RDF as Graph-Based, Diagrammatic Logic: Syntax, Semantics, and Calculus

Frithjof Dau[1] and Jonathan Hayes[2]

[1] Dept. of Mathematics, Darmstadt University of Technology, Germany
[2] Center for Web Research, Dept. of Computer Science, Universidad de Chile
dau@mathematik.tu-darmstadt.de jonathan.hayes@gmx.de

Abstract. The Resource Description Framework (RDF) is a standard for representing information on the Web to make it machine-understandable and -processable. This paper explores a complementary view in which we investigate RDF as a logic system for humans. RDF is developed as a graph-based logic system, i.e., a *diagrammatic reasoning system*. Moreover, a sound and complete set of transformation rules is provided which act on the diagrammatic representations of RDF.

1 Introduction: RDF and Graphs

The Resource Description Framework (RDF) is a standard for representing information on the Web. The information is encoded as statements, which are subject-predicate-object triples of information resources. RDF is a data format designed to enable semantic interoperability between computers, and the largest part of its specification provides the means for this in defining the core vocabulary, an XML serialization, inferencing, etc. However, RDF's triple syntax (and much less RDF/XML code) is not easily read and understood by humans.

As RDF was developed to be a machine-accessible metadata format, why care for representations for humans? Tim Berners-Lee says in [2] that 'the Web will reach its full potential when it becomes an environment where data can be shared and processed by automated tools *as well as by people*.' (emphazising by Dau/Hayes). Particularly for RDF, there are several reasons to investigate a human-centered approach. First, there is the natural need of RDF authors and tool developers to revise RDF specifications they created. Second, it appears that RDF becomes attractive as a general data format where heterogenous information must be integrated (e.g., [3]), which implies the need of visualization for the same reasons as in the domains the data originates from. Third, RDF provides the possibility of reasoning on data—however, it is not satisfactory to leave logic entirely to computers: humans must be able to follow machine-generated proofs.

An approach to make RDF accessible for humans is given in the RDF specification: the *RDF Primer* [10] introduces a visualization of RDF models by representing statement subjects and objects by nodes, which are connected by predicate-labeled directed edges which correspond to statements. However, this is only an informal representation, because it does not account for peculiarities

of RDF such as the fact that a statement predicate may appear as the subject of another statement. A critique of this default approach is given in [5, 6], along with an investigation of alternate RDF graph representations. While the *RDF Bipartite Graphs* presented there satisfy the requirement of a formal graph representation of RDF, issues such as entailment and diagrammatic reasoning have not been addressed. This is the goal of this paper.

RDF's use is not limited to only encode information, it supports also reasoning. [7] provides semantics for RDF statements based on model theory, and complete systems of inference rules specify how RDF models may evolve. In this understanding, RDF is a logic system with a syntax, a semantics based on mathematical model theory, and a calculus. The main difference between RDF and symbolic logic systems is the diagrammatic potential of RDF. There are already diagrammatic logic systems: for example, *Semantic Networks* and *Topic Maps*, which are used for knowledge representation and processing in artificial intelligence, or *Existential Graphs* as introduced by C. S. Peirce. The most important DRS, particularly for RDF, are J. Sowa's *Conceptual Graphs (CG)* (see [11, 13]), which are a system for expressing meaning in a "logically precise, human-readable, and computationally tractable" form [12]. In fact, the *RDF Primer* states that RDF draws upon various ideas from knowledge representation, including CGs—for example, T. Berners-Lee studied the relation between RDF and CGs [1]. Unfortunately, CGs lack preciseness from a logical and mathematical view: a comprehensive critique can be found in the PhD thesis of the first author [4].

The goal of this paper is to develop RDF as a Diagrammatic Reasoning System (*DRS*), to provide the means for both human-accessible and mathematically precise RDF representation and reasoning on RDF models. We will call this framework *RDF/DRS*. Comparing to the RDF's XML serialization, one could emphasize the antagonism:

1. RDF/DRS is a kind of front-end of RDF for humans
2. RDF/XML is a kind of back-end for storing RDF by machines.

As already stated, for elaborating RDF as a DRS, it is not sufficient to consider only the classic, "static" visualization of RDF data in form of graphs. We also have to consider a visual representation of transformations of RDF graphs—a sort of "dynamic" component of visualization. The most important class of transformations for RDF graphs are those which derive an *entailed RDF graph* (see [7]) from another.

Although [6] provides—by means of *RDF Bipartite Graphs*—a mathematically sound graph representation, the study of a graph-syntactical calculus conducted in this paper has a different scope. While [6] argued for the importance of an unambiguous and redundancy-free graph foundation of RDF for use as a data structure in software and to ensure the applicability of graph theory's results, the use of RDF as DRS as we present it here is geared towards a human audience. Particularly, for humans it is convenient to allow different kinds of redundancy in the representations, because this enables the graphs to carry some

sort of 'rhetorical structure' (nonetheless, the triple notation for graphs allows redundancies as well in that the same resource may appear repeatedly in various triples). This will be discussed in more detail later on.

In this paper we concentrate on the visualization of RDF graph transformations. The goal of the formalism to be presented here is to make the transformations understandable by breaking them up into a sequence of elementary steps, very much in the way formulas are subsequently derived in mathematical proofs.

Let us consider a simple example of entailment between two graphs. (Simple) entailment is defined in [7] as follows:

> Suppose that M is a mapping from a set of blank nodes to some set of literals, blank nodes and URI references; then any graph obtained from a graph G by replacing some or all of the blank nodes N in G by M(N) is an *instance* of G.
> *Interpolation Lemma.* S entails a graph E if and only if a subset of S is an instance of E.

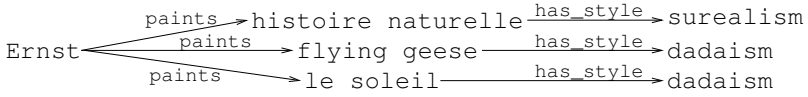
Consider the following example: The RDF graph $\mathcal{G}_1$ consisting of the triples

```
(Ernst, paints, histoire_naturelle), (Ernst, paints, flying_geese),
(Ernst, paints, le_soleil), (histoire_naturelle, has_style, surrealism),
(flying_geese, has_style, dadaism), (le_soleil, has_style, dadaism)
```

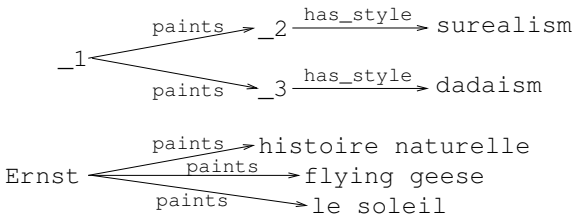
entails the following RDF graph[1] $\mathcal{G}_2$:

```
(Ernst, paints, histoire_naturelle), (Ernst, paints, flying_geese),
(Ernst, paints, le_soleil),   (_1, paints, _2),   (_1, paints, _3),
(_2, has_style, dadaism), (_3, has_style, surrealism)
```

By only considering the entailment definition in [7] it is difficult for humans to decide whether an RDF graph entails the other or not: it is arduous to check all possible instance-describing mappings from $\mathcal{G}_2$ to $\mathcal{G}_1$ to see whether an instance of $\mathcal{G}_2$ is a subgraph of $\mathcal{G}_1$. Instead, using a graphical representation it is much easier to apprehend the entailment between these graphs because one can see that $\mathcal{G}_2$ expresses less information than $\mathcal{G}_1$. This shall be exemplified. The first graph can be represented as follows:



The second graph is:



[1] Note that the entailed graph has more triples than the graph we started with.

Now we see that $\mathcal{G}_2$ is composed of two copies of $\mathcal{G}_1$ such that in each copy, some specific information of $\mathcal{G}_1$ is removed. To be more precise: The upper part of $\mathcal{G}_2$ is a copy of $\mathcal{G}_1$ where the information about the painting 'flying geese' is removed, and where some names are replaced by blanks. Thus, it is entailed by $\mathcal{G}_1$, as it contains less information than $\mathcal{G}_1$. Similar, the lower part of $\mathcal{G}_2$ is entailed by $\mathcal{G}_1$ as well, as the information about the paint styles has been removed. Thus, both parts of $\mathcal{G}_2$ can be understood as copies of the first graph, where some information is removed, hence $\mathcal{G}_2$ is by entailed $\mathcal{G}_1$.

The transformation rules which will be provided in this paper are based on the ideas presented in the example. We will have rules which allow to

1. change a graph without changing the represented information (like making a copy of a statement or a subgraph), or
2. remove small parts of the information in a graph (like erasing a statement or replacing names by blanks).

This transformation rules will be described mathematically, but having our goal in mind to develop RDF as a DRS, they are designed to be performed on the graphical representations of RDF graphs. In short: They are diagrammatic transformation rules. To provide an example for the rules, we will later show how $\mathcal{G}_2$ can be syntactically be derived from $\mathcal{G}_1$ with these rules.

In the following sections the outlined elaboration of RDF/DRS is carried out. In Sec. 2, the syntax is provided, and some normal forms of RDF graphs are defined. In Sec. 3, the semantics of RDF/DRS, which is mainly the semantics provided in [7] or [8], is presented. In Sec. 4, the calculus, i.e., the transformation rules, are provided, and it is shown that the calculus is sound and complete. Finally, in Sec. 5, an outlook for further research is given. In this paper, due to space limitations, no proofs are provided. The proofs can be found in an extended version of this paper (see the remark after the bibliography).

2 Syntax

Like in every logic system, we first have to fix the syntax of the system, starting with the vocabulary we will use.

Definition 1 (Vocabulary). *Let* Blanks *be a countably infinite set. The elements of* Blanks *are called* blanks*. A triple* $\mathsf{Voc} := (\mathsf{URIs}, \mathsf{TypedLit}, \mathsf{PlainLit})$*, consisting of three sets* URIs*,* $\mathsf{TypedLit}, \mathsf{PlainLit}$ *which are pairwise disjoint and disjoint from* Blanks *is called* vocabulary*. The members of the three sets are called* universal resource identifyers*,* typed literals*, and* plain literals*, resp. We set* $\mathsf{Lit} := \mathsf{TypedLit} \cup \mathsf{PlainLit}$*. The elements of* Lit *are called* literals*. The elements of* $\mathsf{URIs} \cup \mathsf{TypedLit} \cup \mathsf{PlainLit}$ *are called* names*.*

The well-known triple notation of RDF is defined as follows:

Definition 2 (RDF graph, Triple Notation). *An* RDF graph over $\mathsf{Voc} := (\mathsf{URIs}, \mathsf{TypedLit}, \mathsf{PlainLit})$ in triple notation *is a set of triples* (s, p, o) *with*

$$(s, p, o) \in (\mathsf{URIs} \cup \mathsf{Blanks}) \times \mathsf{URIs} \times (\mathsf{URIs} \cup \mathsf{Blanks} \cup \mathsf{Lit})$$

for each triple. Given a triple (s, p, o)*, we call* s *the* subject*,* p *the* predicate *and* o *the* object *of the triple.*

RDF graphs have a peculiarity which does usually not occur in common mathematical logic: Some entities can occur as predicates in some statements and as subjects or objects in other statements. As the sets of properties of an RDF graph is not disjoint from its set of subjects and objects, it is not possible to transform an RDF graph in triple notation to an ordinary mathematical graph (V, E) by assigning to each object or subject a node of the graph and assigning to each predicate an edge which links an subject node to an object node. In the informal graphical representations for RDF graphs, there are different "workarounds" to cope with this problem (for example, edges which link vertices and edges, or the appearance of the same name both on vertices and edges). A comprehensive discussion of this problem can be found in [5], the diploma thesis of one of the authors.

A reasonable graph formalization of RDF graphs is to assign to each *subject, property and object* a node in a mathematical graph, and to assign to each *statement*, i.e. a triple (s, p, o), a hyper-edge which links the nodes of s, p, and o. Each RDF graph in triple notation can be translated to a graph formalization in this sense, thus the graph formalizations can be understood as explicit representations of RDF graphs in triple notation as mathematical graphs. This is the underlying understanding in [5], where one possible translation of RDF graphs in triple notation to mathematical graphs is mathematically elaborated. In this work, in contrast to [5], we understand and investigate the graph formalizations of RDF graphs as structures on its own.

In the next definition, the graph formalization of RDF graphs are defined.

Definition 3 (Graph Formalization of RDF graphs). *A structure* $\mathcal{G} := (V, E, \nu, \kappa)$ *is called* graph formalization of an RDF graph over $\mathsf{Voc} := (\mathsf{URIs}, \mathsf{TypedLit}, \mathsf{PlainLit})$*, iff*[2]

- V *and* E *are disjoint, finite sets whose elements are called* vertices *and* edges*.*
- $\nu : E \to V^3$ *is a mapping such that each vertex* v *occurs in a triple* $\nu(e)$ *for an edge (i.e., each vertex is incident with at least one edge)*
- $\kappa : V \to \mathsf{Voc}$ *is a mapping such that for each* $e \in E$*, we have*

$$\kappa(e\big|_1) \in (\mathsf{URIs} \cup \mathsf{Blanks}), \quad \kappa(e\big|_2) \in \mathsf{URIs}, \quad \kappa(e\big|_3) \in (\mathsf{URIs} \cup \mathsf{Blanks} \cup \mathsf{Lit})$$

For edges e*, we will often write* $e = (v_1, v_2, v_3)$ *instead of* $\nu(e) = (v_1, v_2, v_3)$*, and we will often denote the* i*th vertex* v_i*,* $i = 1, 2, 3$*, with* $e\big|_i$*.*

For $v \in V$ *let* $E_v := \{e \in E \mid \exists\, i.\nu(e)\big|_i = v\}$ *be the set of all edges which are incident with* v*. Analogously, for* $e \in E$ *let* $V_e := \{v \in V \mid \exists\, i.\nu(e)\big|_i = v\}$*.*

The set $\{bl \in \mathsf{Blanks} \mid$ *there exists a* $v \in V$ *with* $\kappa_b(v) = bl\}$ *is called the* set of blanks of $\mathcal{G}$*. The set of URIs of* $\mathcal{G}$*, etc. is defined analogously.*

[2] As usual in mathematics, we use 'iff' for 'if and only if'.

In the following, we will mainly work with the graph formalization of RDF graphs. As it will be clear from the context when RDF graphs in triple notation or graph formalizations of RDF graphs are used we will omit the phrases "graph formalization" or "triple notation" and simply speak of 'RDF graphs'.

We will diagrammatically represent the graph formalizations of RDF graphs as follows: Each vertex is drawn as an oval with the label $\kappa(v)$ inscribed in it. An edge (v_1, v_2, v_2) is drawn as small circle which is connected with three lines, labeled with s, p, and o, resp., to the ovals of v_1, v_2, and v_3.

To see an example for this convention, consider the first four axiomatic triples provided in *RDF Semantics* below. A graph formalization and its graphical representation of the four triples is given in Fig. 1.

```
(rdf:type rdf:type rdf:Property) (rdf:subject rdf:type rdf:Property)
(rdf:predicate rdf:type rdf:Property) (rdf:object rdf:type rdf:Property)
```

$$\begin{aligned}\mathcal{G} := (&\{v_1, v_2, v_3, v_4, v_5, v_6\}, \{e_1, e_2, e_3, e_4\},\\ &\{(e_1, (v_1, v_5, v_6)), (e_2, (v_2, v_5, v_6)),\\ &(e_3, (v_3, v_5, v_6)), (e_4, (v_4, v_5, v_6))\},\\ &\{(v_1, rdf:subject), (v_2, rdf:predicate),\\ &(v_3, rdf:object), (v_4, rdf:type),\\ &(v_5, rdf:type), (v_6, rdf:Property)\})\end{aligned}$$

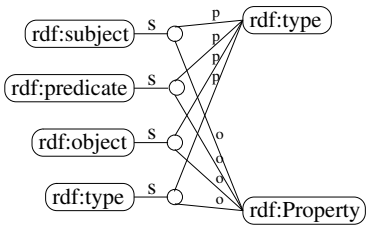


Fig. 1. Four RDF triples, a graph formalization of them and its diagrammatic representation

Next, we have to fix some elementary technical notations. A *subgraph* of an RDF graph in triple notation is simply a subset of the triples. For graph formalizations of RDF graphs, subgraphs are defined as follows:

Definition 4 (Subgraph). *Let (V, E, ν, κ) be a graph formalization of an RDF graph over* $\mathsf{Voc} := (\mathsf{URIs}, \mathsf{TypedLit}, \mathsf{PlainLit})$. *A* subgraph *of (V, E, ν, κ) is an RDF graph (V', E', ν', κ') with $V' \subseteq V$, $E' \subseteq E$, $\nu' = \nu\big|_{E'}$ and $\kappa' = \kappa\big|_{V'}$. If we have moreover $E_v \subseteq E'$ for each $v \in V'$, the subgraph is called* closed.

It is of no significance which blanks in an RDF graph are used to express existential quantification. For this reason, one considers RDF graph to be identical if they differ only in the name of the occurring blanks (this corresponds to the well-known alpha conversion of formulas in mathematical logic). In [7], this convention is fixed by defining some RDF graphs to be *equivalent*. This definition is now transferred to graph formalizations of RDF graphs.

Definition 5 (Equivalence of RDF graphs). *Let $\mathcal{G}_1 := (V_1, E_1, \nu_1, \kappa_1), \mathcal{G}_2 := (V_2, E_2, \nu_2, \kappa_2)$ be two RDF graphs. We say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent, if there are a bijective mappings $f_V : V_1 \to V_2$, $F_E : E_1 \to E_2$ and $f_{\mathsf{Blanks}} : \mathsf{Blanks} \to \mathsf{Blanks}$ with:*

– *Each* $e = (v_1, v_2, v_3) \in E_1$ *satisfies* $f_E(v_1, v_2, v_3) = (f_V(v_1), f_V(v_2), f_V(v_3))$,
– $\kappa(f_V(v)) = \kappa(v)$ *for each* $v \in V$ *with* $\kappa(v) \in \mathsf{URIs} \cup \mathsf{TypedLit} \cup \mathsf{PlainLit}$, *and*
– $\kappa(f_V(v)) = f_{\mathsf{Blanks}}(\kappa(v))$ *for each* $v \in V$ *with* $\kappa(v) \in \mathsf{Blanks}$.

Compared to RDF graphs in triple notation, their graph formalizations provide richer means to express some given amount of information. This is mainly due to the fact that a vertex in an RDF graph can be incident with several edges. Assume that we have several RDF statements where an entity occurs (as subject, predicate, or object). If we use RDF graphs in triple notation, then we will have a corresponding number of occurrences of this entity in some triples. In contrast to that, if we use graph formalizations of RDF graphs, we can *choose* the number of vertices which stand for this entity. The richer means to express information, together with the various ways a graph formalization of an RDF graph can be graphically represented, provides the possibility to *emphasize* some amount of the information in an RDF graph. In this understanding, RDF graphs provide a 'rhetorical structure'. For machine processing of RDF, this feature is, of course, unnecessary. However, this is an adequate approach for humans.

To exemplify the rhetorical structure of RDF graphs, we consider again the first four axiomatic triples from *RDF Semantics*. The RDF graph in Fig. 2 covers only the meaning of the first triple `rdf:type rdf:type rdf:Property`, but three times in different ways.

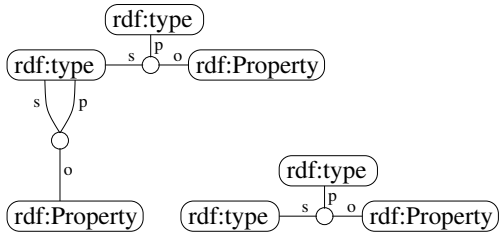


Fig. 2. An RDF graph with redundant edges

he graph of Fig. 1, as well as the two graphs in Figs. 3 and 4 are possible 'translations' of all four triples. Their different underlying structure can be used to emphasize some specific view on the represented information. The graph of Fig. 3 is fragmented into elementary graphs (i.e., graphs which carry atomic information) and is therefore easy to read, but no information is emphasized. It corresponds directly to the RDF Graph in triple notation. The graph of Fig. 1 emphasizes that `subject`, `predicate` and `object` stand in the same relation to `type` and `Property`, but it is also easy to see that `type` plays a special role, as it appears both as subject and predicate of an edge. The graph of Fig. 4 emphasizes that all four entities `subject`, `predicate`, `object` and `type` stand in the same relation to `type` and `Property`, but to see that `type` differs somehow

from the other three names one has to keep track of all labels and realize that two different vertices are labeled with `type`.

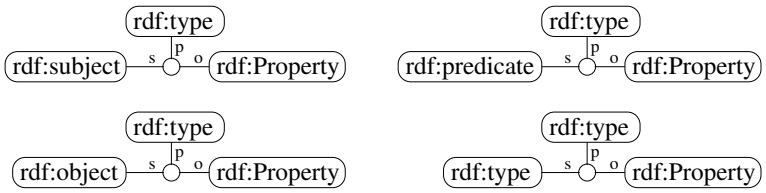


Fig. 3. A RDF graph for the four triples in anti-normal form

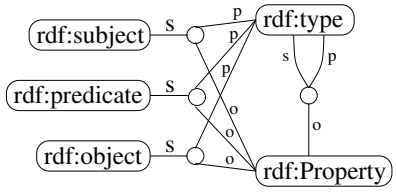


Fig. 4. A RDF graph for the four triples in normal form

The graphs in Figs. 3 and 4 are in some sense normalized. More precisely: In the first graph, each vertex is connected exactly once to an edge. Up to addition of redundant copies of edges, it has the maximum number of vertices. For the third graph, we have an opposite situation: Each vertex has a different label, i.e., this graph contains the minimal number of vertices. These two ways of normalization shall be formalized in the next definition.

Definition 6 (Normal Form and Anti-Normal Form). *Let $\mathcal{G} := (V, E, \nu, \kappa)$ be an RDF graph which has no redundant copies of edges, i.e., if $e = (v_1, v_2, v_3)$ and $f = (w_1, w_2, w_3)$ are two edges with $\kappa(v_i) = \kappa(w_i)$ for $i = 1, 2, 3$, then we have $e = f$.*

We say that $\mathcal{G}$ is in normal form*, if different vertices always have different labels, i.e., if it satisfies the following condition: For all $v_1, v_2 \in V$ with $\kappa(v_1) = \kappa(v_2)$, we have $v_1 = v_2$.*

We say that $\mathcal{G}$ is in anti-normal form*, if each vertex is incident with exactly one edge, in exactly one place, and if it has furthermore no redundant copies of edges. That is, it has to satisfy the following condition: If v is an vertex, and if $e = (v_1, v_2, v_3)$ and $f = (w_1, w_2, w_3)$ are two edges with $v_i = v = w_j$ for some $i, j \in \{1, 2, 3\}$, then we have $e = f$ and $i = j$.*

Next, we have to define two operations on RDF graphs which are already defined in [7] for graphs in triple notation: The *join* and *merge* of graphs. In the diagrammatical representation for graph formalization of RDF graphs, the join

of some graphs is to juxtapose them, i.e., to simply write them side by side. But the join of graphs can usually *not* be understood as their logical conjunction. This shall be exemplified: consider the one-triple RDF graphs `(_1 has_color blue)` (which means something has the color blue) and `(_1 has_color red)` (meaning something is red)—but their join should be 'something is blue and something is red' rather than 'something is blue and red'. So, only if the joined graphs have no blanks in common, their join corresponds to logical conjunction. In this case, we will speak of the *merge* of graphs. Formally, the join and merge are defined as follows:

Definition 7 (Join and Merge of graphs). *Let $n \in \mathbb{N}_0$ and for $i = 1, \ldots, n$ let $\mathcal{G}_i := (V_i, E_i, \nu_i, \kappa_i)$ be a RDF graph. W.l.o.g. we assume that all V_i and E_i, $i = 1, \ldots, n$, are pairwise disjoint. The* join of the graphs $\mathcal{G}_i$ *is defined to be the following RDF graph $\mathcal{G} := (V, E, \nu, \kappa)$:*

$$V := \bigcup_{i=1,\ldots,n} V_i \ , \quad E := \bigcup_{i=1,\ldots,n} E_i \ , \quad \nu := \bigcup_{i=1,\ldots,n} \nu_i \ , \text{ and } \kappa := \bigcup_{i=1,\ldots,n} \kappa_i \ .$$

In the graphical representation, the join of the $\mathcal{G}_i$ is simply noted by writing the graphs next to each other, i.e. we write:

$$\mathcal{G}_1 \ \mathcal{G}_2 \ \ldots \ \mathcal{G}_n \quad .$$

A join of the graphs $\mathcal{G}_i$ is also called merge of the graphs $\mathcal{G}_i$*, if the graphs $\mathcal{G}_i$ do not have blanks in common, i.e., for each $i, j \in \{1, \ldots, n\}$, $v \in V_i$, $w \in V_j$ with $\kappa_i(v) = \kappa_j(w) \in \mathsf{Blanks}$ it holds $i = j$.*

Note that the join or merge of an empty set of graphs is the empty graph.

Moreover, recall that RDF graphs are only considered up to equivalence. Thus, given some RDF graphs $\mathcal{G}_i$, $i = 1, \ldots, n$, we can always merge them by replacing each graph $\mathcal{G}_i$ with an equivalent graph $\mathcal{G}'_i$ such that the graphs $\mathcal{G}'_i$ do not have any blanks in common, and form the join of the graphs $\mathcal{G}'_i$.

3 Semantics

In this section, the semantics for RDF graphs is defined. It is based on mathematical model theory and adopted from [7, 8] for RDF. The models for RDF are defined as follows:

Definition 8 (Interpretation). *An* interpretation or model $\mathcal{I}$ for a vocabulary $\mathsf{Voc} := (\mathsf{URIs}, \mathsf{TypedLit}, \mathsf{PlainLit})$ *is a structure* $(\mathsf{IR}, \mathsf{IP}, \mathsf{IEXT}, \mathsf{IS}, \mathsf{IL})$ *where*

- IR *is a set of* resources*, called the* domain *or* universe *of $\mathcal{I}$.*
- IP *is a set of* properties *of $\mathcal{I}$.*
- $\mathsf{IEXT} : \mathsf{IP} \to \mathfrak{P}(\mathsf{IR} \times \mathsf{IR})$ *is a mapping.*
- $\mathsf{IS} : \mathsf{URIs} \to \mathsf{IR} \cup \mathsf{IP}$ *is a mapping.*
- $\mathsf{IL} : \mathsf{Lit} \to \mathsf{IR}$ *is a mapping.*

This definition of an interpretation is nearly the definition of an interpretation given in [7, 8], with one slight change: In [7, 8], the mapping IL maps only *typed* literals to resources, and an interpretation contains a subset $\mathsf{LV} \subseteq \mathsf{IR}$ with $\mathsf{LV} \supseteq \mathsf{Lit}$ (i.e., the set of untyped literals is assumed to be a subset of IR). From the RDF viewpoint, if an interpretation is implemented in an RDF application, it is reasonable to assume that the untyped literals are members of the interpretation. But from a mathematical point of view, a clear distinction between the names of the vocabulary and the objects (the resources) of an interpretation is appropriate. This change in the definition of an interpretation is not substantial: An interpretation according to Def. 8 can easily be transformed to an interpretation according to [7] by setting $\mathsf{LV} := \mathsf{IL}[\mathsf{PlainLit}]$.

Moreover, please note that the relationship between IR and IP is not specified. From the viewpoint of 'classical' mathematical logic, one would assume that IR and IP are disjoint. From the RDF viewpoint, it is quite normal to assume that $\mathsf{IP} \subseteq \mathsf{IR}$ holds. This is because properties may occur in RDF graphs as subjects or objects of other triples as well, i.e., a property may be applied to another property. Both cases, $\mathsf{IR} \cap \mathsf{IP} = \emptyset$ and $\mathsf{IP} \subseteq \mathsf{IR}$, are allowed.

In the next definition, RDF graphs are evaluated in structures. If a graph and an interpretation are given, we do not assume that they are based on the same vocabulary. But if a given name of the graph is not interpreted in the model, the semantics will always yield that the graph is false in the model. For this reason, we will usually assume that the vocabularies of a graph and an interpretation are the same.

Definition 9 (Interpreting RDF graphs in Models, Entailment). *Let* $\mathcal{G} := (V, E, \nu, \kappa)$ *be an RDF graph over a vocabulary* $\mathsf{Voc}_\mathcal{G} := (\mathsf{URIs}_\mathcal{G}, \mathsf{TypedLit}_\mathcal{G}, \mathsf{PlainLit}_\mathcal{G})$ *and let* $\mathcal{I} := (\mathsf{IR}, \mathsf{IP}, \mathsf{IEXT}, \mathsf{IS}, \mathsf{IL})$ *be an interpretation over a vocabulary* $\mathsf{Voc}_\mathcal{I} := (\mathsf{URIs}_\mathcal{I}, \mathsf{TypedLit}_\mathcal{I}, \mathsf{PlainLit}_\mathcal{I})$.

A function $I : V \to \mathsf{IR}$ *is an* interpretation function*, if it satisfies*

1. *If* $v_1, v_2 \in V$ *are vertices with* $\kappa(v_1) = \kappa(v_2)$*, then* $I(v_1) = I(v_2)$*, and*
2. $I(v) = (\mathsf{IL} \cup \mathsf{IR})(n)$ *for each each name* $n \in \mathsf{URIs}_\mathcal{I} \cup \mathsf{PlainLit}_I \cup \mathsf{TypedLit}_I$ *and each vertex* $v \in V$ *with* $\kappa(v) = n$.

We say the graph $\mathcal{G}$ holds in the model $\mathcal{I}$*, which is written* $\mathcal{I} \models \mathcal{G}$*, iff*

1. *for each vertex* $v \in V$*, we have* $\kappa(v) \in \mathsf{URIs}_\mathcal{I} \cup \mathsf{TypedLit}_\mathcal{I} \cup \mathsf{PlainLit}_\mathcal{I}$*, and*
2. *there exists an interpretation function* $I : V \to \mathsf{IR}$ *such that for each edge* $e = (v_1, v_2, v_3) \in E$ *and* $s := \kappa(v_1)$*,* $p := \kappa(v_2)$*,* $o := \kappa(v_3)$*, we have* $I(p) \in \mathsf{IP}$ *and* $(I(s), I(o)) \in \mathsf{IEXT}(I(p))$ *(this condition is called* edge condition*).*

If $\mathcal{G}_a, \mathcal{G}_b$ *are RDF graphs such that* $\mathcal{I} \models \mathcal{G}_b$ *holds for each interpretation* $\mathcal{I}$ *with* $\mathcal{I} \models \mathcal{G}_a$*, we say that* $\mathcal{G}_a$ (semantically) entails $\mathcal{G}_b$*, and write* $\mathcal{G}_a \models \mathcal{G}_b$.

4 Calculus

In the following, a calculus for graph formalizations of RDF graphs is provided. This calculus consists of five transformation rules. In the following, each of these

rules is described in a semi-formal manner, before a precise definition is provided. In the definition, the term 'fresh' is used for vertices or edges, similar to its common use in logic: given a formula, a 'fresh' variable is a variable which does not occur in the formula. Analogously, given a graph (V, E, ν, κ), a vertex or edge x is called *fresh* is we have $x \notin V \cup E$. Now, let $\mathcal{G} := (V, E, \nu, \kappa)$ be given. The following transformations are allowed:

1. **erasing an edge**
   Let e be an edge. Then e may be erased (and each vertex v which is only incident with e has to be erased as well).
   That is, we construct the graph $\mathcal{G}^e := (V^e, E^e, \nu^e, \kappa^e)$ where we set $V^{(e)} := V \backslash \{v \in V_e \,|\, v \notin V_f \text{ for any } f \in E \text{ with } f \neq e\}$, $E^{(e)} := E \backslash \{e\}$, $\nu^{(e)} := \nu\big|_{E^e}$, and $\kappa^{(e)} := \kappa | E^e$.
2. **generalizing a label**
   Let $bl \in \mathsf{Blanks}$ be a blank which does not appear in $\mathcal{G}$, i.e., for each vertex $v \in V$ we have $\kappa(v) \neq bl$. Let $V' \subseteq V$ be a set of vertices which are identically labeled, i.e., we have $\kappa(v_1) = \kappa(v_2)$ for all $v_1, v_2 \in V'$. Then, for each $v \in V$, $\kappa(v)$ may be replaced by $\kappa(v) := bl$.
   That is, we construct the graph $\mathcal{G}^g := (V, E, \nu, \kappa^g)$ with $\kappa^g(w) := \kappa(w)$ for all $w \notin V'$ and $\kappa^g(v) := bl$ for all $w \in V'$.
3. **merging two vertices**
   Let v_1, v_2 be two different vertices with $\kappa(v_1) = \kappa(v_2)$. Then v_2 may be merged into v_1 (i.e., v_2 is erased and, for every edge $e \in E$, $e\big|_i = v_2$ is replaced by $e\big|_i = v_1$).
   That is, we construct the graph $\mathcal{G}^m := (V^m, E, \nu^m, \kappa)$ with $V^m := V \backslash \{v_2\}$, and for all $e \in E$ and $i = 1, 2, 3$ we have $\nu^m(e)(i) := \nu(e)(i)$, if $\nu(e)(i) \neq v_2$, and $\nu^m(e)(i) := v_1$ else.
4. **splitting a vertex**
   Let v_1 be a vertex, incident with edges $e_1, \ldots, e_n$. Then the following may be done: A fresh vertex v_2 with $\kappa(v_2) := \kappa(v_1)$ is inserted. On $e_1, \ldots, e_n$, arbitrary occurrences of v_1 may be replaced by v_2.
   That is, we construct a[3] graph $\mathcal{G}^i := (V^i, E, \nu^i, \kappa)$ with $V^m := V \,\dot\cup\, \{v_2\}$, for all $e \in E$ and $i = 1, 2, 3$ we have $\nu^m(e)(i) := \nu(e)(i)$, if $\nu(e)(i) \neq v_1$ and $\nu^m(e)(i) \in \{v_1, v_2\}$ else, and which satisfies $E_{v_1} \neq \emptyset \neq E_{v_2}$ (otherwise the structure is not a well-formed RDF graph).
5. **iterating an edge**
   Let e be an edge with $\nu(e) = (v_1, v_2, v_3)$. Then a fresh edge e' with $\nu(e') := (v_1, v_2, v_3)$ and $\kappa(e') := \kappa(e')$ may be inserted.
   That is, we construct the graph $\mathcal{G}^i := (V, E^i, \nu^i, \kappa)$ with $E^{(i)} := E \,\dot\cup\, \{e'\}$ and $\nu^i := \nu \,\dot\cup\, \{(e', (v_1, v_2, v_3))\}$.

Based on these transformation, we can now define what a *formal proof* with RDF graphs is.

[3] Even for a fixed vertex, this rule allows to derive different graphs.

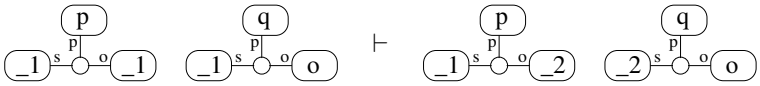
Definition 10 (Syntactical Entailment Relation). *Let $\mathcal{G}_a$, $\mathcal{G}_b$ be two RDF graphs. Then $\mathcal{G}_b$* can be derived from *$\mathcal{G}_a$ (which is written $\mathcal{G}_a \vdash \mathcal{G}_b$), if there is a finite sequence $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n)$ with $\mathcal{G}_1 = \mathcal{G}_a$ and $\mathcal{G}_b = \mathcal{G}_n$ such that each $\mathcal{G}_{i+1}$ is derived from $\mathcal{G}_i$ by applying one of the rules above. The sequence is called* a proof for *$\mathcal{G}_a \vdash \mathcal{G}_b$. Two graphs $\mathcal{G}_1, \mathcal{G}_2$ with $\mathcal{G}_1 \vdash \mathcal{G}_2$ and $\mathcal{G}_2 \vdash \mathcal{G}_1$ are said to be* provably equivalent*.*

If $\mathfrak{H} := \{\mathcal{G}_i \mid i \in I\}$ is a (possibly empty) set of RDF graphs, then a graph $\mathcal{G}$ can be derived from $\mathfrak{H}$ *if there is a finite subset $\{\mathcal{G}_1, \ldots, \mathcal{G}_n\} \subseteq \mathfrak{H}$ such that $\mathcal{G}$ can be derived from the merge of $\mathcal{G}_1 \ldots \mathcal{G}_n$.*

We want to make some remarks to these rules.

In none of the rules, new names are added to a graph. Thus, if a graph $\mathcal{G}$ over a vocabulary Voc is given, and $\mathcal{G}'$ is derived from $\mathcal{G}$, then $\mathcal{G}'$ is an RDF graph over Voc as well.

In the rule 'generalizing a label', it is important that the label of a subset of vertices and not only from a single vertex can be generalized. Otherwise, the following derivation would not be possible.



Obviously, the rules 'merging two vertices' and 'splitting a vertex' are inverse to each other. That is, if $\mathcal{G}'$ can be derived from $\mathcal{G}$ with the rule 'merging two vertices', then $\mathcal{G}$ can be derived from $\mathcal{G}'$ with the rule 'splitting a vertex', and vice versa.

To grasp the complete possibilities of the rules 'splitting a vertex' and 'merging two vertices', it is important to note that a vertex v can be incident up to three times with an edge e (edges $e = (v, v, v)$ are allowed). These are the occurrences of v on e, and it possible to replace *some* of the occurrences by the copy v' of v. To see an example: The graph of Fig. 1 is obtained from the graph of Fig. 4 by splitting the vertex labeled with `rdf:type`, and vice versa, the graph of Fig. 4 is obtained from the graph of Fig. 1 by merging the vertices labeled with `rdf:type`.

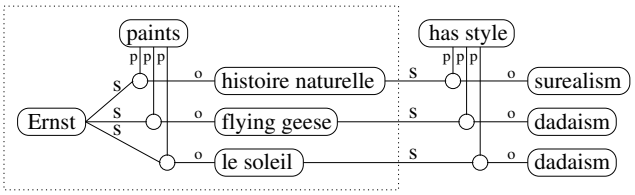
The rule 'iterating an edge' can be reversed with the rule 'erasing an edge'. On the other hand, let a redundant edge $e = (v_1, v_2, v_3)$ be given, i.e., there is another edge $f \neq e$, $f = (w_1, w_2, w_3)$ with $\kappa(v_i) = \kappa(w_i)$ for $i = 1, 2, 3$. If e is erased with the rule 'erasing an edge', then this transformation can be reversed by iterating and isolating e and merging the fresh vertices v_1', v_2', v_3' into w_1, w_2, w_3, respectively. If the rule 'iterating an edge' is applied to an edge $e = (v_1, v_2, v_3)$, then the new edge e' is incident with v_1, v_2, v_3 as well (we have $e' = (v_1, v_2, v_3)$). But, applying the rule 'splitting a vertex' to v_1, v_2 and v_3 allows to 'isolate' the incident vertices of e' (we will call this transformation *iterating and isolating the edge* e). That is, it is possible to build a graph $\mathcal{G}^{is} := (V^{is}, E^{is}, \nu^{is}, \kappa^{is})$ with $V^{(is)} := V \dot\cup \{v_1', v_2', v_3'\}$ for fresh vertices v_1', v_2', v_3', $E^{(is)} := E \dot\cup \{e'\}$, $E^{(is)} := E \dot\cup \{e'\}$, $\nu^{is} := \nu \dot\cup \{(e', (v_1', v_2', v_3')\}$ and $\kappa^i := \kappa \cup \{(v_1', \kappa(v_1)), (v_2', \kappa(v_2)), (v_3', \kappa(v_3))\}$.

The last idea can easily be extended to arbitrary subgraphs. To see this, let $\mathcal{G} := (V, E, \nu, \kappa)$ be an RDF graph with a subgraph $\mathcal{G}_s := (V_s, E_s, \nu_s, \kappa_s)$. For each $e \in E_s$, we can make a copy e' of e with the rule 'iterating an edge'. Afterwards, we can split each $v \in V_s$ such that for each edge $e \in E$ where v is incident with, the copy v' of v is incident with the copy e' of e. Let us call this transformation *iterating the subgraph* $\mathcal{G}_s$. We have seen that if $\mathcal{G}'$ is obtained from $\mathcal{G}$ by iterating a subgraph, we have $\mathcal{G} \vdash \mathcal{G}'$, i.e., $\mathcal{G}'$ can be derived from $\mathcal{G}$.

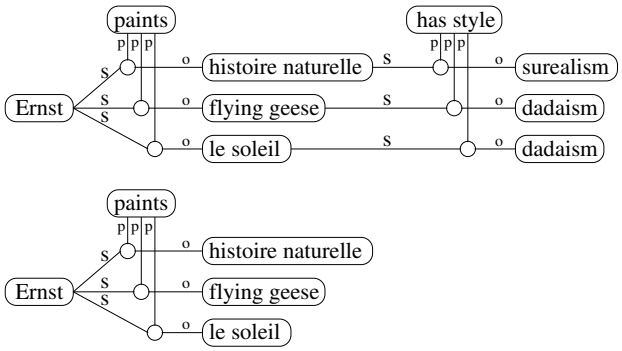
Similar to the rule 'iterating an edge', the rule 'erasing an edge' can be extended to subgraphs as well. But we have to take care of one important detail. Suppose again we have an RDF graph $\mathcal{G} := (V, E, \nu, \kappa)$ with a subgraph $\mathcal{G}_s := (V_s, E_s, \nu_s, \kappa_s)$, and we want to remove all vertices $v \in V_s$ and all edges $e \in E_s$ from $\mathcal{G}$. But if V_s contains a vertex v which is incident with an edge $e \in E \backslash E_s$ (i.e., $E_v \not\subseteq E_s$), then removing v, but not e, does not yield a well-formed RDF graph. Assume on the other hand that we have $E_v \subseteq E_s$ for each $v \in V_s$, i.e., assume that $\mathcal{G}_s$ is a *closed* subgraph. Then removing all edges $e \in E_s$ with the rule 'erasing an edge' automatically yields a graph $\mathcal{G}'$ where all vertices $v \in V_s$ are removed as well. Let us call this transformation *erasing the closed subgraph* $\mathcal{G}_s$. Obviously, if $\mathcal{G}'$ is obtained from $\mathcal{G}$ by erasing a closed subgraph, we have $\mathcal{G} \vdash \mathcal{G}'$.

In [7] is is shown that given two RDF graphs $\mathcal{G}$, $\mathcal{G}'$ in triple notation, then $\mathcal{G}$ entails $\mathcal{G}'$ iff a subgraph of $\mathcal{G}$ is an instance of $\mathcal{G}'$. If we call $\mathcal{G}'$ a *generalization* of $\mathcal{G}$ iff $\mathcal{G}$ is an instance of $\mathcal{G}'$, we see that $\mathcal{G}$ entails $\mathcal{G}'$ iff $\mathcal{G}'$ is a generalization of a subgraph of $\mathcal{G}$. For RDF graphs in triple notation, an instance of a graph may contain less triples, thus a generalization of a graph may contain more triples. For the graph representation of graphs, the rule 'generalizing a label' does not change the number of edges, i.e., the number of statements. For increasing the number of statements, the rule 'iterating an edge' is needed. Hence the idea of constructing a generalization for graph formalizations of RDF graphs is reflected by the rules 'iterating an edge' and 'generalizing a label'. Moreover, constructing a subgraph of an RDF Graph in triple notation for graph formalizations of RDF Graphs is reflected by the rule 'erasing an edge'. Thus, roughly spoken, these three rules for graph representations of RFD graphs reflect the calculus for RDF graphs in triple notation. The remaining two rules 'merging two vertices' and 'splitting a vertex' are needed because graph formalizations of RDF Graphs provide richer means to express rhetoric structures (as discussed above).

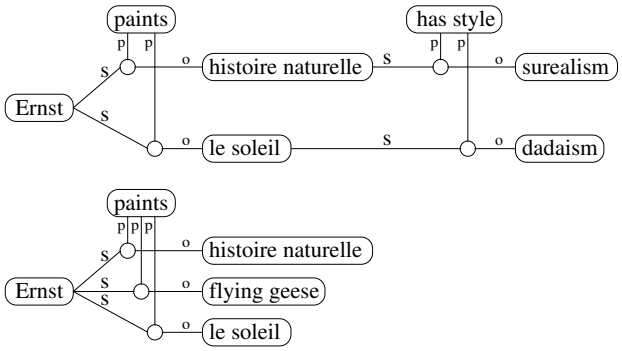
To exemplify the calculus, we return to the 'Max Ernst example' of the introduction: two graphs, one entailing the other, were presented in triple notation. A diagram of a possible graph formalization for the starting graph is depicted below. Furthermore, a dotted rectangle indicates a subgraph of this graph.



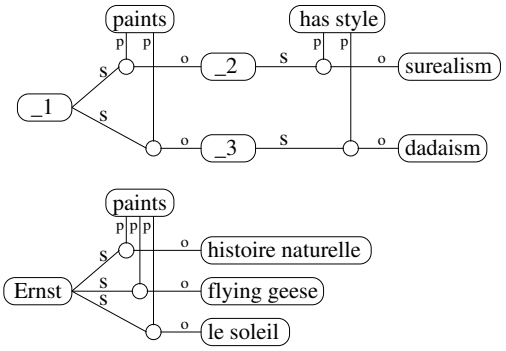
Now we do the following: First, we iterate the subgraph, i.e., we generate a redundant copy of the information given by the subgraph.



Next, we can remove some information by removing two edges.



Finally we replace some of the names in the graph by blanks.



This is a possible graph formalization for the second graph of the example in the introduction.

The next lemma shows that, using the rules ‘merging two vertices’, ‘splitting a vertex’ and ’erasing an edge’ (where only redundant copies of edges are removed), each RDF graph can be transformed into normal form or anti-normal form. This will turn out useful in the proof of the completeness.

Lemma 1 (Normalizing graphs). *Using the rules 'splitting an vertex', 'merging two vertices' and 'erasing an edge', each RDF graph $\mathcal{G}$ is provably equivalent to a RDF graph $\mathcal{G}_n$ in normal form, and to a RDF graph $\mathcal{G}_{an}$ in anti-normal form.*

Proof: We start to show how $\mathcal{G}$ is transformed into normal form. First, we apply the rule 'splitting an vertex' until each vertex is incident with exactly one edge, in exactly one place. Then, redundant copies of edges are erased with the erasure rule. Because all these transformations may b e reversed (with the rule 'merging two vertices' and 'iterating and isolating an edge') the new constructed graph $\mathcal{G}_{an}$ is equivalent to $\mathcal{G}$. Obviously, the new graph is in anti-normal form.

Analogously, if we apply the rule 'merging two vertices' until different vertices always have different labels, and if we again erase afterwards redundant copies of edges are erased with the erasure rule, we get a graph $\mathcal{G}_n$ which is equivalent to $\mathcal{G}$ and which is in normal form. □

In the following, we will show that the calculus is adequate, i.e., sound and complete. We start with the soundness of the rules.

Theorem 1 (Soundness of the Rules). *Two RDF graphs $\mathcal{G}_a$, $\mathcal{G}_b$ satisfy* $\mathcal{G}_a \vdash \mathcal{G}_b \implies \mathcal{G}_a \models \mathcal{G}_b$.

Proof: We have to show the following: If $\mathcal{I}$ is an interpretation with $\mathcal{I} \models \mathcal{G}_a$, and $\mathcal{G}_b$ is derived from $\mathcal{G}_a$ with one of the rules, then we have $\mathcal{I} \models \mathcal{G}_b$. This will be done for each rule separately.

As an RDF graph $\mathcal{G}$ can only hold in an interpretation $\mathcal{I}$ if each name which appears in $\mathcal{G}$ belongs to the vocabulary of $\mathcal{I}$, we can assume w.l.o.g. that $\mathcal{G}_a$, $\mathcal{G}_b$ and $\mathcal{I}$ are based on the same vocabulary. Now a RDF graph $\mathcal{G}$ holds in an interpretation $\mathcal{I}$, if there is a interpretation function I which satisfies the edge conditions for $\mathcal{G}$. In this proof, this will be denoted by $(\mathcal{I}, I) \models \mathcal{G}$.

As we have $\mathcal{I} \models \mathcal{G}$, we have an Interpretation function I_a with $(\mathcal{I}, I_a) \models \mathcal{G}_a$. For each rule, we have to construct an interpretation function I_b with $(\mathcal{I}, I_b) \models \mathcal{G}_b$.

1. **erasing an edge** For $I_b := I_a$ we obviously have $(\mathcal{I}, I_b) \models \mathcal{G}_b$, since for $\mathcal{G}_b$, we have to check the same edge conditions, except the condition for e.
2. **generalizing a label** As bl does not appear in $\mathcal{G}$ and all vertices $v \in V'$ are labeled the same, $I_b := I_a$ is an interpretation function for $\mathcal{G}_b$ (the first condition for interpretation functions is satisfied). Again it is obvious that we have $(\mathcal{I}, I_b) \models \mathcal{G}_b$.
3. **merging two vertices** We set $I_b := I_a \backslash \{(v_1, I_a(v_1))\}$. As $I_a(v_1) = I_a(v_2)$, I_b satisfies $(\mathcal{I}, I_b) \models \mathcal{G}_b$.
4. **splitting a vertex** We set $I_b := I_a \cup \{(v_2, I_a(v_1))\}$. I_b is an interpretation function (particularly, the first condition for interpretation functions is satisfied) with $(\mathcal{I}, I_b) \models \mathcal{G}_b$.
5. **iterating an edge** We set $I_b := I_a$. For checking $(\mathcal{I}, I_b) \models \mathcal{G}_b$, we have to check all edge conditions for $\mathcal{G}_b$, which are exactly the edge conditions for $\mathcal{G}_a$ plus the additional edge condition for the fresh edge e'. Since e' has the same

vertices as e, this edge condition is satisfied, because the edge condition for e is satisfied. Thus we have $(\mathcal{I}, I_b) \models \mathcal{G}_b$. □

Before we show that the rules are complete of well, we have to introduce the notion of *Herbrand models*. Herbrand models are commonly used in completeness proofs for mathematical logic, particularly for first order predicate logic. Usually, they are not uniquely defined, but this is different for RDF graphs. As we have in RDF graphs only the possibility to express existential quantification and logical conjunction, we can assign to each RDF graph a unique Herbrand model which encodes exactly the same information as the RDF graph. For this reason, we call these models *standard models* as well.

Definition 11 (Herbrand Model). *Let $\mathcal{G} := (V, E, \nu, \kappa)$ be a RDF graph over a vocabulary* Voc*. Then $\mathcal{I}_\mathcal{G}$ is defined as follows:*

- $\mathsf{IR} := \mathsf{URIs} \cup \mathsf{Lit} \cup \{bl \in \mathsf{Blanks} \mid$ *It exists a vertex $v \in V$ with $\kappa(v) = bl$*$\}$
- $\mathsf{IP} := \{\kappa(v) \mid v \in V$ *and there exists an edge e with $e\big|_2 = v$*$\}$
- $\mathsf{IEXT}(p) = \{(s, o) \mid$ *it exists a $e \in E$ with $\kappa(e\big|_1) = s$, $\kappa(e\big|_2) = p$, $\kappa(e\big|_3) = o$*$\}$
- IS *and* IL *are the identity mappings on* URIs *resp.* Lit.

The model $\mathcal{I}_\mathcal{G}$ is called Herbrand model *or* standard model *for $\mathcal{G}$.*

Using standard models of RDF graphs, we can now show that the transformation rules are complete. As we have already shown the soundness as well, we therefore have that the calculus is adequate.

Theorem 2 (Completeness of the Rules). *Let $\mathcal{G}_a$ and $\mathcal{G}_b$ be two RDF graphs over* Voc *with $\mathcal{G}_a \models \mathcal{G}_b$. Then we have $\mathcal{G}_a \vdash \mathcal{G}_b$.*

Proof: Due to Lem. 1 we assume that both $\mathcal{G}_a := (V_a, E_a, \nu_a, \kappa_a)$ and $\mathcal{G}_b := (V_b, E_b, \nu_b, \kappa_b)$ are in anti-normal form. Moreover, as we consider graphs only up to equivalence, we assume that $\mathcal{G}_a$ and $\mathcal{G}_b$ have no blanks in common.

Let $\mathcal{I}_a := (\mathsf{IR}_a, \mathsf{IP}_a, \mathsf{IEXT}_a, \mathsf{IS}_a, \mathsf{IL}_a)$ be the Herbrand model of $\mathcal{G}_a$. We have $\mathcal{I}_a \models \mathcal{G}_b$, i.e. there is an interpretation function $I_b : V_b \to \mathsf{IR}_a$ such that for each edge $e = (v_1, v_2, v_3) \in E_b$ and $s := \kappa(v_1)$, $p := \kappa(v_2)$, $o := \kappa(v_3)$, we have $I_b(p) \in \mathsf{IP}$ and $(I_b(s), I_b(o)) \in \mathsf{IEXT}(I_b(p))$.

Due to the definition of the Herbrand model, the mapping $\lambda_N := I_b$ is a mapping from the set of blanks and names of $\mathcal{G}_b$ into the set of blanks and names of $\mathcal{G}_a$ which is the identity mapping on the names of Voc, and we have a mapping $\lambda_E : E_b \to E_a$ which satisfies

$$\begin{array}{l}\text{For each } e := (v_1, v_2, v_3) \in E_b \text{ and } \lambda_E(e) := (w_1, w_2, w_3) \in E_a \\ \text{we have } \lambda_N(\kappa(v_i)) = \kappa(w_i) \text{ for } i = 1, 2, 3\end{array} \qquad (*)$$

The mapping λ_E is possibly neither injective nor surjective, but we can transform $\mathcal{G}_a$ in order to make λ_E bijective. This will be done now.

Assume first that we have $\lambda_E(e_1) = \lambda_E(e_2)$ for two edges $e_1, e_2 \in E_b$. Set $f := \lambda_E(e_1)$. We can apply the rule 'iterating and isolating an edge' to f in order to obtain a copy f' of f, and we can change λ_E to λ'_E as follows: We set $\lambda'_E(e_1) := f$ and $\lambda'_E(e_2) := f'$, otherwise we set $\lambda'_E(e) := \lambda_E(e)$. As we do

not add or remove names from $\mathcal{G}_b$, the mapping λ_N remains unchanged. This step is repeated as often as necessary until we obtain a graph $\mathcal{G}_a^{(1)}$ from $\mathcal{G}_a$ (particularly, we have $\mathcal{G}_a \vdash \mathcal{G}_a^{(1)}$) with a mapping $\lambda_N^{(1)} := \lambda_N$ and an injective mapping $\lambda_E^{(1)} : E_b \to E_a^{(1)}$ which satisfies $(*)$ as well.

Next, we remove all edges $f \in E_a^{(1)}$ which are not in the range of $\lambda_E^{(1)}$. The resulting graph is denoted by $\mathcal{G}_a^{(2)}$, the restrictions of $\lambda_N^{(1)}$ and $\lambda_E^{(1)}$ to $\mathcal{G}_a^{(2)}$ by $\lambda_N^{(2)}$ and $\lambda_E^{(2)}$. Again, now for $\mathcal{G}_a^{(2)}$, $\lambda_N^{(1)}$ and $\lambda_E^{(1)}$, the condition $(*)$ is fulfilled. Obviously, the mapping $\lambda_N^{(2)}$ is bijective. Moreover, as in $\mathcal{G}_a^{(2)}$ and $\mathcal{G}_b$ each vertex is incident with exactly one edge, in exactly one place, we obtain moreover a bijective mapping $\lambda_V^{(2)} : V_b \to V_a^{(2)}$ which respects the incidence relation, i.e. if $e \in E_b$ is an edge and $v = e\big|_i$ for an $i = 1, 2, 3$; we have $\lambda_V^{(2)}(v) = \lambda_E^{(2)}(e)\big|_i$. That is, the underlying graph structures of $\mathcal{G}_a^{(2)}$ and $\mathcal{G}_b$ are isomorphic, only the labels of the vertices differ.

For each blank bl of $\mathcal{G}_b$ we set

$$V_{bl} := \{w \in V_a^{(2)} \mid \text{it exists a } v \in V_b \text{ with } \kappa_b(v) = bl \text{ and } \lambda_V(v) = w\}\,.$$

Now, for each such blank bl, we can apply the rule 'generalizing a label' to $\mathcal{G}_a^{(2)}$ in order to label all vertices in V_{bl} with the blank bl. Note that this is possible because we assumed that the set of blanks in $\mathcal{G}_a$ (thus the set of blanks in $\mathcal{G}_a^{(2)}$) is disjoint from the set of blanks in $\mathcal{G}_b$.

The resulting graph $\mathcal{G}_b^{(2)}$ obviously is isomorphic to $\mathcal{G}_a$ (whereas isomorphisms between RDF graphs are defined canonically), thus we are done. □

5 Outlook

This paper is only a first step to develop RDF as a diagrammatic reasoning system. The authors are currently working on versions of this logic system where the graph formalizations of RDF graphs are restricted to be in normal form resp. in anti-normal form. This corresponds to the triple model of RDF without any rhetorical structure in the graph formalization. For these cases, the calculus is not complete anymore, and has to be redesigned.

The next logical step is to extend this system to capture RDF Schema semantics, particularly to the property and class hierarchies. Moreover, it has to be investigated how this system can be extended to further logical operations, like `or` and `not`. For this, we can use the results of [4], where the cuts of Peirce's existential graphs, which are used to logically negate some subgraphs of the graphs, are added to a fragment of conceptual graphs which is very similar to the RDF graphs investigated here. In [4], adding cuts to conceptual graphs can be compared to make the step from the (trivially decidable) existential-conjunctive fragment of first order predicate logic to full first order predicate logic.

References

1. T. Berners-Lee: *Conceptual Graphs and the Semantic Web.*

http://www.w3.org/DesignIssues/CG.html
2. T. Berners-Lee, E. Miller: *The Semantic Web lifts off* http://www.ercim.org/publication/Ercim_News/enw51/berners-lee.html
3. T. Bray: *Ongoing: DC Intel* http://www.tbray.org/ongoing/When/200x/2003/09/06/ICMWG,2003
4. F. Dau: *The Logic System of Concept Graphs with Negation (And Its Relationship to Predicate Logic).* Lecture Notes in Artificial Intelligence, Vol. 2892, ISBN 3-540-20607-8. Springer, Berlin–Heidelberg–New York, 2003.
5. J. Hayes: A Graph Model for RDF. Diploma thesis, Technische Universität Darmstadt, Department of Computer Science, Germany, http://purl.org/net/jhayes/rdfgraphmodel.html (2004)
6. J. Hayes, C. Gutierrez: *Bipartite Graphs as Intermediate Model for RDF.* In: Proceedings of ISWC 2004, The Third International Semantic Web Conference, Hiroshima, Japan, November 7 - 11, 2004. Lecture Notes in Computer Science, Springer Verlag (2004).
7. P. Hayes: *RDF Semantics: W3C Recommendation.* 10 February 2004. http://www.w3.org/TR/rdf-mt/
8. P. Hayes, C. Menzel: *A Semantics for the Knowledge Interchange Format.* Proceedings of 2001 Workshop on the IEEE Standard Upper Ontology, August 2001. http://reliant.teknowledge.com/IJCAI01/HayesMenzel-SKIF-IJCAI2001.pdf
9. G. Klyne, J.J. Carroll: *Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation.* http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (10 February 2004)
10. F. Manola, E. Miller: *RDF Primer.* http://www.w3.org/TR/rdf-primer/
11. J. F. Sowa: *Conceptual Structures: Information Processing in Mind and Machine.* Addison Wesley Publishing Company Reading, 1984.
12. J. F. Sowa: *Conceptual Graphs Summary.* In: T. E. Nagle, J. A. Nagle, L. L. Gerholz, P. W. Eklund (Eds.): Conceptual Structures: current research and practice, Ellis Horwood, 1992, 3–51.
13. J. F. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Brooks Cole Publishing Co., Pacific Grove, CA, 2000.