea is to encode provenance semantics contained in a particular set of axioms and constraints as logic rules, then employ a powerful reasoning framework to systematically generate and test all possible orders. In our current prototype, we encoded the temporal axioms and constraints in [4] as logic rules. Our system facilitates experimenting with different provenance semantics: a provenance researcher can simply add, modify, or delete axioms and study the repercussions using actual examples.

Below, we first introduce a variant of OPM, modeled in Datalog. We then describe the Provenance Analyzer framework and implementation. We end with a brief discussion of related work and conclusions.

2 Provenance Model

In our Datalog variant of the Open Provenance Model (OPM) [8], we focus on data *artifacts* and *processes* and the relationships between them. The key relations are `used`, `wasGeneratedBy`, `wasDerivedFrom`, and `wasInformedBy`, with shorthands `usd`, `gen`, `der`, and `inf`, respectively.[3] An OPM graph $G = (V,E)$ is a directed graph whose nodes $V = D \cup I$ are *data artifacts* D or *process invocations* I; and edges $E \subseteq E_{usd} \cup E_{gen} \cup E_{der} \cup E_{inf}$, $E_{usd} = I \times D$, $E_{gen} = D \times I$, $E_{der} = D \times D$, $E_{inf} = I \times I$. We use the relations as shown in Fig. 5 to capture the node and edge information in our system.

Extension of OPM. In [4], Kwasnikowska *et al.* extend the OPM model with distinct *precise* edges, in addition to the (default) imprecise edges. This model is represented by the relations in Fig. 6 in addition to those in Fig. 5. Using this model, a provenance graph $G = (V,R,E)$ is a directed graph whose nodes $V = D \cup I$ are *data artifacts* D or *process invocations* I; R is a set of roles; and edges $E \subseteq E_p \cup E_m$, precise edges $E_p = E_{pGen} \cup E_{pUsd} \cup E_{pDer}$, and imprecise edges $E_m = E_{der} \cup E_{inf} \cup E_{gen} \cup E_{usd}$, where $E_{pGen} = D \times R \times I$, $E_{pUsd} = I \times R \times D$, $E_{pDer} = D \times R \times D$, $E_{usd} = I \times D$, $E_{gen} = D \times I$, $E_{der} = D \times D$, and $E_{inf} = I \times I$.

```
pGen(D,R,I).    % data D was generated by
                % invocation I with role R
pUsd(I,R,D).    % invocation I used data D
                % with role R
pDer(D2,R,D1). % data D2 was derived form
                % D1 with role R
```

Fig. 6: Extended Datalog schema for precise edges.

Formal temporal extension of OPM. A formal temporal semantics for OPM in the form of a translation from OPM graph patterns to temporal constraints is given in [4] (e.g., an artifact cannot be used before it is created). The OPM graphs can optionally be annotated with additional constraints for specific events if they are known in the context of a given computation. With the temporal semantics, OPM is extended as $G = (V,R,E,S)$, where $S = (Ev,T,M)$, Ev is the set of events, T is a set of time points, $M \subset Ev \times T$ is the mapping from events to time points, and $Ev = Ev_{create} \cup Ev_{use} \cup Ev_{begin} \cup Ev_{end}$. There is an Ev_{create} event for each data artifact and an Ev_{use} event for each use. Each process has an Ev_{begin} and Ev_{end} event. The relation `time`/1 contains the set of available time points and the `at`/2 relation maintains the mapping of an event to a time point.[4] The predicates used to encode this temporal extension are in Fig. 7.

```
event(E).       % event identifier E
create(E,D).    % create event for the data D
begin(E,I).     % begin event of invocation I
end(E,I).       % end event of invocation I
useR(E,I,R,D). % precise use event of data D by
                % invocation I with role R
use(E,I,D).     % imprecise use event of data D
                % by invocation I
time(T).        % time point T
at(E,T).        % mapping of event E and time T
```

Fig. 7: Datalog schema for temporal relations

[3] We ignore the `agents`, `wasControlledBy`, and `wasTriggeredBy` entities in the OPM model.

[4] We write R/α to indicate that relation R has arity α.

```
(1) after(X,Y) :- ev(X), ev(Y), X != Y,
                  not after(Y,X).
(2) after(X,Y) :- after(X,Z), after(Z,Y).
(3) :- after(X,Y), after(Y,X).
```

Fig. 8: Rules producing all strict total orders over ev/1

3 Other Preliminaries

Before presenting the details of our approach, we recall some necessary background and basic definitions on partial and total orders, as these are essential to our approach. We also briefly introduce Answer Set Programming (ASP), which we use in each of our approaches to search the problem space.

Total and Partial Orders. A (non-strict) *partial order* is a binary relation $\leq$ over a set X that is *reflexive*, *antisymmetric*, and *transitive*. A *total order* is a partial order which is also *total*, i.e., all pairs $a, b \in X$ are comparable: $a \leq b$ or $b \leq a$ holds. Generally, there are many *linear extensions* (total orders) that agree with a given partial order, and the existence of at least one is guaranteed [9]. For example, consider $X = \{a, b, c\}$ with the partial order: $a \leq b$ and $a \leq c$. Two linear extensions are $a \leq b \leq c$ and $a \leq c \leq b$. A single total order can be obtained by a simple, linear time algorithm, e.g., see Cormen *et al.* [2]. Sometimes (partial and total) orders are defined using a *strict* relation $<$ instead of $\leq$: Such a strict total order is *transitive* and *trichotomous*, i.e., exactly one of $a < b$, $a > b$, or $a = b$ holds. An example of a logic program that creates all strict total orders over a given set of events ev/1 is shown in Fig. 8: The non-stratifed rule (1) is used to generate alternative "guesses": for any $X \neq Y$, either $X < Y$ or $Y < X$; (2) infers transitive edges; and (3) rules out cases where $a < b$ and $b < a$ (including the special case of reflexivity $a < a$).[5]

Answer Set Programming (ASP). In our approach we generate many models representing the possible schedules of events, then eliminate those that contradict the axioms. Specifically, we use a Generate-and-Test pattern from answer set programming (ASP) [6], i.e., we use sets of *stable models* [3] to represent all possible worlds that are consistent with the given provenance assertions.

Let us consider the ASP program shown in Fig. 8. If it is run in combination with the facts $\texttt{ev}(a)$, $\texttt{ev}(b)$, and $\texttt{ev}(c)$, then it will generate $3! = 6$ distinct stable models that differ in the valuation of after/2: e.g., among them would be $\{\texttt{after}(a,b), \texttt{after}(b,c)\}$ and $\{\texttt{after}(b,a), \texttt{after}(a,c)\}$. Both are stable models because they "reproduce themselves", when reducing the negative literals of the ground-instantiated program according to their values in the model, then computing the unique minimal model of the reduced positive program.

[5] The empty head in (3) is shorthand for "False": if the body evaluates to true, the constraint is violated and the model is discarded.

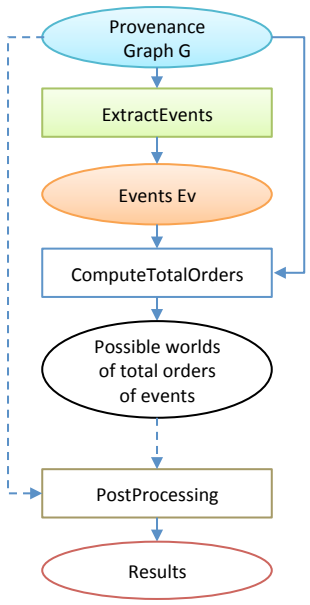

Fig. 9: Provenance Analyzer Framework with main steps ExtractEvents, ComputeTotalOrders, and PostProcessing. There are different approaches for ComputeTotalOrders. PostProcessing is optional.

4 Provenance Analyzer

4.1 Framework

Provenance Analyzer consists of three main steps: (i) ExtractEvents, (ii) ComputeTotalOrders, and (iii) PostProcessing (see Fig. 9). Given an OPM provenance graph, the first step extracts events using the logic rules in Fig. 11. Rules (1–3) check if the provenance graph is legal. These rules are based on the definition of legal OPM as stated in [4]: Rule (1) checks if there are any write conflicts (i.e., a data artifact is "precisely generated" by more than one process invocation). Rules (2) and (3) check if for a precise wasDerivedFrom edge (pDer) there is a precise used edge (pUsd) and a precise wasGeneratedBy edge (pGen). Rules (4–8) extract temporal events from the OPM graph using the relations described in Fig. 7. Rule (4) extracts a create event with identifier E for a data artifact D. The id relation maps data artifacts D and process invocations I to unique event identifiers E. Rule (5) extracts a use event for precise use of a data artifact, where useId provides the event identi-

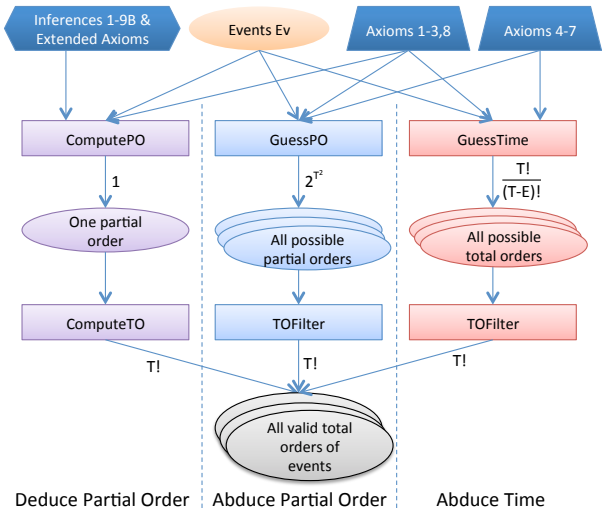

Fig. 10: Three alternative implementations of `ComputeTotalOrders` in Fig. 9: `DeducePO` computes a partial order using all axioms and inference rules from [4] and then computes all total orders. `AbducePO` computes all partial event orders and removes the ones which contradict the axioms. `AbduceTime` maps events to time points in all possible ways and removes the ones which violate the axioms.

fier E for the precise use of a data artifact D by a process invocation I with role R. Rule (6) extracts a `use` event for an imprecise use of a data artifact, where `useIdNoR` provides the event identifier E for the imprecise use of a data artifact D by a process invocation I. Rules (7) and (8) extract the `begin` and `end` events for a process invocation, respectively.

`ComputeTotalOrders` generates all total orders that are compatible with the given axioms. Various approaches can be considered for this step. Our current prototype implements three approaches, (i) `Deduce Partial Order (DeducePO)`, (ii) `Abduce Partial Order (AbducePO)`, and (iii) `Abduce Time (AbduceTime)`. In `DeducePO`, we use the axioms and inference rules provided in [4] to compute the unique partial order of the events and then compute all possible total orders. In `AbducePO`, we use abduction to generate all possible partial orders of events and implement the temporal axioms in form of constraints (*denial* rules) to filter out the partial orders that do not respect the temporal axioms, and in `AbduceTime`, we generate all possible ways (i.e., possible models) of mapping events with time points and remove the models that violate any temporal axioms. The logical architecture of these three approaches is shown in Fig. 10.

In the `PostProcessing` step, which is optional, various analysis of interests can be performed on the possible models of the events. For example, to check if a relationship between two events is in (i) some models, (iii) all models, (iii) no models, (iv) or even in a certain percentage of models.

```
% Legal OPM Integrity Constraints
(1) :- pGen(D,_,I1), pGen(D,_,I2), not I1=I2.
(2) pair(D2,R,D1) :- pGen(D2,_,I), pUsd(I,R,D1).
(3) :- pDer(D2,R,D1), not pair(D2,R,D1).

% Events extraction from OPM relations
(4) create(E,D)    :- data(D), id(D,c,E).
(5) useR(E,I,R,D)  :- pUsd(I,R,D), useId(I,R,D,E).
(6) use(E,I,D)     :- usd(I, D), useIdNoR(I,D,E).
(7) begin(E,I)     :- invoc(I), id(I,b,E).
(8) end(E,I)       :- invoc(I), id(I,e,E).
```

Fig. 11: Rules to extract events from an OPM graph.

```
% Axiom 2
(1) leq(B,C) :- pGen(A,_,P), id(P,b,B), id(A,c,C).
(2) leq(C,E) :- pGen(A,_,P), id(A,c,C), id(P,e,E).

% Axiom 8
(3) leq(U,C) :- pDer(A,R,B), pUsd(P,R,B),
                pGen(A,_,P), useR(U,P,R,B),
                id(A,c,C).
```

Fig. 12: Axiom 2 and 8 in the `DeducePO` approach.

4.2 Deduce Partial Order (DeducePO)

Our first approach follows that of [4], then computes all total orders: new edges in the OPM graph are inferred, and then, using the old and new edges along with the deductive procedure given in [4], a partial order is created (`ComputePO` in Fig. 10) that contains exactly the constraints implied by the axioms. Rules deducing constraints specified by Axioms 2[6] and Axiom 8[7] are shown in Fig. 12. Other axioms are implemented similarly.

From there, each possible total ordering that satisfies the partial order is generated with ASP.

4.3 Abduce Partial Order (AbducePO)

In our `AbducePO` approach, `GuessPO` step as shown in Fig. 10, uses abduction to generate all possible valuations of the `after(E1, E2)` predicate, representing that `E1` happens strictly after `E2`. Denial rules are applied that ensure that `after(E1, E2)` respects the temporal axioms. Rules enforcing axioms 2 and 8 are shown in Fig. 13. Other axioms are implemented in the similar way.

[6] if artifact A precisely depends on process P, then $begin(P) \leq create(A) \leq end(P)$

[7] if artifacts A and B and process P form a use-derive-generate triangle in which A is derived from B in role r, P used B in the same role, and A was precisely generated by P, then $use(P,r,B) \leq create(A)$

```
% Artifact can't be created before process has
  started.
% tc_after is a transitive closure over after
:- pGen(D,_,I), begin(E1,I), create(E2,D),
   tc_after(E1, E2).

% Artifact can't be created after process has
  ended.
:- pGen(D,_,I), create(E1,D), end(E2,I),
   tc_after(E1, E2).

% Artifact can't be created before the use
:- pGen(D2,_,I), use(E1,I,R,D1), create(E2,D2),
   pDer(D2,R,D1), tc_after(E1, E2).
```

Fig. 13: Implementation of axiom 2 and 8 in form of denial rules in the AbducePO approach.

TOFilter removes all partial orders that do not trivially correspond to total orders by forming a chain of after facts.

4.4 Abduce Time

In this approach the basic idea is to generate all possible ways (models) events can be mapped to time points. We use the Datalog rules shown in Fig. 14 for this purpose. Rules 1 and 2 map exactly one time point to an event. In some of these models two or more events may be mapped to a single time point. Rule 3 removes such models from the list of possible models.

Now, among these remaining possible models, some may not confirm to the axioms as specified in the [4] paper. We implement these axioms as denial rules and for example we showed the implementation of axioms 2 and 8 in Fig. 14. Axiom 2 is implemented using rules 4 and 5 and axiom 8 is implemented using rule 6. Rule 4 removes the models in which data artifacts are created before the process that created the artifact has started. In the similar way, rule 5 removes the modes in data artifacts are created after the process that created the artifact has ended. Rule 6 removes the models in which a data artifact is used before it is created.

As a final step using the rules 7 through 9 are used to remove models with any gap in between timepoints. All the possible models which pass this step have all the events in total order, which have no ambiguities and thus can be used for further analysis.

4.5 Post Processing

We discuss two useage of our Provenance Analyzer and we believe this can be used for many other interesting purposes.

```
% Assigning only one timepoint to an event
(1) at(T,E) :- time(T), event(E), not blk(T,E).
(2) blk(T,E) :- time(T), at(TE,E), TE != T.

% No two events happen at the same time
(3) :- at(T,E1), at(T,E2), E1 != E2.

% Axiom 2
(4) :- pGen(D,_,I), begin(E1,I), create(E2,D),
       at(T1,E1), at(T2,E2), T1>T2.
(5) :- pGen(D,_,I), create(E1,D), end(E2,I),
       at(T1,E1), at(T2,E2), T1>T2.

% Axiom 8
(6) :- pGen(D2,_,I), useR(E1,I,R,D1),
       create(E2,D2), pDer(D2,R,D1),
       at(T1,E1), at(T2,E2), T1>T2.

% Removing parsely distributed events
(7) eventsAt(T) :- at(T,_).
(8) eventsAfter(T) :- time(T), at(T2,_), T2>T.
(9) :- time(T), eventsAfter(T), not eventsAt(T).
```

Fig. 14: AbduceTime approach, in which events are mapped to time points in all possible ways (models) using rules 1,2, and 3. Axioms are implemented as denial rules to remove the illegal models. Implementation of axioms 2 and 8 are shown in rules 4 though 6. Rules 7 though 9 removes models with gaps in time points.

A common query to ask of our system (which we will call the LEQ query) is whether for events E_1 and E_2, is $T(E_1) \leq T(E_2)$ in (a) some models, (b) all models, or (c) no models. This can be efficiently solved directly on the deductively generated partial order taken in combination with whether or not the user chooses to allow simultaneous events; for details, see the appendix.

Another interesting use of Provenance Analyzer can be to understand the impact of axioms and inferences. Note that AbducePO and AbduceTime can both arrive at the exact same total orders as the deductive approach, but solely using the axioms and without the accompanying inference rules. This makes them a useful tool to experiment with altered axioms and examine effects without worrying that the rules used in the deductive approach will no longer be sound and complete.

5 Evaluation

We evaluated Provenance Analyzer from the point of views of (i) Number of models, and (ii) Execution time.

Number of Models. We introduce a new provenance graph, to facilitate the explanation of number of models

Fig.	Models	List of events extracted
3	5	$P : begin, B : create, A : create, B : use, P : end$
4	2	$B : create, P : begin, B : use, A : create, P : end$
15	16	$P : begin, B : create, B : use, C : create, C : use, A : create, P : end$

Table 1: Number of total orders and event lists

Time Points	DeducePO	AbducePO	AbduceTime
7	0.021	0.042	0.016
10	0.021	0.042	0.028
50	0.022	0.042	1.014
80	0.022	0.043	8.691
100	0.022	0.043	22.817

Table 2: Runtimes (sec) for the seven events in Fig. 15.

generation, as shown in Fig. 15, which is a complex variant of our earlier examples. Table 1 shows the number of models generated for the provenance graphs as shown in Fig. 3, Fig. 4, and Fig. 15.

We see that adding edges to an existing graph and leaving the nodes unchanged (as in going from Fig. 3 to Fig. 4) tends to reduce the number of possible worlds. Fig. 15 has got lot more models. This is due to the inclusion of additional nodes. For each additional data item there is a new time point representing its creation and one for each use, and for each new process invocation there is a start and an end timepoint. These new time points increase the number of possible orders, even if there are concomitantly more constraints on those total orders.

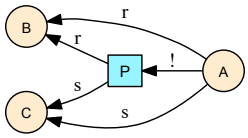

Fig. 15: Data artifact A `wasGeneratedBy` process P, which `used` both B and C

Execution Time. Given the provenance as shown in Fig. 15, the `AbduceTime` approach outperforms the more guided `DeducePO` when generating all legal total orders of events. This is only the case, though, if there are exactly as many time points defined in `time/1` as events to order. If there are more time points than the number of events, then the abductive component of `AbduceTime` will attempt to generate interpretations that use them by leaving gaps in time. These will be consolidated by `TOFilter`, which removes models containing gaps, but it takes time to generate them and determine that they must be removed later. The execution of the `AbduceTime` approach is $O(T^E)$, where T is the number of time points and E is the number of events, whereas the other two approaches do not use the `time/1` predicate at all, so they are unaffected. One might want to allow gaps to constrain at what time points events can occur. Note that `TOFilter` will eliminate these, so it should not be used in this situation.

Generating all total orders of events quickly becomes infeasible with even moderately-sized input traces. With the `DeducePO` approach, it is possible to stop after generating the single, most informed partial order, and not generate the total orders at all. Both `AbduceTime` and `AbducePO` generate all partial orders, in which some of them do not obey the definition of a partial order and are discarded. The problem is that the number of partial orders is significantly larger than the number of all total orders. This is obvious when one observes that `TOFilter` removes some partial orders and is still left with all strict total orders.

We envision future work to make the partial and total order generation steps more efficient based on the observation that if a partial order is not legal, then a partial order that is identical, except with some inputs that were incomparable made comparable, is also illegal. Use of subsumption during execution should allow the pruning of redundant branches in the search tree.

6 Related work

There has been some work to explore possible models captured by a given provenance graph. In [7], the authors have presented an encoding of PROV, the emerging W3C recommendation for a provenance data model and a successor of OPM. Provenance graphs, inference rules and constraints are realized in Datalog. Such an encoding allows intuitive, declarative queries on provenance graphs. Furthermore, the constraint-solving capabilities of the DLV Datalog engine [5], which is freely available for non-commercial use, are used to demonstrate automated validation of PROV constraints, e.g., to detect temporal inconsistencies and illegal cycles in the provenance graph.

In [1], the authors argue that data provenance can be at times ambiguous or simply inaccurate. They identified likely provenance quality issues and established crucial quality dimensions toward measuring quality.

7 Conclusion

To understand the processing history of data artifacts, it is often important to understand the order of events. Time is only an optional annotation in OPM and provenance recorders may not capture all the time information. Thus, a provenance graph may not be sufficient to infer a total order of events. [4] addresses limitations of OPM by providing a formal semantics that introduces precise and imprecise edges and a temporal semantics. This maps an OPM graph to a set of ordering constraints between time-points. Utilizing its axioms and inferences, they show that there is a deductive procedure for generating the partial order that corresponds to the graph as implied by the axioms.

We have developed a prototype, Provenance Analyzer, in which we have implemented the axioms and inference rules as specified by [4] to compute the partial order of events from the given provenance graph.

Our framework for experimenting with axioms allows for manual elucidation of the effect of alterations to the axioms and inference rules. In [4], a subset of the axioms and inference rules is shown to be sufficient to generate all correct inequalities between timepoints where neither represents the use of an artifact (so-called no-use inequalities). Our tool could be used to automate an exhaustive search through all combinations of axiom and rule subsets and test empirically which approaches might coincide for a given set of examples. Although this wouldn't constitute a formal proof of equivalence, it would still allow to test and debug variant sets of axioms and find candidate pairs of potentially equivalent (or definitely non-equivalent) rule sets.

Our tool can be downloaded from `http://code.google.com/p/provenance-analyzer/`.

Acknowledgements. Work supported in part by NSF awards DGE-0841297, OCI-0830944, and IIS-1118088.

References

[1] CHEAH, Y., AND PLALE, B. Provenance analysis: Towards quality provenance. In *E-Science (e-Science), 2012 IEEE 8th International Conference on* (2012), IEEE, pp. 1–8.

[2] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. *Introduction to Algorithms*, 2nd ed. MIT Press, 2001, pp. 549–552. Section 22.4: Topological sort.

[3] GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic programming* (1988), vol. 161.

[4] KWASNIKOWSKA, N., MOREAU, L., AND VAN DEN BUSSCHE, J. A formal account of the open provenance model. Tech. Rep. 21819, University of Southampton, December 2010.

[5] LEONE, N., PFEIFER, G., FABER, W., CALIMERI, F., DELLARMI, T., EITER, T., GOTTLOB, G., IANNI, G., IELPA, G., KOCH, C., ET AL. The dlv system. *Logics in Artificial Intelligence* (2002), 537–540.

[6] LIFSCHITZ, V. What is answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2008), pp. 1594–1597.

[7] MISSIER, P., AND BELHAJJAME, K. A prov encoding for provenance analysis using deductive rules. *IPAW* (2012).

[8] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B., SIMMHAN, Y., STEPHAN, E., AND DEN BUSSCHE, J. V. The open provenance model core specification (v1.1). *Future Generation Computer Systems 27*, 6 (2011), 743–756.

[9] SCHRÖDER, B. S. W. *Ordered sets: an introduction*. Birkhauser, 2003.

8 Appendix

LEQ on All/Some/No models. In Section 4.5, we claimed that it was possible to efficiently perform the query "For events E_1 and E_2, is $T(E_1) \leq T(E_2)$ for all, some, or no models?" (Where $T(x)$ should be read as 'the time at which x occurred') Consider the partial order deductively generated by the technique specified in [4]. If the constraint $T(E_1) \leq T(E_2)$ is derived, then $T(E_1) \leq T(E_2)$ is true for all possible total orders. If the reverse constraint is derived, then $T(E_2) \leq T(E_1)$. In terms of the original question, this means that either $T(E_1) > T(E_2)$ in all models if simultaneous events are forbade, or $T(E_1) = T(E_2)$ in some models if they are allowed. So if simultaneous events are disallowed, then $T(E_1) \leq T(E_2)$ in no models, whereas if they are allowed, then $T(E_1) \leq T(E_2)$ in some models. If no constraint exists between $T(E_1)$ and $T(E_2)$, then some total orders exist in which $T(E_1) < T(E_2)$ and some in which $T(E_1) > T(E_2)$. If this were not the case, then either $T(E_1) \leq T(E_2)$ or the reverse constraint $T(E_2) \leq T(E_1)$ would be derived.

LEQ using AbducePO and AbduceTime. `LEQ` can also be solved by post-processing of models generated by the `AbducePO` or `AbduceTime` approaches. Normally, it is infeasible to perform queries that make claims about multiple models. Some ASP systems, for example DLV, do have support for brave/cautious (the query evaluates to true if it is true in some/all models) reasoning without enumerating all possible worlds. We created a post-processor that creates an EDB from a logic program by executing that program under ASP semantics and, from answer sets $\{A_1, ..., A_n\}$, creates a fact $p(i, \bar{X})$ for each $p(\bar{X}) \in A_i$. Now `LEQ`, as well as well as more complex queries that go beyond what can be accomplished with brave/cautious reasoning, can be easily expressed in this *multiverse reification*.