Semantics of the PROV Data Model

W3C Working Group Note 30 April 2013

This version:
http://www.w3.org/TR/2013/NOTE-prov-sem-20130430/
Latest published version:
http://www.w3.org/TR/prov-sem/
Previous version:
http://www.w3.org/TR/2013/WD-prov-sem-20130312/
Editor:
James Cheney, University of Edinburgh

This document is also available in this non-normative format: PDF

Abstract

Provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness. PROV-DM is the conceptual data model that forms a basis for the W3C provenance (PROV) family of specifications.

This document presents a model-theoretic semantics for the PROV data model, viewing PROV-DM statements as atomic formulas in the sense of first-order logic, and viewing the constraints and inferences specified in PROV-CONSTRAINTS as a first-order theory. It is shown that valid PROV instances (in the sense of PROV-CONSTRAINTS) correspond to satisfiable theories. This information may be useful to researchers or users of PROV to understand the intended meaning and use of PROV for modeling information about the actual history, derivation or evolution of Web resources. It may also be useful for development of additional constraints or inferences for reasoning about PROV or integration of PROV with other Semantic Web vocabularies. It is **not** proposed as a canonical or required semantics of PROV and does not place any constraints on the use of PROV.

The PROV Document Overview describes the overall state of PROV, and should be read before other PROV documents.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

PROV Family of Documents

This document is part of the PROV family of documents, a set of documents defining various aspects that are necessary to achieve the vision of inter-operable interchange of provenance information in heterogeneous environments such as the Web. These documents are listed below. Please consult the [PROV-OVERVIEW] for a guide to reading these documents.

- PROV-OVERVIEW (Note), an overview of the PROV family of documents [PROV-OVERVIEW];
- PROV-PRIMER (Note), a primer for the PROV data model [PROV-PRIMER];
- PROV-O (Recommendation), the PROV ontology, an OWL2 ontology allowing the mapping of PROV to RDF [PROV-O];
- PROV-DM (Recommendation), the PROV data model for provenance [PROV-DM];
- PROV-N (Recommendation), a notation for provenance aimed at human consumption [PROV-N];
- PROV-CONSTRAINTS (Recommendation), a set of constraints applying to the PROV data model [PROV-CONSTRAINTS];
- PROV-XML (Note), an XML schema for the PROV data model [PROV-XML];
- PROV-AQ (Note), the mechanisms for accessing and querying provenance [PROV-AQ];
- PROV-DICTIONARY (Note) introduces a specific type of collection, consisting of key-entity pairs [PROV-DICTIONARY];
- PROV-DC (Note) provides a mapping between PROV and Dublin Core Terms [PROV-DC];
- PROV-SEM (Note), a declarative specification in terms of first-order logic of the PROV data model (this document);
- PROV-LINKS (Note) introduces a mechanism to link across bundles [PROV-LINKS].

Implementations Encouraged

The Provenance Working Group encourages implementations that make use of or extend the semantics in this document. Although work on this document by the Provenance Working Group is complete, errors may be recorded in the errata or and these may be addressed in future revisions.

Please Send Comments

This document was published by the Provenance Working Group as a Working Group Note. If you wish to make comments regarding this document, please send them to public-prov-comments@w3.org (subscribe, archives). All comments are welcome.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated,

replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Introduction

Provenance is a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing. [PROV-DM] This document complements the PROV-DM specification [PROV-DM] that defines a data model for provenance on the Web, and the PROV-CONSTRAINTS specification [PROV-CONSTRAINTS] that specifies definitions, inferences, and constraints that can be used to reason about PROV documents, or determine their validity. This document provides a formal semantics of PROV, providing a formal counterpart to the informal descriptions and motivations given elsewhere in PROV specifications.

1.1 Purpose of this document

The PROV-DM and PROV-CONSTRAINTS specifications give motivating examples that provide an intuition about the meaning of the constructs. For some concepts, such as use, start, end, generation, invalidation, and derivation, the meaning is either obvious or situation-dependent. However, during the development of PROV, the importance of additional concepts became evident, but the intuitive meaning or correct use of these concepts were not clear. For example, the $alternateOf$ and $specializationOf$ relations are used in PROV to relate different entities that present aspects of "the same thing". Over time the working group came to a consensus about these concepts and how they are to be used, but this understanding is based on abstract notions that are not explicit in PROV documents; instead, some of their properties are captured formally through certain constraints and inferences, while others are not captured in PROV specifications at all.

The purpose of this document is to present the working group's consensus view of the semantics of PROV, using tools from mathematical logic, principally model theory (though our use of these tools is lightweight). This information may be useful to users for understanding the intent behind certain features of PROV, to researchers investigating richer forms of reasoning over provenance, or to future efforts building upon PROV. It is intended as an exploration of **one** semantics for PROV, **not** a definitive specification of the **only** semantics of PROV. We provide a semantics that satisfies all of the constraints on valid PROV instances, and such that valid PROV instances correspond to satisfiable theories: every valid instance has a model, and vice versa.

The semantics has some appealing properties. Specifically, it provides a declarative counterpart to the operational definition of validity taken in PROV-CONSTRAINTS. In the specification, validity is defined via a normalization process followed by constraint checking on the normal form. This approach was adopted to keep the specification closer to implementations, although other implementations are possible and allowed. In addition to providing a semantics, this document shows that the operational presentation of PROV validity checking is equivalent to the declarative presentation adopted here. This could help justify alternative approaches to validity checking.

This document mostly considers the semantics of PROV statements and instances. PROV documents can consist of multiple instances, such as named bundles. The semantics do not cover general PROV documents, but the semantics can be used on each instance in a document separately, just as PROV-CONSTRAINTS specifies that each instance in a document is to be validated separately. So, in the rest of this document, we discuss only PROV instances and not PROV documents. The semantics of extensions of PROV, such as dictionaries [PROV-DICTIONARY] and linking across bundles [PROV-LINKS], are beyond the scope of this document.

This document has been reviewed by the Working Group, but the theorems and proofs have not been formally peer-reviewed in the sense of an academic paper. Thus, the Working Group believes this document is an appropriate starting point for future study of the semantics of PROV, but further work may be needed.

1.2 Structure of this document

- Section 2 summarizes the basic concepts from mathematical logic used in the semantics, recapitulates how PROV statements can be viewed as atomic formulas, and introduces some auxiliary formulas.
- Section 3 presents the mathematical structures used for situations that PROV statements can describe.
- Section 4 defines the semantics of PROV statements and auxiliary formulas, indicating when a given formula is satisfied in a structure.
- Section 5 presents the inferences and constraints from PROV-CONSTRAINTS as first-order formulas, and gives brief justifications for their soundness.
- Section 6 summarizes the main results relating PROV-CONSTRAINTS validation to the semantics, including soundness and a weak form of completeness: a PROV instance is valid if and only if it has a model.

1.3 Audience

This document assumes familiarity with [PROV-DM] and [PROV-CONSTRAINTS] and employs (a simplified form of) [PROV-N] notation. In particular it assumes familiarity with the concepts from logic, and the relationship between PROV statements and instances and first-order formulas and theories, respectively, presented in Section 2.5 of PROV-CONSTRAINTS.

This document may be useful to users of PROV who have a formal background and are interested in the rationale for some of the constructs of PROV; for researchers investigating extensions of PROV or alternative approaches to reasoning about PROV; or for future efforts on provenance standardization.

2. Basics

2.1 Identifiers

A lowercase symbol $x,y,\ldots$ on its own denotes an identifier. Identifiers are viewed as variables from the point of view of logic. Identifiers denote objects, and two different identifiers x and y may denote equal or different objects. We write $Identifiers$ for the set of identifiers of interest in a given situation (typically, the set of identifiers present in the PROV instance of interest).

2.2 Attributes and Values

We assume a set $Attributes$ of attribute labels and a set $Values$ of possible values of attributes. To allow for the fact that some attributes can have undefined or multiple values, we sometimes use the set $P(Values)$, that is, the set of sets of values. Thus, we can use the empty set to stand for an undefined value and $\{a,b\}$ to stand for the set of values of a two-valued attribute.

2.3 Times

We assume an ordered set $(Times, \leq)$ of time instants, where $Times \subseteq Values$ and $\leq$ is a linear order.

2.4 Atomic Formulas

The following atomic formulas correspond to the statements of PROV-DM. We assume that definitions 1-4 of PROV-CONSTRAINTS have been applied in order to expand all optional parameters; thus, we use uniform notation $r(id, a_1, \ldots, a_n)$ instead of the semicolon notation $r(id; a_1, \ldots, a_n)$.

Each parameter is either an identifier, a constant (e.g. a time or other literal value in an attribute list), or a null symbol "$-$". Placeholder symbols "$-$" can only appear in the specified arguments pl in $wasAssociatedWith$ and a, g, u in $wasDerivedFrom$, as shown in the grammar below.

$$
\begin{array}{rcl}
atomic\_formula & ::= & element\_formula \\
 & | & relation\_formula \\
 & | & auxiliary\_formula \\
element\_formula & ::= & entity(id, attrs) \\
 & | & activity(id, st, et, attrs) \\
 & | & agent(id, attrs) \\
relation\_formula & ::= & wasGeneratedBy(id, e, a, t, attrs) \\
 & | & used(id, e, a, t, attrs) \\
 & | & wasInvalidatedBy(id, e, a, t, attrs) \\
 & | & wasStartedBy(id, a_2, e, a_1, t, attrs) \\
 & | & wasEndedBy(id, a_2, e, a_1, t, attrs) \\
 & | & wasAssociatedWith(id, ag, act, pl, attrs) \\
 & | & wasAssociatedWith(id, ag, act, -, attrs) \\
 & | & wasAttributedTo(id, e, ag, attrs) \\
 & | & actedOnBehalfOf(if, ag_2, ag_1, act, attrs) \\
 & | & wasInformedBy(id, a_2, a_1, attrs) \\
 & | & wasDerivedFrom(id, e_2, e_1, act, g, u, attrs) \\
 & | & wasDerivedFrom(id, e_2, e_1, -, -, -, attrs) \\
 & | & wasInfluencedBy(id, x, y, attrs) \\
 & | & alternateOf(e_1, e_2) \\
 & | & specializationOf(e_1, e_2) \\
 & | & hadMember(c, e) \\
auxiliary\_formula & ::= & x \text{ strictlyPrecedes } y \\
 & | & x \text{ precedes } y \\
 & | & notNull(x) \\
 & | & typeOf(x, ty) \\
attrs & ::= & [attr_1 = v_1, \ldots, attr_n = v_n] \\
ty & ::= & entity \\
 & | & activity \\
 & | & agent \\
 & | & Collection \\
 & | & EmptyCollection
\end{array}
$$

Remark

We include the standard PROV collection types ($Collection$ and $EmptyCollection$) and the membership relation $hadMember$; however, we do not model dictionaries or the insertion or deletion relations in PROV-DICTIONARY [PROV-DICTIONARY], since these are not part of the PROV recommendations. If these features are incorporated into future standards, their semantics (and the soundness of the associated constraints) should be modeled. We omit the $prov$ prefixes from the $Collection$ and $EmptyCollection$ types.

As stated in the Introduction, we do not explicitly model bundles or PROV documents; however, each instance can be viewed as a set of formulas and can be modeled separately. The semantics of the standard features of PROV can be defined without talking about multiple instances; however, the $mentionOf$ relation in PROV-LINKS [PROV-LINKS] is intended to support linking across bundles. Future editions of PROV may incorporate $mentionOf$ or other cross-instance assertions, and if so this semantics should be generalized in order to provide a rationale for such an extension and to establish the soundness of constraints associated with $mentionOf$.

2.5 First-Order Formulas

We also consider the usual connectives and quantifiers of first-order logic.

$$
\begin{array}{rcl}
\phi & ::= & atomic\_formula \\
 & | & True \\
 & | & False \\
 & | & x = y \\
 & | & \neg\, \phi \\
 & | & \phi_1 \wedge \phi_2 \\
 & | & \phi_1 \vee \phi_2 \\
 & | & \phi_1 \Rightarrow \phi_2 \\
 & | & \forall x.\phi \\
 & | & \exists x.\phi
\end{array}
$$

3. Structures and Interpretations

In this section we define mathematical structures W that can be used to interpret PROV formulas and instances. A structure consists of a collection of sets, functions and relations. The components of a structure W are given in the rest of the section in *components*, highlighted in boxes.

Remark

We use the term "component" here in a different sense than in PROV-DM. Here, the components are parts of a large definition, whereas PROV-DM defines six components that group different parts of the PROV data model.

3.1 Things

Things is a set of things in the situation being modeled. Each thing has an associated set of $Events$ and attributes whose values can change over time. Different kinds of $Events$ are specified further below.

To model this, a structure W includes:

Component 1 (things)

1. a set $Things$ of things
2. a set $Events$ of events
3. a function $events{:}Things \to P(Events)$ from things to associated sets of events.
4. a function $value{:}Things \times Attributes \times Events \to P(Values)$ giving the possible values of each attribute of a $Thing$ at the instant of a given event.
5. Attributes are only defined during the events of a thing, that is, $value(T,a,evt) \neq \emptyset$ implies $evt \in events(T)$.

The range of $value$ is the set $P(Values)$, indicating that $value$ is essentially a multi-valued function that returns a set of values (possibly empty). When $value(x,a,evt)=\emptyset$, we say that attribute a is undefined for x at event evt.

Note that this description does not say what the structure of a $Thing$ is, only how it may be described in terms of its events and attribute values. A thing could be a record of fixed attribute values; it could be a bear; it could be the Royal Society; it could be a transcendental number like π. All that matters from our point of view is that we know how to map the $Thing$ to its events and attribute mapping.

The identity of a Thing is not observable through its attributes or events, so it is possible for two different $Things$ to be indistinguishable by their attribute values and events. That is, if the set of $Things=\{T_0,T_1\}$ and the attributes are specified as $value(T_0,a,evt)=value(T_1,a,evt)$ for each $evt \in Events$ and $a \in Attributes$, this does not imply that $T_0=T_1$.

$Things$ are associated with certain kinds of $Objects$ called $Entities$, defined in the next subsection. Specifically, the function $thingOf$ associates an $Entity$ to a $Thing$.

3.2 Objects

$Things$ are things in the world that have attributes that can change over time. $Things$ may not have distinguishing features that are readily observable and permanent. In PROV, we do not talk explicitly about $Things$, but instead we talk about various objects that have discrete, fixed features, and relationships among these objects. Some objects, called $Entities$, are associated with $Things$, and their fixed attributes need to match those of the associated $Thing$ during their common events. Others correspond to agents, activities, or identifiable interactions among them.

In this section, we detail the different subsets of $Objects$, and give disjointness constraints and associated functions. Generally, these constraints are necessary to validate disjointness constraints from PROV-CONSTRAINTS [PROV-CONSTRAINTS].

An *Object* is described by a set of events and attributes with fixed values. Objects encompass entities, activities, agents, and interactions (i.e., usage, generation, and other events or influence relations). To model this, a structure includes:

Component 2 (objects)

1. a set $Objects$
2. a function $events{:}Objects \to P(Events)$ from objects to associated sets of events.
3. a function $value{:}Objects \times Attributes \to P(Values)$.

Intuitively, $events(e)$ is the set of events in which e participated. The set $value(e,a)$ is the set of values of attribute a during the object's events.

As with *Things*, the range of $value$ is sets of values, making $value$ effectively a multivalued function. It is also possible to have two different objects that are indistinguishable by their attributes and associated events. Objects are not things, and the sets of $Objects$ and $Things$ are disjoint; however, certain objects, namely entities, are associated with things.

Remark

Disjointness between $Objects$ and $Things$ is not necessary but is assumed in order to avoid confusion between the different categories (time-varying $Things$ vs fixed $Objects$).

3.2.1 Entities

An *entity* is a kind of object that fixes some aspects of a thing. We assume:

Component 3 (entities)

1. a set $Entities \subseteq Objects$ of entities, disjoint from $Activities$ below.
2. a function $thingOf{:}Entities \to Things$ that associates each $Entity$ e with a $Thing$, such that $events(e) \subseteq events(thingOf(e))$ and for each

$evt \in events(e)$ and for each attribute a we have $value(e,a) \subseteq value(thingOf(e),a,evt)$.

Remark

Although both entities and things can have undefined or multiple attribute values, their meaning is slightly different: for a thing, $value(x,a,evt)=\emptyset$ means that the attribute a has no value at event evt, whereas for an entity, $value(x,a)=\emptyset$ only means that the thing associated to entity x need not have a fixed value for a during the events of x. This does not imply that $value(thingOf(e),a,evt)=\emptyset$ when $evt \in events(e)$.

Furthermore, all of the attribute values of the entity must be present in the associated thing throughout the events of the entity. For example, suppose $value(thingOf(e),a,evt)$ is $\{1\}$ at some event $evt \in events(e)$ and $value(thingOf(e),a,evt')=\{2\}$ at some other event evt'. Then $value(e,a)$ must be $\emptyset$ because there is no other set of values that is simultaneously contained in both $\{1\}$ and $\{2\}$.

Remark

In the above description of how *Entities* relate to *Things*, we require $value(e,a) \subseteq value(thingOf(e),a,evt)$ whenever $evt \in events(e)$. Intuitively, this means that if we are talking about a *Thing* indirectly by describing an *Entity*, then any attributes we ascribe to the *Entity* must also describe the associated *Thing* during their common events. Attributes of both *Entities* and *Things* are multi-valued, so there is no inconsistency in saying that an entity has two different values for some attribute. In some situations, further uniqueness constraints or range constraints could be imposed on attributes.

Only *Entities* are associated with *Things*, and this association is necessary to provide an interpretation for the *alternateOf* and *specializationOf* relations. It might also make sense to associate *Agents*, *Activities*, and *Interactions* with *Things*, or with some other structures; however, this is not necessary to model any of the current features of PROV, so in the interest of simplicity we do not do this.

3.2.1.1 Plans

We identify a specific subset of the entities called *plans*:

Component 4 (plans)

A set $Plans \subseteq Entities$ of plans.

3.2.1.2 Collections

We identify another specific subset of the entities called *collections*, with the following associated structure:

Component 5 (collections)

- A set $Collections \subseteq Entities$
- A membership function $members: Collections \rightarrow P(Entities)$ mapping each collection to its set of members.

3.2.2 Activities

An *activity* is an object corresponding to a continuing process rather than an evolving thing. We introduce:

Component 6 (activities)

1. A set $Activities \subseteq Objects$ of activities.
2. Functions $startTime: Activities \rightarrow Times$ and $endTime: Activities \rightarrow Times$ giving the start and end time of each activity.
3. Activities are disjoint from Entities: $Entities \cap Activities = \emptyset$.

3.2.3 Agents

An agent is an object that can act, by controlling, starting, ending, or participating in activities. An agent is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity. Agents can act on behalf of other agents. An agent may be a particular type of entity or activity; an agent cannot be both entity and activity because the sets of entities and activities are disjoint. We introduce:

Component 7 (agents)

A set $Agents \subseteq Objects$ of agents.

Remark

There is no requirement that every agent is either an activity or an entity.

3.2.4 Influences

We consider a set $Influences \subseteq Objects$ which has disjoint subsets $Events$ connecting entities and activities, $Associations$ between agents and activities, $Attributions$ between entities and agents, $Communications$ between pairs of activities, $Delegations$ between pairs of agents, and $Derivations$ that describe chains of generation and usage steps. These kinds of influences are discussed further below. Influences are disjoint from entities, activities and agents.

Component 8 (influences)

1. A set $Influences = Events \cup Associations \cup Communications \cup Delegations \cup Derivations \subseteq Objects$
2. The sets $Events$, $Associations$, $Communications$, $Delegations$ and $Derivations$ are all pairwise disjoint.
3. Influences are disjoint from entities, agents and activities: $Influences \cap (Entities \cup Activities \cup Agents) = \emptyset$
4. An associated function $influenced : Influences \rightarrow Objects \times Objects$ giving the source and target of each influence.

3.2.4.1 Events

An $Event$ is an instantaneous influence that relates an activity to an entity (either of which could also be an agent). Events have types including usage, generation, invalidation, starting and ending. Events are instantaneous. We introduce:

Component 9 (events)

1. A set $Events \subseteq Influences$ of events, partitioned into disjoint subsets $Starts, Ends, Generations, Usages, Invalidations$.
2. A function $time : Events \rightarrow Times$.
3. A quasi-ordering on events $\preceq \subset Events \times Events$. We write $e \prec e'$ when $e \preceq e'$ and $e' \not\preceq e$ hold.
4. A function $started : Starts \rightarrow Activities \times Entities \times Activities$, such that $started(start) = (a, e, a')$ implies $start \in events(a) \cap events(e) \cap events(a')$.
5. A function $ended : Ends \rightarrow Activities \times Entities \times Activities$, such that $ended(end) = (a, e, a')$ implies $end \in events(a) \cap events(e) \cap events(a')$.
6. A function $used : Usages \rightarrow Activities \times Entities$ such that $used(use) = (a, e)$ implies $use \in events(a) \cap events(e)$.
7. A function $generated : Generations \rightarrow Entities \times Activities$ such that $generated(gen) = (a, e)$ implies $gen \in events(a) \cap events(e)$.
8. A function $invalidated : Invalidations \rightarrow Entities \times Activities$ such that $invalidated(inv) = (a, e)$ implies $inv \in events(a) \cap events(e)$.

3.2.4.2 Associations

An $Association$ is an influence relating an agent to an activity and optional plan. To model associations, we introduce:

Component 10 (associations)

A set $Associations \subseteq Influences$ with associated function $associatedWith : Associations \rightarrow Agents \times Activities \times Plans_{\perp}$.

3.2.4.3 Attributions

An $Attribution$ is an influence relating an entity to an agent. To model attributions, we introduce:

Component 11 (attributions)

A set $Attributions \subseteq Influences$ with associated function $attributedTo : Attributions \rightarrow Entities \times Agents$.

3.2.4.4 Communications

A $Communication$ is an influence indicating exchange of information between activities. To model communications, we introduce:

Component 12 (communications)

A set $Communications \subseteq Influences$ with associated function $communicated : Communications \rightarrow Activities \times Activities$.

3.2.4.5 Delegations

A $Delegation$ is an influence relating two agents. To model delegations, we introduce:

Component 13 (delegations)

A set $Delegations \subseteq Influences$ and associated function $actedFor: Delegations \to Agents \times Agents \times Activities$

3.2.4.6 Derivations

A *Derivation* is an influence chaining one or more generation and use steps. To model derivations, we introduce an auxiliary notion of *derivation path*. These paths are of the form

$$ent_n \cdot g_n \cdot act_n \cdot u_n \cdot ent_{n-1} \cdot \ldots \cdot ent_1 \cdot g_1 \cdot act_1 \cdot u_1 \cdot ent_0$$

where the ent_i are entities, act_i are activities, g_i are generations, and u_i are usages.

Formally, we consider the (regular) language:

$$DerivationPaths = Entities \cdot (Generations \cdot Activities \cdot Usages \cdot Entities)^+$$

with the constraints that for each derivation path:

- for each substring $ent \cdot g \cdot act$ we have $generated(g) = (ent, act)$, and
- for each substring $act \cdot u \cdot ent$ we have $used(u) = (act, ent)$.

Component 14 (derivations)

A set $Derivations \subseteq Influences$ with an associated function $derivationPath: Derivations \to DerivationPaths$ linking each derivation to a derivation path.

Remark

The *derivationPath* function links each $d \in Derivations$ to a derivation path. A derivation has exactly one associated derivation path. However, if the PROV-N statement `wasDerivedFrom(e_2,e_1,-,-,-)` is asserted in an instance, there may be multiple derivation paths linking e_2 to e_1, each corresponding to a different path, identified by different derivations $d \in Derivations$.

A derivation path implies the existence of at least one chained generation and use step. However, not all such potential derivation paths are associated with derivations; there can (and in general will) be many such paths that are not associated with derivation steps. In other words, because we require derivations to be explicitly associated with derivation paths, it is not sound to infer the existence of a derivation from the existence of an alternating generation/use chain.

The reason why we need paths and not just individual derivation steps is to reflect that $wasDerivedFrom(id, e_2, e_1, -, -, -, attrs)$ formulas can represent multiple derivation steps. However, there is no way to force a derivation to take multiple steps. Any valid PROV instance has a model in which all derivation paths are one-step.

3.3 Additional axioms

Above we have stated some properties of the components. We impose some additional properties that relate several components, as follows:

Component 15 (axioms)

1. If $generated(g) = (e, a_1)$ and $used(u) = (a_2, e)$ then there exists $c \in Communications$ such that $communicated(c) = (a_2, a_1)$.
2. If $e \in Entities$ then there exist gen, inv, a, a' such that $generated(gen) = (e, a)$ and $invalidated(inv) = (e, a')$.
3. If $started(start) = (a_2, e, a_1)$ then there exists gen such that $generated(gen) = (e, a_1)$.
4. If $ended(end) = (a_2, e, a_1)$ then there exists gen such that $generated(gen) = (e, a_1)$.
5. If $d \in Derivations$ and $prov:Revision \in value(d, prov:type)$ and there exists $w \in (Generations \cup Activities \cup Uses \cup Entities)^*$ such that $derivationPath(deriv) = e_2 \cdot w \cdot e_1 \in DerivationPaths$ then $thingOf(e_1) = thingOf(e_2)$.
6. If $attributedTo(att) = (e, ag)$ then there exist gen, $assoc$ and a such that $generated(gen) = (e, a)$ and $associatedWith(assoc) = (a, ag)$.
7. If $actedFor(deleg) = (ag_2, ag_1, act)$ then there exist $assoc_1, assoc_2, pl_1, pl_2$ such that $associatedWith(assoc_1) = (ag_1, act, pl_1)$ and $associatedWith(assoc_2) = (ag_2, act, pl_2)$.
8. If $generated(id) = (e, a)$ then $influenced(id) = (e, a)$.
9. If $used(id) = (e, a)$ then $influenced(id) = (e, a)$.
10. If $communicated(id) = (a_2, a_1)$ then $influenced(id) = (a_2, a_1)$.
11. If $started(id) = (a_2, e, a_1)$ then $influenced(id) = (a_2, e)$.
12. If $ended(id) = (a_2, e, a_1)$ then $influenced(id) = (a_2, e)$.
13. If $invalidated(id) = (e, a)$ then $influenced(id) = (e, a)$.
14. If $derivationPath(id) = e_2 \cdot w \cdot e_1$ then $influenced(id) = (e_2, e_1)$.
15. If $attributedTo(id) = (e, ag)$ then $influenced(id) = (e, ag)$.
16. If $associatedWith(id) = (a, ag, pl)$ then $influenced(id) = (a, ag)$.
17. If $actedFor(id) = (ag_2, ag_1)$ then $influenced(id) = (ag_2, ag_1)$.
18. If $generated(gen) = (e, a) = generated(gen')$ then $gen = gen'$.
19. If $invalidated(inv) = (e, a) = invalidated(inv')$ then $inv = inv'$.
20. If $started(st) = (a, e_1, a')$ and $started(st') = (a, e_2, a')$ then $st = st'$.
21. If $ended(end) = (a, e_1, a')$ and $ended(end') = (a, e_2, a')$ then $end = end'$.
22. If $started(st) = (a, e, a')$ then $st \leq evt$ for all $evt \in events(a) - Invalidations$.

23. If $ended(end)=(a,e,a')$ then $evt \preceq end$ for all $evt \in events(a) - Invalidations$.
24. If $generated(gen)=(e,a)$ then $gen \preceq evt$ for all $evt \in events(e)$.
25. If $invalidated(inv)=(e,a)$ then $evt \preceq inv$ for all $evt \in events(e)$.
26. For any derivation $deriv$, with path $derivationPath(deriv)=w$, if $e_2 \cdot g \cdot a \cdot u \cdot e_1$ is a substring of w where $e_1,e_2 \in Entities$, $g \in Generations$, $u \in Usages$ and $a \in Activities$ then $u \preceq g$.
27. For any derivation $deriv$, with path $derivationPath(deriv)=e_2 \cdot w \cdot e_1$, if $generated(gen_1)=(e_1,a_1)$ and $generated(gen_2)=(e_2,a_2)$ then $gen_1 \prec gen_2$.
28. If $associatedWith(assoc)=(a,ag,pl)$ and $started(start)=(a,e_1,a_1)$ and $invalidated(inv)=(ag,a_2)$ then $start \preceq inv$.
29. If $associatedWith(assoc)=(a,ag,pl)$ and $generated(gen)=(ag,a_1)$ and $ended(end)=(a,e_2,a_2)$ then $gen \preceq end$.
30. If $associatedWith(assoc)=(a,ag,pl)$ and $started(start)=(a,e_1,a_1)$ and $ended(end)=(ag,e_2,a_2)$ then $start \preceq end$.
31. If $associatedWith(assoc)=(a,ag,pl)$ and $started(start)=(ag,e_1,a_1)$ and $ended(end)=(a,e_2,a_2)$ then $start \preceq end$.
32. If $attributedTo(attrib)=(e,ag)$ and $generated(gen_1)=(ag_1,a_1)$ and $generated(gen_2)=(e,a_2)$ then $gen_1 \preceq gen_2$.
33. If $attributedTo(attrib)=(e,ag)$ and $started(start)=(ag_1,e_1,a_1)$ and $generated(gen)=(e,a_2)$ then $start \preceq gen$.
34. If $actedFor(deleg)=(ag_2,ag_1,a)$ and $generated(gen)=(ag_1,a_1)$ and $invalidated(inv)=(ag_2,a_2)$ then $gen \preceq inv$.
35. If $actedFor(deleg)=(ag_2,ag_1,a)$ and $started(start)=(ag_1,e_1,a_1)$ and $ended(end)=(ag_2,e_2,a_2)$ then $start \preceq end$.
36. If $e \in Entity$ and $prov{:}emptyCollection \in value(e,prov{:}type)$ then $e \in Collections$ and $members(e)=\emptyset$.

These properties are called *axioms*, and they are needed to ensure that the PROV-CONSTRAINTS inferences and constraints hold in all structures.

Remark

Axioms 22 and 23 do not require that invalidation events originating from an activity follow the activity's start event(s) or precede its end event(s). This is because there is no such constraint in PROV-CONSTRAINTS. Arguably, there should be a constraint analogous to Constraint 34 that specifies that any invalidation event in which an activity participates must follow the activity's start event(s) and precede its end event(s).

Here, we exempt invalidations from axioms 22 and 23 in order to simplify the proof of weak completeness.

3.4 Putting it all together

A *PROV structure* W is a collection of sets, functions, and relations containing all of the above described components and satisfying all of the associated properties and axioms. If we need to talk about the objects or relations of more than one structure then we may write $W_1.Objects$, $W_1.Things$, etc.; otherwise, to decrease notational clutter, when we consider a fixed structure then the names of the sets, relations and functions above refer to the components of that model.

Some features of PROV structures are relatively obvious or routine, corresponding directly to features of PROV and associated inferences. For example, the functions $used, generated, invalidated, started, ended$ mapping events to their associated entities or activities, and $communicated, associatedWith, attributedTo, actedFor$ associating other types of influences with appropriate data.

On the other hand, some features are more distinctive, and represent areas where formal modeling has been used to guide the development of PROV. Derivation paths are one such distinctive feature; they correspond to an intuition that derivations may describe one or multiple generation-use steps leading from one entity to another. Another distinctive feature is the use of $Things$, which correspond to changing, real-world things, as opposed to $Entities$, which correspond to limited views or perspectives on $Things$, with some fixed aspects. The semantic structures of $Things$ and $Entities$ provide a foundation for the $alternateOf$ and $specializationOf$ relations.

3.5 Interpretations

We need to link identifiers to the objects they denote. We do this using a function which we shall call an *interpretation*. An interpretation is a function $\rho : Identifiers \to Objects$ describing which object is the target of each identifier. The mapping from identifiers to objects may **not** change over time; only $Objects$ can be denoted by $Identifiers$.

4. Semantics

In what follows, let W be a fixed structure with the associated sets and relations discussed in the previous section, and let ρ be an interpretation of identifiers as objects in W. The annotations [WF] refer to well-formedness constraints that correspond to typing constraints.

4.1 Satisfaction

Consider a formula ϕ, a structure W and an interpretation ρ. We define notation $W,\rho \models \phi$ which means that ϕ is satisfied in W,ρ. For atomic formulas, the definition of the satisfaction relation is given in the next few subsections. We give the standard definition of the semantics of the other formulas:

Semantics 16 (first-order-logic-semantics)

1. $W,\rho \models True$ always holds.
2. $W,\rho \models False$ never holds.
3. $W,\rho \models x=y$ holds if and only if $\rho(x)=\rho(y)$.
4. $W,\rho \models \neg\phi$ holds if and only if $W,\rho \models \phi$ does not hold.
5. $W,\rho \models \phi \wedge \psi$ holds if and only if $W,\rho \models \phi$ and $W,\rho \models \psi$.
6. $W,\rho \models \phi \vee \psi$ holds if either $W,\rho \models \phi$ or $W,\rho \models \psi$.
7. $W,\rho \models \phi \Rightarrow \psi$ holds if $W,\rho \models \phi$ implies $W,\rho \models \psi$.
8. $W,\rho \models \exists x.\phi$ holds if there exists some $obj \in Objects$ such that $W,\rho[x:=obj] \models \phi$.
9. $W,\rho \models \forall x.\phi$ holds if there for every $obj \in Objects$ we have $W,\rho[x:=obj] \models \phi$.

Remark

In the semantics above, note that the domain of quantification is the set of $Objects$; that is, quantifiers range over entities, activities, agents, or influences (which are in turn further subdivided into types of influences). $Things$ and relations cannot be referenced directly by identifiers.

Remark

A PROV instance I consists of a set of statements, each of which can be translated to an atomic formula following the definitional rules in PROV-CONSTRAINTS, possibly by introducing fresh existential variables. Thus, we can view an instance I as a set of atomic formulas $\{\phi_1,\ldots,\phi_n\}$, or equivalently a single formula $\exists x_1,\ldots,x_k.\ \phi_1\wedge\cdots\wedge\phi_n$, where $x_1,\ldots,x_k$ are the existential variables of I.

4.2 Attribute matching

We say that an object obj matches attributes $[attr_1{=}val_1,\ldots]$ in structure W provided: for each attribute $attr_i$, we have $val_i{\in}W.value(obj,attr_i)$. This is sometimes abbreviated as: $match(W,obj,attrs)$.

4.3 Semantics of Element Formulas

4.3.1 Entity

An entity formula is of the form $entity(id,attrs)$ where id denotes an entity.

Entity formulas $entity(id,attrs)$ can be interpreted as follows:

Semantics 17 (entity-semantics)

$W,\rho{\models}entity(id,attrs)$ holds if and only if:

1. [WF] id denotes an entity $ent{=}\rho(id){\in}Entities$.
2. the attributes match: $match(W,ent,attrs)$.

Remark

Not all of the attributes of an entity object are required to be present in an entity formula about that object. For example, the following formulas all hold if x denotes an entity e such that $value(e,a){=}\{4,5\},value(e,b){=}\{6\}$ hold:

```
entity(x,[])
entity(x,[a=5])
entity(x,[a=4,a=5])
entity(x,[a=4,b=6])
```

Note that PROV-CONSTRAINTS normalization will merge these formulas to a single one:

```
entity(x,[a=4,a=5,b=6])
```

4.3.2 Activity

An activity formula is of the form $activity(id,st,et,attrs)$ where id is a identifier referring to the activity, st is a start time and et is an end time, and $attrs$ are the attributes of activity id.

Semantics 18 (activity-semantics)

$W,\rho{\models}activity(id,st,et,attrs)$ holds if and only if:

1. [WF] The identifier id maps to an activity $act{=}\rho(id){\in}Activities$.
2. $\rho(st){\in}Times$ is the activity's start time, that is: $startTime(act){=}\rho(st)$.
3. $\rho(et)$ is the activity's end time, that is: $endTime(act){=}\rho(et)$.
4. There exists $start,e,a$ such that $started(start){=}(act,e,a)$, and for all such start events $startTime(act) = time(start).
5. There exists end,e',a' such that $ended(end){=}(act,e',a')$, and for all such end events $endTime(act){=}time(end)$.
6. The attributes match: $match(W,act,attrs)$.

Remark

The above definition is complicated for two reasons. First, we need to ensure that every activity has a start and end event. Second, when an $activity$ formula is asserted, we need to make sure all of the associated start and end event times match.

4.3.3 Agent

An agent formula is of the form $agent(id,attrs)$ where id denotes the agent and $attrs$ describes additional attributes.

Semantics 19 (agent-semantics)

$W,\rho \models agent(id,attrs)$ holds if and only if:

1. [WF] id denotes an agent $ag=\rho(id) \in Agents$.
2. The attributes match: $match(W,ag,attrs)$.

4.4 Semantics of Relations

4.4.1 Generation

The generation formula is of the form $wasGeneratedBy(id,e,a,t,attrs)$ where id is an event identifier, e is an entity identifier, a is an activity identifier, $attrs$ is a set of attribute-value pairs, and t is a time.

Semantics 20 (generation-semantics)

$W,\rho \models wasGeneratedBy(id,e,a,t,attrs)$ holds if and only if:

1. [WF] The identifier id denotes a generation event $evt=\rho(id) \in Generations$.
2. [WF] The identifier e denotes an entity $ent=\rho(e) \in Entities$.
3. [WF] The identifier a denotes an activity $act=\rho(a) \in Activities$.
4. The event evt occurred at time $\rho(t) \in Times$, i.e. $time(evt)=\rho(t)$.
5. The activity act generated ent via evt, i.e. $generated(evt)=(ent,act)$.
6. The attribute values match: $match(W,evt,attrs)$.

4.4.2 Use

The use formula is of the form $used(id,a,e,t,attrs)$ where id denotes an event, a is an activity identifier, e is an object identifier, $attrs$ is a set of attribute-value pairs, and t is a time.

Semantics 21 (usage-semantics)

$W,\rho \models used(id,a,e,t,attrs)$ holds if and only if:

1. [WF] The identifier id denotes a usage event $evt=\rho(id) \in Usages$.
2. [WF] The identifier a denotes an activity $act=\rho(id) \in Activities$.
3. [WF] The identifier e denotes an entity $ent=\rho(e) \in Entities$.
4. The event evt occurred at time $\rho(t) \in Times$, i.e. $time(evt)=\rho(t)$.
5. The activity act used obj via evt, i.e. $used(evt)=(act,ent)$.
6. The attribute values match: $match(W,evt,attrs)$.

4.4.3 Invalidation

The invalidation formula is of the form $wasInvalidatedBy(id,e,a,t,attrs)$ where id is an event identifier, e is an entity identifier, a is an activity identifier, $attrs$ is a set of attribute-value pairs, and t is a time.

Semantics 22 (invalidation-semantics)

An invalidation formula $W,\rho \models wasInvalidatedBy(id,e,a,t,attrs)$ holds if and only if:

1. [WF] The identifier id denotes an invalidation event $evt=\rho(id) \in Invalidations$.
2. [WF] The identifier e denotes an entity $ent=\rho(e) \in Entities$.
3. [WF] The identifier a denotes an activity $act=\rho(a) \in Activities$.
4. The event evt occurred at time $\rho(t) \in Times$, i.e. $time(evt)=\rho(t)$.
5. The activity act invalidated ent via evt, i.e. $invalidated(evt)=(ent,act)$.
6. The attribute values match: $match(W,evt,attrs)$.

4.4.4 Association

An association formula has the form $wasAssociatedWith(id,a,ag,pl,attrs)$.

Semantics 23 (association-plan-semantics)

$W,\rho \models wasAssociatedWith(id,a,ag,pl,attrs)$ holds if and only if:

1. [WF] $assoc$ denotes an association $assoc=\rho(id) \in Associations$.
2. [WF] a denotes an activity $act=\rho(a) \in Activities$.
3. [WF] ag denotes an agent $agent=\rho(ag) \in Agents$.

4. [WF] pl denotes a plan $plan=\rho(pl)\in Plans$.
5. The association associates the agent with the activity and plan, i.e. $associatedWith(assoc)=(agent,act,plan)$.
6. The attributes match: $match(W,assoc,attrs)$.

Semantics 24 (assocation-semantics)

$W,\rho\models wasAssociatedWith(id,a,ag,-,attrs)$ holds if and only if:

1. [WF] $assoc$ denotes an association $assoc=\rho(id)\in Associations$.
2. [WF] a denotes an activity $act=\rho(a)\in Activities$.
3. [WF] ag denotes an agent $agent=\rho(ag)\in Agents$.
4. The association associates the agent with the activity and no plan, i.e. $associatedWith(assoc)=(agent,act,\perp)$.
5. The attributes match: $match(W,assoc,attrs)$.

4.4.5 Start

A start formula $wasStartedBy(id,a_2,e,a_1,t,attrs)$ is interpreted as follows:

Semantics 25 (start-semantics)

$W,\rho\models wasStartedBy(id,a_2,e,a_1,t,attrs)$ holds if and only if:

1. [WF] id denotes a start event $evt=\rho(id)\in Starts$.
2. [WF] a_2 denotes an activity $act_2=\rho(a_2)\in Activities$.
3. [WF] e denotes an entity $ent=\rho(e)\in Entities$.
4. [WF] a_1 denotes an activity $act_1=\rho(a_1)\in Activities$.
5. The event happened at time t, that is, $\rho(t)==time(evt)$.
6. The activity act_1 started act_2 via entity ent: that is, $started(evt)=(act_2,ent,act_1)$.
7. The attributes match: $match(W,evt,attrs)$.

4.4.6 End

An activity end formula $wasEndedBy(id,a_2,e,a_1,t,attrs)$ is interpreted as follows:

Semantics 26 (end-semantics)

$W,\rho\models wasEndedBy(id,a_2,e,a_1,t,attrs)$ holds if and only if:

1. [WF] id denotes an end event $evt=\rho(id)\in Ends$.
2. [WF] a_2 denotes an activity $act_2=\rho(a_2)\in Activities$.
3. [WF] e denotes an entity $ent=\rho(e)\in Entities$.
4. [WF] a_1 denotes an activity $act_1=\rho(a_1)\in Activities$.
5. The event happened at the end of act_2, that is, $\rho(t)=endTime(act_2)=time(evt)$.
6. The activity act_1 ended act_2 via entity ent: that is, $ended(evt)=(act_2,ent,act_1)$.
7. The attributes match: $match(W,evt,attrs)$.

4.4.7 Attribution

An attribution formula $wasAttributedTo(id,e,ag,attrs)$ is interpreted as follows:

Semantics 27 (attribution-semantics)

$W,\rho\models wasAttributedTo(id,e,ag,attrs)$ holds if and only if:

1. [WF] id denotes an association $assoc=\rho(id)\in Associations$.
2. [WF] e denotes an entity $ent=\rho(e)\in Entities$.
3. [WF] ag denotes an agent $agent=\rho(ag)\in Agents$.
4. The entity was attributed to the agent, i.e. $attributedTo(assoc)=(ent,agent)$.
5. The attributes match: $match(W,assoc,attrs)$.

4.4.8 Communication

A communication formula $wasInformedBy(id,a_2,a_2,attrs)$ is interpreted as follows:

Semantics 28 (communication-semantics)

$W,\rho\models wasInformedBy(id,a_2,a_1,attrs)$ holds if and only if:

1. [WF] id denotes a communication $comm=\rho(id)\in Communications$.
2. [WF] a_1,a_2 denote activities $act_1=\rho(a_1)\in Activities, act_2=\rho(a_2)\in Activities$.
3. There exist gen,use,ent such that $communicated(comm)=(act_2,act_1)$ and $generated(gen)=(ent,act_1)$ and $used(use)=(act_2,ent)$.
4. The attributes match: $match(W,comm,attrs)$.

4.4.9 Delegation

The $actedOnBehalfOf(id,ag_2,ag_1,act,attrs)$ relation is interpreted as follows:

Semantics 29 (delegation-semantics)

$W,\rho\models actedOnBehalfOf(id,ag_2,ag_1,act,attrs)$ holds if and only if:

1. [WF] id denotes a delegation $deleg=\rho(id)\in Delegations$.
2. [WF] a denotes an activity $act=\rho(a)\in Activities$.
3. [WF] ag_1,ag_2 denote agents $agent_1=\rho(ag_1), agent_2=\rho(ag_2)\in Agents$.
4. The agent $agent_2$ acted for the agent $agent_1$ with respect to the activity act, i.e. $actedFor(deleg)=(agent_2,agent_1,act)$.
5. The attributes match: $match(W,deleg,attrs)$.

4.4.10 Derivation

Derivation formulas can be of one of two forms:

- $wasDerivedFrom(id,e_2,e_1,a,g,u,attrs)$, which specifies an activity, generation and usage event. For convenience we call this a ***precise derivation***.
- and $wasDerivedFrom(id,e_2,e_1,-,-,-,attrs)$, which does not specify an activity, generation and usage event. For convenience we call this an ***imprecise derivation***.

4.4.10.1 Precise

A precise derivation formula has the form $wasDerivedFrom(id,e_2,e_1,a,g,u,attrs)$.

Semantics 30 (derivation-precise-semantics)

$W,\rho\models wasDerivedFrom(id,e_2,e_1,act,g,u,attrs)$ holds if and only if:

1. [WF] id denotes a derivation $deriv=\rho(id)\in Derivations$.
2. [WF] e_1,e_2 denote entities $ent_1=\rho(e_1), ent_2=\rho(e_2)\in Entities$.
3. [WF] a denotes an activity $act=\rho(a)\in Activities$.
4. [WF] g denotes a generation event $gen=\rho(g)\in Generations$.
5. [WF] u denotes a use event $use=\rho(u)\in Usages$.
6. The derivation denotes a one-step derivation path linking the entities via the activity, generation and use: $derivationPath(deriv)=ent_2\cdot gen\cdot act\cdot use\cdot ent_1$.
7. The attribute values match: $match(W,deriv,attrs)$.

4.4.10.2 Imprecise

An imprecise derivation formula has the form $wasDerivedFrom(id,e_2,e_1,-,-,-,attrs)$.

Semantics 31 (derivation-imprecise-semantics)

$W,\rho\models wasDerivedFrom(id,e_2,e_1,-,-,-,attrs)$ holds if and only if:

1. [WF] id denotes a derivation $deriv=\rho(id)\in Derivations$.
2. [WF] e_1,e_2 denote entities $ent_1=\rho(e_1), ent_2=\rho(e_2)\in Entities$.
3. $derivationPath(deriv)=ent_2\cdot w\cdot ent_1$ for some w.
4. The attribute values match: $match(W,deriv,attrs)$.

4.4.11 Influence

Semantics 32 (influence-semantics)

$W,\rho\models wasInfluencedBy(id,o_2,o_1,attrs)$ holds if and only if at least one of the following hold:

1. [WF] id denotes an influence $inf=\rho(id)\in Influences$.
2. [WF] o_1 and o_2 denote objects $obj_1=\rho(o_1)\in Objects$ and $obj_2=\rho(o_2)\in Objects$.
3. The influence inf links o_2 with o_1; that is, $influenced(inf)=(o_2,o_1)$.

4. The attribute values match: $match(W,deriv,attrs)$.

4.4.12 Specialization

The $specializationOf(e_1,e_2)$ relation indicates when one entity formula presents more specific aspects of another.

Semantics 33 (specialization-semantics)

$W,\rho \models specializationOf(e_1,e_2)$ holds if and only if:

1. [WF] Both e_1 and e_2 are entity identifiers, denoting entities $ent_1=\rho(e_1)\in Entities$ and $ent_2=\rho(e_2)\in Entities$.
2. The two entities present aspects of the same thing, that is, $thingOf(ent_1)=thingOf(ent_2)$.
3. The events of ent_1 are contained in those of ent_2, i.e. $events(ent_1)\subseteq events(ent_2)$.
4. For each attribute $attr$ we have $value(ent_1,attr)\supseteq value(ent_2,attr)$.
5. At least one of these inclusions is strict: that is, either $events(ent_1)\subsetneq events(ent_2)$ or for some $attr$ we have $value(ent_1,attr)\supsetneq value(ent_2,attr)$.

Remark

The second criterion says that the two Entities present (possibly different) aspects of the same Thing. Note that the third criterion allows ent_1 and ent_2 to have the same events (or $events(ent_2)$ can be larger). The last criterion allows ent_1 to have more defined attributes than ent_2, but they must include the attributes defined by ent_2. Two different entities that have the same attributes can also be related by specialization. The fifth criterion (indirectly) ensures that specialization is irreflexive.

4.4.13 Alternate

The $alternateOf$ relation indicates when two entity formulas present (possibly different) aspects of the same thing. The two entities may or may not overlap in time.

Semantics 34 (alternate-semantics)

$W,\rho \models alternateOf(e_1,e_2)$ holds if and only if:

1. [WF] Both e_1 and e_2 are entity identifiers, denoting $ent_1=\rho(e_1)$ and $ent_2=\rho(e_2)$.
2. The two objects refer to the same underlying Thing: $thingOf(ent_1)=thingOf(ent_2)$

4.4.14 Membership

The $hadMember$ relation relates a collection to an element of the collection.

Semantics 35 (membership-semantics)

$W,\rho \models hadMember(c,e)$ holds if and only if:

1. [WF] Both e_1 and e_2 are entity identifiers, denoting $coll=\rho(c)\in Collections$ and $ent=\rho(e)\in Entities$.
2. The entity ent is a member of the collection $coll$: that is, $ent\in members(coll)$.

4.5 Semantics of Auxiliary Formulas

In this section, we define the semantics of additional formulas concerning ordering, null values, and typing. These are used in the logical versions of constraints.

4.5.1 Precedes and Strictly Precedes

The precedes relation x precedes y holds between two events, one taking place before (or simultaneously with) another. Its meaning is defined in terms of the quasiordering on events specified by $\preceq$. The semantics of strictly precedes (x strictlyPrecedes y) is similar, only x must take place strictly before y. It is interpreted as $\prec$, which we recall is defined from $\preceq$ as $x\prec y \iff x\preceq y$ and $y\not\preceq x$.

Semantics 36 (precedes-semantics)

1. $W,\rho \models x$ precedes y holds if and only if $\rho(x),\rho(y)\in Events$ and $\rho(x)\preceq\rho(y)$.
2. $W,\rho \models x$ strictlyPrecedes y holds if and only if $\rho(x),\rho(y)\in Events$ and $\rho(x)\prec\rho(y)$.

Remark

The ordering of time values associated to events is unrelated to the event ordering. For example:

```
entity(e)
activity(a1)
activity(a2)
wasGeneratedBy(gen1; e, a1, 2011-11-16T16:05:00)
wasGeneratedBy(gen2; e, a2, 2012-11-16T16:05:00) //different date
```

This instance is valid, and must satisfy precedence constraints gen_1 precedes gen_2 and gen_2 precedes gen_1, but this does not imply anything about the relative orderings of the associated times, or vice versa.

4.5.2 notNull

The $notNull(x)$ formula is used to specify that a value may not be the null value $\perp$. The symbol "$-$" always denotes the null value (i.e. $\rho(-)=\perp$).

Semantics 37 (notNull-semantics)

$W,\rho \models notNull(e)$ holds if and only if $\rho(e) \neq \perp$.

4.5.3 typeOf

The typing formula $typeOf(x,t)$ constrains the type of the value of x.

Semantics 38 (typeOf-semantics)

1. $W,\rho \models typeOf(e,entity)$ holds if and only if $\rho(e) \in Entities$.
2. $W,\rho \models typeOf(a,activity)$ holds if and only if $\rho(a) \in Activities$.
3. $W,\rho \models typeOf(ag,agent)$ holds if and only if $\rho(ag) \in Agents$.
4. $W,\rho \models typeOf(c,Collection)$ holds if and only if $\rho(c) \in Collections$.
5. $W,\rho \models typeOf(c,EmptyCollection)$ holds if and only if $\rho(c) \in Collections$ and $members(\rho(c)=\varnothing$.

5. Inferences and Constraints

In this section we restate all of the inferences and constraints of PROV-CONSTRAINTS in terms of first-order logic. For each, we give a proof sketch showing why the inference or constraint is sound for reasoning about the semantics. We exclude the definitional rules in PROV-CONSTRAINTS because they are only needed for expanding the abbreviated forms of PROV-N statements to the logical formulas used here.

5.1 Inferences

Inference 5 (communication-generation-use-inference)

$\forall id,a_2,a_1,attrs.$
$wasInformedBy(id,a_2,a_1,attrs)$
$\Rightarrow \exists e,gen,t_1,use,t_2.\ wasGeneratedBy(gen,e,a_1,t_1,[]) \wedge used(use,a_2,e,t_2,[])$

Proof

This follows immediately from the semantics of $wasInformedBy$.

QED

Inference 6 (generation-use-communication-inference)

$\forall gen,e,a_1,t_1,attrs_1,use,a_2,t_2,attrs_2.$
$wasGeneratedBy(gen,e,a_1,t_1,attrs_1) \wedge used(use,a_2,e,t_2,attrs_2)$
$\Rightarrow \exists id.\ wasInformedBy(id,a_2,a_1,[])$

Proof

This follows from the semantics of $wasInformedBy$ and Axiom 1.

QED

Inference 7 (entity-generation-invalidation-inference)

$\forall e,attrs.$
$entity(e,attrs)$
$\Rightarrow \exists gen,a_1,t_1,inv,a_2,t_2.\ wasGeneratedBy(gen,e,a_1,t_1,[]) \wedge wasInvalidatedBy(inv,e,a_2,t_2,[])$

Proof

This follows from Axiom 2, which requires that generation and invalidation events exist for each entity.

QED

Inference 8 (activity-start-end-inference)
$\forall a,t_1,t_2,attrs.$
$activity(a,t_1,t_2,attrs)$
$\Rightarrow \exists start,e_1,a_1,end,a_2,e_2.\ wasStartedBy(start,a,e_1,a_1,t_1,[]) \wedge wasEndedBy(end,a,e_2,a_2,t_2,[])$

Proof

This follows from the semantics of activity formulas, specifically the requirement that start and end events exist for the activity.

QED

Inference 9 (wasStartedBy-inference)
$\forall id,a,e_1,a_1,t,attrs.$
$wasStartedBy(id,a,e_1,a_1,t,attrs)$
$\Rightarrow \exists gen,t_1.\ wasGeneratedBy(gen,e_1,a_1,t_1,[])$

Proof

This follows from Axiom 3.

QED

Inference 10 (wasEndedBy-inference)
$\forall id,a,e_1,a_1,t,attrs.$
$wasEndedBy(id,a,e_1,a_1,t,attrs)$
$\Rightarrow \exists gen,t_1.\ wasGeneratedBy(gen,e_1,a_1,t_1,[])$

Proof

This follows from Axiom 4.

QED

Inference 11 (derivation-generation-use-inference)
$\forall id,e_2,e_1,a,gen_2,use_1,attrs.$
$notNull(a) \wedge notNull(gen_2) \wedge notNull(use_1) \wedge wasDerivedFrom(id,e_2,e_1,a,gen_2,use_1,attrs)$
$\Rightarrow \exists t_1,t_2.\ used(use_1,a,e_1,t_1,[]) \wedge wasGeneratedBy(gen_2,e_2,a,t_2,[])$

Proof

This follows from the semantics of precise derivation steps.

QED

Inference 12 (revision-is-alternate-inference)
$\forall id,e_1,e_2,a,g,u.$
$wasDerivedFrom(id,e_2,e_1,a,g,u,[prov{:}type{=}prov{:}Revision])$
$\Rightarrow alternateOf(e_2,e_1)$

Proof

This follows from the semantics of derivation steps (precise or imprecise) and Axiom 5.

QED

Inference 13 (attribution-inference)
$\forall att,e,ag,attrs.$
$wasAttributedTo(att,e,ag,attrs)$
$\Rightarrow \exists a,t,gen,assoc,pl.\ wasGeneratedBy(gen,e,a,t,[]) \wedge wasAssociatedWith(assoc,a,ag,pl,[])$

Proof

This follows from the semantics of generation, association, and attribution, by Axiom 6.

QED

Inference 14 (delegation-inference)
$\forall id,ag_1,ag_2,a,attrs.$
$actedOnBehalfOf(id,ag_1,ag_2,a,attrs)$
$\Rightarrow \exists id_1,pl_1,id_2,pl_2.\ wasAssociatedWith(id_1,a,ag_1,pl_1,[]) \wedge wasAssociatedWith(id_2,a,ag_2,pl_2,[])$

Proof

This follows from the semantics of association and delegation, by Axiom 7.

QED

Inference 15 (influence-inference)

1. $\forall id,e,a,t,attrs.$ $wasGeneratedBy(id,e,a,t,attrs) \Rightarrow wasInfluencedBy(id,e,a,attrs)$
2. $\forall id,a,e,t,attrs.$ $used(id,a,e,t,attrs) \Rightarrow wasInfluencedBy(id,a,e,attrs)$
3. $\forall id,a_2,a_1,attrs.$ $wasInformedBy(id,a_2,a_1,attrs) \Rightarrow wasInfluencedBy(id,a_2,a_1,attrs)$
4. $\forall id,a_2,e,a_1,t,attrs.$ $wasStartedBy(id,a_2,e,a_1,t,attrs) \Rightarrow wasInfluencedBy(id,a_2,e,attrs)$
5. $\forall id,a_2,e,a_1,t,attrs.$ $wasEndedBy(id,a_2,e,a_1,t,attrs) \Rightarrow wasInfluencedBy(id,a_2,e,attrs)$
6. $\forall id,e,a,t,attrs.$ $wasInvalidatedBy(id,e,a,t,attrs) \Rightarrow wasInfluencedBy(id,e,a,attrs)$
7. $\forall id,e_2,e_1,a,g,u,attrs.$ $wasDerivedFrom(id,e_2,e_1,a,g,u,attrs) \Rightarrow wasInfluencedBy(id,e_2,e_1,attrs)$
8. $\forall id,e,ag,attrs.$ $wasAttributedTo(id,e,ag,attrs) \Rightarrow wasInfluencedBy(id,e,ag,attrs)$
9. $\forall id,a,ag,pl,attrs.$ $wasAssociatedWith(id,a,ag,pl,attrs) \Rightarrow wasInfluencedBy(id,a,ag,attrs)$
10. $\forall id,ag_2,ag_1,a,attrs.$ $actedOnBehalfOf(id,ag_2,ag_1,a,attrs) \Rightarrow wasInfluencedBy(id,ag_2,ag_1,attrs)$

Proof

This follows via Axioms 8 through 17.

QED

Inference 16 (alternate-reflexive)
$\forall e.$
$entity(e)$
$\Rightarrow alternateOf(e,e)$

Proof

Suppose $ent=\rho(e)$. Clearly $ent \in Entities$ and $thingOf(ent)=thingOf(ent)$, so $W,\rho \models alternateOf(e,e)$.

QED

Inference 17 (alternate-transitive)
$\forall e_1,e_2,e_3.$
$alternateOf(e_1,e_2) \wedge alternateOf(e_2,e_3)$
$\Rightarrow alternateOf(e_1,e_3)$

Proof

Suppose $ent_1=\rho(e_1)$ and $ent_2=\rho(e_2)$ and $ent_3=\rho(e_3)$. Then by assumption ent_1, ent_2, and ent_3 are in $Entities$ and $thingOf(e_1)=thingOf(e_2)$ and $thingOf(e_2)=thingOf(e_3)$, so $thingOf(e_1)=thingOf(e_3)$, as required to conclude $W,\rho \models alternateOf(e_1,e_3)$.

QED

Inference 18 (alternate-symmetric)
$\forall e_1,e_2.$
$alternateOf(e_1,e_2)$
$\Rightarrow alternateOf(e_2,e_1)$

Proof

Suppose $ent_1=\rho(e_1)$ and $ent_2=\rho(e_2)$. Then by assumption both ent_1 and ent_2 are in $Entities$ and $thingOf(e_1)=thingOf(e_2)$, as required to conclude $W,\rho \models alternateOf(e_2,e_1)$.

QED

Inference 19 (specialization-transitive)
$\forall e_1,e_2,e_3.$
$specializationOf(e_1,e_2) \wedge specializationOf(e_2,e_3)$
$\Rightarrow specializationOf(e_1,e_3)$

Proof

Suppose the conditions for specialization hold of ent_1 and ent_2 and for ent_2 and ent_3, where $ent_1=\rho(e_1)$ and $ent_2=\rho(e_2)$ and $ent_3=\rho(e_3)$. Then $events(e_1) \subseteq events(e_2) \subseteq events(e_3)$. Moreover, $value(obj_2,attr) \supseteq value(obj_3,attr)$, and similarly $value(obj_1,attr) \supseteq value(obj_2,attr)$ so $value(obj_1,attr) \supseteq value(obj_3,attr)$. Finally, at least one of the inclusions between obj_1 and obj_2 is strict, so the same is the case for obj_1 and obj_3.

QED

Inference 20 (specialization-alternate-inference)
$\forall e_1,e_2.$
$specializationOf(e_1,e_2)$
$\Rightarrow alternateOf(e_1,e_2)$

Proof

If $ent_1=\rho(e_1)$ and $ent_2=\rho(e_2)$ are specializations, then $thingOf(ent_1)=thingOf(ent_2)$.

QED

Inference 21 (specialization-attributes-inference)
$\forall e_1,attrs,e_2.$
$entity(e_1,attrs) \wedge specializationOf(e_2,e_1)$
$\Rightarrow entity(e_2,attrs)$

Proof

Suppose $ent_1=\rho(e_1)$ and $ent_2=\rho(e_2)$. Suppose (att,v) is an attribute-value pair in $attrs$. Since $entity(e_1,attrs)$ holds, we know that $v \in value(ent_1,att)$. Thus $v \in value(ent_2,att)$ since $value(ent_2,att) \supseteq value(ent_1,att)$. Since this is the case for all attribute-value pairs in $attrs$, and since e_2 obviously denotes an entity, we can conclude $W,\rho \models entity(e_2,attrs)$.

QED

5.2 Constraints

5.2.1 Uniqueness constraints

Constraint 22 (key-object)

1. $\forall id, attrs_1, attrs_2 . entity(id, attrs_1) \wedge entity(id, attrs_2) \Rightarrow entity(id, attrs_1 \cup attrs_2)$
2. $\forall id, t_1, t'_1, t_2, t'_2, attrs_1, attrs_2 .\ activity(id, t_1, t_2, attrs_1) \wedge activity(id, t'_1, t'_2, attrs_2) \Rightarrow activity(id, t_1, t_2, attrs_1 \cup attrs_2) \wedge t_1 = t'_1 \wedge t_2 = t'_2$
3. $\forall id, attrs_1, attrs_2 . agent(id, attrs_1) \wedge agent(id, attrs_2) \Rightarrow agent(id, attrs_1 \cup attrs_2)$.

Proof

These properties follow immediately from the definitions of the semantics of the respective assertions, because functions are used for the underlying data.

QED

Constraint 23 (key-properties)

1. $\forall id, e, e', a, a', t, t', attrs_1, attrs_2 .$
   $wasGeneratedBy(id, e, a, t, attrs) \wedge wasGeneratedBy(id, e', a', t', attrs_2)$
   $\Rightarrow wasGeneratedBy(id, e, a, t, attrs_1 \cup attrs_2) \wedge e = e' \wedge a = a' \wedge t = t'$
2. $\forall id, e, e', a, a', t, t', attrs_1, attrs_2 .$
   $used(id, a, e, t, attrs) \wedge used(id, a', e', t', attrs_2)$
   $\Rightarrow used(id, a, e', t, attrs_1 \cup attrs_2) \wedge e = e' \wedge a = a' \wedge t = t'$
3. $\forall id, a_1, a_2, a'_1, a'_2, attrs_1, attrs_2 .$
   $wasInformedBy(id, a_1, a_2, attrs) \wedge wasInformedBy(id, a'_1, a'_2, attrs_2)$
   $\Rightarrow wasInformedBy(id, a_1, a_2, attrs_1 \cup attrs_2) \wedge a_1 = a'_1 \wedge a_2 = a'_2$
4. $\forall id, e, e' a_1, a_2, a'_1, a'_2, t, t', attrs_1, attrs_2 .$
   $wasStartedBy(id, a_2, e, a_1, t, attrs_1) \wedge wasStartedBy(id, a'_2, e', a'_1, t', attrs_2)$
   $\Rightarrow wasStartedBy(id, a_2, e, a_1, t, attrs_1 \cup attrs_2) \wedge a_1 = a'_1 \wedge e = e' \wedge a_2 = a'_2 \wedge t = t'$
5. $\forall id, e, e' a_1, a_2, a'_1, a'_2, t, t', attrs_1, attrs_2 .$
   $wasEndedBy(id, a_2, e, a_1, t, attrs_1) \wedge wasEndedBy(id, a'_2, e', a'_1, t', attrs_2)$
   $\Rightarrow wasEndedBy(id, a_2, e, a_1, t, attrs_1 \cup attrs_2) \wedge a_1 = a'_1 \wedge e = e' \wedge a_2 = a'_2 \wedge t = t'$
6. $\forall id, e, e', a, a', t, t', attrs_1, attrs_2 .$
   $wasInvalidatedBy(id, e, a, t, attrs_1) \wedge wasInvalidatedBy(id, e', a', t', attrs_2)$
   $\Rightarrow wasInvalidatedBy(id, e, a, t, attrs_1 \cup attrs_2) \wedge e = e' \wedge a = a' \wedge t = t'$
7. $\forall id, e_1, e'_1, e_2, e'_2, a, a', g, g', u, u', attrs_1, attrs_2 .$
   $wasDerivedFrom(id, e_2, e_1, a, g_2, u_1, attrs_1) \wedge wasDerivedFrom(id, e'_2, e'_1, a', g'_2, u'_1, attrs_2)$
   $\Rightarrow wasDerivedFrom(id, e_2, e_1, a, g_2, u_1, attrs_1 \cup attrs_2) \wedge e_1 = e'_1 \wedge e_2 = e'_2 \wedge a = a' \wedge g = g' \wedge u = u'$
8. $\forall id, e, e', ag, ag', attrs_1, attrs_2 .$
   $wasAttributedTo(id, e, ag, attrs_1) \wedge wasAttributedTo(id, e', ag', attrs_2)$
   $\Rightarrow wasAttributedTo(id, e, ag, attrs_1 \cup attrs_2) \wedge e = e' \wedge ag = ag'$
9. $\forall id, a, a', ag, ag', pl, pl', attrs_1, attrs_2 .$
   $wasAssociatedWith(id, a, ag, pl, attrs_1) \wedge wasAssociatedWith(id, a', ag', pl', attrs_2)$
   $\Rightarrow wasAssociatedWith(id, a, ag, pl, attrs_1 \cup attrs_2) \wedge a = a' \wedge ag = ag' \wedge pl = pl'$
10. $\forall id, ag_1, ag'_1, ag_2, ag'_2, a, a', attrs_1, attrs_2 .$
    $actedOnBehalfOf(id, ag_2, ag_1, a, attrs_1) \wedge actedOnBehalfOf(id, ag'_2, ag'_1, a', attrs_2)$
    $\Rightarrow actedOnBehalfOf(id, ag_2, ag_1, a, attrs_1 \cup attrs_2) \wedge ag_1 = ag'_1 \wedge ag_2 = ag'_2 \wedge a = a'$
11. $\forall id, o_1, o_2, o'_1, o'_2, attrs_1, attrs_2 .$
    $wasInfluencedBy(id, o'_2, o'_1, attrs_1) \wedge wasInfluencedBy(id, o'_2, o'_1, attrs_2)$
    $\Rightarrow wasInfluencedBy(id, o_2, o_1, attrs_1 \cup attrs_2) \wedge o_1 = o'_1 \wedge o_2 = o'_2$

Proof

These properties follow immediately from the definitions of the semantics of the respective assertions, again because functions are used for the underlying data.

QED

Constraint 24 (unique-generation)
$\forall gen_1,gen_2,e,a,t_1,t_2,attrs_1,attrs_2.$
$wasGeneratedBy(gen_1,e,a,t_1,attrs_1) \wedge wasGeneratedBy(gen_2,e,a,t_2,attrs_2)$
$\Rightarrow gen_1=gen_2$

Proof

This follows from Axiom 18.

QED

Constraint 25 (unique-invalidation)
$\forall inv_1,inv_2,e,a,t_1,t_2,attrs_1,attrs_2.$
$wasInvalidatedBy(inv_1,e,a,t_1,attrs_1) \wedge wasInvalidatedBy(inv_2,e,a,t_2,attrs_2)$
$\Rightarrow inv_1=inv_2$

Proof

This follows from Axiom 19.

QED

Constraint 26 (unique-wasStartedBy)
$\forall start_1,start_2,a,e_1,e_2,a_0,t_1,t_2,attrs_1,attrs_2.$
$wasStartedBy(start_1,a,e_1,a_0,t_1,attrs_1) \wedge wasStartedBy(start_2,a,e_2,a_0,t_2,attrs_2)$
$\Rightarrow start_1=start_2$

Proof

This follows from Axiom 20.

QED

Constraint 27 (unique-wasEndedBy)
$\forall end_1,end_2,a,e_1,e_2,a_0,t_1,t_2,attrs_1,attrs_2.$
$wasEndedBy(end_1,a,e_1,a_0,t_1,attrs_1) \wedge wasEndedBy(end_2,a,e_2,a_0,t_2,attrs_2)$
$\Rightarrow end_1=end_2$

Proof

This follows from Axiom 21.

QED

Constraint 28 (unique-startTime)
$\forall start,a_1,a_2,t,t_1,t_2,e,attrs,attrs_1.$
$activity(a_2,t_1,t_2,attrs) \wedge wasStartedBy(start,a_2,e,a_1,t,attrs_1)$
$\Rightarrow t_1=t$

Proof

This follows from the semantics of $wasStartedBy$, since the start times must both match that of the activity.

QED

Constraint 29 (unique-endTime)
$\forall end,a_1,a_2,t,t_1,t_2,e,attrs,attrs_1.$
$activity(a_2,t_1,t_2,attrs) \wedge wasEndedBy(end,a_2,e,a_1,t,attrs_1)$
$\Rightarrow t_2=t$

Proof

This follows from the semantics of *wasEndedBy*, since the end times must both match that of the activity.

QED

5.2.2 Ordering constraints

Constraint 30 (start-precedes-end)
$\forall start,end,a,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasStartedBy(start,a,e_1,a_1,t_1,attrs_1) \wedge wasEndedBy(end,a,e_2,a_2,t_2,attrs_2)$
$\Rightarrow start$ precedes end

Proof

This follows from Axiom 22.

QED

Constraint 31 (start-start-ordering)
$\forall start_1,start_2,a,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasStartedBy(start_1,a,e_1,a_1,t_1,attrs_1) \wedge wasStartedBy(start_2,a,e_2,a_2,t_2,attrs_2)$
$\Rightarrow start_1$ precedes $start_2$

Proof

This follows from Axiom 22.

QED

Constraint 32 (end-end-ordering)
$\forall end_1,end_2,a,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasEndedBy(end_1,a,e_1,a_1,t_1,attrs_1) \wedge wasEndedBy(end_2,a,e_2,a_2,t_2,attrs_2)$
$\Rightarrow end_1$ precedes end_2

Proof

This follows from Axiom 23.

QED

Constraint 33 (usage-within-activity)

1. $\forall start,use,a,e_1,e_2,a_1,t_1,t_2,attrs_1,attrs_2.$
   $wasStartedBy(start,a,e_1,a_1,t_1,attrs_1) \wedge used(use,a,e_2,t_2,attrs_2)$
   $\Rightarrow start$ precedes use
2. $\forall use,end,a,e_1,e_2,a_2,t_1,t_2,attrs_1,attrs_2.$
   $used(use,a,e_1,t_1,attrs_1) \wedge wasEndedBy(end,a,e_2,a_2,t_2,attrs_2)$
   $\Rightarrow use$ precedes end

Proof

Part 1 follows from Axiom 22 and part 2 follows from Axiom 23.

QED

Constraint 34 (generation-within-activity)

1. $\forall start,gen,e_1,e_2,a,a_1,t_1,t_2,attrs_1,attrs_2.$
   $wasStartedBy(start,a,e_1,a_1,t_1,attrs_1) \wedge wasGeneratedBy(gen,e_2,a,t_2,attrs_2)$
   $\Rightarrow start$ precedes gen
2.

$\forall gen,end,e,e_1,a,a_1,t,t_1,attrs,attrs_1.$
$wasGeneratedBy(gen,e,a,t,attrs) \wedge wasEndedBy(end,a,e_1,a_1,t_1,attrs_1)$
$\Rightarrow gen$ precedes end

Proof

Part 1 follows from Axiom 22 and part 2 follows from Axiom 23.

QED

Constraint 35 (wasInformedBy-ordering)
$\forall id,start,end,a_1,a'_1,a_2,a'_2,e_1,e_2,t_1,t_2,attrs,attrs_1,attrs_2.$
$wasInformedBy(id,a_2,a_1,attrs) \wedge wasStartedBy(start,a_1,e_1,a'_1,t_1,attrs_1) \wedge wasEndedBy(end,a_2,e_2,a'_2,t_2,attrs_2)$
$\Rightarrow start$ precedes end

Proof

This follows from the semantics of $wasInformedBy$, Axiom 24, and the previous two constraints, because $wasInformedBy$ implies the existence of intermediate generation and usage events linking a_1 and a_2 through an entity e. The generation of e must precede its use.

QED

Constraint 36 (generation-precedes-invalidation)
$\forall gen,inv,e,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasGeneratedBy(gen,e,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv,e,a_2,t_2,attrs_2)$
$\Rightarrow gen$ precedes inv

Proof

This follows from Axiom 24.

QED

Constraint 37 (generation-precedes-usage)
$\forall gen,use,e,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasGeneratedBy(gen,e,a_1,t_1,attrs_1) \wedge used(use,a_2,e,t_2,attrs_2)$
$\Rightarrow gen$ precedes use

Proof

This follows from Axiom 24.

QED

Constraint 38 (usage-precedes-invalidation)
$\forall use,inv,a_1,a_2,e,t_1,t_2,attrs_1,attrs_2.$
$used(use,a_1,e,t_1,attrs_1) \wedge wasInvalidatedBy(inv,e,a_2,t_2,attrs_2)$
$\Rightarrow use$ precedes inv

Proof

This follows from Axiom 25.

QED

Constraint 39 (generation-generation-ordering)
$\forall gen_1,gen_2,e,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasGeneratedBy(gen_1,e,a_1,t_1,attrs_1) \wedge wasGeneratedBy(gen_2,e,a_2,t_2,attrs_2)$
$\Rightarrow gen_1$ precedes gen_2

Proof

This follows from Axiom 24.

QED

Constraint 40 (invalidation-invalidation-ordering)
$\forall inv_1,inv_2,e,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$wasInvalidatedBy(inv_1,e,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv_2,e,a_2,t_2,attrs_2)$
$\Rightarrow inv_1$ precedes inv_2

Proof

This follows from Axiom 25.

QED

Constraint 41 (derivation-usage-generation-ordering)
$\forall d,e_1,e_2,a,gen_2,use_1,attrs.$
$notNull(a) \wedge notNull(gen_2) \wedge notNull(use_1) \wedge wasDerivedFrom(d,e_2,e_1,a,gen_2,use_1,attrs)$
$\Rightarrow use_1$ precedes gen_2

Proof

This follows from Axiom 26.

QED

Constraint 42 (derivation-generation-generation-ordering)
$\forall d,gen_1,gen_2,e_1,e_2,a,a_1,a_2,g,u,t_1,t_2,attrs,attrs_1,attrs_2.$
$wasDerivedFrom(d,e_2,e_1,a,g,u,attrs) \wedge wasGeneratedBy(gen_1,e_1,a_1,t_1,attrs_1) \wedge wasGeneratedBy(gen_2,e_2,a_2,t_2,attrs_2)$
$\Rightarrow gen_1$ strictlyPrecedes gen_2

Proof

This follows from Axiom 27.

QED

Constraint 43 (wasStartedBy-ordering)

1. $\forall gen,start,e,a,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasGeneratedBy(gen,e,a_1,t_1,attrs_1) \wedge wasStartedBy(start,a,e,a_2,t_2,attrs_2)$
   $\Rightarrow gen$ precedes $start$
2. $\forall start,inv,e,a,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasStartedBy(start,a,e,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv,e,a_2,t_2,attrs_2)$
   $\Rightarrow start$ precedes inv

Proof

Part 1 follows from Axiom 24. Part 2 follows from Axiom 25.

QED

Constraint 44 (wasEndedBy-ordering)

1. $\forall gen,end,e,a,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasGeneratedBy(gen,e,a_1,t_1,attrs_1) \wedge wasEndedBy(end,a,e,a_2,t_2,attrs_2)$
   $\Rightarrow gen$ precedes end
2. $\forall end,inv,e,a,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasEndedBy(end,a,e,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv,e,a_2,t_2,attrs_2)$
   $\Rightarrow end$ precedes inv

Proof

Part 1 follows from Axiom 24. Part 2 follows from Axiom 25.

QED

Constraint 45 (specialization-generation-ordering)
$\forall gen_1,gen_2,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$specializationOf(e_2,e_1) \wedge wasGeneratedBy(gen_1,e_1,a_1,t_1,attrs_1) \wedge wasGeneratedBy(gen_2,e_2,a_2,t_2,attrs_2)$
$\Rightarrow gen_1$ precedes gen_2

Proof

This follows from Axiom 24 and the fact that if e_2 specializes e_1 then all of the events of e_2 are events of e_1. Thus, the generation of e_1 precedes all events of e_2.

QED

Constraint 46 (specialization-invalidation-ordering)
$\forall inv_1,inv_2,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
$specializationOf(e_1,e_2) \wedge wasInvalidatedBy(inv_1,e_1,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv_2,e_2,a_2,t_2,attrs_2)$
$\Rightarrow inv_1$ precedes inv_2

Proof

This follows from Axiom 25 and the fact that if e_2 specializes e_1 then all of the events of e_2 are events of e_1. Thus, the invalidation of e_1 follows all events of e_2.

QED

Constraint 47 (wasAssociatedWith-ordering)

In the following inferences, pl may be a placeholder -.

1. $\forall assoc,start_1,inv_2,ag,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasAssociatedWith(assoc,a,ag,pl,attrs) \wedge wasStartedBy(start_1,a,e_1,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv_2,ag,a_2,t_2,attrs_2)$
   $\Rightarrow start_1$ precedes inv_2
2. $\forall assoc,gen_1,end_2,ag,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasAssociatedWith(assoc,a,ag,pl,attrs) \wedge wasGeneratedBy(gen_1,ag,a_1,t_1,attrs_1) \wedge wasEndedBy(end_2,a,e_2,a_2,t_2,attrs_2)$
   $\Rightarrow gen_1$ precedes end_2
3. $\forall assoc,start_1,end_2,ag,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasAssociatedWith(assoc,a,ag,pl,attrs) \wedge wasStartedBy(start_1,a,e_1,a_1,t_1,attrs_1) \wedge wasEndedBy(end_2,ag,e_2,a_2,t_2,attrs_2)$
   $\Rightarrow start_1$ precedes end_2
4. $\forall assoc,start_1,end_2,ag,e_1,e_2,a_1,a_2,t_1,t_2,attrs_1,attrs_2.$
   $wasAssociatedWith(assoc,a,ag,pl,attrs) \wedge wasStartedBy(start_1,ag,e_1,a_1,t_1,attrs_1) \wedge wasEndedBy(end_2,a,e_2,a_2,t_2,attrs_2)$
   $\Rightarrow start_1$ precedes end_2

Proof

The four parts follow from Axiom 28 through Axiom 31 respectively.

QED

Constraint 48 (wasAttributedTo-ordering)

1. $\forall att,gen_1,gen_2,e,a_1,a_2,t_1,t_2,ag,attrs,attrs_1,attrs_2.$
   $wasAttributedTo(att,e,ag,attrs) \wedge wasGeneratedBy(gen_1,ag,a_1,t_1,attrs_1) \wedge wasGeneratedBy(gen_2,e,a_2,t_2,attrs_2)$
   $\Rightarrow gen_1$ precedes gen_2
2. $\forall att,start_1,gen_2,e,e_1,a_1,a_2,ag,t_1,t_2,attrs,attrs_1,attrs_2.$
   $wasAttributedTo(att,e,ag,attrs) \wedge wasStartedBy(start_1,ag,e_1,a_1,t_1,attrs_1) \wedge wasGeneratedBy(gen_2,e,a_2,t_2,attrs_2)$
   $\Rightarrow start_1$ precedes gen_2

Proof

These properties follow from Axiom 32 and Axiom 33.

QED

Constraint 49 (actedOnBehalfOf-ordering)

1. $\forall del,gen_1,inv_2,ag_1,ag_2,a,a_1,a_2,t_1,t_2,attrs,attrs_1,attrs_2.$
$actedOnBehalfOf(del,ag_2,ag_1,a,attrs) \wedge wasGeneratedBy(gen_1,ag_1,a_1,t_1,attrs_1) \wedge wasInvalidatedBy(inv_2,ag_2,a_2,t_2,attrs_2)$
$\Rightarrow gen_1$ precedes inv_2
2. $\forall del,start_1,end_2,ag_1,ag_2,a,a_1,a_2,e_1,e_2,t_1,t_2,attrs,attrs_1,attrs_2.$
$actedOnBehalfOf(del,ag_2,ag_1,a,attrs) \wedge wasStartedBy(start_1,ag_1,e_1,a_1,t_1,attrs_1) \wedge wasEndedBy(end_2,ag_2,e_2,a_2,t_2,attrs_2)$
$\Rightarrow start_1$ precedes end_2

Proof

These properties follow from Axiom 34 and Axiom 35.

QED

5.2.3 Typing constraints

Constraint 50 (typing)

1. $\forall e,attrs.$
$entity(e,attrs)$
$\Rightarrow typeOf(e,entity)$
2. $\forall ag,attrs.$
$agent(ag,attrs)$
$\Rightarrow typeOf(ag,agent)$
3. $\forall a,t_1,t_2,attrs.$
$activity(a,t_1,t_2,attrs)$
$\Rightarrow typeOf(a,activity)$
4. $\forall u,a,e,t,attrs.$
$used(u,a,e,t,attrs)$
$\Rightarrow typeOf(a,activity) \wedge typeOf(e,entity)$
5. $\forall g,a,e,t,attrs.$
$wasGeneratedBy(g,e,a,t,attrs)$
$\Rightarrow typeOf(a,activity) \wedge typeOf(e,entity)$
6. $\forall inf,a_2,a_1,t,attrs.$
$wasInformedBy(inf,a_2,a_1,t,attrs)$
$\Rightarrow typeOf(a_1,activity) \wedge typeOf(a_2,activity)$
7. $\forall start,a_2,e,a_1,t,attrs.$
$wasStartedBy(start,a_2,e,a_1,t,attrs)$
$\Rightarrow typeOf(a_1,activity) \wedge typeOf(a_2,activity) \wedge typeOf(e,entity)$
8. $\forall end,a_2,e,a_1,t,attrs.$
$wasEndedBy(end,a_2,e,a_1,t,attrs)$
$\Rightarrow typeOf(a_1,activity) \wedge typeOf(a_2,activity) \wedge typeOf(e,entity)$
9. $\forall inv,a,e,t,attrs.$
$wasInvalidatedBy(inv,e,a,t,attrs)$
$\Rightarrow typeOf(a,activity) \wedge typeOf(e,entity)$
10. $\forall id,e_2,e_1,a,g_2,u_1,attrs.$
$notNull(a) \wedge notNull(g_2) \wedge notNull(u_1) \wedge wasDerivedFrom(id,e_2,e_1,a,g_2,u_1,attrs)$
$\Rightarrow typeOf(e_2,entity) \wedge typeOf(e_1,activity) \wedge typeOf(a,activity)$
11. $\forall id,e_2,e_1,attrs.$
$wasDerivedFrom(id,e_2,e_1,-,-,-,attrs)$
$\Rightarrow typeOf(e_2,entity) \wedge typeOf(e_1,activity)$
12. $\forall id,e,ag,attrs.$
$wasAttributedTo(id,e,ag,attrs)$
$\Rightarrow typeOf(e,entity) \wedge typeOf(ag,agent)$
13. $\forall id,a,ag,pl,attrs.$
$notNull(pl) \wedge wasAssociatedWith(id,a,ag,pl,attrs)$
$\Rightarrow typeOf(a,activity) \wedge typeOf(ag,agent) \wedge typeOf(pl,entity)$
14.

$\forall id,a,ag,attrs.$
$wasAssociatedWith(id,a,ag,-,attrs)$
$\Rightarrow typeOf(a,activity)\wedge typeOf(ag,agent)$

15. $\forall id,ag_2,ag_1,a,attrs.$
$actedOnBehalfOf(id,ag_2,ag_1,a,attrs)$
$\Rightarrow typeOf(ag_2,agent)\wedge typeOf(ag_1,agent)\wedge typeOf(a,activity)$

16. $\forall e_2,e_1.$
$alternateOf(e_2,e_1)$
$\Rightarrow typeOf(e_2,entity)\wedge typeOf(e_1,entity)$

17. $\forall e_2,e_1.$
$specializationOf(e_2,e_1)$
$\Rightarrow typeOf(e_2,entity)\wedge typeOf(e_1,entity)$

18. $\forall c,e.$
$hadMember(c,e)$
$\Rightarrow typeOf(c,Collection)\wedge typeOf(e,entity)$

19. $\forall c.$
$entity(c,[prov{:}type{=}prov{:}emptyCollection]))$
$\Rightarrow typeOf(c,entity)\wedge typeOf(c,Collection)\wedge typeOf(c,EmptyCollection)$

Proof

Each typing constraint follows immediately from well-formedness criteria marked [WF] in the corresponding semantics for formulas. The final constraint requires Axiom 36.

QED

5.2.4 Impossibility constraints

Constraint 51 (impossible-unspecified-derivation-generation-use)

1. $\forall id,e_1,e_2,g,attrs.$
$notNull(g)\wedge wasDerivedFrom(id,e_2,e_1,-,g,-,attrs)$
$\Rightarrow False$
2. $\forall id,e_1,e_2,u,attrs.$
$notNull(u)\wedge wasDerivedFrom(id,e_2,e_1,-,-,u,attrs)$
$\Rightarrow False$
3. $\forall id,e_1,e_2,g,u,attrs.$
$notNull(g)\wedge notNull(u)\wedge wasDerivedFrom(id,e_2,e_1,-,g,u,attrs)$
$\Rightarrow False$

Proof

Each part follows from the fact that the semantics of $wasDerivedFrom$ only allows formulas to hold when either all three of a,g,u are "$-$" (denoting $\perp$) or none of them are.

QED

Constraint 52 (impossible-specialization-reflexive)

$\forall e.$
$specializationOf(e,e)$
$\Rightarrow False$

Proof

This follows from the fact that in the semantics of $specializationOf$, the requirement that one of the inclusions is strict implies that the two entities cannot be the same.

QED

Constraint 53 (impossible-property-overlap)

For each r and $s\in\{used,wasGeneratedBy,wasInvalidatedBy,wasStartedBy,wasEndedBy,wasInformedBy,wasAttributedTo,wasAssociatedWith,actedOnBehalfOf\}$ such that r and s are different relation names, the following constraint holds:

$\forall id,a_1,\ldots,a_m,b_1,\ldots,b_n.$
$r(id,a_1,\ldots,a_m) \wedge s(id,b_1,\ldots,b_n)$
$\Rightarrow False$

Proof

This follows from the assumption that the different kinds of influences are disjoint sets, characterized by their types. Note that generic influences are allowed to overlap with more specific kinds of influence.

QED

Constraint 54 (impossible-object-property-overlap)

For each $p \in \{entity, activity, agent\}$ and each $r \in \{used, wasGeneratedBy, wasInvalidatedBy, wasStartedBy, wasEndedBy, wasInformedBy, wasAttributedTo, wasAssociatedWith, actedOnBehalfOf, wasInfluence$ the following constraint holds:

$\forall id,a_1,\ldots,a_m,b_1,\ldots,b_n.$
$p(id,a_1,\ldots,a_m) \wedge r(id,b_1,\ldots,b_n)$
$\Rightarrow False$

Proof

This follows from the assumption that influences are distinct from other objects (entities, activities or agents).

QED

Constraint 55 (entity-activity-disjoint)

$\forall id.$
$typeOf(id,entity) \wedge typeOf(id,activity)$
$\Rightarrow False$

Proof

This follows from the assumption that entities and activities are disjoint.

QED

Constraint 56 (membership-empty-collection)

$\forall c,e.$
$hasMember(c,e) \wedge typeOf(c,EmptyCollection)$
$\Rightarrow False$

Proof

This follows from the definition of the semantics of $typeOf(c,EmptyCollection)$, which requires that there are no members of the collection denoted by c.

QED

6. Soundness and Completeness

Above we have presented arguments for the soundness of the constraints and inferences with respect to the semantics. Here, we relate the notions of ***validity*** and ***normal form*** defined in PROV-CONSTRAINTS to the semantics.

6.1 Soundness

Our main soundness result is:

Theorem 39 (soundness-theorem)

Let W be a PROV structure, that is, a structure providing all of the components above and satisfying all of the axioms.

1. If I is an instance and $W \models I$ and I' is obtained from I by applying one of the PROV inferences, then $W \models I'$.
2. If I is an instance and $W \models I$ and I' is obtained from I by applying one of the PROV key or uniqueness constraints, then $W \models I'$.
3. If I is an instance and $W \models I$ then I has a normal form I' and $W \models I'$.
4. If I is a normal form and $W \models I$ then I satisfies all of the ordering, typing and impossibility constraints.
5. If $W \models I$ then I is valid.

Proof

For part 1, the arguments are as in the previous section.

For part 2, if $W \models I$ then since W satisfies the logical forms of all uniqueness and key constraints, constraint application cannot fail on I and $W \models I'$.

For part 3, proceed by induction on a terminating sequence of inference or uniqueness constraint steps: if I is in normal form then we are done. If I is not in normal form then if an inference is applicable, then use part 1; if a uniqueness constraint is applicable, then use part 2.

For part 4, the arguments are as in the previous section for each constraint.

Finally, for part 5, suppose $W \models I$. Then $W \models I'$ where I' is the normal form of I by part 2. By part 3, I' satisfies all of the remaining constraints, so I is valid.

QED

6.2 Weak Completeness

In this section we give a translation from valid PROV instances to structures, and show that a valid PROV instance has a model. We call this property *weak completeness*.

Remark

The term *weak* refers to the fact that there are still some inferences that are sound in the semantics but not enforced by validation. For example, consider the following (valid) PROV instance fragment:

```
entity(e,[a=1])
agent(e,[b=2])
```

This instance is valid and has a model, but in every model satisfying the instance, it is also true that:

```
entity(e,[a=1,b=2])
agent(e,[a=1,b=2])
```

Thus, weak completeness captures the fact that every valid instance has a model, but does not imply that a valid instance satisfies all of the deductions possible in that model.

Let I be a valid PROV instance that is in normal form. We define a structure $M(I)$ as follows, by giving the sets, functions and relations specified in the components in Section 3, and finally verifying that the axioms hold.

First, without loss of generality, we assume that all times specified in activity or event formulas in I are ground values. If not, set each variable in such a position to some dummy value. This is justified by the following fact:

Lemma 40 (time-grounding)

If I is valid then $S(I)$ is valid, where S is any substitution that maps time variables to time constants.

Proof

First, consider a substitution $S=[t:=c]$ that maps a single time variable to a constant. It is straightforward to check that if I is in normal form, then $S(I)$ is in normal form, since none of the inferences or uniqueness constraints can be enabled by changing a time variable uniformly in I. Similarly, the remaining constraints are insensitive to the time values, so $S(I)$ is in normal form and satisfies all of the remaining constraints just as I does. The general case of a substitution that replaces multiple time variables with constants is a straightforward generalization since we can view such a substitution as a composition of single-variable substitutions.

QED

6.2.1 Sets

The sets of structure $M(I)$ are:

$$
\begin{array}{rcl}
\mathit{Entities} & = & \{id|I \models \mathit{typeOf}(id,\mathit{entity})\} \\
\mathit{Plans} & = & \{pl|\exists id,ag,ac,attrs.\ \mathit{wasAssociatedWith}(id,ag,act,pl,attrs) \in I, pl \neq -\} \\
\mathit{Collections} & = & \{c|I \models \mathit{typeOf}(c,\mathit{prov}{:}\mathit{Collection}) \text{ or } I \models \mathit{typeOf}(c,\mathit{prov}{:}\mathit{EmptyCollection})\} \\
\mathit{Activities} & = & \{id|I \models \mathit{typeOf}(id,\mathit{activity})\} \\
 & \cup & \{a_{id},a'_{id}|id \in \mathit{Entities}\} \\
 & \cup & \{a_{id}|\exists id,e_2,e_1.\ \mathit{wasDerivedFrom}(id,e_2,e_1,-,-,-,attrs) \in I\} \\
\mathit{Agents} & = & \{id|I \models \mathit{typeOf}(id,\mathit{agent})\} \\
\mathit{Usages} & = & \{id|\exists a,e,t,attrs.\ \mathit{used}(id,a,e,t,attrs) \in I\} \\
 & \cup & \{u_{id}|\exists id,e_2,e_1,attrs.\ \mathit{wasDerivedFrom}(id,e_2,e_1,-,-,-,attrs) \in I\} \\
\mathit{Generations} & = & \{id|\exists e,a,t,attrs.\ \mathit{wasGeneratedBy}(id,e,a,t,attrs) \in I\} \\
 & \cup & \{g_{id}|\exists id,e_2,e_1,attrs.\ \mathit{wasDerivedFrom}(id,e_2,e_1,-,-,-,attrs) \in I\} \\
 & \cup & \{g_{id}|id \in \mathit{Entities}\} \\
\mathit{Invalidations} & = & \{id|\exists e,a,t,attrs.\ \mathit{wasInvalidatedBy}(id,e,a,t,attrs) \in I\} \\
 & \cup & \{i_{id}|id \in \mathit{Entities}\} \\
\mathit{Starts} & = & \{id|\exists a,e,a',t,attrs.\ \mathit{wasStartedBy}(id,a,e,a',t,attrs) \in I\} \\
\mathit{Ends} & = & \{id|\exists a,e,a',t,attrs.\ \mathit{wasEndedBy}(id,a,e,a',t,attrs) \in I\} \\
\mathit{Events} & = & \mathit{Usages} \cup \mathit{Generations} \cup \mathit{Invalidations} \cup \mathit{Starts} \cup \mathit{Ends} \\
\mathit{Associations} & = & \{id|\exists ag,act,pl,attrs.\ \mathit{wasAssociatedWith}(id,ag,act,pl,attrs) \in I\} \\
\mathit{Attributions} & = & \{id|\exists e,ag,attrs.\ \mathit{wasAttributedTo}(id,e,ag,attrs) \in I\} \\
\mathit{Delegations} & = & \{id|\exists ag_2,ag_1,attrs.\ \mathit{actedOnBehalfOf}(id,ag_2,ag_1,act,attrs) \in I\} \\
\mathit{Communications} & = & \{id|\exists a_2,a_1,attrs.\ \mathit{wasInformedBy}(id,a_2,a_1,attrs) \in I\} \\
\mathit{Derivations} & = & \{id|\exists e_2,e_1,a,g,u,attrs.\ \mathit{wasDerivedFrom}(id,e_2,e_1,a,g,u,attrs) \in I\} \\
\mathit{Influences} & = & \mathit{Events} \cup \mathit{Associations} \cup \mathit{Attributions} \cup \mathit{Communications} \cup \mathit{Delegations} \\
 & \cup & \{id|\exists o_2,o_1,attrs.\ \mathit{wasInfluencedBy}(id,o_2,o_1,attrs) \in I\} \\
\mathit{Objects} & = & \mathit{Entities} \cup \mathit{Activities} \cup \mathit{Agents} \cup \mathit{Influences}
\end{array}
$$

In the definitions of *Entities*, *Collections*, *Activities* and *Agents* we use the notation $I \models \mathit{typeOf}(id,t)$ to indicate that id must have type t in I according to the typing constraints. For example, for entities, this means that the set *Entities* contains all identifiers e,e' appearing in the $\mathit{entity}(e,attrs)$, $\mathit{alternateOf}(e,e')$, or $\mathit{specializationOf}(e,e')$ formulas, as well as all tose appearing in the appropriate positions of other formulas, as specified in the typing constraints.

In the definitions of *Activities*, *Generations*, *Invalidations*, and *Usages* we write a_{id}, g_{id}, i_{id} and u_{id} respectively to indicate additional activities, generations and usages added for imprecise derivations or entities.

In addition, to define the set of *Things*, we introduce an equivalence relation on *Entities* as follows:

$$e_1 \equiv e_2 \Longleftrightarrow \mathit{alternateOf}(e_1,e_2) \in I$$

The fact that this is an equivalence relation follows from the fact that I is in normal form, since the constraints on *alternateOf* ensure that it is an equivalence relation. Recall that given an equivalence relation $\equiv$ on some set X, the *equivalence class* of $x \in X$ is the set $[x]_{\equiv} = \{y \in X | x \equiv y\}$. The *quotient* of X by an equivalence relation on X is the set of equivalence classes, $X_{\equiv} = \{[x]_{\equiv} | x \in X\}$. Now we define the set of *Things* as the quotient of $\equiv$-equivalence classes of *Entities*.

$$\mathit{Things} = \mathit{Entities}/_{\equiv} = \{[e]_{\equiv} | e \in \mathit{Entities}\}$$

Observe that since I is normalized and valid, entities and activities are disjoint, the influences are disjoint from entities, activities, and agents, and the different subsets of events and influences are pairwise disjoint, as required.

6.2.2 Functions

First, we consider the functions associated with *Entities*.

$$
\begin{array}{rcl}
\mathit{events}'(e) & = & \{id|\mathit{used}(id,a,e,t,attrs) \in I\} \\
 & \cup & \{id|\mathit{wasGeneratedBy}(id,e,a,t,attrs) \in I\} \\
 & \cup & \{id|\mathit{wasInvalidatedBy}(id,e,a,t,attrs) \in I\} \\
 & \cup & \{id|\mathit{wasStartedBy}(id,a,e,a',t,attrs) \in I\} \\
 & \cup & \{id|\mathit{wasEndedBy}(id,a,e,a',t,attrs) \in I\} \\
 & \cup & \{g_e,i_e\} \\
\mathit{events}(e) & = & \mathit{events}'(e) \cup \bigcup_{\mathit{specializationOf}(e',e) \in I} \mathit{events}'(e') \\
\mathit{value}'(e,a) & = & \{v|\mathit{entity}(e,attrs) \in I, (a=v) \in attrs\} (a \neq uniq) \\
\mathit{value}'(e,uniq) & = & \{uniq_e\} \\
\mathit{value}(e,a) & = & \mathit{value}'(e) \cup \bigcup_{\mathit{specializationOf}(e,e') \in I} \mathit{value}'(e') \\
\mathit{thingOf}(e) & = & [e]_{\equiv}
\end{array}
$$

Above, we introduce a fresh attribute name *uniq*, not already in use in I, along with a fresh value e and for each entity e we add a value $uniq_e$ to $\mathit{values}(e,uniq)$. This construction ensures that if an entity is a specialization of another in I then the specialization relationship will hold in $M(I)$. We also define the set of all events involved in e as the set of events immediately involved in e or any specialization of e. Similarly, the values of attributes of e are those immediately declared for e along with those of any e' that e specializes. We also introduce dummy generation and invalidation events for each entity e, along with activities a_e,a'_e to perform them.

Similarly, for *Things*, we employ an auxiliary function $\mathit{events}{:}\mathit{Things} \rightarrow P(\mathit{Events})$ that collects the set of all events in which one of the entities constituting the thing participated.

$$\begin{aligned} events(T) &= \bigcup_{e \in T} events(e) \\ value(T,a,evt) &= \bigcup_{e \in T, evt \in events(e)} value(e,a) \end{aligned}$$

The functions *events*, *startTime* and *endTime* mapping activities to their start and end times are defined as follows:

$$\begin{aligned} events(a) &= \{id | used(id,a,e,t,attrs) \in I\} \\ &\cup \{id | wasGeneratedBy(id,e,a,t,attrs) \in I\} \\ &\cup \{id | wasInvalidatedBy(id,e,a,t,attrs) \in I\} \\ &\cup \{id | wasStartedBy(id,a,e,a',t,attrs) \in I\} \\ &\cup \{id | wasEndedBy(id,a,e,a',t,attrs) \in I\} \\ &\cup \{g_e, i_e\} \\ startTime(id) &= t_1 \text{ where } activity(a,t_1,t_2,attrs) \in I \\ endTime(id) &= t_2 \text{ where } activity(a,t_1,t_2,attrs) \in I \end{aligned}$$

The start and end times are arbitrary (say, some zero value) for activities with no *activity* formula declaring the times. The above definitions of *startTime* and *endTime* ignore any start times asserted in *wasStartedBy* or *wasEndedBy* formulas. If both *activity* and *wasStartedBy*/*wasEndedBy* statements are present, then they must match, but PROV-CONSTRAINTS does not require that the times of multiple start or end events match for an activity with no *activity* statement.

Remark

The following valid instance exemplifies the above discussion, when $t_1 \neq t_2$:

```
wasStartedBy(id1;a,e1,a1,t1,[])
wasStartedBy(id2;a,e2,a2,t2,[])
```

This instance becomes invalid if we add an $activity(a,[])$ statement, because it expands to $activity(a,T_1,T_2,[])$ where T_1,T_2 are existential variables, and uniqueness constraints require that $t_1=T_1=t_2$, which leads to uniqueness constraint failure.

For other *Objects* besides *Entities* and *Activities*, the associated sets of *Events* are defined to be empty. (An *Agent* that happens to be an *Entity* or *Activity* will have the set of events defined above for the appropriate kind of object. Note that since *Entities* and *Activities* are disjoint, this definition is unambiguous.)

The function *time* mapping *Events* to their *Times* is defined as follows:

$$\begin{aligned} time(id) &= t \text{ where } used(id,a,e,t,attrs) \in I \\ time(id) &= t \text{ where } wasGeneratedBy(id,e,a,t,attrs) \in I \\ time(id) &= t \text{ where } wasInvalidatedBy(id,e,a,t,attrs) \in I \\ time(id) &= t \text{ where } wasStartedBy(id,a,e,a',t,attrs) \in I \\ time(id) &= t \text{ where } wasEndedBy(id,a,e,a',t,attrs) \in I \end{aligned}$$

This definition is deterministic because the sets of identifiers of different *Events* are disjoint, and the associated times are unique.

The functions giving the interpretations of the different identified influences are as follows:

$$\begin{aligned} used(id) &= (a,e) \text{ where } used(id,a,e,t,attrs) \in I \\ used(u_{id}) &= (a_{id},e_1) \text{ where } wasDerivedFrom(id,e_2,e_1,-,-,-,attrs) \in I \\ generated(id) &= (e,a) \text{ where } wasGeneratedBy(id,e,a,t,attrs) \in I \\ generated(g_{id}) &= (e_2,a_{id}) \text{ where } wasDerivedFrom(id,e_2,e_1,-,-,-,attrs) \in I \\ generated(g_e) &= (e,a_e) \text{ where } e \in Entities \\ invalidated(id) &= (e,a) \text{ where } wasInvalidatedBy(id,e,a,t,attrs) \in I \\ invalidated(i_e) &= (e,a'_e) \text{ where } e \in Entities \\ started(id) &= (a,e,a') \text{ where } wasStartedBy(id,a,e,a',t,attrs) \in I \\ ended(id) &= (a,e,a') \text{ where } wasEndedBy(id,a,e,a',t,attrs) \in I \\ associatedWith(id) &= (ag,act,pl) \text{ where } wasAssociatedWith(id,ag,act,pl,attrs) \in I \\ attributedTo(id) &= (e,ag) \text{ where } wasAttributedTo(id,e,ag,attrs) \in I \\ actedFor(id) &= (ag_2,ag_1,act) \text{ where } actedOnBehalfOf(id,ag_2,ag_1,act,attrs) \in I \\ communicated(id) &= (a_2,a_1) \text{ where } wasInformedBy(id,a_2,a_1,attrs) \in I \\ derivationPath(id) &= e_2 \cdot g \cdot a \cdot u \cdot e_1 \text{ where } wasDerivedFrom(id,e_2,e_1,a,g,u,attrs) \in I \\ derivationPath(id) &= e_2 \cdot g_{id} \cdot a_{id} \cdot u_{id} \cdot e_1 \text{ where } wasDerivedFrom(id,e_2,e_1,-,-,-,attrs) \in I \end{aligned}$$

Note that since I is normalized and valid, by the uniqueness constraints these functions are all well-defined. In the case for imprecise derivations, we generate additional activities, generations and usages linking e_2 to e_1.

The definition of the *influenced* function is more involved, and is as follows:

$$\begin{aligned} influenced(id) &= used(id) \cup generated(id) \cup invalidated(id) \\ &\cup \{(a,e) | (a,e,a') \in started(id)\} \\ &\cup \{(a,e) | (a,e,a') \in ended(id)\} \\ &\cup \{(ag,act) | (ag,act,pl) \in associatedWith(id)\} \\ &\cup attributedTo(id) \\ &\cup \{(ag_2,ag_1) | (ag_2,ag_1,act) \in actedFor(id)\} \\ &\cup communicated(id) \\ &\cup \{(e_2,e_1) | e_2 \cdot w \cdot e_1 \in derivationPath(id)\} \\ &\cup \{(o_2,o_1) | wasInfluencedBy(id,o_2,o_1) \in I\} \end{aligned}$$

This definition ensures that by construction $influenced(id)$ contains all of the other associated relationships. For any specific id, however, most of the above sets will be empty, and the final line will often be redundant. It is not always redundant, because it is possible to assert an unspecified influence in I.

It is straightforward to verify (by their definitions) that the event sets associated with entities and activities satisfy the side-conditions in Component 9.

Finally, the collection membership function $members$ is defined as follows:

$$members(c)=\{e|hadMember(c,e)\in I$$

6.2.3 Relations

We introduced a relation $\equiv$ corresponding to $alternateOf$ above, in defining $Things$, but this relation is not a component of the semantics.

The event ordering relation is defined as follows:

$$evt\preceq evt'\Longleftrightarrow(evt,evt')\in G_I$$

closed under reflexivity and transitivity. Here, we are using a slight abuse of notation: we write G_I for the directed graph that is used during validation of I to test for cycles among event ordering constraints. See Sec. 7.1 of PROV-CONSTRAINTS [PROV-CONSTRAINTS].

6.2.4 Axioms

To verify that the construction of $M(I)$ yields a PROV structure, we must ensure that all of the axioms and side-conditions in the components are satisfied. As noted above, the disjointness constraints are satisfied by construction.

For each axiom we give the corresponding justification:

1. Axiom 1 follows because I is normalized with respect to Inference 6.
2. Axiom 2 follows from the construction, since we add dummy generation and invalidation events for every entity.
3. Axioms 3 and 4 follow because I is normalized with respect to Inference 9 and 10 respectively.
4. Axiom 5 follows because I is normalized with respect to Inference 12.
5. Axioms 6 and 7 follow because I is normalized with respect to Inference 13 and 14 respectively.
6. Axioms 8 through 17 follow because I is normalized with respect to Inference 15.
7. Axioms 18 through 21 follow because I is normalized with respect to uniqueness constraints 24 through 27.
8. Axiom 22 follows because constraints 30, 31, 33, 34 ensure that a start event for an activity precedes any other start, end, usage or generation events involving that activity.
9. Axiom 23 follows because constraints 30, 32, 33, 34 ensure that an end event for an activity follows any other events involving that activity.
10. Axiom 24 follows because constraints 34, 36, 37, 39 ensure that a generation event for an entity precedes any other events involving that entity.
11. Axiom 25 follows because constraints 36, 38, 40, 43, 44 ensure that an invalidation event for an entity follows any other generation, usage, or invalidation events involving that entity.
12. Axiom 26 follows from constraint 41.
13. Axiom 27 follows from constraint 42 and from the fact that the event ordering constraint graph G_I associated with a valid instance I cannot have any cycles involving a strict precedence edge.
14. Axioms 28 through 31 follow from Constraint 47.
15. Axioms 32 and 33 follow from Constraint 48.
16. Axioms 34 and 35 follow from Constraint 49.
17. Axiom 36 follows from Constraint 50, part 19, and the semantics of $typeof$.

6.2.5 Main results

The main results of this section are that if a valid PROV instance I has a model $M\models I$ that satisfies all of the inferences and constraints. Thus, a form of completeness holds: every valid PROV instance has a model.

Theorem 41 (weak-completeness-theorem)

Suppose J is a valid PROV instance. Then there exists a PROV structure M such that $M\models J$.

Proof

First, we consider the case where J itself is a valid, normalized PROV instance I, with no existential variables, and let $M(I)$ be the corresponding structure. Then $M(I)$ is a PROV structure, satisfying all of the axioms (and hence all of the inferences and constraints) stated above.

Moreover, $M(I)\models I$, as can be verified on a case-by-case basis for each type of formula by considering its semantics and the definition of the construction of M. Most cases are straightforward; we consider the cases of $alternateOf$ and $specializationOf$ since they are among the most interesting.

- Suppose $alternateOf(e_1,e_2)\in I$. We wish to show that $M(I)\models alternateOf(e_1,e_2)$. Since there are no existential variables in I, we know that $e_1,e_2\in M(I).Entities$. Moreover, $e_1\equiv e_2$ according to the equivalence relation defined above, and so $thingOf(e_1)=[e_1]_{\equiv}=[e_2]_{\equiv}=thingOf(e_2)$, so we can conclude that $M(I)\models alternateOf(e_1,e_2)$.
- Suppose $specializationOf(e_1,e_2)\in I$. We wish to show that $M(I)\models specializationOf(e_1,e_2)$. Again, clearly $e_1,e_2\in Entities$, and since I satisfies all inferences, we know that $alternateOf(e_1,e_2)\in I$ so clearly $thingOf(e_2)=thingOf(e_1)$ as argued above. Next,

$$\begin{aligned} events(e_1) &= events'(e_1) \cup \bigcup_{specializationOf(e',e_1)\in I} events'(e') \\ &\subseteq events'(e_2) \cup \bigcup_{specializationOf(e',e_2)\in I} events'(e_2) \\ &= events(e_2) \end{aligned}$$

because $specializationOf(e_1,e_2)\in I$ and all e' that are specializations of e_1 are also specializations of e_2. Furthermore, for each $attr$,

$$\begin{aligned} value(e_1,attr) &= value'(e_1,attr) \cup \bigcup_{specializationOf(e_1,e')\in I} value'(e',attr) \\ &\supseteq value'(e_2,attr) \cup \bigcup_{specializationOf(e_2,e')\in I} value'(e',attr) \\ &= value(e_2,attr) \end{aligned}$$

for the same reason. Finally, by construction $uniq_{e_1}\in value(e_1,uniq)$ and $uniq_{e_1}\notin value(e_2,uniq)$ so the inclusion is strict for the special attribute $uniq$. Thus, we have verified all of the conditions necessary to conclude $M(I)\models specializationOf(e_1,e_2)$.

Next, we show how to handle a normalized, valid I contains existential variables $x_1,\ldots,x_n$. Choose fresh constants $c_1,\ldots,c_n$ of appropriate types for the existential variables and define $\rho(x_i)=c_i$. Then $M(\rho(I))\models\rho(I)$ by the above argument. Moreover, $M(\rho(I)),\rho\models I$. So $M(\rho(I))$ is itself the desired model.

Finally, to handle the case where J is an arbitrary valid instance, we need to show that if J is not in normal form, and normalizes to some I such that $M\models I$, then $M\models J$. We can prove this by induction on the length of the sequence of normalization steps. The base case, when $J=I$, is established already. Suppose J normalizes in $n+1$ steps and we can perform one normalization step on it to obtain J', which normalizes to I in n steps. By induction, we know that $M\models J'$. For each possible normalization step, we must show that if $M\models J'$ then $M\models J$.

First consider inference steps. These add information, that is, $J'\supseteq J$. Hence it is immediate that $M\models J$ since every formula in J is in J', and all formulas of J' are satisfied in M.

Next consider uniqueness constraint steps, which may involve merging formulas. That is, $J=J_0\cup\{r(id,a_1,\ldots,a_n,attrs_1),r(id,b_1,\ldots,b_n,attrs_2)\}$ and $J'=S(J_0)\cup\{r(id,S(a_1),\ldots,S(a_n),attrs_1\cup attrs_2)\}$, where S is a unifying substitution making $S(a_i)=S(b_i)$ for each $i\in\{1,\ldots,n\}$. Since $M\models J'$, we must have $M,\rho\models J'$ for some ρ, and therefore we must also have that $M,\rho\models S(J_0)$ and $M,\rho\models r(id,S(a_1),\ldots,S(a_n),attrs_1\cup attrs_2)$. We can extend ρ to a valuation ρ' such that $M,\rho'\models S(x_1)=x_1\wedge\cdots\wedge S(x_k)=x_k$ where $dom(S)=\{x_1,\ldots,x_k\}$. Also, $M,\rho'\models J_0$ and $M,\rho'\models r(id,a_1,\ldots,a_n,attrs_1\cup attrs_2)$. Moreover, since S is a unifier, we also have $M,\rho'\models r(id,b_1,\ldots,b_n,attrs_1\cup attrs_2)$. Finally, since we can always remove attributes from an atomic formula without damaging its satisfiability, we can conclude that $M,\rho'\models r(id,a_1,\ldots,a_n,attrs_1)\wedge r(id,b_1,\ldots,b_n,attrs_2)$. To conclude, we have shown $M\models J_0\cup\{r(id,a_1,\ldots,a_n,attrs_1),r(id,b_1,\ldots,b_n,attrs_2)\}$, that is, $M\models J$, as desired.

QED

A. Acknowledgements

This document has been produced by the PROV Working Group, and its contents reflect extensive discussion within the Working Group as a whole as well as feedback and comments from external reviewers. Thanks specifically to Khalid Belhajjame, Tom De Nies, Paolo Missier, Simon Miles, Luc Moreau, Satya Sahoo, Jan van den Bussche, Joachim Van Herwegen, and Antoine Zimmermann for detailed feedback.

We would also like to acknowledge Schloss Dagstuhl - Leibniz Center for Informatics, because significant progress was made on this document at Dagstuhl Seminar 12091 (Principles of Provenance) that took place from February 26 to March 2, 2012.

Thanks also to Robin Berjon for the ReSPec.js specification writing tool and to MathJax for their LaTeX-to-HTML conversion tools, both of which aided in the preparation of this document.

Members of the Provenance Working Group at the time of publication of this document were: Ilkay Altintas (Invited expert), Reza B'Far (Oracle Corporation), Khalid Belhajjame (University of Manchester), James Cheney (University of Edinburgh, School of Informatics), Sam Coppens (iMinds - Ghent University), David Corsar (University of Aberdeen, Computing Science), Stephen Cresswell (The National Archives), Tom De Nies (iMinds - Ghent University), Helena Deus (DERI Galway at the National University of Ireland, Galway, Ireland), Simon Dobson (Invited expert), Martin Doerr (Foundation for Research and Technology - Hellas(FORTH)), Kai Eckert (Invited expert), Jean-Pierre EVAIN (European Broadcasting Union, EBU-UER), James Frew (Invited expert), Irini Fundulaki (Foundation for Research and Technology - Hellas(FORTH)), Daniel Garijo (Universidad Politécnica de Madrid), Yolanda Gil (Invited expert), Ryan Golden (Oracle Corporation), Paul Groth (Vrije Universiteit), Olaf Hartig (Invited expert), David Hau (National Cancer Institute, NCI), Sandro Hawke (W3C/MIT), Jörn Hees (German Research Center for Artificial Intelligence (DFKI) Gmbh), Ivan Herman, (W3C/ERCIM), Ralph Hodgson (TopQuadrant), Hook Hua (Invited expert), Trung Dong Huynh (University of Southampton), Graham Klyne (University of Oxford), Michael Lang (Revelytix, Inc.), Timothy Lebo (Rensselaer Polytechnic Institute), James McCusker (Rensselaer Polytechnic Institute), Deborah McGuinness (Rensselaer Polytechnic Institute), Simon Miles (Invited expert), Paolo Missier (School of Computing Science, Newcastle university), Luc Moreau (University of Southampton), James Myers (Rensselaer Polytechnic Institute), Vinh Nguyen (Wright State University), Edoardo Pignotti (University of Aberdeen, Computing Science), Paulo da Silva Pinheiro (Rensselaer Polytechnic Institute), Carl Reed (Open Geospatial Consortium), Adam Retter (Invited Expert), Christine Runnegar (Invited expert), Satya Sahoo (Invited expert), David Schaengold (Revelytix, Inc.), Daniel Schutzer (FSTC, Financial Services Technology Consortium), Yogesh Simmhan (Invited expert), Stian Soiland-Reyes (University of Manchester), Eric Stephan (Pacific Northwest National Laboratory), Linda Stewart (The National Archives), Ed Summers (Library of Congress), Maria Theodoridou (Foundation for Research and Technology - Hellas(FORTH)), Ted Thibodeau (OpenLink Software Inc.), Curt Tilmes (National Aeronautics and Space Administration), Craig Trim (IBM Corporation), Stephan Zednik (Rensselaer Polytechnic Institute), Jun Zhao (University of Oxford), Yuting Zhao (University of Aberdeen, Computing Science).

B. References

B.1 Informative references

[PROV-AQ]
Graham Klyne; Paul Groth; eds. *Provenance Access and Query*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-aq-20130430/

[PROV-CONSTRAINTS]
James Cheney; Paolo Missier; Luc Moreau; eds. *Constraints of the PROV Data Model*. 30 April 2013, W3C Recommendation. URL: http://www.w3.org/TR/2013/REC-prov-constraints-20130430/

[PROV-DC]
Daniel Garijo; Kai Eckert; eds. *Dublin Core to PROV Mapping*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-dc-20130430/

[PROV-DICTIONARY]
Tom De Nies; Sam Coppens; eds. *PROV Dictionary: Modeling Provenance for Dictionary Data Structures*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-dictionary-20130430/

[PROV-DM]
Luc Moreau; Paolo Missier; eds. *PROV-DM: The PROV Data Model*. 30 April 2013, W3C Recommendation. URL: http://www.w3.org/TR/2013/REC-prov-dm-20130430/

[PROV-LINKS]
Luc Moreau; Timothy Lebo; eds. *Linking Across Provenance Bundles*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-links-20130430/

[PROV-N]
Luc Moreau; Paolo Missier; eds. *PROV-N: The Provenance Notation*. 30 April 2013, W3C Recommendation. URL: http://www.w3.org/TR/2013/REC-prov-n-20130430/

[PROV-O]
Timothy Lebo; Satya Sahoo; Deborah McGuinness; eds. *PROV-O: The PROV Ontology*. 30 April 2013, W3C Recommendation. URL: http://www.w3.org/TR/2013/REC-prov-o-20130430/

[PROV-OVERVIEW]
Paul Groth; Luc Moreau; eds. *PROV-OVERVIEW: An Overview of the PROV Family of Documents*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

[PROV-PRIMER]
Yolanda Gil; Simon Miles; eds. *PROV Model Primer*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/

[PROV-XML]
Hook Hua; Curt Tilmes; Stephan Zednik; eds. *PROV-XML: The PROV XML Schema*. 30 April 2013, W3C Note. URL: http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/