Part II of III
Provenance Constraints & Inferences

Paolo Missier, Newcastle University
Khalid Belhajjame, University of Manchester
James Cheney, University of Edinburgh
Overview

- Motivation
  - Basic constraints on provenance records
- Background
  - Data exchange, chase
- Normalization and validity
  - Definitions
  - Inferences
  - Constraints
  - Termination
  - Bundles
- Conclusions
Motivation

- PROV vocabulary allows nonsense
  - "scruffy" provenance

- This is by design: want to encourage uptake
  - making it easy for people to "start small"

- BUT, also want to encourage reasoning
  - so need some principles / rationale underlying reasoning
• WG initially discussed constraints informally as part of PROV-DM

• Question arose:
  – how to enforce?
  – what constraints are "reasonable"?

• Basic principles:
  – constraints / validity must be decidable
  – constraint included only if:
    • clear specification of how to check it
    • no objections/intuitive counterexamples
  – Formal semantics: desirable but not done yet
    • thus, constraints may not exactly match "intuitive" semantics
Background: Data exchange

• PROV-N is (almost) a relational data model...
• Can we just use classic TGDs and EGDs?
• Tuple-generating dependencies:
  – $R(x,y), S(y,z) \rightarrow T(x,z)$
  – $T(x,z) \rightarrow \exists y. R(x,y), S(y,z)$
• Equality-generating dependencies:
  – $R(x,y), R(x,y') \rightarrow y = y'$
• Additional constraints:
  – $R(x,y), R(y,z) \rightarrow y < z$ (ordering)
  – $R(x,y), S(y,x) \rightarrow \bot$ (impossibility)
Complications

• PROV-N is **almost** relational...
  – but not quite

• optional values
  – `activity(a,[type=presentation])`
  – `→ activity(a,_t1,_t2,[type=presentation])`

• missing values
  – `wasAssociatedWith(ag,act,-)`

• attributes: (lightweight) nesting
  – `entity(e,[color=red])`
Variables & Unification

• We allow PROV instances (datasets) to contain *existential variables* (?X)
  – standing for unknown, but present values
  – essentially, labeled nulls in DB-speak

• We can learn (through applying constraints) that an existential variable = some other

• This is called *unification*, and defined as usual in logic / logic programming:
  – ?X =? c ⇒ apply subst ?X = c
  – f(t₁,...,tₙ) =? f(u₁,...,uₙ) ⇒ t₁ =? u₁, ..., tₙ =? uₙ
Validity and normalization

- PROV-CONSTRAINTS defines a notion of valid PROV data
- Definitions: syntactic desugaring
- Inferences: $\approx$ TGDs
- Uniqueness/key constraints: $\approx$ EGDs
- Additional ordering constraints

- Auxiliary notion of normalization
  - $\approx$ TGD/EGD chase
Running Example

entity(e1,[a=1])
entity(e2,[b=2])
specializationOf(e1,e2)

activity(a,t1,-)
used(u;a,e1,t)
wasStartedBy(a,e1,-,-,-)

wasEndedBy(a,e3,-,t2)
**Definitions**: specify how to map arbitrary PROV-N to "core"

**Core means:**
- "nullable" optional parameters explicit
- "non-nullable" optional parameters replaced with variables (aka "labeled nulls")
- all optional attribute lists replaced with []

**Not yet specified**: mappings between PROV-O (RDF), PROV-XML, PROV-N
- Should be cleaned up but scoped out of WG
Example: Expanding definitions

entity(e1,[a=1])
entity(e2,[b=2])
specializationOf(e1,e2)

activity(a,t1,?T2,[])
used(u;a,e1,t,[])
wasStartedBy(?S;a,e1,?A1,?T1,[])

wasEndedBy(?E;a,e3,?A2,t2,[])
Inferences: essentially, TGDs
  – specifying some "implicit" knowledge

Examples:

Inference 7 (entity-generation-invalidation-inference)

\[
\text{IF entity}(e, \_\text{attrs}) \quad \text{THEN} \quad \text{there exist}_\gen, _\al, _\tl, _\inv, _\a2, \text{and}_\t2 \text{such that} \\
\text{wasGeneratedBy}(_\gen; e, _\al, _\tl, []) \quad \text{and} \quad \text{wasInvalidatedBy}(_\inv; e, _\a2, _\t2, []). \\
\]

  – every entity has a start and end event

Inference 5 (communication-generation-use-inference)

\[
\text{IF wasInformedBy}(_\id; a2, a1, _\text{attrs}) \quad \text{THEN} \quad \text{there exist}_e, _\gen, _\tl, _\use, \text{and}_\t2, \text{such that} \\
\text{wasGeneratedBy}(_\gen; e, _\al, _\tl, []) \quad \text{and} \quad \text{used}(_\use; a2, e, _\t2, []) \quad \text{hold.}
\]

Inference 6 (generation-use-communication-inference)

\[
\text{IF wasGeneratedBy}(_\gen; e, _\al, _\tl, _\text{attrs}1) \quad \text{and} \quad \text{used}(_\id2; a2, e, _\t2, _\text{attrs}2) \quad \text{hold} \\
\text{THEN} \quad \text{there exists}_\id \text{such that} \quad \text{wasInformedBy}(_\id; a2, a1, []). \\
\]

  – communication "defined as" generation + use
Example: inferences

```prolog
entity(e1, [a=1, b=2])
entity(e2, [b=2])
specializationOf(e1, e2)
wasGeneratedBy(?G1; e1, ?A1', ?T1', [])
wasGeneratedBy(?G2; e2, ?A2', ?T2', [])
wasInvalidatedBy(?I1; e1, ?A1'', ?T1'', [])
wasInvalidatedBy(?I2; e2, ?A2'', ?T2'', [])
activity(a, t1, ?T2, [])
used(u; a, e1, t, [])
wasStartedBy(?S; a, e1, ?A1, ?T1, [])
wasGeneratedBy(?G1''; e1, ?A1, ?T1'', [])
wasEndedBy(?E; a, e3, ?A2, t2, [])
wasGeneratedBy(?G3; e3, ?A2, ?T3, [])
```
• Constraints:
  – key/uniqueness: merge redundant records (or fail)
  – event-ordering: avoid causal loops
  – typing: check that identifiers have consistent types
    • e.g. nothing is both an entity and an activity
  – impossibility: check that certain impossible things don't occur
Key constraints & merging

• Similar to database key/FD constraints

<table>
<thead>
<tr>
<th>Constraint 22 (key-object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The identifier field ( id ) is a <strong>KEY</strong> for the <code>entity(id,attrs)</code> statement.</td>
</tr>
<tr>
<td>2. The identifier field ( id ) is a <strong>KEY</strong> for the <code>activity(id,t1,t2,attrs)</code> statement.</td>
</tr>
<tr>
<td>3. The identifier field ( id ) is a <strong>KEY</strong> for the <code>agent(id,attrs)</code> statement.</td>
</tr>
</tbody>
</table>

• Subtlety: Attributes
  - this is legal:
    
    \[
    \text{activity}(a,-,t2,[a=1])
    \]
    
    \[
    \text{activity}(a,t1,-,[a=2,b=3])
    \]

  - resolve by **merging**:
    
    \[
    \text{activity}(a,t1,t2,[a=1,a=2,b=3])
    \]
Example: Uniqueness

entity(e1, [a=1, b=2])
entity(e2, [b=2])
specializationOf(e1, e2)
wasGeneratedBy(?G1; e1, ?A1', ?T1', [])
wasGeneratedBy(?G2; e2, ?A2', ?T2', [])
wasInvalidatedBy(?I1; e1, ?A1'', ?T1'', [])
wasInvalidatedBy(?I2; e2, ?A2'', ?T2'', [])
activity(a, t1, t2, [])
used(u; a, e1, t, [])
wasStartedBy(?S; a, e1, ?A1, t1, [])
wasGeneratedBy(?G1'; e1, ?A1, ?T1'', [])
wasEndedBy(?E; a, e3, ?A2, t2, [])
wasGeneratedBy(?G3; e3, ?A2, ?T3, [])
Additional uniqueness constraints

• Additional "vanilla" functional dependencies:

Constraint 24 (unique-generation)

\[
\text{IF wasGeneratedBy}(\text{gen}1; e, a, _t1, _attrs1) \text{ and wasGeneratedBy}(\text{gen}2; e, a, _t2, _attrs2), \text{THEN gen}1 = \text{gen}2.
\]

Constraint 28 (unique-startTime)

\[
\text{IF activity}(a2, _t1, _t2, _attrs) \text{ and wasStartedBy}(_\text{start}; a2, e, _a1, t, _attrs), \text{THEN t}1 = t.
\]

• handled in usual way, by unification
  – which may lead to subsequent merging
  – or failure, if corresponding arguments differ

• Key constraints and uniqueness constraints may fail
  – meaning the PROV data is invalid
Ordering constraints

• Specify that certain **events** happen in a reasonable order
  – Events: generation, use, invalidation, start, end

• Examples: entity lifetime

![Diagram of events](image)

---

**Constraint 36 (generation-precedes-invalidation)**

\[
\text{IF } \text{wasGeneratedBy}(\text{gen}; \text{e}, \_a1, \_t1, \_attrs1) \text{ and } \text{wasInvalidatedBy}(\text{inv}; \text{e}, \_a2, \_t2, \_attrs2) \text{ THEN gen precedes inv.}
\]

**Constraint 37 (generation-precedes-usage)**

\[
\text{IF } \text{wasGeneratedBy}(\text{gen}; \text{e}, \_a1, \_t1, \_attrs1) \text{ and } \text{used}(\text{use}; \_a2, \text{e}, \_t2, \_attrs2) \text{ THEN gen precedes use.}
\]

**Constraint 38 (usage-precedes-invalidation)**

\[
\text{IF } \text{used}(\text{use}; \_a1, \text{e}, \_t1, \_attrs1) \text{ and } \text{wasInvalidatedBy}(\text{inv}; \text{e}, \_a2, \_t2, \_attrs2) \text{ THEN use precedes inv.}
\]
Checking ordering constraints

- Event order is transitive
- There must not be **cycles** involving strict order steps
- Most constraints do not impose strict ordering
  - this is the only strict one:

  **Constraint 42 (derivation-generation-generation-ordering)**

  In this constraint, any of _a, _g, _u MAY be placeholders.

  **IF** wasDerivedFrom(_d; e2,e1,_a,_g,_u,attrs) and wasGeneratedBy(gen1; e1,_a1,_t1,attrs1) and wasGeneratedBy(gen2; e2,_a2,_t2,attrs2) **THEN** gen1 strictly precedes gen2.

- Time ordering does not imply event ordering
  - Applications may infer event ordering from time if they want.
entity(e1,[a=1,b=2])
entity(e2,[b=2])
specializationOf(e1,e2)
wasGeneratedBy(?G1;e1,?A1',?T1',[])
wasGeneratedBy(?G2;e2,?A2',?T2',[])
wasInvalidatedBy(?I1;e1,?A1'',?T1'',[])
wasInvalidatedBy(?I2;e2,?A2'',?T2'',[])
activity(a,t1,t2,[])
used(u;a,e1,t,[])
wasStartedBy(?S;a,e1,?A1,t1,[])
wasGeneratedBy(?G1';e1,?A1,?T1'',[])
wasEndedBy(?E;a,e3,?A2,t2,[])
wasGeneratedBy(?G3;e3,?A2,?T3,[])
Typing constraints

- Specify possible types of identifiers
  - Identifiers can have multiple types
  - but some combinations forbidden

Constraint 50 (typing)

1. IF entity(e, attrs) THEN 'entity' ∈ typeOf(e).
2. IF agent(ag, attrs) THEN 'agent' ∈ typeOf(ag).
3. IF activity(a, t1, t2, attrs) THEN 'activity' ∈ typeOf(a).
4. IF used(u; a, e, t, attrs) THEN 'activity' ∈ typeOf(a) AND 'entity' ∈ typeOf(e).
5. IF wasGeneratedBy(g; e, a, t, attrs) THEN 'entity' ∈ typeOf(e) AND 'activity' ∈ typeOf(a).
6. IF wasInformedBy(i; a2, a1, t, attrs) THEN 'activity' ∈ typeOf(a2) AND 'activity' ∈ typeOf(a1).
7. IF wasStartedBy(i; a2, e, a1, t, attrs) THEN 'activity' ∈ typeOf(a2) AND 'entity' ∈ typeOf(e) AND 'activity' ∈ typeOf(a1).
8. IF wasEndedBy(i; a2, e, a1, t, attrs) THEN 'activity' ∈ typeOf(a2) AND 'entity' ∈ typeOf(e) AND 'activity' ∈ typeOf(a1).
9. IF wasInvalidatedBy(i; e, a, t, attrs) THEN 'entity' ∈ typeOf(e) AND 'activity' ∈ typeOf(a).
10. IF wasDerivedFrom(i; e2, e1, a, g2, u1, attrs) THEN 'entity' ∈ typeOf(e2) AND 'entity' ∈ typeOf(e1) AND 'activity' ∈ typeOf(a). In this constraint, a, g2, and u1 MUST NOT be placeholders.
11. IF wasAttributedTo(i; e, a, g, attrs) THEN 'entity' ∈ typeOf(e) AND 'agent' ∈ typeOf(a).
12. IF wasAssociatedWith(i; a, a1, attrs) THEN 'activity' ∈ typeOf(a) AND 'agent' ∈ typeOf(a1) AND 'activity' ∈ typeOf(a). In this constraint, a1 MUST NOT be a placeholder.
13. IF actedOnBehalfOf(i; a2, a1, attrs) THEN 'agent' ∈ typeOf(a2) AND 'agent' ∈ typeOf(a1) AND 'activity' ∈ typeOf(a).
14. IF alternateOf(e2, e1) THEN 'entity' ∈ typeOf(e2) AND 'entity' ∈ typeOf(e1).
15. IF specializationOf(e2, e1) THEN 'entity' ∈ typeOf(e2) AND 'entity' ∈ typeOf(e1).
16. IF hadMember(c, e) THEN 'prov:Collection' ∈ typeOf(c) AND 'entity' ∈ typeOf(c) AND 'entity' ∈ typeOf(e).
17. IF entity(c, [prov:type='prov:EmptyCollection']) THEN 'entity' ∈ typeOf(c) AND 'prov:Collection' ∈ typeOf(c) AND 'prov:EmptyCollection' ∈ typeOf(c).
Impossibility constraints

- Property and object ids disjoint

   **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:

   IF \( r(id; a_1, \ldots, a_m) \) and \( s(id; b_1, \ldots, b_n) \) THEN INVALID.

- Different property ids disjoint

   **Constraint 54 (impossible-object-property-overlap)**

   For each \( p \) in \{entity, activity or agent\} and for each \( r \) in \{used, wasGeneratedBy, wasInvalidatedBy, wasInfluencedBy, wasStartedBy, wasEndedBy, wasInformedBy, wasDerivedFrom, wasAttributedTo, wasAssociatedWith, actedOnBehalfOf\}, the following impossibility constraint holds:

   IF \( p(id, a_1, \ldots, a_m) \) and \( r(id; b_1, \ldots, b_n) \) THEN INVALID.

- Entities and activities disjoint

   **Constraint 55 (entity-activity-disjoint)**

   IF 'entity' \( \in \) typeof(id) AND 'activity' \( \in \) typeof(id) THEN INVALID.
Validation

- **Step 1: Expand definitions**
  - removing syntactic sugar; expanding optional parameters to existential variables

- **Step 2: Apply inferences & key/uniqueness constraints**
  - adding information, or merging/unifying to remove redundant information
  - this may fail if there are inconsistencies

- **Step 3: Check ordering, typing and impossibility constraints**
  - on the "normal form" obtained by step 2
  - (i.e. universal instance)
Termination

- Normalization is (essentially) TGD/EGD chase
  - Does not terminate in general!
  - But terminating classes are known
    - weak acyclicity [Fagin, Kolaitis, Miller, Popa ICDT 2003; TCS 2005]
    - others from Datalog $^{\exists}$ [Leone et al.], Datalog $^{\pm}$ [Cali et al.]
  - We can prove termination by stratification:

<table>
<thead>
<tr>
<th>Stage #</th>
<th>Inference</th>
<th>Hypotheses</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19, 20, 21</td>
<td>specializationOf</td>
<td>specializationOf, entity</td>
</tr>
<tr>
<td>2</td>
<td>7, 8, 13, 14</td>
<td>entity, activity, wasAttributedTo, actedOnBehalfOf</td>
<td>wasInvalidatedBy, wasStartedBy, wasEndedBy, wasAssociatedWith</td>
</tr>
<tr>
<td>3</td>
<td>9, 10</td>
<td>wasStartedBy, wasEndedBy</td>
<td>wasGeneratedBy</td>
</tr>
<tr>
<td>4</td>
<td>11, 12</td>
<td>wasDerivedFrom</td>
<td>wasGeneratedBy, used, alternateOf</td>
</tr>
<tr>
<td>5</td>
<td>16, 17, 18</td>
<td>alternateOf, entity</td>
<td>alternateOf</td>
</tr>
<tr>
<td>6</td>
<td>5, 6</td>
<td>wasInformedBy, generated, used</td>
<td>wasInformedBy, generated, used</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>many</td>
<td>wasInfluencedBy</td>
</tr>
</tbody>
</table>
- Stages 1, 5: Datalog (hence w.a.)
- Stages 2, 3, 4, 7: hypotheses disjoint from conclusions (hence w.a.)
- Stage 6: w.a. by test from [FKMP '05]
Bundles

- PROV-CONSTRAINTS mostly concerns *instances*
  - corresponding to a single perspective/description
- PROV documents can contain multiple instances, including named bundles
- Validity is "pointwise"
  - bundle names have to be distinct
- Future work: possible relations that link across bundles
  - semantics unclear; appropriate inferences/constraints also unclear.
  - modal logic? reasoning about contexts?
• PROV-CONSTRAINTS is:
  – a set of community-agreed rules for validating provenance (yawn?)
  – a real-world application of classic database concepts
    • techniques from data exchange turned out to be exactly what was needed [FKMP 2003;2005]
    • techniques from Datalog\(^3\) / Datalog\(^\pm\) may also be useful in future
  – Open questions:
    • efficient / maintainable / extensible implementation?
    • translation of relational constraints/inferences to OWL/ontologies/Datalog engines?
    • relating relational & ontology-style presentations of data (round-tripping valid PROV-O/PROV-N/PROV-XML)?