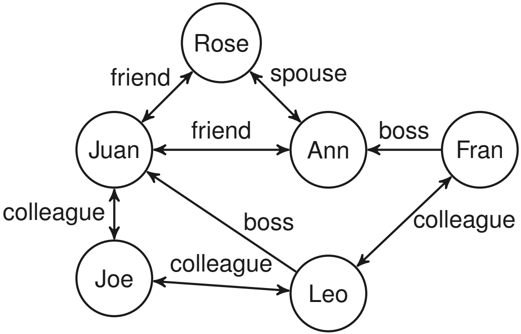
A Recursive Approach for Defining Reachability Queries over Graph Databases

Graph Structured Data is now everywhere



- Facebook, Twitter
- DBPedia (Wikipedia represented as a graph)
- Biological databases, geological databases, social databases...

Web of Data needs graphs

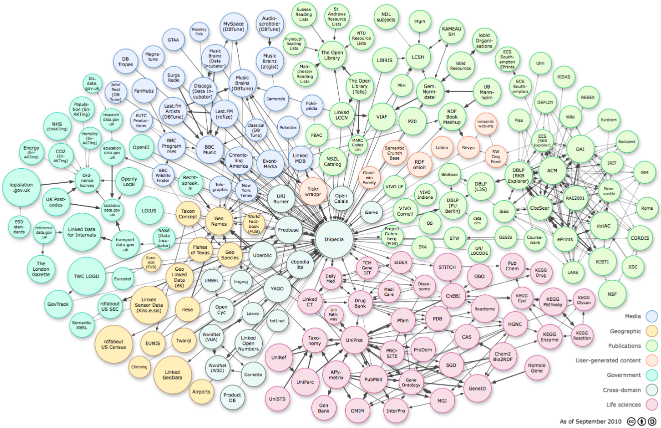
Semantic Web:

Integrate semantic content to web resources

Semantic Web (some projects)

KEGG
Kyoto Encyclopedia of Genes and Genomes
DBpedia
Linked GeoData
PubMed
United States™ Census 2010
OPEN LIBRARY
CiteSeer.IST
Scientific Literature Digital Library

Aerial View of Linked Data sources

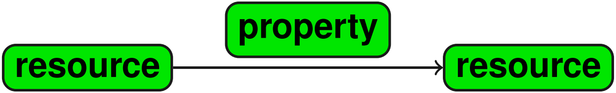


Semantic Web $\longrightarrow$ Graph Database

Web resources are modeled as nodes of a graph,
edges are relations between resources

Semantic Web ⟶ Graph Database

Web resources are modeled as nodes of a graph,
edges are relations between resources



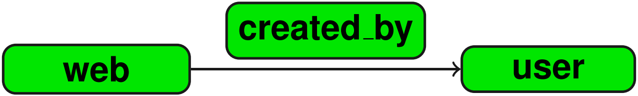
property

resource

resource

Semantic Web ⟶ Graph Database

Web resources are modeled as nodes of a graph,
edges are relations between resources



Querying Semantic Web:

SPARQL:

SPARQL Protocol and RDF Query Language

- Standardized query language for RDF
- Query language with strong relational flavor

Querying Semantic Web:

SPARQL:

SPARQL Protocol and RDF Query Language

- Standardized query language for RDF
- Query language with strong relational flavor

- Think of Relational Algebra or SQL over graphs (selection, projection, joins, unions, etc...)

Connectivity Queries:
a novel challenge

Talk about paths in graphs

Connectivity Queries: a novel challenge

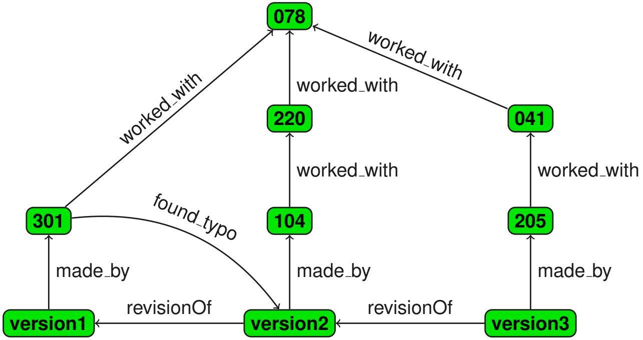
Talk about paths in graphs

- Is node A connected to node B?
- Is it connected by a path satisfying certain properties?
- Other possibilities: return (simple) paths, aggregation, etc.

Connectivity Queries:
a novel challenge

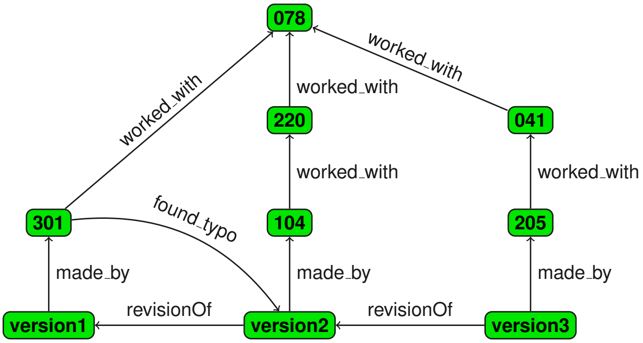
- In a social network: Am I connected to a superstar?
- Workflows (biological):
  how is the path from process A to process B?
- Maps: shortest path form A to B?

Example: Provenance of a DBPedia Article



- Users who might have made a typo:

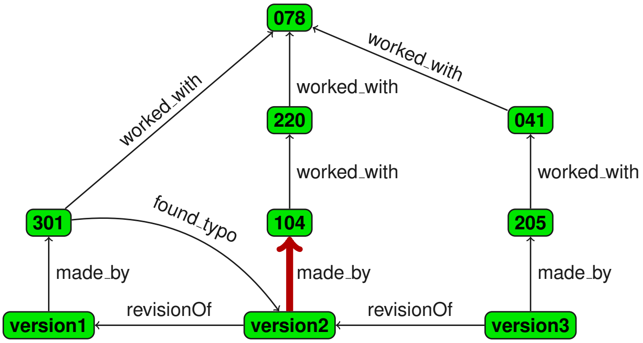
Example: Provenance of a DBPedia Article



- Users who might have made a typo:

104, 220, 078

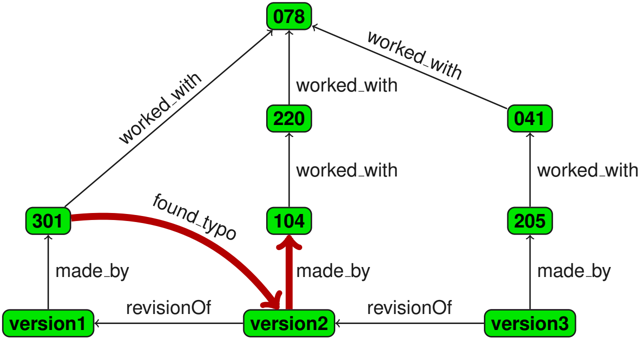
Example: Provenance of a DBPedia Article



- Users who might have made a typo:

104, 220, 078

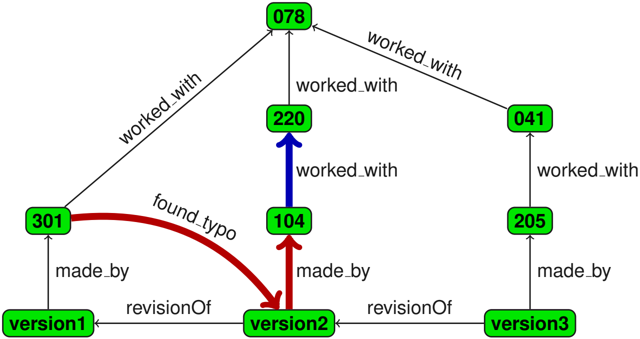
Example: Provenance of a DBPedia Article



- Users who might have made a typo:

104, 220, 078

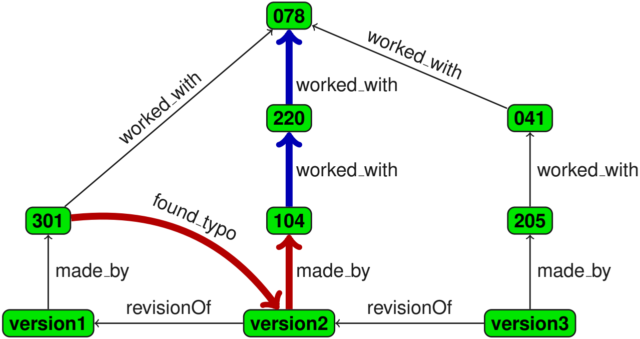
Example: Provenance of a DBPedia Article



- Users who might have made a typo:

104, 220, 078

Example: Provenance of a DBPedia Article



- Users who might have made a typo:

104, 220, 078

This talk:

Connectivity queries for Semantic Web databases

Juan L. Reutter

DataLab
Dept. of Computer Science
PUC Chile

Outline

1. Graph databases
   Most studied connectivity queries

2. Relationship between connectivity and recursion
   An algebra for querying graphs

3. Where to go from here

Outline

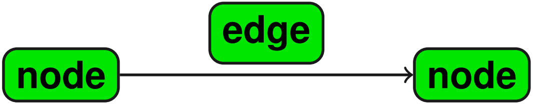
1. Graph databases
   Most studied connectivity queries

2. Relationship between connectivity and recursion
   An algebra for querying graphs

3. Where to go from here

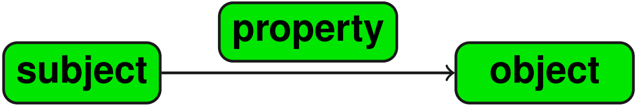
Graph databases (for this talk):

We abstract them as triples:



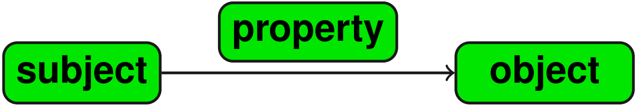
Graph databases (for this talk):

We abstract them as triples:



Graph databases (for this talk):

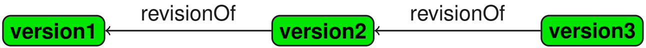
We abstract them as triples:



A graph database is a collection of triples.

RDF graphs can be represented by relational databases.

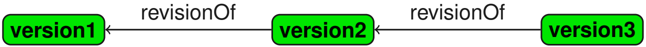
RDF graphs can be represented by relational databases.



Relational Representation:

Subj.	Prop.	Obj.
version3	revisionOf	version2
version2	revisionOf	version1

RDF graphs can be represented by relational databases.

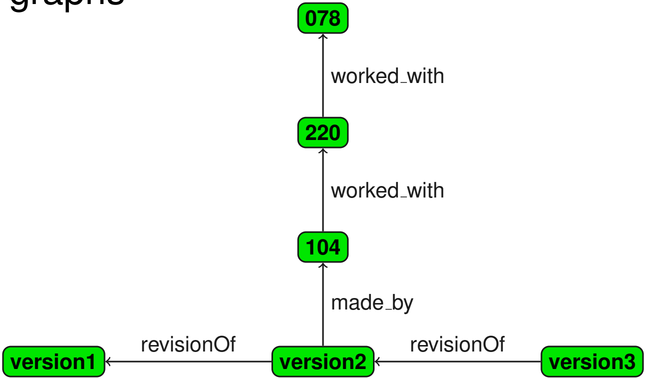


Relational Representation:

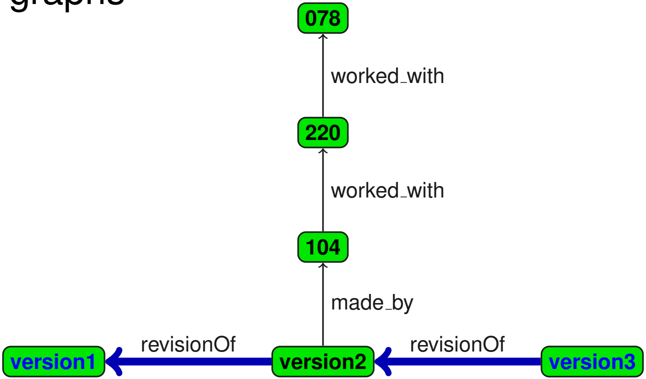
Subj.	Prop.	Obj.
version3	revisionOf	version2
version2	revisionOf	version1

The distinction is in the queries that we want to ask
(such as connectivity queries)

Paths in graphs



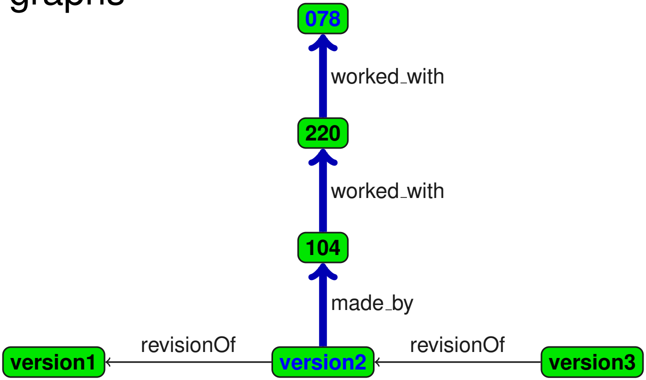
Paths in graphs



The label of the path is

$$\text{revisionOf} \cdot \text{revisionOf}$$

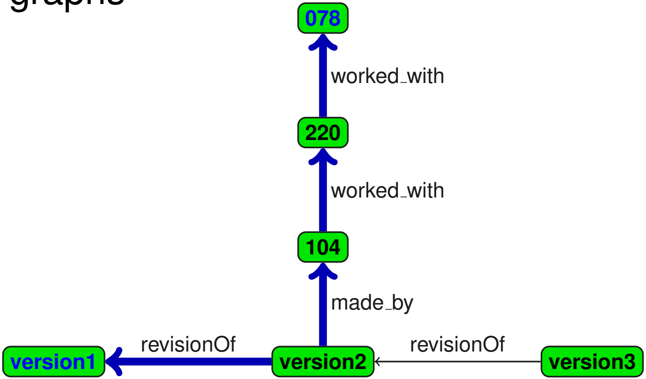
Paths in graphs



The label of the path is

made_by · worked_with · worked_with

Paths in graphs



The label of the path is

$$\text{revisionOf}^{-} \cdot \text{made\_by} \cdot \text{worked\_with} \cdot \text{worked\_with}$$

Regular Path Queries (RPQs)

- One of the most studied connectivity query languages
- Part of SPARQL (query language for RDF)
- Most graph DB systems claim to support it

Regular Path Queries (RPQs)

- One of the most studied connectivity query languages
- Part of SPARQL (query language for RDF)
- Most graph DB systems claim to support it

Idea:
Check paths that are given by a certain regular expression

Regular Path Queries (RPQs), formally

Let Σ be a set of properties

RPQs are of the form

$$(x, e, y)$$

Where e is a regular expression over Σ.

Example in Social Networks

$$(x, \text{works\_with}^*, y),$$

finds all people that are connected via co-worker relationship.

Example in Crime Networks

$$(x, \mathrm{reports}^{+}, \mathit{Boss})$$

finds all people that reports directly or indirectly to the Boss.

Example in Crime Networks

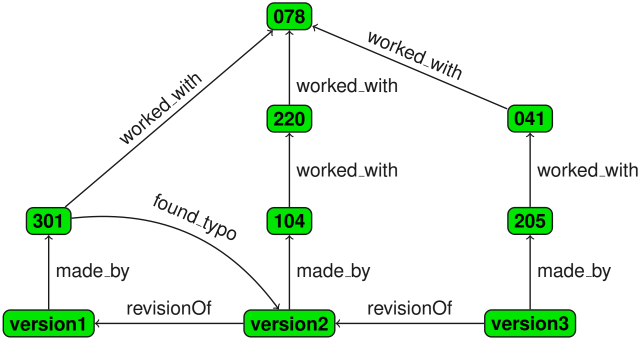
$$(x, \text{reports}^{+}, \mathit{Boss})$$

finds all people that reports directly or indirectly to the Boss.

$$(x, \Sigma^{*} \cdot \text{reports} \cdot \Sigma^{*} \cdot \text{reports} \cdot \Sigma^{*}, \mathit{Boss})$$

finds all people connected to Boss by a path that runs via at least two intermediaries.

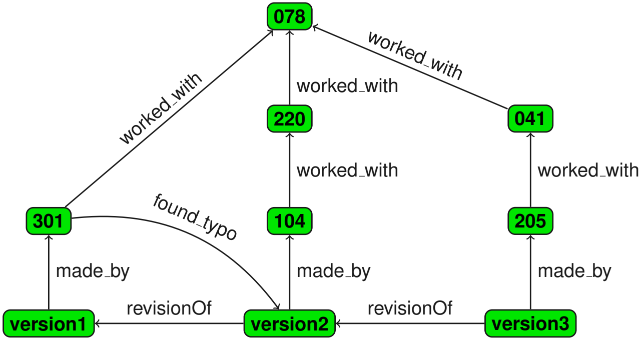
Regular Path Queries



- Revision and people involved

$$made\_by \cdot (worked\_with)^*$$

Regular Path Queries

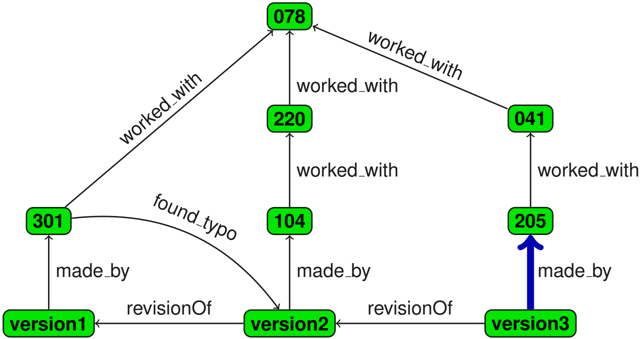


- Revision and people involved

$$\textit{made\_by} \cdot (\textit{worked\_with})^*$$

078 was involved in version3.

Regular Path Queries

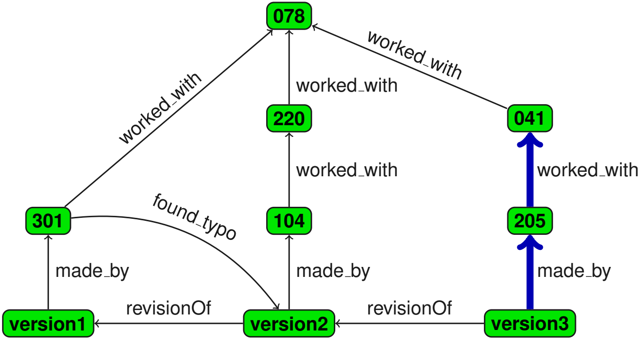


- Revision and people involved

$$\mathit{made\_by} \cdot (\mathit{worked\_with})^*$$

078 was involved in version3.

Regular Path Queries

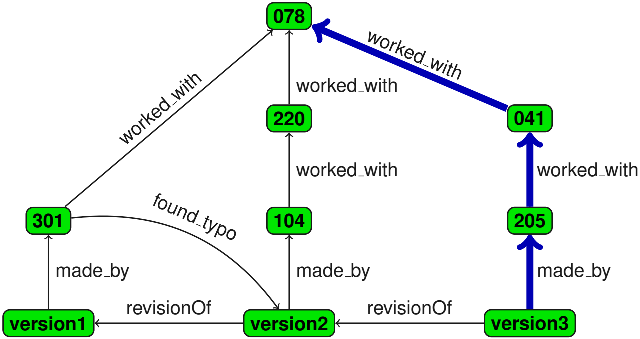


- Revision and people involved

$$made\_by \cdot (worked\_with)^*$$

078 was involved in version3.

Regular Path Queries

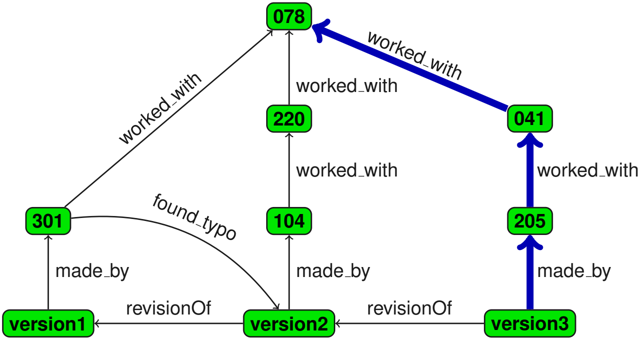


- Revision and people involved

$$made\_by \cdot (worked\_with)^*$$

078 was involved in version3.

Regular Path Queries



- Revision and people involved

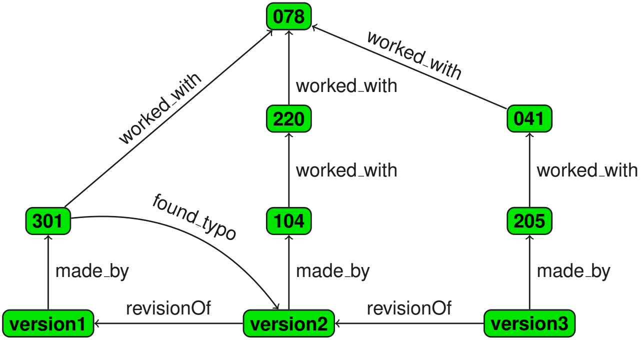
$$\mathit{made\_by} \cdot (\mathit{worked\_with})^*$$

078 was involved in version3.

RPQs is used as a primitive for querying paths

- Supported by various systems
- Simple, declarative language (easy to state what you want)
- Really efficient evaluation:
  Automata techniques allow to use fast reachability algorithms to evaluate RPQs.
- Lots of extensions

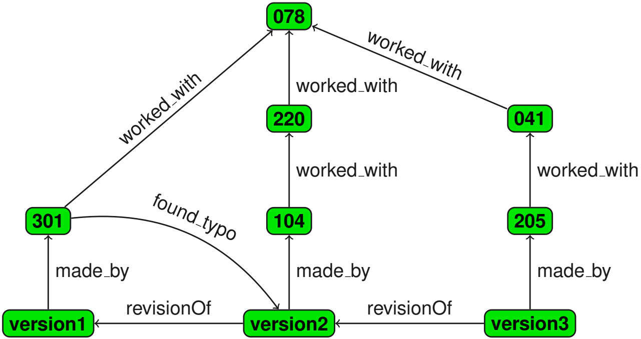
Nested Regular Expressions



- People who might have made a typo:

$$[\mathit{made\_by}^{-} \cdot \mathit{found\_typo}^{-}] \cdot (\mathit{worked\_with})^{*}$$

Nested Regular Expressions

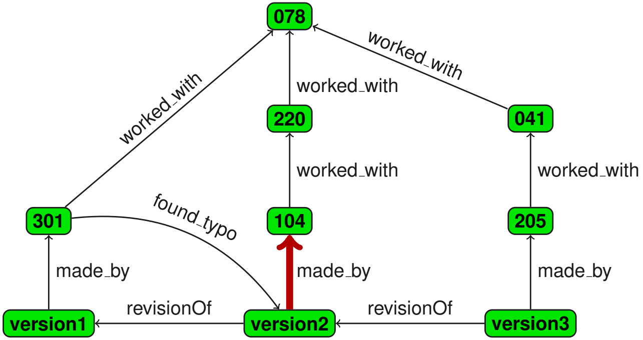


- People who might have made a typo:

$$[made\_by^{-} \cdot found\_typo^{-}] \cdot (worked\_with)^{*}$$

104, 220, 078

Nested Regular Expressions

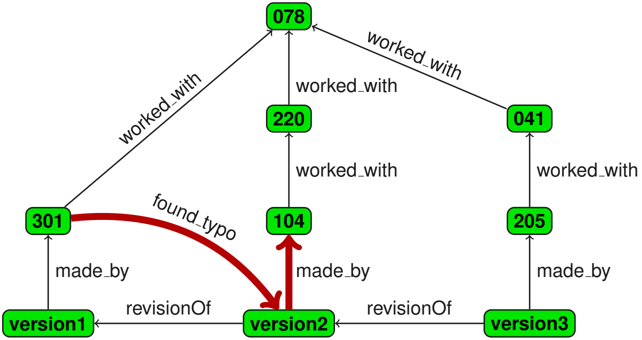


- People who might have made a typo:

$$[made\_by^{-} \cdot found\_typo^{-}] \cdot (worked\_with)^{*}$$

104, 220, 078

Nested Regular Expressions

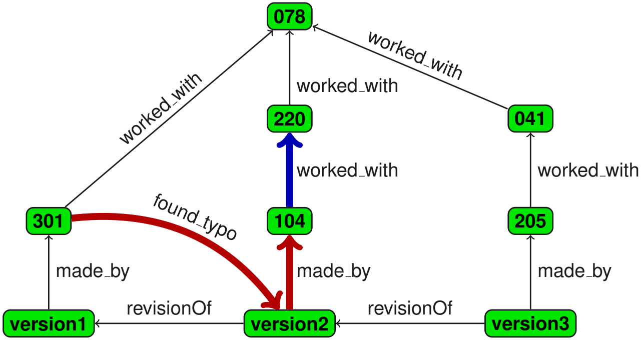


- People who might have made a typo:

$$[\mathit{made\_by}^{-} \cdot \mathit{found\_typo}^{-}] \cdot (\mathit{worked\_with})^{*}$$

104, 220, 078

Nested Regular Expressions

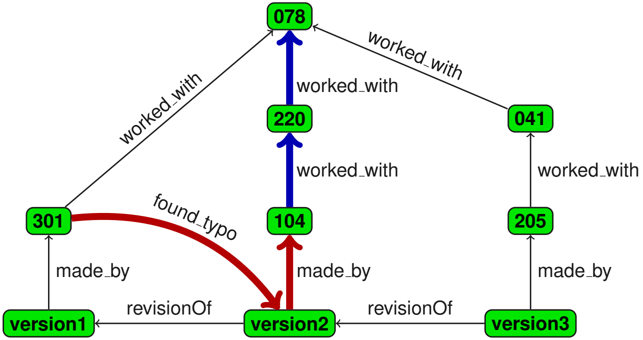


- People who might have made a typo:

$$[\mathit{made\_by}^- \cdot \mathit{found\_typo}^-] \cdot (\mathit{worked\_with})^*$$

104, 220, 078

Nested Regular Expressions



- People who might have made a typo:

$$[made\_by^{-} \cdot found\_typo^{-}] \cdot (worked\_with)^{*}$$

104, 220, 078

Outline

1. Graph databases
   Most studied connectivity queries

2. Relationship between connectivity and recursion
   An algebra for querying graphs

3. Where to go from here

In practice, things are not that simple

Graph DB systems struggle to support RPQs.

- Neo4j only supports concatenation, star

In practice, things are not that simple

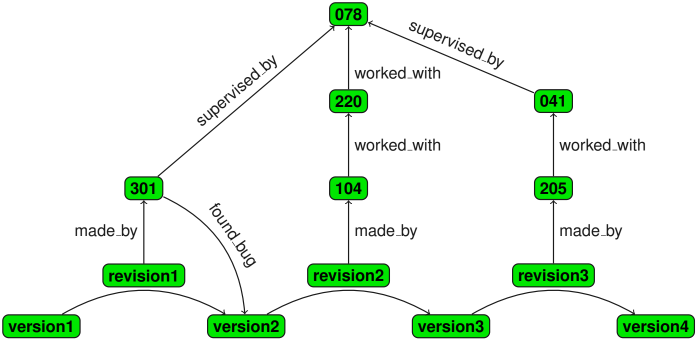
While RPQs are now part of SPARQL

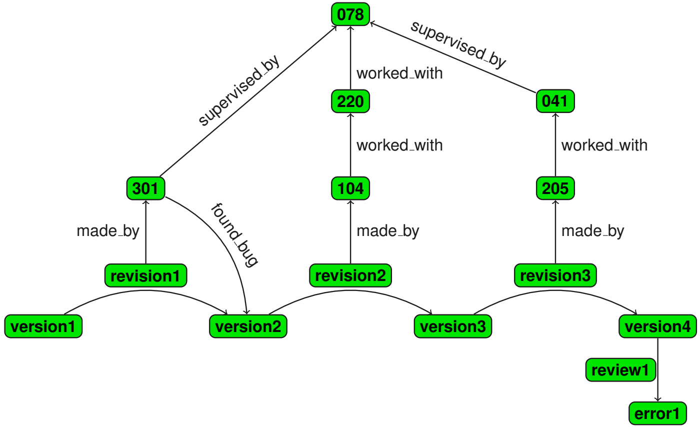
- Semantics not clearly defined
  (many changes in last years)
- no clear guidelines for implementation

Problem with RPQs: limited expressivity

to study it we use SPARQL (with RPQs)

more or less like SQL + RPQs


078
supervised_by
worked_with
supervised_by
220
041
worked_with
worked_with
301
104
205
found_bug
made_by
made_by
made_by
revision1
revision2
revision3
version1
version2
version3
version4



078

supervised_by

worked_with

supervised_by

220

041

worked_with

worked_with

301

104

205

made_by

found_bug

made_by

made_by

revision1

revision2

revision3

version1

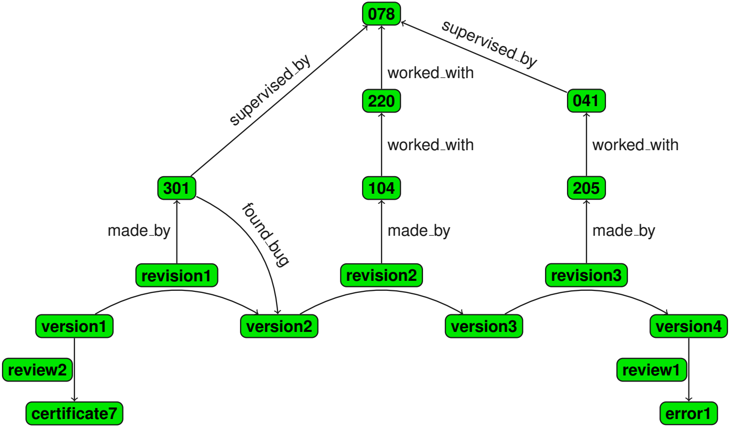
version2

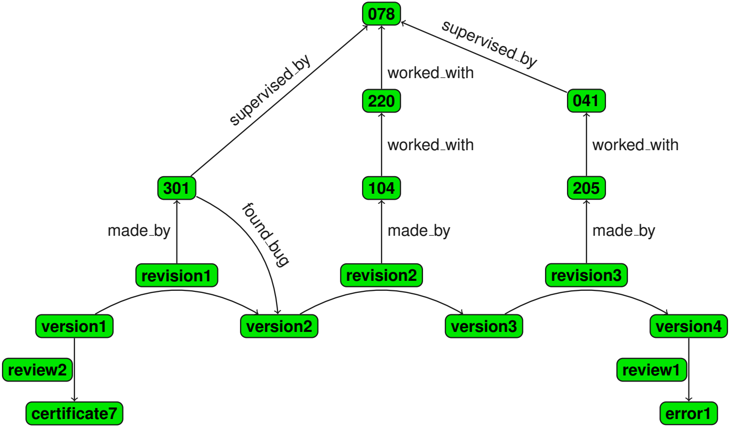
version3

version4

review1

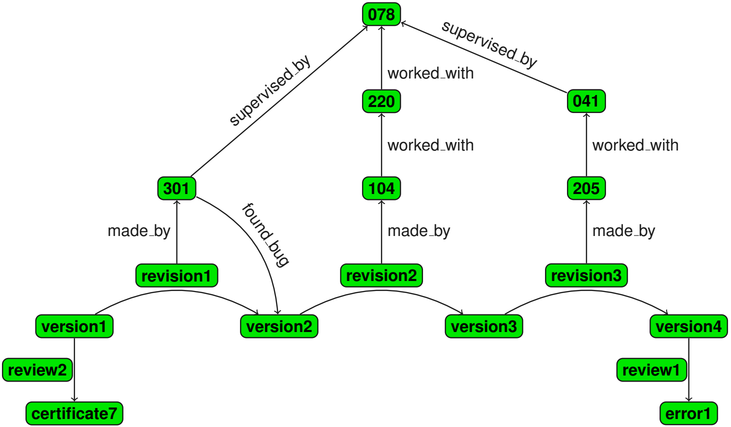
error1

078
supervised_by
supervised_by
worked_with
220
041
worked_with
worked_with
301
104
205
found_bug
made_by
made_by
made_by
revision1
revision2
revision3
version1
version2
version3
version4
review2
review1
certificate7
error1



Q_1 :

Find all versions with an error
That originate from a valid version
And return the latest revision



Q_2 : Find all versions with an error
That originate from a valid version
And the person responsible for this

What is the problem?

- Q_1 can be expressed using SPARQL (essentially SQL + RPQs)
- Q_2 can NOT be expressed in SPARQL
- Conclusion:
  - Need to reason while moving along paths
  - Cant do this using RPQs or similar primitives

Problem with RPQs: implementation

- Simple and efficient implementation using automata theory

Problem with RPQs: implementation

Traditionally, DB systems do not implement techniques that rely on automata theory.

- XML and XPath

Why not base implementation on relational queries

Next

Understanding connectivity queries

from a relational point of view

Next

Understanding connectivity queries

from a relational point of view

What are RPQs? What does (x, a^*, y) means?

- Let $S \circ S'$ be the composition of binary relations S and S' :

$$S \circ S' = \{(x, z) \mid (x, y) \in S \ \wedge (y, z) \in S'\}$$

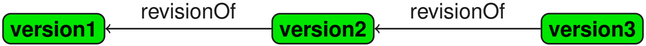
What does (x, a^*, y) means?

- Take binary relation A given by all x, y that are connected via label a in the graph.

$$(x, a^+, y) = A \cup A \circ A \cup A \circ A \circ A \cup \ldots$$

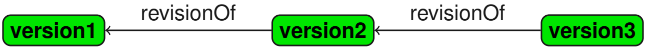
Compose A with itself over and over again... until we reach a fixed point.

$(x, \text{revisionOf}^*, y)$



R	Prop.	Obj.
	version3	version2
	version2	version1

$(x, \text{revisionOf}^*, y)$

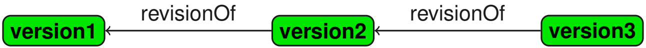


R	Prop.	Obj.
	version3	version2
	version2	version1

$(x, \text{revisionOf}^*, y) =$

R^*	Prop.	Obj.
	version3	version2
	version2	version1
	version3	version1

$(x, \text{revisionOf}^{*}, y)$



R	Prop.	Obj.
	version3	version2
	version2	version1

$(x, \text{revisionOf}^{*}, y) =$

R^{*}	Prop.	Obj.
	version3	version2
	version2	version1
	version3	version1
	version3	version3
	version2	version2
	version1	version1

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

$$S(x_1, x_2) \qquad S'(x_{1'}, x_{2'})$$

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

$$S(x_1, x_2) \qquad S'(x_{1'}, x_{2'})$$

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

$$S(x_1, x) \qquad S'(x, x_{2'})$$

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

$$S(x_1, x) \qquad S'(x, x_{2'})$$

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

$$S(x_1, x) \qquad S'(x, x_{2'})$$

But from the eyes of relational algebra...

Composition is just a join!

$$S \circ S' = S \underset{2=1'}{\overset{1,2'}{\bowtie}} S'$$

Thus reachability is just recursive iteration of joins.

TriAL: an algebra for graphs

We now define an algebra for triples, that:

- can express RPQs
- can even express queries such as Q2
- Is based on relational algebra

Composing ternary relation

We need to manage triples...
No obvious way to do it

$$(x, y, x) \quad \circ \quad (x', y', z')$$

Composing ternary relation

We need to manage triples...
No obvious way to do it

$$(x, y, x) \quad \circ \quad (x', y', z')$$

We take the approach of relational algebra, and define all possible compositions.

- Triple joins

- Triple joins

$$R \quad \bowtie \quad R'$$

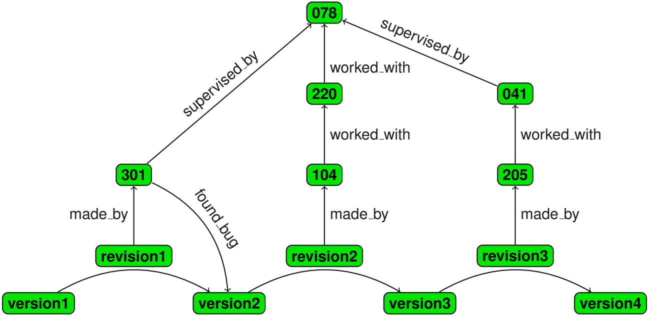
- Triple joins

$$R \overset{1,3',3}{\bowtie} R'$$

- Triple joins

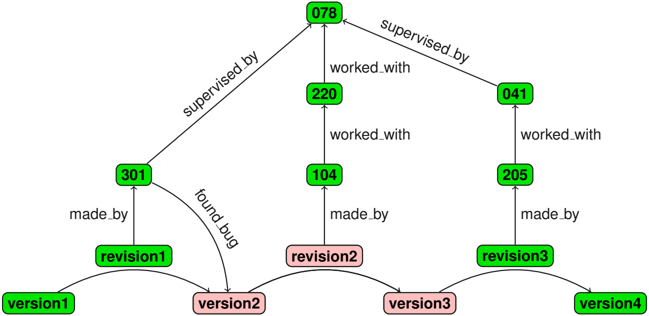
$$R \underset{2=1'}{\overset{1,3',3}{\bowtie}} R'$$

A simple join



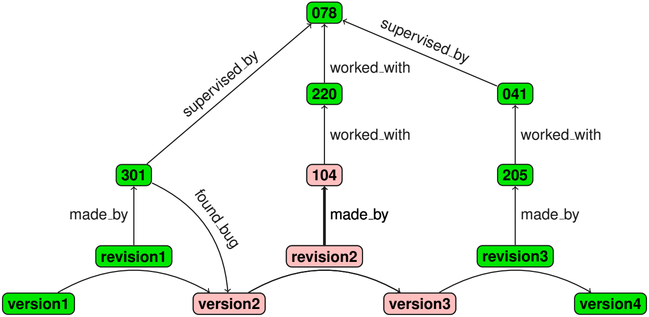
$$E \underset{2=1'}{\overset{1,3',3}{\bowtie}} E$$

A simple join



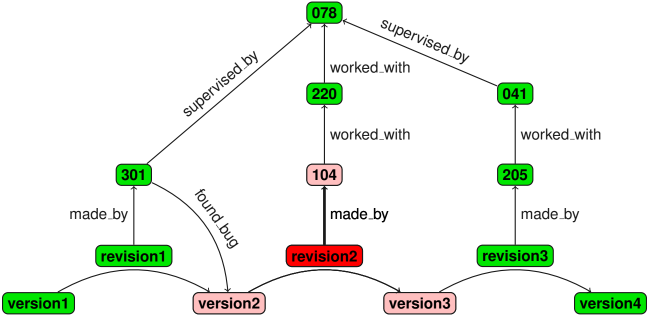
$$E \overset{1,3',3}{\underset{2=1'}{\bowtie}} E$$

A simple join



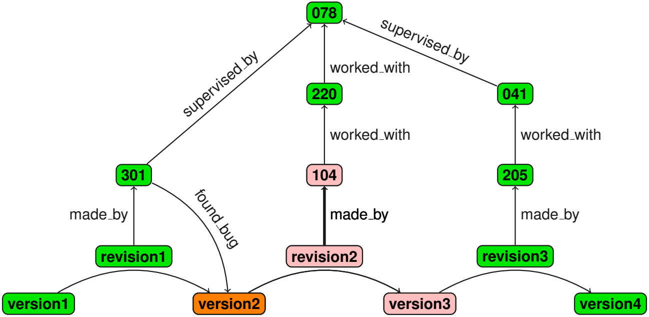
$$E \underset{2=1'}{\overset{1,3',3}{\bowtie}} E$$

A simple join



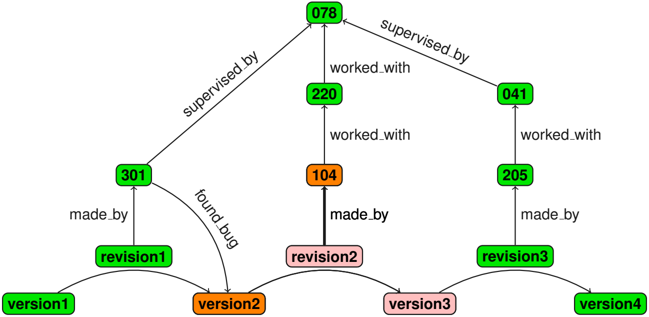
$$E \underset{2=1'}{\overset{1,3',3}{\bowtie}} E$$

A simple join



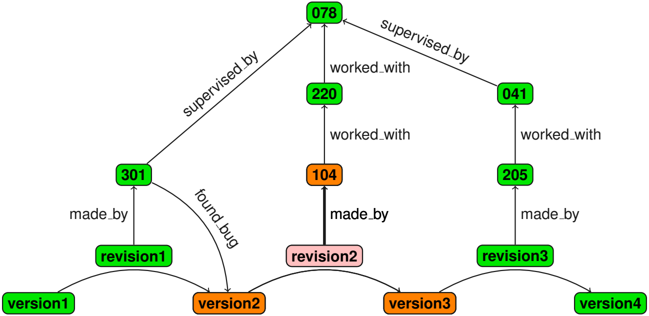
$$E \underset{2=1'}{\overset{1,3',3}{\bowtie}} E$$

A simple join



$$E \underset{2=1'}{\overset{1,3',3}{\bowtie}} E$$

A simple join



$$E \underset{2=1'}{\overset{1,3',3}{\bowtie}} E$$

TriAL: An algebra of triples

- R: set of triples
  Relational representation of a an RDF graph

A TriAL expression is built using

- Set R of triples
- Joins $\bowtie$
- Union $\cup$
- Difference $\setminus$

Adding recursion – TriAL*

- For binary reachability we just iterate the join
- For triples things are not symmetric
  - In particular some joins are not associative
  - So we need both left and right star

Adding recursion – TriAL*

- For binary reachability we just iterate the join
- For triples things are not symmetric
  - In particular some joins are not associative
  - So we need both left and right star

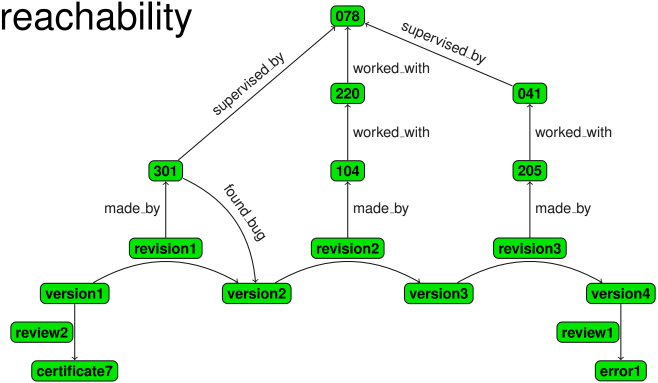
$$(e \bowtie)^* \quad = \quad \emptyset \cup e \cup \ e \bowtie e \ \cup \ (e \bowtie e) \bowtie e \ \cup \ \ldots,$$

Adding recursion – TriAL*

- For binary reachability we just iterate the join
- For triples things are not symmetric
  - In particular some joins are not associative
  - So we need both left and right star

$$
\begin{aligned}
(e \bowtie)^* &= \emptyset \cup e \cup e \bowtie e \cup (e \bowtie e) \bowtie e \cup \ldots, \\
(\bowtie e)^* &= \emptyset \cup e \cup e \bowtie e \cup e \bowtie (e \bowtie e) \cup \ldots
\end{aligned}
$$

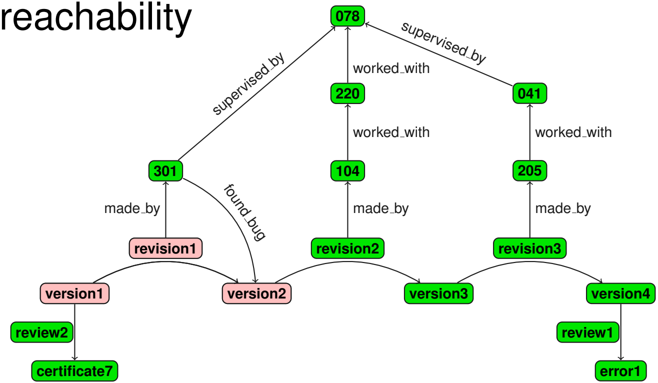
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$(E \bowtie^{1,2',3'}_{3=1'})^*$$

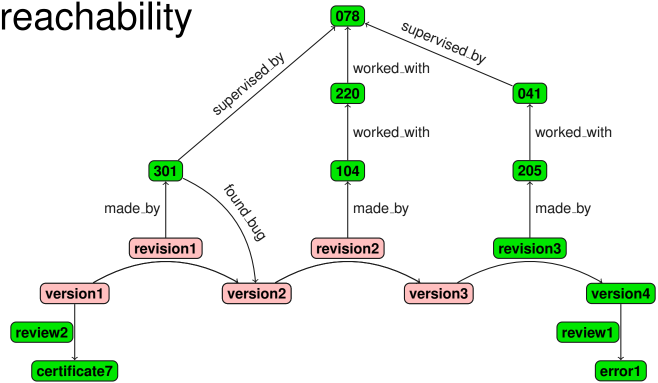
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$(E \bowtie^{1,2',3'}_{3=1'})^*$$

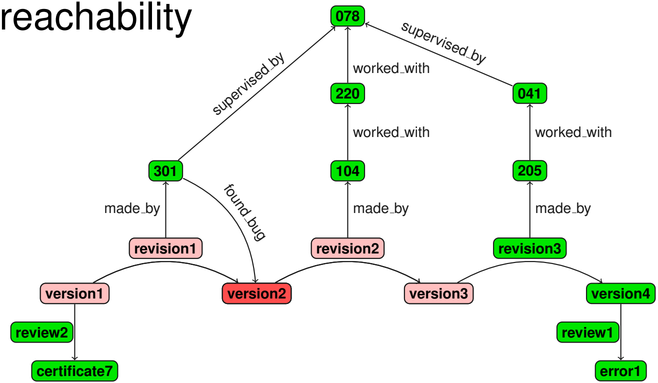
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$\left(E \bowtie^{1,2',3'}_{3=1'}\right)^*$$

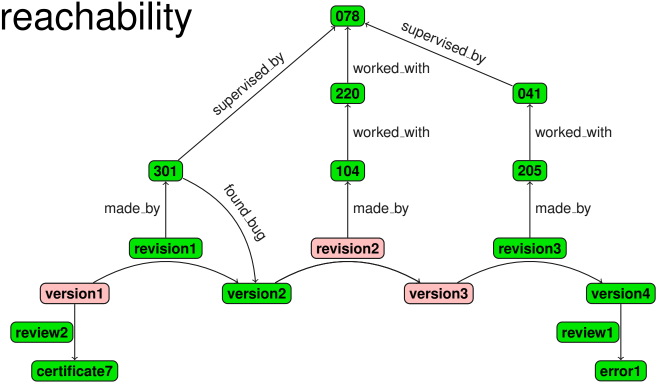
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$(E \bowtie^{1,2',3'}_{3=1'})^*$$

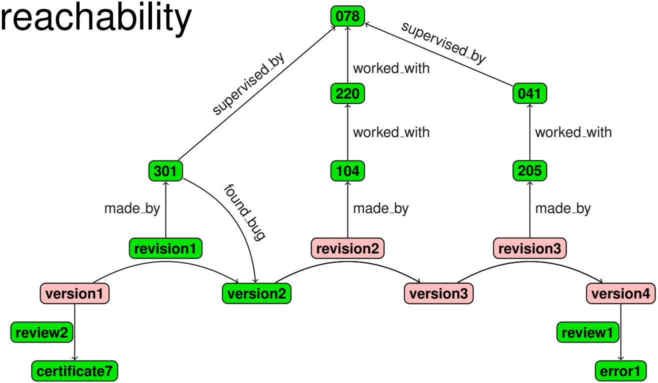
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$(E \bowtie^{1,2',3'}_{3=1'})^*$$

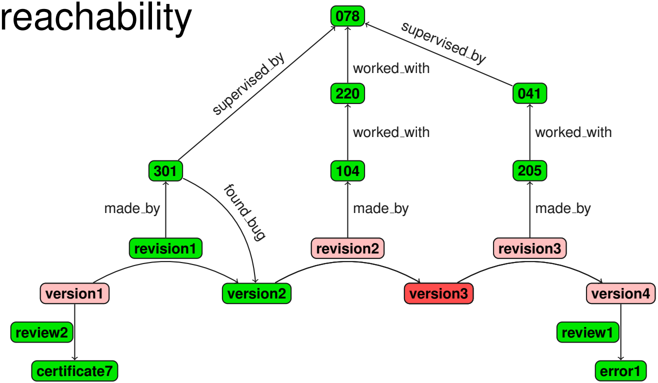
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$(E \bowtie^{1,2',3'}_{3=1'})^*$$

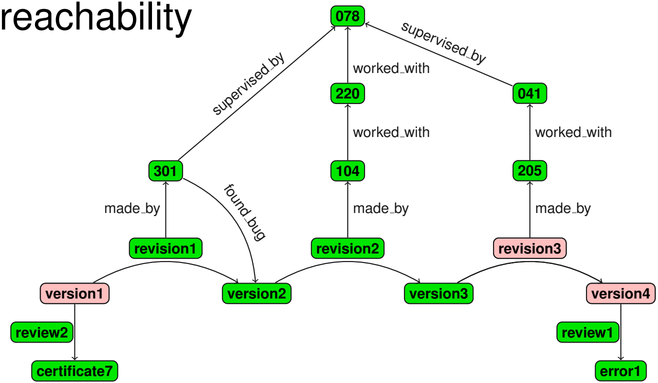
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$\left(E \bowtie^{1,2',3'}_{3=1'}\right)^*$$

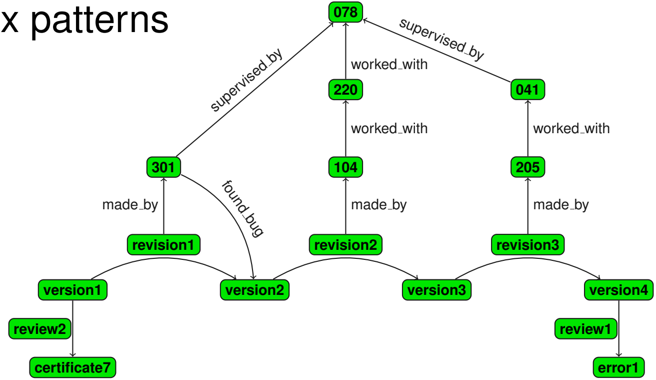
Simple reachability



- Q_1 : Find all versions with an error
  That originate from a valid version
  And return the latest revision

$$(E \bowtie^{1,2',3'}_{3=1'})^*$$

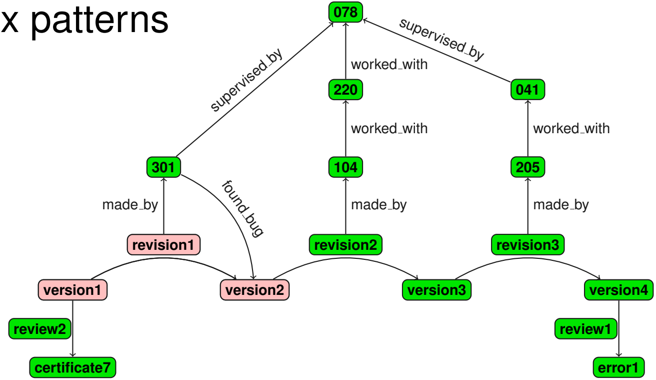
Complex patterns



- Q_2 : Find all versions with an error
That originate from a valid version
And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

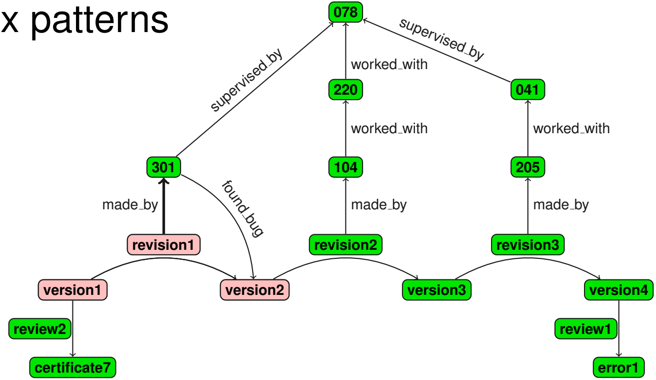
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^{*} \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^{*}$$

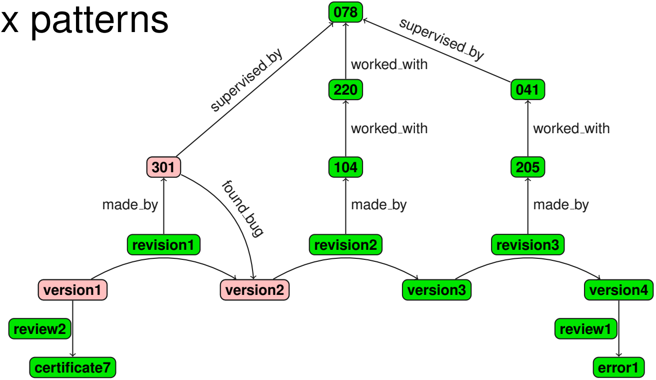
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

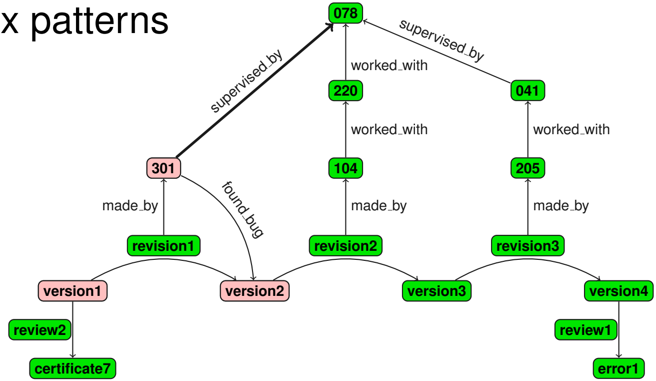
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^{*} \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^{*}$$

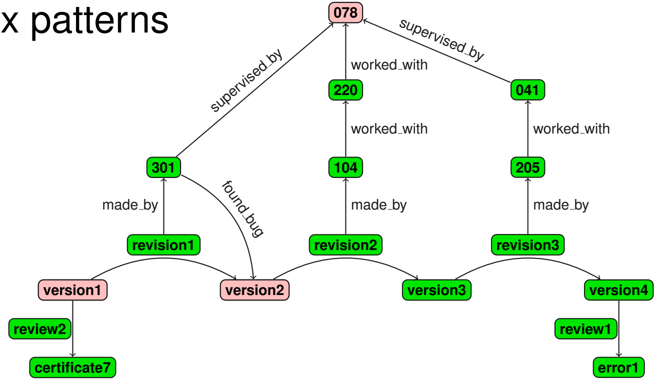
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

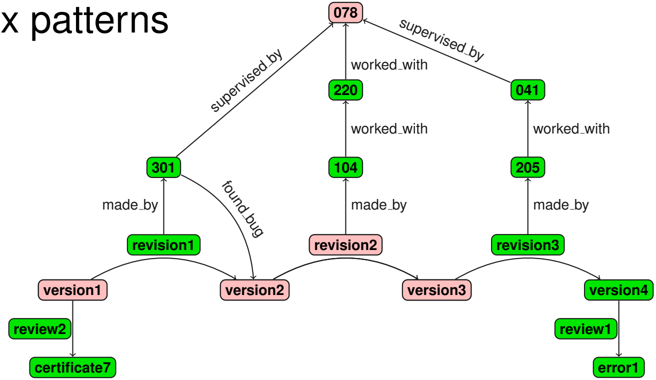
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

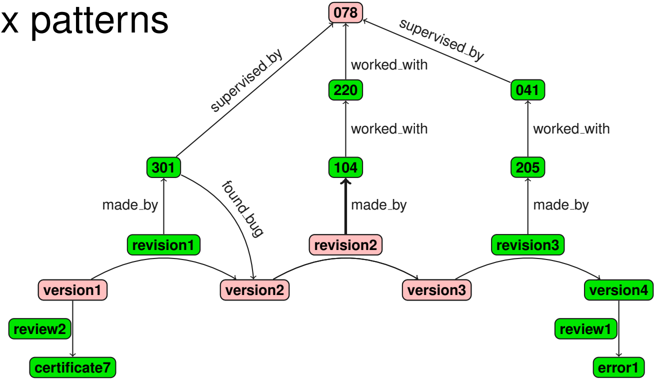
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

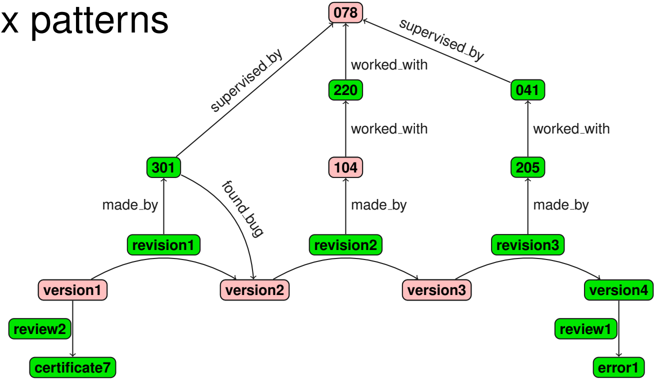
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \mathop{\Join}\limits^{1,3',3}_{2=1'})^* \mathop{\Join}\limits^{1,2,3'}_{3=1',2=2'})^*$$

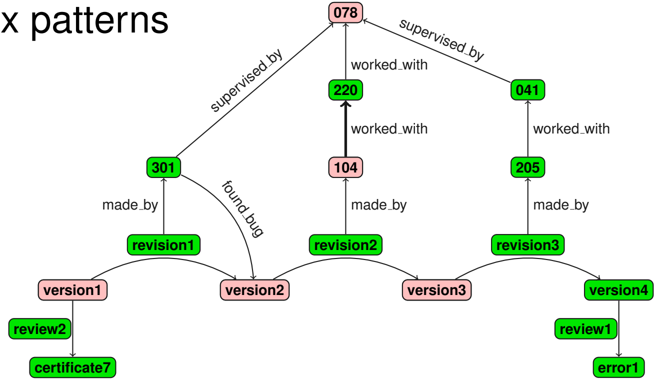
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

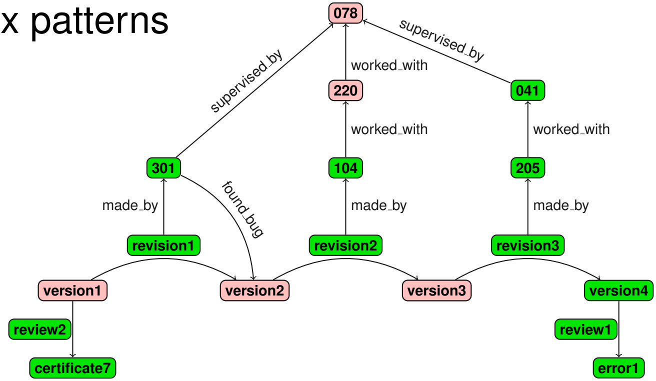
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

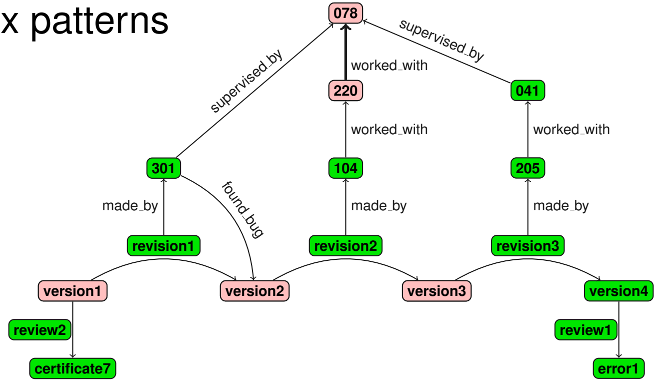
Complex patterns



- Q_2 :
  Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

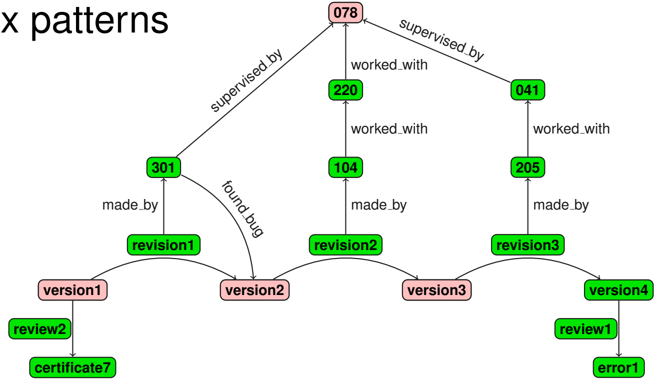
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^{*} \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^{*}$$

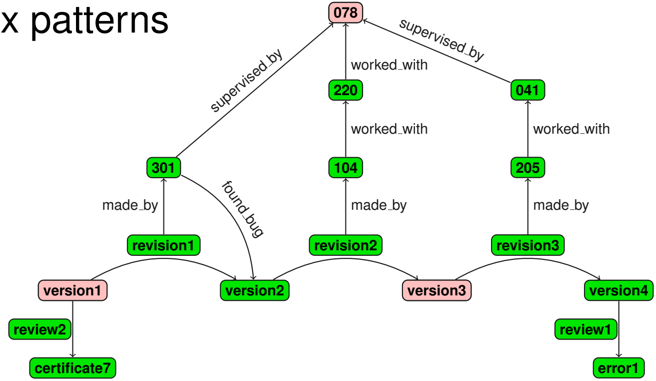
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

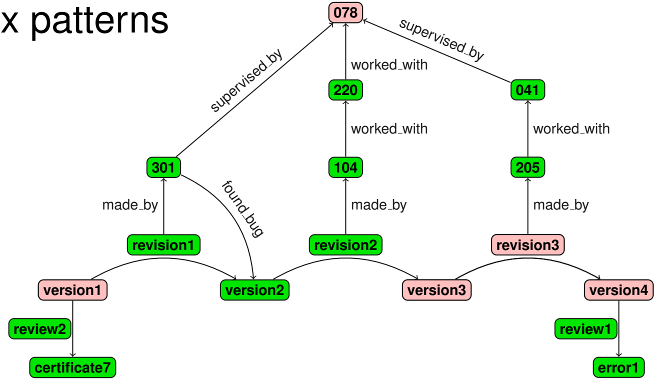
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^{*} \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^{*}$$

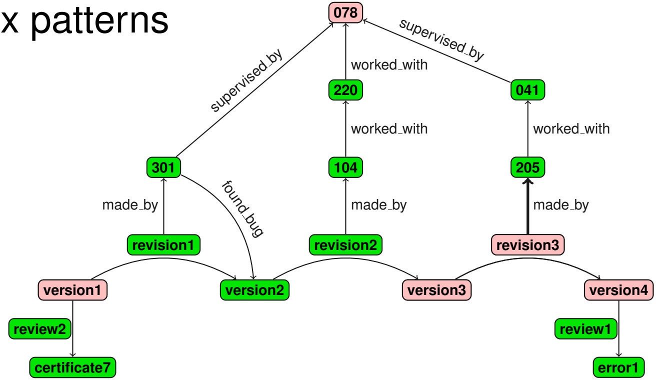
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

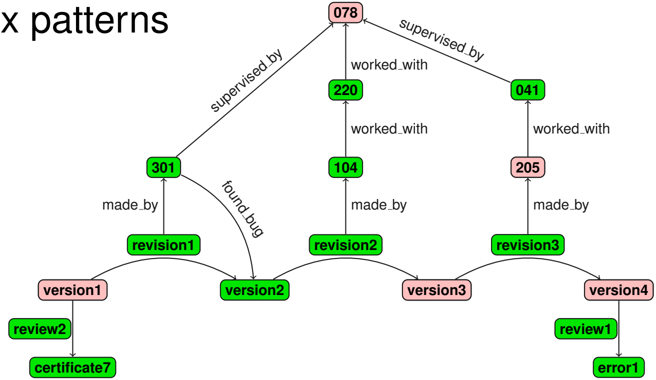
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^* \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^*$$

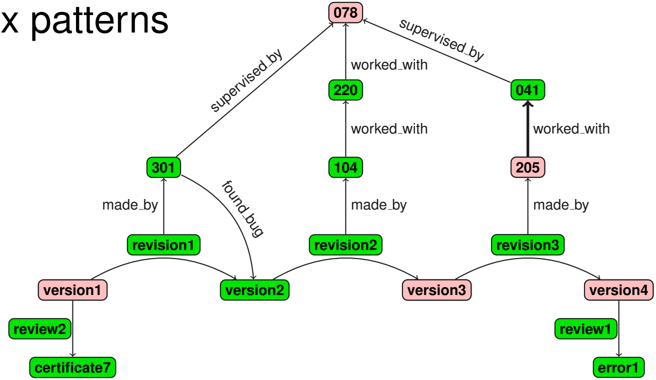
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

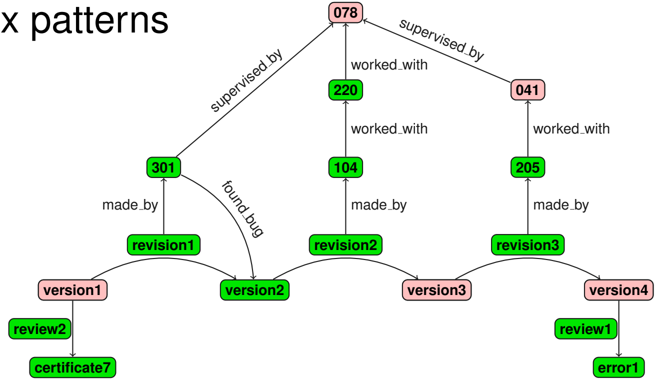
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^{*} \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^{*}$$

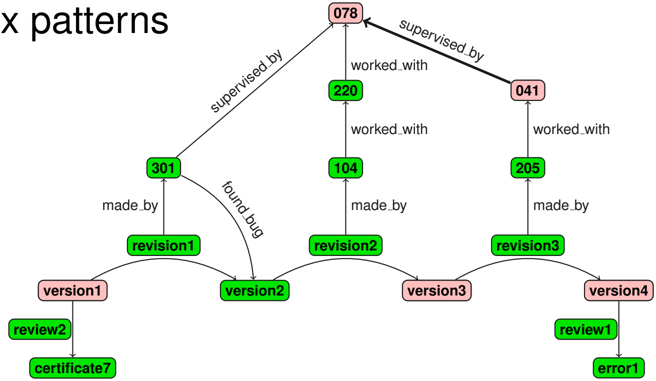
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \mathop{\bowtie}\limits^{1,3',3}_{2=1'})^{*} \mathop{\bowtie}\limits^{1,2,3'}_{3=1',2=2'})^{*}$$

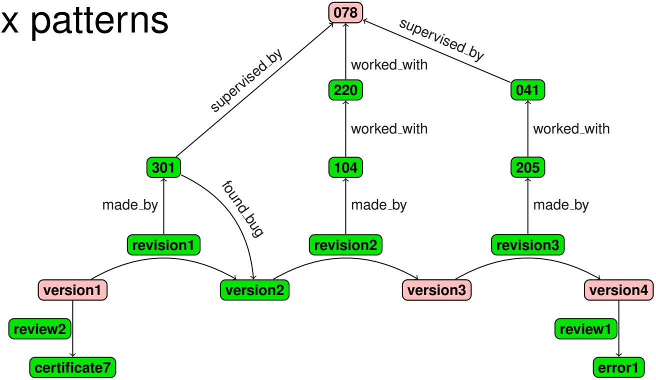
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

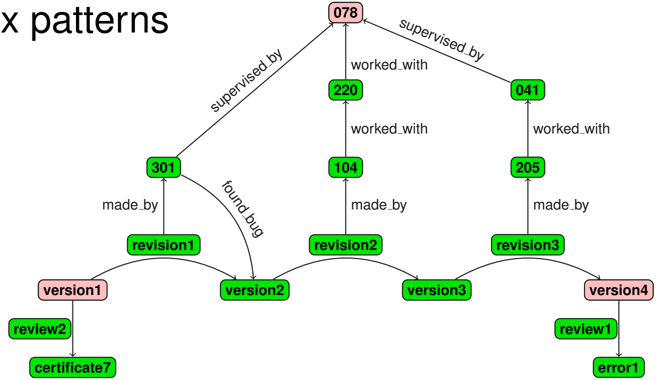
Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \overset{1,3',3}{\underset{2=1'}{\bowtie}})^{*} \overset{1,2,3'}{\underset{3=1',2=2'}{\bowtie}})^{*}$$

Complex patterns



- Q_2 : Find all versions with an error
  That originate from a valid version
  And the person responsible

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*$$

TriAL*:

- Similar complexity bounds to RPQs
- Evaluation algorithm uses dynamic programming

TriAL*: fragment of relational algebra

- well known formalism
- can be translated into SQL- like statements (or datalog-like)
- fits right onto relational query implementations but we might need new heuristics

Other theoretical advantages of TriAL*

- know expressive power: First Order logic with transitive closure and fixed amount of variables
- Can I pose this query?

Outline

1. Graph databases
   Most studied connectivity queries

2. Relationship between connectivity and recursion
   An algebra for querying graphs

3. Where to go from here

What now?

Lets get back to what we know, and study connectivity queries from a relational perspective

- Shortest path
- Count number of paths
- Aggregate on paths (total distance, etc)

What now?

Include TriAL in graph implementations

- Include it in SPARQL
- compare performance with Neo4j, Dex
- RDF DBMS (Jenna, etc)