Electronic Communications of the EASST
Volume 10 (2008)


# Proceedings of the Seventh International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2008)

## Graph Transformations for the Resource Description Framework

Benjamin Braatz and Christoph Brandt

16 pages

Guest Editors: Claudia Ermel, Reiko Heckel, Juan de Lara
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# Graph Transformations for the Resource Description Framework

**Benjamin Braatz**[1] **and Christoph Brandt**[2]

[1] bbraatz@cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

[2] christoph.brandt@uni.lu
Computer Science and Communications Research Unit
Université du Luxembourg, Luxembourg

**Abstract:** The Resource Description Framework (RDF) is a standard developed by the World Wide Web Consortium (W3C) to facilitate the representation and exchange of structured (meta-)data in the "Semantic Web". While there is a large body of work dealing with inference on RDF, a concept for transformation and manipulation is still missing. Since RDF uses graphs as a formal basis, this paper proposes the use of algebraic graph transformations with their wealth of well-known constructions and results for this purpose. It turns out that RDF graphs are an interesting application area for graph transformation methods, where some significant differences to classical graphs yield practically relevant solutions for features like attribution, typing and globally unique nodes.

**Keywords:** Resource Description Framework, Algebraic Graph Transformation, Category Theory

## 1 Introduction

The Resource Description Framework (RDF) (see [MM04]) consists of a set of specifications, which provide an abstract syntax, semantics and several concrete representations for storage, exchange and reasoning on arbitrary (meta-)data. Its primary use case is the "Semantic Web", in which the creation of globally distributed knowledge bases is envisaged. In Section 2 we summarise the abstract syntax of RDF and provide a categorical framework for it.

The theoretical treatment of RDF mainly consists of inference mechanisms, which allow to derive information from RDF data stores by the use of inference rules like, e. g., the transitivity of a predicate. This focus is also apparent in the SPARQL Query Language for RDF (see [PS07]), the proposed standard for accessing an RDF data store, which in contrast to SQL does not contain any structures for manipulating the data themselves, but only constructs for retrieval.

At the present time, the modification of data in RDF stores is mostly implemented by adding and removing individual data items. For many use cases, such as data in web-based applications or the use of RDF as an abstract syntax for visual languages, it would, however, be desirable to restrict this to sensible rule-based transformations, which can be modelled and analysed formally.

This contribution proposes the use of algebraic graph transformation for the purpose of manipulating RDF data stores. Such a transformation approach allows for the formal treatment of reversibility, dependencies and conflicts of transformations. Moreover, it facilitates the use of grammars to restrict the possible structures and their development. In Section 3 we investigate several well-known graph transformation approaches concerning their suitability for RDF and in Section 4 we present our proposal for an RDF transformation concept.

As a use case for rule-based transformations, consider a web-based application storing bibliographical information in an RDF data store. With a set of grammar rules we can restrict the possible modifications to these data, such that only sensible information is represented at any given time, e. g., it can be ensured that each publication has a title and an author or an editor. Complex rules, derived from the simple ones in the grammar by certain constructions such as union and sequential composition, could be used to implement and formally analyse changes done by committing a whole form in the web interface. For example, the effect of a form to add or edit a whole publication of a certain type can be given by a rule constructed from all grammar rules applicable to this type of publication. Reversibility allows us to undo any changes done by applying a rule, where the analysis of dependencies would allow us to undo any of the last unrelated changes instead of only the very last one. Moreover, the analysis of conflicts between rules applied to the same graphs would enable us to resolve concurrent modifications to the data store. In Section 5 the possibilities, which are already achieved by this contribution, are summarised and possible lines of future work are related to their benefits in this example scenario.

## 2 The Resource Description Framework

In this section we reformulate and slightly extend the theoretical underpinning of RDF in the framework of category theory. Some of the specific phenomena of RDF theory correspond to well-known constructions in category theory and can hence be treated more elegantly in this setting, which makes this reformulation worthwhile independently of the transformation approach developed in the following sections.

### 2.1 RDF Graphs

The basic building blocks of RDF are Uniform Resource Identifiers (URIs) (see [BFM98]) and literals. For the purposes of this paper we assume to have a given set $\mathsf{URI}$ of URIs, where we will use XML Namespaces (see [BHLT06]) to shorten URIs. The namespaces rdf:, rdfs: and xsd: are used for pre-defined URIs in the corresponding RDF and XML Schema specifications.

Literals are Unicode strings (see [Uni07]), which can be typed literals (with a URI denoting the data type of the literal) or plain literals (with an optional language tag denoting the human language of the literal). We formalise this by assuming a set $\mathsf{String}$ of all Unicode strings and a set $\mathsf{Lang}$ of all language tags (including the empty tag to support optionality). The set of all literals is then constructed by $\mathsf{Lit} := (\mathsf{URI} \times \mathsf{String}) + (\mathsf{Lang} \times \mathsf{String})$, where $\times$ denotes the (Cartesian) product and $+$ the disjoint union of sets.

The typed literals facilitate the attribution of an RDF graph by literal values from arbitrary pre- or self-defined data types, which are given by corresponding string representations. For

example, the datatypes of XML Schema (see [BM04]) are recommended in the RDF specifications, such that literals like $(\mathsf{xsd{:}integer}, 42)$ or $(\mathsf{xsd{:}date}, 2008\text{-}03\text{-}29)$ may be used in RDF graphs. Hence, an extension of the theory by algebraic specifications or similar techniques as it is necessary for attributed graphs (see, e. g., [EEPT06]) is not needed in RDF.

RDF graphs are now given by sets of statements, where a statement is a triple consisting of a subject, a predicate and an object. In the language of graph theory subjects and objects are nodes and statements are edges labelled with predicates. Subjects and objects can be URIs, literals or "blank nodes", where blank nodes are nodes, which do not have a global identity, but are local to the graph. Predicates are always given by URIs.

**Definition 1** (RDF Graph) An RDF graph $G = (G_{\text{Blank}}, G_{\text{Triple}})$ consists of a set $G_{\text{Blank}}$ of blank nodes and a set $G_{\text{Triple}} \subseteq (G_{\text{Blank}} + \mathsf{URI} + \mathsf{Lit}) \times \mathsf{URI} \times (G_{\text{Blank}} + \mathsf{URI} + \mathsf{Lit})$ of triples.

*Remark* 1 (Differences to RDF specifications) *The formal specifications of RDF (see [KC04] and [Hay04]) assume the blank nodes to be drawn from an infinite set, which is given globally. Our approach is to keep the blank nodes local to the graphs and use category theoretical machinery to ensure disjointness of blank nodes from unrelated graphs in the following subsections.*

*Moreover, the RDF specifications only allow literals as objects, but not as subjects of triples. The relaxation of this requirement is, however, common in later literature (see e. g. [MPG07]) and eases our formal treatment significantly.*

Since the edges are defined by a subset of the possible triples, there may be at most one edge with a given predicate between the same nodes. Thus, RDF graphs are a special kind of simple graphs, as opposed to multigraphs, which may have an arbitrary number of edges between two nodes. This is justified by the interpretation of edges as true statements about the world, since it should not make any difference, how often a statement is assured to be true, but only if it is true at all.

The use of global sets of URIs and literals enables the distributed creation and storage of information regarding the same entities and resources, where their connection is established by the usage of identical URIs without the need to give explicit relations or morphisms.

In order to allow the typing of nodes by classes, the declaration of domains and codomains for predicates, and hierarchies of classes and predicates, RDF defines some special URIs for these concepts in "vocabularies" (see [BG04]). In [MPG07] it is shown that from the rather verbose vocabulary in [BG04] only the predicates rdf:type, rdfs:dom, rdfs:range, rdfs:subClassOf and rdfs:subPropertyOf are needed to achieve the same essential structure. A triple $(s, \mathsf{rdf{:}type}, c)$ states that one of the classes of node $s$ is represented by the node $c$. Triples $(p, \mathsf{rdfs{:}dom}, c)$ and $(p, \mathsf{rdfs{:}range}, d)$ require for a triple $(s, p, o)$ using the URI $p$ as a predicate that $c$ is among the classes of the subject $s$ and $d$ among the classes of the object $o$. A triple $(c, \mathsf{rdfs{:}subClassOf}, d)$ means that each node of class $c$ is also of class $d$ and, finally, a triple $(p, \mathsf{rdfs{:}subPropertyOf}, q)$ implies that for each statement $(s, p, o)$ also the statement $(s, q, o)$ holds.

This concept, where schema information and typing are represented internally in the graph, is fundamentally different from the classical approach in graph transformation, where schema information is kept in a type graph (possibly with inheritance structure) and typings are given by a morphism from the graph into the type graph. The advantages of the RDF approach lie in

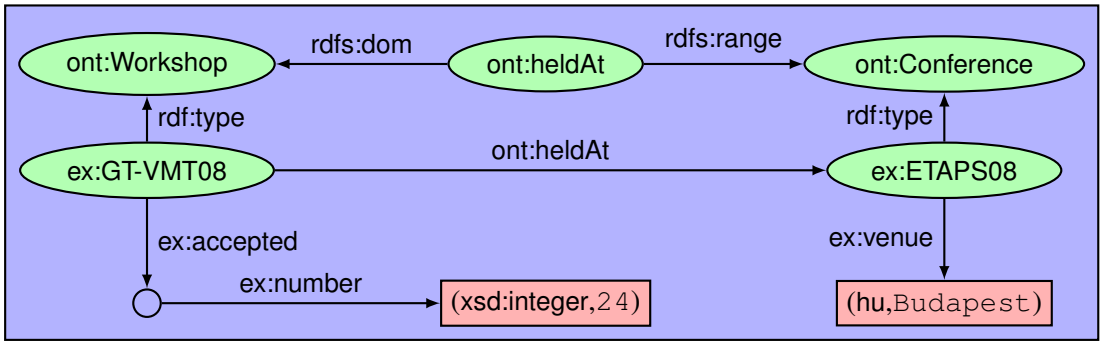



Figure 1: Example of an RDF graph

its flexibility: Nodes may have several classes or no classes at all instead of exactly one class in the case of typed graphs. This facilitates the representation of information, for which there is no schema available yet, and the application of different schemata to the same instance information. Moreover, schemata and typing of an RDF graph may be easily modified at runtime.

*Example* 1 (RDF graph) *In Figure 1 a small example of an RDF graph is depicted, where some information about GT-VMT 2008 is represented. URIs are visualised by ellipses, literals by rectangles, and blank nodes by circles. The triples of the RDF graph are given by the arrows in the figure.*

*In the upper row a simple ontology for workshops being held at conferences is introduced. This schema is instantiated in the second row to state that the workshop GT-VMT 2008 is held at the conference ETAPS 2008. We use different namespaces,* ont: *for the ontology and* ex: *for the instance information, to illustrate the possibility that the ontology namespace and its elements are defined elsewhere and just imported into this graph.*

*The third row gives some additional information about the number of accepted contributions for GT-VMT 2008 by a typed literal, and about the venue of ETAPS 2008 by a plain literal, where the language tag states that the Hungarian name of the venue is "Budapest". This is not really necessary in this case, since Budapest has the same name in most languages, but is already useful for cities like Beijing, Moscow, Rome or Munich, which have different names and transcriptions in different languages.*

## 2.2 RDF Graph Homomorphisms

To obtain a category of RDF graphs we define RDF graph homomorphisms, which capture the structural relationships between RDF graphs. Essentially these are subgraph relations modulo a translation of the blank nodes, which means that blank nodes can be renamed, identified and included into a larger set of blank nodes before the accordingly translated triples are included into the triples of the codomain graph.

**Definition 2** (RDF Graph Homomorphism) Given two RDF graphs $G$ and $H$, an RDF graph homomorphism $h\colon G \to H$ is given by a translation function $h_{\text{Blank}}\colon G_{\text{Blank}} \to H_{\text{Blank}}$, such that

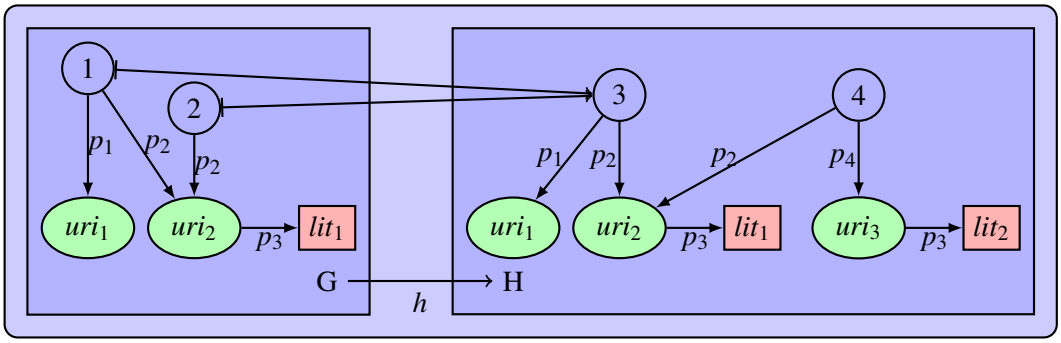



Figure 2: Example of an RDF graph homomorphism

$(h_{\text{Blank}})^{\#}(G_{\text{Triple}}) \subseteq H_{\text{Triple}}$, where the extension $(h_{\text{Blank}})^{\#}$ of $h_{\text{Blank}}$ to triples is constructed by $(h_{\text{Blank}})^{\#} := (h_{\text{Blank}} + \text{id}_{\text{URI}} + \text{id}_{\text{Lit}}) \times \text{id}_{\text{URI}} \times (h_{\text{Blank}} + \text{id}_{\text{URI}} + \text{id}_{\text{Lit}})$.

*Remark* 2 (Relationship to concepts of RDF) *RDF graph homomorphisms are not considered explicitly in the normative RDF specifications. However, a notion of graph equivalence, corresponding to a bijective homomorphism in the sense of the above definition, is defined in [KC04] and [Hay04], where there are some technical differences resulting from the different treatment of blank nodes mentioned in Remark 1.*

*Example* 2 (RDF graph homomorphism) *In Figure 2 an example is shown, which abstractly illustrates the possibilities of an RDF graph homomorphism. The blank nodes* 1 *and* 2 *of graph G are identified to the blank node* 3 *in graph H, i. e., the homomorphism h is non-injective. In order to satisfy the required triple set inclusion, the triples* $(3, p_1, uri_1)$, $(3, p_2, uri_2)$ *and* $(uri_2, p_3, lit_1)$ *are included in H. Moreover, H also has some additional information not in the image of h, namely the blank node* 4 *and the triples* $(4, p_2, uri_2)$, $(4, p_4, uri_3)$ *and* $(uri_3, p_3, lit_2)$, *i. e., h is non-surjective.*

Our first result is that RDF graphs and their homomorphisms in fact establish a category. Moreover, arbitrary limits and colimits can be constructed in this category.

**Proposition 1** (Category **RDFHom**) *RDF graphs and RDF graph homomorphisms constitute a category, denoted by* **RDFHom**. *This category is complete and cocomplete, i. e., it has limits and colimits over all diagrams.*

*Proof sketch.* Composition and identities are just the composition and identities of the underlying translations of blank nodes, i. e., composition and identities in the category **Set**, which are known to satisfy associativity of composition and neutrality of identities. The required triple set inclusions for compositions follow easily from the underlying inclusions, while they are immediately obvious for identical triple sets.

Colimits can be constructed by first taking the colimit $C_{\text{Blank}}$ of the blank nodes in **Set**. Then the triple sets in the diagram can be translated via the morphisms into the colimit to become triple sets over $C_{\text{Blank}}$, where we can finally take the set-theoretic union of these translated triple

sets to be the triple set $C_{\text{Triple}}$ of the colimit graph.

Limits can be constructed similarly by first taking the limit $L_{\text{Blank}}$ of the blank nodes in **Set**. The reverse translation of the triple sets in the diagram to triple sets over $L_{\text{Blank}}$ can be built by $\{(s,p,o) \mid (h_{\text{Blank}})^{\#}(s,p,o) \in G_{\text{Triple}}\}$ for all graphs $G$ and corresponding functions $h_{\text{Blank}} \colon L_{\text{Blank}} \to G_{\text{Blank}}$. Finally, the triple set $L_{\text{Triple}}$ of the limit is obtained by the set-theoretic intersection of the translated triple sets. □

*Remark* 3 (Interpretation of limits and colimits) *The merge of RDF graphs defined in [Hay04], which takes the union of RDF graphs, while "standardising apart" common blank nodes, is naturally obtained as the coproduct of RDF graphs in our category-theoretical setting. The more complex colimits can be used to construct merges over common blank nodes, which are protected from being standardised apart.*

*Intersections of RDF graphs are not treated explicitly in the RDF specifications. The limit constructions in* **RDFHom** *can be used to formalise such intersections under common blank nodes.*

## 2.3 RDF Graph Instantiations

In [Hay04] blank nodes are interpreted as existential variables and an instance of an RDF graph is defined to be a graph, where some of the blank nodes are replaced by concrete URIs or literals. This leads to the following definition of a more general kind of morphism on RDF graphs. Note that the name "instantiation" does not refer to the instantiation of a schema or ontology, but to the instantiation of a blank node.

**Definition 3** (RDF Graph Instantiation) Given RDF graphs $G$ and $H$, an RDF graph instantiation $i \colon G \to H$ is given by an assignment function $i_{\text{Blank}} \colon G_{\text{Blank}} \to H_{\text{Blank}} + \mathsf{URI} + \mathsf{Lit}$, such that $(i_{\text{Blank}})^{\#}(G_{\text{Triple}}) \subseteq H_{\text{Triple}}$, where the extension $(i_{\text{Blank}})^{\#}$ of $i_{\text{Blank}}$ to triples is constructed by $(i_{\text{Blank}})^{\#} := (i_{\text{Blank}} + \mathrm{id}_{\mathsf{URI}} + \mathrm{id}_{\mathsf{Lit}}) \times \mathrm{id}_{\mathsf{URI}} \times (i_{\text{Blank}} + \mathrm{id}_{\mathsf{URI}} + \mathrm{id}_{\mathsf{Lit}})$.

*Remark* 4 (Related concepts) *RDF graph instantiations are a combination of instances and subgraph relationships in the sense of [Hay04]. This is particularly interesting, because the Interpolation Lemma of [Hay04] states: "S entails a graph E if and only if a subgraph of S is an instance of E." Using the notion of RDF graph instantiation this is simplified to: "S entails E if and only if there is an instantiation $i \colon E \to S$." The characterisation of entailment by graph homomorphisms is also examined in [Bag05], where RDF graphs are translated into directed, labelled multigraphs and RDF entailment is shown to correspond to their morphisms.*

*Instantiations may also be used to formalise queries on an RDF data store. A data store, given by an RDF graph D, is queried using a pattern, given by another RDF graph P. The result of the query should be the set of all possible instantiations $i \colon P \to D$. In [CF07] a related but more complex approach is taken, where SPARQL queries and RDF data sets are translated to conceptual graphs and the results are computed by finding conceptual graph homomorphisms.*

*Example* 3 (RDF graph instantiation) *The RDF graph instantiation $i \colon G \to H$ in Figure 3 identifies the blank nodes 1 and 2 to the URI $uri_2$ and maps the blank node 3 to the literal $lit_1$. In terms of entailment this means that the triple $(uri_2, p_1, uri_1)$ in H entails the triple $(1, p_1, uri_1)$ in*

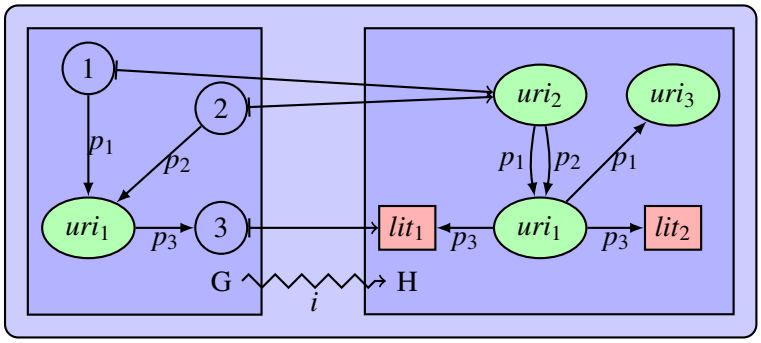



Figure 3: Example of an RDF graph instantiation

*$G$, $(uri_2, p_2, uri_1)$ entails $(2, p_2, uri_1)$ and $(uri_1, p_3, lit_1)$ entails $(uri_1, p_3, 3)$. The non-surjectivity of the instantiation $i$ means that not all information in $H$ is used to entail $G$. In this and following figures we depict instantiations by jagged arrows, while plain homomorphisms are shown as normal arrows.*

In the following proposition we show that RDF graph instantiations give rise to another category, which comprises **RDFHom** as a subcategory. This category is neither complete nor cocomplete. In Subsection 4.1 we will, however, examine circumstances under which pushouts of instantiations exist.

**Proposition 2** (Category **RDFInst**) *RDF graphs and RDF graph instantiations constitute a category, denoted by* **RDFInst**, *with* **RDFHom** $\subseteq$ **RDFInst**. *This category does not have limits and colimits in general.*

*Proof sketch.* The composition of instantiations $i\colon G \to H$ and $j\colon H \to K$ can be obtained by $(j \circ i)_{\mathrm{Blank}} := (j_{\mathrm{Blank}} + \mathrm{id}_{\mathsf{URI}} + \mathrm{id}_{\mathsf{Lit}}) \circ i_{\mathrm{Blank}}$. Identities $\mathrm{id}_G$ are given by embeddings $\mathrm{id}_{G,\mathrm{Blank}}$ of the blank node set $G_{\mathrm{Blank}}$ into the coproduct $G_{\mathrm{Blank}} + \mathsf{URI} + \mathsf{Lit}$. Associativity of composition and neutrality of identities follow from the corresponding properties of **Set**. The required triple set inclusions are again direct consequences of the underlying inclusions in the composed instantiations. The inclusion of **RDFHom** into **RDFInst** is obvious, since each blank node translation $h_{\mathrm{Blank}}\colon G_{\mathrm{Blank}} \to H_{\mathrm{Blank}}$ is also an assignment $h_{\mathrm{Blank}}\colon G_{\mathrm{Blank}} \to H_{\mathrm{Blank}} + \mathsf{URI} + \mathsf{Lit}$, which simply does not use the possibilities of the extended codomain.

The existence of colimits is impeded by instantiations of the same blank node to different URIs or literals, which cannot be reconciled. The existence of general limits is on the other hand inhibited by the non-existence of a final object, into which unique instantiations are impossible, since a blank node can be instantiated to arbitrary URIs or literals. □

## 3 Which Transformation Approach?

In this section we will try to find an algebraic graph transformation approach suitable for transforming RDF graphs, which should not only provide a formal basis for the transformations them-

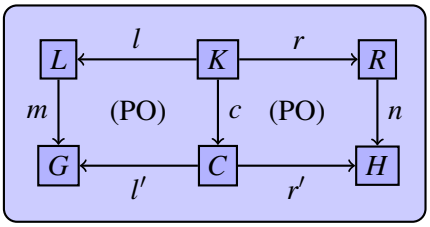



Figure 4: Double pushout transformation

selves, but also for features like reversibility (to support undoing of editing transformations) and reasoning about dependencies (to achieve structured version histories).

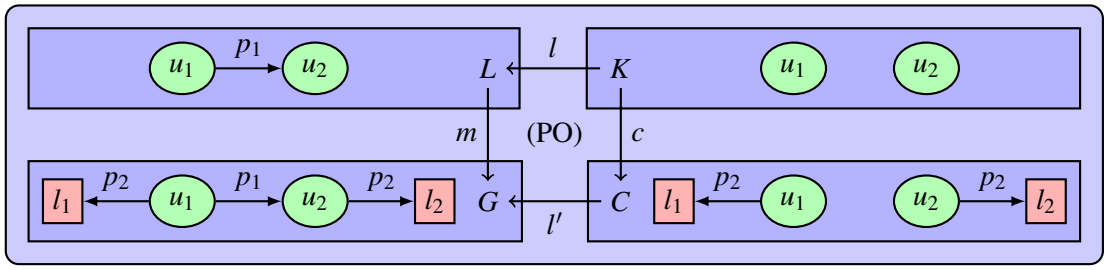
### 3.1 Double Pushout Approach

The Double Pushout (DPO) approach is one of the predominant approaches in algebraic graph transformation. Its theory has been reformulated in the framework of adhesive HLR categories in [EEPT06]. The main idea of DPO transformations is to split the task of transforming a graph into the deletion of graph elements by a pushout complement and the creation of new elements by a pushout. A DPO transformation is shown in Figure 4, where the rule $L \leftarrow K \rightarrow R$ is applied to the graph $G$ via the match $m\colon L \rightarrow G$ by first constructing the context $C$ by a pushout complement and then the resulting graph $H$ with comatch $n\colon R \rightarrow H$ by a pushout of $R$ and $C$ over $K$.

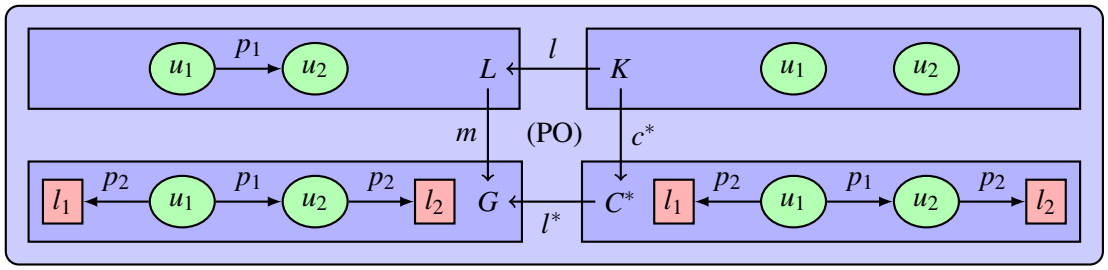
DPO transformations are always reversible due to their symmetric structure. If a graph $H$ is obtained from a graph $G$ by the application of a rule $L \leftarrow K \rightarrow R$, then a graph isomorphic to $G$ may be reconstructed from $H$ by the inverse rule $R \leftarrow K \rightarrow L$. Moreover, there is an extensive theory about the dependencies and conflicts between DPO transformation rules.

Most instantiations of the DPO approach ensure or require unique pushout complements in order to facilitate unique transformation results. This is not possible for RDF graphs as can be seen in the example in Figure 5, where two different pushout complements for the same given situation are shown. The ambiguity results from the fact that the triple $(u_1, p_1, u_2)$, deleted in the context graph $K$, can be deleted as in $C$ in Figure 5(a) or it can be preserved as in $C^*$ in Figure 5(b). The graph $G$ is a pushout (set-theoretic union of the triples) in both cases.

Since this ambiguity results from RDF graphs being a specialisation of simple graphs, a similar problem arises when applying the DPO approach to them. One possibility to overcome this is the restriction to rules, which never delete an edge without also deleting one of its adjacent nodes. Such a solution is, e. g., pursued in [BHKR07], where an edge in a multigraph is translated to a "dummy node" in a corresponding simple graph and, hence, all rules deleting an edge in the multigraph delete at least this very node in the translated simple graph. Theses rules are, however, too restrictive in our scenario, because they would only allow to delete triples involving blank nodes (and only while also removing the blank node itself) but not statements made solely about URIs and literals. Another way to resolve the problem is the use of a different transformation approach, two of which are shortly discussed in the following subsection.




(a) Pushout complement with deletion




(b) Pushout complement without deletion

Figure 5: Ambiguity of pushout complements in **RDFHom**

### 3.2 Single and Sesqui Pushout Approach

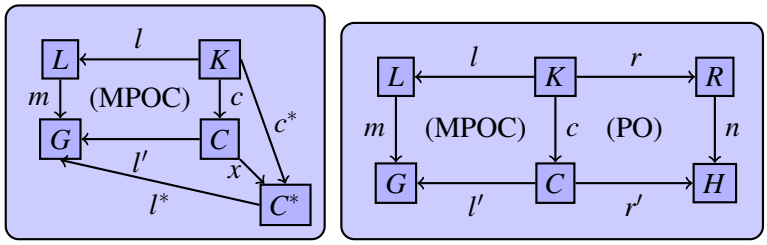
An early alternative approach to algebraic graph transformation is the Single Pushout (SPO) approach studied in [EHK+97]. Instead of splitting deletion and creation into seperate constructions the SPO approach achieves both at the same time by pushouts in a suitable category of partial morphisms. Among the key characteristics of such partial pushouts are the deletion in unknown context (if a node is deleted all edges connected to this node are also deleted even if they are not mentioned in the rule) and the precedence of deletion over preservation (if a deleted node is identified to a preserved one by the match the node is deleted leading to a partial comatch).

A recent proposal is the Sesqui Pushout (SqPO) approach introduced in [CHHK06]. It also features deletion in unknown context but does not allow the identification of deleted and preserved nodes. This is achieved by a split into deletion and creation similar to the DPO approach, where deletion is modelled by final pullback complements instead of pushout complements.

The SPO and SqPO approaches both result in deletion in unknown context, which is not desirable in our case, because it impedes the reversibility of transformations. Implicitly deleted edges may not be reconstructed. While this could be resolved by using, e. g., an SqPO approach with a condition prohibiting dangling edges or an SPO approach with an additional condition to avoid identifications, we opt for modifying the DPO approach in the following section.

### 3.3 The Need for a Modified Approach

Since we want to have unique transformation results, but do not want deletion in unknown context, we cannot use one of the previous approaches unmodified. Returning to Figure 5 we observe




(a) Minimal pushout complement (b) MPOC-PO transformation

Figure 6: MPOC and MPOC-PO transformation

that the result we want to achieve in this situation is the deletion of the triple. Hence, our approach, developed in the following section for RDF, is to resolve the ambiguity by canonically selecting the minimal pushout complement (MPOC), i. e., a pushout complement in which as much as possible is deleted.

**Definition 4** (Minimal Pushout Complement) Given morphisms $l\colon K \to L$ and $m\colon L \to G$ in an arbitrary category, a minimal pushout complement $C$ of $l$ and $m$ with morphisms $l'\colon C \to G$ and $c\colon K \to C$ is a pushout complement, i. e., $G$ is a pushout of $L$ and $C$ over $K$, such that for each pushout complement $C^*$ of $l$ and $m$ with morphisms $l^*\colon C^* \to G$ and $c^*\colon K \to C^*$ there is a unique morphism $x\colon C \to C^*$ with $l' = l^* \circ x$ and $c^* = x \circ c$ (cf. Figure 6(a)).

An MPOC-PO transformation is now given by a DPO transformation with the additional side condition that the deletion in the left-hand square of Figure 6(b) is an MPOC, while the creation in the right-hand square is still done by a pushout.

**Definition 5** (MPOC-PO Transformation) An MPOC-PO transformation rule is given by a span $L \leftarrow K \rightarrow R$ of morphisms in an arbitrary category. An application of this rule to an object $G$ via a morphism $m\colon L \to G$ as match is given by the diagram in Figure 6(b) resulting in the object $H$ with the morphism $n\colon R \to H$ as comatch.

This should preserve most of the well-behavedness of the DPO approach, because the only difference is the added side condition, which ensures uniqueness of the transformation result.

**Proposition 3** (Uniqueness of Transformation Results) *The result of an MPOC-PO transformation is unique up to isomorphism.*

*Proof sketch.* Since two minimal pushout complements have unique morphisms in both directions and their compositions have to be the respective identities as unique endomorphisms, they are obviously isomorphic.

Moreover, pushouts are unique up to isomorphism in any category and, hence, the result of first constructing an MPOC and then a pushout is also unique up to isomorphism. □

Note that the existence of POCs, the existence of MPOCs for non-unique POCs and the existence of pushouts are not guaranteed in general. In the next section we will give constructions

for pushouts and MPOCs in **RDFInst**, which under certain conditions also imply their existence.

The use of MPOCs as a way to canonically select the result of a deletion may also be beneficial when applying the DPO approach to simple graphs or other categories, where there are non-unique pushout complements.

# 4 RDF Graph Transformations and Basic Results

In this section we apply MPOC-PO transformations to RDF graphs, where the necessary constructions are explicitly developed and sufficient conditions on the rule and match morphisms are given to ensure existence of the transformation results. Moreover, conditions for the reversiblity of RDF graph transformations are shortly discussed.

## 4.1 MPOC-PO Transformations for RDF

For the creation of elements we need pushouts in **RDFInst**. Sufficient conditions for their existence can be achieved by restricting one of the instantiations in the given span to be an injective homomorphism. This ensures that contradictions can arise neither due to assignments of one blank node to different URIs or literals (the homomorphism cannot assign a blank node to a URI or literal, but only to a blank node) nor due to the identification of several blank nodes assigned to different URIs or literals (injectivity prevents identifications).

**Theorem 1** (Pushouts in **RDFInst**) *Given an injective RDF graph homomorphism $r\colon K \to R$ and an RDF graph instantiation $c\colon K \to C$, a pushout $H$ with an injective RDF graph homomorphism $r'\colon C \to H$ and an RDF graph instantiation $n\colon R \to H$ can be constructed by*

- *the blank node set $H_{\text{Blank}} := C_{\text{Blank}} + (R_{\text{Blank}} \setminus r_{\text{Blank}}(K_{\text{Blank}}))$ with the injection $r'_{\text{Blank}}$ of $C_{\text{Blank}}$ into the coproduct and the assignment $n_{\text{Blank}}$, which acts on nodes from $K_{\text{Blank}}$ as $c_{\text{Blank}}$ does and on the new nodes as an injection into the coproduct, and*
- *the triple set $H_{\text{Triple}} := (r'_{\text{Blank}})^{\#}(C_{\text{Triple}}) \cup (n_{\text{Blank}})^{\#}(R_{\text{Triple}})$.*

*Proof sketch.* The construction of $H_{\text{Blank}}$ ensures that the assignment $n_{\text{Blank}}$ is well-defined, because the inclusion of $C_{\text{Blank}}$ enables its acting like $c_{\text{Blank}}$ on the preserved nodes, while the addition of the new elements of $R_{\text{Blank}}$ facilitates their injective mapping. The union construction of the triple set guarantees the satisfaction of the corresponding triple set inclusions, while the commutativity $n_{\text{Blank}} \circ r_{\text{Blank}} = r'_{\text{Blank}} \circ c_{\text{Blank}}$ also holds by construction.

For each other graph $H^*$ with instantiations $n^*\colon R \to H^*$ and $r^*\colon C \to H^*$, such that $n^* \circ r = r^* \circ c$ an assignment $x_{\text{Blank}}\colon H_{\text{Blank}} \to H^*_{\text{Blank}} + \mathsf{URI} + \mathsf{Lit}$ is uniquely induced by the requirement that it acts on blank nodes from $C$ as $r^*$ does ($x \circ r' = r^*$) and on blank nodes from $R$ as $n^*$ does ($x \circ n = n^*$). □

*Remark* 5 (Asymmetric Construction of Pushout) *In category theory pushouts are usually obtained symmetrically by first constructing a coproduct, i. e., some kind of disjoint union, followed by a coequaliser identifying the common elements in the interface. In general, this leads to renaming of all elements in both given graphs.*

*We have chosen a concrete, asymmetric construction in this theorem in order to highlight that the (possibly very large) host graph can be left unchanged, while the additional blank nodes are added disjointly. This is also closer to possible implementations, which should construct the pushout directly instead of following the elegant but inefficient categorical construction.*

For the deletion we will use the concept of MPOC introduced in Definition 4. In the following theorem we give sufficient conditions and a construction for MPOCs in **RDFInst**. Two of these conditions, namely the identification and the dangling condition, are well-known from the ordinary DPO approach, while the third ensures that deleted blank nodes are not assigned to URIs or literals, which cannot be deleted.

**Theorem 2** (Minimal Pushout Complements in **RDFInst**) *Given an injective RDF graph homomorphism $l\colon K \to L$ and an RDF graph instantiation $m\colon L \to G$, such that the deleted blank nodes in $L_{\text{Blank}} \setminus l_{\text{Blank}}(K_{\text{Blank}})$*

- *are not identified to other nodes (identification condition),*
- *are not assigned to URIs or literals, and*
- *are not assigned to blank nodes, which are used in triples in $G_{\text{Triple}} \setminus (m_{\text{Blank}})^{\#}(L_{\text{Triple}})$ (dangling condition),*

*an MPOC $C$ with an injective RDF graph homomorphism $l'\colon C \to G$ and an RDF graph instantiation $c\colon K \to C$ can be constructed by*

- *the blank node set $C_{\text{Blank}} := G_{\text{Blank}} \setminus m_{\text{Blank}}(L_{\text{Blank}} \setminus l_{\text{Blank}}(K_{\text{Blank}}))$ with the inclusion $l'_{\text{Blank}}$ of this set into $G_{\text{Blank}}$ and the assignment $c_{\text{Blank}}$, which behaves like $m_{\text{Blank}} \circ l_{\text{Blank}}$, and*
- *the triple set $C_{\text{Triple}} := G_{\text{Triple}} \setminus (m_{\text{Blank}})^{\#}(L_{\text{Triple}} \setminus (l_{\text{Blank}})^{\#}(K_{\text{Triple}}))$.*

*Proof sketch.* The assignment $c_{\text{Blank}}$ is well-defined and the triple inclusion of $(c_{\text{Blank}})^{\#}(K_{\text{Triple}})$ into $C_{\text{Triple}}$ is satisfied, because the blank nodes in the range of $m_{\text{Blank}} \circ l_{\text{Blank}}$ and the triples from $K_{\text{Triple}}$ are explicitly not removed in $C$, while the commutativity $m_{\text{Blank}} \circ l_{\text{Blank}} = l'_{\text{Blank}} \circ c_{\text{Blank}}$ also holds by this construction.

The blank node set $G_{\text{Blank}}$ may be reconstructed from $L_{\text{Blank}}$ and $C_{\text{Blank}}$ by the pushout construction in Theorem 1, since the removed triples $m_{\text{Blank}}(L_{\text{Blank}} \setminus l_{\text{Blank}}(K_{\text{Blank}}))$ are exactly the ones that are added by the pushout. Moreover, the pushout construction also recovers the triple set $G_{\text{Triple}}$. Hence, the construction in fact leads to a pushout complement.

For each other pushout complement $C^*$ with instantiations $c^*\colon K \to C^*$ and $l^*\colon C^* \to G$ the blank node set $C^*_{\text{Blank}}$ has to be isomorphic to $C_{\text{Blank}}$, because otherwise the pushout construction would also result in a non-isomorphic blank node set. The blank node bijection $x_{\text{Blank}}\colon C_{\text{Blank}} \to C^*_{\text{Blank}}$ is then already uniquely determined by the requirement $l'_{\text{Blank}} = l^*_{\text{Blank}} \circ x_{\text{Blank}}$. The triple set $(x_{\text{Blank}})^{\#}(C_{\text{Triple}})$ has to be included in $C^*_{\text{Triple}}$, since $C^*$ has to contain all triples, which cannot be reconstructed from $L_{\text{Triple}}$ by the pushout, and all triples contained in $K_{\text{Triple}}$ in order to be a pushout complement. □

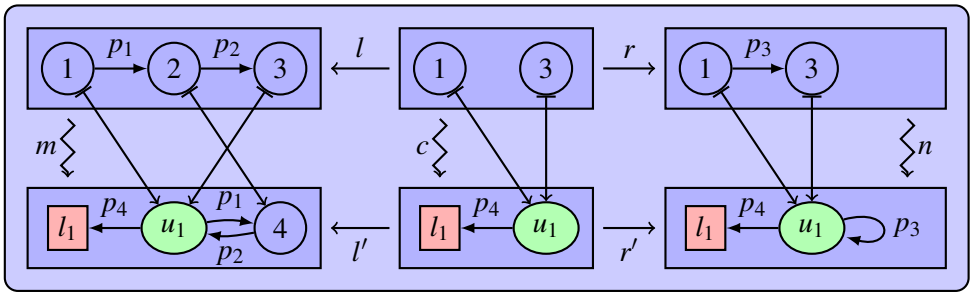



Figure 7: Example of an RDF graph transformation

*Remark* 6 (Pushouts and MPOCs of proper instantiations) *The conditions given in Theorem 1 and Theorem 2 are sufficient, but not necessary, respectively. In fact, it is possible to construct pushouts of (proper) RDF graph instantiations if the assignments of blank nodes do not contradict. For our application to RDF graph transformation the given constructions over homomorphisms are, however, general enough, since we do not want to instantiate blank nodes in the rule but only in the match.*

We now have both constructions that are necessary to define RDF graph transformations as MPOC-PO transformations.

**Definition 6** (RDF Graph Transformation) An RDF graph transformation rule is given by a span $L \leftarrow K \rightarrow R$ of injective RDF graph homomorphisms. An application of this rule to an RDF graph $G$ via an RDF graph instantiation $m\colon L \rightarrow G$ as match is given by a MPOC-PO transformation as defined in Definition 5 resulting in the RDF graph $H$ with the RDF graph instantiation $n\colon R \rightarrow H$ as comatch.

*Example* 4 (RDF graph transformation) *In Figure 7 an example of an RDF graph transformation is depicted, where the rule replaces a sequential occurrence of predicates $p_1$ and $p_2$ by a single occurrence of $p_3$, while deleting the intermediate blank node. Such a rule is not typical for* reasoning *in RDF, since inference only yields additional triples, but does not delete anything. We claim, however, that such deleting transformations are useful for* editing *RDF data, e. g., if the predicates $p_1$ and $p_2$ are deprecated and shall be replaced by $p_3$.*

The following corollary summarises the existence and uniqueness properties of RDF graph transformations.

**Corollary 1** (Applicability and Uniqueness of RDF Transformations) *An RDF graph transformation rule $L \leftarrow K \rightarrow R$ is applicable via a match instantiation $m\colon L \rightarrow G$ if $m$ does not identify deleted blank nodes to other nodes, does not assign deleted blank nodes to URIs or literals and does not assign deleted blank nodes to blank nodes occuring in triples not in the range of $m$. In this case the resulting RDF graph $H$ with comatch $n\colon R \rightarrow H$ is unique up to isomorphism.*

*Proof.* This corollary follows directly from Theorem 2 and Theorem 1 regarding the existence

of the result and from Proposition 3 regarding its uniqueness. □

### 4.2 Reversible Transformations

While DPO transformations are always reversible due to their symmetric structure, MPOC-PO transformations are only reversible if the creation pushout is also an MPOC. In RDF graph transformations this is the case if none of the additional triples in $R$ is already present in $C$.

**Theorem 3** (Reversibility of RDF Graph Transformations) *Given the RDF graph transformation in Figure 6(b), the application of the inverse rule $R \leftarrow K \rightarrow L$ to the graph $H$ via the match $n\colon R \rightarrow H$ is possible and leads to a graph $G'$ isomorphic to $G$ provided that $(r'_{\text{Blank}})^{\#}(C_{\text{Triple}}) \cap (n_{\text{Blank}})^{\#}(R_{\text{Triple}}) = (n_{\text{Blank}} \circ r_{\text{Blank}})^{\#}(K_{\text{Triple}}) = (r'_{\text{Blank}} \circ c_{\text{Blank}})^{\#}(K_{\text{Triple}})$.*

*Proof sketch.* The given condition ensures that the right square is not only a pushout, but that $C$ is also an MPOC in this square, since all common triples of $C_{\text{Triple}}$ and $R_{\text{Triple}}$ are also in the interface $K_{\text{Triple}}$ and will therefore not be removed by the MPOC construction. □

In order to constrain RDF graph transformations to reversible cases negative application conditions could be used, which disallow all triples added by the rule from being already present in the host graph. To achieve the same expressibility additional rules could be added, which have these triples in their left-hand sides and just perform the other changes of the rule. A complete discussion is, however, outside the scope of this paper.

## 5 Summary and Future Work

In this contribution we have formalised RDF in a category theoretical framework, provided a transformation approach for RDF graphs and shown under which circumstances transformations are applicable and reversible. These results can provide a useful basis for the rule-based modification and creation of RDF data.

Considering the application scenario sketched in the introduction, the theory developed so far allows us to define a set of grammar rules. If an RDF data store is only modified using these rules, certain structural conditions can be ensured, such as a publication in the bibliography application having at least one title. Moreover, the reversibility of transformations allows us to undo the last rule-based changes to the data store.

Another use case for transformation rules is the automatic application of a set of rules on all possible matches, which could be used, e. g., to add references to a related ontology, where the (non-deleting) rules would require certain predicates and types from the present ontology in their left-hand sides and add the corresponding predicates and types from the new ontology in their right-hand sides. Since the left- and right-hand sides can be whole graphs, intermediate structures in the source ontology can be by-passed by simpler structures in the target ontology. Vice versa, intermediate blank nodes may be introduced, if the target ontology needs them.

Future work should transfer theoretical results for negative application conditions to RDF graph transformations. These would significantly enhance the expressibility of the rules by prohibiting their application in certain contexts. This enhancement can be used to ensure that certain

structures, such as the title of a publication in our scenario, can exist only once. On the other hand it is needed to ensure termination of automatic transformations (by prohibiting the right-hand side to be already present in the graph) and reversibility of rule applications (as already mentioned in Subsection 4.2).

Analysis techniques for dependencies and conflicts should be adapted to RDF transformations. These would enable us to develop a transformation-based version control system for RDF data stores, in which independent changes can be reverted independently and merged automatically. Moreover, the analysis of critical pairs for transformation rules could be used to propose default resolutions for merge conflicts.

## Bibliography

[Bag05] J.-F. Baget. RDF Entailment as a Graph Homomorphism. In Gil et al. (eds.), *The Semantic Web – ISWC 2005*. LNCS 3729, pp. 82–96. Springer, 2005.
doi:10.1007/11574620_9
ftp://ftp.inrialpes.fr/pub/exmo/publications/baget2005a.pdf

[BFM98] T. Berners-Lee, R. T. Fielding, L. Masinter. RFC 2396 – Uniform Resource Identifiers (URI): Generic Syntax. Internet Engineering Task Force (IETF), Aug. 1998.
http://tools.ietf.org/html/rfc2396

[BG04] D. Brickley, R. V. Guha (eds.). *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium (W3C), Feb. 2004.
http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[BHKR07] I. Boneva, F. Hermann, H. Kastenberg, A. Rensink. Simulating Multigraph Transformations Using Simple Graphs. In Ehrig and Giese (eds.), *Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*. Electronic Communications of the EASST 6. 2007.
http://eceasst.cs.tu-berlin.de/index.php/eceasst/article/view/62

[BHLT06] T. Bray, D. Hollander, A. Layman, R. Tobin (eds.). *Namespaces in XML 1.0 (Second Edition)*. World Wide Web Consortium (W3C), Aug. 2006.
http://www.w3.org/TR/2006/REC-xml-names-20060816/

[BM04] P. V. Biron, A. Malhotra (eds.). *XML Schema Part 2: Datatypes (Second Edition)*. World Wide Web Consortium (W3C), Oct. 2004.
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[CF07] O. Corby, C. Faron-Zucker. Implementation of SPARQL Query Language Based on Graph Homomorphism. In Priss et al. (eds.), *Conceptual Structures: Knowledge Architectures for Smart Applications. Proc. ICCS 2007*. LNCS/LNAI 4604, pp. 472–475. Springer, 2007.
doi:10.1007/978-3-540-73681-3_37

[CHHK06] A. Corradini, T. Heindel, F. Hermann, B. König. Sesqui-Pushout Rewriting. In Corradini et al. (eds.), *Graph Transformations. Proc. ICGT 2006*. LNCS 4178, pp. 30–45. Springer, 2006.
doi:10.1007/11841883_4

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.
doi:10.1007/3-540-31188-2

[EHK+97] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, A. Corradini. Algebraic Approaches to Graph Transformation – Part II: Single Pushout Approach and Comparison with Double Pushout Approach. In Rozenberg (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations*. Chapter 4. World Scientific, 1997.

[Hay04] P. Hayes (ed.). *RDF Semantics*. World Wide Web Consortium (W3C), Feb. 2004.
http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[KC04] G. Klyne, J. J. Carroll (eds.). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium (W3C), Feb. 2004.
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[MM04] F. Manola, E. Miller (eds.). *RDF Primer*. World Wide Web Consortium (W3C), Feb. 2004.
http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[MPG07] S. Muñoz, J. Pérez, C. Gutierrez. Minimal Deductive Systems for RDF. In Franconi et al. (eds.), *The Semantic Web: Research and Applications. Proc. ESWC 2007*. LNCS 4519, pp. 53–67. Springer, 2007.
doi:10.1007/978-3-540-72667-8_6
http://www2.ing.puc.cl/~jperez/papers/minimal-rdf-camera-ready-ext.pdf

[PS07] E. Prud'hommeaux, A. Seaborne (eds.). *SPARQL Query Language for RDF*. World Wide Web Consortium (W3C), Nov. 2007.
http://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/

[Uni07] The Unicode Consortium. The Unicode Standard, Version 5.0.0. 2007.
http://www.unicode.org/versions/Unicode5.0.0/