RDF dataset anonymization robust to data interlinking

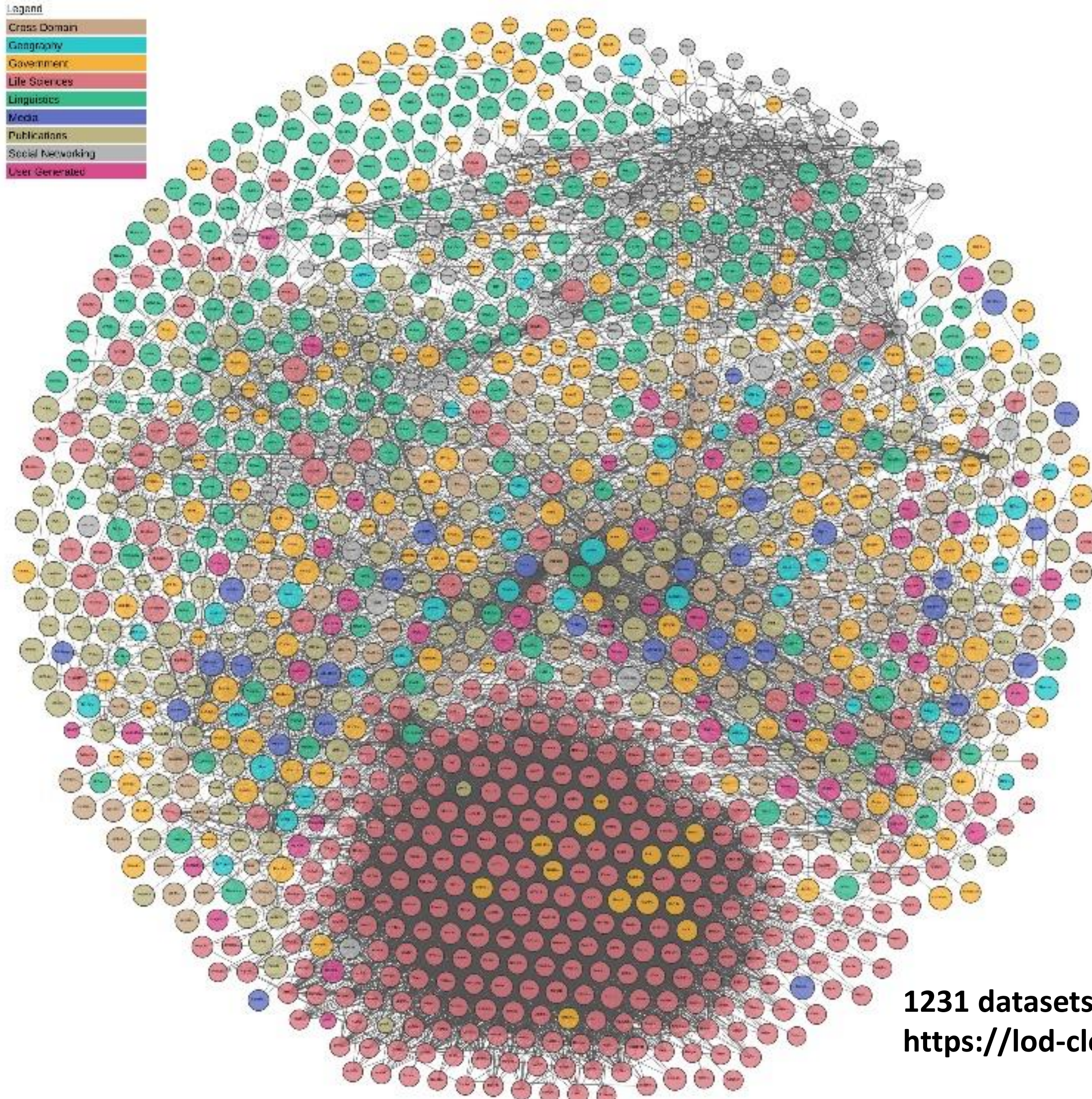
Marie-Christine Rousset

Université Grenoble Alpes & Institut Universitaire de France

Joint work with

Rémy Delanaux, Angela Bonifati and Romuald Thion (Univ. Lyon1-LIRIS)

Linked Open Data



1231 datasets in November 2018
https://lod-cloud.net/

The standards underlying Linked Open Data

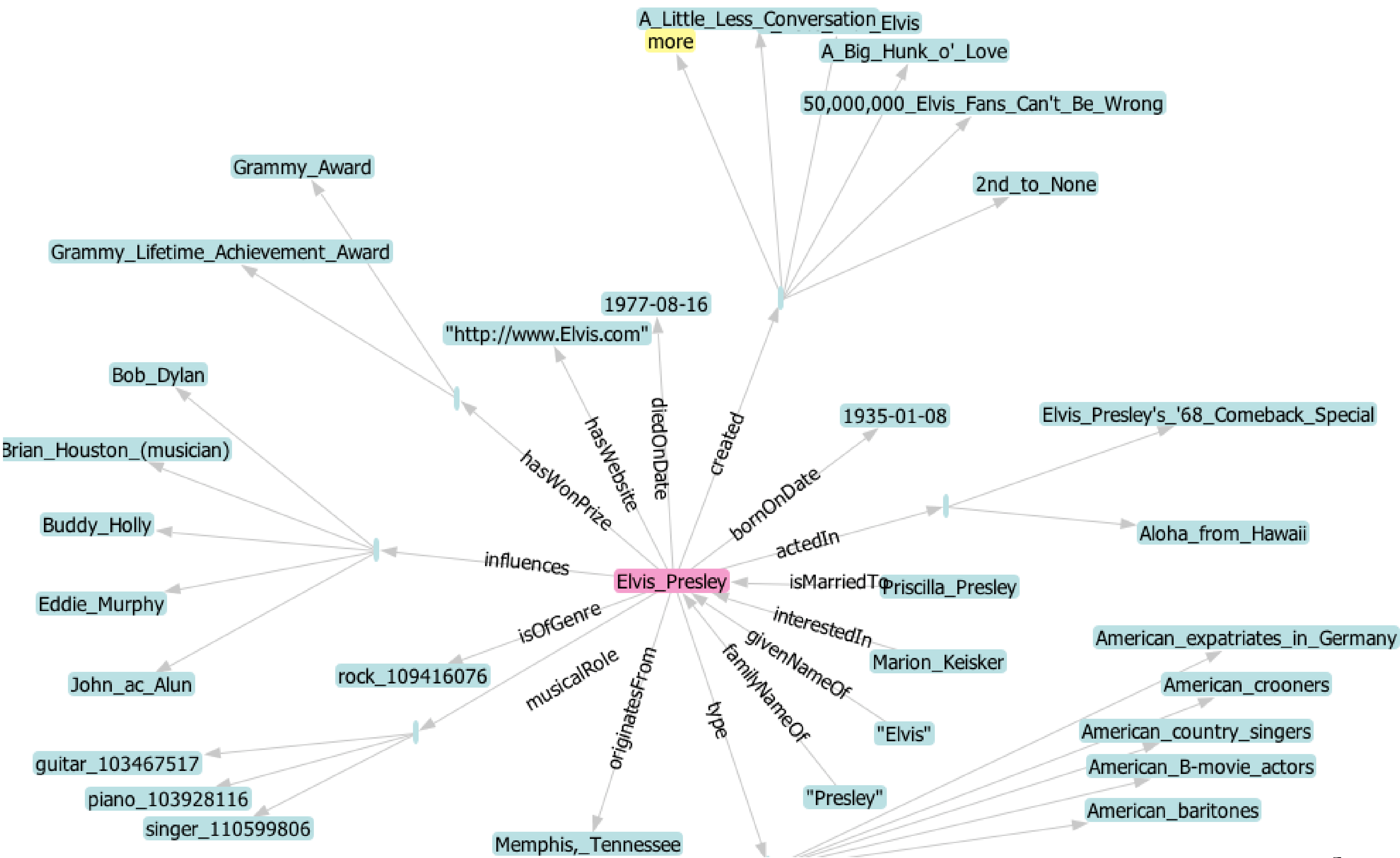
- http, URIs and namespaces
  - For identifying and naming entities without ambiguity
    - URIs: Uniform Resource Identifiers
    - Namespace:
      - A name in a namespace consists of a namespace identifier and a local name.
      - No homonym within a given namespace
- RDF (Resource Description Framework)
  - For declaring facts on entities as triples

    <subject, relation/property, object/value>
- RDFS (RDF Schema) and OWL
  - For grouping entities into classes structured in class hierarchies
  - For providing semantics to the relations and properties
- SPARQL
  - For asking queries to endpoints accessible through web services
    - http://rdf.insee.fr/sparql

RDF : a graph model

- The RDF data model allows writing labeled graph using triples
  - A triple has three components
    - A **subject**: URI or a blank node (unnamed URI)
    - A **property** or predicate: URI
    - An **object**: URI, blank node or literal (string)
  - A triple is written: **subject property object**.
- An RDF graph is a set of triples
  - Its nodes are (labeled with) the subjects and objects of the triples: one node per URI
  - Its edges are (labeled with) the properties of the triples

Example (from Yago)



Endpoint SPARQL DBpedia

http://fr.dbpedia.org/sparql

- Municipalities of Ile de France region with more than 100.000 inhabitants with their mayor?

SELECT ?commune ?maire

WHERE {

?commune <http://dbpedia.org/ontology/region> <http://fr.dbpedia.org/resource/Île-de-France> .

?commune rdf:type dbpedia-owl:PopulatedPlace .

?commune dbpedia-owl:populationTotal ?population .

?commune prop-fr:maire ?maire

FILTER (?population > 100000) }

commune	maire
http://fr.dbpedia.org/resource/Paris	"Anne Hidalgo"@fr
http://fr.dbpedia.org/resource/Boulogne-Billancourt	"Pierre-Christophe Baguet"@fr
http://fr.dbpedia.org/resource/Montreuil_(Seine-Saint-Denis)	"Patrice Bessac"@fr
http://fr.dbpedia.org/resource/Saint-Denis_(Seine-Saint-Denis)	"Didier Paillard"@fr
http://fr.dbpedia.org/resource/Val-de-Marne	http://fr.dbpedia.org/resource/Laurent_Cathala
http://fr.dbpedia.org/resource/Argenteuil_(Val-d'Oise)	"Georges Mothron"@fr

SPARQL queries based on Basic Graph Patterns (BGPs)

- Conjunctive queries:

  SELECT ?v1 ?v2 …?vk

  WHERE {TP1. TP2. ….TPn}

  – Each TPi is a triple with variables and without blank nodes(triple pattern)
  – A variable can appear in any position of a triple pattern
  – A join variable is a variable occuring in several triple patterns

    => TP1, TP2, …, TPn is thus a graph pattern

- The evaluation of a conjunctive query over an RDF dataset DS is based on the existence of **mappings μ** from the variables in the query to URIs, blank nodes or literals appearing in DS such that **for every i, μ(TPi) $\in$ DS**
  – μ(TP) is the triple obtained by replacing every occurrence of each variable ?x by μ(?x)
  – μ is an application:
  – **The answer set** is the set of mappings μ such that for every i, μ(TPi) $\in$ DS, projected on the distinguished variables, **represented as a table**
    - One column per distinguished variable
    - One row per mapping, with the corresponding values of the distinguished variables

Example

Q1: SELECT * WHERE {?x vCard:N ?y. ?y vCard:Family ''Smith''. ?y vCard:Given ?givenName}

Q2: SELECT ?x ?givenName WHERE {?x vCard:N ?y. ?y vCard:Family ''Smith''. ?y vCard:Given ?givenName}

Dataset DS

```
@prefix vCard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

<http://somewhere/MattJones/>  vCard:FN   "Matt Jones" .
<http://somewhere/MattJones/>  vCard:N    _:b0 .
_:b0  vCard:Family "Jones" .
_:b0  vCard:Given  "Matthew" .

<http://somewhere/RebeccaSmith/> vCard:FN    "Becky Smith" .
<http://somewhere/RebeccaSmith/> vCard:N     _:b1 .
_:b1 vCard:Family "Smith" .
_:b1 vCard:Given  "Rebecca" .

<http://somewhere/JohnSmith/>    vCard:FN    "John Smith" .
<http://somewhere/JohnSmith/>    vCard:N     _:b2 .
_:b2 vCard:Family "Smith" .
_:b2 vCard:Given  "John"  .

<http://somewhere/SarahJones/>   vCard:FN    "Sarah Jones" .
<http://somewhere/SarahJones/>   vCard:N     _:b3 .
_:b3 vCard:Family  "Jones" .
_:b3 vCard:Given   "Sarah" .
```

Answers returned by Q1 against DS

x	y	givenName
<http://somewhere/RebeccaSmith>	_:b1	''Rebecca''
<http://somewhere/JohnSmith>	_:b2	''John''

Answers returned by Q2 against DS

x		givenName
<http://somewhere/RebeccaSmith>		''Rebecca''
<http://somewhere/JohnSmith>		''John''

Counting queries

- COUNT (Q) where Q is a conjunctive query

Answer(Count(Q), DS) = | Answer(Q, DS) |

Q3: SELECT ?x WHERE {?x vCard:N ?y. ?y vCard:Family ''Smith''. }

Answer (Count(Q3), DS) = 2

- SPARQL syntax:

SELECT (COUNT ?x) WHERE {?x vCard:N ?y. ?y vCard:Family ''Smith''. }

SPARQL Update queries

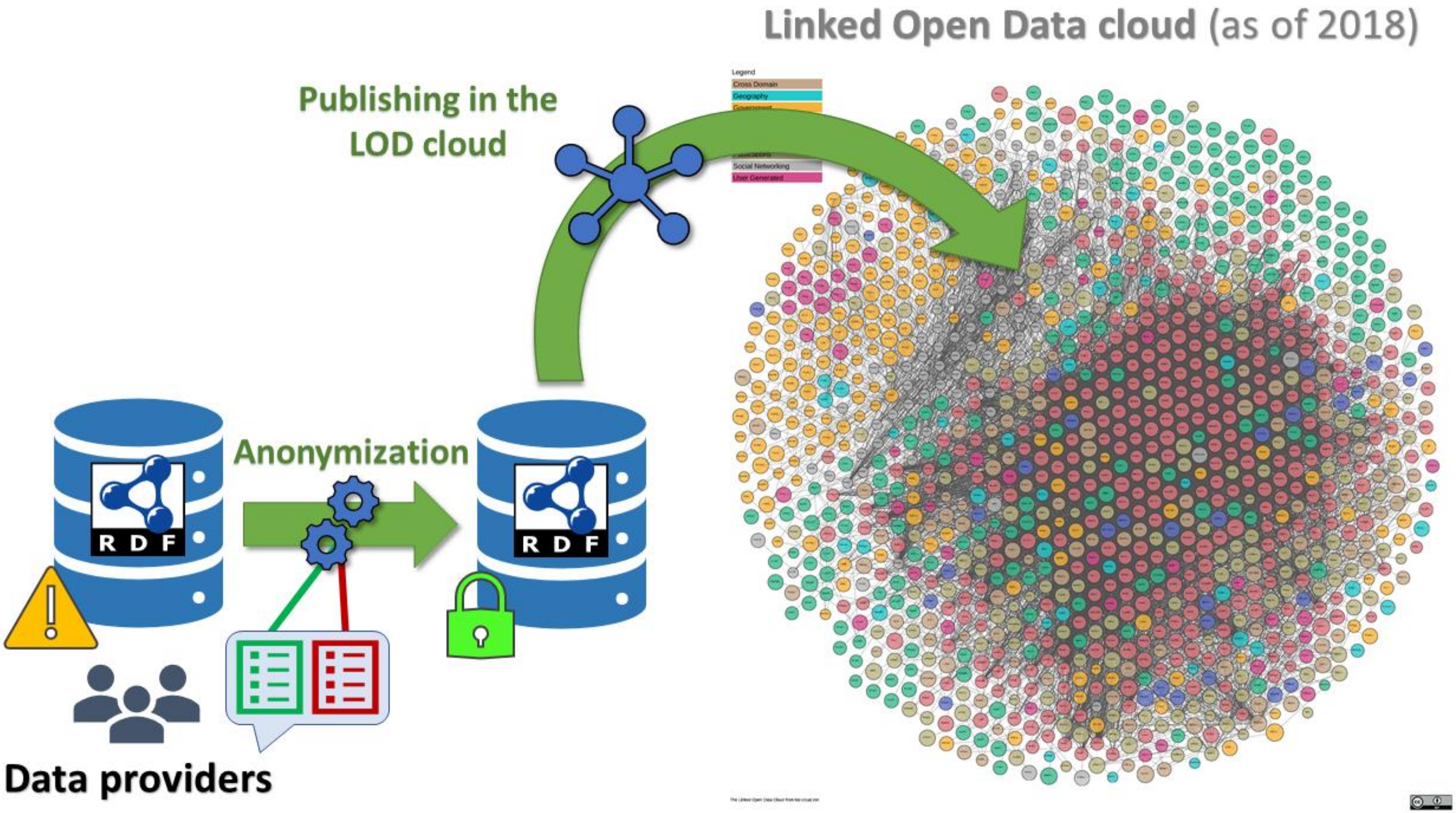
```
# UPDATE outline syntax : general form:
MODIFY [ <uri> ]*
DELETE { template }
INSERT { template }
[ WHERE { pattern } ]
```

- *template*
  - An extension of a BGP **with possible blank nodes**
- *pattern* is a BGP
- *pattern* is evaluated as in the SPARQL query
  - SELECT * WHERE {*pattern*}
  - all the values of the variables are used in the INSERT and DELETE *templates* for defining the triples to be inserted or deleted
- The deletion of triples happens before the insertion
- DELETE (respectively INSERT) queries: particular case with an empty INSERT *template* (respectively an empty DELETE *template*)

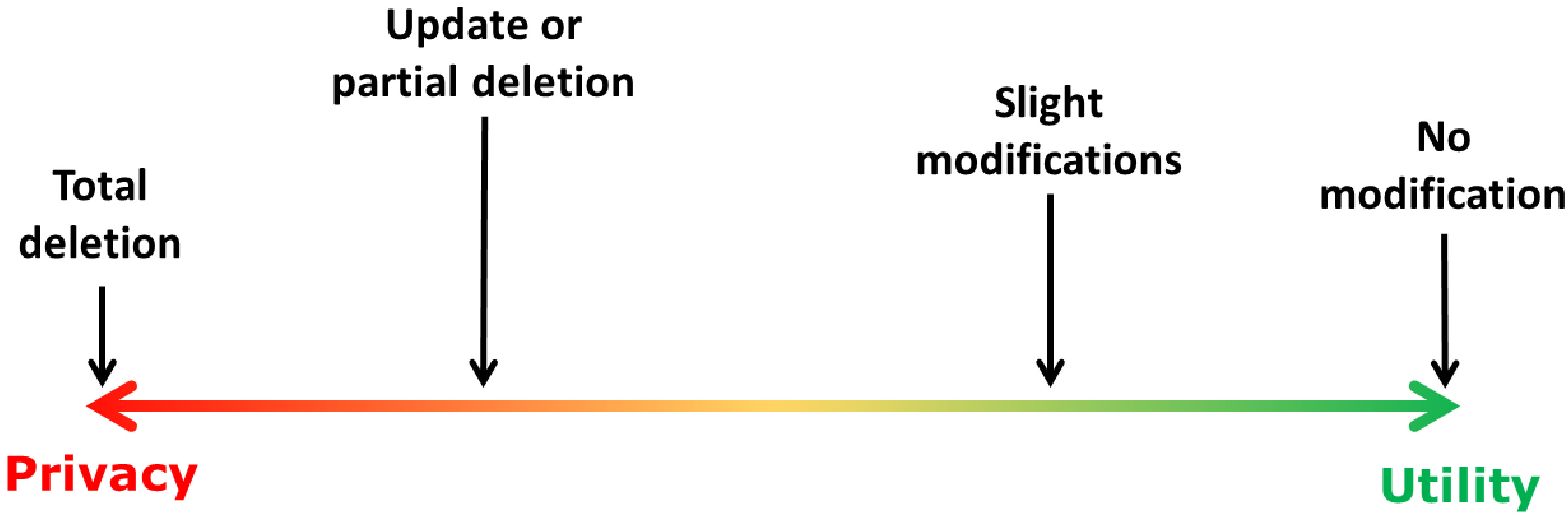
Example

```
DELETE  {   ?c  tcl:user    ?u. }
INSERT  {   ?c  tcl:user    []. }
WHERE   {   ?c  a   tcl:Journey.
            ?c  tcl:user        ?u.
            ?c  geo:latitude    ?lat.
            ?c  geo:longitude   ?long. }
```

Linked Data anonymization



which tradeoff between privacy and utility?



Different existing approaches

- Add noise in the data
  - Differential privacy [1,2]
    - strong mathematical guarantees of non-disclosure of any individual information
    - maximise the accuracy of statistical queries
    - low utility of answers returned by precise queries
- Suppress or generalize information
  - K-anonymity [3,4,5]
    - atleast k records with indistinguishable values over quasi-identifiers of sensitive information
    - measure the resulting loss of information
- Apply access control policies [6,7]
  - Data is unchanged but permissions are required to query it
    - distinguish users with different privileges
    - define authentication rules to control whether a given user is allowed to issue a given query
  - Not particularly adapted to Linked Open Data setting

Our approach

- Declarative specification of privacy and utility policies as a set of SPARQL conjunctive and/or counting queries
- Sound and data-independent algorithms for computing anonymizations operations as SPARQL update queries
  - with the guarantee that the resulting datasets satisfy both the privacy and utility policies
  - even when linking the anonymized dataset with any external RDF dataset

Specification of privacy and utility policies

- Privacy / Utility policies: a set of queries
- An anonymyzed dataset Anonym(DS) satisfies:
  - a privacy policy if **for each** privacy query **p** , the evaluation of **p** on **Anonym(DS)** does not return any tuple of constants: **no answer** or **tuples of blank nodes**
  - an utility policy if **for each** utility query **u** , the evaluation of **u** returns the **same results on Anonym(DS) and on DS**

Privacy policy

```
SELECT ?ad
WHERE {
  ?u  a              tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

```
SELECT ?u ?lat ?long
WHERE {
  ?c  a              tcl:Journey.
  ?c  tcl:user       ?u.
  ?c  geo:latitude   ?lat.
  ?c  geo:longitude  ?long.
}
```

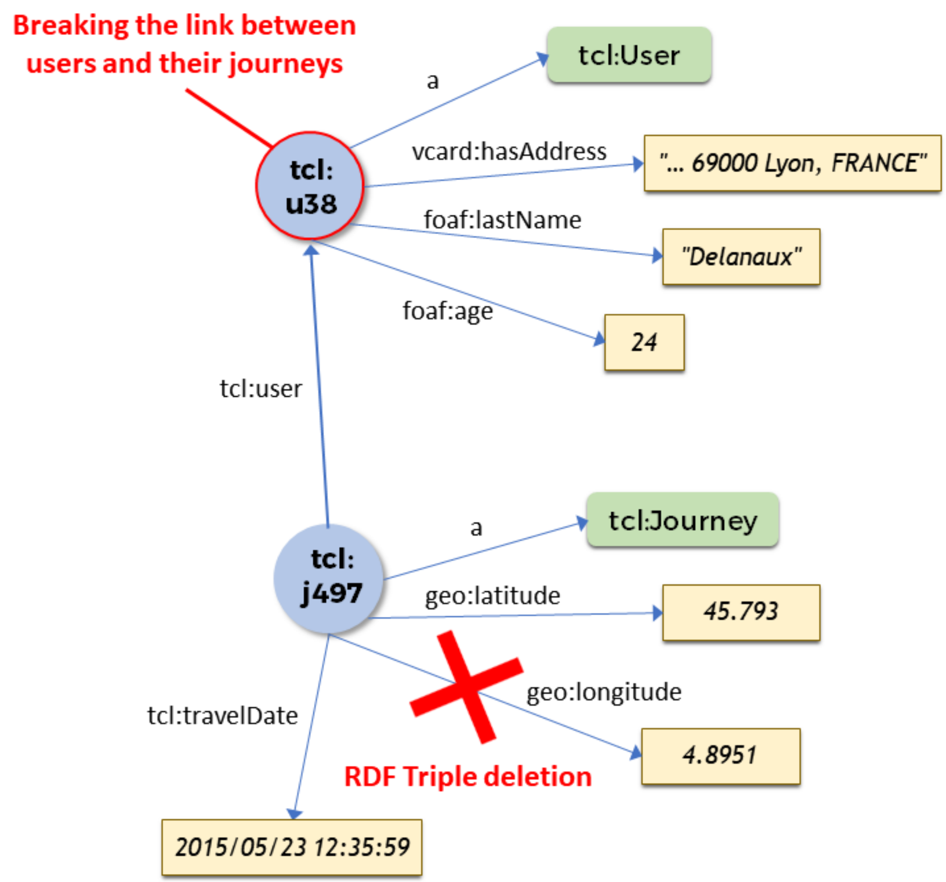
Utility policy

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
```

```
SELECT ?c ?lat ?long
WHERE {
  ?c  a              tcl:Journey.
  ?c  geo:latitude   ?lat.
  ?c  geo:longitude  ?long.
}
```

Query-based anonymization operators

- SPARQL update queries
  - delete triples
  - replace URIs by blank nodes



Delete queries for triple deletion

Deletion query

- Particular case of update queries, with no replacement value
- DELETE $D(\bar{x})$ WHERE $W(\bar{x}, \bar{y})$

- Example:
  – delete all the triples corresponding to the property hasAddress of users

```
DELETE  {    ?u  vcard:hasAddress   ?ad.}
WHERE   {    ?u  a    tcl:User.
             ?u  vcard:hasAddress   ?ad.}
```

Update queries for replacement of Uris by blank nodes

Update query

- DELETE $D(\bar{x})$ INSERT $I(\bar{y})$ WHERE $W(\bar{x}, \bar{z})$

- Example:
  - Replace with blank nodes the URIs of users for which the location of their journeys is known

```
DELETE  {   ?c  tcl:user      ?u. }
INSERT  {   ?c  tcl:user      []. }
WHERE   {   ?c  a   tcl:Journey.
            ?c  tcl:user             ?u.
            ?c  geo:latitude       ?lat.
            ?c  geo:longitude      ?long. }
```

Problem 1

Compatibility checking between privacy and utility policies

Problem 1: a decision problem

- Compatibility checking between two given privacy and utility policies:
  - for any dataset DS that violates **a privacy query p,** there exists a sequence O of operations such that O(DS) satisfies both policies
- **Incompatibility** if there exists an utility query **u** contained into the privacy query **p**
  - u is contained in p $\Leftrightarrow$ for any DB, Answer(u,DB) $\subseteq$ Answer(p,DB)
  - Let DS the dataset obtained from body(u) by replacing each variable by distinct URIs, and let $\underline{a}$ the tuple of the URIs corresponding to the result variables:
    - By construction, $\underline{a} \in$ Answer(u,DS), and since u is contained in p: $\underline{a} \in$ Answer(p, DS)
  - Therefore DS violates the privacy query p
  - Suppose there exists O such that O(DS) satisfies both policies:
    - $\underline{a} \in$ Answer(u,O(DS)) (since Answer(u,O(DS)) = Answer(u,DS))
    - and, by query containment, $\underline{a} \in$ Answer(p,O(DS))

$\Rightarrow$ Contradiction (the privacy policy is not satisfied)

Query containment problem

- Extensively studied in database theory
  - Many results of complexity and algorithms
  - NP-complete for conjunctive queries [8]

    Algorithm: q1 contained in q2 ?
    - body(q1) seen as a database by freezing its variables
    - Evaluate q2 over this database
      - If the answer set is not empty, return YES, Otherwise return NO
  - Illustration:

    q1(X): R(X,Y), R(Y,Z), R(Z,Z)

    q2(X): R(X,Y), R(Y,Z1), R(Y,Z2)

    – Freezing the variables in q1: X $\rightarrow$a1, Y $\rightarrow$a2, Z $\rightarrow$a3

    – Answer(q2, DB(q1)) = {a1,a2}

    $\Rightarrow$ **q1 is contained in q2**

Back to SPARQL

```
p: SELECT ?ad WHERE
        {?u a tcl:User. ?u vcard:hasAddress ?ad}

u: SELECT ?ad WHERE
        {?u a tcl:User. ?u vcard:hasAddress ?ad.
        ?ad :professionalAddress true. }
```

- u is contained in p
- Incompatible privacy and utility policies

Back to SPARQL (continued)

p: SELECT ?ad WHERE

{?u a tcl:User. ?u vcard:hasAddress ?ad.

?ad :professionalAddress false.}

u: SELECT ?ad WHERE

{?u a tcl:User. ?u vcard:hasAddress ?ad.

?ad :professionalAddress true. }

- u <u>is not contained</u> in p
- Compatible privacy and utility policies

Back to SPARQL (continued)

p: SELECT ?ad WHERE

{?u a tcl:User. ?u vcard:hasAddress ?ad}

u: SELECT ?ad WHERE

{?u a tcl:User.

?u vcard:hasProfessionnalAddress ?ad}

- u is <u>not contained</u> in p
- compatible privacy and utility policies ?
  – What if we have the following knowledge K

vcard:hasProfessionnalAddress rdfs:subPropertyOf vcard:hasAddress

Back to query containment

- Extend the definition:

  q1 is contained in q2 modulo K

  if for every a: q1(a), K |= q2(a)

- Adapt the algorithms accordingly:
  - Rewrite the q2 query using K or complete the q1 query using K

Rewriting(p,K): {p, p' }

p': SELECT ?ad WHERE

{?u a tcl:User. ?u vcard:hasProfessionalAddress ?ad}

⇒ add p' as a privacy query in the privacy policy

Completed(u,K): SELECT ?ad WHERE

{?u a tcl:User. ?u vcard:hasProfessionnalAddress ?ad.

?u vcard:hasAddress ?ad}

⇒ check query containment of Completed(u,K) with the privacy policies

Refinement of incompatible policies

- Either by constraining the the privacy queries

p: SELECT ?ad WHERE
{?u a tcl:User. ?u vcard:hasPersonalAddress ?ad}

- Or by generalizing the utility queries

u: SELECT ?ad WHERE
{ ?ad :professionalAddress true. }

$\Rightarrow$ to be done and/or validated by the data provider

Problem 2

Build anonymization operations
to satisfy compatible privacy and utility policies
when applied to **a given dataset** or **to any dataset**

Problem 2: a construction problem

- Given two **compatible** privacy and utility policies, **build candidate sequences of anonymization operations** such that their application **to any dataset DS** satisfy both privacy and utility policies
- Our contribution
  - a two-step algorithm that builds a **set of update queries**
    - **Step1:** for each privacy query $\mathbf{p_i}$ consider in isolation each triple pattern and if it can be mapped with a triple pattern in an utility query, build the set O($\mathbf{p_i}$) of all the possible update queries (Delete or IR replacement)
    - **Step2:** compute the cartesian product Ω: O($\mathbf{p_1}$) x…x O($\mathbf{p_i}$)x…x O($\mathbf{p_n}$)
  - Soundness property:
    - For every i, if O($\mathbf{p_i}$) is non empty, every o $\in$ O($\mathbf{p_i}$) satisfies the single privacy policy made of $\mathbf{p_i}$ and the global utilily policy made of all the utility queries
    - If Ω is not empty, for any set S $\in$ Ω, for any dataset DS, for any ordering O of the operations in S, **O(DS) satisfies both privacy and utility policies**

Illustration by example: Step1.1

Privacy query #1:

```
SELECT ?ad
WHERE {
  ?u  a  tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step 1.1

Privacy query #1:

```
SELECT ?ad
WHERE {
  ?u  a  tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step 1.1

Privacy query #1:

```
SELECT ?ad
WHERE {
  ?u  a  tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step 1.1

Privacy query #1:

```
SELECT ?ad
WHERE {
  ?u  a  tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step 1.1

Privacy query #1:

```
SELECT ?ad
WHERE {
  ?u  a  tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: result of Step1.1

Output for P1:

$$O_1 = \{\texttt{DELETE}\ \{?u\ \texttt{vcard:hasAddress}\ ?ad.\} \qquad (op_1)$$

$$\texttt{WHERE}\ \{?u\ \texttt{a}\ \texttt{tcl:User}.\ ?u\ \texttt{vcard:hasAddress}\ ?ad.\},$$

$$\texttt{DELETE}\ \{?u\ \texttt{vcard:hasAddress}\ ?ad.\} \qquad (op_2)$$

$$\texttt{INSERT}\ \{[]\ \texttt{vcard:hasAddress}\ ?ad.\}$$

$$\texttt{WHERE}\ \{?u\ \texttt{a}\ \texttt{tcl:User}.\ ?u\ \texttt{vcard:hasAddress}\ ?ad.\},$$

$$\texttt{DELETE}\ \{?u\ \texttt{vcard:hasAddress}\ ?ad.\} \qquad (op_3)$$

$$\texttt{INSERT}\ \{?u\ \texttt{vcard:hasAddress}\ [].\}$$

$$\texttt{WHERE}\ \{?u\ \texttt{a}\ \texttt{tcl:User}.\ ?u\ \texttt{vcard:hasAddress}\ ?ad.\}\}$$

Illustration by example: Step1.2

Privacy query #2:

```
SELECT ?u ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  tcl:user  ?u.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step1.2

Privacy query #2:

```
SELECT ?u ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  tcl:user  ?u.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step1.2

Privacy query #2:

```
SELECT ?u ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  tcl:user  ?u.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step1.2

Privacy query #2:

```
SELECT ?u ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  tcl:user  ?u.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: Step1.2

Privacy query #2:

```
SELECT ?u ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  tcl:user  ?u.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Utility queries:

```
SELECT ?u ?age
WHERE {
  ?u  a  tcl:User.
  ?u  foaf:age  ?age.
}
SELECT ?c ?lat ?long
WHERE {
  ?c  a  tcl:Journey.
  ?c  geo:latitude ?lat.
  ?c  geo:longitude  ?long.
}
```

Illustration by example: result of Step1.2

Output for P2:

$$O_2 = \{\texttt{DELETE}\,\{?c\ \texttt{tcl:User}\ ?u.\}\ \ (op_4)$$
$$\texttt{WHERE}\,\{?c\ \texttt{a}\ \texttt{tcl:Journey.}\ \ldots\},$$

$$\texttt{DELETE}\,\{?c\ \texttt{tcl:User}\ ?u.\}\ \ (op_5)$$
$$\texttt{INSERT}\,\{[\,]\,\texttt{tcl:User}\ ?u.\}$$
$$\texttt{WHERE}\,\{?c\ \texttt{a}\ \texttt{tcl:Journey.}\ \ldots\},$$

$$\texttt{DELETE}\,\{?c\ \texttt{tcl:User}\ ?u.\}\ \ (op_6)$$
$$\texttt{INSERT}\,\{?c\ \texttt{tcl:User}\ [\,].\}$$
$$\texttt{WHERE}\,\{?c\ \texttt{a}\ \texttt{tcl:Journey.}\ \ldots\}\}$$

Illustration by example: result of Step2

We have $O_1 = \{op_1, op_2, op_3\}$ and $O_2 = \{op_4, op_5, op_6\}$

Which gives 9 possible sets of operations:

$$
\begin{aligned}
Ops = \Big\{ & \{op_1, op_4\}, \{op_1, op_6\}, \{op_1, op_6\}, \\
& \{op_2, op_4\}, \{op_2, op_5\}, \{op_2, op_6\}, \\
& \{op_3, op_4\}, \{op_3, op_5\}, \{op_3, op_6\} \Big\}
\end{aligned}
$$

Properties of the algorithms

- Soundness
  - If the output is not empty, the input privacy and utility policies are compatible, and
  - the application to any input DS of every set of update queries returned by the algorithm leads to a dataset that satisfies the input privacy and utility policies
- Complexity
  - Step 1:
    - polynomial in time (O(size(P) x size(U))
    - output size: O(size(P))
  - Step 2 : exponential in the number n of privacy queries
    - cartesian product of n sets of size in O(size(P)))
  - Constant data complexity:
    - Data-independent algorithms
- Runtime efficiency in practice:
  - 0.84s on average for policies of 10 queries each

Limitations of the approach

- Deleting triples may guarantee privacy but not safety

- A safe anonymization instance (DS, O, P) preserves privacy for the union of O(DS) with external data

  Definition (generalization of the safety definition introduced in [10]):

  for any external dataset G, for every privacy query $p \in P$,

  for any tuple of constants $\underline{c}$,

  if $\underline{c} \in$ Answer(p, O(DS) U G) then $\underline{c} \in$ Answer(p, G)

Example

Privacy query

P: SELECT ?x WHERE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology.}

URIs of people seen by a member of a service in a hospital having an oncology department should not be disclosed.

Dataset DS to anonymize

:bob :seenBy :mary. :mary :member :service1.

:ann :seenBy :mary. :service1 :hasDept :oncology.

Anonymization operation

O1: DELETE {?x :seenBy ?y} WHERE {?x :seenBy ?y}

O1(DS)

:mary :member :service1. :service1 :hasDept :oncology.

External dataset G

:bob :seenBy :mary,

- Empty answer set for the privacy query P evaluated on O1(DS) and on G

but

- :bob is returned as answer of the privacy query P evaluated on O1(DS) U G

⇒ The problem for safety comes from a possible join between an internal and external URI **(:mary** in our example)

⇒ **Solution:** identify such critical URIs and replace them by blank nodes.

Critical terms of a query

- Result variables
- Join variables, URIs, literals
  - several occurrences in the query body
- Example: Query Q

  SELECT ?x ?y WHERE { ?x :seenBy ?z. ?z :specialistOf ?y.
  ?v a :VIP. ?v :isHospitalized true}

  - Critical terms: ?x, ?y, ?z, ?v
  - Q has two connected components,
    - G1 = { ?x :seenBy ?z. ?z :specialistOf ?y.}
    - G2 = { ?v a :VIP. ?v :isHospitalized true.}

    G2 does not contain any result variable

    => expresses a boolean condition for Q to be satisfied

A sufficient condition for safety of an anonymization instance (DS, O, P)

For every connected component Gc of all the privacy queries in P

1. **for all critical variable** or **URI x** in Gc, for all triple t in Gc where x appears and for each mapping μ such that $\mu(t) \in$ O(DS), **μ(x) is a blank node**
2. **each triple (s p v)** in Gc such that **v is a join literal** and **s is neither a join variable nor a join URI** has **no image in O(DS) by a mapping**
3. if Gc **does not contain any result variable**, then **there exists a triple pattern in Gc without any image i**n O(DS) **by a mapping**

Problem 3

Check /build safe anonymization operations

Problem 3: data-independent safety problem

- Build anonymizations that are guaranteed to be safe when applied to any input dataset.
- Our contribution (under submission)
  - **Algorithm 1**: build a sequence O1 of update queries that makes the sufficient condition for safety satisfied **on any updated dataset**
    - Preserves joins between blank nodes and thus some utility counting queries
    - Requires to build as many update queries as subsets of each connected component

    ⇒worst-case exponential complexity in the size of the privacy queries
  - **Algorithm 2: a polynomial approximation of Algorihm1**
    - construct a sequence O2 of update queries that replace, in each triple pattern, every critical term (variable or IRI) with a fresh blank node.
  - **Property: for any dataset DS**
    - **O1 and O2 are safe anonymizations**
    - **DS |= O1(DS) |= O2(DS)**

Algorithm1

Algorithm 1: Find update operations to ensure safety

Input : a privacy policy $\mathcal{P}$ of queries $P_i = \langle \bar{x}_i, G_i \rangle$

Output : a sequence of operations O which is safe for $\mathcal{P}$

function `find-safe-ops(`$\mathcal{P}$`):`
- Let $O = \langle\,\rangle$;
- **for** $P_i \in \mathcal{P}$ **do**
  - **forall the** *connected components* $G_c \subseteq G_i$ **do**
    - Let $I_V := [\,]$ and $I_L := [\,]$;
    - **forall the** $(s, p, o) \in G_c$ **do**
      - **if** $s \in \mathbf{V} \vee s \in \mathbf{I}$ **then** $I_V[s] = I_V[s] + 1$;
      - **if** $o \in \mathbf{V} \vee o \in \mathbf{I}$ **then** $I_V[o] = I_V[o] + 1$;
      - **if** $o \in \mathbf{L}$ **then** $I_L[o] := I_L[o] \cup \{(s, p, o)\}$;
    - Let $V_{crit} := \{v \mid I_V[v] > 1\} \cup \{v \mid v \in \bar{x}_i \wedge \exists \tau \in G_c \text{ s.t. } v \in \tau\}$;
    - Let $SG_c = \{X \mid X \subseteq G_c \wedge X \neq \emptyset\}$ ordered by decreasing size;
    - **forall the** $X \in SG_c$ **do**
      - Let $X' := X$ and $\bar{x}' = \{v \mid v \in V_{crit} \wedge \exists \tau \in X \text{ s.t. } v \in \tau\}$;
      - **forall the** $x \in \bar{x}'$ **do**
        - Let $b \in \mathbf{B}$ be a fresh blank node;
        - $X' := X'[x \leftarrow b]$;
      - $O := O + \langle$`DELETE` X `INSERT` X' `WHERE` X `isNotBlank(`$\bar{x}'$`)`$\rangle$
    - Let $L_{crit} := \{l \mid |I_L[l]| > 1\}$;
    - **forall the** $l \in L_{crit}$ **do**
      - Let $G' := \{(t, p, l) \mid (t, p, l) \in I_L[l] \wedge t \notin V_{crit} \wedge p \notin V_{crit}\}$;
      - **forall the** $\tau \in G'$ **do**
        - $O := O + \langle$`DELETE` τ `WHERE` $\tau\rangle$;
    - **if** $\bar{x}_i = \emptyset$ **then**
      - Let $\tau \in G_c$ `// non-deterministic choice`
      - $O := O + \langle$`DELETE` τ `WHERE` $G_c\rangle$
- **return** O;

Illustration by example

Privacy query

P: SELECT ?x WHERE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology.}

URIs of people seen by a member of a service in a hospital having an oncology department should not be disclosed.

First update query computed by Algorithm1:

O2: DELETE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology}
INSERT {_:b1 :seenBy _:b2. _:b2 :member _:b3. _:b3 :hasDept :oncology}
WHERE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology}

Resulting anonymization of DS:

:bob :seenBy :mary. :mary :member :service1.
:ann :seenBy :mary. :service1 :hasDept :oncology.

O2(DS):

_:1 :seenBy _:2 . _:2 :member _:3. _:3 :hasDept :oncology.
_:4 :seenBy _:5 . _:5 :member _:6. _:6 :hasDept :oncology.

Update query returned by Algorithm2:

O3: DELETE {?x :seenBy ?y. ?y' :member ?z. ?z' :hasDept :oncology}
INSERT {_:b1 :seenBy _:b2. _:b3 :member _:b4. _:b5 :hasDept :oncology}
WHERE {?x :seenBy ?y. ?y' :member ?z. ?z' :hasDept :oncology}

O3(DS): _:1 :seenBy _:2 . _:3 :member _:4. _:5 :hasDept :oncology.
_:6 :seenBy _:7 . _:8 :member _:9. _:10 :hasDept :oncology.

Counting utility queries preserved

- Given P with a single connected component, at least one result variable and no join literal, let O the result of Algorithm1 applied to {P}: for every dataset DS

$$Answer(Count(P), O(DS)) = Answer(Count(P), DS).$$

P: SELECT ?x WHERE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology.}

No disclosure of URIs of people seen by a member of a service in a hospital having an oncology department

Count (P)

While preserving the number of people seen by a member of a service in a hospital having an oncology department

O2 guarantees it

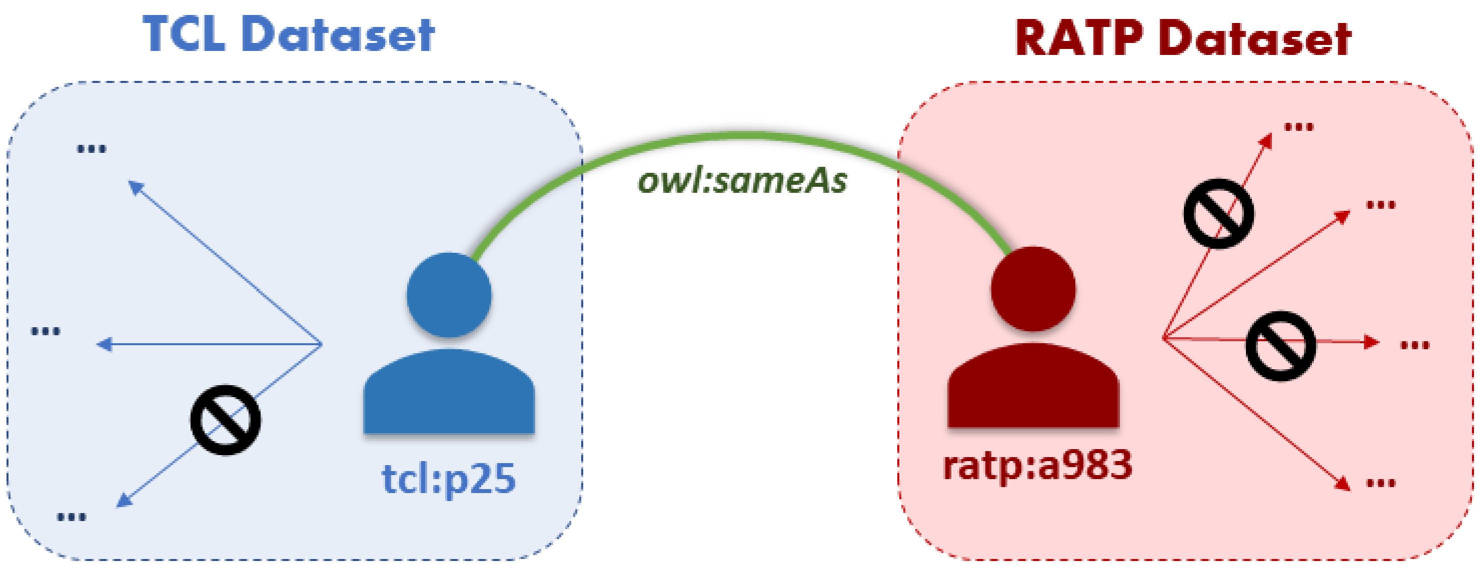
DELETE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology}

INSERT {_:b1 :seenBy _:b2. _:b2 :member _:b3. _:b3 :hasDept :oncology}

WHERE {?x :seenBy ?y. ?y :member ?z. ?z :hasDept :oncology}

but O3 does not

Safety modulo sameAs links



- sameAs links interpreted as equality between entities
- Semantics of answering queries modulo sameAs:

  a is an answer of Q over DS modulo a set sameAS of owl:sameAs links if there exists **o1 owl:sameAs o'1**,…, **ok owl:sameAs o'k** in closure(sameAs) such that **a is an answer of Q over DS'** where DS' is obtained from DS **by replacing each oi by o'i**

Theorem: Algorithm1 ensures safety modulo a set of **explicit** sameAs links between entities (including blank nodes)

Safety modulo sameAs links inferred by knowledge (e.g., OWL constraints)

- Functional or inverse functional properties
  - **inverse functionality of *bossOf*** expresses that every person has only one boss.

⇒ may lead to re-identifying blank nodes

DS = {:bob :seenBy :mary. :bob :bossOf _:b1. _:b1 :bossOf :ann.}

O(DS) = {_:b :seenBy :mary. _:b :bossOf _:b1. _:b1 :bossOf :ann.}

G = {:bob :bossOf :jim. :jim :bossOf :ann.}

  - From O(DS)U G_0 and the inverse functionality of :bossOf, it can be inferred
    - :jim :sameAs _:b1
    - :bob :sameAs _:b

    ⇒_:b is re-identified as :bob, which is returned as answer of P over O(DS)U G modulo sameAs, and

    ⇒ the anonymization operation O is not safe anymore

A possible solution

- add a privacy query for each functional property p

  SELECT ?x WHERE {?x p ?y.}

- and for each inverse functional property q

  SELECT ?x WHERE {?y q ?x.}

⇒the update queries returned by our algorithms will replace

  - each URI in subject position of a functional property by a fresh blank node,
  - and each URI in an object position of an inverse functional property by a fresh blank node.

  ⇒ in the previous example, :ann in (:b1:bossOf:ann) would be replaced by a fresh blank node.

Safety modulo completeness of a property

- Closure of a property available as an external source
  - suppose that the closure of the property :seenBy is known as being stored in G':

    :bob :seenBy :mary. :alice :seenBy :mary.

    :john :seenBy :ann. :tim :seenBy :ann.
  - knowing that G' is the complete extension of :seenBy allows to infer _:b :sameAs :bob and thus to re-identify the blank node _:b.
- Possible solution:
  - add a privacy query SELECT ?x ?y WHERE {?x p ?y }

  for each property p for which we suspect that a closure could occur in the LOD.

Conclusion

- A query-based approach for specifying privacy and utility policies
- Algorithms for building anonymization operations as update queries
  - Soundness and complexity
  - Data-independent
- Future directions:
  - Measure the loss of information of anonymization operations
  - Study the robustness to additional knowledge
  - Consider the data-dependent version of the safety problem to see if it could lead to more specific anonymization operations while guaranteeing safety.
  - Combine our logical approach with other approaches

Bibliography

[1] Dwork, C.: Differential privacy. In: ICALP 2006

[2] Machanavajjhala, A., He, X., Hay, M.: Differential privacy in the wild: A tutorial on current practices & open challenges. PVLDB 9(13), (2016)

[3] Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5), 2002

[4] Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. TKDD 1(1), 3 (2007)

[5]Heitmann, B., Hermsen, F., Decker, S.: k - rdf-neighbourhood anonymity: Combining structural and attribute-based anonymisation for linked data. In: PrivOn@ISWC. CEUR Workshop Proceedings, vol. 1951. 2017

[6] Kirrane, S., Mileo, A., Decker, S.: Access control and the resource description framework: A survey. Semantic Web 8(2), 2017

[7] Villata, S., Delaforge, N., Gandon, F., Gyrard, A.: An access control model for linked data. In: OTM Workshops. LNCS, vol. 7046, 2011

[8] T. Millstein, A.Levy, M. Friedman, Query Containment for Data Integration Systems. Proceedings PODS 2000

[9] Delanaux, R., Bonifati, A., Rousset, M., Thion, R.: Query-based linked data anonymization. In: ISWC 2018,

[10] Grau, B.C., Kostylev, E.V.: Logical foundations of privacy-preserving publishing of linked data. In: AAAI 2016