RDF graph anonymization robust to data linkage

Companion appendix

Remy Delanaux[1], Angela Bonifati[1], Marie-Christine Rousset[2,3], and Romuald Thion[1]

[1] Université Lyon 1, LIRIS CNRS, Villeurbanne, France
[name].[surname]@univ-lyon1.fr
[2] Université Grenoble Alpes, CNRS, INRIA, Grenoble INP, Grenoble, France
[name].[surname]@imag.fr
[3] Institut Universitaire de France, Paris, France

1 Additional illustrative examples

We use the same example graph G and policy P as in the main paper, with the privacy policy stating that IRIs of people seen by a member of a service in a hospital having an oncology department should not be disclosed.

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

G that is made of the following triples:

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

Example 1 illustrates the strategy consisting in replacing only one constant per triple by a blank node so as to break chains in the RDF graph likely to enable mappings from join terms in the privacy query. As shown by the example, this strategy does not guarantee safety for all the anonymization instances. Then, Example 2 shows that it is sometimes mandatory to replace all the constants in some critical triples by blank nodes in order to guarantee safety. While the latter might seem a pervasive operation, it still preserves some utility, such as the possibility of evaluating joins and counting queries.

Example 1. Let O_2 be the following update query:

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

Applying the anonymization operation O_2 to G results in the anonymized RDF graph $O_2(G)$ made of the following triples:

```
_:b1 :seenBy :mary.  _:b2 :member :service1.  _:b3 :hasDept :oncology.
```

$O_2(G)$ is a safe anonymization, since it is impossible to force the mapping from the query path {`?y :member ?z. ?z :has_dept :oncology`} to $O_2(G)$ because of the distinct blank nodes. Then the only way to find a mapping is to have a corresponding path in G'. But, the union with $O_2(G)$ will just produce `_:b1` as answer to the privacy query, which is not a constant.

Now, let O_3 be the following update query:

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

Like O_2, O_3 replaces only one constant per triple by a blank node, as shown in the result $O_3(G)$:

```
_:b1 :seenBy :mary.  :mary :member _:b2.  _:b2 :hasDept :oncology.
```

In contrast with $O_2(G)$, $O_3(G)$ is not safe (while still preserving privacy), as its union with an external RDF graph G' containing the triple (`:bob`, `:seenBy`, `:mary`) would return the constant `:bob` as an answer to the query P.

Example 2. Since Example 1 shows that replacing only one constant per triple may not guarantee safety, we have to consider update queries replacing all the constants in some triples by blank nodes, like the following update query O_4:

```
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}
```

The result RDF graph $O_4(G)$ is made of the following triples:

```
_:b1 :seenBy _:b2.  _:b2 :member _:b3.  _:b3 :hasDept :oncology.
```

Similarly to $O_2(G)$, it can be shown that $O_4(G)$ is safe. In addition, since all the occurrences of the join variables are replaced by the same blank nodes, the result of the counting query $\mathsf{Count}(P)$ is preserved i.e. it returns the same value as when evaluated on the original RDF graph G. This is not true anymore if we apply the following update query O_5 which breaks all join terms:

```
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}
```

In fact $O_5(G)$ is more general than $O_4(G)$, i.e., $O_4(G) \models O_5(G)$.

All these examples show the necessity of studying the different strategies to produce safe anonymization operations according to some desired utility, which we address in the main paper.

2 Proofs

2.1 Technical lemmas

First a classical fact on the monotonicity of conjunctive queries.

Lemma 1. *Let GP and GP' be graph patterns with $GP \subseteq GP'$. Moreover, let G and G' be RDF graphs with $G \subseteq G'$. The following inclusions hold for all sets $\bar{x}$ of variables of GP.*

$$\mathsf{Ans}(\langle \bar{x}, GP' \rangle, G) \subseteq \mathsf{Ans}(\langle \bar{x}, GP \rangle, G) \subseteq \mathsf{Ans}(\langle \bar{x}, GP \rangle, G')$$

Proof. Let $\bar{c} \in \mathsf{Ans}(\langle \bar{x}, GP' \rangle, G)$. By definition, there exists some μ such that $\mu(\bar{x}) = \bar{c}$ and $\mu(GP') \subseteq G$ and in particular $\mu(GP) \subseteq \mu(GP')$. By the transitivity of inclusion, both inclusion holds.

Now, we show by induction on the length of chains that subgraphs forming a partition of a connected component must be connected together through a join term.

Lemma 2. *Let GP_1 and GP_2 be a partition of a same connected component GP, then there exists a triple $t_1 \in GP_1$ and a triple $t_2 \in GP_2$ with a (join) term in common.*

Proof. Since GP_1 and GP_2 are not empty, there exists a triple $t_1 \in GP_1$ and a triple $t_2 \in GP_2$. Assume that t_1 an t_2 do not have a constant or variable in common in subject or object position. We proceed by contradiction.

Let c_1 be a constant or a variable in t_1 and let c_2 be a constant or variable of t_2. Since c_1 and c_2 appear in the same connected component GP, there exists a path of length n from c_1 to c_2, i.e., there exist chains $p^1, \ldots, p^n$ and $c^1, \ldots, c^{n+1}$ such that $c_1 = c^1$, $c_2 = c^{n+1}$ and for every $i \in [1, n]$ either $(c^i, p^i, c^{i+1}) \in GP$ or $(c^{i+1}, p^i, c^i) \in GP$.

Let k the greatest index such that *all* triples (c^i, p^i, c^{i+1}) or (c^{i+1}, p^i, c^i) with $i \leq k$ are in GP_1. If $k = n$, then $c^{k+1} = c_2$ and c_2 is common to two triples in GP_1 and GP_2, a contradiction. Otherwise, if $k < n$, then $(c^{k+1}, p^{k+1}, c^{k+2})$ or $(c^{k+2}, p^{k+1}, c^{k+1})$ is in GP_2 and c^{k+1} is common to two triples in GP_1 and GP_2, a contradiction again.

2.2 Proof of Theorem 1

The proof of Theorem 1 is based on two separate lemmas. First we deal with the case where the privacy query is *not* boolean with only one connected component. We show that replacing critical terms (i.e., result variables, join variables and join constants in subject or object position) with blank nodes guarantees that $(G, O, \{P\})$ is *safe* for any graph G.

Lemma 3. *Let $(G, O, \{P\})$ be an anonymization instance where $P = \langle \bar{x}, GP \rangle$ is a query made of a unique connected component and at least one distinguished variable. For all critical term x of GP, for all triple $\tau \in GP$ where x appears, for each anonymization mapping μ s.t. $\mu(\tau) \in O(G)$ if $\mu(x) \in \mathbf{B}$, then $(G, O, \{P\})$* is safe.

Proof. The main idea is to prevent linking $O(G)$ and G' together by using blank nodes. The key assumption that blank nodes from different graphs are disjoint.

We proceed by contradiction. Suppose that $(G, O, \{P\})$ is *not* safe, thus there exists a graph G', a tuple of constants $\bar{c} \in \mathsf{Ans}(P, O(G) \cup G')$ with $\bar{c} \notin \mathsf{Ans}(P, G')$. Therefore by the definition of Ans, there exists a mapping μ such that $\mu(\bar{x}) = \bar{c}$ and $\mu(GP) \subseteq O(G) \cup G'$ but $\mu(GP) \not\subseteq G'$. Let GP_1 be the antecedent of $\mu(GP) \cap O(G)$, that is, GP_1 is the largest subgraph of GP such that $\mu(GP_1) \subseteq O(G)$. Let GP_2 be its complement, $GP_2 = GP \setminus GP_1$.

If $GP_1 = \emptyset$, we are done, because this contradicts the assumption $\mu(GP) \not\subseteq G'$. Thus, there is some $\tau \in GP_1$. Now, if $GP_2 = \emptyset$, then all variables $x \in \bar{x}$ appear in GP_1, by hypothesis on $O(G)$, $\mu(x) \in \mathbf{B}$, but $\mu(x) \in \bar{c}$ forbids $\mu(x)$ to be a blank for all x, as $\bar{x}$ is not empty by hypothesis, we obtain a contradiction. Thus there is some $\tau' \in GP_2$. So both GP_1 and GP_2 are non empty and thus form a partition of GP.

By Lemma 2, there exist $\tau_1 \in GP_1$ and $\tau_2 \in GP_2$ with a join variable or a join constant in common, let x be this join variable or constant. By assumption, if $\mu(\tau_1) \in O(G)$ then $\mu(x)$ is a blank node of $O(G)$. However, blank nodes are local, so the blank nodes of $O(G)$ are disjoint from those of G', contradicting that $\mu(\tau_2) \in G'$ with $\mu(x)$ being a blank node of G'.

Note that it is important to consider anonymization mappings that can replace *IRIs and literals*, and not only variables. Indeed, the common node obtained by Lemma 2 may be either a variable, an IRI or a literal.

Now we deal with the corner case where the privacy query has no result variable. Indeed, in that particular case, replacing all critical variables and constants by blank nodes is not enough because such a transformation preserves the existence of images of boolean queries.

Lemma 4. *Let $(G, O, \{P\})$ be an anonymization instance where $P = \langle \emptyset, GP \rangle$ is a query with only one connected component. Assume that $(G, O, \{P\})$ already satisfies the conditions of Lemma 3. If there exists some triple pattern $\tau \in GP$ with no image in $O(G)$ by any anonymization mapping, then $(G, O, \{P\})$* is safe.

Proof. When the query is boolean, the safety condition amounts to check that if there exists an instance of GP is $O(G) \cup G'$, then there must be an instance of GP in G' alone. Basically, we have to enhance the proof of Lemma 3 to cover the case where $\bar{x} = \emptyset$. Assume that we have the same set up until we have proved that GP_1 is not empty. With the new condition added in the current, there exists some triple pattern $\tau'' \in GP_1$ with no image in $O(G)$, thus $\mu(GP_1) \not\subseteq O(G)$ and GP_2 cannot be empty. The rest of the proof is similar.

Lemma 5. *Let $A_1 = (G, O, \{Q_1\})$ and $A_2 = (G', O, \{Q_2\})$ be two safe anonymization instances where $Q_1 = \langle \bar{x}_1, GP_1 \rangle$ and $Q_2 = \langle \bar{x}_2, GP_2 \rangle$ are two queries made of exactly one connected component without common terms or variables. The anonymization instance $(G, O, \{Q\})$ with $Q = \langle \bar{x}_1 \cup \bar{x}_2, GP_1 \cup GP_2 \rangle$ is safe as well.*

Proof. Assume that there is some $\bar{c} \in \mathsf{Ans}(Q, O(G) \cup G')$ for an arbitrary G' where $\bar{c} = \bar{c}_1 \cup \bar{c}_2$ where c_1 (resp. c_2) is the restriction of $\bar{c}$ to the image of $\bar{x}_1$ (resp. $\bar{x}_2$).

As A_1 (resp. A_2) is safe, any tuple of constants $\bar{c}_1$ (resp. $\bar{c}_2$) found when evaluating Q_1 (resp. Q_2) on $O(G) \cup G'$ already exists in G'. Therefore there exists a mapping μ_1 such that $\mu_1(\bar{x}_1) = \bar{c}_1$ and $\mu_1(GP_1) \subseteq G'$ (resp. μ_2 s.t. $\mu_2(\bar{x}_2) = \bar{c}_2$ and $\mu_2(GP_2) \subseteq G'$). Since GP_1 and GP_2 are disjoint, they constitute two different connected components of $GP_1 \cup GP_2$. We can create a mapping μ' such that $\mu'(GP_1 \cup GP_2) \subseteq G'$ using μ_1 for the variables of Q_1 and μ_2 for the variables of Q_2. Finally, $\bar{c} = \bar{c}_1 \cup \bar{c}_2 = \mu_1(\bar{x}_1) \cup \mu_2(\bar{x}_2) = \mu'(\bar{x}_1) \cup \mu'(\bar{x}_2)$, therefore, $\bar{c}$ is an answer of Q over G which concludes the proof.

Now we can assemble all the pieces to obtain the proof of Theorem 1 that we recall here:

Theorem 1. *An anonymization instance $(G, O, \mathcal{P})$ is* safe *if the following conditions hold for every connected component GP_c of all privacy queries $P \in \mathcal{P}$:*

- **(i)** *for all critical term x of GP_c, for all triple $\tau \in GP_c$ where x appears, for each anonymization mapping μ s.t. $\mu(\tau) \in O(G)$, $\mu(x) \in \mathbf{B}$ holds;*
- **(ii)** *if GP_c does not contain any result variable, then there exists a triple pattern of GP_c without any image in $O(G)$ by an anonymization mapping.*

Proof. To show that $(G, O, \mathcal{P})$ with $\mathcal{P} = \{P_1, \ldots, P_n\}$ is safe we have to show that each $(G, O, \{P_i\})$ is safe. The conditions of Theorem 1 are exactly those of Lemma 3 and Lemma 4 for each connected component P_i^j of P_i, thus each $(G, O, \{P_i^j\})$ is safe.

Consider $\bar{c} \in \mathsf{Ans}(P_i, O(G) \cup G')$. Let $GP_i = \mathsf{body}(P_i)$ and let $GP_i^j = \mathsf{body}(P_i^j)$. By the definition of Ans, there exists a mapping μ such that $\mu(\bar{x}) = \bar{c}$ and $\mu(GP_i) \subseteq O(G) \cup G'$. By definition $GP_i^j \subseteq GP^i$ holds, thus by the monotonicity of queries (Lemma 1), $\mu(GP_i^j) \subseteq O(G) \cup G'$ holds as well for each connected component GP_i^j. Since each P_i^j is safe, there exists μ^j such that $\mu^j(GP_i^j) \subseteq G'$ for each GP_i^j. As the subgraphs GP_i^j are connected components, they cannot share terms. By induction on the number of connected components, using repeatedly Lemma 5, we can construct an anonymization mapping μ' such that $\mu'(GP_i^j) \subseteq G'$ for all GP_i^j, but $GP_i = \bigcup GP_i^j$, so $\mu'(GP_i) \subseteq G'$. We are now left to prove that $\mu'(\bar{x}) = \bar{c}$. For each variable $x \in \bar{x}$, there is a unique connected component $GP_i^{j_x}$ where x appears and $\bar{c}$ is nothing else but $\bigcup_{x \in \bar{x}} \mu^{j_x}(x)$.

Algorithm 1: Find update operations to ensure safety

Input : a set $\mathcal{P}$ of privacy conjunctive queries $P_i = \langle \bar{x}_i, GP_i \rangle$
Output : a sequence O of operations which is safe for $\mathcal{P}$

```
1  function find-safe-ops(P):
2      Let O = ⟨ ⟩;
3      for P_i ∈ P do
4          forall connected components GP_c ⊆ GP_i do
5              Let I := [ ];
6              forall (s, p, o) ∈ GP_c do
7                  if s ∈ V ∨ s ∈ I then I[s] = I[s] + 1;
8                  if o ∈ V ∨ o ∈ I ∨ o ∈ L then I[o] = I[o] + 1;
9              Let x̄_c := {v | v ∈ x̄_i ∧ ∃τ ∈ GP_c s.t. v ∈ τ};
10             Let T_crit := {t | I[t] > 1} ∪ x̄_c;
11             Let SGP_c = {X | X ⊆ GP_c ∧ X ≠ ∅ ∧ X is connected} ordered by
                decreasing size;
12             forall X ∈ SGP_c do
13                 Let X' := X and x̄' = {t | t ∈ T_crit ∧ ∃τ ∈ X s.t. t ∈ τ};
14                 forall x ∈ x̄' do
15                     Let b ∈ B be a fresh blank node;
16                     X' := X'[x ← b];
17                 O := O + ⟨DELETE X INSERT X' WHERE X isNotBlank(x̄')⟩
18             if x̄_c = ∅ then
19                 Let τ ∈ GP_c // non-deterministic choice
20                 O := O + ⟨DELETE τ WHERE GP_c⟩
21     return O;
```

2.3 Proof of Theorem 2

Theorem 2. *Let $O =$ `find-safe-ops`$(\mathcal{P})$ be the sequence of anonymization operations returned by Algorithm 1 applied to the set $\mathcal{P}$ of privacy queries: O is safe for $\mathcal{P}$. The worst-case computational complexity of Algorithm 1 is exponential in in the size of $\mathcal{P}$.*

Proof. The main idea is to show that the conditions of Theorem 1 are satisfied when O is executed on G. Let $\mathcal{P}$ be a privacy policy with queries $P_i = \langle \bar{x}_i, GP_i \rangle$, let $O =$ `find-safe-ops` $(\mathcal{P})$ and let G be an *arbitrary* RDF graph. For each connected component $GP_c \subseteq GP_i$ of each privacy query P_i, Algorithm 1 generates

- a sequence of operations at Line 17, mimicking condition (i) of Theorem 1;
- a non-deterministic delete operation at Line 20, mimicking condition (ii).

Conditions (i) and (ii) encapsulated in Theorem 1 correspond respectively to Lemmas 3 and 4. If we show that for each connected component GP_c Algorithm 1 generates a sequence of operation satisfying conditions (i) and (ii), the safety of O for $\mathcal{P}$ will follow from Theorem 1.

First of all, in Algorithm 1, please remark that by construction T_{crit} is the set of critical terms. Also, please note that the order of operations *is* relevant. Indeed, Lemmas 3 and 4 are cumulative: the second one relies on the first one. For GP_c a connected component of the privacy query $P_i = \langle \bar{x}_i, GP_i \rangle$, let $\bar{x}_c$ be the subset of variables from $\bar{x}_i$ that appear in GP_c. We consider both conditions separately.

Condition (i) We have $\bar{x}_c \neq \emptyset$. The generated operations $O = \langle O_1, \ldots, O_p \rangle$ are only those from Line 17, the operations at Line 20 is not triggered. Let x be a join variable and let $GP_x \subseteq GP_c$ the (unique) subset of GP_c where x appears. Let GP'_x be the largest subset of GP_x with an image in $O(G)$, call μ the mapping s.t., $\mu(GP'_x) \subseteq O(G)$. We have to show that $\mu(x) \in \mathbf{B}$. Let $m = |GP'_x|$ be the cardinality of GP'_x. There exists one operation $O_k \in O$ of the form $O_k =$ `DELETE` X_k `INSERT` X'_k `WHERE` X_k `isNotBlank`(x) which is triggered by μ when $k = m$, that is when $X_k = GP'_x$ in the loop at Line 12. If there is no such O_k, the condition is vacuously satisfied as $GP'_x = \emptyset$. The operation O_k is exactly the replacement of all $\mu(x)$ by the fresh blank b of Line 15 thus $\mu(x) \in \mathbf{B}$ and the condition (i) is satisfied, hence the safety of O for P_i by Lemma 3.

Note that the construct `isNotBlank`(x) ensures that O_k is unique in O if it exists, indeed, if $\mu(x) \in \mathbf{B}$ for some k, no other other O_l with $l \geq k$ is triggered by the definition of update queries. Hence, this condition is not needed to ensure safety but ensures that the operation in O do not destruct too much information.

Condition (ii) Now GP_c is a query with no result variable. The operation `DELETE` τ `WHERE` GP_c at Line 20 deletes all occurrences of τ such that $\mu(\tau) \in G$ thus, the condition (ii) required by Lemma 4 is satisfied and GP_c is safe.

For the worst-case complexity, the size considered is defined here as $\mathsf{size}(\mathcal{P}) = \Sigma_{P_i \in \mathcal{P}} |GP_i|$. The loop at Line 12 is executed exactly $2^n - 1$ times where n is the size of the largest connected component of G_i. Thus considering for instance $\mathcal{P} = \{P\}$ where P's body is made of a single component of cardinality n, the loop generates an exponential number of operations at Line 17.

2.4 Proof of Theorem 3

Theorem 3. *The worst-case computational complexity of Algorithm 2 is polynomial in the size of $\mathcal{P}$. Let O and O' be the result of applying respectively Algorithm 1 and Algorithm 2 (with the same non deterministic choices) to a set $\mathcal{P}$ of privacy queries: for any graph G, $(G, O', \mathcal{P})$* is safe *and $G \models O(G)$ and $O(G) \models O'(G)$.*

Proof. The proof of the safety of Algorithm 2 is similar to Theorem 2, the only difference lies at *Condition (i)*, the operations for *Condition (ii)* being the same. The actual difference between Algorithm 1 and Algorithm 2 is that the latter generates a fresh blank node *triple by triple* breaking multiple occurrences of join variables instead of preserving them. Indeed, for each join variable x of the connected component GP_c under scrutiny, the operation generated at Line 17,

Algorithm 2: Find update operations to ensure safety modulo sameAs

Input : a privacy policy $\mathcal{P}$ of queries $P_i = \langle \bar{x}_i, GP_i \rangle$
Output : a sequence of operations O safe modulo sameAs for $\mathcal{P}$)

```
1  function find-safe-ops-sameas(P):
2      Let O = ⟨ ⟩;
3      for P_i ∈ P do
4          forall connected components GP_c ⊆ GP_i do
               /* [...Lines 3 to 10 identical to find-safe-ops...]     */
10             Let x̄' := {v | v ∈ x̄_i ∧ ∃τ ∈ GP_c s.t. v ∈ τ};
11             Let T_crit := {t | I[t] > 1} ∪ x̄';
12             forall τ ∈ GP_c do
13                 forall x ∈ T_crit do
                       /* Every occurrence of a critical x is replaced by
                          a different blank node                          */
14                     G' := {τ[x ← [ ]]};
15                     Let v ∈ V a fresh variable;
16                     G'' := {τ[x ← v]};
17                 O := O + ⟨DELETE G'' INSERT G' WHERE G'' isNotBlank(x̄')⟩;
               /* [...End of algorithm identical to find-safe-ops...]  */
18     return O;
```

call it O_c, replaces it with a fresh blank node. There is exactly one such O_c for each GP_c, thus the output of Algorithm 2 is polynomial: the source of the exponential complexity of Algorithm 1 has been eliminated. Indeed, the loop at Line 12 do not browse the subsets $SGP_c \subseteq GP_c$ anymore but only its elements $\tau \in GP_c$.

Recall that for two graphs with blank nodes $A \models B$ amounts to show that there exists some substitution μ of blank nodes to terms such that $\nu(B) \subseteq A$. The point is thus to exhibit such a mapping.

First of all, if $G' \subseteq G$ then clearly $G \models G'$ (pick ν as the identity), so the property holds for all delete operations in O which are common to both Algorithm 1 and Algorithm 2. Theses operations being the very same ones when the same non-deterministic choices are made by the two algorithms.

It is left to show that the update operations (which are not mere deletions) guarantee $G \models O(G)$ as well. The argument is the core of the proof of Theorem 4: Algorithm 1 constructs a bijective μ renaming of join IRIs to blank nodes with inverse μ_1^{-1} with $\mu_1^{-1}(O(G)) = G$ which implies that $G \models O(G)$.

We are left to prove that $O(G) \models O'(G)$. For Algorithm 2, we observe that the renaming is not a function anymore because the same literal that occurs multiple times is mapped to *different blank* nodes. However, by construction as each occurrence of a critical τ is replaced by a fresh blank node, there exists some μ_2^{-1} from blank nodes to IRIs such that $\mu_2^{-1}(O'(G)) = G$. So now, construct μ_3^{-1} such that $\mathsf{dom}(\mu_3^{-1}) = \mathsf{dom}(\mu_2^{-1})$ defined by $\mu_3^{-1} = \mu_1 \circ \mu_2^{-1}$ which maps each

blank node generated by O' to the one obtained by O. The mapping μ_3^{-1} is such that $\mu_3^{-1}(O'(G)) = \mu_1(\mu_2^{-1}(O'(G))) = \mu_1(G) = O(G)$ thus $O(G) \models O'(G)$.

2.5 Proof of Theorem 4

Theorem 4. *Let $O =$ `find-safe-ops`$(\{P\})$ be the output of Algorithm 1 applied to a privacy query P with no boolean connected component. For every RDF graph G, $O(G)$ satisfies the the condition* $\mathsf{Ans}(\mathsf{Count}(P), O(G)) \geq \mathsf{Ans}(\mathsf{Count}(P), G)$.

Proof. The conditions of Theorem 4 are those of Lemma 3. The sufficient condition to ensure safety is thus to replace critical IRIs and variables by blank nodes: no `DELETE` operation is performed. The key point to prove Property 4 is that `INSERT` operations produced by Algorithm 1 are in fact a one-to-many renaming of IRIs to blank nodes. Indeed in the update query produced ate Line 17, each critical term is replaced by a *fresh* blank node, but no two different IRIs are mapped to the same one.

2.6 Proof of Theorem 5

Theorem 5. *Let O be the result of applying Algorithm 1 to a set $\mathcal{P}$ of privacy queries: for any set* sameAs *of explicit `:sameAs` links, O* is safe modulo sameAs *for $\mathcal{P}$.*

Proof. The proof is similar to the proof of Theorem 2, the only case of interest being that of condition (i), the rest is similar. Let G be the RDF graph to anonymize, G' be an external graph, and $P = \langle \bar{x}, GP \rangle$ be a privacy query. Let $\bar{c}$ be a tuple of constants such that $\bar{c} \in \mathsf{Ans}_{\mathsf{sameAs}}(P, O(G) \cup G')$.

By Definition 11 $\exists (b_0, :\mathsf{sameAs}, b'_0), ..., (b_n, :\mathsf{sameAs}, b'_n) \subseteq \mathsf{closure}(\mathsf{sameAs})$ statements, such that $\bar{c} \in \mathsf{Ans}(P, (O(G) \cup G')[b_i \leftarrow b'_i])$.

Consider the largest subset $GP_1 \subseteq GP$ such that $\mu(GP_1) \subseteq O(G)[b_i \leftarrow b'_i]$ and its complement $GP_2 = GP \setminus GP_1 \subseteq G'[b_i \leftarrow b'_i]$. With an argument similar to the proof of Lemma 3, GP_1 and GP_2 constitute a partition of GP so there must exist $\tau_1 \in GP_1$ and $\tau_2 \in GP_2$ with a join term variable or IRI call it x. Thus, there exist terms b_k and b'_k such that $\mu(x) = b'_k$ for some k with $(b_k : \mathsf{sameAs}, b'_k) \in \mathsf{closure}(\mathsf{sameAs})$. On the one hand, b_k is a term of $O(G)[b_i \leftarrow b'_i]$ because $\tau_1 \in G_1$, on the other hand b_k is a term of $G'[b_i \leftarrow b'_i]$ because $\tau_1 \in G_2$.

We have to show that $(b_k : \mathsf{sameAs}, b'_k)$ cannot be found in $\mathsf{closure}(\mathsf{sameAs})$. Recall that $\mathsf{closure}(\mathsf{sameAs})$ is the reflexive, symmetric and transitive closure of sameAs. The proof is by structural induction on the proof of $(b_k : \mathsf{sameAs}, b'_k) \in \mathsf{closure}(\mathsf{sameAs})$, consider the last deduction rule used to conclude that $(b_k : \mathsf{sameAs}, b'_k) \in \mathsf{closure}(\mathsf{sameAs})$:

Case $(b_k : \mathsf{sameAs}, b'_k) \in \mathsf{sameAs}$**.** The operation at Line 17 of Algorithm 1 ensures that b_k is a fresh blank of $O(G)$ which cannot appear in G' or in sameAs since blank nodes are only local to the graph they appear in, a contradiction.

Case $(b_k : \mathsf{sameAs}, b_k) \in \mathsf{closure}(\mathsf{sameAs})$ **by reflexivity.** This is the nominal case of Lemma 3 with an argument similar to the previous one.

Case $(b_k : \mathsf{sameAs}, b'_k) \in \mathsf{closure}(\mathsf{sameAs})$ **by symmetry.** A contradiction again by the inductive hypothesis $(b'_k, : \mathsf{sameAs}, b_k) \in \mathsf{closure}(\mathsf{sameAs})$.

Case $(b_k : \mathsf{sameAs}, b'_k) \in \mathsf{closure}(\mathsf{sameAs})$ **by transitivity.** By hypothesis there exists b''_k such that $(b_k : \mathsf{sameAs}, b''_k) \in \mathsf{closure}(\mathsf{sameAs})$ and $(b''_k : \mathsf{sameAs}, b'_k) \in \mathsf{closure}(\mathsf{sameAs})$. Here the contradiction is on $(b_k : \mathsf{sameAs}, b''_k) \in \mathsf{closure}(\mathsf{sameAs})$.

3 Reference privacy policies

3.1 TCL graph

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?user ?name
WHERE {
  ?user foaf:givenName ?name .
}
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?user ?surname
WHERE {
  ?user foaf:familyName ?surname .
}
```

```
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
SELECT ?user ?address
WHERE {
  ?user vcard:hasAddress ?address .
}
```

```
PREFIX tcl: <http://localhost/>
SELECT ?user ?birth
WHERE {
  ?user tcl:birthday ?birth .
}
```

```
PREFIX datex: <http://vocab.datex.org/terms#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX tcl: <http://localhost/>
SELECT ?name ?surname ?startSubDate
WHERE {
```

```
  ?user a tcl:User .
  ?user foaf:givenName ?name .
  ?user foaf:familyName ?surname .
  ?user datex:subscription ?subDate .
  ?tabDate datex:subscriptionStartTime ?startSubDate .
}
```

```
PREFIX datex: <http://vocab.datex.org/terms#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX tcl: <http://localhost/>
SELECT ?name ?surname  ?endSubDate
WHERE {
  ?user a tcl:User .
  ?user foaf:givenName ?name .
  ?user foaf:familyName ?surname .
  ?user datex:subscription ?subDate .
  ?tabDate datex:subscriptionStopTime ?endSubDate .
}
```

```
PREFIX datex: <http://vocab.datex.org/terms#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX tcl: <http://localhost/>
SELECT ?name ?surname ?subType
WHERE {
  ?user a tcl:User .
  ?user foaf:givenName ?name .
  ?user foaf:familyName ?surname .
  ?user datex:subscription ?subDate .
  ?tabDate datex:subscriptionReference ?subType .
}
```

3.2 Swedish Heritage graph

```
PREFIX dc: <http://purl.org/dc/elements/1.1/> .
PREFIX dcterm: <http://purl.org/dc/terms/> .
SELECT ?c ?t ?m
WHERE {
  ?s dc:creator ?c .
  ?s dcterm:spatial ?t .
  ?s dcterm:temporal ?m .
}
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/> .
PREFIX dcterm: <http://purl.org/dc/terms/> .
SELECT ?c ?d ?m
WHERE {
  ?s dc:creator ?c .
  ?s dc:description ?d .
  ?s dcterm:temporal ?m .
}
```

3.3 Drugbank graph

```
PREFIX dbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/> .
SELECT ?b
WHERE {
  ?d  a dbank:drugs.
  ?d  dbank:affectedOrganism  ?o .
  ?d  dbank:brandedDrug ?b.
}
```

```
PREFIX dbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/> .
SELECT ?b
WHERE {
  ?d  a dbank:drugs.
  ?d  dbank:affectedOrganism  ?o .
  ?d  dbank:brandName ?b.
}
```