Part II of III
Provenance Constraints & Inferences

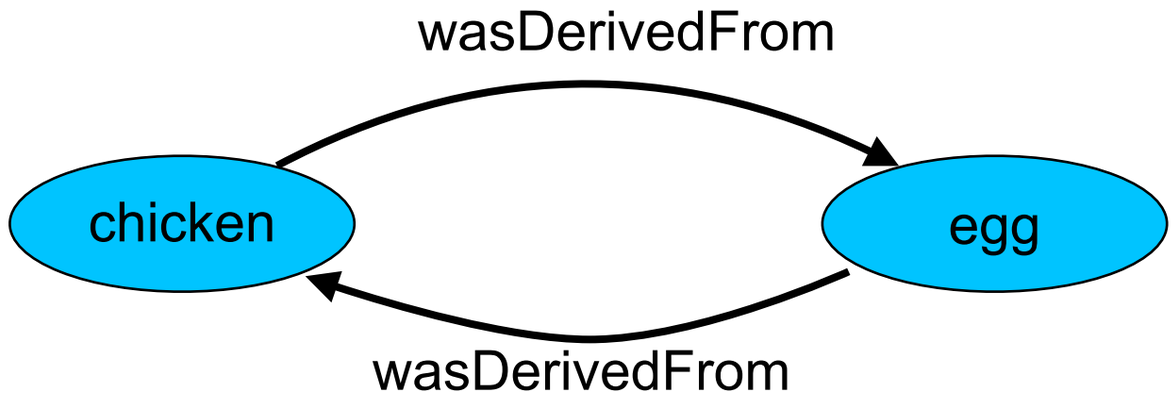
Paolo Missier, Newcastle University
Khalid Belhajjame, University of Manchester
James Cheney, University of Edinburgh

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

Process

- 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:
  - $?X =_? c \Longrightarrow$ apply subst ?X = c
  - $?X =_? ?Y \Longrightarrow$ apply subst ?X = ?Y
  - $f(t_1,...,t_n) =_? f(u_1,...,u_n) \Longrightarrow t_1 =_? u_1, ..., t_n =_? u_n$

Validity and normalization

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

- Auxiliary notion of *normalization*
  - ≈ 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

- **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

- Inferences: essentially, TGDs
  - specifying some "implicit" knowledge
- Examples:

Inference 7 (entity-generation-invalidation-inference)

IF `entity(e,_attrs)` THEN there exist `_gen`, `_a1`, `_t1`, `_inv`, `_a2`, and `_t2` such that `wasGeneratedBy(_gen; e,_a1,_t1,[])` and `wasInvalidatedBy(_inv; e,_a2,_t2,[])`.

  - every entity has a start and end event

Inference 5 (communication-generation-use-inference)

IF `wasInformedBy(_id; a2,a1,_attrs)` THEN there exist `e`, `_gen`, `_t1`, `_use`, and `_t2`, such that `wasGeneratedBy(_gen; e,a1,_t1,[])` and `used(_use; a2,e,_t2,[])` hold.

Inference 6 (generation-use-communication-inference)

IF `wasGeneratedBy(_gen; e,a1,_t1,_attrs1)` and `used(_id2; a2,e,_t2,_attrs2)` hold THEN there exists `_id` such that `wasInformedBy(_id; a2,a1,[])`

  - communication "defined as" generation + use

Example: inferences

```
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

- 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

Constraint 22 (key-object)

1. The identifier field `id` is a KEY for the `entity(id,attrs)` statement.
2. The identifier field `id` is a KEY for the `activity(id,t1,t2,attrs)` statement.
3. The identifier field `id` is a KEY for the `agent(id,attrs)` statement.

- Subtlety: Attributes
  - this is legal:

```
activity(a,-,t2,[a=1])
activity(a,t1,-,[a=2,b=3])
```

  - resolve by **merging**:
  - `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,[])
```

?T1=t1
?T2=t2

Additional uniqueness constraints

- Additional "vanilla" functional dependencies:

Constraint 24 (unique-generation)

IF `wasGeneratedBy(gen1; e,a,_t1,_attrs1)` and `wasGeneratedBy(gen2; e,a,_t2,_attrs2)`, THEN `gen1 = gen2`.

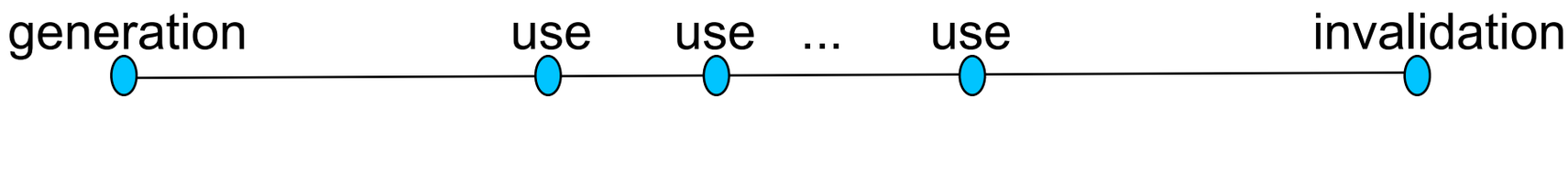
Constraint 28 (unique-startTime)

IF `activity(a2,t1,_t2,_attrs)` and `wasStartedBy(_start; a2,_e,_a1,t,_attrs)`, THEN `t1=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



Constraint 36 (generation-precedes-invalidation)

IF `wasGeneratedBy(gen; e,_a1,_t1,_attrs1)` and `wasInvalidatedBy(inv; e,_a2,_t2,_attrs2)` THEN `gen` precedes `inv`.

Constraint 37 (generation-precedes-usage)

IF `wasGeneratedBy(gen; e,_a1,_t1,_attrs1)` and `used(use; _a2,e,_t2,_attrs2)` THEN `gen` precedes `use`.

Constraint 38 (usage-precedes-invalidation)

IF `used(use; _a1,e,_t1,_attrs1)` and `wasInvalidatedBy(inv; e,_a2,_t2,_attrs2)` 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)

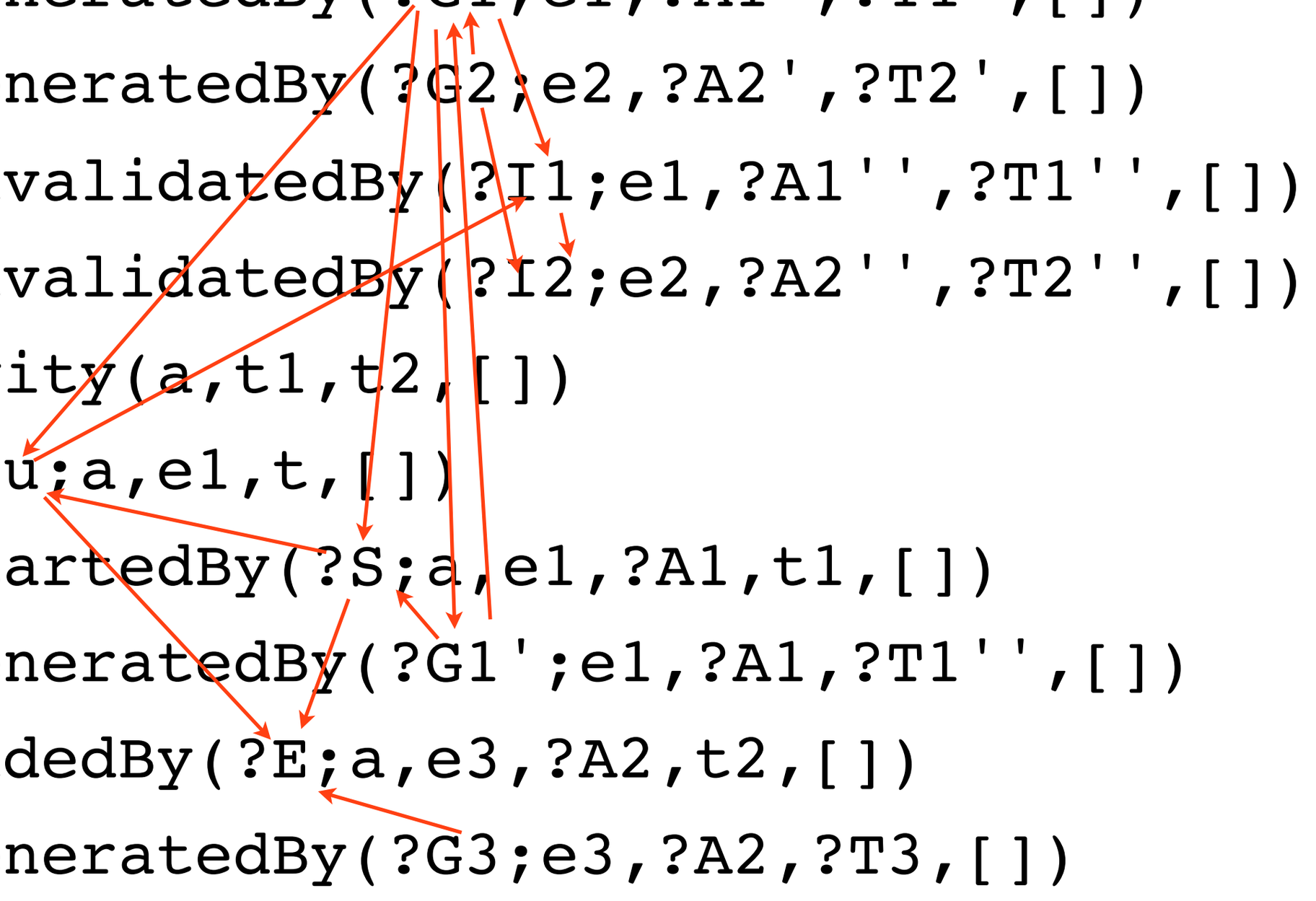
In this constraint, any of `_a`, `_g`, `_u` MAY be placeholders.

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.

Example: Ordering

```
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(gen; e,a,t,attrs)` THEN `'entity' ∈ typeOf(e)` AND `'activity' ∈ typeOf(a)`.
6. IF `wasInformedBy(id; a2,a1,attrs)` THEN `'activity' ∈ typeOf(a2)` AND `'activity' ∈ typeOf(a1)`.
7. IF `wasStartedBy(id; a2,e,a1,t,attrs)` THEN `'activity' ∈ typeOf(a2)` AND `'entity' ∈ typeOf(e)` AND `'activity' ∈ typeOf(a1)`.
8. IF `wasEndedBy(id; a2,e,a1,t,attrs)` THEN `'activity' ∈ typeOf(a2)` AND `'entity' ∈ typeOf(e)` AND `'activity' ∈ typeOf(a1)`.
9. IF `wasInvalidatedBy(id; e,a,t,attrs)` THEN `'entity' ∈ typeOf(e)` AND `'activity' ∈ typeOf(a)`.
10. IF `wasDerivedFrom(id; 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 `wasDerivedFrom(id; e2, e1, -, -, -, attrs)` THEN `'entity' ∈ typeOf(e2)` AND `'entity' ∈ typeOf(e1)`.
12. IF `wasAttributedTo(id; e,ag,attr)` THEN `'entity' ∈ typeOf(e)` AND `'agent' ∈ typeOf(ag)`.
13. IF `wasAssociatedWith(id; a,ag,pl,attrs)` THEN `'activity' ∈ typeOf(a)` AND `'agent' ∈ typeOf(ag)` AND `'entity' ∈ typeOf(pl)`. In this constraint, `pl` MUST NOT be a placeholder.
14. IF `wasAssociatedWith(id; a,ag,-,attrs)` THEN `'activity' ∈ typeOf(a)` AND `'agent' ∈ typeOf(ag)`.
15. IF `actedOnBehalfOf(id; ag2,ag1,a,attrs)` THEN `'agent' ∈ typeOf(ag2)` AND `'agent' ∈ typeOf(ag1)` AND `'activity' ∈ typeOf(a)`.
16. IF `alternateOf(e2, e1)` THEN `'entity' ∈ typeOf(e2)` AND `'entity' ∈ typeOf(e1)`.
17. IF `specializationOf(e2, e1)` THEN `'entity' ∈ typeOf(e2)` AND `'entity' ∈ typeOf(e1)`.
18. IF `hadMember(c,e)` THEN `'prov:Collection' ∈ typeOf(c)` AND `'entity' ∈ typeOf(c)` AND `'entity' ∈ typeOf(e)`.
19. 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 $r(id; a_1,...,a_m)$ and $s(id; b_1,...,b_n)$ 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 $p(id,a_1,...,a_m)$ and $r(id; b_1,...,b_n)$ THEN INVALID.

- Entities and activities disjoint

Constraint 55 (entity-activity-disjoint)

IF `'entity' ∈ typeOf(id)` AND `'activity' ∈ 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

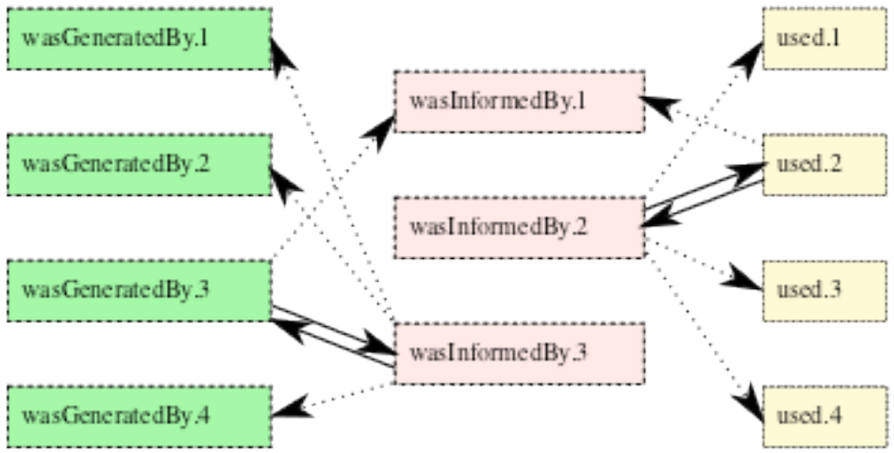
- Normalization is (essentially) TGD/EGD chase
  - Does not terminate in general!
  - But terminating classes are known
    - **weak acyclicity** [Fagin, Kolaitis, Miller, Popa ICDT 2003; TCS 2005]
    - others from Datalog$^{\exists}$ [Leone et al.], Datalog$^{\pm}$ [Cali et al.]
  - We can prove termination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

Stage #	Inference	Hypotheses	Conclusions
1	19, 20, 21	`specializationOf`	`specializationOf, entity`
2	7, 8, 13, 14	`entity, activity, wasAttributedTo, actedOnBehalfOf`	`wasInvalidatedBy, wasStartedBy, wasEndedBy, wasAssociatedWith`
3	9, 10	`wasStartedBy, wasEndedBy`	`wasGeneratedBy`
4	11, 12	`wasDerivedFrom`	`wasGeneratedBy, used, alternateOf`
5	16, 17, 18	`alternateOf, entity`	`alternateOf`
6	5, 6	`wasInformedBy, generated, used`	`wasInformedBy, generated, used`
7	15	many	`wasInfluencedBy`

Termination (2)

Stage #	Inference	Hypotheses	Conclusions
1	19, 20, 21	specializationOf	specializationOf, entity
2	7, 8, 13, 14	entity, activity, wasAttributedTo, actedOnBehalfOf	wasInvalidatedBy, wasStartedBy, wasEndedBy, wasAssociatedWith
3	9, 10	wasStartedBy, wasEndedBy	wasGeneratedBy
4	11, 12	wasDerivedFrom	wasGeneratedBy, used, alternateOf
5	16, 17, 18	alternateOf, entity	alternateOf
6	5, 6	wasInformedBy, generated, used	wasInformedBy, generated, used
7	15	many	wasInfluencedBy

- 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?

Conclusions

- 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$^{\exists}$ / 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 (roundtripping valid PROV-O/PROV-N/PROV-XML)?