Query-Oriented Summarization of RDF Graphs

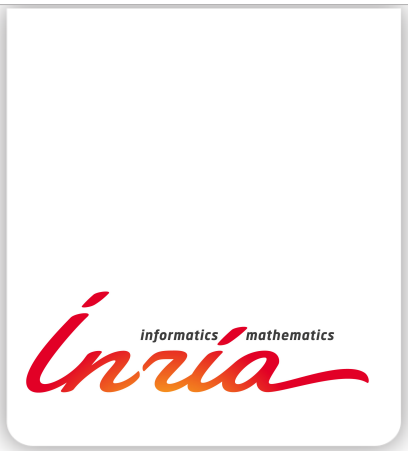
Šejla Čebirić, François Goasdoué, Ioana Manolescu

Query-Oriented Summarization of RDF Graphs

Šejla Čebirić, François Goasdoué, Ioana Manolescu

RESEARCH REPORT
N° 8920
June 2016
Project-Teams CEDAR

ISSN 0249-6399 ISRN INRIA/RR--8920--FR+ENG



Query-Oriented Summarization of RDF Graphs

Šejla Čebirić, François Goasdoué, Ioana Manolescu

Project-Teams CEDAR

Research Report n° 8920 — version 3 — initial version June 2016 — revised version February 2017 — 45 pages

Abstract: The Resource Description Framework (RDF) is the W3C's graph data model for Semantic Web applications. RDF graphs are often large and heterogeneous, thus users may have a hard time getting familiar with the structure and semantics of a graph.
We consider the problem of building *automatically, with no user input, compact RDF graph summaries which represent the complete structure and semantics of the graph, and are representative and accurate for a large useful dialect of SPARQL*. We provide the first technique for building out of an RDF graph a summary of its explicit and implicit data (the latter is due to RDF semantic constraints); a summary for which this is possible is termed *complete*. We introduce four novel summaries and show that two of them are complete. We provide a sufficient condition for RDF summarization completeness, and show that bisimulation-based summaries previously studied satisfy this condition.
We implemented a summarization tool and demonstrate its effectiveness through a set of experiments.

Key-words: Semantic Web, RDF, data summary, inference, reasoning, data compression

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Résumés orientés requêtes de graphes RDF

Résumé : RDF est le modèle de données du W3C, fondé sur les graphes, pour les applications du Web Sémantique. Les graphes RDF sont souvent larges et hétérogènes, compliquant la tâche des utilisateurs qui tentent de se familiariser avec leurs structure et leur sémantique. Nous étudions la construction *automatique, sans aucune information fournie par l'utilisateur, de résumés compacts de graphes RDF, qui représente leur structure et sémantique complète, et soient représentatifs et précis pour un dialecte large de requêtes SPARQL.* Nous sommes les premiers à présenter une technique pour construire, à partir d'un graphe RDF, un résumé de ses données explicites et implicites; ces dernières peuvent être présentes dans le graphe en vertu de ses contraintes sémantiques. Nous appellons un résumé pour lequel une telle technique existe, un résumé *complet*. Nous introduisons quatre nouveaux résumés de graphes, et montrons que deux d'entr'eux sont complets. Nous présentons une condition suffisante pour qu'un résumé RDF soit complet, et montrons que les résumés basés sur la bissimulation bidirectionnelle de graphes satisfont cette condition.

Nous avons développé un outil de résumé de graphes, et nous démontrons son intérêt et son efficacité de façon expérimentale, sur des données synthétiques ainsi que sur des données réelles.

Mots-clés : Web Sémantique, RDF, résumé de données, inférence, raisonnement, compression de données

1 Introduction

The Resource Description Framework (RDF) is a graph-based data model promoted by the W3C as the standard for Semantic Web applications. Its associated query language is SPARQL. RDF graphs are *varied*, coming from scientific applications, social or online media, government data etc. They are often *large*, and *heterogeneous*, i.e., resources described in an RDF graph may have very different sets of properties. An RDF resource may have one or several *types* (which may or may not be related to each other) or lack types. *RDF Schema* (RDFS) statements are sometimes part of an RDF graph, in which case they lead to **implicit data**. For instance, if the schema states that anyone *having a driver license* is a *Human* and we know *Bill has the driver license 123*, then the fact that *Bill* is a *Human* is implicitly present in the graph. According to the W3C RDF and SPARQL specification, **the semantics of an RDF graph $\mathtt{G}$, typically denoted $\mathtt{G}^{\infty}$, comprises both its explicit and implicit data**; in particular, SPARQL query answers must be computed *reflecting all the explicit and implicit data*. These features make an RDF graph complex, both structurally and conceptually; it is intrinsically hard to get familiar with it, especially if a schema is absent or describes only part of the data.

We study the problem of *RDF summarization*, that is: given any input RDF graph $\mathtt{G}$, *efficiently* build a *compact* RDF graph $\mathtt{G}_{/\equiv}$ which *summarizes both the structure and semantics of* $\mathtt{G}$. Our summaries can be used as a **GUI** with the help of a graph visualization library, e.g., Dot or Prefuse, and/or to **inform RDF processing tools about the complete graph structure and semantics**; for instance, they can be used to suggest queries to users exploring an unknown RDF graph. Toward this goal, we summarize *without any user input or parameter setting*. The literature features many summaries, e.g., [11, 9, 7] and some focus on RDF, e.g., [34, 21, 27] (see also Section 9). The main novelty of our work is to build from $\mathtt{G}$ a summary of **both its explicit and implicit data** (the latter may not be present in $\mathtt{G}$ although it holds there). We call such summaries *complete*, and we are the first to define a framework for complete summarization of semantic-rich RDF graphs. Our summaries can be built efficiently, are 2 to 6 orders of magnitude smaller than $\mathtt{G}$, and never larger than $\mathtt{G}$.

The contributions made in this work are as follows:

1. We define *RDF node equivalence relations* taking into account the particular features of RDF, and based on them define *RDF graph summaries as quotient graphs*. We show that any such summary is representative (has some answer for any query that has answers on $\mathtt{G}$) and accurate (any query having answers on the summary has answer on *some* graph which it summarizes[1]).

2. We provide a *sufficient condition* on an RDF node equivalence relation, for the RDF summary defined based on this relation to be *complete*. Moreover, we are the first to provide a concrete *method* for summarizing the explicit and implicit data without materializing the implicit data. As our experiments confirm, completeness is desirable as one can *get the full graph summary without spending the time to materialize and the space to store the implicit data*. We adapt the classic bisimulation-based summary definitions from the literature to our framework, and show that the *RDF bisimulation summaries defined in our framework are complete*. This novel result nicely complements existing knowledge about bisimulation summaries e.g., [25, 28, 19, 30, 7].

3. We introduce *four novel RDF summaries*, based on new RDF node equivalence relations (Section 4 and 5). We show that *two of them are complete* (based on our sufficient condition), while the *two others* are not. Interestingly, incompleteness turns out to be due to *separate treatment of RDF nodes having types*; this contrasts with the prominent role played by resource types in previous RDF summarization proposals [34, 21, 27]. We compare our summaries and analyze relationships between them in Section 6.

4. We have devised *summary building algorithms* (Section 7) and implemented our proposal in a summary building tool available for download at [1]. We demonstrate through *experiments* the strong size reduction through summarization for many real and synthetic datasets and the scalability of our summary building algorithms in Section 8.

[1] As commonly the case, many graphs may have the same summary.

Assertion	Triple	Relational notation
Class	$\mathtt{s}$ rdf:type $\mathtt{o}$	$\mathtt{o}(\mathtt{s})$
Property	$\mathtt{s}\ \mathtt{p}\ \mathtt{o}$	$\mathtt{p}(\mathtt{s}, \mathtt{o})$

Constraint	Triple	OWA interpretation
Subclass	$\mathtt{s} \prec_{sc} \mathtt{o}$	$\mathtt{s} \subseteq \mathtt{o}$
Subproperty	$\mathtt{s} \prec_{sp} \mathtt{o}$	$\mathtt{s} \subseteq \mathtt{o}$
Domain typing	$\mathtt{p} \hookleftarrow_{d} \mathtt{o}$	$\Pi_{\text{domain}}(\mathtt{s}) \subseteq \mathtt{o}$
Range typing	$\mathtt{p} \hookrightarrow_{r} \mathtt{o}$	$\Pi_{\text{range}}(\mathtt{s}) \subseteq \mathtt{o}$

Figure 1: RDF (top) & RDFS (bottom) statements.

As an image is worth a thousand words, the Web site [1] provides graphical representations of many sample summaries.

2 Preliminaries

We introduce RDF graphs and queries in Section 2.1, and requirements for our RDF summaries in Section 2.2.

2.1 RDF Graphs and Queries

An *RDF graph* (or *graph*, in short) is a set of *triples* of the form $\mathtt{s}\ \mathtt{p}\ \mathtt{o}$. A triple states that its *subject* $\mathtt{s}$ has the *property* $\mathtt{p}$, and the value of that property is the *object* $\mathtt{o}$. We consider only well-formed triples, as per the RDF specification [37], using uniform resource identifiers (URIs), typed or untyped literals (constants) and blank nodes (unknown URIs or literals). Blank nodes are essential features of RDF allowing to support *unknown URI/literal tokens*. These are conceptually similar to the labeled nulls or variables used in incomplete relational databases [2], as shown in [10].

Notations. We use $\mathtt{s}$, $\mathtt{p}$, and $\mathtt{o}$ as placeholders for subjects, properties and objects, respectively. Literals are shown as strings between quotes, e.g., "$string$". Figure 1 (top) shows how to use triples to describe resources, that is, to express class (unary relation) and property (binary relation) assertions. The RDF standard [37] has a set of *built-in classes and properties*, as part of the rdf: and rdfs: pre-defined namespaces. We use these namespaces exactly for these classes and properties, e.g., rdf:type specifies the class(es) to which a resource belongs. *For brevity, we will sometimes use τ to denote* rdf:type.

For example, the following RDF graph $\mathtt{G}$ describes a book, identified by doi_1: its author (a blank node _:b_1 related to the author name), title and date of publication:

$$\mathtt{G} = \begin{array}{l}\{\mathtt{doi}_1 \text{ rdf:type } \mathtt{Book},\ \ \mathtt{doi}_1\ \mathtt{writtenBy}\ \_{:}b_1,\\ \ \mathtt{doi}_1\ \mathtt{hasTitle}\ \text{“}\mathtt{Le\ Port\ des\ Brumes}''\text{,}\\ \ \_{:}b_1\ \mathtt{hasName}\ \text{“}\mathtt{G.\ Simenon}\text{”},\ \ \mathtt{doi}_1\ \mathtt{publishedIn}\ \text{“}\mathtt{1932}\text{”}\}\end{array}$$

RDF Schema (RDFS). RDFS allows enhancing the descriptions in RDF graphs by declaring *semantic constraints* between the classes and the properties they use. Figure 1 (bottom) shows the four main kinds of RDFS constraints, and how to express them through triples. For concision, we denote the properties expressing *subclass*, *subproperty*, *domain* and *range* constraints by the symbols $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ and $\hookrightarrow_r$, respectively. Here, "domain" denotes the first, and "range" the second attribute of every property. The RDFS constraints (Figure 1) are interpreted under the *open-world assumption (OWA)* [2], i.e., as *deductive* constraints: if the triples hasFriend rdfs:domain Person and Anne hasFriend Marie hold in the graph, then so does the triple Anne rdf:type Person. The latter is due to the domain constraint in Figure 1.

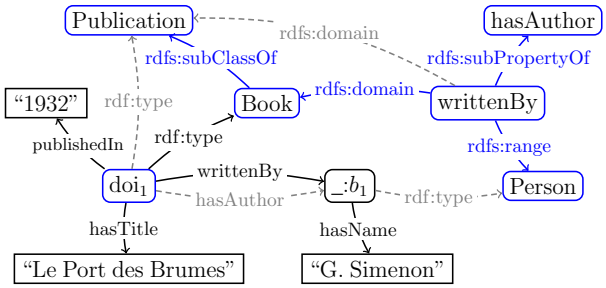


Figure 2: RDF graph and its saturation.

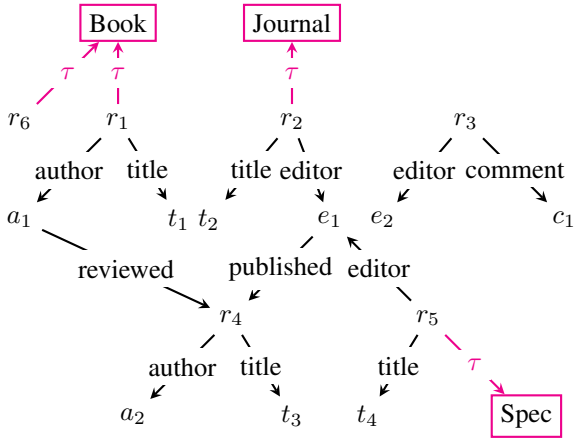


Figure 3: RDF graph used for summarization.

Implicit triples are an important RDF feature, considered part of the RDF graph even though they are not explicitly present in it, e.g., Anne rdf:type Person above. W3C names *RDF entailment* the mechanism through which, based on a set of explicit triples and some *entailment rules*, implicit RDF triples are derived. We denote by $\vdash^{i}_{\text{RDF}}$ *immediate entailment*, i.e., the process of deriving new triples through a *single* application of an entailment rule. More generally, a triple $\mathtt{s\ p\ o}$ is entailed by a graph $\mathtt{G}$, denoted $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$, if and only if there is a sequence of applications of immediate entailment rules that leads from $\mathtt{G}$ to $\mathtt{s\ p\ o}$ (where at each step, triples previously entailed are also taken into account).

For instance, assume that the RDF graph $\mathtt{G}$ above is extended with the constraints: Book $\prec_{sc}$ Publication, writtenBy $\prec_{sp}$ hasAuthor, writtenBy $\hookleftarrow_{d}$ Book and writtenBy $\hookrightarrow_{r}$ Person.

The resulting graph is depicted in Figure 2. Its implicit triples are those represented by dashed-line edges.

RDF graph saturation. The immediate entailment rules allow defining the finite *saturation* (a.k.a. closure) of an RDF graph $\mathtt{G}$, which is the RDF graph $\mathtt{G}^{\infty}$ defined as the fixpoint obtained by repeatedly applying $\vdash^{i}_{\text{RDF}}$ rules on $\mathtt{G}$.

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). An obvious connection holds between the triples entailed by a graph $\mathtt{G}$ and its saturation: $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$ if and only if $\mathtt{s\ p\ o} \in \mathtt{G}^{\infty}$.

The semantics of an RDF graph is its saturation. RDF entailment is part of the RDF standard itself; in particular, *the answers to a query posed on* $\mathtt{G}$ *must take into account all triples in* $\mathtt{G}^{\infty}$ [37].

For presentation purposes, we may use a *triple-based* or a *graph-based* representation of an RDF

graph:

1. The triple-based representation of an RDF graph. We see an RDF graph $\mathtt{G}$ as a *partition* of its triples into three components $\mathtt{G} = \langle \mathtt{D_G}, \mathtt{S_G}, \mathtt{T_G} \rangle$, where: (i) $\mathtt{S_G}$, the **schema** component, is the set of all $\mathtt{G}$ triples whose properties are $\prec_{sc}$, $\prec_{sp}$, $\hookleftarrow_d$ or $\hookrightarrow_r$; (ii) $\mathtt{T_G}$, the **type** component, is the set of τ triples from $\mathtt{G}$; (iii) $\mathtt{D_G}$, the **data** component, holds all the remaining triples of $\mathtt{G}$. Note that each of $\mathtt{D_G}$, $\mathtt{S_G}$, and $\mathtt{T_G}$ is an RDF graph by itself.

Further, we call *data property* any property $\mathtt{p}$ occurring in $\mathtt{D_G}$, and *data triple* any triple in $\mathtt{D_G}$. For illustration, we use the graph in Figure 3: class nodes are shown in purple boxes, while $\mathtt{T_G}$ triples appear as purple arrows; $\mathtt{D_G}$ consists of the triples shown in black, while $\mathtt{S_G}$ is empty. For brevity, *we will denote the data properties appearing in Figure 3 by their starting letters*: a, t, e, c, r and p.

2. The graph-based representation of an RDF graph. As per the RDF specification [37], *the set of nodes* of an RDF graph is the set of subjects and objects of triples in the graph, while its *edges* correspond to its triples. We define three categories of RDF graph nodes: (i) a *class node* is any node whose URI appears as subject or object of a $\prec_{sc}$ triple, or object of a $\hookleftarrow_d$ or $\hookrightarrow_r$ or τ triple; (ii) a *property node* is any node whose URI appears as subject/object of a $\prec_{sp}$ triple, or subject of a $\hookleftarrow_d$ or $\hookrightarrow_r$ triple, or property of a triple[2] (iii) a *data node* as any node that is neither a class nor a property node. Note that the sets of class nodes and of property nodes may intersect (indeed, nothing in the RDF specification forbids it), while class nodes are disjoint from data nodes, and property nodes are disjoint from data nodes, too.

Size and cardinality notations. We denote by $|\mathtt{G}|_n$ the number of nodes in a graph $\mathtt{G}$, and by $|\mathtt{G}|$ its number of edges. Further, for a given attribute $x \in \{\mathtt{s}, \mathtt{p}, \mathtt{o}\}$ and graph $\mathtt{G}$, we note $|\mathtt{G}|^0_x$ the number of distinct values of the attribute x within $\mathtt{G}$. For instance, $|\mathtt{D_G}|^0_{\mathtt{p}}$ is the number of distinct properties in the data component of $\mathtt{G}$.

Queries. We consider the SPARQL dialect consisting of *basic graph pattern* (BGP) queries, a.k.a. conjunctive queries, widely considered in research but also in real-world applications [20]. A BGP is a set of *triple patterns*, or triples in short; each triple has a subject, property and object, some of which can be variables.

Notations. We use the usual conjunctive query notation $q(\bar{x})$:- $t_1, \ldots, t_\alpha$, where $\{t_1, \ldots, t_\alpha\}$ are triple patterns; the query head variables $\bar{x}$ are called *distinguished variables*, and are a subset of those in $t_1, \ldots, t_\alpha$. For boolean queries, $\bar{x}$ is empty. The head of q is $q(\bar{x})$, its body is $t_1, \ldots, t_\alpha$; x, y, z, etc. denote variables.

Query answering. The *evaluation* of a query q against $\mathtt{G}$ has access only to the explicit triples of $\mathtt{G}$, thus it may fail to return the complete answer; the latter is obtained by evaluating q against $\mathtt{G}^\infty$. For instance, the query below asks for name of the author of "Le Port des Brumes":

$$q(x_3) \text{ :- } x_1 \ \mathtt{hasAuthor} \ x_2, \ x_2 \ \mathtt{hasName} \ x_3$$
$$x_1 \ \mathtt{hasTitle} \ \text{“}\mathtt{Le\ Port\ des\ Brumes}\text{”}$$

Its answer against the explicit and implicit triples of our sample graph is: $q(\mathtt{G}^\infty) = \{\langle \text{“G. Simenon”} \rangle\}$.

Note that evaluating q only against $\mathtt{G}$ leads to the empty answer, which is obviously incomplete. This is why, in keeping with the RDF and SPARQL standard, our work considers query *answering*.

2.2 Principles and Design Choices

We decide that *the summary $\mathtt{G}_{/\equiv}$ of an RDF graph $\mathtt{G}$ is an RDF graph itself*. This gives a natural basis for completeness, as RDF schema statements can be part of the summary; it also allows to use existing RDF tools to manipulate the summary. Further, we require summaries to satisfy the following conditions:

[2] A property node must be a *node*, i.e., merely appearing in a property position does not make an URI a property node; it also needs to appear a subject or object in the same or another triple.

- *Schema independence* It must be possible to summarize G whether or not it has RDF schema information.
- *No user input, no parameters* Summarization must not depend user input nor on manually-specified parameters, as the summary should help users get familiar with the data (and not assume this is already the case).
- *Semantic completeness* The summary of G must reflect all the explicit data and the implicit data, given that the semantic of G is its saturation.
- Tolerance to heterogeneity The summary should enable the recognition of "similar" resources despite some amount of heterogeneity in their properties and/or types

The following properties are of a more quantitative nature:

- *Compactness* The summary should be typically smaller than the RDF graph, ideally by many orders of magnitude.
- *Representativeness* The summary should preserve as much information as possible about the structure $\mathtt{G}$[3].
- *Accuracy* The summary should avoid, to the extent possible, reflecting data that does not exist in G.

To meet our goals of completeness and representativeness, we will not represent frequent graph structures differently / at the expense of the less frequent ones. Instead, we need a logical form of representation, based on queries, as we explain below.

Criteria for representativeness and accuracy. For *representativeness*, queries having answers on G should also have answers on the summary. Symmetrically, for *accuracy*, a query that can be answered on the summary, should also be answerable on the RDF graph itself.

Given an *RDF query language (dialect)* $\mathcal{Q}$, we define:

Definition 1. (QUERY-BASED REPRESENTATIVENESS) *Let* G *be any RDF graph.* $\mathtt{G}_{/\equiv}$ *is* $\mathcal{Q}$*-representative of* G *if and only if for any query* $q \in \mathcal{Q}$ *such that* $q(\mathtt{G}^\infty) \neq \emptyset$*, we have* $q((\mathtt{G}_{/\equiv})^\infty) \neq \emptyset$.

Representativeness is desirable as the summary can be used to help users formulate queries: therefore, it is important to reflect all graph patterns that may occur in the data.

To define accuracy, recall that *several graphs may have the same summary*, as summarization loses some information from the input graph; if two RDF graphs differ only with respect to that information, they have the same summary. We term *inverse set* of a summary $\mathtt{G}_{/\equiv}$, the set of all RDF graphs whose summary is $\mathtt{G}_{/\equiv}$ (the inverse set is purely conceptual, i.e., we never need to compute it). This leads to the accuracy criterion:

Definition 2. (QUERY-BASED ACCURACY) *Let* G *be any RDF graph,* $\mathtt{G}_{/\equiv}$ *its summary, and* $\mathcal{G}$ *the inverse set of* $\mathtt{G}_{/\equiv}$*. The summary* $\mathtt{G}_{/\equiv}$ *is* $\mathcal{Q}$-accurate *if for any query* $q \in \mathcal{Q}$ *such that* $q((\mathtt{G}_{/\equiv})^\infty) \neq \emptyset$*, there exists* $\mathtt{G}' \in \mathcal{G}$ *such that* $q(\mathtt{G}'^\infty) \neq \emptyset$.

The above characterizes the accuracy of a summary with respect to *any graph it may correspond to*.

Which dialect should we define representativeness and accuracy for? To get compact summaries, we decide they should not include subject and object values (whether they are URIs, literals or blank nodes); this follows the observation that there are many orders of magnitude more distinct subject and objects, than there are distinct properties in an RDF graph [38]. However, we chose to preserve types (classes) and data properties, as these are essential features of an RDF graph. These choices rule out representativeness for queries with URIs or literals in subjects or objects positions. Instead, we identify two large and useful classes of BGPs:

[3] Clearly, trade-offs exist between compactness and representativeness; we discuss them in Section 6.

Definition 3. (RELATIONAL AND RBGP* QUERIES) *A* relational BGP *(RBGP, in short) query is a BGP query whose body has: (i) URIs in all the property positions, (ii) a URI in the object position of every τ triple, and (iii) variables in any other positions.*

An extended relational *(RBGP*, in short) query is a BGP query whose body has (i) URIs or variables in all the property positions, (ii) a URI in the object position of every τ triple, and (iii) variables in any other positions.*

Both languages forbid URIs or literals in subject and object positions, and require that if type triples are specified in the query, the type is known. They differ in that RBGPs require URIs in the property positions, whereas RBGP* also allow variables there. Clearly, RBGPs are a restriction of RBGP*. A sample RBGP is:

$$q(x_1, x_3) \text{ :- } x_1\ \tau\ \texttt{Book},\ x_1\ \texttt{author}\ x_2,\ x_2\ \texttt{reviewed}\ x_3$$

while a similar RBGP* is:

$$q^*(x_1, x_3) \text{ :- } x_1\ \tau\ \texttt{Book},\ x_1\ \texttt{author}\ x_2,\ x_2\ y\ x_3$$

We define *RBGP* representativeness* and *RBGP* accuracy* by instantiating $\mathcal{Q}$ in Definition 1 and Definition 2, respectively, to RBGP* queries (Definition 3).

Discussion: preserving joins on literals While an RGBP (or RBGP*) query cannot enforce that the subject, property or object in a triple is a certain literal or URI (in other words, it cannot express selections), it does express joins. In particular, if the same literal appears in several places in the graph, an RBGP*-representative summary must preserve the joins enabled by these multiple occurrences. For instance, consider the graph:

$\texttt{s}_1$ rdf:type `city`	$\texttt{s}_1$ `zipCode` "91120"
$\texttt{s}_2$ rdf:type `company`	$\texttt{s}_2$ `employeeNo` "91120"
$\texttt{s}_3$ rdf:type `person`	$\texttt{s}_3$ `livesIn` "91120"

An RBGP* representative summary *must* have some results to the hypothetical query asking "cities and companies such that the zipcode of the first is the number of employees of the second", because on the above graph, the query does have results (due to $\texttt{s}_1$ and $\texttt{s}_2$). One may find this query meaningless, and argue that its non-emptiness on `G` is an accident. However, the very similar RGBP query "people living in the zipcode of a city" (which has results based on $\texttt{s}_1$ and $\texttt{s}_3$) is meaningful. In general, without human user input, it is hard to detect that the summary should preserve the results of one, and not of the other. Thus, in the remainder of the work, *we rely on RBGP* representativeness, knowing it preserves literal-based joins which may sometimes be seen as accidental*. To disable this, it suffices to redefine BGP (thus, RBGP and RBGP*) query semantics to consider that for query embedding purposes, two occurrences of the same literal in different places of the graph are not equal, and all our framework would adapt to this; we do not pursue this option further. (Note that the above discussion only concerns *joins on literals*; in contrast, joins on URI are a fundamental aspect of RDF (*linked*) data, and we consider they must be preserved by a meaningful summary.)

3 RDF summarization

Keeping in mind the requirements previously stated, to ensure sound-foundation summaries while still keeping open a space of options, we define our summaries through the classical notion of *quotient graph* from graph theory, parameterized by a *node equivalence relation*. We recall this below:

Definition 4. (QUOTIENT GRAPH) *Let A be a label set, $G = (V, E)$ be a labeled directed graph whose vertices are V, whose edges are $E \subseteq V \times V \times A$, with labels from A. Let $\equiv\ \subseteq V \times V$ be an equivalence relation over the nodes of V.*

The quotient graph of G using $\equiv$*, denoted* $G_{/\equiv}$*, is a labeled directed graph having* (i) *a node* n_S *for each set* S *of equivalent* V *nodes, and* (ii) *an edge* $n_{S_1} \xrightarrow{a} n_{S_2}$ *for some label* $a \in A$ *iff there exist two nodes* $n_1 \in S_1$ *and* $n_2 \in S_2$ *such that the edge* $n_1 \xrightarrow{a} n_2 \in E$*.*

Quotient-based graph summaries have been used in the past. The novelty of our work lies in the equivalence notions we define, and which in particular enable summary completeness w.r.t. explicit and implicit data. We will rely on:

Definition 5. (RDF NODE EQUIVALENCE) *Let* $\equiv$ *be a binary relation between the nodes of an RDF graph. We say* $\equiv$ *is an* RDF node equivalence relation *(or* RDF equivalence*, in short) iff* (i) $\equiv$ *is an equivalence relation in the classical sense (it is reflexive, symmetric and transitive),* (ii) *any class node is* $\equiv$ *only to itself, and* (iii) *any property node is* $\equiv$ *only to itself.*

where class and property nodes were introduced in Section 2.1.

Next, we first identify interesting relations between the data properties of an RDF graph (Section 3.1). Building on this, we define two RDF node equivalence relations based on data properties, as well as another based on node types (Section 3.2); we also reframe the well-known notion of graph bisimulation [15] in our RDF equivalence setting. We formalize RDF summaries and state some of the important properties they all enjoy (Section 3.3); our novel RDF equivalence relations will lead to defining different summaries in Section 4 and 5.

3.1 Data property relationships and cliques

We start by considering *relations between data properties* in a graph G. The simplest relationship is co-occurrence, when a resource is the source and/or target of two data properties. We generalize this into:

Definition 6. (PROPERTY RELATIONS AND CLIQUES) *Let* $\mathtt{p}_1$*,* $\mathtt{p}_2$ *be two data properties in* G*:*

1. $\mathtt{p}_1, \mathtt{p}_2 \in \mathtt{G}$ *are* source-related *iff either:* (i) *a data node in* G *is the subject of both* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, or* (ii) G *holds a data node* r *and a data property* $\mathtt{p}_3$ *such that* r *is the subject of* $\mathtt{p}_1$ *and* $\mathtt{p}_3$*, with* $\mathtt{p}_3$ *and* $\mathtt{p}_2$ *being source-related.*
2. $\mathtt{p}_1, \mathtt{p}_2 \in \mathtt{G}$ *are* target-related *iff either:* (i) *a data node in* G *is the object of both* $\mathtt{p}_1$ *and* $\mathtt{p}_2$*, or* (ii) G *holds a data node* r *and a data property* $\mathtt{p}_3$ *such that* r *is the object of* $\mathtt{p}_1$ *and* $\mathtt{p}_3$*, with* $\mathtt{p}_3$ *and* $\mathtt{p}_2$ *being target-related.*

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

For illustration, consider the sample graph in Figure 3. Here, properties a and t are source-related due to r_1 (condition (i) in the definition). Similarly, t and e are source-related due to r_2; consequently, a and e are source-related (condition (ii)). Properties r and p are target-related due to r_4. The non-empty source cliques are $SC_1 = \{a, t, e, c\}$, $SC_2 = \{r\}$ and $SC_3 = \{p\}$, whereas the non-empty target cliques are $TC_1 = \{a\}$, $TC_2 = \{t\}$, $TC_3 = \{e\}$, $TC_4 = \{c\}$ and $TC_5 = \{r, p\}$.

A source clique can be seen as "all the data properties of same-kind resources"; above, it makes sense to group together properties corresponding to various kinds of publications, such as author, title, editor, and comment. Similarly, a target clique comprises "the data properties of which same-kind resources are values".

It is easy to see that the set of non-empty source (or target) property cliques is a partition over the data properties of G. Further, if a resource $r \in \mathtt{G}$ has data properties, they are all in the same source clique; similarly, all the properties of which r is a value are in the same target clique. This allows us to refer to *the source (or target) clique of* r, denoted $SC(r)$ and $TC(r)$. If r is not the value of any property (respectively, has no property), we consider the target (respectively, source) clique of r to be $\emptyset$.

The target and source cliques of the resources in the graph shown in Figure 3 are shown in Table 1.

node	r_1	r_2	r_3	r_4	r_5
$SC(r)$	SC_1	SC_1	SC_1	SC_1	SC_1
$TC(r)$	$\emptyset$	$\emptyset$	$\emptyset$	TC_5	$\emptyset$
node	a_1	t_1	t_2	e_1	e_2
$SC(r)$	SC_2	$\emptyset$	$\emptyset$	SC_3	$\emptyset$
$TC(r)$	TC_1	TC_2	TC_2	TC_3	TC_3
node	c_1	t_4	a_2	t_3	r_6
$SC(r)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$TC(r)$	TC_4	TC_2	TC_1	TC_2	$\emptyset$

Table 1: Source and target cliques of the sample RDF graph.

Definition 7. (PROPERTY DISTANCE IN A CLIQUE) *The* distance *between two data properties* $\mathtt{p}, \mathtt{p}'$ *in a source (resp. target) clique is:*

- 0 *if there exists a resource in* $\mathtt{G}$ *that is the subject (resp. object) of both;*
- *otherwise, the smallest integer* n *such that* $\mathtt{G}$ *holds resources* $r_0, \ldots, r_n \in \mathtt{G}$ *and data properties* $\mathtt{p}_1, \ldots, \mathtt{p}_n$ *such that* r_0 *is the subject (resp. object) of* $\mathtt{p}$ *and* $\mathtt{p}_1$, r_1 *is the subject (resp. object) of* $\mathtt{p}_1$ *and* $\mathtt{p}_2$, *...,* r_n *is the subject (resp. object) of* $\mathtt{p}_n$ *and* $\mathtt{p}'$.

In Figure 3, the distance between a and t is 0 since r_1 has both. The distance between a and e is 1, while the distance between a and c is 2. Clearly, $\mathtt{p}$ and $\mathtt{p}'$ are at distance n for $n > 0$ iff a resource has both $\mathtt{p}$ and $\mathtt{p}''$, and further $\mathtt{p}''$ is at distance $n-1$ from $\mathtt{p}'$.

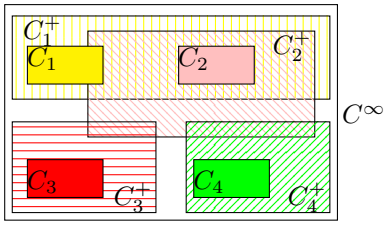


Figure 4: Sample cliques of $\mathtt{G}$, their saturations, and their enclosing clique in $\mathtt{G}^\infty$.

Source and target cliques are defined w.r.t. a given graph $\mathtt{G}$. What is the *impact of saturating* $\mathtt{G}$ *on the cliques*? In $\mathtt{G}^\infty$, every $\mathtt{G}$ resource has all the data properties it had in $\mathtt{G}$, therefore two data properties belonging to a $\mathtt{G}$ clique are also in the same clique of $\mathtt{G}^\infty$. Further, if the schema of $\mathtt{G}$ comprises $\prec_{sp}$ constraints, a resource may have in $\mathtt{G}^\infty$ a data property that it did not have in $\mathtt{G}$. In turn, this leads to $\mathtt{G}^\infty$ cliques which "subsume and fuse" several $\mathtt{G}$ cliques into one.

For a given clique C of $\mathtt{G}$, we call *saturated clique* and denote C^+ the of all the properties in C and all their generalizations (superproperties). Observe that C^+ reflects only C and the schema of $\mathtt{G}$; in particular, it does not reflect the data properties shared by resources in $\mathtt{G}^\infty$, and thus in general C^+ is *not* a clique of $\mathtt{G}^\infty$. The following Lemma characterizes the relationships between cliques of $\mathtt{G}$, their saturated versions C^+, and cliques of $\mathtt{G}^\infty$:

Lemma 1 (Saturation vs. property cliques)**.** *Let* C, C_1, C_2 *be distinct non-empty source (or target) cliques of* $\mathtt{G}$.

1. *There exists exactly one source (resp. target) clique* C^∞ *corresponding to* $\mathtt{G}^\infty$ *such that* $C \subseteq C^\infty$.

2. *If $C_1^+ \cap C_2^+ \neq \emptyset$ then all the properties in C_1 and C_2 are in the same $\mathtt{G}^\infty$ clique C^∞.*
3. *Any non-empty source (or target) clique C^∞ is a union of the form $C_1^+ \cup \cdots \cup C_k^+$ for some $k \geq 1$, where each C_i is a non-empty source (resp. target) clique of $\mathtt{G}$. Further,* the above saturated cliques are all connected through intersections*: for any C_i, C_j where $1 \leq i, j \leq k$ with $i \neq j$, there exist some cliques $D_1 = C_i, \ldots, D_n = C_j$ in the set $\{C_1, \ldots, C_k\}$ such that:*
$$D_1^+ \cap D_2^+ \neq \emptyset, \ldots, D_{n-1}^+ \cap D_n^+ \neq \emptyset$$
4. *Let $\mathtt{p}_1, \mathtt{p}_2$ be two data properties in $\mathtt{G}$, whose source (or target) cliques are C_1 and C_2. Properties $\mathtt{p}_1, \mathtt{p}_2$ are in the same source (resp. target) clique C^∞ corresponding to $\mathtt{G}^\infty$ if and only if there exist k non-empty source (resp. target) cliques of $\mathtt{G}$, $k \geq 0$, denoted $D_1, \ldots, D_k$ such that:*
$$C_1^+ \cap D_1^+ \neq \emptyset, D_1^+ \cap D_2^+ \neq \emptyset, \ldots, D_{k-1}^+ \cap D_k^+ \neq \emptyset, D_k^+ \cap C_2^+ \neq \emptyset.$$

Proof. We prove the lemma only for source cliques; the proof for the target cliques is very similar.

1. Any resource $r \in \mathtt{G}$ having two data properties also has them in $\mathtt{G}^\infty$; thus, any data properties in the same source clique in $\mathtt{G}$ are also in the same source clique in $\mathtt{G}^\infty$. The unicity of C^∞ is ensured by the fact that the source cliques of $\mathtt{G}^\infty$ are by definition disjoint.

2. C_1^+ and C_2^+ intersect on property $\mathtt{p}$ iff there exist some $\mathtt{p}_1 \in C_1$ and $\mathtt{p}_2 \in C_2$ which are specializations of the same $\mathtt{p}$ (one, but not both, may also be $\mathtt{p}$ itself). Independently, we know that there exist $r_1, r_2 \in \mathtt{G}$ such that r_1 has $\mathtt{p}_1$ and r_2 has $\mathtt{p}_2$; in $\mathtt{G}^\infty$, r_1 has $\mathtt{p}_1$ and $\mathtt{p}$, thus these two properties are in the same $\mathtt{G}^\infty$ clique. Similarly, r_2 has $\mathtt{p}_2$ and $\mathtt{p}$, which ensures that $\mathtt{p}$ is also in the same $\mathtt{G}^\infty$ source clique.

3. Let $\{\mathtt{p}_1, \ldots, \mathtt{p}_k\}$ be the data properties that appear both in $\mathtt{G}$ and in C^∞; it follows from the saturation rules and the definition of cliques, that $k > 0$. For $1 \leq i \leq k$, let C_i be the $\mathtt{G}$ source clique comprising $\mathtt{p}_i$. Applying lemma point 1., $C_i \subseteq C^\infty$ for each $1 \leq i \leq k$. Further, it is easy to see that $C_i^+ \subseteq C^\infty$, since any property that saturation adds to C_i^+ is also added by saturation to C^∞. Thus, $\bigcup_{1 \leq i \leq k} C_i^+ \subseteq C^\infty$.

   Let us now show that $C^\infty \subseteq \bigcup_{1 \leq i \leq k} C_i^+$. Let $\mathtt{p} \in C^\infty$ be a data property, then there exists a resource r having $\mathtt{p}$ in $\mathtt{G}^\infty$. Then, in $\mathtt{G}$, r has a property $\mathtt{p}'$ which is either $\mathtt{p}$, or is such that $\mathtt{p}' \prec_{sp} \mathtt{p}$ in $\mathtt{G}^\infty$. Then, in $\mathtt{G}^\infty$, r has both $\mathtt{p}$ and $\mathtt{p}'$, which entails that $\mathtt{p}' \in C^\infty$. Therefore, $\mathtt{p}'$ is a data property occurring both in C^∞ and in $\mathtt{G}$, therefore $\mathtt{p}'$ is one of the properties $\mathtt{p}_i$, for some $1 \leq i \leq k$, that is, $\mathtt{p}' \in C_i$, and accordingly, $\mathtt{p} \in C_i^+$ due to $\mathtt{p}' \prec_{sp} \mathtt{p}$.

   Thus, any data property $\mathtt{p} \in C^\infty$ is part of some C_i^+.

   We must still show that the saturated cliques intersect. If $k = 1$ the statement is trivially true. Suppose $k \geq 2$ and the statement is false. Let $\mathcal{C}$ denote the set $\{C_1, \ldots, C_m\}$; the cliques in $\mathcal{C}$ are pairwise disjoint by definition. Let $\mathcal{I} \subseteq \mathcal{C}$ be a *maximal* subset of $\mathcal{C}$ cliques such that the saturations of $\mathcal{I}$ cliques all intersect (directly or indirectly). Let $\mathcal{J} = \mathcal{C} \setminus \mathcal{I}$ be the complement of $\mathcal{I}$; if the last part of 4. is false, $\mathcal{J}$ is not empty. We denote $\mathcal{I}^+$, respectively $\mathcal{J}^+$, the set of the saturated cliques from $\mathcal{I}$, resp. $\mathcal{J}$.

   No data property $\mathtt{p}_i$ from $\mathcal{I}^+$ can be source-related in $\mathtt{G}^\infty$ to any data property $\mathtt{p}_j$ from $\mathcal{J}^+$. This is because source-relatedness requires a resource r having in $\mathtt{G}^\infty$ both $\mathtt{p}_i$ and a property $\mathtt{p}$ source-related to $\mathtt{p}_j$. If such a property $\mathtt{p}$ existed, it would belong both to $\mathcal{I}^+$ (since $\mathtt{p}$ has a common source with $\mathtt{p}_i$) and to $\mathcal{J}^+$ (since $\mathtt{p}$ is source-related to $\mathtt{p}_j$); or, $\mathcal{I}^+$ and $\mathcal{J}^+$ have no property in common.

   The lack of source-relatedness in $\mathtt{G}^\infty$ between $\mathtt{p}_i$ and $\mathtt{p}_j$ chosen as above contradicts the hypothesis that they are part of the same source clique of $\mathtt{G}^\infty$, namely C^∞.

4. The statement follows quite directly as a consequence of the previous one, concluding our proof.

□

The lemma is crucial as it states the relationship between the cliques (thus, the summaries) of G and $\mathtt{G}^\infty$; our goal is to build the summary of the latter *without* having access to it, but just access to G. Figure 4 illustrates the lemma. The clique C^∞ of $\mathtt{G}^\infty$ encloses the cliques C_1, C_2, C_3, C_4 of G (Lemma item 1). C_2^+ intersects all of C_1^+, C_3^+ and C_4^+, thus they are all in the same clique C^∞ (item 2), which is the union of C_1^+, C_2^+, C_3^+ and C_4^+ (item 3). A chain of intersecting cliques connects e.g., C_1^+ through C_2^+ to C_4^+. Item 4 is not illustrated, as individual properties do not show; it follows quite directly from item 3.

3.2 Equivalence relations among graph nodes

We are now ready to introduce two crucial notions of RDF node equivalence based on property cliques:

Definition 8. (STRONG AND WEAK EQUIVALENCE) *Strong (denoted $\equiv_{\mathtt{S}}$) and weak (denoted $\equiv_{\mathtt{W}}$) equivalence are RDF node equivalence relations. Two data nodes of* G *are* strongly equivalent*, denoted $n_1 \equiv_{\mathtt{S}} n_2$, iff they have the same source* and *target cliques.*

Two data nodes are weakly equivalent*, denoted $n_1 \equiv_{\mathtt{W}} n_2$, 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*.*

Observe that strong equivalence implies weak equivalence.

In Figure 3, the resources r_1, r_2, r_3, r_5 are *strongly* equivalent to each other, as well as t_1, t_2, t_3, t_4. Moreover, $r_1, ..., r_5$ are *weakly* equivalent to each other due to their common source clique SC_1, as well as t_1, t_2, t_3, t_4 due to their common target clique; the same holds for a_1 and a_2, and separately for e_1 and e_2. In general, weakly equivalent resources can be connected as exemplified in Figure 5, by an *alternating sequence of source and target cliques*.

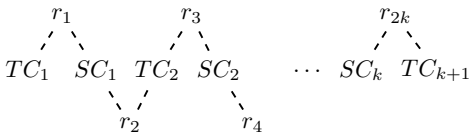


Figure 5: Weakly equivalent resources of G and their cliques.

RDF bisimulation-based node equivalence The classic concept of graph bisimulation has been intensively used to define graph summaries, from [25] to [7] and many others. To clarify the connection with our work, we revisit bisimulation as an RDF equivalence relation, as follows:

Definition 9. (FW RDF BISIMILARITY) *Forward (FW) RDF bisimilarity is an RDF node equivalence relation such that two data nodes $n_1, n_2 \in$ G are FW-bisimilar, denoted $n_1 \equiv_{\mathtt{fw}} n_2$, iff (i) for every* G *data triple of the form n_1 p m_1 there exists a* G *data triple of the form n_2 p m_2 such that $m_1 \equiv_{\mathtt{fw}} m_2$, and conversely (ii) for every* G *data triple of the form n_2 p m_2 there exists a* G *data triple of the form n_1 p m_1 such that $m_1 \equiv_{\mathtt{fw}} m_2$.*

Similarly, a backward (BW) RDF bisimilarity, noted $\equiv_{\mathtt{bw}}$, is defined by inverting the subject and object positions of triples in the definition above. Further, a forward and backward (FB) RDF bisimilarity between data nodes, noted $\equiv_{\mathtt{fb}}$ is defined by nodes that are both FW and BW similar.

In Figure 3, nodes t_1, t_2, t_4 are BW-bisimilar, as they are only targets of the title property from BW-similar sources r_1, r_2, r_5, since they have no incoming edges. However, no two resources among r_1 to r_5 are FW-bisimilar (thus, they are not FB-bisimilar either), as they all have different sets of outgoing properties. This example illustrates that *bisimilarity is rare in heterogeneous graphs*, and accordingly, *bisimulation summaries are almost as large at the data* e.g., 70% to 99% reported in [19]. Thus,

bisimulation summaries do not satisfy our compactness requirement. Several works considered *bounded* bisimilarity, i.e., accept nodes as bisimilar if they surrounding subgraphs *of diameter at most k edges* are bisimilar; small values of k reduce summary size, but then (i) structural and semantic properties are only guaranteed on a distance of at most k, (ii) it is hard (and lengthy) for users to guess k through repeated summarization.

The two novel (weak and strong) and three recycled (FW, BW and FB) equivalences mentioned above ignore possible type triples. The last very simple RDF node equivalence we introduce is:

Definition 10. (TYPE-BASED EQUIVALENCE) *Type-based equivalence (denoted $\equiv_{\mathtt{T}}$) between graph nodes is an RDF equivalence defined as follows. Two data nodes n_1 and n_2 of $\mathtt{G}$ are type-equivalent, denoted $n_1 \equiv_{\mathtt{T}} n_2$, iff they have exactly the same set of types, which is non-empty.*

Recall that some or even all nodes in an RDF graph may lack types; all such nodes are equivalent from the viewpoint of $\equiv_{\mathtt{T}}$.

Clearly, each equivalence relation defines a partition over the data nodes in $\mathtt{G}$.

3.3 RDF summaries

With the RDF equivalence notions in place, we define RDF summaries directly as graph quotients:

Definition 11. (RDF SUMMARY) *Given an RDF graph $\mathtt{G}$ and an RDF relation $\equiv$ among the nodes of $\mathtt{G}$, the summary of $\mathtt{G}$ by $\equiv$, denoted $\mathtt{G}_{/\equiv}$, is the quotient of $\mathtt{G}$ by the RDF node equivalence relation $\equiv$.*

Thus, any RDF node equivalence (in particular, any of the six: $\equiv_{\mathtt{W}}$, $\equiv_{\mathtt{S}}$, $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$, $\equiv_{\mathtt{fb}}$ and $\equiv_{\mathtt{T}}$ introduced in the previous section) leads to a (possibly distinct) quotient summary. It immediately follows from the definition that *the size of a summary is bounded by that of its summarized graph*. In practice, as we will show, our summaries are much smaller than the graph.

Representation function f: By the summary definition, to every node in $\mathtt{G}$ corresponds exactly one node in the summary $\mathtt{G}_{\equiv}$. We call *representation function* and denote $f_{\equiv}$ (or simply f, when this does not cause confusion) the function associating a summary node to each $\mathtt{G}$ node; we say $f(n)$ *represents* n in the summary.

Importantly, any RDF summary enjoys the following property:

Property 1. (SCHEMA PRESERVATION) *An RDF graph $\mathtt{G}$ and an RDF summary $\mathtt{G}_{/\equiv}$ of it have the same schema triples, i.e., $\mathtt{S}_{\mathtt{G}} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ holds.*

An RDF summary preserves the schema of the summarized graph because (i) class and property nodes, *RDF-equivalent only to themselves*, are preserved through a quotient that uses an RDF node equivalence relation, and (ii) schema triples only contain class or property nodes as subjects/objects. As we shall see, preserving the schema is necessary (but not sufficient) to achieve summary completeness (Section 2.2). Further, since the summary is an RDF graph, the subjects of type and data triples in $\mathtt{G}_{/\equiv}$ must be URIs.

Further, we find that:

Proposition 1. (SUMMARY REPRESENTATIVENESS) *An RDF summary $\mathtt{G}_{/\equiv}$ is RBGP*-representative.*

Proof. We prove the statement for **RBGP** queries; Proposition 2 (below) carries the statement over to RBGP*.

Let q be a query such that $q(\mathtt{G}^{\infty}) \neq \emptyset$; we need to show that $q((\mathtt{G}_{/\equiv})^{\infty}) \neq \emptyset$.

Let $\phi : q \to \mathtt{G}^{\infty}$ be an embedding, assigning to each query variable v, a node from $\mathtt{G}^{\infty}$; we extend ϕ to say it maps triple patterns from q into triples from $\mathtt{G}^{\infty}$. We need to produce an embedding from q into $(\mathtt{G}_{/\equiv})^{\infty}$.

First, consider a triple pattern t of q whose embedding $\phi(t) \in \mathtt{G}^{\infty}$ also belongs to $\mathtt{G}$.

- If $\phi(t)$ is a schema triple, then $\phi(t)$ is also in $\mathtt{G}_{/\equiv}$, since $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema triples.
- Else if $\phi(t)$ is a type triple of the form $\mathtt{s}\ \tau\ \mathtt{c}$, the triple $f(\mathtt{s})\ \tau\ \mathtt{c}$ belongs to $\mathtt{G}_{/\equiv}$, thus also to $(\mathtt{G}_{/\equiv})^\infty$.
- Otherwise, $\phi(t) = \mathtt{s}\ \mathtt{p}\ \mathtt{o} \in \mathtt{G}$ is a data triple, $\mathtt{G}_{/\equiv}$ holds the triple $f(\mathtt{s})\ \mathtt{p}\ f(\mathtt{o})$, thus $(\mathtt{G}_{/\equiv})^\infty$ also comprises it.

Now consider a triple pattern t' of q whose embedding $\phi(t') \in \mathtt{G}^\infty$ does not belong to $\mathtt{G}$.

- If $\phi(t')$ is a *schema* triple, then by definition of RDF entailment $\phi(t') \in (\mathtt{S}_\mathtt{G})^\infty$, thus it is also in $(\mathtt{G}_{/\equiv})^\infty$ since $\mathtt{S}_\mathtt{G} = \mathtt{S}_{\mathtt{G}_{/\equiv}} \subseteq \mathtt{G}_{/\equiv}$ by definition of a summary.
- Else if $\phi(t')$ is a *data* triple in $\mathtt{G}^\infty$, then by definition of RDF entailment, this triple $\phi(t')$ must be entailed by a *data* triple $t_d = \mathtt{s}_d\ \mathtt{p}_d\ \mathtt{o}_d$ in $\mathtt{G}$ and a subproperty constraints t_s, i.e., a schema triple in $\mathtt{S}^\infty$. As explained above, $f(\mathtt{s}_d)\ \mathtt{p}_d\ f(\mathtt{o}_d) \in \mathtt{G}_{/\equiv}$; at the same time, t_s also belongs $(\mathtt{G}_{/\equiv})^\infty$, since $\mathtt{S}_\mathtt{G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S}_\mathtt{G})^\infty = (\mathtt{S}_{\mathtt{G}_{\equiv}})^\infty$. It follows that the inference step which entailed $\phi(t')$ from t_d and t_s in $\mathtt{G}^\infty$ also applies on $f(\mathtt{s})\ \mathtt{p}_d\ f(\mathtt{o}_d)$ and t_s in $(\mathtt{G}_{/\equiv})^\infty$.
- Otherwise, $\phi(t')$ is a τ triple in $\mathtt{G}^\infty$. This may result either:
  - from a $\mathtt{D}_\mathtt{G}$ triple $t_d = \mathtt{s}_d\ \mathtt{p}_d\ \mathtt{o}_d$ and a triple $t_s \in (\mathtt{S}_\mathtt{G})^\infty$, if t_s is a $\hookleftarrow_d$ or $\hookrightarrow_r$ triple. This case is very similar to the one above.
  - from a $\mathtt{T}_\mathtt{G}$ triple of the form $\mathtt{s}\ \tau\ \mathtt{c}_1$ and a schema triple $t_s = \mathtt{c}_1 \prec_{sc} \mathtt{c}_2 \in (\mathtt{S}_\mathtt{G})^\infty$, such that $\phi(t') = \mathtt{s}\ \tau\ \mathtt{c}_2$. In this case, $\mathtt{G}_{/\equiv}$ holds the triple $f(\mathtt{s})\ \tau\ \mathtt{c}_1$ which is also present in $(\mathtt{G}_{/\equiv})^\infty$, thus the same inference step applies in $(\mathtt{G}_{/\equiv})^\infty$ to produce $f(\mathtt{s})\ \tau\ \mathtt{c}_2$, since $\mathtt{S}_\mathtt{G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S}_\mathtt{G})^\infty = (\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$.

Thus, any q triple mapped by ϕ into a data $\mathtt{G}^\infty$ triple (which may or may not explicitly belong to $\mathtt{G}$) is also mapped into a corresponding triple in $(\mathtt{G}_{/\equiv})^\infty$.

To conclude this proof, we now need to show that, in addition to the fact that each q triple that has an embedding in $\mathtt{G}^\infty$ has also necessarily an embedding in $\mathtt{G}^\infty_{/\equiv}$, if q has an embedding in $\mathtt{G}^\infty$, then q has also an embedding in $(\mathtt{G}_{/\equiv})^\infty$. This amounts to show that any two q triples t_1 and t_2 that join and that have an embedding in $\mathtt{G}^\infty$ also embed in $(\mathtt{G}_{/\equiv})^\infty$ (i.e., q joins are preserved).

- If both $\phi(t_1)$ and $\phi(t_2)$ are schema triples in $\mathtt{G}^\infty$, then these two triples are also in $(\mathtt{G}_{/\equiv})^\infty$, since $\mathtt{S}_\mathtt{G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S}_\mathtt{G})^\infty = \mathtt{S}^\infty_{\mathtt{G}_{/\equiv}}$.
- Else if both $\phi(t_1)$ and $\phi(t_2)$ are non-schema triples in $\mathtt{G}^\infty$:
  - If both $\phi(t_1)$ and $\phi(t_2)$ are data triples in $\mathtt{G}^\infty$, there exists a triple t'_1) (respt'_2) in $\mathtt{G}$, with same subject/object, from which $\phi(t_1)$ (resp. $\phi(t_2)$) is entailed using a sub-property constraint t^1_s (resp. t^2_s) from $\mathtt{S}^\infty$. Since $\phi(t_1)$ and t'_1 (resp. $\phi(t_2)$ and t'_2) have the same subject and object values, then t'_1 and t'_2 have same values on the places where t_1 and t_2 join. Therefore, if we assume that $t'_1 = \mathtt{s}_1\ \mathtt{p}_1\ \mathtt{o}_1$ and $t'_2 = \mathtt{s}_2\ \mathtt{p}_2\ \mathtt{o}_2$, the $\mathtt{G}_{/\equiv}$ triples $f(\mathtt{s}_1)\ \mathtt{p}_1\ f(\mathtt{o}_1)$ and $f(\mathtt{s}_2)\ \mathtt{p}_2\ f(\mathtt{o}_2)$ necessarily have same values on the places where t_1 and t_2 join. Moreover, since $\mathtt{S}_\mathtt{G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $(\mathtt{S}_\mathtt{G})^\infty = (\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$, these two $\mathtt{G}_{/\equiv}$ triples and the above-mentioned t^1_s and t^2_s schema triples, produce the counterpart triples of $\phi(t_1)$ and $\phi(t_2)$ in $(\mathtt{G}_{/\equiv})^\infty$, which have same subject and object values. Thus, the q triples t_1 and t_2 embed in these two $(\mathtt{G}_{/\equiv})^\infty$ triples, if they embed in $\phi(t_1)$ and $\phi(t_2)$ in $\mathtt{G}^\infty$.

- Else if both $\phi(t_1)$ and $\phi(t_2)$ are type triples in $\mathtt{G}^\infty$, say $\phi(t_1) = u\ \tau\ c_1$ and $\phi(t_2) = u\ \tau\ c_2$, then $t_1 = x\ \tau\ c_1$ and $t_2 = x\ \tau\ c_2$ by definition of an RBGP query. As in the cases of single q triple embeddings, $\phi(t_1)$ (resp. $\phi(t_2)$) results either from a $\mathtt{G}$ triple $u\ \tau\ c$ and a $\mathtt{S}_\mathtt{G}^\infty$ triple $c\ \prec_{sc}\ c_1$, or a $\mathtt{G}$ triple $u\ p\ u_1$ and a $\mathtt{S}_\mathtt{G}^\infty$ triple $p\ \hookleftarrow_d\ c_1$, or a $\mathtt{G}$ triple $u_1\ p\ u$ and a $(\mathtt{S}_\mathtt{G})^\infty$ triple $p\ \hookrightarrow_r\ c_1$. Therefore, since $\mathtt{S}_\mathtt{G} = \mathtt{S}_{\mathtt{G}_{/\equiv}}$ hence $\mathtt{S}_\mathtt{G}^\infty = (\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$, for $\phi(t_1)$ (resp. $\phi(t_2)$), there are either a $\mathtt{G}_{/\equiv}$ triple $f(u)\ \tau\ c$ and a $(\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$ triple $c\ \prec_{sc}\ c_1$, or a $\mathtt{G}_{/\equiv}$ triple $f(u)\ p\ f(u_1)$ and a $(\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$ triple $p\ \hookleftarrow_d\ c_1$, or a $\mathtt{G}_{/\equiv}$ triple $f(u_1)\ p\ f(u)$ and a $(\mathtt{S}_{\mathtt{G}_{/\equiv}})^\infty$ triple $p\ \hookrightarrow_r\ c_1$, which entail $f(u)\ \tau\ c_1$ and $f(u)\ \tau\ c_2$ in $\mathtt{G}_{/\equiv}^\infty$. Thus, the q triples t_1 and t_2 embed in these two $\mathtt{G}_{/\equiv}^\infty$ triples, if they embed in $\phi(t_1)$ and $\phi(t_2)$ in $\mathtt{G}^\infty$.
- Otherwise, $\phi(t_1)$ is a data triple in $\mathtt{G}^\infty$ and $\phi(t_2)$ is a type triple in $\mathtt{G}^\infty$. This case is very similar to the two above case, hence we do not detail it.

• Otherwise, $\phi(t_1)$ is a schema triple in $\mathtt{G}^\infty$ and $\phi(t_2)$ is not a schema triples in $\mathtt{G}^\infty$. In this case, since q is an RBGP query, q triples must be such that t_1 is $\mathtt{s}_1\ \mathtt{p}_1\ \mathtt{o}_1$ with $\mathtt{p}_1 \in \{\prec_{sc}, \prec_{sp}, \hookleftarrow_d, \hookrightarrow_r\}$ and t_2 is either $\mathtt{s}_2\ p\ \mathtt{o}_2$ or $\mathtt{s}_2\ \tau\ c$.

Since $\mathtt{G}^\infty$ and $\mathtt{G}_{/\equiv}^\infty$ have the same schema, $\phi(t_1)$ also belongs to $\mathtt{G}_{/\equiv}^\infty$. Now:

- If t_2 is $\mathtt{s}_2\ p\ \mathtt{o}_2$ then $\phi(t_2) = s, p, o$ must be either in $\mathtt{G}$ or entailed from a $\mathtt{G}$ data triple s, p', o and a $\mathtt{S}_\mathtt{G}^\infty$ sub-property triple $p', \prec_{sp}, p$ (see above, for single q triple embedding). If $\phi(t_2)$ is in $\mathtt{G}$, then $f(s)\ p\ f(o)$ is in $\mathtt{G}_{/\equiv}$, hence in $\mathtt{G}_{/\equiv}^\infty$. Otherwise, $f(s)\ p'\ f(o)$ is in $\mathtt{G}_{/\equiv}$, $p'\ \prec_{sp}\ p$ is in $\mathtt{S}_{\mathtt{G}_{/\equiv}}$ (since $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema), thus $f(s)\ p\ f(o)$ is in $\mathtt{G}_{/\equiv}^\infty$. Since t_1 and t_2 joins, s and/or p are class/property nodes. If s (resp. o) is a class/property node, then $f(s) = s$ (resp. $f(o) = o$). Hence, $\phi(t_1) \in \mathtt{G}_{/\equiv}^\infty$ joins with $f(s)\ p\ f(o) \in \mathtt{G}_{/\equiv}^\infty$.
- If t_2 is $\mathtt{s}_2\ \tau\ c$ then $\phi(t_2) = s\ \tau\ c$ must be either in $\mathtt{G}$ or entailed from (i) a $\mathtt{G}$ data triple $s\ p\ o$ and a $\mathtt{S}_\mathtt{G}^\infty$ triple $p\ \hookleftarrow_d\ c$, or a $\mathtt{G}$ data triple $s_1\ p\ s$ and a $\mathtt{S}_\mathtt{G}^\infty$ triple $p\ \hookrightarrow_r\ c$, or (iii) a $\mathtt{G}$ type triple $s\ \tau\ c'$ and a $\mathtt{S}_\mathtt{G}^\infty$ triple $c'\ \prec_{sc}\ c$ (see above, for single q triple embedding). If $\phi(t_2)$ is in $\mathtt{G}$, then $f(s)\ \tau\ c$ is in $\mathtt{G}_{/\equiv}$, hence in $\mathtt{G}_{/\equiv}^\infty$. Otherwise, $f(s)\ p\ f(o)$ or $f(s_1)\ p\ f(s)$ or $f(s)\ \tau\ c'$ is in $\mathtt{G}_{/\equiv}$, and (since $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ have the same schema) thus $f(s)\ \tau\ c$ is in $\mathtt{G}_{/\equiv}^\infty$. Since t_1 and t_2 can only join on $\mathtt{s}_2$, s is a class or property nodes, hence $f(s) = s$. Therefore, $\phi(t_1) \in \mathtt{G}_{/\equiv}^\infty$ joins with $f(s)\ \tau\ c \in \mathtt{G}_{/\equiv}^\infty$.

□

Proposition 2. *RBGP representativeness entails RBGP* representativeness.*

Proof. Let q^* be an RBGP* query which is non-empty on $\mathtt{G}^\infty$ and $\mathtt{G}_{/\equiv}$ be an RBGP-representative summary of $\mathtt{G}$. We show that non-emptiness of q^* on $\mathtt{G}^\infty$ entails its non-emptiness on $(\mathtt{G}_{/\equiv})^\infty$.
Given that $q^*(\mathtt{G}^\infty)$ is non-empty, there exists at least an embedding of q^* into $\mathtt{G}^\infty$; let q be the query obtained by replacing in q^*, each variable occurring in the property position by the concrete property matching it in $\mathtt{G}^\infty$. Clearly, q has results on $\mathtt{G}^\infty$, and since $\mathtt{G}_{/\equiv}$ is RBGP representative, q also has results on $(\mathtt{G}_{/\equiv})^\infty$. Therefore, $q(\mathtt{G}_{/\equiv}^\infty) \neq \emptyset$, and given that $q \subseteq q^*$ (query containment), it follows that $q^*((\mathtt{G}_{/\equiv})^\infty) \neq \emptyset$. □

One can prove in a very similar fashion that summaries are also representative with respect to *regular path expression queries* such as the property paths introduced in SPARQL 1.1: any regular path expression query whose results are non empty on an RDF graph $\mathtt{G}$ also has non-empty results on a summary of $\mathtt{G}$. As a consequence, our summaries also preserve reachability in $\mathtt{G}$: if a path leads from n_1 to n_2 in $\mathtt{G}$, the same path also leads from $f(n_1)$, representing n_1 in the summary, to $f(n_2)$.

We are interested in summaries having the following property:

Definition 12. (FIXPOINT PROPERTY) *A summary* $\mathtt{G}_{/\equiv}$ *has the* fixpoint property *iff for any graph* $\mathtt{G}$, $(\mathtt{G}_{/\equiv})_{/\equiv} = \mathtt{G}_{/\equiv}$ *holds.*

Intuitively, the fixpoint property expresses the fact that a summary cannot be summarized further, i.e., $\mathtt{G}_{/\equiv}$ is its own summary. This is desirable as we wish our summaries to be as compact as possible. It turns out that any summary has this property (which easily follows from the fact that the summary is a quotient).

Proposition 3. (SUMMARY FIXPOINT) *An RDF summary based on the* $\equiv_{\mathtt{S}}$, $\equiv_{\mathtt{W}}$, $\equiv_{\mathtt{T}}$, $\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$ *or* $\equiv_{\mathtt{fb}}$ *RDF node equivalence relation has the fixpoint property.*

Proof. We start by some remarks which do not depend on the specific equivalence relation used. Suppose that a summary does not have the fixpoint property, i.e., $\mathtt{G}_{/\equiv} \neq (\mathtt{G}_{/\equiv})_{/\equiv}$. By the definition of a quotient graph, if $(\mathtt{G}_{/\equiv})_{/\equiv}$ and $\mathtt{G}_{/\equiv}$ differ, then $(\mathtt{G}_{/\equiv})_{/\equiv}$ must have *less nodes* than $\mathtt{G}_{/\equiv}$: on one hand, a quotient summary always has at most as many nodes as the summarized graph; on the other hand, if $\mathtt{G}$ and $\mathtt{G}_{/\equiv}$ had the same number of nodes, they would also have the same number of edges, since the summary edges are completely determined by the summary nodes and the edges of the original graph.

We now split the discussion for our three RDF equivalence relations and the three ones based on bisimulation:

Case $\equiv_{\mathtt{S}}$: if two $\mathtt{G}_{/\equiv}$ nodes n_1, n_2 are $\equiv_{\mathtt{S}}$-equivalent, then both nodes have the same source and target cliques. In this case, n_1 (resp. n_2) represents $\mathtt{G}$ nodes, each of which has the same source and target cliques as n_1 *and* n_2. Therefore, by definition of a graph quotient, all $\mathtt{G}$ nodes represented by n_1 and n_2 would end up in a single $\mathtt{G}_{/\equiv}$ node, a contradiction.

Case $\equiv_{\mathtt{W}}$: if $(\mathtt{G}_{/\equiv})_{/\equiv}$ has less nodes than $\mathtt{G}_{/\equiv}$, then there exist two $\mathtt{G}_{/\equiv}$ nodes n_1, n_2 that have the same non-empty source or target cliques. In this case, there exists some property common to these same cliques, hence all the $\mathtt{G}$ nodes represented by n_1 *and* n_2 would end up in a single $\mathtt{G}_{/\equiv}$ node by the definition of a graph quotient using $\equiv_{\mathtt{W}}$, a contradiction.

Case $\equiv_{\mathtt{T}}$: if two $\mathtt{G}_{/\equiv}$ nodes n_1, n_2 are $\equiv_{\mathtt{T}}$-equivalent, then both nodes have the same non-empty type sets. In this case, n_1 (resp. n_2) represents $\mathtt{G}$ nodes, every of which has the same non-empty type sets as n_1 *and* n_2. Therefore, by the definition of a graph quotient, all $\mathtt{G}$ nodes represented by n_1 and n_2 end up in a single $\mathtt{G}_{/\equiv}$ node, a contradiction.

Case $\equiv_{\mathtt{fw}}$: if two $\mathtt{G}_{/\equiv}$ nodes n_1, n_2 are $\equiv_{\mathtt{fw}}$-equivalent, then for every $\mathtt{G}_{/\equiv}$ node m_1 such that $n_1\ \mathtt{p}\ m_1 \in \mathtt{G}_{/\equiv}$ there exists a $\mathtt{G}_{/\equiv}$ node m_2 such that $n_2\ \mathtt{p}\ m_2 \in \mathtt{G}_{/\equiv}$ and $m_1 \equiv_{\mathtt{fw}} m_2$ and similarly for every $\mathtt{G}_{/\equiv}$ node m_2 such that $n_2\ \mathtt{p}\ m_2 \in \mathtt{G}_{/\equiv}$ there exists a $\mathtt{G}_{/\equiv}$ node m_1 such that $n_1\ \mathtt{p}\ m_1 \in \mathtt{G}_{/\equiv}$ and $m_1 \equiv_{\mathtt{fw}} m_2$. By definition of $\equiv_{\mathtt{fw}}$, $n_1\ \mathtt{p}\ m_1 \in \mathtt{G}_{/\equiv}$ iff $r_i^1\ \mathtt{p}\ r_i^{1\prime} \in \mathtt{G}$ with n_1 representing r_i^1 and m_1 representing $r_i^{1\prime}$, for $1 \leq i \leq k$. Similarly, $n_2\ \mathtt{p}\ m_2 \in \mathtt{G}_{/\equiv}$ iff $r_j^2\ \mathtt{p}\ r_j^{2\prime} \in \mathtt{G}$ with n_2 representing r_j^2 and m_2 representing $r_j^{2\prime}$, for $1 \leq j \leq l$. It therefore follows that the $\mathtt{G}$ nodes r_i^1 and r_j^2 are $\equiv_{\mathtt{fw}}$-equivalent, for $1 \leq i \leq k$ and $1 \leq j \leq l$, hence are represented by a single $\mathtt{G}_{/\equiv}$ node n (instead of two nodes n_1 and n_2), a contradiction.

Case $\equiv_{\mathtt{bw}}$: the proof derives directly from the above one by considering $\mathtt{G}_{/\equiv}$ triples in which n_1, n_2 are objects (instead of subjects).

Case $\equiv_{\mathtt{fb}}$: the proof derives directly from the two above ones by considering that n_1, n_2 are subjects of $\mathtt{G}_{/\equiv}$ triples and also objects of some other $\mathtt{G}_{/\equiv}$ triples. □

This has an immediate interesting consequence, considering that an RDF summary belongs to its own inverse set:

Proposition 4. (ACCURACY) *Any RDF summary with the fixpoint property is accurate.*

Proof. The proof follows from the fact that any summary has a fixpoint property: a summary $\mathtt{G}_{/\equiv}$ which is its own summary, corresponds at least to the RDF graph $\mathtt{G}_{/\equiv}$ itself. □

An important structural property relates $\mathtt{G}$, $\mathtt{G}^\infty$ and the representation function f:

Proposition 5. (SUMMARIZATION HOMOMORPHISM) *Let $\mathtt{G}$ be an RDF graph, $\mathtt{G}_{/\equiv}$ its summary and f the corresponding representation function from $\mathtt{G}$ nodes to $\mathtt{G}_{/\equiv}$ nodes. f defines an homomorphism from $\mathtt{G}^\infty$ to $(\mathtt{G}_{/\equiv})^\infty$.*

Proof. We first show that an homomorphism can be established from the node sets of $\mathtt{G}^\infty$ to that of $(\mathtt{G}_{/\equiv})^\infty$.

Recall from Section 2.1 that RDF saturation with RDFS constraints only adds edges between graph nodes, but does not add nodes. Thus, a node n is in $\mathtt{G}^\infty$ iff n is in $\mathtt{G}$. Further, by the definition of our quotient-based summaries (Definition 11), n is in $\mathtt{G}$ iff $f(n)$ is in $\mathtt{G}_{/\equiv}$. Finally, again by the definition of saturation, $f(n)$ is in $\mathtt{G}_{/\equiv}$ iff $f(n)$ is in $(\mathtt{G}_{/\equiv})^\infty$.

Therefore, every $\mathtt{G}^\infty$ node n maps the $f(n)$ $(\mathtt{G}_{/\equiv})^\infty$ node (*).

Next, we show that there is a one-to-one mapping between $\mathtt{G}^\infty$ edges and those of $(\mathtt{G}_{/\equiv})^\infty$.

If $n_1\ p\ n_2$ is an edge in $\mathtt{G}^\infty$, at least one of the following two situations holds:

- $n_1\ p\ n_2$ is an edge in $\mathtt{G}$. This holds iff $f(n_1)\ p\ f(n_2)$ is an edge in $\mathtt{G}_{/\equiv}$, by definition of an RDF summary. Finally, if $f(n_1)\ p\ f(n_2)$ is an edge in $\mathtt{G}_{/\equiv}$, then $f(n_1)\ p\ f(n_2)$ is also an edge in $(\mathtt{G}_{/\equiv})^\infty$.
- $n_1\ p'\ n_2$ is an edge in $\mathtt{G}$, and $p' \prec_{sp} p$ is in $\mathtt{S}_\mathtt{G}^\infty$, thus $n_1\ p\ n_2$ is produced by saturation in $\mathtt{G}^\infty$. In this case, we show similarly to the preceding item that $f(n_1)\ p'\ f(n_2)$ is an edge in $(\mathtt{G}_{/\equiv})^\infty$, hence $f(n_1)\ p\ f(n_2)$ is also an edge added to $(\mathtt{G}_{/\equiv})^\infty$ by saturation, since $(\mathtt{G}_{/\equiv})^\infty$ and $\mathtt{G}^\infty$ have the same (saturated) schema triples (Property 1).

If $n_1\ \tau\ c$ is an edge in $\mathtt{G}^\infty$, at least one of the following two situations holds:

- $n_1\ \tau\ c$ is an edge in $\mathtt{G}$. This holds iff $f(n_1)\ \tau\ c$ is an edge in $\mathtt{G}_{/\equiv}$, by definition of an RDF summary (recall that $f(c) = c$ for classes). Finally, if $f(n_1)\ \tau\ c$ is an edge in $\mathtt{G}_{/\equiv}$, then $f(n_1)\ \tau\ c$ is also an edge in $(\mathtt{G}_{/\equiv})^\infty$.
- $n_1\ p\ n_2$ is an edge in $\mathtt{G}$ and $p \hookleftarrow_d c$ (or $p \hookrightarrow_r c$) is in $\mathtt{S}_\mathtt{G}^\infty$, thus $n_1\ \tau\ c$ is produced by saturation in $\mathtt{G}^\infty$. In this case, we show similarly as above that $f(n_1)\ p\ f(n_2)$ is an edge in $(\mathtt{G}_{/\equiv})^\infty$, hence $f(n_1)\ \tau\ c$ is also an edge added to $(\mathtt{G}_{/\equiv})^\infty$ by saturation, since $(\mathtt{G}_{/\equiv})^\infty$ and $\mathtt{G}^\infty$ have the same (saturated) schema triples (Property 1).

Therefore, every $\mathtt{G}^\infty$ edge $n_1\ p\ n_2$ (resp. $n_1\ \tau\ c$) maps into the $(\mathtt{G}_{/\equiv})^\infty$ edge $f(n_1)\ p\ f(n_2)$ (resp. $f(n_1)\ \tau\ c$) (**).

From (*) and (**), it follows that f is an homomorphism from $\mathtt{G}^\infty$ to $(\mathtt{G}_{/\equiv})^\infty$. □

We now consider the completeness requirement introduced in Section 2.2. We want the summary of $\mathtt{G}$ to reflect both the explicit and the implicit triples of $\mathtt{G}$, that is, we would like to compute $(\mathtt{G}^\infty)_{/\equiv}$ starting from $\mathtt{G}_{/\equiv}$ and without saturating $\mathtt{G}$. Recall from Section 2 that the semantics of an RDF graph is its saturation; thus, any graph having *the same saturation as* $(\mathtt{G}^\infty)_{/\equiv}$ can be seen as the complete summary of $\mathtt{G}$. Finally, observe that RDF summary nodes are just representatives of the original $\mathtt{G}$ nodes, thus the identity of summary nodes is not as important as the summary structure. Therefore, we are interested in computing from $\mathtt{G}$ and without saturating $\mathtt{G}$, an RDF graph *whose saturation is isomorphic to the saturation of* $(\mathtt{G}^\infty)_{/\equiv}$. We formalize this by:

Definition 13. (SUMMARY COMPLETENESS) *The RDF summary defined by a given RDF node equivalence relation $\equiv$ is complete iff $(\mathtt{G}^\infty)_{/\equiv} \sim ((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, where $\sim$ denotes isomorphism between the saturations of two RDF graphs.*

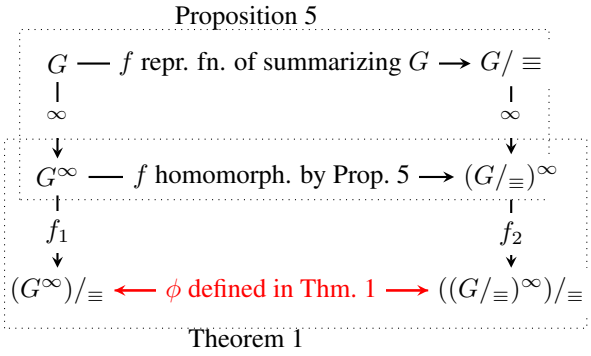


Figure 6: Illustration for Proposition 5 and Theorem 1.

In the above, we used $\sim$ to denote an isomorphism between two saturated RDF graphs, which moreover is the identity on class and property nodes (introduced in Section 2).

Definition 13 gives *the method we found* to compute a graph equivalent to (having the same saturation as) $(\mathtt{G}^\infty)_{/\equiv}$ as follows: summarize $\mathtt{G}$; saturate the result; then summarize it again. In general, we cannot rule out the existence of other methods to find $(\mathtt{G}^\infty)_{/\equiv}$ without saturating $\mathtt{G}$. It is, however, worth noting that computing $(\mathtt{G}_{/\equiv})^\infty$ (two processing steps starting from $\mathtt{G}$, instead of three) does *not* suffice, for the RDF equivalence notions (and thus, the summaries) that we introduce. This will be exemplified e.g., in Figure 9.

The first major result of our work is:

Theorem 1 (Condition for summary completeness)**.** *Let $\mathtt{G}$ be an RDF graph, $\mathtt{G}_{/\equiv}$ its summary and $f_\equiv$ the corresponding representation function from $\mathtt{G}$ nodes to $\mathtt{G}_{/\equiv}$ nodes.*

If the RDF node equivalence relation $\equiv$ satisfies: for every pair (n_1, n_2) of $\mathtt{G}$ nodes, $n_1 \equiv n_2$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv f(n_2)$ in $(\mathtt{G}_{/\equiv})^\infty$, then the summary of $\mathtt{G}$ through $\equiv$ is complete.

Proof. We start by introducing some **notations** (see Figure 6). Let f_1 be the representation function from $\mathtt{G}^\infty$ into $(\mathtt{G}^\infty)_{/\equiv}$, and f_2 be the representation function from $(\mathtt{G}_{/\equiv})^\infty$ into $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$.

Let the function φ be a function from the $(\mathtt{G}^\infty)_{/\equiv}$ nodes to the $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes defined as: $\varphi(f_1(n)) = f_2(f(n))$ for n any $\mathtt{G}^\infty$ node.

Suppose that for every pair (n_1, n_2) of $\mathtt{G}$ nodes, $n_1 \equiv n_2$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv f(n_2)$ in $(\mathtt{G}_{/\equiv})^\infty$ holds. Let us show that this condition suffices to ensure $(\mathtt{G}^\infty)_{/\equiv} \equiv ((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ holds, i.e., the φ function defines an isomorphism from $(\mathtt{G}^\infty)_{/\equiv}$ to $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$.

First, let us show that φ is a bijection from all the $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all the $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes. Since for every pair n_1, n_2 of $\mathtt{G}^\infty$ nodes, $n_1 \equiv n_2$ iff $f(n_1) \equiv f(n_2)$ in $(\mathtt{G}_{/\equiv})^\infty$, it follows that $(\mathtt{G}^\infty)_{/\equiv}$ and $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ have the same number of nodes (*).

Further, a given node n in $(\mathtt{G}^\infty)_{/\equiv}$ represents a set of equivalent nodes $n_1, \ldots, n_k$ from $\mathtt{G}^\infty$. By hypothesis, $n_1 \equiv \cdots \equiv n_k$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv \cdots \equiv f(n_k)$ in $\mathtt{G}^\infty_{/\equiv}$ holds. Hence, every node $n = f_1(n_1) = \cdots = f_1(n_k)$ of $(\mathtt{G}^\infty)_{/\equiv}$ maps to a distinct node $n' = f_2(f(n_1)) = \cdots = f_2(f(n_k))$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ (**).

Similarly, a given node n' in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ represents a set of equivalent nodes $n'_1 = f(n_1), \ldots, n'_k = f(n_k)$ in $(\mathtt{G}_{/\equiv})^\infty$. By hypothesis, $f(n_1) \equiv \cdots \equiv f(n_k)$ in $\mathtt{G}^\infty_{/\equiv}$ iff $n_1 \equiv \cdots \equiv n_k$ in $\mathtt{G}^\infty$ holds. Hence, every node $n' = f_2(f(n_1)) = \cdots = f_2(f(n_k))$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ maps to a distinct node $n = f_1(n_1) = \cdots = f_1(n_k)$ of $(\mathtt{G}^\infty)_{/\equiv}$ (***).

From (*), (**) and (***), it follows that φ is a bijective function from all the $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all the $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes.

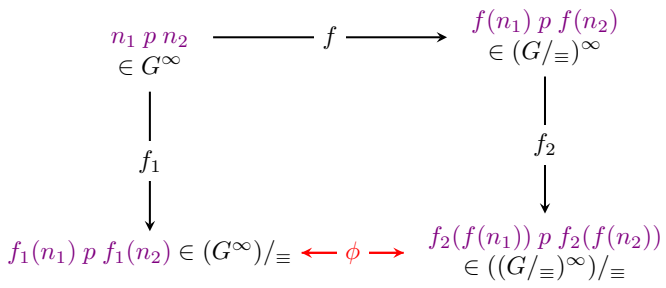


Figure 7: Diagram for the proof of Theorem 1.

Now, let us show that φ defines an isomorphism from $(\mathtt{G}^\infty)_{/\equiv}$ to $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$.

For every edge $n'_1\ p\ n'_2$ in $(\mathtt{G}^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge $n_1\ p\ n_2$ in $\mathtt{G}^\infty$ such that $n'_1\ p\ n'_2 = f_1(n_1)\ p\ f_1(n_2)$. Figure 7 illustrates the discussion. Further, if $n_1\ p\ n_2$ is in $\mathtt{G}^\infty$, then $f(n_1)\ p\ f(n_2)$ is in $(\mathtt{G}_{/\equiv})^\infty$ (Proposition 5), hence $f_2(f(n_1))\ p\ f_2(f(n_2))$ is in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$. Therefore,

- since for every $f_1(n_1)\ p\ f_1(n_2)$ edge in $(\mathtt{G}^\infty)_{/\equiv}$, there is an edge $f_2(f(n_1))\ p\ f_2(f(n_2))$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, and
- since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes,
- it follows that $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ contains the image of all $(\mathtt{G}^\infty)_{/\equiv}$ $f_1(n_1)\ p\ f_1(n_2)$ triples through φ (*).

Now, for every edge $n''_1\ p\ n''_2$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge $n'_1\ p\ n'_2$ in $(\mathtt{G}_{/\equiv})^\infty$ such that $n''_1\ p\ n''_2 = f_2(n'_1)\ p\ f_2(n'_2)$. Hence, by Proposition 5, there exists an edge $n_1\ p\ n_2$ in $\mathtt{G}^\infty$ such that $n'_1\ p\ n'_2 = f(n_1)\ p\ f(n_2)$. Moreover, since $n_1\ p\ n_2$ is in $\mathtt{G}^\infty$, $f_1(n_1)\ p\ f_1(n_2)$ is in $(\mathtt{G}^\infty)_{/\equiv}$. Therefore, since for every $f_2(f(n_1))\ p\ f_2(f(n_2))$ edge in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, there is an edge $f_1(n_1)\ p\ f_1(n_2)$ in $(\mathtt{G}^\infty)_{/\equiv}$, and since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes, $(\mathtt{G}^\infty)_{/\equiv}$ contains the image of all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ $n''_1\ p\ n''_2$ triples through φ^{-1} (**).

Similarly, for every edge $n'_1\ \tau\ c$ in $(\mathtt{G}^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge $n_1\ \tau\ c$ in $\mathtt{G}^\infty$ such that $n'_1\ \tau\ c = f_1(n_1)\ \tau\ c$. Further, if $n_1\ \tau\ c$ is in $\mathtt{G}^\infty$, then $f(n_1)\ \tau\ c$ is in $(\mathtt{G}_{/\equiv})^\infty$ (Proposition 5), hence $f_2(f(n_1))\ \tau\ c$ is in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$. Therefore,

- since for every $f_1(n_1)\ \tau\ c$ edge in $(\mathtt{G}^\infty)_{/\equiv}$, there is an edge $f_2(f(n_1))\ \tau\ c$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, and
- since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes,
- it follows that $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ contains the image of all $(\mathtt{G}^\infty)_{/\equiv}$ $f_1(n_1)\ \tau\ c$ triples through φ (*').

Now, for every edge $n''_1\ \tau\ c$ in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, by definition of an RDF summary, there exists an edge $n'_1\ \tau\ c$ in $(\mathtt{G}_{/\equiv})^\infty$ such that $n''_1\ \tau\ c = f_2(n'_1)\ \tau\ c$. Hence, by Proposition 5, there exists an edge $n_1\ \tau\ c$ in $\mathtt{G}^\infty$ such that $n'_1\ \tau\ c = f(n_1)\ \tau\ c$. Moreover, since $n_1\ \tau\ c$ is in $\mathtt{G}^\infty$, $f_1(n_1)\ \tau\ c$ is in $(\mathtt{G}^\infty)_{/\equiv}$. Therefore, since for every $f_2(f(n_1))\ \tau\ c$ edge in $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$, there is an edge $f_1(n_1)\ \tau\ c$ in $(\mathtt{G}^\infty)_{/\equiv}$,

and since $\varphi(f_1(n)) = f_2(f(n))$, for n any $\mathtt{G}^\infty$ node, is a bijective function from all $(\mathtt{G}^\infty)_{/\equiv}$ nodes to all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ nodes, $(\mathtt{G}^\infty)_{/\equiv}$ contains the image of all $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$ $n_1''\ \tau\ c$ triples through φ^{-1} (**').

From (*) and (**), and, (*') and (**'), it follows that φ defines an isomorphism from $(\mathtt{G}^\infty)_{/\equiv}$ to $((\mathtt{G}_{/\equiv})^\infty)_{/\equiv}$. □

The condition is sufficient, and we will use it to establish completeness for our summaries based on strong and weak equivalence; finding a necessary condition is currently open. For the summaries based on bisimulation, we readily establish based on Theorem 1:

Theorem 2. *$\equiv_{\mathtt{fw}}$, $\equiv_{\mathtt{bw}}$ and $\equiv_{\mathtt{fb}}$ summaries are complete.*

Proof. We first prove the claim for $\equiv_{\mathtt{fw}}$.

We show this result using the sufficient condition stated in Theorem 1. That is, $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$ holds iff $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$ holds.

This holds for class nodes and for property nodes since, by definition, they are only equivalent to themselves through some RDF node equivalence relation.

Now, consider two data nodes n_1, n_2 in $\mathtt{G}^\infty$ such that $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$, and let us show that $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$.

If $n_1 \equiv_{\mathtt{fw}} n_2$ holds in $\mathtt{G}^\infty$, then for every triple $n_1\ \mathtt{p}\ m_1$ there exists a triple $n_2\ \mathtt{p}\ m_2$ such that $m_1 \equiv_{\mathtt{fw}} m_2$ holds, and conversely for every triple $n_2\ \mathtt{p}\ m_2$ there exists a triple $n_1\ \mathtt{p}\ m_1$ such that $m_1 \equiv_{\mathtt{fw}} m_2$ holds.

Let $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$ be the set of outgoing properties from n_1 to m_1 and from n_2 to m_2 in $\mathtt{G}^\infty$.

In $\mathtt{G}$, the set of outgoing properties from n_1 to m_1, denoted $\mathcal{P}_{n_1 \to m_1}$ is a subset of $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$, since by definition the saturation of a graph only adds edges; similarly, in $\mathtt{G}$, the set of outgoing properties from n_2 to m_2, denoted $\mathcal{P}_{n_2 \to m_2}$ is a subset of $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$, which may be different from $\mathcal{P}_{n_1 \to m_1}$.

By definition of a $\equiv_{\mathtt{fw}}$-summary, the set of outgoing properties from $f(n_1)$ to $f(m_1)$ in $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$ is exactly $\mathcal{P}_{n_1 \to m_1}$ and similarly the set of outgoing properties from $f(n_2)$ to $f(m_2)$ in $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$ is exactly $\mathcal{P}_{n_2 \to m_2}$.

Since $\mathtt{G}$ and $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$ have the same schema (Property 1), it follows that in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$, the set of outgoing properties from $f(n_1)$ to $f(m_1)$, and from $f(n_2)$ to $f(m_2)$, is exactly $\mathcal{P}^\infty_{n_1,n_2 \to m_1,m_2}$ (data edges can only be added through $\prec_{sp}$ constraints).

Since the above holds for any pair of data nodes n_1, n_2 such that $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$, and for any of their $\mathtt{G}^\infty$ outgoing edges $n_1\ \mathtt{p}\ m_1$ and $n_2\ \mathtt{p}\ m_2$, hence $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$ holds.

Now, consider two data nodes $f(n_1), f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$ such that $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$ and let us show that $n_1 \equiv_{\mathtt{fw}} n_2$ holds in $\mathtt{G}^\infty$.

If $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ holds in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$, then for every triple $f(n_1)\ \mathtt{p}\ f(m_1)$ there exists a triple $f(n_2)\ \mathtt{p}\ f(m_2)$ such that $f(m_1) \equiv_{\mathtt{fw}} f(m_2)$ holds, and conversely for every triple $f(n_2)\ \mathtt{p}\ f(m_2)$ there exists a triple $f(n_1)\ \mathtt{p}\ f(m_1)$ such that $f(m_1) \equiv_{\mathtt{fw}} f(m_2)$ holds.

Let $\mathcal{P}^\infty_{f(n_1),f(n_2) \to f(m_1),f(m_2)}$ be the set of outgoing properties from $f(n_1)$ to $f(m_1)$ and from $f(n_2)$ to $f(m_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$.

In $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$, the set of outgoing properties from $f(n_1)$ to $f(m_1)$, denoted $\mathcal{P}_{f(n_1) \to f(m_1)}$ is a subset of $\mathcal{P}^\infty_{fn_1),f(n_2) \to f(m_1),f(m_2)}$, since by definition the saturation of a graph only adds edges; similarly, in $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$, the set of outgoing properties from $f(n_2)$ to $f(m_2)$, denoted $\mathcal{P}_{f(n_2) \to f(m_2)}$ is a subset of $\mathcal{P}^\infty_{f(n_1),f(n_2) \to f(m_1),f(m_2)}$, which may be different from $\mathcal{P}_{f(n_1) \to f(m_1)}$.

By definition of a $\equiv_{\mathtt{fw}}$-summary, the set of outgoing properties from n_1 to m_1 in $\mathtt{G}$ is exactly $\mathcal{P}_{f(n_1) \to f(m_1)}$ and similarly the set of outgoing properties from n_2 to m_2 in $\mathtt{G}$ is exactly $\mathcal{P}_{f(n_2) \to f(m_2)}$.

Since $\mathtt{G}$ and $\mathtt{G}_{/\equiv_{\mathtt{fw}}}$ have the same schema (Property 1), it follows that in $\mathtt{G}^\infty$, the set of outgoing properties from n_1 to m_1, and from n_2 to m_2, is exactly $\mathcal{P}^\infty_{f(n_1),f(n_2) \to f(m_1),f(m_2)}$ (data edges can only be added through $\prec_{sp}$ constraints).

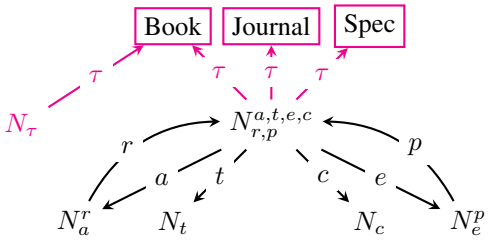


Figure 8: Weak summary of the RDF graph in Figure 3.

Since the above holds for any pair of data nodes $f(n_1), f(n_2)$ such that $f(n_1) \equiv_{\mathtt{fw}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$, and for any of their $(\mathtt{G}_{/\equiv_{\mathtt{fw}}})^\infty$ outgoing edges $f(n_1)$ p $f(m_1)$ and $f(n_2)$ p $f(m_2)$, hence $n_1 \equiv_{\mathtt{fw}} n_2$ in $\mathtt{G}^\infty$ holds.

The proof for $\equiv_{\mathtt{bw}}$ directly derives from the above one by considering incoming edges instead of outgoing ones; the proof for $\equiv_{\mathtt{fb}}$ then derives from those of $\equiv_{\mathtt{fw}}$ and $\equiv_{\mathtt{bw}}$ by considering both incoming and outgoing edges. □

4 Weak-equivalence summaries

In this section, we explore summaries based on the weak equivalence $\equiv_{\mathtt{W}}$ of graph nodes.

4.1 Weak summary

Our first new summary is solely based on weak equivalence:

Definition 14. (WEAK SUMMARY) *The weak summary of the graph* G*, denoted* $\mathtt{G}_{/\mathtt{W}}$*, is its quotient graph w.r.t. the weak equivalence relation* $\equiv_{\mathtt{W}}$*.*

For each set of $\equiv_{\mathtt{W}}$-equivalent G nodes, there is exactly one node in $\mathtt{G}_{/\mathtt{W}}$. Note that the partition of G nodes into $\equiv_{\mathtt{W}}$ equivalence classes is also *a partition of* G *data properties at the source*, that is: the sources of G edges labeled with a given data property p are all weakly equivalent. For instance, in Figure 3, the sources of all "editor" edges are in the weak equivalence class $\{r_1, r_2, r_3, r_4, r_5\}$.

The $\equiv_{\mathtt{W}}$ partition over the set of nodes of G also induces a *partition of the source cliques of* G. Indeed, if $n_1 \equiv_{\mathtt{W}} n_2$, their source cliques are connected through a chain of alternating cliques as sketched in Figure 5; conversely, such a chain, by definition, only connects weakly related resources. By a symmetrical reasoning, the weak equivalence classes of G also lead to a *partition over the target cliques of* G. For instance, in Figure 3, to the equivalent resource set $\{r_1, \ldots, r_5\}$ corresponds the set of source cliques $\{SC_1\}$ and the set of target cliques $\{TC_5\}$. Thus, *to each set S of weakly equivalent* G *nodes one can associate through a bijection, a set of* G *source cliques, and a set of* G *target cliques*. This leads to:

Definition 15. (SET REPRESENTATIVE) *Let N be any injective function taking as input two sets of URIs, and returning a new URI. Let S be an equivalence class of* G *data nodes w.r.t. $\equiv_{\mathtt{W}}$. The weak summary node representing all S nodes, also called the* set representative *of S in the summary, is:* $N(\bigcup_{r\in S} TC(r), \bigcup_{r\in S} SC(r))$.

In the above, N is called on: the set of data properties from all the target cliques of S nodes; and the set of data properties from all the source cliques of S nodes. We will use N to denote any function which assigns URIs to nodes in quotient (RDF) graphs.

Notations. We use N_r to denote the weak summary node representing a resource $r \in S$, and N_{SC}^{TC} to denote $N(TC, SC)$. For simplicity, we will mostly *omit the set delimiters* when showing TC and SC, and omit one such set altogether if it is empty.

For any resource $r \in \mathtt{G}$ which has types and a non-empty source or target clique, its types are carried to N_r in the weak summary.

In the particular case where a data resource $r \in \mathtt{G}$ is neither the source nor the target of data properties, i.e., $TC(r) = SC(r) = \emptyset$ (thus r can only appear in τ triples), r is represented by $N(\emptyset, \emptyset)$ which we denote N_τ in the sequel. Observe that if a resource r such that $TC(r) = SC(r) = \emptyset$ has types, the weak summary carries the respective types to N_τ.

The weak summary of the graph in Figure 3 is shown in Figure 8. Its nodes are: $N_{r,p}^{a,t,e,c}$ for the relatedness partition set $\{r_1, \ldots, r_5\}$. The target properties of this node are $TC(r_4)$ since the other nodes have empty target clique; the source properties are those in $SC(r_1)$ which is also the source clique of all the other resources in the set; N_a^r for the set $\{a_1, a_2\}$; N_t for the relatedness partition set $\{t_1, t_2, t_3, t_4\}$ etc. The edges from $N_{r,p}^{a,t,e,c}$ to N_a and N_t copy the outgoing edges of r_1, represented by $N_{r,p}^{a,t,e,c}$; the edge to N_e^p is due to r_2 and r_3; the edge to N_c is due to r_3. The edge from N_a^r to $N_{r,p}^{a,t,e,c}$ is due to a_1, and the edge from N_e^p to $N_{r,p}^{a,t,e,c}$ is due to e_1. The τ edges outgoing $N_{r,p}^{a,t,e,c}$ are due to the resources r_1, r_2 and r_3; the creation of N_τ (shown in purple font) is due to the node r_6 in the original graph.

The weak summary has the following important properties:

Proposition 6. (UNIQUE DATA PROPERTIES) *Each* $\mathtt{G}$ *data property appears exactly once in* $\mathtt{G}_{/\mathtt{W}}$.

Proof. First, note that any two weak summary nodes n_1, n_2 cannot be targets of the same data property. Indeed, if such a data property $\mathtt{p}$ existed, let TC be the target clique it belongs to. By the definition of the weak summary, n_1 was created by a call to $N(UTC_1, USC_1)$ such that TC is included in the union of target cliques UTC_1. Similarly, n_2 was created by a call to $N(UTC_2, USC_2)$ such that TC is included in the union of target cliques UTC_2. Then, $UTC_1 \cap UTC_2 \supseteq TC \neq \emptyset$, which contradicts the fact that different equivalence classes of $\mathtt{G}$ nodes correspond to disjoint sets of target cliques. The same holds for the sets of properties of which weak summary nodes are sources. Thus, any data property has at most one source and at most one target in $\mathtt{G}_{/\mathtt{W}}$. Further, by the definition of the summary as a quotient, every data property present in $\mathtt{G}$ also appears in the summary. Thus, there is exactly one $\mathtt{p}$-labeled edge in $\mathtt{G}_{/\mathtt{W}}$ for every data property in $\mathtt{G}$. □

Importantly, the above Property 6 warrants that the number of data edges in $\mathtt{G}_{/\mathtt{W}}$ is exactly $|\mathtt{D}_\mathtt{G}|_\mathtt{p}^0$, the number of distinct data properties in $\mathtt{G}$. Thus, its number of data nodes is at most $2|\mathtt{D}_\mathtt{G}|_\mathtt{p}^0$. The number of type triples in $\mathtt{G}_{/\mathtt{W}}$ is bound by $min(|\mathtt{T}_\mathtt{G}|_e, 2|\mathtt{D}_\mathtt{G}|_\mathtt{p}^0 * |\mathtt{T}_\mathtt{G}|_\mathtt{o}^0|)$: the latter corresponds to the case when every data node in $\mathtt{G}_{/\mathtt{W}}$ is of every type in $\mathtt{T}_\mathtt{G}$.

The next important property enjoyed by weak summaries is *completeness*, in the sense of Definition 13. Figure 9 exemplifies this, by tracing the transformation of a graph $\mathtt{G}$ on one hand, into $\mathtt{G}_{/\mathtt{W}}$, $(\mathtt{G}_{/\mathtt{W}})^\infty$, and then $((\mathtt{G}_{/\mathtt{W}})^\infty)_{/\mathtt{W}}$; and on the other hand, from $\mathtt{G}$ to $\mathtt{G}^\infty$ then $(\mathtt{G}^\infty)_{/\mathtt{W}}$. The graph at the bottom right in the figure is at the same time $((\mathtt{G}_{/\mathtt{W}})^\infty)_{/\mathtt{W}}$ and $(\mathtt{G}^\infty)_{/\mathtt{W}}$.

$\mathtt{W}$-summary completeness is our most technically invoved result. To establish it, we rely on the next Proposition:

Proposition 7. (SAME CLIQUES-$\mathtt{W}$) $\mathtt{G}^\infty$ *and* $(\mathtt{G}_{/\mathtt{W}})^\infty$ *have identical source clique sets, and identical target cliques sets. Further, any node* n *in* $\mathtt{G}^\infty$ *has the source clique* SC *and target clique* TC *iff* $f(n)$ *in* $(\mathtt{G}_{/\mathtt{W}})^\infty$ *has the source clique* SC *and target clique* TC*, with* f *the representation function (recall Proposition 5).*

Proposition 7 is established based on Proposition 5, and on the following Lemma:

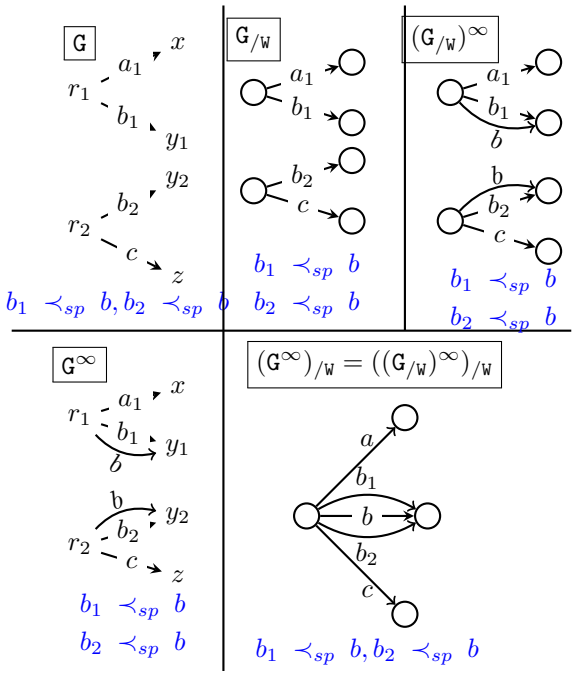


Figure 9: Weak summary completeness illustration (summary nodes shown as unlabeled circles).

Lemma 2. *Data properties are target-related (resp. source-related) in* $(\mathtt{G}_{/\mathtt{W}})^{\infty}$ *iff they are target-related (resp. source-related) in* $\mathtt{G}^{\infty}$.

Proof. We prove the lemma for target-related properties.

"Only if": If data properties are target-related in $(\mathtt{G}_{/\mathtt{W}})^{\infty}$, then they belong to a same target clique $TC_{\mathtt{W}}^{\infty}$ in $(\mathtt{G}_{/\mathtt{W}})^{\infty}$.

By Lemma 1, point 3, it follows that $TC_{\mathtt{W}}^{\infty}$ is the union of the saturations of a set of $\mathtt{G}_{/\mathtt{W}}$ cliques $(TC_{\mathtt{W}}^{1})^{+}, (TC_{\mathtt{W}}^{2})^{+}, \ldots, (TC_{\mathtt{W}}^{m})^{+}$. Then:

- For every $1 \leq j \leq m$:
  - $TC_{\mathtt{W}}^{j}$ is the target clique of a $\mathtt{G}_{/\mathtt{W}}$ node n^{j};
  - n^{j} represents a set of weakly-equivalent $\mathtt{G}$ resources, which are targets only of properties in $TC_{\mathtt{W}}^{j}$. Thus, the properties in $TC_{\mathtt{W}}^{j}$ are target-related in $\mathtt{G}$.
  - Thus, in $\mathtt{G}^{\infty}$, also, the properties in $TC_{\mathtt{W}}^{j}$ are target-related.
  - From this and the definition of a saturated graph and of a saturated target clique, it follows that the properties from $(TC_{\mathtt{W}}^{j})^{+}$ are target-related in $\mathtt{G}^{\infty}$.
- Further, still by Lemma 1, point 4, each $(TC_{\mathtt{W}}^{j})^{+}$ intersects at least another $(TC_{\mathtt{W}}^{l})^{+}$ for $1 \leq l \neq j < m$, thus the target properties in all the $(TC_{\mathtt{W}}^{j})^{+}$ for $1 \leq j \leq m$, and in particular $\mathtt{p}$, are target-related to each other in $\mathtt{G}^{\infty}$. Thus, $\mathtt{p}$ is target-related in $\mathtt{G}^{\infty}$ to all properties from $TC_{\mathtt{W}}^{\infty}$.

"If": if data properties are target-related in $\mathtt{G}^{\infty}$, then they belong to a same target clique TC^{∞} in $\mathtt{G}^{\infty}$. Let $n_1, \ldots, n_k$ be the set of all $\mathtt{G}$ resources which are values of some properties in TC^{∞}. By definition of an RDF summary and Proposition 5, each image of $f(n_i) = N(\cup_{r \in S_{n_i}} TC(r), \cup_{r \in S_{n_i}} SC(r))$ of n_i,

for $1 \leq i \leq k$ is at least the object of the same properties as n_i, hence all the properties of TC^∞ in $\mathtt{G}^\infty$ are target-related in $(\mathtt{G}_{/\mathtt{W}})^\infty$. □

The completeness of W-summaries, stated by Theorem 3 below, is then shown based on Proposition 7 by proving that W-summaries enjoy the sufficient condition stated by Theorem 1.

Theorem 3 (W-completeness)**.** *Weak summaries are complete.*

Proof. We show that W summaries enjoy the sufficient condition for completenes stated in Theorem 1. That is, given two nodes n_1, n_2 in $\mathtt{G}^\infty$, f the representation homomorphism corresponding to the weak-equivalence relation $\equiv_{\mathtt{W}}$, and $f(n_1)$, $f(n_2)$ the images of n_1, n_2 in $(\mathtt{G}_{/\mathtt{W}})^\infty$ through f (recall Proposition 5), let us prove that: $n_1 \equiv_{\mathtt{W}} n_2$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv_{\mathtt{W}} f(n_2)$ in $(\mathtt{G}_{/\equiv_{\mathtt{W}}})^\infty$.
Two summary nodes are weakly-equivalent if they have the same non-empty source or target clique. The claim to prove hence immediately follows from Property 7, which guarantees that n_1 and $f(n_1)$, resp. n_2 and $f(n_2)$, have the same source and target cliques. □

Weak summary size. The size of the weak summary is at most that of the original graph. This bound is reached if all nodes have distinct source and target properties. Moreover, Property 6 entails that the number of data edges in $\mathtt{G}_{/\mathtt{W}}$ is exactly the number of distinct data properties in G. Thus, its number of data nodes is at most twice this number. The number of type triples in $\mathtt{G}_{/\mathtt{W}}$ is bound by the smallest between the number of type edges in G.

4.2 Typed weak summary

In this section, we introduce a variant of the W-summary, based on the simple idea that RDF type information, when available, can be used to consider equivalent resources having the same *set of types* (recall that an RDF node may have 0, 1 or more types, not necessarily related). This summary represents all G nodes having the exact same types together, while for untyped resources, it reverts to the quotient by the weak equivalence relation $\equiv_{\mathtt{W}}$. Such type-conditioned summarization can be seen as applying two consecutive quotients to G. We start by introducing two helper notions:

Definition 16. (TYPE-BASED SUMMARY) *The type-based summary of* G*, denoted* $\mathtt{G}_{/\mathtt{T}}$*, is the summary of* G *through the* $\equiv_{\mathtt{T}}$ *equivalence relation.*

$\mathtt{G}_{/\mathtt{T}}$ groups together typed resources which have the same non-empty set of types, and copies each untyped G node, since none of them is equivalent to any other node. We assume available an *injective* function C which, called on a non-empty set of class URIs from G, returns a fresh URI (not in G). Figure 10 illustrates a type-based summary; observe the nodes produced by calls to C.

Further, we introduce:

Definition 17. (U-WEAK EQUIVALENCE AND SUMMARY)
Two nodes in a graph G *are* untyped-weak equivalent*, denoted* $n_1 \equiv_{\mathtt{UW}} n_2$*, iff* n_1 *and* n_2 *have no type in* G *and* $n_1 \equiv_{\mathtt{W}} n_2$*.*
The untyped-weak summary of G*, denoted* $\mathtt{G}_{/\mathtt{UW}}$*, is its summary through* $\equiv_{\mathtt{UW}}$*.*

Untyped weak equivalence is the weak equivalence $\equiv_{\mathtt{W}}$ restricted to untyped resources only. The untyped-weak summary $\mathtt{G}_{/\mathtt{UW}}$ treats the G triples whose subject and object lack types exactly like $\mathtt{G}_{/\mathtt{W}}$, and leaves all typed resources untouched. We can now define:

Definition 18. (TYPED WEAK SUMMARY) *The typed weak summary* $\mathtt{G}_{/\mathtt{TW}}$ *of an RDF graph* G *is the untyped-weak summary of the type-based summary of* G*, namely* $(\mathtt{G}_{/\mathtt{T}})_{\mathtt{UW}}$*.*

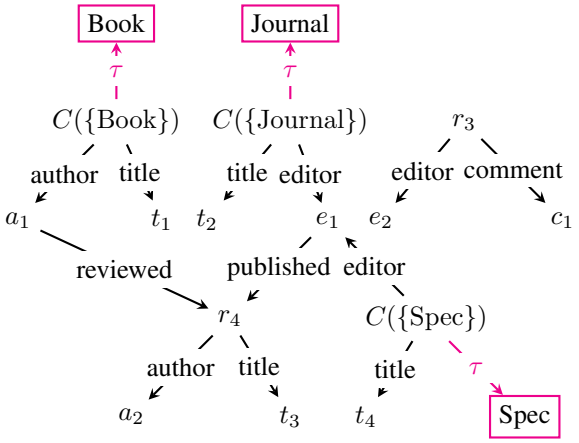


Figure 10: Type-based summary $\mathtt{G}_{/\mathtt{T}}$ of the RDF graph in Figure 3.

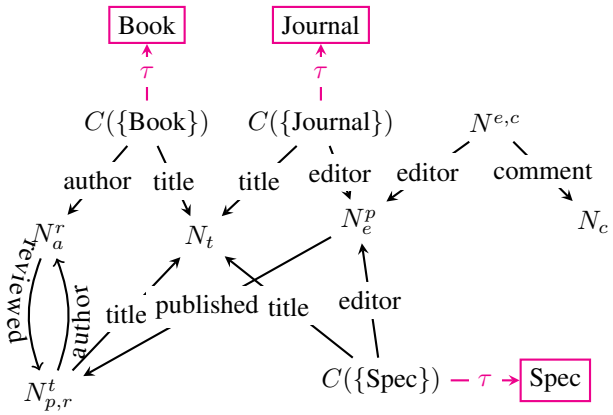


Figure 11: Typed weak summary $\mathtt{G}_{/\mathtt{TW}}$ of the graph $\mathtt{G}$ in Figure 3.

For example, Figure 11 shows the typed weak summary of the RDF graph in Figure 3. Compared with $\mathtt{G}_{/\mathtt{T}}$ from which it is built (Figure 10), the typed weak summary represents all editors with a single node, all titles with a single node etc., like $\mathtt{G}_{/\mathtt{W}}$ (Figure 8). However, in contrast with $\mathtt{G}_{/\mathtt{W}}$, $\mathtt{G}_{/\mathtt{TW}}$ represents publications by three distinct nodes, depending on their types.

Proposition 8. (TW-INCOMPLETENESS) *Typed weak summaries are incomplete.*

Figure 12 shows a counter-example. In $\mathtt{G}$ and $\mathtt{G}_{/\mathtt{TW}}$, resources are untyped; a typed resource only appears due to saturation and the constraint $a \hookleftarrow_d c$. Thus, in $\mathtt{G}_{/\mathtt{TW}}$, one (untyped) node represents all data property subjects; in $(\mathtt{G}_{/\mathtt{TW}})^{\infty}$, this (single) node gains a type, and in $((\mathtt{G}_{/\mathtt{TW}})^{\infty})_{/\mathtt{TW}}$, it is represented by a single node. In contrast, in $\mathtt{G}^{\infty}$, one resource is typed and the other one isn't, leading to distinct nodes in $(\mathtt{G}^{\infty})_{/\mathtt{TW}}$. This cannot be isomorphic with the result obtained from $\mathtt{G}$ if one starts by TW summarization, thus by Definition 13, $\mathtt{G}_{/\mathtt{TW}}$ is not complete.

Typed weak summary size. The size of the typed weak summary is at most that of the original graph. This bound is reached if all nodes have distinct types.

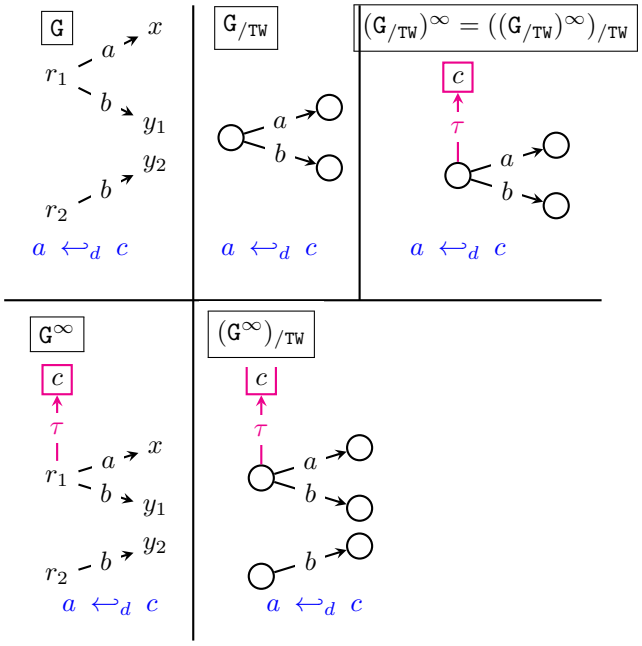


Figure 12: Typed weak summary completeness counter-example.

5 Strong equivalence summaries

We now present summaries based on the strong RDF node equivalence. Strong summaries (Section 5.1) are mainly based on this; typed strong summaries (Section 5.2) give preeminence to types.

5.1 Strong summary

Definition 19. (STRONG SUMMARY) *The* strong summary $\mathtt{S}$ *is an RDF summary based on the strong equivalence* $\equiv_S$.

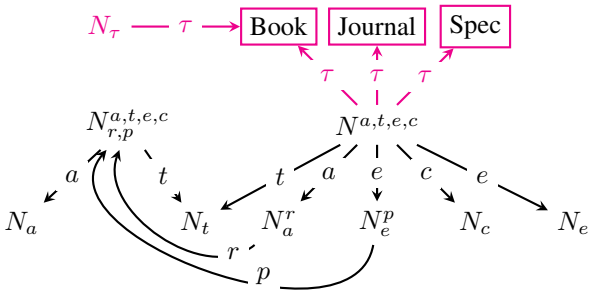


Figure 13: Strong summary of the RDF graph in Figure 3.

Recall from the definition of $\equiv_S$ that only data nodes having the same source clique and the same target clique are equivalent in this sense. Thus, it is easy to establish a bijection between any pair (source

clique, target clique) of a data node in $\mathtt{G}$, and a node in the strong summary. To follow through this intuition, in this section, we denote by N_{SC}^{TC} the node representing the set of $\equiv_{\mathtt{S}}$-equivalent nodes of $\mathtt{G}$ having the target clique TC and the source clique SC.

For example, the strong summary of the graph of Figure 3 is shown in Figure 13. The strong summary comprises the nodes:

- $N(\emptyset, SC_1) = N^{a,t,e,c}$, for r_1, r_2, r_3 and r_5;
- $N(TC_5, SC_1) = N_{r,p}^{a,t,e,c}$, for r_4;
- $N(TC_1, SC_2) = N_r^a$, for a_1;
- $N(TC_2, \emptyset) = N_t$, for t_1, t_2, t_3 and t_4;
- $N(TC_3, SC_3) = N_e^p$, for e_1;
- $N(TC_3, \emptyset) = N_e$, for e_2;
- $N(TC_4, \emptyset) = N_c$, for c_1;
- $N(TC_1, \emptyset) = N_a$, for a_2;
- $N(\emptyset, \emptyset) = N_\tau$ for r_6;
- Book, Journal, and Spec are copied from the schema of $\mathtt{G}$.

As a result, the number of data nodes in the data component $\mathtt{D}_{\mathtt{S}_\mathtt{G}}$ is bound by $min(|\mathtt{D}_\mathtt{G}|_n, (|\mathtt{D}_\mathtt{G}|_e^0)^2)$ (recall the notations introduced in Section 2.1). Indeed, $\mathtt{S}_\mathtt{G}$ cannot have more data nodes than $\mathtt{G}$; also, it cannot have more nodes than the number of source cliques times the number of target cliques, and each of these is upper-bounded by $|\mathtt{D}_\mathtt{G}|_e^0$. By a similar reasoning, the number of data triples in $\mathtt{S}_\mathtt{G}$ is bound by $min(|\mathtt{D}_\mathtt{G}|_e, (|\mathtt{D}_\mathtt{G}|_e^0)^4)$. In the worst case, $\mathtt{T}_{\mathtt{S}_\mathtt{G}}$ has as many nodes (and triples) as $\mathtt{T}_\mathtt{G}$; $\mathtt{S}_{\mathtt{S}_\mathtt{G}}$ is identical to $\mathtt{S}_\mathtt{G}$.

Similarly to weak summaries, strong summaries are *complete* (Definition 13); this is our last major theoretical result. We prove this based on counterparts of statements established for $\mathtt{G}_\mathtt{W}$:

Proposition 9. (SAME CLIQUES-S) *$\mathtt{G}^\infty$ and $(\mathtt{G}_{/\mathtt{S}})^\infty$ have identical source clique sets, and identical target cliques sets. Further, any node n in $\mathtt{G}^\infty$ has the source clique SC and target clique TC iff $f(n)$ in $(\mathtt{G}_{/\mathtt{S}})^\infty$ has the source clique SC and target clique TC, with f the representation function (recall Proposition 5).*

The above Proposition follows directly from Proposition 5 and the following Lemma:

Lemma 3. *Data properties are target-related (resp. source-related) in $(\mathtt{G}_{/\mathtt{S}})^\infty$ iff they are target-related (resp. source-related) in $\mathtt{G}^\infty$.*

Proof. "If": if data properties are target-related in $\mathtt{G}^\infty$, then they belong to a same target clique TC^∞ in $\mathtt{G}^\infty$. Let $n_1, \ldots, n_k$ be the set of all $\mathtt{G}$ resources which are values of some properties in TC^∞. By definition of an RDF summary and Proposition 5, each image of $f(n_i) = N(TC(n_i), SC(n_i))$ of n_i, for $1 \leq i \leq k$ is at least the object of the same properties as n_i, hence all the properties of TC^∞ in $\mathtt{G}^\infty$ are target-related in $(\mathtt{G}_{/\mathtt{S}})^\infty$.
"Only If": if two data properties p_1 and p_2 are target-related in $(\mathtt{G}_{/\mathtt{S}})^\infty$, then they belong to a same target clique $TC^{\mathtt{S},\infty}$, in which they are at distance $n \geq 0$, i.e., they are target-related because of a set $\bigcup_{i=0}^{n}\{r_{i+1}\}$ of nodes (recall Definition 7) which all have the target clique $TC^{\mathtt{S},\infty}$. In $\mathtt{G}_{/\mathtt{S}}$, each such r_{i+1} has a target clique $TC_i^\mathtt{S} \subseteq TC^{\mathtt{S},\infty}$, moreover each r_{i+1} results from a set of $\mathtt{G}$ nodes $n_{i+1}^j, j \geq 1$, which by definition of a strong RDF summary, have all the source clique $TC_i^\mathtt{S}$. Hence, every such n_{i+1}^j node has target clique $TC^{\mathtt{S},\infty}$ in $\mathtt{G}^\infty$ (since $\mathtt{G}$ and $\mathtt{G}_{/\mathtt{S}}$ have the same schema), in which p_1 and p_2 are target related. □

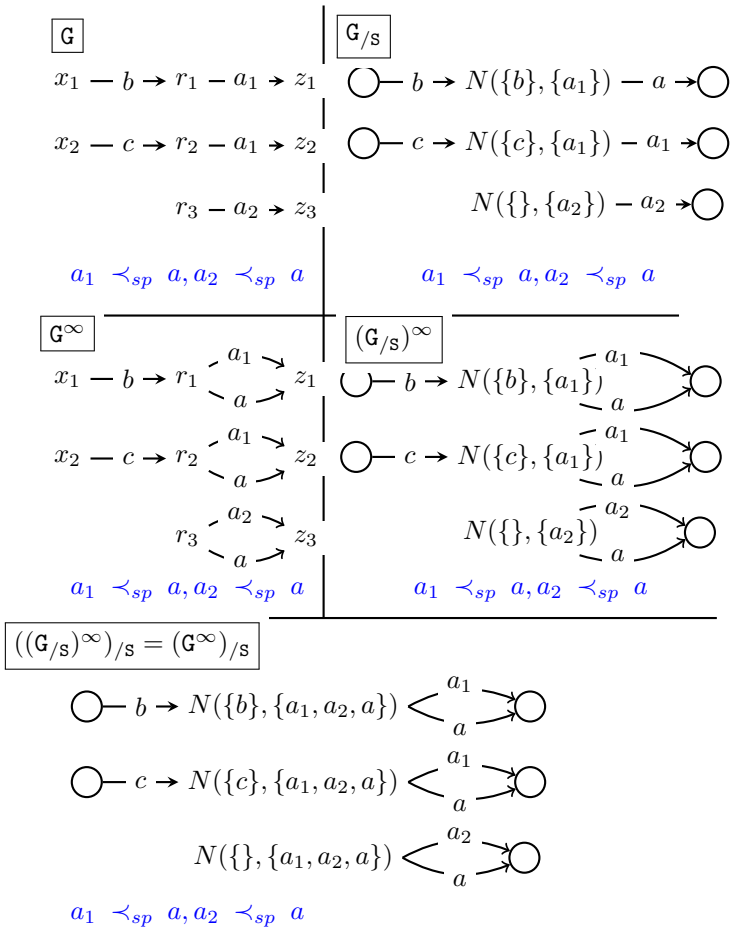


Figure 14: Illustration of the strong completeness statement (some summary nodes are shown as unlabeled circles).

Based on Proposition 9, we prove that $\mathtt{G}_\mathtt{S}$ enjoys the sufficient condition stated by Theorem 1:

Theorem 4 (S-completeness)**.** *Strong summaries are complete.*

Proof. We show that S summaries enjoy the sufficient condition for completenes stated in Theorem 1. That is, given two nodes n_1, n_2 in $\mathtt{G}^\infty$, f the representation homomorphism corresponding to the weak-equivalence relation $\equiv_\mathtt{S}$, and $f(n_1)$, $f(n_2)$ the images of n_1, n_2 in $(\mathtt{G}_{/\mathtt{S}})^\infty$ through f (recall Proposition 5), let us prove that: $n_1 \equiv_\mathtt{S} n_2$ in $\mathtt{G}^\infty$ iff $f(n_1) \equiv_\mathtt{S} f(n_2)$ in $(\mathtt{G}_{/\equiv_\mathtt{S}})^\infty$.
Two summary nodes are strongly-equivalent if they have the same non-empty source or target clique. The claim to prove hence immediately follows from Property 7, which guarantees that n_1 and $f(n_1)$, resp. n_2 and $f(n_2)$, have the same source and target cliques. □

Figure 14 illustrates completeness on an example.

Strong summary size. The number of data nodes in the data component $\mathtt{D}_{\mathtt{S}_\mathtt{G}}$ is bound by $min(|\mathtt{D}_\mathtt{G}|_n, (|\mathtt{D}_\mathtt{G}|_e^0)^2)$ (recall the notations introduced in Section 2.1). Indeed, $\mathtt{S}_\mathtt{G}$ cannot have more data nodes than G; also, it cannot have more nodes than the number of source cliques times the number of target cliques, and each

of these is upper-bounded by $|\mathtt{D}_\mathtt{G}|^0_e$. By a similar reasoning, the number of data triples in $\mathtt{S}_\mathtt{G}$ is bound by $min(|\mathtt{D}_\mathtt{G}|_e, (|\mathtt{D}_\mathtt{G}|^0_e)^4)$. In the worst case, $\mathtt{T}_{\mathtt{S}_\mathtt{G}}$ has as many nodes (and triples) as $\mathtt{T}_\mathtt{G}$; $\mathtt{S}_{\mathtt{S}_\mathtt{G}}$ is identical to $\mathtt{S}_\mathtt{G}$.

5.2 Typed strong summary

The summary presented here is the counterpart of the typed weak one from Section 4.2, but based on strong equivalence.

Definition 20. (U-STRONG EQUIVALENCE AND SUMMARY) *Two nodes in a graph* G *are* untyped-strong equivalent*, denoted* $n_1 \equiv_{\mathtt{US}} n_2$*, iff* n_1 *and* n_2 *have no type in* G *and* $n_1 \equiv_{\mathtt{S}} n_2$*. The untyped-strong summary of* G*, denoted* $\mathtt{G}_{/\mathtt{US}}$*, is its summary through* $\equiv_{\mathtt{US}}$.

Untyped strong equivalence restricts strong equivalence to untyped resources only. The summary $\mathtt{G}_{/\mathtt{US}}$ summarizes untyped data resources in strong fashion, and leaves all typed resources untouched (each is $\equiv_{\mathtt{US}}$ to itself). We can now define:

Definition 21. (TYPED STRONG SUMMARY) *The typed strong summary* $\mathtt{G}_{/\mathtt{TS}}$ *of an RDF graph* G *is the untyped-strong summary of the type-based summary of* G*, namely:* $(\mathtt{G}_{/\mathtt{T}})_{/\mathtt{US}}$.

In our example, it turns out that $\mathtt{G}_{/\mathtt{TS}}$ for the RDF graph in Figure 3 coincides with the $\mathtt{G}_{/\mathtt{TW}}$ one shown in Figure 11. As can be easily seen from their definitions, the type-weak and type-strong summaries behave identically on the triples involving typed resources; on the untyped ones, the difference is of the same nature as the difference between the strong and weak summaries.

Proposition 10. (TYPED STRONG INCOMPLETENESS)
Typed strong summaries are incomplete.

More generally, in the presence of domain ($\hookleftarrow_d$) or range ($\hookrightarrow_r$) RDF schema constraints, one cannot compute $(\mathtt{G}^\infty)_{/\mathtt{TS}}$ from G because the TS summary represents typed resources differently from the untyped ones. The $\hookleftarrow_d$ and $\hookrightarrow_r$ constraints may turn untyped resources into typed ones, thus leading to divergent representations of the data nodes of G in $\mathtt{G}_{/\mathtt{TS}}$ and respectively $(\mathtt{G}^\infty)_{/\mathtt{TS}}$. Thus, one cannot build from G, without saturating it, a graph isomorphic to $(\mathtt{G}^\infty)_{/\mathtt{TS}}$.
Typed strong summary size. Similarly to the type weak summary, the size is at most that of the original graph. This bound is also reached if all nodes have distincts types.

6 Summary comparison

How do our summaries relate to each other, and how can one compare them? $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{S}}$ are complete (Theorems 3 and 4), while $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$ are not (Propositions 8 and 10).

Composing summaries. One can easily prove that $(\mathtt{G}_{/\mathtt{S}})_{/\mathtt{W}} = \mathtt{G}_{/\mathtt{W}}$, i.e., one could compute $\mathtt{G}_{/\mathtt{W}}$ by first summarizing G into $\mathtt{G}_{/\mathtt{S}}$, and then applying weak summarization on this (typically much smaller) graph; similarly, one can show that $(\mathtt{G}_{/\mathtt{TS}})_{/\mathtt{TW}} = \mathtt{G}_{/\mathtt{TW}}$. It is also the case that $(\mathtt{G}_{/\mathtt{W}})_{/\mathtt{S}} = \mathtt{G}_{/\mathtt{W}}$, i.e., strong summarization cannot compress a weak summary further, and similarly $(\mathtt{G}_{/\mathtt{TW}})_{/\mathtt{TS}} = \mathtt{G}_{/\mathtt{TW}}$. Figure 15 summarizes the main relationships between summaries.

Size comparison. As a consequence of the above, $\mathtt{G}_{/\mathtt{W}}$ has at most as many nodes and as most as many edges as $\mathtt{G}_{/\mathtt{S}}$ (weak summarizes at least as much as strong), and similarly, the typed weak summary $\mathtt{G}_{/\mathtt{TW}}$ summarizes at least as much as the typed strong summary $\mathtt{G}_{/\mathtt{TS}}$.

Accuracy comparison. Another interesting measure is the closeness between a graph and its summary. By Definition 11, any subgraph of the G can be embedded through homomorphism in any summary of

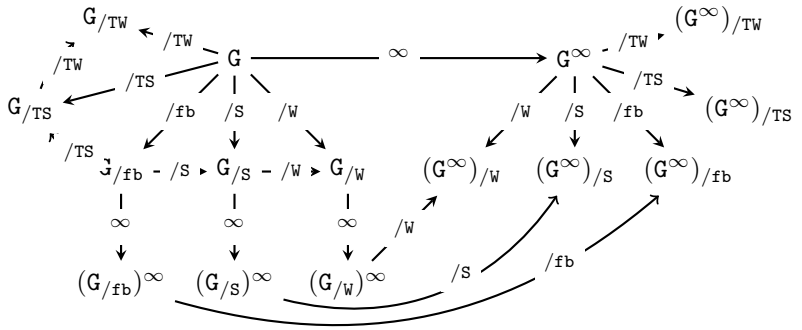


Figure 15: Main relations between our summaries.

G; however, the opposite is not true, as the representation of data nodes by summary nodes may lead to subgraphs in the summary which do not exist in G. For instance, $G_{/W}$ in Figure 8 a node has outgoing edges labeled a and e, whereas such a node does not exist in the original graph (Figure 3). Formally:

Definition 22. (ACCURACY LOSS) *The accuracy loss of a summary $G_{/\equiv}$ w.r.t. G, denoted $AL(G_{/\equiv}, G)$, is the percentage of connected subgraphs of $G_{/\equiv}$ which are not isomorphic to any subgraph of G.*

The k-level accuracy loss, denoted $AL_k(G_{/\equiv}, G)$, for some $k \geq 2$, is the percentage of connected subgraphs of $G_{/\equiv}$ of size at most k, which lack isomorphic counterparts in G.

The lower AL, the more accurate the summary is w.r.t. G. Computing $AL(G_{/\equiv}, G)$ may be costly because of an excessively high number of summary subgraphs to check. Since $AL_k(G_{/\equiv}, G) \leq AL(G_{/\equiv}, G)$, we measure $AL_k(G_{/\equiv}, G)$ and use it as a lower bound approximation for AL. Note that small $G_{\equiv}$ subgraphs show up in queries at least as frequently as larger ones, thus accuracy errors on small subgraphs is an interesting measure.

It is easy to see that for any graph G, summary $G_{/\equiv}$ and integers k_1, k_2 such that $k_1 < k_2$, $AL_{k_1}(G_{/\equiv}, G) \leq AL_{k_2}(G_{/\equiv}, G)$. This is because any summary subgraph of size k_1 which has no isomorphic image in the graph, is a subgraph of many subgraphs of size k_2, which obviously are not isomorphic to a subgraph of G, either.

Clearly, AL and AL_k depend the RDF equivalence $\equiv$. At one extreme, if every node is only equivalent to itself, the summary has the size of G, and no accuracy loss. Conversely, if all data nodes are equivalent, the summary has a single node, having all types from G and an edge going from the node to itself for every data property in G. This maximizes AL for a given graph.

Relation with bisimulation summaries. Recast in our framework, the three RDF bisimulation summaries are complete (Theorem 2). Our W and S summaries are symetric w.r.t. G edge source and target nodes, thus the comparison with the unidirectional FW and BW bisimulation summaries is limited. A deeper comparison is possible and interesting with the (symetric) FB-bisimulation summary $G_{/fb}$. Since $G_{/fb}$ preserves G structure and thus node cliques, we have $(G_{/fb})_{/W} = G_{/W}$ and $(G_{/W})_{/fb} = G_{/W}$, and similar statements for $G_{/S}$, $G_{/TW}$, and $G_{/TS}$. Different from our summaries, $G_{/fb}$ guarantees $AL = 0$, but it is usually *much* larger, as illustrated in Section 3.2 and in our experiments, making it impractical for our goals.

7 Summarization algorithms

We have implemented a summarization tool (15.000 lines, Java 1.8) which, given an RDF graph G, builds $G_{/W}$, $G_{/TW}$, $G_{/S}$ and $G_{/TS}$. G is initially stored in a PostresSQL server, in a dictionary-encoded triple table; summaries are built mostly in memory and relying also on a few SQL queries. For efficiency, the algorithms for building G_W and G_{TW} are significantly different from following the summary definitions, and

we outline them below. G_S and G_{TS} are built in a fashion closer to the definitions. We introduce the data structures and then describe our algorithms.

	Weak	Typed Weak	Strong	Typed Strong
rd, dr	✓	✓	✓	✓
dcls	✓	✓	✓	✓
clsd		✓		✓
dpSrc, dpTarg	✓	✓		
srcDps, targDps	✓	✓		
dtp	✓	✓	✓	✓
sc, tc			✓	✓
sToSc, oToTc			✓	✓
scToSrc, tcToTarg			✓	✓

Figure 16: Overview of the data structures for different summaries.

7.1 Data structures

Currently, our summary graphs are built in memory, based on the Trove library[4] providing efficient collection data structures. We represent G URIs by integers, and store them in a set of map/ multi-map data structures, listed in Figure 16, together with the summaries which use them. All structures are not needed by all summaries, but many of them are used repeatedly. We describe them below, underlined for visibility.

rd maps G data nodes to their the $G_{/\equiv}$ representative; the dr multimap stores the reverse mapping (summary node to all G node it represents). We call *class set* a set of types attached to a same resource in G; to each non-empty class set corresponds exactly one node in $G_{/TW}$ and $G_{/TS}$. clsd associates the corresponding summary node to each class set, while dcls stores the class set of every (typed) data node from G. In a weak summary $G_{/W}$, every data property appears only once (Property 6); in $G_{/TW}$, appears *at most once with an untyped source and untyped target* (it may appear several times e.g., with subjects of different types and/or with typed vs. untyped subjects, see the four *title* edges in the $G_{/TW}$ summary in Figure 11). The maps dcls, dpSrc and dpTarg associate to a data property, its source and target (in G_W), respectively, its (possible) single untyped source and untyped target (in G_{TW}). Conversely, for each G data node (for G_W), respectively, for each *untyped* G data node ($G_{/TW}$), the sets of data properties it is a source, respectively target of are stored in the maps srcDps (targDps). dtp maps every data property to the summary triple(s) it appears in. Each source and target clique is assigned an integer ID, and the IDs are stored in two lists, sc and tc. sToSc and oToTc associate to each data node, the ID of its source respectively target clique. Finally, for each source (target) clique ID, the $G_{/S}$ (and $G_{/TS}$) nodes having that clique are stored in scToSrc and tcToTarg.

7.2 Summarization algorithms

Building $G_{/W}$ and $G_{/TW}$ requires a single pass over the data. We create candidate summary nodes as we go over the G triples, and record the association between G nodes and the respective summary nodes in the maps described above, gradually gaining knowledge about the data properties that each G node is a source and target of. When nodes share a source or target property, their summary representative nodes are fused (again based on Property 6). Building $G_{/S}$ and $G_{/TS}$ require a first pass to compute the cliques, and a second one to build a summary node for each (source clique, target clique) pair associated to one or several G nodes.

[4] http://trove.starlight-systems.com/

Algorithm 1 Summarizing data triples in $\mathtt{G}_{/\mathtt{W}}$

Input: Data triples table $\mathtt{D}_{\mathtt{G}}$, the summary $\mathtt{G}_{/\mathtt{W}}$
Output: Data triples represented in $\mathtt{G}_{/\mathtt{W}}$
1: **for each** s p o in $\mathtt{D}_{\mathtt{G}}$ **do**
2: $src \leftarrow$ getSource(s, p, $\mathtt{G}_{/\mathtt{W}}$)
3: $targ \leftarrow$ getTarget(o, p, $\mathtt{G}_{/\mathtt{W}}$)
4: // getTarget may have modified src and vice-versa
5: $src \leftarrow$ getSource(s, p, $\mathtt{G}_{/\mathtt{W}}$)
6: $targ \leftarrow$ getTarget(o, p, $\mathtt{G}_{/\mathtt{W}}$)
7: **if** !existsDataTriple($\mathtt{G}_{/\mathtt{W}}$, src, p, $targ$) **then**
8: createDataTriple($\mathtt{G}_{/\mathtt{W}}$, src, p, $targ$)
9: **procedure** CREATEDATATRIPLE($\mathtt{G}_{/\mathtt{W}}$, src, p, $targ$)
10: dtp.put(p, src p $targ$)
11: dpSrc.put(p, src), srcDps.put(src, p)
12: dpTarg.put(p, $targ$), targDps.put($targ$, p)

Summarizing data triples in $\mathtt{G}_{/\mathtt{W}}$. Algorithm 1 shows the procedure for summarizing data triples in a weak summary. The methods $getSource$ and $getTarget$ implement the representation functions, which map data nodes from G to data nodes in $\mathtt{G}_{/\mathtt{W}}$. These methods create the respective nodes the first time they are called for a specific data property; on subsequent calls, the respective nodes are returned.

Algorithm 2 shows how we map the subjects of data triples in G to data nodes in $\mathtt{G}_{/\mathtt{W}}$. After the method $getSource$ has been executed, the source node of the property p, denoted src_u, and the node representing the subject s, denoted src_s must be the same. If neither src_u nor src_s exist yet, src will be a new data node representing s (line 5). In the cases when one of the nodes exist and the other does not, we simply use the existing node as src (lines 5-10). Moreover, if src_s had not existed, it means that s was unrepresented, so we assign src as its representative (line 7). On the other hand, if both src_u and src_s exist and they are the same, it does not matter which one we choose as src (line 12). When they differ, we have to merge them (line 14). The $mergeDataNodes$ method replaces the node with less edges. The remaining node becomes the source/target of all the edges of the replaced node, and it is assigned to represent all the resources represented by the replaced node. Therefore, this method updates the replaced node in any of the maps rd, dr, dpSrc, srcDps, dpTarg, targDps, with the remaining node. Effectively, merging data nodes that are attached to common properties *gradually builds property cliques*. The method $getTarget$ in Algorithm 1 is very similar to $getSource$, the only difference being in passing the object o instead of the subject, and working with untyped property targets instead of sources.

Algorithm 3 outlines the summarization of type triples in $\mathtt{G}_{/\mathtt{W}}$. It iterates over the subjects and classes of all type triples and tries to represent each resulting pair as follows. We look up the summary data node src representing s. If s is attached to any data property in G, this means s has already been represented when summarizing data triples and we assign its representative data node to src and we add the class to the class set of src. Otherwise, s has some types but no data property, thus we record it as a typed-only resource. Once all type triples whose subjects also had some data property have been represented, if there are any typed-only resources, we represent them as well. The procedure representTypedOnly creates the single data node N_τ representing all resources from the list of typed-only resources and having *all* the types of typed-only resources (recall Section 4.1).

7.2.1 Typed weak summmary

Summarizing data triples in $\mathtt{G}_{/\mathtt{TW}}$. The basic procedure for summarizing data triples in $\mathtt{G}_{/\mathtt{TW}}$ (Algorithm 4) is very similar to the one of weak presented in Algorithm 1, with the difference that the entries

Algorithm 2 Representing the subject s of a data property p in G with a data node in $\mathtt{G}_{/\mathtt{W}}$

Input: s, p, $\mathtt{G}_{/\mathtt{W}}$
Output: Data triples represented in $\mathtt{G}_{/\mathtt{W}}$ Variables:

src_u data node representing an untyped source of p

src_s data node representing a (possibly typed) resource s

1: **procedure** GETSOURCE(s, p, $\mathtt{G}_{/\mathtt{W}}$)
2: $src_u \leftarrow \mathtt{dpSrc}.get(\mathtt{p})$
3: $src_s \leftarrow \mathtt{rd}.get(\mathtt{s})$
4: **if** $src_u = \perp \wedge src_s = \perp$ **then**
5: **return** createDataNode($\mathtt{G}_{/\mathtt{W}}$, s)
6: **else if** $src_u \neq \perp \wedge src_s = \perp$ **then**
7: $\mathtt{rd}.put(\mathtt{s}, src)$, $\mathtt{dr}.put(src, \mathtt{s})$
8: **return** src_u
9: **else if** $src_u = \perp \wedge src_s \neq \perp$ **then**
10: **return** src_s
11: **else if** $src_s = src_u$ **then**
12: **return** src_s
13: **else**
14: **return** mergeDataNodes($\mathtt{G}_{/\mathtt{W}}$, src_s, src_u)
15: **procedure** CREATEDATANODE($\mathtt{G}_{/\equiv}$, r)
16: $d \leftarrow$ newInteger
17: $\mathtt{rd}.put(r, d)$, $\mathtt{dr}.put(d, r)$
18: **return** d

are stored to the maps only if src and $targ$ respectively, are untyped[5]. Further, the mapping of data nodes in $\mathtt{G}_{/\mathtt{TW}}$ is similar to the one of weak summary; they diverge when both src_s and src_u exist. *The decision on merging data nodes depends on the typing of* src_s. If src_s is typed, or it is untyped but already the same as src_u, we choose src_s as src (lines 12-13). On the other hand, if the two nodes are both untyped and different from each other, we must merge them (line 15). Therefore, in the typed weak summary only untyped data nodes may be merged, while the weak allows for the merging of typed data nodes as well. More precisely, since in the weak summary the data triples are represented *before* the type triples, the typing information is not yet available - the merged nodes may or may not be typed. However, in the typed weak summary the type triples are represented first and taken into account during the summarization of data triples to build a finer-grained summary.

Summarizing type triples in $\mathtt{G}_{/\mathtt{TW}}$. Algorithm 5 shows how we summarize type triples in the typed weak summary. For each distinct subject we retrieve its class set (line 3) and the representative data node of the class set (line 4). Observe that in a typed weak summary there is one data node *per distinct class set* which represents all subjects having the class set. If such a data node already exists for the given class set, we simply store it as the representative of the current subject (line 6). Otherwise, we create a new data node d, which represents s and has the same class set as s (line 8). Further, d is stored as the representative of its class set (line 9).

Summarizing schema. Prior to summarization, the encoded schema table was created. Since the schema of G and $\mathtt{G}_{/\equiv}$ is the same, no action is needed. **JDBC**. The results of all queries issued to Postgres are fetched through JDBC. A JDBC parameter that can have an impact on the summarization time is the fetch

[5]Since in $\mathtt{G}_{/\mathtt{TW}}$ all typed data nodes are created before untyped ones and the assigned values are sequential, the method $isTyped(d)$ is as simple as checking whether d is less than or equal to the highest typed data node.

Algorithm 3 Summarizing type triples in $\mathtt{G}_{/\mathtt{W}}$

Input: Type triples table $\mathtt{T}_{\mathtt{G}}$, the summary $\mathtt{G}_{/\mathtt{W}}$
Output: Type triples represented in $\mathtt{G}_{/\mathtt{W}}$

```
1:  toRes ← [ ]; toCls ← [ ]
2:  for each (s, c) in T_G do
3:      repr ← representTypeTripleG_/W, s, c
4:      if repr = false then
5:          toRes.add(s), toCls.add(c)
6:  if toRes.size > 0 then
7:      representTypedOnly(G_/W, toRes, toCls)
8:  procedure REPRESENTTYPETRIPLE(G_/W, s, c)
9:      d ← rd.get(s)
10:     if d =⊥ then
11:         return false
12:     cls_d ← dcls.get(d), cls_d.add(d, c)
13:     return true
14: procedure REPRESENTTYPEDONLY(G_/W, toRes, toCls)
15:     N_τ ← newInteger
16:     for each r in toRes do
17:         rd.put(r, N_τ), dr.put(N_τ, r)
18:     cls_d ← dcls.get(N_τ)
19:     for each c in toCls do
20:         cls_d.add(c)
```

size. Larger fetch size is better because there are less real connections to the database to get all the results. But too large a fetch size will make you wait until that many results are available. The optimal fetch size will vary depending on the hardware setting.

7.2.2 Strong summary

Summarizing data triples in $\mathtt{G}_{/\mathtt{S}}$. Instead of building the cliques gradually as in the weak summary, in Algorithm 6 we compute all source and target cliques at the very beginning (lines 1-2). Then in lines 3-15, for each data node d in $\mathtt{G}$, a representative data node d_{repr} is assigned in $\mathtt{G}_{/\mathtt{S}}$. This data node d_{repr} must have the exact same source and target clique as d, and there can be at most one such node for any two source and target cliques. Finally, for each data triple $\mathtt{s}\ \mathtt{p}\ \mathtt{o}$ in $\mathtt{D}_{\mathtt{G}}$, we retrieve the representative data node of $\mathtt{s}$, denoted src, and of $\mathtt{o}$, denoted $targ$, forming a data triple $src\ \mathtt{p}\ targ$ in $\mathtt{G}_{/\mathtt{S}}$. The method for creating the data triple is the same as for the weak summary in Algorithm 1.

The procedure buildSourceCliques($\mathtt{D}_{\mathtt{G}}$, $\mathtt{G}_{/\mathtt{S}}$) invokes the procedure computeSourceClique($\mathtt{G}_{/\mathtt{S}}$, $\mathtt{s}$, $P_{\mathtt{s}}$) shown in Algorithm 7 for each subject in $\mathtt{D}_{\mathtt{G}}$ and the set of data properties of which it is the source.

First, in Algorithm 7 (lines 3-6) we try to find an existing source clique, denoted $srcClq$, that shares at least one property with $P_{\mathtt{s}}$. If such a clique exists we simply add to it all the properties from $P_{\mathtt{s}}$ (line 8). Otherwise, $srcClq$ is a new set with all the properties of $P_{\mathtt{s}}$, and we add it to the list of all source cliques $\mathtt{sc}$ in $\mathtt{G}_{/\mathtt{S}}$ (lines 10-11). Finally, $srcClq$ is assigned to $\mathtt{s}$ in $\mathtt{sToSc}$ (line 12).

The procedures for building target cliques are very similar to these described for the source cliques, so we omit the discussion.

Summarizing type triples in $\mathtt{G}_{/\mathtt{S}}$. The types in $\mathtt{G}_{/\mathtt{S}}$ are summarized in the same manner as for the $\mathtt{W}_{\mathtt{G}}$, described in Algorithm 3.

Algorithm 4 Representing the subject of a data property in $\mathtt{G}$ with a data node in $\mathtt{G}_{/\mathtt{TW}}$

Input: The subject $\mathtt{s}$ of the data triple, the property $\mathtt{p}$ of $\mathtt{s}$, the summary $\mathtt{G}_{/\mathtt{TW}}$
Output: Data triples represented in $\mathtt{G}_{/\mathtt{TW}}$

procedure GETSOURCE($\mathtt{s}$, $\mathtt{p}$, $\mathtt{G}_{/\mathtt{TW}}$)
  $src_u \leftarrow \mathtt{dpSrc}.get(\mathtt{p})$
  $src_s \leftarrow \mathtt{rd}.get(\mathtt{s})$
  if $src_u = \bot \wedge src_s = \bot$ **then**
    return createDataNode($\mathtt{G}_{/\mathtt{TW}}$, $\mathtt{s}$)
  else if $src_u \neq \bot \wedge src_s = \bot$ **then**
    $\mathtt{rd}.put(\mathtt{s}, src)$, $\mathtt{dr}.put(src, \mathtt{s})$
    return src_u
  else if $src_u = \bot \wedge src_s \neq \bot$ **then**
    return src_s
  else if $src_u \neq \bot \wedge src_s \neq \bot$ **then**
    if isTyped(src_s) $\vee$ $(src_s = src_u)$ **then**
      return src_s
    else
      return mergeDataNodes($\mathtt{G}_{/\mathtt{TW}}$, src_s, src_u)

Algorithm 5 Summarizing type triples in $\mathtt{G}_{/\mathtt{TW}}$

Input: Type triples table $\mathtt{T}_{\mathtt{G}}$, the summary $\mathtt{G}_{/\mathtt{TW}}$
Output: Type triples represented in $\mathtt{G}_{/\mathtt{TW}}$

$typ \leftarrow$ eval`SELECT s, c FROM` $\mathtt{T}_{\mathtt{G}}$ `ORDER BY s`
for each distinct $\mathtt{s}$ in typ **do**
  $cls_{\mathtt{s}} \leftarrow \mathtt{dcls}.get(\mathtt{s})$
  $d \leftarrow \mathtt{clsd}.get(cls_{\mathtt{s}})$
  if $d \neq \bot$ **then**
    $\mathtt{rd}.put(\mathtt{s}, d)$, $\mathtt{dr}.put(d, \mathtt{s})$
  else
    $d \leftarrow$ createDataNode$\mathtt{G}_{/\mathtt{TW}}$, $cls_{\mathtt{s}}$, $\mathtt{s}$
    $\mathtt{clsd}.put(cls_{\mathtt{s}}, d)$

procedure CREATEDATANODE($\mathtt{G}_{/\mathtt{TW}}$, r, cls_r)
  $d \leftarrow$ newInteger
  $\mathtt{rd}.put(r, d)$, $\mathtt{dr}.put(d, r)$, $\mathtt{dcls}.put(d, cls_r)$
  return d

Algorithm 6 Summarizing data triples in $\mathtt{G}_{/\mathtt{S}}$

Input: Data triples table $\mathtt{D}_{\mathtt{G}}$, the summary $\mathtt{G}_{/\mathtt{S}}$
Output: Data triples represented in $\mathtt{G}_{/\mathtt{S}}$
Variables:
$scId$**,** $tcId$ - source (target) clique ID
D_{SC}, D_{TC} - the set of source (target) data nodes in $\mathtt{G}_{/\mathtt{S}}$ representing the source clique $scId$ (target clique $tcId$)

1: buildSourceCliques$\mathtt{D}_{\mathtt{G}}$, $\mathtt{G}_{/\mathtt{S}}$
2: buildTargetCliques$\mathtt{D}_{\mathtt{G}}$, $\mathtt{G}_{/\mathtt{S}}$
3: **for each** distinct data node d in $\mathtt{D}_{\mathtt{G}}$ **do**
4: $scId \leftarrow \mathtt{sToSc}.get(d)$, $tcId \leftarrow \mathtt{oToTc}.get(d)$
5: $D_{SC} \leftarrow \mathtt{scToSrc}.get(scId)$
6: $D_{TC} \leftarrow \mathtt{tcToTarg}.get(tcId)$
7: **if** $D_{SC} \cap D_{TC} \neq \emptyset$ **then**
8: $\mathtt{rd}.put(r, d_{repr})$, $\mathtt{dr}.put(d_{repr}, r)$
9: **else**
10: $d_{repr} \leftarrow$ createDataNode($\mathtt{G}_{/\mathtt{S}}$, d)
11: $\mathtt{scToSrc}.put(scId, d_{repr})$
12: $\mathtt{tcToTarg}.put(tcId, d_{repr})$
13: **for each** $\mathtt{s}$ $\mathtt{p}$ $\mathtt{o}$ in $\mathtt{D}_{\mathtt{G}}$ **do**
14: $src \leftarrow \mathtt{rd}.get(\mathtt{s})$, $targ \leftarrow \mathtt{rd}.get(\mathtt{o})$
15: **if** !existsDataTriple($\mathtt{G}_{/\mathtt{S}}$, src, $\mathtt{p}$, $targ$) **then**
16: createDataTriple($\mathtt{G}_{/\mathtt{S}}$, src, $\mathtt{p}$, $targ$)

Algorithm 7 Computing source cliques in $\mathtt{G}_{/\mathtt{S}}$

Input: The summary $\mathtt{G}_{/\mathtt{S}}$, subject $\mathtt{s}$ from $\mathtt{D}_{\mathtt{G}}$, the set of data properties $P_{\mathtt{s}}$ of which $\mathtt{s}$ is the source
Output: $\mathtt{sc}$ list and $\mathtt{sToSc}$ map of $\mathtt{G}_{/\mathtt{S}}$ populated

1: **procedure** COMPUTESOURCECLIQUE($\mathtt{G}_{/\mathtt{S}}$, $\mathtt{s}$, $P_{\mathtt{s}}$)
2: $srcClq \leftarrow \perp$
3: **for each** c **in** $\mathtt{sc}$ **do**
4: **if** $c \cap P_{\mathtt{s}} \neq \emptyset$ **then**
5: $srcClq \leftarrow c$
6: **break**;
7: **if** $srcClq \neq \perp$ **then**
8: $srcClq.addAll(P_{\mathtt{s}})$
9: **else**
10: $srcClq.addAll(P_{\mathtt{s}})$
11: $\mathtt{sc}.add(srcClq)$
12: $\mathtt{sToSc}.put(\mathtt{s}, \mathtt{sc}.\text{indexOfsrcClq}+1)$

Graph G	$\|G\|$	$\|G_{/W}\|$ cf_W	$\|G_{/TW}\|$ cf_{TW}	$\|G_{/S}\|$ cf_S	$\|G_{/TS}\|$ cf_{TS}
John Peel (BBC Radio)	271k	33 8223	51 5320	99 2741	61 4448
INSEE Geo	369k	311 1186	360 1025	317 1164	361 1022
DBpedia Person	8M	10 **788957**	12 **657464**	28 **281770**	17 **464092**
DBLP	150M	71 **2123767**	258 **584447**	206 **731978**	285 **529079**
LinkedCT	49M	442 **111050**	505 **97196**	729 **67331**	598 **82081**
LinkedMDB	6.14M	8 **118155**	12 **78770**	8 **118155**	12 **78770**
LUBM	1M	162 **7579**	267 **4599**	208 **5903**	267 **4599**
LUBM	10M	162 **74013**	267 **44907**	206 **58204**	267 **44907**
LUBM	100M	162 **705897**	267 **428297**	209 **547155**	267 **428297**
WatDiv	10M	124 **88035**	982 **11117**	249 **43841**	1187 **9196**
BSBM	1M	1046 956	3054 327	1067 937	3054 327
BSBM	10M	4066 2591	14813 711	4101 2569	14813 711
BSBM	100M	13390 7775	40270 2585	13429 7753	40270 2585

Figure 17: Summary graph sizes and compression factors (rounded down to the closest smaller integer) for various graphs.

Proposition 11. (ALGORITHM CORRECTNESS) *Given an RDF graph* G*: Algorithms 1 and 3, together, build the weak summary* $G_{/W}$*. Algorithms 4 and 5 build the typed weak summary* $G_{/TW}$*. Algorithm 6 shows how to build the strong summary* S_G*; building* TS_G *is very similar.*

Algorithm complexity. The complexity of building $G_{/W}$ and $G_{/TW}$ is dominated by the cost of merging summary nodes, in the worst case, once per triple in G, leading to $O(|G| \cdot DPG)$, where DPG is the number of distinct data properties in G. To build $G_{/S}$ and $G_{/TS}$, we actually compute the cliques. First, we find the distinct data properties of every distinct data node in G, at a cost of $O(N \cdot \log(N) + |G|)$, where N is the number of data nodes in G. Then, for each data node n, we examine all the cliques known so far, to see if some should be fused due to the data properties of n: in the worst case we may do DPG fusions at a cost of DPG each, thus a total cost of DPG^2. Further, if there are fusions, the associations between the data nodes previously visited and their cliques need to be updated. A (very pessimistic) upper bound for the number of nodes concerned is N. Thus, the complexity of this stage is in $O(N(N + DPG^2))$; this dominates the complexity of the first step and thus dictates the complexity of building $G_{/S}$ and $G_{/TS}$. As our experiments show next, the actual scale-up is much better, and close to linear in $|G|$.

Computing accuracy loss. We developed a tool which enumerates summary subgraphs and for each subgraph, asks an existential SQL query to check if the subgraph has matches in the data. Our algorithm enumerates smallest subgraphs first, and re-uses the knowledge that a certain subgraph has no matches in G to avoid asking queries that correspond to larger graphs comprising the empty one (as we can ascertain the larger graph has no matches, either).

8 Experimental results

Settings. We implemented our algorithms in Java 1.8, and relied on Postgres for storing the integer-encoded triple table. The computer had an Intel Xeon CPU E5-2640 v4 @2.40GHz and 124 GB of memory; we used PostgreSQL v9.6 with 30 GB of shared buffers and 640 MB working memory. The JVM had 70 GB of RAM.

Datasets. We summarized *synthetic* RDF graphs: BSBM [3], LUBM [12] and WatDiv (`http://dsg.uwaterloo.ca/watdiv/`) benchmark datasets, as well as *real-life* graphs: the John Peel BBC Radio Sessions (`http://dbtune.org/bbc/peel/`), French INSEE Geo geographic data (`http://rdf.insee.fr/geo/index.html`), DBLP, the DBpedia Person dataset (`http://wiki.dbpedia.org/Downloads2015-04`), Linked Clinical Trials (LCT) [24] and LinkedMDB [14]. Their sizes (in triples) are denoted $|\mathtt{G}|$ in Figure 17.

Summary size. We define the **compression factor** $cf_{\equiv}$ as $|\mathtt{G}|/|\mathtt{G}_{/\equiv}|$, the ratio between the numbers of triples in G and in the summary. (Since schema triples are the same, compression comes from the representation of G data triples through $\mathtt{G}_{/\equiv}$ triples.) Figure 17 shows the size of various summary and the resulting compression factor underneath; cf values higher than 10^4 appear in bold font. The compression factor is at least 300, more usually above 10^3, and up to $2 \cdot 10^6$ for DBLP dataset; $\mathtt{G}_{/\mathtt{W}}$ is typically smallest, while the $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$ may be up to one order of magnitude larger. The LUBM and BSBM 1M and 100M triples synthetic datasets offer an interesting contrast. LUBM compresses rather well; the number of data edges does not increase as the data grows. This enables higher cf values from $4 \cdot 10^3$ to $7 \cdot 10^5$. BSBM is harder to compress, which correlates with its higher number of distinct data properties (642 to 8082), which also grows with the data size, classes grow from 159 to 2019, and schema triples from 150 to 2010). BSBM compression factors range from 327 to 7775, more modest but still very significant. *The compression factor increases with the data size* in both cases, highlighting summarization's particular interest for large graphs. For LinkedMDB, compression factors higher than 10^5 contrast with **the compression factor of 1.42** reported in [19] for the **`fb`-bisimulation summary**.

A more detailed look at BSBM summarization is provided in Figure 18, which shows node and edge counts for summaries of BSBM graphs of various sizes. The horizontal axis is labeled in input triples; the vertical axis is in log scale. For reference, we plot the number of class nodes of G next to the summary data nodes, and the number of schema edges next to the edge counts. The figure shows that $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{S}}$ strongly summarize graph structure, while $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$ which isolate typed from untyped data nodes lead to larger, more complex summaries.

Summarization time. Figure 19 shows the average time (in seconds, 3 executions) to build summaries; they go up to 800 seconds for 100M triples. Times for the LUBM and BSBM synthetic data are plotted (both axis are logarithmic, the horizontal axis labeled in $|\mathtt{G}|$ triples). The graphs show that summarization time grows almost linearly, and the time to build $\mathtt{G}_{/\mathtt{W}}$ grows fastest. This is because building $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{S}}$ maximizes the number of nodes for which cliques are analyzed and fused; in contrast, when building $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$, *the cliques of typed data nodes are never constructed (directly or indirectly)*. We also see that $\mathtt{G}_{/\mathtt{S}}$ and $\mathtt{G}_{/\mathtt{TS}}$ are oftentimes more expensive to build than $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{TW}}$; this is because of the extra effort needed to build the cliques, while $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{TW}}$ use a more direct, faster method. We find summarization times acceptable, also considering that this is an off-line task.

Building complete summaries. Figure 20 shows the time to build the *complete* weak and strong summaries: by saturating and then summarizing ($(\mathtt{G}^{\infty})_{/\mathtt{W}}$ and $(\mathtt{G}^{\infty})_{/\mathtt{S}}$); and separately, by summarizing G first, then saturating and then summarizing again, taking advantage of Theorems 3 and 4. For saturation, we used the algorithm described in [10]. $\times_{\mathtt{W}}$, respectively $\times_{\mathtt{S}}$ denote the speed-up factors achieved thanks to the completeness theorems. Figure 20 shows that computing $(\mathtt{G}^{\infty})_{/\mathtt{W}}$, $(\mathtt{G}^{\infty})_{/\mathtt{S}}$ by the indirect method given by the completeness theorems is much faster; indeed, this method reduces the size of the graph in

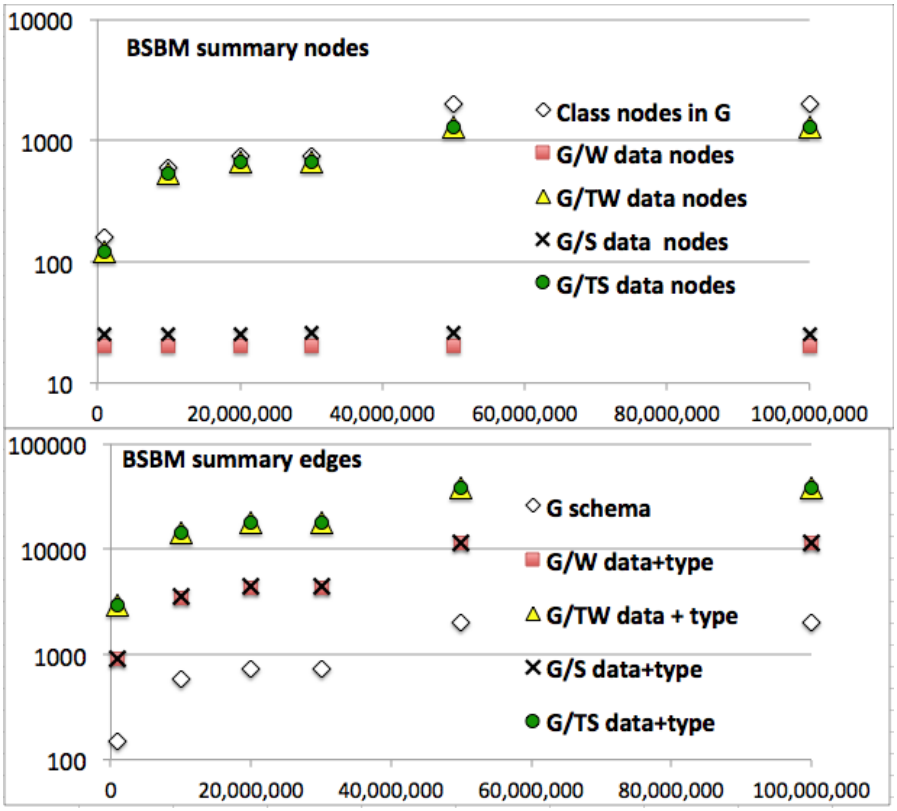


Figure 18: Summary node counts (top) and edge counts (bottom) for BSBM graphs.

the first place, making subsequent operations more efficient.

Dataset	Weak	Strong	Typed Weak	Typed Strong
John Peel	$AL_2 = 81$ $AL_3 = 96$	$AL_2 = 29$ $AL_3 = 53$	$AL_2 = 15$ $\mathbf{AL_3 = 30}$	$\mathbf{AL_2 = 14}$ $\mathbf{AL_3 = 30}$
INSEE Geo	$AL_2 = 41$ $AL_3 = 68$	$AL_2 = 36$ $AL_3 = 64$	$\mathbf{AL_2 = 1}$ $\mathbf{AL_3 = 5}$	$\mathbf{AL_2 = 1}$ $\mathbf{AL_3 = 5}$
LUBM 1M	$AL_2 = 61$ $AL_3 = 83$	$AL_2 = 30$ $AL_3 = 53$	$\mathbf{AL_2 = 0}$ $\mathbf{AL_3 = 0}$	$\mathbf{AL_2 = 0}$ $\mathbf{AL_3 = 0}$
DBpedia People	$\mathbf{AL_2 = 0}$ $AL_3 = 2$	$\mathbf{AL_2 = 0}$ $AL_3 = 4$	$\mathbf{AL_2 = 0}$ $AL_3 = 1$	$\mathbf{AL_2 = 0}$ $\mathbf{AL_3 = 0}$
DBLP	$AL_2 = 48$	$\mathbf{AL_2 = 38}$	$AL_2 = 44$	$AL_2 = 39$
BSBM 1M	$AL_2 = 78$	$\mathbf{AL_2 = 0}$	$AL_2 = 65$	$\mathbf{AL_2 = 0}$

Table 2: Bounded accuracy loss values.

Accuracy loss. Table 2 shows the accuracy loss AL_k (in %, rounded to the lower closest integer) for $k \in \{2, 3\}$ on various RDF graphs; the lowest value(s) on each row are in boldface. $\mathtt{G}_{/\mathtt{W}}$ and $\mathtt{G}_{/\mathtt{S}}$ have comparable accuracy losses, while on the other hand $\mathtt{G}_{/\mathtt{TW}}$ and $\mathtt{G}_{/\mathtt{TS}}$ are significantly more accurate. The difference is dramatic for LUBM and DBpedia, where TW and TS have 0 accuracy loss for $k = 2$, just as the DBpedia TS summary for $k = 3$, while the DBpedia TW summary has a minimum loss of 1% for $k = 3$.

Experiment conclusion. Our experiments have shown that our summaries can be built efficiently, and they reduce graph sizes by factors hundreds up to $2 \cdot 10^6$. This confirms their power of compactly

Dataset	$\|G\|$	$G_{/W}$	$G_{/S}$	$G_{/TW}$	$G_{/TS}$
John Peel	271k	0.33	1.37	0.43	0.98
INSEE Geo	369k	0.58	1.65	0.55	1.50
DBpedia	8M	9.73	25.85	6.82	21.80
DBLP	150M	434.88	636.56	175.87	12412.49
LinkedCT	42M	74.55	314.75	88.73	252.38
LinkedMDB	6.14M	1.12	3.49	0.90	3.06
LUBM	1M	1.34	3.30	1.33	3.20
LUBM	10M	13.92	28.27	10.42	31.14
LUBM	100M	221.62	403.97	132.89	457.63
WatDiv	10M	14.78	22.77	8.33	21.02

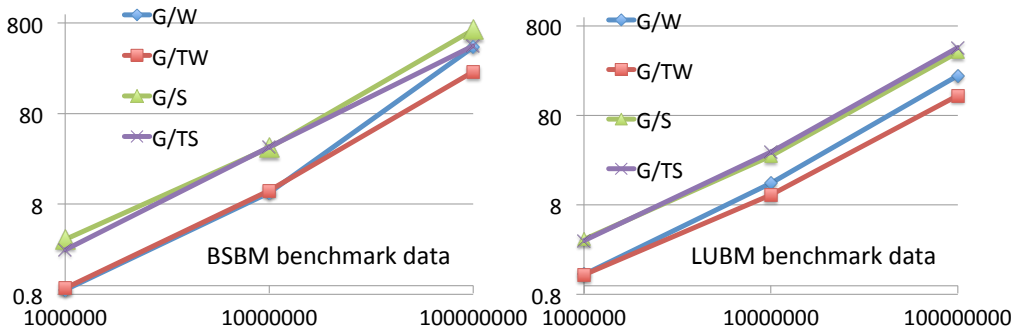


Figure 19: Summarization times (s) for various RDF datasets

Dataset	$(G^{\infty})_{/W}$	$((G_{/W})^{\infty})_{/W}$	$\times_W$	$(G^{\infty})_{/S}$	$((G_{/S})^{\infty})_{/S}$	$\times_S$
INSEE Geo	49.17	1.25	39.33	45.94	1.70	27.02
LUBM 1M	20.77	1.42	14.62	21.27	2.72	7.8
LUBM 10M	220.36	12.31	17.90	219.64	34.04	6.45
BSBM 1M	22.88	1.42	16.11	16.33	3.25	5.02
BSBM 10M	301.98	134.06	2.25	213.73	71.13	3.01
DBLP	465.09	14.83	31.36	434.87	43.28	10.04

Figure 20: Times (s) to generate semantically complete summaries.

representing RDF graphs. $G_{/S}$ and $G_{/TS}$ take longer to compute than $G_{/W}$ and $G_{/TW}$ since the former actually compute cliques, while the latter rely instead on Property 6 for a more efficient method. Weak summaries are often less accurate, but the opposite also occurs, e.g., on the DBPedia person dataset; weak summaries compress more (and thus are smaller). If summary size or building time are a concern, we recommend using $G_{/W}$; otherwise, G_S may give finer-granularity information.

9 Related Work

Graph summarization has a long history and many applications. We survey general graph summarization works (with no specific focus on RDF graphs) in Section 9.1, then discuss RDF-specific summarization proposals in Section 9.2.

9.1 Non-RDF graph summaries

Such summaries ignore graph semantics, thus summary completeness, a main focus of our work, is not in general guaranteed.

Several works built graph summaries as a support for graph indexing. In this context, for each summary node, we store the IDs of the original graph ndoes represented by the summary node; this set is typically called the extent. Given a query, evaluating the query on the summary and then using the summary node extents allows to obtain the query results.

Dataguides [11], and in particular the so-called strong dataguide, were among the first proposals for structural summaries (also called a-posteriori schemas) of RDF graphs. Dataguides do not compare directly with our approach, since they are not quotient graphs: in particular, a graph node may be represented by more than one dataguide node, which is impossible in a quotient-based summary. A dataguide may be larger than the original graph, and its construction has worst-case exponential time complexity.

[25, 6, 16, 8] build quotient summaries based on (sometimes unidirectional, sometimes k-bounded) bisimulation; we have discussed the relationship between our and bisimulation summaries in Section 6. The complexity of building a bisimulation summary is typically $O(\mathtt{E} \cdot \log(\mathtt{N}))$, where $\mathtt{E}$ is the number of edges and $\mathtt{N}$ is the number of nodes in $\mathtt{G}$.

[24] explores efficient answering of neighbor queries over a compressed social network graph, while minimizing the number of bits per edge required for storage, by relying on covers of an input graph. Thus, the focus is entirely different than ours, which is efficiently representing all RBGP queries from an RDF graph $\mathtt{G}$ into its summaries, while preserving the semantics of $\mathtt{G}$, by grouping similar nodes based on equivalence classes.

[9] considers reachability and graph pattern queries on labeled graphs, and builds *answer-preserving summaries* for such queries; this property subsumes accuracy and representativeness. However, their query semantics is not based on graph homomorphisms, which underlie SPARQL and the dialects we use, but on *bounded graph simulation*. Under this semantics, answering a query becomes P (instead of NP), at the price of not preserving the query structure (i.e., joins).

[32] proposes the SNAP (Summarization on Grouping Nodes on Attributes and Pairwise Relationships) technique which, given a set of properties the user is interested in, produces a summary that groups the interesting nodes in the incoming RDF graph, based on their roles with respect to the user-specified properties. (The authors consider graphs where nodes have "attributes" and "relationships"; this can be seen as a restriction of RDF where we do not consider types nor schema information.) SNAP partitions nodes into classes that may end up being too many and too small (two few nodes will be equivalent). For more flexibility, the authors also propose k-SNAP where the similarity requirement between nodes of the same class is relaxed and the summary is computed taking into account a space budget (i.e., , a maximum summary size). Thus, while SNAP is based on strict graph relations, k-SNAP mixes in a quantitative component (with a threshold of 50% proposed by the authors at some point etc.). Our work is different in that we request no user input and propose complete, logical-oriented summaries.

[5] introduces an aggregation framework for OLAP operations on labeled graphs. The authors assume available an OLAP-style set of dimension with their hierarchies, and measures, and based on them they define a "graph cube" and investigate efficient methods for computing it. The authors also frame graph topological information as dimensions, and use them for graph aggregation. This is quite different from our framework, which is based on logical representation of structure and semantics.

[4] considers node- and edge-labeled graphs and focuses on building a set of randomized summaries, so that one can mine the summary set instead of the original graph for better performance, with guaranteed bounds on the information loss thus incurred. Semantics and properties such as those of our summaries are out of their scope.

[39] seeks to identify frequent subgraphs and store extent information from those, so as to answer queries asking for the respective part of the graph. This approach by design does not reflect all data and

is based on numeric information, as opposed to our summaries. [40] is very similar, but considers trees instead of graphs.

A graph *core* C for a given graph G is a graph such that an isomorphism exists between G and C, and C is the smallest graph with this property[6]. Our summaries are not cores of G, since we cannot guarantee a homomorphism from any of them to the G. In exchange, all of our summaries can be built in polynomial time in the size of G, while computing the core is much harder.

A separate brand of works is interested in computing approximate summaries of graphs, to be used to efficiently answer queries with an acceptable level of error [26, 18]. The focus is on graph compression while preserving bounded-error query answering and/or the ability to fully reconstruct the graph from the summary, with the help of so-called "corrections" (edges to add to or remove from the "expansion" of the summary into the regular part of the graph it derives from). [22] focuses on summarizing undirected, node-labeled graphs; nodes are partitioned, while edge information is preserved under the form of the number of edges within each partition set, and number of edges between every two such sets. These works contrast with the full representativity, accuracy and completeness we aim at.

[23] surveys many other quantitative, mining-oriented graph sampling and summarization methods.

9.2 RDF graph summaries

Several works have more recently target the construction of RDF graph summaries. Still, most ignore RDF semantics, or make limitating assumptions on the input RDF graph, while we target general graphs, with or without schema statements. Overall, our work is the first to consider the issue of building complete RDF summaries, and to provide a method for doing it and a sufficient completeness condition.

[36] builds a tree-structured index of an RDF graph by picking some "centroid" RDF graph nodes and organizing information about the graph in a tree; if one views the index as a summary, it is not a quotient, and since inference is not considered, it is not complete, either.

[28] introduces a *simulation* RDF quotient based on *triple (not node) equivalence*; the summary is not an RDF graph. (Intuitively, *simulation* is a unidirectional "half" of bisimulation, e.g., to define FW RDF similarity, remove item (ii) from Definition 9). For compactness, they bound the simulation, for small k values, e.g., 2; this enables compression factors of about 10^4. They use summaries to reduce query join effort, but do not consider the possible schema, nor summary completeness.

[30] considers the problem of efficiently building a quotient summary of RDF graphs based on the bisimulation relationship between nodes. The authors present several algorithms, in particular based on MapReduce, but do not reserve any special treatment to RDF class and/or property nodes; this would allow in particular to summarize different classes into a single node, e.g., if the classes only participate to triples as subjects of the subclassOf property. Such schema triple summarization compromises completeness, as it is not possible to build the summary of $\mathtt{G}^\infty$ out of a summary that does not preserve the schema of G.

[33] has as main goal to answer queries over RDF graphs and proposes to help this based on a structure index, which is defined as a bisimulation quotient; the authors show that the summary is representative. Further, the authors study limited versions of the bisimulation quotient by considering: only forward bisimulation; only backward bisimulation; only neighborhoods of a certain length. The authors do not consider graph semantics, nor completeness.

[29] presents SAP HANA's approach for working with graph data. They consider a restricted RDF model where every node has exactly one type; to simplify the experience of users querying graphs, they propose that users specify some simple summarization rules (summarization here viewed as aggregation) together with their query, rules whcih the system applies to group the query results. Beyond its restricted data model and lack of support for RDFS inference, the work has clearly a very different focus and does not aim to produce an interesting representation of the whole graph as we do.

[6] https://en.wikipedia.org/wiki/Core_(graph_theory)

[21] introduces an RDF summary graph which they use to support keyword search in RDF graphs. They restrict RDF graphs so that (i) one resource can have at most one type, and (ii) same-type resources have similar sets of data properties. These restrictions make the summarization problem significantly different from ours, as we do not make such restrictive assumptions. The authors do not consider inference either.

[13] applies a graph clustering algorithm (in particular METIS[7]) to determine the best partitioning to use in order to store an RDF graph in a distributed system. The partitioning criterium is quantitative (METIS seeks to minimize edges across partitions, thus strong connections will tend to keep nodes together). Such partitioning is not a quotient graph and properties such as accuracy and completeness cannot be guaranteed.

[7] also considers summaries defined as quotients based on bisimulation, disregarding the role played by schema triples, and proposes a highly parallel implementation based on the GraphChi framework. They show how to use the summary as a support for query evaluation: incoming SPARQL queries are evaluated on the summary and then the results on the summary are transformed in results on the original graph by exploring the extents of summary nodes.

[17] uses a density-based clustering algorithm to identify nodes with similar structure in an RDF graph; the goal is to propose an RDF type for each cluster of resources and identify relations between these types. Thus, the authors do not consider RDF schema information that the graph may already have, nor are they interested in accuracy, completeness etc. for the result of their clustering.

[35] aims to summarize an RDF Schema based also on the frequency of representation of various schema features in the data graph. The approach makes several limitative assumptions on the RDF data model, e.g., that each resource has exactly one type etc. While our summaries do not reduce the size of an RDF Schema, they allow to significantly compress general RDF graphs (with or without schema information) and enjoy several interesting properties.

[31] focus on summarizing RDF graphs which may have subtype and subproperty triples, but they do not consider domain and range, and instead rely on *abstract knowledge patterns*, expressing something related but different: the pattern $\mathtt{c}_1\ \mathtt{p}\ \mathtt{c}_2$ states that a dataset holds at least one resource of type $\mathtt{c}_1$ having the property $\mathtt{p}$ whose value is a resource of type $\mathtt{c}_2$. (In contrast, recall that the domain constraint $\mathtt{p} \hookleftarrow_d \mathtt{c}_1$ states that any resource having property $\mathtt{p}$ is also of class $\mathtt{c}_1$.) A summary is a set of types, a set of AKPs, and some statistics; the authors are only concerned with the minimality of their summaries and do not consider properties such as accuracy, representativeness, completeness etc.

10 Conclusion

We studied the construction of *complete* summaries of RDF data, accounting for all its structure and semantics; in particular, we focused on *quotient* graph summaries, widely studied in the literature. We introduced four novel summaries, two of which (as well as bisimulation-based ones) are complete. All our summaries are representative and accurate w.r.t. a large and useful dialect of SPARQL; they can be built efficiently and achieve RDF graph compression factors of 10^2 to $2 \cdot 10^6$; the highest values are reached for real-life datasets. Future work will improve scalability by leveraging a massively parallel platform such as Spark. We are also interested in devising advanced visualizations of our RDF summaries.

References

[1] RDF Summary project website: https://team.inria.fr/cedar/projects/rdfsummary, 2017.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[7] `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`

[4] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han. Mining graph patterns efficiently via randomized summaries. *PVLDB*, 2(1), 2009.

[5] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: towards online analytical processing on graphs. In *ICDM*, 2008.

[6] Q. Chen, A. Lim, and K. W. Ong. $D(K)$-index: An adaptive structural summary for graph-structured data. In *SIGMOD*, 2003.

[7] M. P. Consens, V. Fionda, S. Khatchadourian, and G. Pirrò. S+EPPs: Construct and explore bisimulation summaries + optimize navigational queries; all on existing SPARQL systems (demonstration). *PVLDB*, 8(12), 2015.

[8] M. P. Consens, R. J. Miller, F. Rizzolo, and A. A. Vaisman. Exploring XML web collections with DescribeX. *ACM TWeb*, 4(3), 2010.

[9] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *SIGMOD*, 2012.

[10] F. Goasdoué, I. Manolescu, and A. Roatiş. Efficient query answering against dynamic RDF databases. In *EDBT*, 2013.

[11] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, 1997.

[12] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3), 2005.

[13] S. Gurajada, S. Seufert, I. Miliaraki, and M. Theobald. Using graph summarization for join-ahead pruning in a distributed RDF engine. In *SWIM*, 2014.

[14] O. Hassanzadeh and M. P. Consens. Linked movie data base. In *LDOW*, 2009.

[15] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

[16] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering indexes for branching path queries. In *SIGMOD*, 2002.

[17] K. Kellou-Menouer and Z. Kedad. Schema discovery in RDF data sources. In *ER*, 2015.

[18] K. Khan, W. Nawaz, and Y. Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.

[19] S. Khatchadourian and M. P. Consens. Constructing bisimulation summaries on a multi-core graph processing framework. In *GRADES*, 2015.

[20] D. Lanti, M. Rezk, G. Xiao, and D. Calvanese. The NPD benchmark: Reality check for OBDA systems. In *EDBT*, 2015.

[21] W. Le, F. Li, A. Kementsietsidis, and S. Duan. Scalable keyword search on large RDF data. *IEEE TKDE*, 26(11), 2014.

[22] K. LeFevre and E. Terzi. GraSS: Graph structure summarization. In *SDM*, 2010.

[23] S.-D. Lin, M.-Y. Yeh, and C.-T. Li. Sampling and summarization for social networks (tutorial), 2013.

[24] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *SIGKDD*, 2010.

[25] T. Milo and D. Suciu. Index structures for path expressions. In *ICDT*, 1999.

[26] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

[27] M. Palmonari, A. Rula, R. Porrini, A. Maurino, B. Spahiu, and V. Ferme. ABSTAT: linked data summaries with abstraction and statistics. In *ESWC Workshops*, 2015.

[28] F. Picalausa, Y. Luo, G. H. L. Fletcher, J. Hidders, and S. Vansummeren. A structural approach to indexing triples. In *ESWC*, 2012.

[29] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner. SynopSys: large graph analytics in the SAP HANA database through summarization. In *GRADES*, 2013.

[30] A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale bisimulation of RDF graphs. In *SWIM*, 2013.

[31] B. Spahiu, R. Porrini, M. Palmonari, A. Rula, and A. Maurino. ABSTAT: ontology-driven linked data summaries with pattern minimalization. In *SumPre*, 2016.

[32] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, 2008.

[33] T. Tran, G. Ladwig, and S. Rudolph. Managing structured and semistructured RDF data using structure indexes. *IEEE TKDE*, 25(9), 2013.

[34] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, 2009.

[35] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis. RDF digest: Efficient summarization of RDF/S KBs. In *ESWC*, 2015.

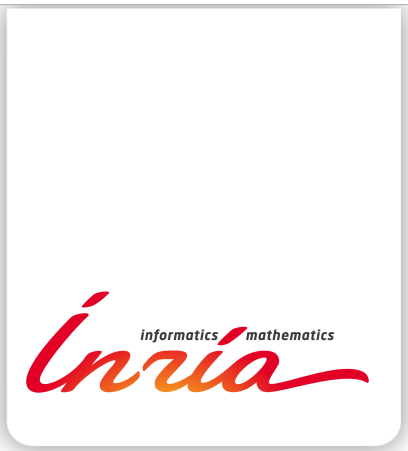
[36] O. Udrea, A. Pugliese, and V. S. Subrahmanian. GRIN: A graph based RDF index. In *AAAI*, 2007.

[37] W3C. Resource description framework. http://www.w3.org/RDF/.

[38] http://gromgull.net/blog/2010/09/btc2010-basic-stats, 2010.

[39] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, 2004.

[40] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: Tree + delta >= graph. In *VLDB*, 2007.



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

1 rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing
Campus de l'École Polytechnique
91120 Palaiseau

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399