Quotient RDF Summaries
Based on Type Hierarchies

Pawel Guzewicz
Université Paris-Saclay,
Inria and École Polytechnique, France
pawel.guzewicz@telecom-paristech.fr

Ioana Manolescu
Inria and École Polytechnique,
Université Paris-Saclay, France
ioana.manolescu@inria.fr

Abstract—Summarization has been applied to RDF graphs to obtain a compact representation thereof, easier to grasp by human users. We present a new brand of quotient-based RDF graph summaries, whose main novelty is to summarize together RDF nodes belonging to the same type hierarchy. We argue that such summaries bring more useful information to users about the structure and semantics of an RDF graph.

I. INTRODUCTION

The structure of RDF graphs is often complex and heterogeneous, making them hard to understand for users who are not familiar with them. This problem has been encountered in the past in the data management community, when dealing with other semi-structured graph data formats, such as the Object Exchange Model (OEM, in short) [1].

Structural summaries for RDF graphs. To help discover and exploit such graphs, [2, 3] have proposed using Dataguide summaries to represent compactly a (potentially large) data graph by a smaller one, computed from it. In contrast with relational databases where the schema is fixed before it is populated with data (a priori schema), a summary is computed from the data (a posteriori schema). Each node from the summary graph represents, in some sense, a set of nodes from the input graph. Many other graph summarization proposals have been made, for OEM [4], later for XML trees with IDREF links across tree nodes (thus turning an XML database into a graph) [5, 6, 7], and more recently for RDF [8, 9, 10, 11, 12, 13]; many more works have appeared in this area, some of which are presented in a recent tutorial [14]. Related areas are concerned with graph compression, e.g. [15], ontology summarization [16] (focusing more on the graph semantics than on its data) etc.

Quotient-based summaries are a particular family of summaries, computed based on a (summary-specific) notion of equivalence among graph nodes. Given an equivalence relation ≡, for each equivalence class C (that is, maximal set of graph nodes comprising nodes all equivalent to each other), the summary has exactly one node nc in the summary. Example of quotient-based summaries include [4, 5, 6, 7, 17, 11, 10, 12]: other summaries (including Dataguides) are not quotient-based.

This work is placed within the quotient-based RDF summarization framework introduced in [12]. That framework adapts the principles of quotient-based summarization to RDF graphs, in particular preserves the semantics (ontology), which may come with an RDF graph, in its summary. This is important as it guarantees that any summary defined within the framework is representative, that is: a query having answers on an RDF graph, has answers on its summary. This allows to use summaries as a first user interface with the data, guiding query formulation. Note that here, query answers take into account both the data explicitly present in the RDF graph, and the data implicitly present in the graph, through reasoning based on the explicit data and the graph’s ontology.

Two RDF summaries introduced in [11] have been subsequently [13] redefined as quotients. They differ in their treatment of the types which may be attached to RDF graph nodes. One is focused on summarizing the structure (non-type triples) first and copies type information to summary nodes afterwards; this may erase the distinctions between resources of very different types, leading to confusing summaries. The other starts by separating nodes according to their sets of types (recall that an RDF node may have one or several types, which may or may not be related to each other). This ignores the relationships which may hold among the different classes present in an RDF graph.

Contribution and outline. To simultaneously avoid the drawbacks of the two proposals above, in this paper we introduce a novel summary based on the same framework. It features a refined treatment of the type information present in an RDF graph, so that RDF graph nodes which are of related types are represented together in the summary. We argue that such a summary is more intuitive and more informative to potential users of the RDF graph.

The paper is organized as follows. We recall the RDF graph summarization framework introduced in [12] which frames our work, as well as the two abovementioned concrete summaries. Then, we formally define our novel summary, and briefly discuss a summarization algorithm and its concrete applicability.

II. RDF GRAPHS AND SUMMARIES

A. RDF and RDF Schema

We view an RDF graph G as a set of triples of the form s p o. A triple states that its subject s has the property p, and the value of that property is the object o. We consider only well-formed triples, as per the RDF specification [19], using uniform resource identifiers (URIs), typed or untyped literals (constants) and blank nodes (unknown URIs or literals).
The RDF standard [19] includes the property rdf:type (τ in short), which allows specifying the type(s) or class(es) of a resource. Each resource can have zero, one or several types, which may or may not be related. We call the set of G triples, whose property is τ, the type triples of G, denoted TG.

**RDF Schema and entailment.** G may include a schema (ontology), denoted SG, and expressed through RDF Schema (RDFS) triples using one of the following standard properties: subclass, subproperty, domain and range, which we denote by the symbols \(<_{sc}\), \(<_{sp}\), \(\rightarrow_d\) and \(\rightarrow_r\), respectively. Our proposal is beneficial in the presence of \(<_{sc}\) schema statements; we do not constrain SG in any way.

RDF entailment is the mechanism through which implicit RDF triples are derived from explicit triples and schema information. In this work, we consider four entailment rules, each based on one of the four properties above: (i) \(c_1 \prec_{sc} c_2\) means any resource of type \(c_1\) is also of type \(c_2\); (ii) \(p_1 \prec_{sp} p_2\) means that as long as a triple \(s p_1 o\) belongs to G, the triple \(s p_2 o\) also holds in G; (iii) \(p \prec_{d} c\) means that any resource \(s\) having the property \(p\) in G is of type \(c\), that is, \(s \tau c\) holds in G; finally (iv) \(p \prec_{r} c\) means that any resource that is a value of the property \(p\) in G, is also of type \(c\).

The fixpoint obtained by applying entailment rules on the triples of G and the schema rules in SG until no new triple is entailed, is termed saturation (or closure) of G and denoted \(G^\infty\). The saturation of an RDF graph is unique (up to blank node renaming), and does not contain implicit triples (they have all been made explicit by saturation).

We view an RDF graph \(G\) as: \(G = SG \cup TG \cup DG\), where the schema \(SG\) and the type triples \(TG\) have been defined above; \(DG\) contains all the remaining triples, whose property is neither \(\tau\) nor \(<_{sc}\) or \(<_{sp}\) or \(\rightarrow_d\) or \(\rightarrow_r\). We call \(DG\) the data triples of \(G\).

In the presence of an RDFS ontology, the semantics of an RDF graph is its saturation; in particular, the answers to a query posed on \(G\) must take into account all triples in \(G^\infty\) [19].

Figure 1 shows an RDF graph we will use for illustration in the paper. Schema nodes and triples are shown in blue; type triples are shown in dotted lines; boxed nodes denote URIs of classes and instances, while d1, d2, e1 etc. denote literal nodes; “desc” stands for “description”.

**B. Quotient RDF summaries**

We recall the summarization framework introduced in [12]. In a graph \(G\), a class node is an URI appearing as subject or object in a \(<_{sc}\) triple, as object in a \(\rightarrow_d\) or \(\rightarrow_r\) triple, or as object in a τ triple. A property node is an URI appearing as subject or object in a \(<_{sp}\) triple, or as subject in a \(\rightarrow_d\) or \(\rightarrow_r\) triple. The framework brings a generic notion of equivalence among RDF nodes:

**Definition 1:** (RDF equivalence) Let \(\equiv\) be a binary relation between the nodes of an RDF graph. We say \(\equiv\) is an RDF equivalence relation iff (i) \(\equiv\) is reflexive, symmetric and transitive, (ii) any class node is equivalent w.r.t. \(\equiv\) to itself, and (iii) any property node is equivalent w.r.t. \(\equiv\) to itself.

Graph nodes which are equivalent will be summarized (or represented) by the same node in the summary. The reason behind class and property nodes being only equivalent to themselves in every RDF equivalence relation, is to ensure that each such node is preserved in the summary, as they appear in the schema and carry important information for the graph’s semantics. A summary is defined as follows:

**Definition 2:** (RDF summary) Given an RDF graph \(G\) and an RDF node equivalence relation \(\equiv\), the summary of \(G\) by \(\equiv\) is an RDF graph denoted \(G_{/\equiv}\) and defined as follows:

- \(G_{/\equiv}\) contains exactly one node for each equivalence class of G nodes through \(\equiv\); each such node has a distinct, “fresh” URI (that does not appear in G).
- For each triple \(s p o\) in G such that \(s_{/\equiv}\), \(o_{/\equiv}\) are the \(G_{/\equiv}\) nodes corresponding to the equivalence classes of \(s\) and \(o\), the triple \(s_{/\equiv} p o_{/\equiv}\) belongs to \(G_{/\equiv}\).

The above definition can also be stated “\(G_{/\equiv}\) is the quotient graph of G by the equivalence relation \(\equiv\)”, based on the classical notion of quotient graph\[^1\] We make two observations:

- Regardless of the chosen \(\equiv\), all SG triples are also part of \(G_{/\equiv}\), as class and property nodes are represented by themselves, and thanks to the way \(G_{/\equiv}\) edges are defined; indeed, G and \(G_{/\equiv}\) have the same schema;
- No particular treatment is given to type triples: how to take them into account is left to each individual \(\equiv\).

Different RDF equivalence relations lead to different summaries. At one extreme, if all data nodes are equivalent, the summary has a single data node; on the contrary, if \(\equiv\) is “empty” (each node is equivalent only to itself), the summary degenerates into G itself. Well-studied equivalence relations for graph quotient summaries are based on the so-called forward, backward, or forward and backward (FB) bisimulation [4], [5], [6], [7]. It has been noted though, e.g. in [20], that RDF graphs

\[^1\]https://en.wikipedia.org/wiki/Quotient_graph
exhibit so much structural heterogeneity that bisimulation-based summaries are very large, almost of the size of G, thus not very useful. In contrast, \[\equiv\] relations which lead to compact summaries, many orders of magnitude smaller than the original graphs.

C. Types in summarization: first or last?

Let us consider how type triples can be used in quotient RDF summaries. Two approaches have been studied in the literature, and in particular in quotient summaries. The approach we will call data-first focuses on summarizing the data (or structure) of G, and then carries (or copies) the possible types of G nodes, to the summary nodes representing them. Conversely, type-first approaches summarize graph nodes first (or only) by their types. Below, we recall two quotient summaries described in \[\[11, 18\]\], which are the starting point of this work; they are both very compact, and illustrate the data-first and type-first approaches respectively. They both rely on the notion of property cliques:

**Definition 3:** (PROPERTY RELATIONS AND CLIQUES) Let \(p_1, p_2\) be two data properties in DG:

1) \(p_1, p_2 \in G\) are source-related iff either: (i) a data node in DG is the subject of both \(p_1\) and \(p_2\), or (ii) DG holds a data node \(r\) and a data property \(p_3\) such that \(r\) is the subject of \(p_1\) and \(p_3\), with \(p_3\) and \(p_2\) being source-related.

2) \(p_1, p_2 \in G\) are target-related iff either: (i) a data node in DG is the object of both \(p_1\) and \(p_2\), or (ii) DG holds a data node \(r\) and a data property \(p_3\) such that \(r\) is the object of \(p_1\) and \(p_3\), with \(p_3\) and \(p_2\) being target-related.

A maximal set of properties in DG which are pairwise source-related (respectively, target-related) is called a source (respectively, target) property clique.

For example, in Figure 1 the properties \(email\) and \(webpage\) are source-related since Alice is the subject of both; \(webpage\) and \(officeHours\) are source-related due to Bob; also due to Alice, \(registeredIn\) and \(attends\) are source-related to the above properties, leading to a source clique \(SC_1 = \{attends, email, webpage, officeHours, registeredIn\}\). Another source clique is \(SC_2 = \{desc, givenIn\}\).

It is easy to see that the set of non-empty source (or target) property cliques is a partition over the data properties of DG. Further, all data properties of a resource \(r \in G\) are all in the same source clique, which we denote \(SC(r)\); similarly, all the properties of which \(r\) is a value are in the same target clique, denoted \(TC(r)\). If \(r\) is not the value of any property (respectively, has no property), we consider its target (respectively, source) is \(\emptyset\). For instance, in our example, \(SC_1\) is the source clique of Alice, Bob, Carole and David, while \(SC_2\) is the source clique of the BigDataMaster and of the HadoopCourse.

**Definition 4:** (WEAK EQUIVALENCE) Two data nodes are weakly equivalent, denoted \(v_1 \equiv_T v_2\), iff: (i) they have the same non-empty source or non-empty target clique, or (ii) they both have empty source and empty target cliques, or (iii) they are both weakly equivalent to another node of G.

![Figure 2: Weak summary of the sample RDF graph in Figure 1](Image 337x498 to 538x738)

**Definition 5:** (WEAK SUMMARY) The weak summary of the graph G, denoted \(G_{\equiv_T}\), is the RDF summary obtained from the weak equivalence \(\equiv_G\).

Figure 2 shows the weak summary of our sample RDF graph. The URLs \(W_1\) to \(W_6\) are “new” summary nodes, representing literals and/or URIs from G. Thus, \(W_3\) represents Alice, Bob, Carole and David together, due to their common source clique \(SC_1\), \(W_7\) represents the course and the master program, due to their common source clique \(SC_2\). Note that the \(givenIn\) edge from G leads to a summary edge from \(W_2\) to itself; also, \(W_1\) carries over the types of the nodes it represents, thus it is both of type MasterProgram and MasterCourse. This example shows that data-first summarization may represent together G resources whose types clearly indicate their different meaning; this may be confusing.

In contrast, the typed weak summary recalled below illustrates the type-first approach:

Let \(\equiv_T\) be an RDF equivalence relation which holds on two nodes iff they have the exact same set of types.

Let \(\equiv_{D_{W}}\) be an RDF equivalence relation which holds on two nodes iff (i) they have no type, and (ii) they are weakly equivalent.

**Definition 6:** (TYPED WEAK SUMMARY) The typed weak summary of an RDF graph G, denoted \(G_{\equiv_{D_{W}}}\), is the summary through \(\equiv_{D_{W}}\) of the summary through \(\equiv_T\) of G:

\[G_{\equiv_{D_{W}}} = (G_{\equiv_T})_{\equiv_{D_{W}}}\]

This double-quotient summarization acts as follows. First, nodes are grouped by their sets of types (inner quotient through \(\equiv_T\)); second, untyped nodes only are grouped according to weak (structural) equivalence.

For instance, our sample G has six typed data nodes (Alice to David, BigDataMaster and HadoopCourse), each of which has a set of exactly one type; all these types are different. Thus, \(\equiv_T\) is empty, and \(G_{\equiv_{D_{W}}}\) (drawing omitted) has eight typed
nodes $UTW_1$ to $UTW_8$, each with a distinct type and the property(ies) of one of these nodes. We now consider $G$’s eight untyped data nodes. We have $d_1 \equiv_w d_2$ due to their common target clique \{desc\}, and similarly $w_1 \equiv_w w_2$ and $h_2 \equiv_y h_3 \equiv_y h_4$. Thus, $G_{TV}$ has four untyped nodes, each of which is an object of desc, email, webpage and respectively officeHours triples.

The typed weak summary, as well as other type-first summaries, e.g. [17], also have limitations:

- They are defined based on the type triples of $G$, which may change through saturation, leading to different $G_{TV}$ summaries for conceptually the same graph (as all $G$ leading to the same $G^\infty$ are equivalent). Thus, for a type-first summary to be most meaningful, one should build it on the saturated graph $G^\infty$. Note the reason for saturation at Figure 8 in [18].
- They do not exploit the relationships which the ontology may state among the types. For instance, AssistantProfessor nodes like Carole are summarized separately from Professors like David, although they are all instructors.

III. Summarization Aware of Type Hierarchies

A. Novel type-based RDF equivalence

Our first goal is to define an RDF equivalence relation which:

1) takes type information into account, thus belongs to the “types-first” approach;
2) leads (through Definition [2]) to a summary which represents together, to the extent possible (see below), nodes that have the same most general type.

Formally, let $C = \{c_1, c_2, \ldots\}$ be the set of class nodes present in $G$ (that is, in SG and/or in TG). We can view these nodes as organized in a directed graph where there is an edge $c_1 \rightarrow c_2$ as long as $G$’s saturated schema $SG^\infty$ states that $c_1$ is a subclass of $c_2$. By a slight abuse of notation, we use $C$ to also refer to this graph. In principle, $C$ could have cycles, but this does not appear to correspond to meaningful schema designs. Therefore, we assume without loss of generality that $C$ is a directed acyclic graph (DAG, in short). In Figure 1 $C$ is the DAG comprising the eight (blue) class nodes and edges between them; this DAG has four roots.

First, assume that $C$ is a tree, e.g., with Instructor as a root type and PhDStudent, AssistantProfessor as its subclasses. In such a case, we would like instances of all the abovementioned types to be represented together, because they are all instances of the top type Instructor. This extends easily to the case when $C$ is a forest, e.g., a second type hierarchy in $C$ could feature

\footnote{Ontology languages such as RDF Schema or OWL feature a top type, that is a supertype of any other type, such as rdfs:Resource. We do not include such a generic, top type in $C$.}

\footnote{If $C$ has cycles, the types in each cycle can all be seen as equivalent, as each is a specialization of all the other, and could be replaced by a single (new) type in a simplified ontology. The process can be repeated until $C$ becomes a DAG, then the approach below can be applied, following which the simplified types can be restored, replacing the ones we introduced. We omit the details.}

Figure 3: Sample RDF schema and min-size cover of the corresponding $C$.

a root type Paper whose subclasses are ConferencePaper, JournalPaper etc. In this case, we aim to represent all authors together because they are instances of Paper.

In general, though, $C$ may not be a forest, but instead it may be a graph where some classes have multiple superclasses, potentially unrelated. For instance, in Figure 1 PhDStudent has two superclasses, Student and Instructor. Therefore, it is not possible to represent $G$ nodes of type PhDStudent based on their most general type, because they have more than one such type. Representing them twice (once as Instructor, once as Student) would violate the framework (Definition 2), in which any summary is a quotient and thus, each $G$ node must be represented by exactly one summary node.

To represent resources as much as possible according to their most general type, we proceed as follows.

**Definition 7:** (Tree Cover) Given a DAG $C$, we call a tree cover of $C$ a set of trees such that: (i) each node in $C$ appears in exactly one tree; (ii) together, they contain all the nodes of $C$; and (iii) each $C$ edge appears either in one tree or connects the root of one tree to a node in another.

Given $C$ admits many tree covers, however, it can be shown that there exists a tree cover with the least possible number of trees, which we will call min-size cover. This cover can be computed in a single traversal of the graph by creating a tree root exactly from each $C$ node having two supertypes such that none is a supertype of the other, and attaching to it all its descendants which are not themselves roots of another tree. For instance, the RDF schema from Figure 1 leads to a min-size cover of five trees:

- A tree rooted at Instructor and the edges connecting it to its children AssistantProfessor and Professor;  
- A single-node tree rooted at PhDStudent;  
- A tree rooted at Student with its child MasterStudent;  
- A single-node tree for MasterProgram and another for MasterCourse.

Figure 3 illustrates min-size covers on a more complex RDF schema, consisting of the types $A$ to $Q$. Every arrow goes from a type to one of its supertypes (for readability, the figure does not include all the implicit subclass relationships, e.g., that $E$ is also a subclass of $H$, $I$, $J$ etc.). The pink areas each denote
a tree in the corresponding min-size cover. $H$ and $L$ are tree roots because they have multiple, unrelated supertypes.

To complete our proposal, we need to make an extra hypothesis on $G$:

\[
(\dagger) \quad \text{Whenever a data node } n \text{ is of two distinct types } c_1, c_2 \text{ which are not in the same tree in the min-size tree cover of } C, \text{ then (i) } c_1 \text{ and } c_2 \text{ have some common subclasses, (ii) among these, there exists a class } c_{1,2} \text{ that is a superclass of all the others, and (iii) } n \text{ is of type } c_{1,2}.
\]

For instance, in our example, hypothesis $(\dagger)$ states that if a node $n$ is an Instructor and a Student, these two types must have a common subclass (in our case, this is PhDStudent), and $n$ must be of type PhDStudent. The hypothesis would be violated if there was another common subclass of Instructor and Student, say MusicLover\(^4\) that was neither a subclass of PhDStudent nor a superclass of it.

$(\dagger)$ may be checked by a SPARQL query on $G$. While it may not hold, we have not found such counter-examples in a set of RDF graphs we have examined (see Section \[\Box\]). In particular, $(\dagger)$ immediately holds in the frequent case when $C$ is a tree (taxonomy) or, more generally, a forest: in such cases, the min-size cover of $C$ is exactly its set of trees, and any types $c_1, c_2$ of a data node $n$ are in the same tree.

When $(\dagger)$ holds, we can state:

**Lemma 1 (Lowest branching type):** Let $G$ be an RDF graph satisfying $(\dagger)$, $n$ be a data node in $G$, $c_s, n$ be the set of types of $n$ in $G$, and $c_s^n$ be the classes from $c_s$ together with all their supertypes (according to the saturated schema of $G$). Assume that $c_s^n \neq \emptyset$.

Then there exists a type $lbt_n$, called lowest branching type, such that:

- $c_s^n = c_s' \cup c_s''$, where $\{lbt_n\} \in c_s'$ and $c_s''$ may be empty;
- the types in $c_s'$ (if any) can be arranged in a tree according to $\leq_s$ relation between them, and the most general one is $lbt_n$;
- if $c_s''$ is not empty, it is at least of size two, and all its types are superclasses of $lbt_n$.

**Proof:** Let’s assume to the contrary that there exists an RDF graph $G_1$ satisfying $(\dagger)$, a node $n$ in $G_1$, $c_s, n$ the set of types of $n$, $c_s^n \neq \emptyset$ is the set of types of $n$ with all their supertypes (according to saturated schema of $G_1$) and there is no lowest branching type for $n$.

Let $G$ be the set of all such RDF graphs and let $G$ be the graph containing a node $n$ that violates the lemma and such that $|c_s^n|$ is the smallest, across all such lemma-violating nodes $n$ in any graph from $G$.

Let $k = |c_s^n|$. Note that $k > 0$ by definition. Let’s consider the cases:

1) $k = 1$ In this case, the lemma trivially holds.

2) $k \geq 2$ In this case, let $t_1, \ldots, t_k$ be the types of node $n$ (their order not important). Let’s consider graph $G'$ which is the same as $G$ but without node $n$ having type $t_j$. From the way we chose $G$ and $G'$, $G'$ satisfies the lemma, thus there exists a lowest branching type $lbt_n$ for $n$ in $G'$. Now, let’s add $t_k$ to the types of $n$ in $G$. There are 3 possibilities:

a) $t_k$ is a subclass of $lbt_n$. Then $lbt_n$ is also lowest branching type after this addition.

b) $t_k$ is a superclass of $lbt_n$. If it’s the only superclass of $lbt_n$ then $t_k$ is a new lowest branching type, else $n$ still admits the lowest branching type $lbt_n$.

c) $t_k$ is neither a super- nor a subclass of $lbt_n$. Then it is in another tree in min-size cover of $G$, thus by $(\dagger)$ it follows that $t_k$ and some other type between $t_1, \ldots, t_{k-1}$ have a common subtype which serves as a lowest branching type for $n$.

From the above discussion we conclude that the node $n$ for which $k = |c_s^n|$ is not the lemma counterexample with the smallest $k$, which contradicts the assumption we made when picking it! Therefore no graph exists in $G$, thus all $G$s satisfy the lemma. \(\Box\)

For instance, let $n$ be Bob in Figure \[\Box\] then $c_{s,n} = \{\text{PhDStudent}\}$, thus $c_{s,n}^\infty = \{\text{PhDStudent, Student, Instructor}\}$. In this case, $lbt_n$ is PhDStudent, $c_{s,n}'$ is $\{\text{PhDStudent}\}$ and $c_{s,n}''$ is $\{\text{Student, Instructor}\}$.

If we take $n$ to be Carole, $c_{s,n}^\infty = \{\text{AssistantProfessor, Instructor}\}$: no type from this set has two distinct superclasses, thus $c_{s,n}''$ must be empty, $lbt_{Carole}$ is Instructor, and $c_{s,n}'$ is $\{\text{AssistantProfessor, Instructor}\}$. By a similar reasoning, $lbt_{David}$ is Instructor, and $lbt_{Alice}$ is Student. When $n$ has a type without subclasses or superclasses, such as BigData-Master, it leads to $c_{s,n}''$ being empty, and $c_{s,n}'$ is $lbt_n$, the only type of $n$. Thus, $lbt_{BigDataMaster}$ is MasterProgram and $lbt_{HadoopCourse}$ is MasterCourse.

For a more complex example, recall the RDF schema in Figure \[\Box\] and let $n$ be a node of type $E$ in an RDF graph having this schema. In this case, $c_{s,n}$ is $\{\text{E, G, H, B, I, J}\}$, $lbt_n$ is $H$, $c_{s,n}'$ is $\{\text{E, G, H}\}$ while $c_{s,n}''$ is $\{\text{B, I, J}\}$. Based on Lemma \[\Box\] we define our novel notion of equivalence, reflecting the hierarchy among the types of $G$ data nodes:

**Definition 8:** (Type-Hierarchy Equivalence) Type hierarchy equivalence, denoted $\equiv_{th}$, is an RDF node equivalence relation defined as follows: two data nodes $n_1$ and $n_2$ are type-hierarchy equivalent, noted $n_1 \equiv_{th} n_2$, if $lbt_{n_1} = lbt_{n_2}$.

From the above discussion, it follows that Carole $\equiv_{th}$ David, matching the intuition that they are both instructors and do not belong to other type hierarchies. In contrast, PhD students (such as Bob) are only type-hierarchy equivalent to each other; they are set apart by their dual Student and Instructor status. Master students such as Alice are only type-hierarchy equivalent among themselves, as they only belong to the student type hierarchy. Every other typed node of $G$ is only type-hierarchy equivalent to itself.
A. Constructing the weak type-hierarchy summary

B. RDF summary based on type hierarchy equivalence

IV. Algorithm and Applications

A. Constructing the weak type-hierarchy summary

B. Applicability

REFERENCES


